Smart Garbage Can System

Final Project Report

CS 190 Spring 2018

University of California, Irvine

Prepared by:

Jacob Tiritilli

Kyle Hefner

Table of Contents

Introduction

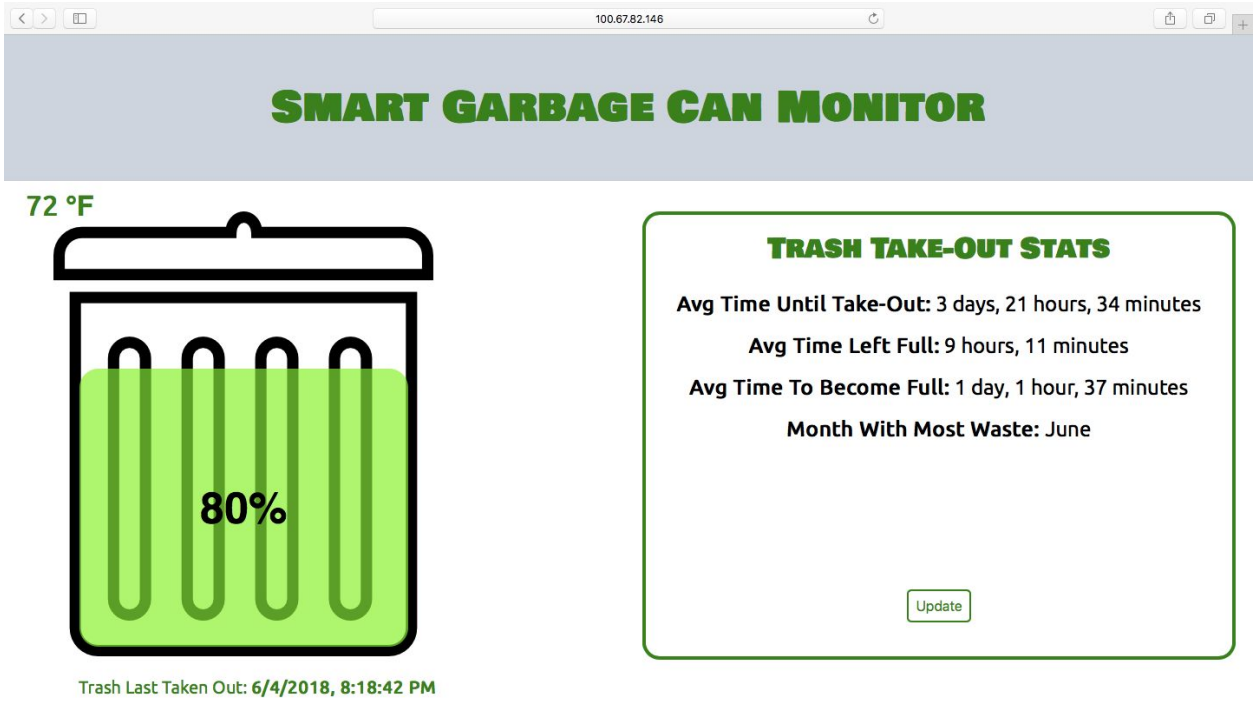
The goal of this project was to essentially model the software and design needed to build a smart and connected trash can. This project leverages internet of things technologies in order to create a trash can that can detect when it is full and send a notification for someone to come and take it out. This functionality has several potential real-world benefits. For example, this technology could be implemented in dumpsters throughout a city, allowing city workers to be notified when a certain dumpster needs to be emptied. This effectively works to reduce waste and contamination caused by overflow, while allowing for a much more efficient method of garbage collection. Workers would no longer have to go through predetermined routes, emptying every garbage collector even when it is only partially full; instead, this intelligent system would allow for more optimized routes where only the garbage collectors that need to be taken out are emptied. These types of smart trash cans are already being implemented in the real world. For example, Bigbelly sells IoT trash cans that offer all of the functionality above. Furthermore, the US Army recently announced that it would be deploying sensors in 100 dumpsters that can transmit data on temperature, tilt, acceleration, and fill level. The system will send a notification to officials when the dumpster reaches 100% full. They also plan to collect this data overtime to provide better predictions on future waste collection needs, while optimizing trash collection routes and preventing overflows.[1]

[1] US Army Dumpster Sensors Special Notice: https://www.fbo.gov/index?s=opportunity&mode=form&id=04a91edf887318401a6195a8aeb49624&tab=core&_cview=0

System Overview

The smart garbage can system modeled by this project essentially monitors the fullness level of a particular garbage collector, and sends the data to a server which allows it to be displayed in a web application. The system also collects data on how often the trash is being taken out, how long it is being left full for before being emptied, and how long it is taking the garbage collector to become full. The current temperature of the trash collector is also read and displayed. In order to achieve this functionality, an ultrasonic sensor is used to measure the height of the trash in inches. This sensor would potentially be placed on the lid of a particular garbage can or dumpster. This reading, along with the current temperature (in fahrenheit) is then sent to a server where the data is processed and stored. The server, which stores the height of the garbage collector, calculates the percentage full based on the current reading from the sensor, and pushes this data to a web application where it can be easily viewed. Also, timestamps are taken every time the garbage collector becomes 0% full or hits 100% full. This marks the times that the trash can became empty and full, respectively. This data is then stored and used to calculate various averages that are also displayed and updatable from the web application. A 32-bit WiFi-enabled microcontroller (MCU) is used to read and transmit the data from both the ultrasonic and temperature sensor installed in the garbage collector.

Web Application



The above screenshot is an image of the primary web application. As seen, the web page displays a graphic representing the current height of the trash in the garbage collector, as well as the current temperature, the time and date that the trash was last emptied, and various statistics that have been calculated from data accumulated overtime. The statistic are updated every time the page is loaded, and can be updated manually through an “Update” button on the page. This button will send a request to the server in order to download the latest statistics. A description of the various statistics and what they represent is provided below:

- “Avg Time Until Take-Out”: This represents the average time that the garbage collector went from being empty to being taken out again after collecting some amount of trash. Note that this statistic purposefully ignores whether or not the garbage can ever reached 100% full, allowing it to record when the garbage was emptied before becoming full.
- “Avg Time Left Full”: This represents the average time that the garbage collector stayed at 100% capacity before being emptied.
- “Avg Time To Become Full”: This represents the average time taken for the garbage collector to go from being empty to being 100% full.

- “Month With Most Waste”: This represents the month that has produced the most waste on average, calculated by the number of times that the trash had to be taken out in each month recorded.

The percentage full that is displayed on the web application is calculated based upon the height of the current garbage collector. This information is requested from the user when he first opens up the web application. A screenshot of what the page looks like is provided below. The height of the trash can must be entered in feet.

Once the user enters the height of the garbage collector, he will be redirected to the main monitor pictured on page 4 of this document.

Technical Description

Hardware Components

A list of the main hardware components used in the project, along with a brief description of their use is as follows:

- **Sparkfun ESP8266 Thing Dev Board**
  (https://www.sparkfun.com/products/13711)
  This is a 32-bit WiFi-enabled MCU used to read the data from both sensors and send it across the network to the server.

- **Ultrasonic Sensor - HC-SR04** (https://www.sparkfun.com/products/13959)
  This is the sensor used to measure the height of the trash in the garbage collector. The data received from the sensor in microseconds is converted to inches, using formulas provided by the datasheet, before being sent to the server.

- **Humidity and Temperature Sensor - RHT03 (DHT-22)**
  (https://www.sparkfun.com/products/10167)
  This sensor is used to read the current temperature, in fahrenheit, which is sent directly to the server.

- **Server**
  The project requires a mains powered and internet-connected machine to run the web server that receives data from the microcontroller, and displays and updates the web application.

Software Components

A list of the main software components used in the project, along with a brief description of their use if as follows:

- **Arduino Language (C/C++)**
  (https://www.arduino.cc/reference/en/)
  The Arduino IDE (v1.8.5) is used to program the microcontroller, using various libraries for the Sparkfun Thing Dev board.

- **Python Programming Language (v3.6.5)**
  (https://www.python.org)
  The server as well as various data processing modules used in the project are written in Python 3.

- **Flask Micro Web Framework (v1.0.2)**
  (http://flask.pocoo.org)
  The Flask web framework for Python is used to implement the web server and handle incoming HTTP requests.

- **Flask-SocketIO (v3.0.0)**
  (http://flask-socketio.readthedocs.io)
  This is a library that implements web socket functionality in Flask, allowing it to interact with clients using the popular Socket.IO (https://socket.io) library. This allows for real-time bidirectional communication between the client and the server. This technology is leveraged in the smart garbage can system in order for the server to notify the client when certain events happen, allowing the web application to be changed dynamically without the page having to be refreshed.

Design Overview

This section seeks to provide a brief technical overview of how the smart garbage IoT system operates on the underlying level, and how the various components work together. The description is broken up into the three major components of the project: the embedded device, the web server, and the client application.

Embedded Device

This is the portion of the project that is embedded in the garbage can or dumpster. The resources used in this component of the project are the Sparkfun Thing Dev Board, ultrasonic sensor, and temperature sensor, that are described on page 6 of this document. The role that these “things” play in the project is relatively simple. Utilizing the ESP8266 WiFi and HTTP libraries, the Sparkfun Thing board connects to a specific WiFi network and continuously reads data from both sensors at a specified interval. In testing the application, an interval of 5 seconds was used; but this, of course, would most likely be much longer in a real-life implementation. After initial setup and connection to a WiFi network is established, the same process happens at every interval. The microcontroller receives data from both sensors, encodes the data in a JSON format, and sends the data to the server through the means of an HTTP POST request. An example of the JSON object sent holding both the latest temperature and distance (in inches) readings is provided below. Note that “dist” is the distance from the top of the garbage container to the top of the trash (i.e., how much space is left in the garbage collector), not the height of the trash measured from the bottom of the container.

```
{
     “dist”: 4.85,
     “temp”: 75.56
}
```

Upon successful delivery, the microcontroller receives an HTTP response of 200 OK from the server, and the process repeats.

Web Server

The web server is where the majority of the processing occurs and is a central component of the system. The smart garbage web server has two main roles: handling data coming in from the embedded device and delivering the web application to the

client. Using the Flask web framework, these two scenarios can be handled with only a few lines of Python code. Consider this generalized example code below, where a GET request would come from a browser to receive the web application and a POST request would come from the embedded device sending the server updated data.

```
import flask

app = flask.Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def handle_request():
     if flask.request.method == 'POST':
          # Process data being sent
          return process_new_data()
     else:
          # Deliver web app content (a GET request)
          return render_web_app()
```

The code sample above represents the core of the server, beyond which numerous tasks then take place. When the server first receives an incoming GET request from a client, it delivers an HTML page the simply prompts the user to enter the height of the trash can in feet (shown on page 5 of this document). Once this data has been entered, it is sent through an HTML form to a specific resource on the server, at which point the server calls functions to set the height of the trash can and then deliver the main web application to the client. The server is now ready to begin storing data received from the garbage can and to begin sending updates to the client. The simplest of these updates is a calculation of what percent full the garbage collector is. Every time the server receives new data from the embedded device on the height of the trash (note that this also pertains to the temperature), it calculates the percentage full based upon the height entered by the user, and sends this data to the client application through the use of a websocket. Furthermore, every time that the trash can becomes 0% full or hits 100% full, a timestamp is taken and written to a JSON file on disk. The timestamps represent when the trash can was initially empty, when it became 100% full, and when it became empty again after just being taken out. Because this data is stored permanently, the server is requested to perform some data analytics everytime the page is rendered, calculating various averages based on the timestamps currently stored. These statistics can then be updated by the client at any time (see description of the Client Application below).

Client Application

This is the endpoint of the system and the portion that is visible to the user. This project incorporates a relatively simple web application that performs minimal processing and is mainly focused on displaying the data to the user in a helpful and easy-to-read way. The JavaScript code running on the client-side has two main roles: listen for any events triggered by the server to provide the client with updated data, and retrieve updated statistics from the server when requested by the user. The former is done through the use of websockets, while the latter is achieved through an AJAX request. The communication via the websocket is achieved through the Socket.IO library with only a few lines of JavaScript code.

```
var socket = io.connect("http://" + location.host + "/update");

socket.on("dist", function (data) {
     // Process data and update DOM
});
```

The code above established a websocket connection with the server on the "/update" namespace and listens for the server to emit an event with the name "dist." When the server triggers the event, the corresponding function is called with the payload from the server passed to it as an argument. With this data, the web application is able to display a graphic of a trash can illustrating the fill level. It also provides an updated reading of the temperature and the time that the trash was last taken out.

Regarding the updating of statistics calculated using data gathered overtime, this is retrieved from the server through the use of an AJAX (Asynchronous JavaScript And XML) request on the client-side. For the purposes of the garbage can monitor, this request happens when the page is rendered for the first time and everytime the user presses an "Update" button to download the latest statistics from the server. The corresponding JavaScript code looks similar to this:

```
var xhttp = new XMLHttpRequest();
var uri = “http://” + location.host + “/stats”;

xhttp.onreadystateexchange = function () {
     if (this.readystate == 4 && this.status == 200) {
          var obj = JSON.parse(xhttp.responseText);
          // Process data and update DOM
     }
};

xhttp.open(“GET”, uri, true);
xhttp.send();
```

When the server receives this request, it calculates new statistics based on the current data stored. It then sends back a JSON response to the client with various fields representing different statistics. An example of what the JSON object would look like is given below:

```
{
     “after-full”: 44138,
     “before-full”: 122998,
     “total”: 1010459,
     “highest-month”: 5
}
```

The time is delivered in seconds, and is converted into a more readable format by a JavaScript function before being displayed on the web page. These fields represent the data corresponding to the four statistics described in page 4 of this document.

Potential Future Directions/Changes

The purpose of this section is to address a few areas where the project falls short, and to discuss a few modifications that would be made if time permitted.

First, as referenced in the project proposal, it was intended for the project to incorporate both a GPS sensor and a load sensor. Unfortunately, in turned out that the code provided with the GPS sensor was not compatible with the Sparkfun Thing Dev board, as it was designed for use with Arduino. And regarding the load sensor, not enough components were acquired. According to the specification from the Sparkun website, four of that type of load sensor was required for it to function properly. If there were to be future modifications to this project, it would ideally incorporate a GPS sensor for tracking the location of the garbage bin. I would also possibly incorporate a load sensor in order to gather data on the weight of the trash produced, or to allow someone to be notified if something unusually heavy was placed in the bin.

Second, the method used for permanently storing data in the current version of this project is not ideal. Due to time constraints and inexperience working with databases, we opted to write the data to JSON files. In order for this project to be scalable, the data would ideally be stored in some sort of database where it could be queried. In a running system where a large amount of data has been produced overtime, it would be unreasonable to bring one extremely large JSON file into memory all at one time.

Finally, this project is currently able to deploy and monitor only one smart trash can. In a real-world implementation, the system would obviously need to accommodate several dumpsters or garbage collectors placed in arbitrary locations. Due to the way that it was designed, the server code could most likely be adapted to include several containers without a very large amount of modification.