Raspbeery Pi

(Or, No One Likes Running Out of Beer)

CSC453 - Software for Wireless Sensor Systems

May 3, 2017

Final Project of Team 7:

Wyatt Maxey
Curtis Moore
Joseph Stanley
Alex Wimer

Effort Breakdown

Component	Weight	Wyatt Maxey	Joseph Stanley	Curtis Moore	Alex Wimer
Design	.25	20%	40%	20%	20%
Hardware	.10	25%	25%	25%	25%
Coding	.30	25%	45%	15%	15%
Debugging	.05	30%	30%	20%	20%
Report	.20	30%	20%	20%	30%
Presentation	.10	15%	15%	35%	35%
Aggregate Contribution	-	.25*20+.1*25+.3*25+.05*30+.2*30+.1*.15=**22.5**	.25*40+.1*25+.3*45+.05*30+.2*20+.1*15=**33.0**	.25*20+.1*25+.3*15+.05*20+.2*20+.1*35=**20.5**	.25*20+.1*25+.3*15+.05*20+.2*30+.1*35 = **22.5**

Introduction

The Raspbeery Pi project is intended to provide an easy-to-use interface by which anyone using a central drink dispensing mechanism may efficiently, accurately, and effectively monitor the remaining amount of a given beverage, estimate when the supply will be depleted, and order more of the beverage. Example uses include bar taps, fountain drink dispensers, and homemade kegged drink dispensers. The inspiration for the project came from one of the team members who homebrews as a hobby. The homebrew community is full of people who have a real do-it-yourself attitude and are avid tinkerers. Our team member originally wanted to create a homebrew monitoring system, but found it highlighted on the “Project Examples” document.

Not wanting to stray from the idea of integrating some sort of way to monitor his homebrewed beer, the team member thought about the idea of tracking the amount of beer left in a keg without having to open the keg and look or just give it a “best guess.” Knowing how much beer is left from a brew is crucially important to a homebrewer because it can take up to two months to complete another brew, which means there could be a large gap without any beer on tap! That would obviously be a living nightmare.

Knowing exactly how much beer is left is a great step in the right direction, but something even more helpful would be the ability to have a mathematically sound estimate of how many days of beer are left. This would make it much easier for our homebrewer to know what the ideal date to start the next brew on is. This application seemed like a great place to make use of the analytics material we learned in lecture. We also wanted to algorithm to not be static, because rarely is this the case in a real life application. The algorithm also uses the final amount used from the tap in a given day to adjust its prediction calculation. This allows the algorithm to become more accurate and adjust to the use habits of its client. This made sense to us because use amounts would obviously vary among users in a real life application. A user in a homebrew setting would (normally at least) go through beer at a much slower rate than a sports bar.

Our team also wanted to place an emphasis on the user interface. While we all may feel at home interacting with software via a command line interface, most end users would not find this practical. We wanted to create and end-user interface that would be sleek, simple, and most of all, practical. The interface needed to display a variety of information and make sure that the information was displayed in a manner so that every data point would have a place and be useful to the user. We wanted the user interface to be something that a homebrewer could proudly display in his or her basement bar, and something that bartenders or shift managers at a bar would be able to quickly glance at and determine if they needed to swap a keg out.

Design

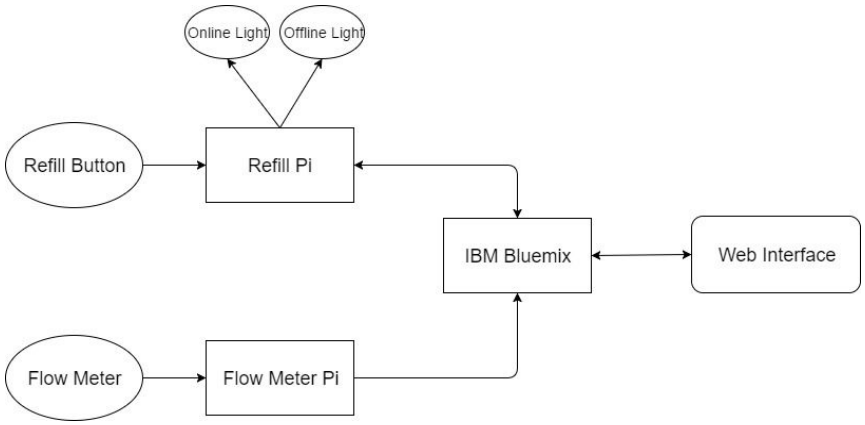

Figure 1. Block diagram displaying the overall system design and flow of information between each component.

Figure 1 shows a snapshot of our full prototype system. The user facing elements in our prototype are the flow meter, the physical refill buttons, the online and offline status lights, and the web interface. We also currently use a Raspberry Pi dedicated to managing one or more flow meters, a Raspberry Pi connected to both the refill buttons and status lights, and also an IBM Bluemix MQTT broker to handle the data passed between each component.

The general flow of information involves a customer pouring a drink, which triggers the associated flowmeter pi to transmit the amount poured to the Bluemix broker. The amount remaining is then updated and correctly displayed via the web interface. As use of the system continues, the containers will naturally run low on stock and must be refilled. Once a container is refilled, an attendant should press the appropriate refill button to indicate the container has been filled and the interface will be updated automatically. Internal rules on Bluemix will then determine if a new supply of that container needs to be ordered (or brewed) depending on the shipping time. Should the product need to be ordered, a request will be sent to the supplier or emailed to the brewmaster. A table is visible on the web page displaying the name of the beverage, the amount currently “on tap”, or the amount of liquid remaining in the container that is connected to the flow meter, the amount of liquid remaining as a whole stock, the estimated days of liquid remaining in stock, and the date of the last stock order.

The user has the ability to change the display names for the three beverages on tap. This feature is present for the user’s benefit, but it does not change the underlying functionality of the program and is optional. The user can also enter “starting values” for the stock of each beverage. This is intended to be used when the system is first installed. During normal use of the system, the only parameters that end users will affect will be the dispensing of the drink and marking a container as refilled. The rest of the calculations, predictions and metrics will be automatically handled by the underlying software. It is important to note that, as users are human, we have provided a method by which they may manually correct stock or predictions in the event the refill button was neglected.

Implementation

Python was the primary language used by our team. Python was used when interfacing with any lights or sensors. It was also used as a pseudo controller for the web interface when interfacing with the IBM Bluemix database and as the main program that executes on the Bluemix platform. Javascript was used within the web interface to interact with the Python pseudo controller. It was also used to retrieve values entered by the user on the web interface and to set data variables that were displayed for the end user. HTML and css were used to create the static web pages and design the end user look and feel. Finally, we used the poissrnd() function in Matlab along with our own predicted drink consumption values to create a simulated mean consumption over a month. The exact files and descriptions we used are:

HTML

- *index.html*: displays live data about each beverage to the end user using a table
- *splash.html*: displays an initial loading animation
- *usage.html*: displays usage charts per beverage
- *beverage.html*: displays information for currently monitored beverage in a table
- *control.html*: control panel for the connected flowmeters
- *system.html*: allows the user to update information per beverage. Intended for use with starting values

Python

- *web_backing.py*: a controller for the web interface. Interacts with the IBM Bluemix database to add and retrieve information about beverages being monitored and ready the information to be displayed for the end user
- *web_app.py*: application that runs on IBM Bluemix. Where other processes give and receive information from Bluemix
- *client_dispense.py*: connects to Bluemix and runs the dispenser application (below)
- *client_inventory.py*: sets physical status lights based on how much of each beverage remains
- *dispenser.py*: runs on clients to monitor the state of the flowmeters. Determines when liquid is being dispensed and measures the amount being dispensed
- *setup.py*: Bluemix setup file for running the app on Bluemix

Other

- *requirements.txt*: Python modules to install that are used by the app
- *Procfile*: routes HTTP requests to the running application
- *manifest.yml*: uses by CloudFoundry when pushing the application to Bluemix
- *app.cfg*: Bluemix configuration file
- *bev1.cfg*: Bluemix configuration file for beverage 1 client
- *bev2.cfg*: Bluemix configuration file for beverage 2 client
- *bev3.cfg*: Bluemix configuration file for beverage 3 client
- *bev4.cfg*: Bluemix configuration file for beverage 4 client
- *stock.cfg*: Bluemix configuration file for device monitoring stoc

Results and Discussion

Goal Formula: $T = mA + b$

$T = Days\ until\ stock\ is\ depleted$

$A = Amount\ of\ beer\ in\ stock$

Number of Customers per Day:	150 customers
Number of Beers Purchased per Customer:	2 beers
Number of Gallons per Beer:	0.125 gallons (or 1 pint)

$$\frac{150\ customers}{1\ day} \times \frac{2\ beers}{1\ customer} \times \frac{0.125\ gallons}{1\ beer} = \frac{37.5\ gallons}{1\ day}$$

Probability of Purchasing Beer #1:	0.5
Probability of Purchasing Beer #2:	0.25
Probability of Purchasing Beer #3:	0.25

Expected Gallons of Beer #1 Purchased (λ_1):	18.75 gallons per day
Expected Gallons of Beer #2 Purchased (λ_2):	9.375 gallons per day
Expected Gallons of Beer #3 Purchased (λ_3):	9.375 gallons per day

With the Matlab function poissrnd(lambda, 1, 30), a simulated mean value over the period of a month can be generated using the expected values above:

Beer #1:	$\mu_1 = 19.233$	$T_1 = 0.0520 \cdot A_1$
Beer #2:	$\mu_2 = 9.2667$	$T_2 = 0.1079 \cdot A_2$
Beer #3:	$\mu_3 = 10.033$	$T_3 = 0.0997 \cdot A_3$

Note that no y-intercept is added because the time left when amount is zero is zero days

Using these equations, we would recommend purchasing 3 barrel kegs (31 gallons per keg) for each order. An order of this size will last approximately one to two weeks. An actual restaurant or bar could update these values and probabilities to more closely match their actual statistics and create a more accurate consumption prediction.

One observation about the design of our prototype is that we were somewhat limited in our hardware capacity. For a real-world implementation of this system, the hardware would appear differently than what we created and should match the environment it exists in. Ideally there would be multiple flowmeters, refill buttons, and status lights all connected to one Pi which ‘controls’ multiple drink containers. The refill buttons and status lights should be attached or nearby to the container which they correspond to in order to increase usability and reduce user error. This setup would still allow many different drinks to be monitored from one point while remaining scalable and easily maintainable.

Related Work and References

https://learn.adafruit.com/adafruit-keg-bot/install-flow-meter - “Install Flowmeter”
https://github.com/IBM-Bluemix/get-started-python - “IBM-Bluemix/get-started-python”
https://codepen.io/TimRuby/pen/jcLia - “CSS Animated Beer Pour (Forked from CSS Beaker Pen)”