Kitchen Appliance Automation: Roast Right

Evan Baytan, Patrick Sites, Samuel Roman and Brian Webb

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* —The Internet of Things (IoT) has become a strong focus this decade for every day electronics to have network capabilities for user interaction and control. One particular subset being home automation devices or appliances. Roast Right aims to provide a convenient and easy to use interface for wirelessly roasting fresh coffee beans so consumers can enjoy a substantially better coffee experience in their own homes. The interfaces closely analyzed include: iOS Application, Web API, Weight Sensors, Power System, Wi-Fi System and LCD Screen. All systems interact through a central hub to then be controlled and monitored by an iOS application.**

***Index Terms* — Home automation, wireless, User centered design, high power, appliances**

I. Introduction

RoastRight: The Smart Coffee Roaster is designed to give users the ability to quickly and easily make some superb coffee roasts in the comfort of their home. It combines the power of the Behmor 1600 Plus coffee roaster – The ability to achieve darker roasts and to roast up to a pound of coffee at a time – with the use of modern communications and multiple input, multiple output control systems. The RoastRight roaster not only senses temperature of the roasting chamber, it also senses how the weight of the beans changes over time and is capable of determining where the beans are in the roasting process. These control systems give users the ability to easily and accurately achieve the exact roast they are looking for, every time!

By combining these control systems with internet connectivity for control and monitoring with your iPhone or Apple Watch, the RoastRight coffee roaster brings coffee roasting into the modern era and provides new levels of customization and ease of use to an industry that has seen little innovation in the past two decades. The apps allow you to build and share custom roasting profiles with your friends so that other RoastRight users can easily replicate your results and try the same coffee that you had that very morning

Whether you source your beans from Brazil or Thailand; whether they have followed a dry process, wet process, or even a more exotic process such as a honey process; whether you prefer a light roast or a dark roast, the RoastRight coffee roaster can handle it. RoastRight can handle any variety of coffee bean and produce the exact roast that you were looking for, every time. Want to try the world's best cup of coffee? Look no further than RoastRight!

II. ROASTER

The core functionality of Roast Right exists through the coffee roaster kitchen appliance. Aside from the front-end iOS application, users will be viewing information about a roast that is in progress through the onboard touch screen display. Wi-Fi, display, weight sensors and multiple core subsystems are powered by an ATMEL ATMEGA2560 MCU. The front-end and onboard interfaces communicate with the coffee roaster through HTTP requests that send commands back to the MCU such as starting a coffee roast or retrieving the status of the coffee roaster itself.

A. Power System

This design will incorporate a power system with multiple power rails. There is a 5 volt rail to power our digital electronics including the microcontroller, WiFi, and LCD. This rail also powers analog systems including load cells and instrumentation amplifiers. This rail has the option of being run off of battery in order to preserve roast data in the event of a power loss – if power is lost while performing a roast, logic is able to remain running so that any data collected during the roast can be transmitted to the server and stored. This rail will need to provide 300mA of sustained current and peaks of up to 600mA.

The second power rail provides 12 volts run some aspects of the control systems. Some parts of the roaster are controlled by relays with a switching voltage of 12 volts. This rail will need to be able to supply up to 550mA of current.

To protect both the circuit and users from damaging and dangerous over current events, we have a 250mA fuse going from the wall into the transformer. There is also over current protection on chip for the 5 volt rail to protect the chip from damage in over current events.

Figure 1, below depicts the completed power system.

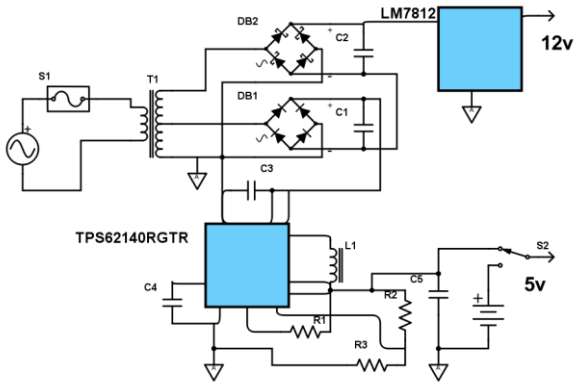

Figure 1. Completed power system. Some elements of the design were given to us courtesy of TI.

B. Thermal Control System

Our design calls for the retrofitting of an existing coffee roaster. This roaster has a variety of components that affect thermal properties of it. There are two different heating elements. The primary heating element provides 1,500 watts of heat in order to quickly increase the temperature of the roasting chamber. The secondary heating element produces 600 watts of heat and is used to burn smoke in the exhaust and produce a more complete combustion, thus lowering the amounts of potentially harmful emissions from the roaster. The whole roaster is designed to pull up to 1,560 watts. In order to meet this power requirement, only one heating element may operate at any given time.

Beyond heating elements, there are also two blower motors. An inlet blower and an outlet blower. The outlet blower turns on at the same time that the secondary element turns on. This begins to allow for hot exhaust out the back of the unit while not having a large effect on roaster temperature due to both the increase of heat added by the secondary element and increase of heat from chemicals in the coffee beans beginning to burn. Once the temperature is at the desired level, we can turn on the inlet blower fan to pull in a large volume of cool air. This will begin to lower the temperature of the roasting chamber. The secondary heating element will remain on to continue burning off smoke until temperatures fall below a threshold temperature where the beans will produce less smoke. At this point, the roasting chamber will begin to quickly cool down.

The heating elements are both controlled by solid state relays. They have a simple on/off control that is able to operate directly off the 5 volt digital signal provided by the microcontroller.

The blower fans have a more complicated control structure. They are primarily controlled via triacs. These triacs have snubber circuits to prevent the inductive nature of the motors from continually triggering the circuit. The gates on the triacs are rated to run from 0.7 volts to 1 volt. In order to operate these triacs without the need to implement an additional power rail, we used pulse width modulation from the microcontroller with a low pass filter to realize a rudimentary DAC. This allowed us to achieve a constant 0.72 volt output from the microcontroller. Without the LPF, there would be 5 volt spikes and our triacs would have quickly been destroyed. With this implementation, we can treat these PWM outputs as simply digital high or low to allow for on/off operation of the blower motors by setting the PWM to either a zero duty cycle or a low duty cycle.

We will be using a K-type thermocouple, which will be mounted in the roasting chamber, to measure the current temperature of our system. Thermocouples are easy to implement and accurate in a wide range of temperatures vastly exceeding what we will be handling. In conjunction with our thermocouple temperature sensor, we will incorporate a cooling fan to help with ventilation inside the roasting chamber. As the temperature reaches the desired range, we will monitor any changes and either operate the heating elements or cooling fans depending on the value read from the thermocouple.

Information from this sensor is used by a software-based control system to control the four elements that effect our thermal system.

C. Additional Controls

The internal light and the drum of the roaster also need to be controlled from the microcontroller. These elements require 12 volts, which cannot be provided directly from the microcontroller. In order to overcome this obstacle, we implemented optoisolators. Optoisolators offer the benefit of allowing these elements to be operated from the microcontroller while still being powered from our 12 volt rail. When these elements are on, a simple output from the microcontroller's digital pins will turn on the optoisolators and enable the 12 volt rail to operate relays which turn on the light or rotate the drum at any time throughout the roasting process.

D. Weight Sensing

As coffee beans begin to roast, the water inside of them is released in the form of steam. A key component to a quality roast is being able to know when the beans have released the optimal amount of water for the type of roast.

To measure this slight change in water content, we will be using 4 YZC-131 load cells. Each load cell has a rating of five kilograms, with a maximum rating of 6.5 kilograms. Each of these load cells will be place under one of the feet of the coffee roaster. We will tare/zero the weight sensors when the roaster is empty so that we can accurately read the weight of the coffee beans.

Load cells operate based on the differing resistance in a Wheatstone Bridge. As the cell is bent from the weight, the outputs from the load cell change. This output is incredibly small, however, due to only the slight bending of the beam. In order to accurately measure the outputs of the load cells, they will pass through an instrumentation amplifier with a gain of 100. The instrumentation amplifier will take the difference in the two outputs and magnify it by the gain. Each instrumentation amplifier will them be fed into the microcontroller's analog-to-digital converter. As the coffee roasts, we will be measuring the outputs of the instrumentation amplifiers to ensure that they are not overcooked for the selected type of roast.

III. Software Integration Overview

From both a hardware and software perspective, there is multiple handoffs or transferring of communications in order for a user to go from starting a roast in the iOS application to the end product which is roasted coffee beans. On the software side of things, the communication for a successful user request is performed across 3 systems: the front-end iOS application, back-end Web API and embedded Arduino programming for both Wi-Fi and the LCD Touch Screen. The user only interacts with the iOS application when viewing roasting data and controlling the roaster, while there is numerous systems running behind the scenes to make it all happen. An overview of the software infrastructure between the iOS application and coffee roaster can be seen in Figure 2.

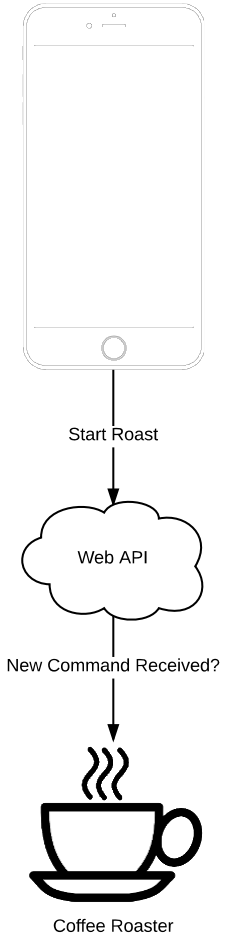

Fig. 2: Software Integration Overview

The primary motivation for this project was to allow users the ability to control and monitor the coffee roaster wirelessly for added convenience and usability through a simple to use iOS application. The iOS application is the client that interacts with the other noted systems as the first point of contact. For example, the user of the iOS application will be able to view information about the various coffee roast types and then eventually decide which of the three to begin a roast for. Various amounts of user data and roast data is being retrieved from the Web API through GET and POST requests. For example, if a user completes a coffee roast, the iOS application will perform a POST request to add to the "History" view that is accessible from a slide out menu. More importantly, if a user starts a coffee roast, a POST request is sent to the Web API to change the roaster mode to "start" and begin the transfer of communication to the Web API and MCU.

While this is occurring, the MCU is continuingly performing GET requests to check if a new command has been received. This is able to be achieved through the Wi-Fi shield and MCU communicating through the embedded programming. The JSON data on the Web API will be modified when the request is made from the iOS application, more specifically, the value for a "roastingStatus" variable changes to the respective mode. Once the roaster or MCU performs a GET request and see's that the mode has changed, it will then begin roasting and also send a POST request back to the Web API indicating the change in mode to differentiate between client and roaster. The fact that the roaster is performing these GET requests to check for a new command being passed made for a somewhat simple interface between the roaster and the Web API in order to be controlled effectively and consistently.

The various systems of communication between the front-end iOS application and the embedded software through Arduino and the MCU make up the infrastructure for controlling the roaster wirelessly. All of the user interaction is within the iOS application, while the "under the hood" processing and computation is done through the Web API and MCU when processing requests and being able to translate requests to hardware control.

Integration testing involving the various software systems was performed to verify the integrity and responsiveness of the communication being performed. A POST request was made from the iOS application to start a coffee roast. The Web API was then updated with the new JSON data and the command being sent. While this is occurring, the MCU is polling the Web API for updates on the status and then controls were updated accordingly.

IV. FRONT-END SOFTWARE

The primary intended operation of the coffee roaster is through the iOS application. The core functionality and usability of the application was designed in such a way to be familiar and user-friendly in order to attract and maintain users. Decisions to incorporate similar user interfaces and user experiences from popularly used applications on the App Store furthered the motivation and integration of the application experience as an entirety.

A high level view of the application workflow can be seen in Figure 3. A user will need to create an account or sign in with their existing account to be able to use the application due to the authentication token that is generated upon account creation and sign in. Once a user is signed in, they can then view the three different roast types and a featured section of different roasts. Numerous view controllers are used for the various content that is displayed to the user from information regarding a roast type or viewing a roast in progress.

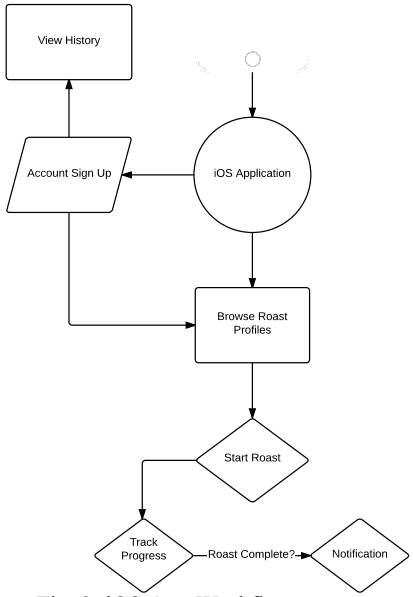

Fig. 3: iOS App Workflow

A. Home Screen & Slide Out Menu View Controllers

Users will be greeted to a simple to navigate interface on the home screen of the iOS application. Figure 4 shows the current version of the home screen running in the Xcode IDE simulator. The menu bar along the top consists of a slide out menu button on the left that will trigger the slide out menu view controller, as can be seen in Figure 5. Beneath the menu bar consists of four large buttons taking up the rest of the screen real state. Users will be able to quickly navigate to a featured roast view controller or select one of the three roast types: Light, Medium and Dark.

In order to create a clean and simple interface, various buttons are hidden in the slide out menu view controller:

Sign Up, Login, History, Favorites and Settings. Users can select any of the buttons to be taken to their respective view controllers. The current design for the slide out menu is planned to be improved to include familiar icons for the respective button names, such as a heart for the Favorites button or a gear for the Settings button, etc.

Fig. 4: Home Page VC

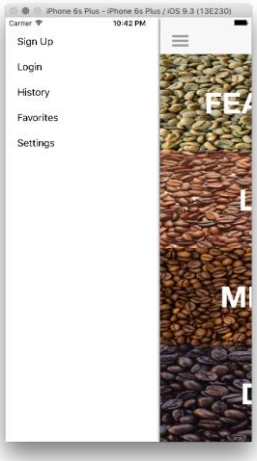

Fig. 5: Slide Out Menu VC

B. Roast Type & Roast In Progress View Controllers

Users will have access to the roast type view controller once they choose between one of the three coffee roast types on the home screen view controller: Light, Medium and Dark. Figures 6 and 7 show the current version of the roast type view controller running in the Xcode IDE simulator. A description of the roast type and the average time to complete a roast of the respective type is displayed

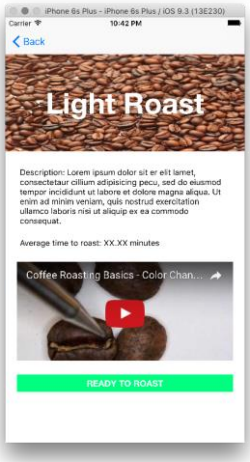

Fig. 6: Roast Type VC

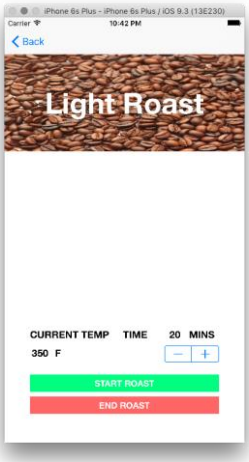

Fig. 7: Roast In Progress VC

in the center of the screen. Beneath both previously noted is an embedded YouTube video that will show the respective roast from start to finish for the user to view prior to them starting a roast of that type. The choice to have embedded media in the application was made to engage users with the application in ways that would help them understand what the end product will look like for the coffee roast.

After the user decides which roast type they want to complete, they can then touch on the "READY TO ROAST" button on the bottom of the roast type view controller. The user will then be taken to the roast in progress view controller in which they will be able to view a static graph in the center of the screen showing the temperature versus time that fluctuates throughout the roasting process. The roasting process for the three different roast types have unique temperature adjustments over time, also known as roast profiles. Beneath the graph will be three fields for current temperature being read from the roasters internal temperature sensor, time remaining for a roast to be completed and the time duration for a roast that can be adjusted with the plus and minus buttons accordingly. Once the user confirms the time duration for the roast, they can then touch on the "START ROAST" button to send the request to the Web API, and then send commands to the roaster to begin the roasting process. A notification will appear on the screen indicating when a roast is complete and the screen will then transition the user back to the home screen view controller.

V. WEB API

In order to 1) allow communication between the iOS application and the roaster, 2) store the user's data, and 3) provide user data integrity we created a RESTful Web API service based in NodeJS.

NodeJS shines in real-time applications by making use of push technologies specifically Websockets, which allow the client and the server to exchange data bi-directionally freely. Furthermore, NodeJS is heavily asynchronous meaning than rather than waiting for data to be received from an API call it would move on instead and simply go back when the data is available thus reducing down time and increasing the overall performance.

A. Data Integrity

One of the primary objectives of the Web API was ensuring that the only user that was able to interact with our roaster via the iOS application was the owner of said roaster. In order to do so we ended up incorporating a technology called JWT.

The JWT essentially allows us to create a unique string for the user that would then allow use to verify the identity of the user and fetch or push the relevant data. The string is essentially an encoded version of the user data that in our case stores the user's username. The user would be able to acquire this string by successfully logging in via a POST request to the /api/user/authenticate route. This value would then be stored in local storage and placed in the header for future requests.

The token would be used whenever we would need to reference the individual making the request. For example, if a user was attempting to start a roast, the client would need to send the token in the header along in the header. We would then be able to update the appropriate user's roast data without fear of altering a random individual's roaster status.

B. Endpoint Design

Keeping in mind that the consumer of the API would be another developer my next objective was ensuring that the endpoints were fairly intuitive.

One of the core functionality of the Web API is allowing communication between the iOS application and the roaster. This is started by the iOS application making a POST request to '/api/control/client?mode=start& roastData=French Roast', which allows us to start a French roast by essentially telling the API the fact that the client would like to start a roast. On the roaster side it would be continuously doing a GET request to '/api/control/ which would allow us to determine if a new command has been submitted. If there has been a new command submitted, we would make a new POST request to the API essentially just verifying that the command was or was not executed. This would be done via a similar roast to the one seen above, POST '/api/control/roaster?mode=start& roastData= French Roast'.

C. Hosting

Given that we opted to design our own backend rather than going with a prebuilt system we needed to find a cloud storage provider to host our Web API, in order to allow the clients—in this case client—to connect from anywhere at any time. Our options were between GCP (Google Cloud Provider), AWS (Amazon Web Services), and Azure.

In the end we chose to go with AWS for two reasons. One given that we were still students we are able to receive a year of AWS basic for free which includes a basic EC2 server. Two the market share is heavily in favor of AWS, as can be seen in figure 8. This led us to conclude that experience in AWS would be substantially more marketable to our future careers.

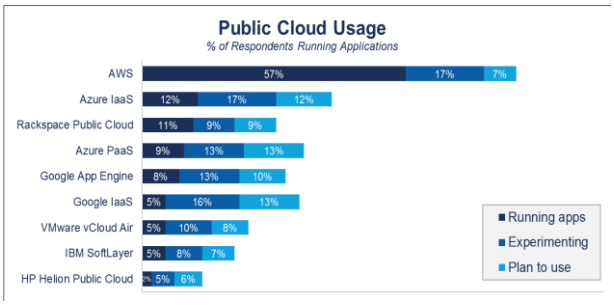

Fig. 8: Cloud Market Share

VI. EMBEDDED SOFTWARE

The embedded programming and development was divided up between two of the CPE members in the group. The division of embedded software development was distributed based on the two respective interfaces that are interfaced directly with the ATMEGA 2560 MCU: Wi-Fi Shield and LCD Touch Screen. Both interfaces communicate directly with the Arduino Mega and programs or "sketches" can be uploaded through a normal USB 2.0 connection via a computer.

A. Wi-Fi Shield

The HTTP requests that are made by the iOS application and LCD touch screen interfaces communicate to the roaster through the Wi-Fi shield. The user can view a roast in progress using the LCD touch screen for information tracking without an iOS device.

Wire connections between the CC3000 Wi-Fi Shield and the PCB is made capable by 6 digital pins, 6 digital ICSP pins, VCC and GND. The 6 digital pins used include: SCK (#13), MISO (#12), MOSI (#11), CS (#10), VBAT_EN (#5) and IRQ (#3). The 6 digital ICSP pins are shared with the LCD touch screen for programmability. The Wi-Fi shield is powered and grounded by the PCB.

Once a GET or POST request is made, an Arduino JSON library is used to parse the JSON data from the request to be able to determine the values for the various body values. For example, in order to correctly set the mode for the roaster to start, pause or stop, the JSON data value for "roastingStatus" is parsed from the request in order to determine the correct POST request to send from the roaster to the Web API. The iOS application and LCD touch screen send unique POST requests indicating the handoff and communication across the software infrastructure and to also better manage failures that may

occur across the numerous hardware and software standards being utilized for the project.

In order to setup the client and allow for the CC3000 to connect with a wireless access point, an Adafruit CC3000 library was utilized. The library supports GET and POST requests, with support for custom headers that are used for the authorization key that is established for a valid user account. The various GET and POST requests being used have unique URI's or unique resource identifiers to indicate the different web pages respective to the command(s) that the user performs. Whether it is retrieving the current status of the roaster, or changing the mode from start to stop, each request will have a unique web page link to differentiate from. The data itself is being read through serial and the client will continue to do so until the connection is closed or a predetermined idle timeout is reached.

B. LCD

In order for the user to a have an aesthetically pleasing manner of determining the current roaster state we have implemented an LCD display which will update depending on the most current command given.

If the roaster did not have a current command to execute, the LCD would show the idle view which is simply the Roast Right logo. Once we have received a new command with the roast to be started we would change the view to Roasting which would show the user the specific information on the roast such as the name of the roast, the bean type the roast is meant for, and finally an ETA.

C. Control Lines & Data

The MCU is tasked with reading and writing to multiple motors and sensors and balancing them in such a way that performance is maintained relatively speaking.

In order to roast beans, we would obviously require some heat and in our case in the form of two heating elements. In order to control the state of the elements of the heating elements we are wiring them to digital out pins which simply a HIGH value if we would like to turn them on and set it to LOW if you would like to be off.

In order to determine whether we should turn on and off the heating elements we are making use of a temperature couple which allows us to determine at what point the current temperature is 10 degrees above our target temperature and when this is the case simply turn off the element as described above. The temperature itself is a digital signal so reading the value is a matter of doing a digitalRead(); on the corresponding pin.

Furthermore, in order to properly roast the beans, we would need to determine the weight of the beans that way we can either increase the or decrease the duration of the roast if necessary. This is done via the 4 load sensors that are able to measure up to 5 kg each. The values are received as an analog signal and are then converted into a Digital signal via the ADC which then are converted into a weight. Once we have all 4 weights we would simply add all 4 up and determine whether the roast would need to be modified.

VII. CONCLUSION

The RoastRight coffee roaster combines a combination of front end, back end, and embedded software in order to provide a complete coffee roasting software package that is tailored to work with the specific hardware that was implemented. The hardware design consists of both an existing product, as well as many custom systems that were integrated into a single package that is capable of producing better coffee with better control than other products on the market. The use of an existing product allowed the opportunity to analyze and modify completed designs that others had implemented without the benefit of any documentation as to what they had done.

VIII. GROUP MEMBERS

Samuel Roman is a Senior Computer Engineering Student at the University of Central Florida. He will be receiving his Bachelor of Science in Computer Engineering in August 2016. His interests include software development in general, video games, and everything nerdy. He will be taking a full time position as a Software Engineer at Millennium Engineering and Integration Company in Melbourne, Florida in mid-August.

Evan Baytan is a senior at the University of Central Florida. He will be receiving his Bachelor of Science in Computer Engineering in August 2016. His interests include iOS development and video games with friends. Evan is currently interviewing for a QA Engineer position with Apple.

Brian Webb is a senior at the University of Central Florida. He will be receiving his Bachelor of Science in Electrical Engineering in August 2016. His interests include playing bass, guitar or drums; learning about and observing space flight, baking and photography. Brian looks to pursue his Master's Degree while working in a professional environment. He has interviewed with US Airforce civilian engineering and would love to work in defense or aerospace.

Patrick Sites is a senior of the University of Central Florida. He will be receiving his Bachelor of Science in Electrical Engineering in August 2016. His interests include diving, biking and mycology. Patrick is planning on working with NASA for Satellite Simulations.