Voice-Enabled System for Power Allocation (VESPA)

Kaden Strand and Michael Thanh

Department of Electrical and Computer Engineering
Colorado State University
Fort Collins, CO 80523

{kstrand, mikeyvxt}@rams.colostate.edu

***Abstract* – This project is the exploration and development of a voice-activated control system for powering devices on and off remotely. The advantages of VESPA over other systems include voice recognition, low-power control systems such as the Raspberry Pi computer, and no dependency on mobile devices, internet, or radio technology.**

Index Terms – embedded, voice-recognition, power, remote

I. Introduction

As the general pace of everyday life continues to increase, scientists and engineers have turned to designing tools that help increase productivity. By saving even a little bit of time, whether that time is faster processing speed on a chip or time saved by reducing necessary physical tasks, overall progress and efficiency may increase dramatically.

In this project, we explore the creation of an embedded system device that will serve to save time in the physical world by allowing users to control power throughout their house without physically standing up or moving. The motivation behind this stems from experience as well as observation – when stationed at a table or office working, it can become distracting and time-consuming to stand up and turn devices in the house on or off. In particular, it can disrupt the flow of concentration and thinking that a user is currently experiencing. By creating a system that can immediately control any other device at any other location in the house, mundane tasks that are not worth interrupting high levels of concentration can be done without interrupting workflow.

II. Related Products

The general idea of a system that can power devices from long distances is not new. In the past few decades, many devices have been created to help make people's lives more convenient, which is almost a necessity [1]. However, one of the biggest disadvantages of a remote-controlled device is in the name – the remote. If the remote is not located nearby, it defeats the entire purpose of the device in the first place. In 1985, Joseph Enterprises, Inc. introduced the first market-released item that could control devices without a remote: the Clapper.

The Clapper is a sound-activated electrical switch and works with any Standard American outlet. It comes equipped with a microphone designed to handle a specific range of frequencies with a built-in bandpass filter as well as a small embedded system to detect differing acoustic patterns to activate corresponding switches [1].

Although the clapper was innovative in its attempt to eliminate the need for a remote, it falls short in a couple of ways:

1) Inconsistency: Although the design of the bandpass filter covers a good range of frequencies, it does not account for acoustic discrepancies. A person with outlying hand size or a room with high reverb may cause the system to behave erratically.

2) Versatility: The system in itself is designed to only control two devices, and each switch only toggles when it receives the corresponding signal.

3) Physical Appearance: Aside from some consumers' embarrassment at clapping at nothing, the device itself is somewhat large, requiring to be plugged into the bottom socket of a wall outlet. It cannot be used to control power to a built-in wall switch unless extensive rewiring is done.

Aside from the Clapper, more recent devices and applications have been created to better automate the home. Z-Wave is a line of products for total home control and automation by a company called Sigma Designs, Inc. Some notable brands and companies using Z-Wave are AT&T's Digital Life, ADT Pulse, Schlage Door Locks, and Nexia Home Intelligence, among many others [2]. Although it is powerful and capable of total automation of the home, it is costly and requires the user to have a mobile device with internet connection. Furthermore, all connected devices in the house must either have their own WiFi connection, or they must be connected to a central system with internet. Insteon is another company that creates devices for home automation, but like Z-Wave, it requires a mobile device and internet access.

For our project, we propose a novel system that requires no remote, no mobile device, and no internet or WiFi. Rather, this embedded system relies solely on electricity and the insertion of noise into the AC circuit of the home.

III. Design

A. Hardware

The beginning of our design was based around using an AC wall intercom to communicate with other locations in the house. The intercoms we used are Westinghouse Novi-Wi2C systems, designed simply to accommodate voice communication between people in a household environment.

By examining the circuits of these intercoms, we were able to discern how the devices worked: at all times, the

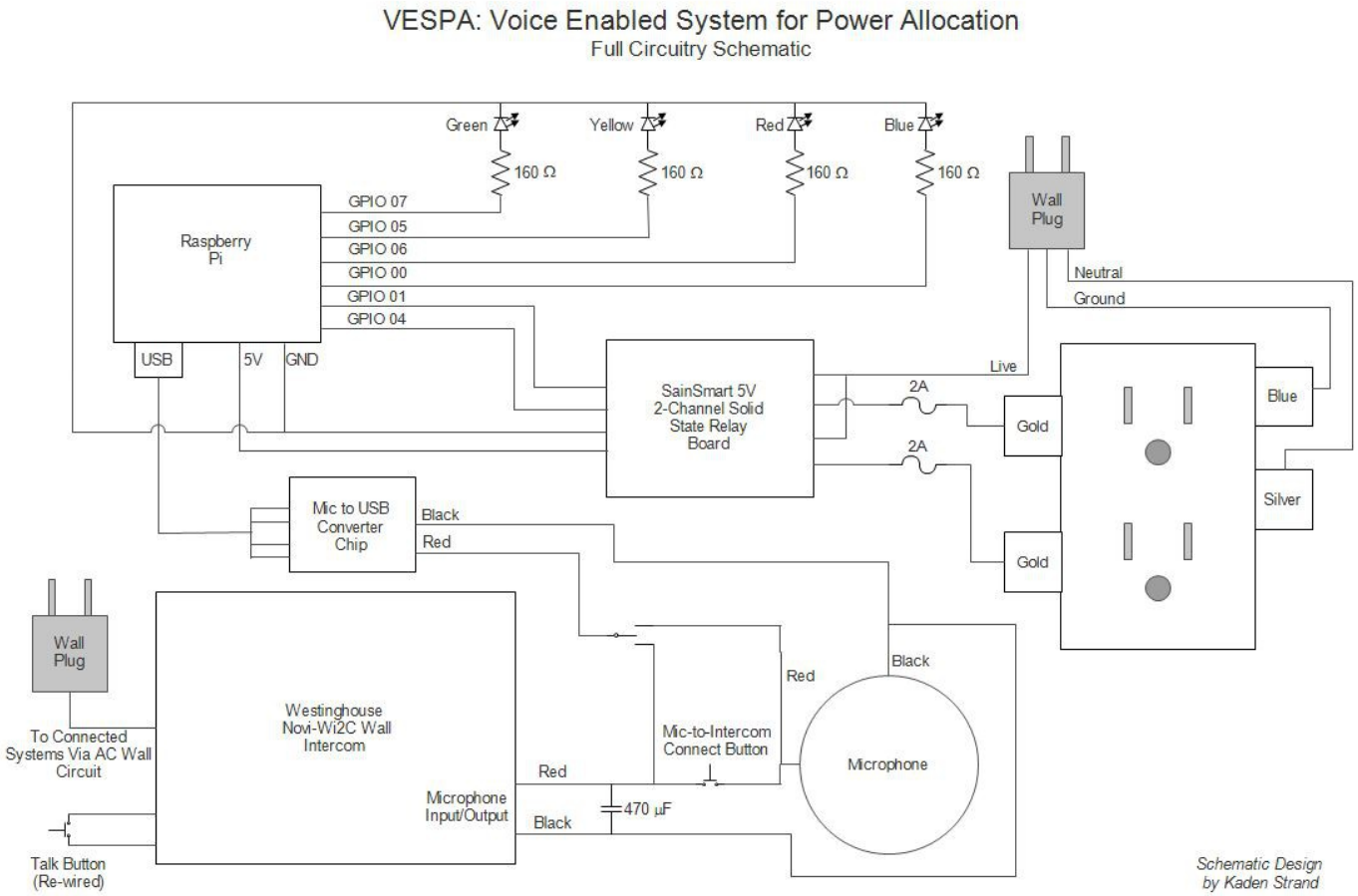

Figure 1: Full Schematic of VESPA Circuitry

singular driver connected within the system is always receiving a signal from other devices. To send a signal, a user presses either the 'Talk' button or the 'Lock' button, and this reverses the polarity of the circuit connected to the driver. This turns the driver into a microphone. At this point, all noises and vibrations are input to the system and converted into high frequency noise. This noise is then added to the wall socket electrical connection so that other intercoms plugged into the same house circuit may receive the noise. Once another intercom receives the noise, it decodes the received signal and then converts it back to a digital signal. Finally, the digital signal is output to the driver, which is polarized as a speaker to produce the sound of the user's voice.

By choosing to use these intercoms, we have solved the initial challenge of delivering a signal to an entire house without using internet, mobile devices, or radio technology. Moreover, by changing the interpreter on the end of each system from a person to a computer, we would be able to issue voice commands to accomplish a variety of tasks by computing what commands were given. For our system, we chose to use Raspberry Pi computers to process the voice input.

Raspberry Pi is the best choice for this type of system because it is small, lightweight, and still powerful enough to run speech recognition software. In addition, the Raspberry Pi also includes General Purpose Input/Output (GPIO) pins which have the capability to act as binary toggles to output voltage. The next problem we encountered, however, was how to use these pins to toggle AC wall outlets. Since these outlets would be powering everyday appliances, there would be high levels of current and voltage that the Raspberry Pi could not safely control and thereby be capable of damaging the computers.

Our solution to this was to integrate power relays within each wall sockets. A relay works by connecting two distinct circuits together. The switch end, which receives input from a Raspberry Pi output pin in our system, controls whether or not the relay's internal electromagnetic coil conducts a current. On the other end is a wall socket connected to the house AC power. When the coil is conducting current, it will attract and close the switch, thereby turning the appliance on. By toggling our Pi's GPIO pins through this relay, we can safely switch on and off the wall socket without damaging the Raspberry Pi.

Each Pi needs to receive input from both a local microphone as well as incoming signals from another system. Since the Pi does not have an input audio jack, we needed to use a USB microphone input to receive the audio. After stripping apart the USB microphone wires, we rerouted the input from the microphone's diaphragm to a hardware switch that chooses between receiving from a local microphone input

and receiving microphone input from a connected system. In addition, we re-routed the "talk" button from the original intercom to be controlled by our own push button. A secondary push button was also included to connect the microphone to the intercom, so that voice signals could be received by the intercom.

Finally, since VESPA is intended to be a convenient, embedded system without a screen, we included four LED lights to indicate the status of the system at any time. As the program first starts, all lights come on. After the program has loaded, the yellow LED indicates "Status Ready" to show that it is receiving voice input and will interpret any commands given to it.

After a person has issued a command, a green LED will indicate "Status Success" or a red LED will indicate "Status Failure" depending on whether or not the system was able to recognize the user's voice command, as well as to indicate if the user gave a valid command. Finally, a blue LED indicates "Status Deferred," which means that the requested device is located at a connected system in a different room.

Our entire setup is enclosed inside a shoebox, with the bottom side emulating a mock "wall" that the system would be installed in. The box conveniently hinges open to display the contents of the system, and it also provides easy transportation. We created two identical "shoebox" systems to prototype the full working system. The full circuit schematic is shown in ***Figure 1*** and the assembled system is shown in ***Figure 2***.

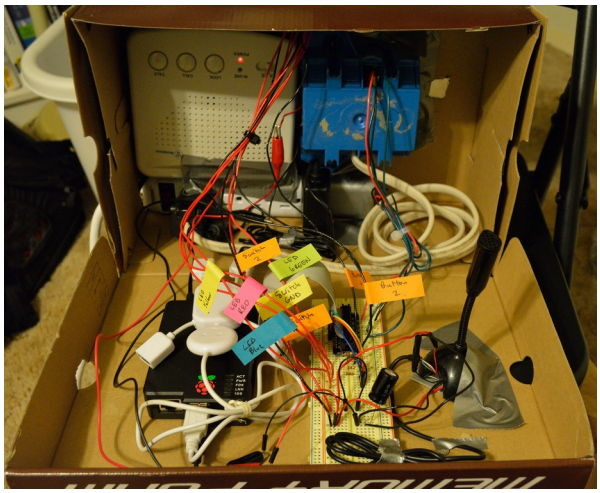

Figure 2: Interior View of VESPA

B. Software

The software included on our Raspberry Pis contains several essential components. These include Sphinx4, a Java-based speech recognition which uses Java Speech Application Program Interface (JSAPI), Pi4J, a Java-based library for controlling the GPIO pins of the Raspberry Pi, and finally, Management, our own Java program written to manage the system.

Sphinx4

CMU Sphinx is a group of speech recognition systems developed by Carnegie Mellon University. Our particular interest was in Sphinx4, which is the latest version of the Sphinx system designed to be a complete rewrite with the goal of being more flexible for research purposes [3]. Sphinx4 includes several high-level voice recognition interfaces, including LiveSpeechRecognizer (using a microphone as the speech source), StreamSpeechRecognizer (using an audio file as the speech source), and SpeechAligner (time-aligns text with speech) [3]. For the purposes of our system, we required the LiveSpeechRecognizer.

For each of the recognition interfaces, four particular attributes must be defined. The first is the acoustic model, which describes a number of variables such as particular voice differentiation (such as between you or your friend), the recording environment, the audio transmission channel, and accents. In our setup, we used the default acoustic model, which does not employ particular accents and assumes the user is using a typical USB microphone.

The second attribute is the Dictionary, which defines the particular set of sounds in which to build words from. The Sphinx4 setup comes with predefined dictionaries for particular languages, but advanced users may build their own. We used the predefined English dictionary.

The third attribute is the grammar model. Unlike the Dictionary, which helps build words from sounds, the grammar model determines which specific words are to be detected. Users are able to comprehensively create their own statistical language model, but for our purposes, we simply built a custom grammar written manually in Java Speech Grammar Format (JSGF).

The final attribute is the source of speech. We configured the Raspberry Pi to receive microphone input from the USB input, and we configured our code to use this microphone as the source for speech recognition.

Pi4J

The Pi4J library is a Java-based library which interfaces with the Raspberry Pi's GPIO pinout header. This library is a convenient tool that provides full access to the GPIO pins without needing to interface with either Python or Scratch from our Java-based system, which would have required greater computational overhead.

This library works by implementing a Java Native Interface (JNI) wrapper of the WiringPi library developed by Gordon Henderson [4]. A JNI allows Java running in a Java Virtual Machine (JVM) to call and be called by native applications in a system (in our case, the Raspberry Pi's pins) that may be written in other languages, such as C, C++, and assembly.

Since it statically compiles against the WiringPi library, which is a native library to access the GPIO pins, no extra dependencies were necessary aside from the Pi4J JAR files. However, due to extra files in the build path, this increases the startup time of the program.

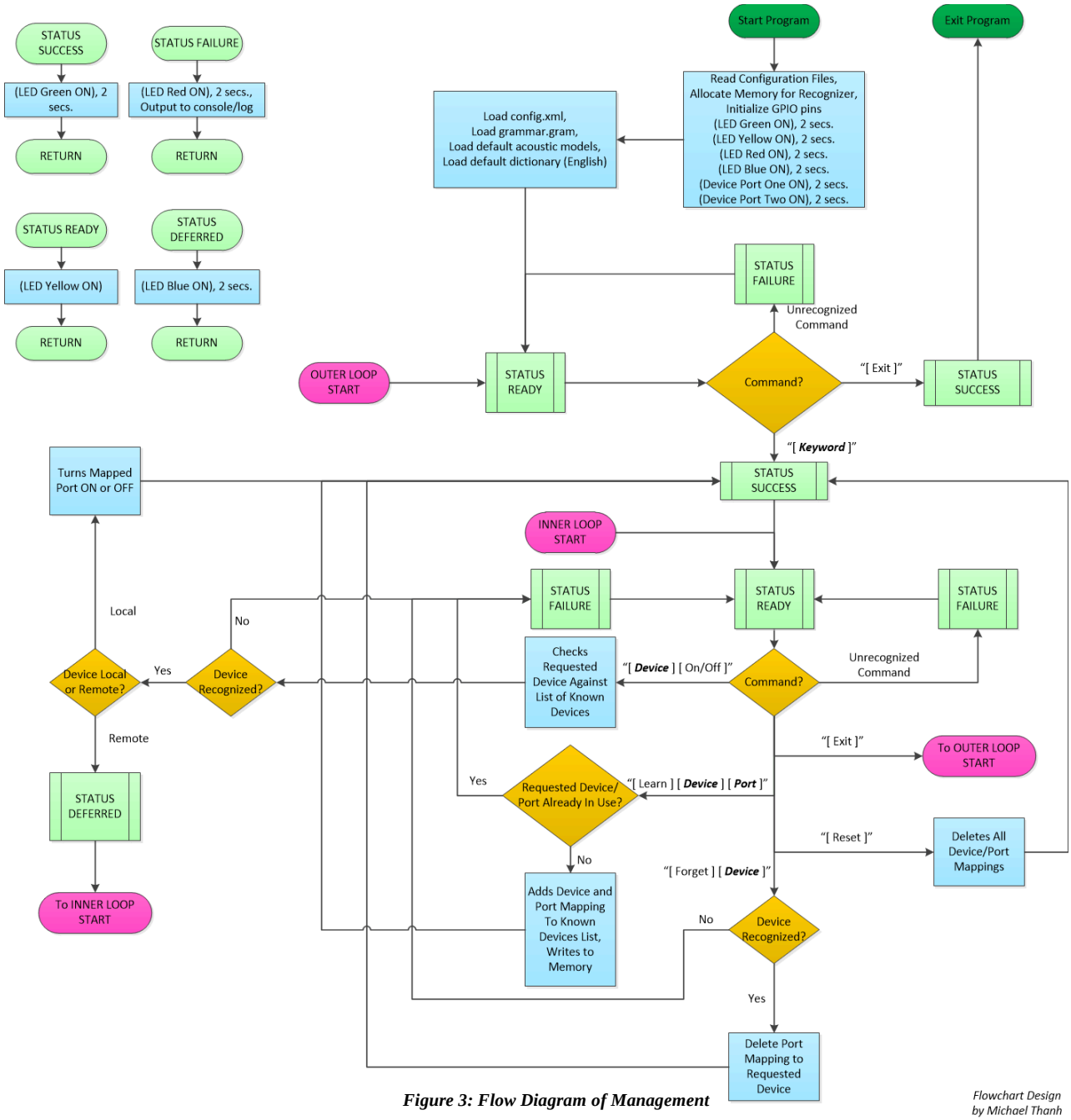

Figure 3: Flow Diagram of Management

Flowchart Design by Michael Thanh

Management

Management is the program that we designed ourselves which incorporates the above two Java code libraries. The structure of this program is relatively simple, and to incorporate all of the components, we needed to include the JAR files in the build path for Java.

The main program loads configuration files defining the acoustic model, dictionary, grammar, and audio source, and then proceeds to continue to a system level loop. In the outer loop, the system waits for input from a user corresponding to a "keyword" that the user defines. In this way, the system does not execute unwanted commands in random conversation.

Once the keyword has been issued, it enters an inner loop where it actively listens for specific commands to be issued. As of now, four commands are available to issue. The next sections detail these commands. The full flowchart of Management is shown in ***Figure 3***.

[Device] [On/Off]: This command calls a particular device to be turned on and off. Within Management, a class called Storage keeps track of device names as well as which GPIO pins they are mapped to. Whenever commands to a particular device are called, Storage checks the command device against what it contains in memory, and then toggles that specific pin. Whenever a requested device does not exist in the system, this turns on the "Status Failure" LED mentioned in the previous section. If the device does exist, but it exists on a system other than this current one, it turns on the

"Status Deferred" and "Status Success" LEDs. If a device is already on or off when an *On/Off* command is issued, no action is taken.

[Learn] [Device] [Port]: The *Learn* command maps a new device into the system. Each VESPA outlet is assigned a specific number for a port. For example, if a user wants to map a new device such as a lamp to port three, the command they issue will be "Learn lamp three." This adds the device to Storage, and Storage writes the list to memory. Thus, if the system is shut down or restarts for any reason, all learned devices will still be retained. If there is already currently a device mapped to a request port, the "Status Failure" LED will turn on. Otherwise, it will turn on the "Status Success" LED and the "Status Deferred" LED if the newly-mapped port exists on a connected system.

[Forget] [Object]: This command unmaps a particular device from a port and erases it from memory. Like the other commands, the respective LEDs will flash on for two seconds if the command is successful or unsuccessful.

[Reset]: This command is similar to the *Forget* command, but unmaps every device from every port and removes all system memory.

IV. Experimental Results and Challenges

During the testing of our system, we ran into several of problems. One of the most prominent problems we encountered was VESPA's inconsistent interpretation of commands. Since the intercoms we incorporated into our project were designed for simple communication and the speaker system was two-way through a single diaphragm, the quality of audio transmitted to other intercoms was not the best. Alongside reduced clarity, there was also a lot of noise, which prompted us to connect a capacitor to act as a low-pass filter into the system so that a clearer signal would reach the Pi.

Even when the received voice input was in the same room through the USB microphone, sometimes the speech recognition software would interpret the wrong command. Since we mapped a command to stop the system with *Exit*, sometimes the program would quit unexpectedly, which ended up wasting a good deal of time.

Alongside speech recognition problems, we also discovered a hardware problem. At the initial start of this project, we had been planning for VESPA to be entirely hands-free, but we discovered that this was not possible: when we routed the input from the microphone in series or in parallel with the input from the intercom, none of the signal was able to reach the Raspberry Pi. Thus, we had to create the previously mentioned switching mechanism that switches between receiving from the intercom and receiving from the microphone.

Despite these challenges, the system responded correctly to voice commands much of the time. Although our system did require a few manual switches and buttons, our system met our goal of not requiring a mobile device or network connection.

V. Conclusion and Further Developments

Overall, this project resulted in a working prototype of a novel power home power control system. Given clearly enunciated commands, the VESPA prototype will correctly turn devices on and off as well as map and unmap requested devices on any connected system, all by voice command. We have included a short video presentation demonstrating the system in action with a simple program (not Management, but one which simply toggles devices on or off) [5]. Management source code has been compressed into a zip archive and included with this report.

For a future version of this system, we would design the AC wall intercom circuit from scratch instead of using an existing system. This way, we could ensure that input from the wall intercom and the microphone would both be received by the Raspberry Pi USB input. Designing the intercom circuit would also allow us to build a better noise filter so that the voice commands could be interpreted more clearly.

Acknowledgment

We would like to thank Dr. Pasricha for supplying Raspberry Pi computers for use with this project, as well as Mr. and Mrs. Thanh for supplying the AC wall intercoms used in the project.

References

[1] Stevens, Carlile R., and Dale E. Reamer. Method and Apparatus for Activating Switches in Response to Different Acoustic Signals. Joseph Enterprises, assignee. Patent US5493618 A. 07 May 1993. Print.

[2] "Seven Reasons Why Z-Wave Belongs in Your Vacation Home." *Z-Wave: Home Control*. Sigma Designs, Inc, n.d. Web. 25 Nov. 2014.

[3] "Sphinx-4 Application Programmer's Guide." - *CMUSphinx Wiki*. Carnegie Mellon University, 04 Nov. 2014. Web. 01 Dec. 2014.

[4] "The Pi4J Project - Home." *The Pi4J Project*. Raspberry Pi Foundation, 29 Nov. 2014. Web. 08 Dec. 2014. <http://pi4j.com/index.html>.

[5] *VESPA*. Prod. Michael Thanh. By Kaden Strand. *YouTube*. Google, Inc., 8 Dec. 2014. Web. 9 Dec. 2014. <https://www.youtube.com/watch?v=5St5_xpeeZ4>.