Preparation of Senior Design Project Report

Harold Healy, EE, Ryan Walsh, CSE, Rahaun Perkins, CSE, and Nicholas Kafasis, CSE

Abstract—**The demand for autonomous systems is rapidly increasing all over the world. Both suppliers and consumers make every effort to use autonomous systems for their efficiency and ease of use. This technology will bring that efficient and easy automated technology to the music industry. Our system is an intermediary device, designed to automatically regulate the volume of an audio source depending on how much ambient noise there is in the surrounding environment.**

I. INTRODUCTION

Autonomous systems are being used all over the world to make tedious tasks disappear. From automatic thermostats regulating the heat of buildings everywhere, to the brightness on smartphones adjusting the screen brightness depending on the light in the surrounding environment. Another nuance that many people find themselves facing is volume control. Whether it be turning up the volume on a TV or turning up the music on a speaker, someone is always adjusting the volume of an audio source depending on the ambient noise in the surrounding environment. The goal of the dynamic volume controller (DVC) in this paper is to automate that adjustment for everyone.

The development of "smart" homes has been a popular area of research recently. There are wide range of ways to connect devices like phones, lights, and TVs in order to automate tasks. According to an IoT Innovation article, "Utilizing integrated technological systems in your home is one of the most significant new trends in digital innovation."[5] Our dynamic volume controller would add another element of automation to the smart home.

There have been similar volume controllers created, but they are either limited in use to very particular scenarios or very expensive. One example is the TOA Electronics Digital Ambient Noise Controller.[6] This model is on the market for well over a thousand dollars and is meant for large scale environments, such as malls and airports. This kind of solution is not suitable for a normal consumer looking for a less expensive product to use in a smaller scale environment. Another product that is currently available is the International Control Systems Automatic TV Controller.[7] This system works exclusively for when the volume on the TV suddenly becomes too loud. We want to make our DVC versatile and inexpensive so that it can be a suitable product for the normal household consumer wanting to further automate his/her house.

We decided on the following requirements to make sure our design provides the best experience for users: (1)Easy to use phone app will allow user to set fixed level of audibility above/below ambient noise, (2) System will not exceed max volume setting, (3) System will not react suddenly to isolated loud noises, (4) System will function in multiple locations within desired room. We chose an iOS app as a user interface in order provide something that consumers are familiar with. The max volume requirement is in place in order to prevent an unstable feedback system, driving the system to an unreasonably high volume and possibly causing hearing damage. The system must not react to isolated loud noises because this would cause rapid changes in volume that would be undesirable to the user. Finally, the system must be able to function within multiple locations of the desired room to allow a flexible and portable system that can be easily moved if the dynamic of the room changes.

The following specifications have been created to make sure that the requirements are satisfied.

Table 1:
Specifications and requirements

Requirements	Specification	Value
System will not exceed max volume setting	Maintain (Signal + Noise) to Signal Ratio within desired range	1.1-2 (1.5 for MDR)
System will not react suddenly to isolated loud noises	System will react to noise above desired ratio only after a certain time period	TBD (Instantaneous Reactions for MDR)
Easy to use phone app will allow user to set fixed level of audibility	System will work within a distance from audio source.	20 ft

Our device will require an initial calibration by the user in the room where the audio source is present. The system would then regulate the volume of that audio source by controlling what it sends to its speaker. The setup of the system will be explained in more detail in the next section of the report.

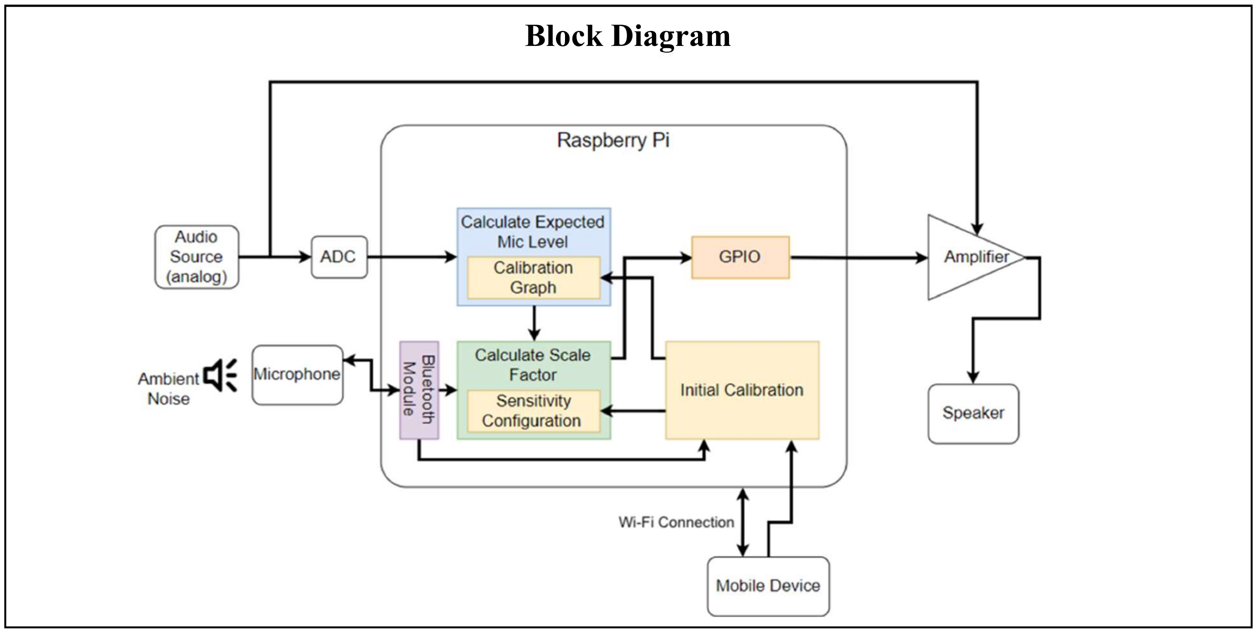


Figure 1: Block Diagram

II. Design

A. *Overview*

Our basic design will be a system that lies on the path between an audio output signal source, e.g. audio from a phone or a TV, and an output speaker to play the source. The system will use an external microphone to capture the source and environment sound levels. The input sound source signal as well as the captured microphone signal will be monitored continuously over time. If the system detects a significant level of ambient noise (i.e., a large ratio of microphone signal volume to expected microphone signal volume based on the input signal), then the system will increase the output volume that is sent to the speaker until a satisfactory level of environment noise to input source signal is regained.

Our design will center around a Raspberry Pi to regulate the entire system. It will be the central hub responsible for monitoring the microphone and original audio source signals, as well as making the necessary calculations on these signals to drive the necessary volume adjustments. The Raspberry Pi will ultimately output a value that will program a digitally programmable analog amplifier, and that amplifier is what will increase the volume of the output signal sent to the output speaker.

As we considered how to build the dynamic volume adjustment system, it was clear that we needed a central processing unit that would be able to handle the monitoring and calculations of several external signals. In addition to considering a Raspberry Pi, we first considered an Arduino to handle the job. As we looked into an Arduino Uno, we saw that it primarily operated with an ATmega328P microcontroller, an 8-bit microcontroller with a clock speed of 16MHz [1]. Additionally, the Arduino had only 32 KB of flash memory and 2 KB of SRAM memory for runtime data.

Lastly, the Arduino's I/O interface includes only 14 I/O pins [2]. We determined that for the live runtime calculation necessary for our system, we would definitely need a device with higher processing power than that of the ATmega328P, and much more memory than that of the Arduino. Additionally, the volume adjustment system design requires several I/O peripherals (microphone, audio source, output signal) to correctly run, and the limitation of 14 digital I/O pins would be a hindrance to try and work a solution around. That is why we turned to a much more computationally capable device—the Raspberry Pi 3 Model B+. The Raspberry Pi operates on a 1.4GHz 64-bit quad-core Arm processor. It also contains 1 GB of SDRAM memory [3]. These processing and memory specifications are much more suitable for the level of processing that we will require for live-time audio monitoring. Additionally, the Pi includes 4 USB 2.0 ports and a 4-pole stereo output port [3], which is suitable for the necessary external peripherals that we must connect to the DVC. A bonus to the Raspberry Pi is that it contains built-in Bluetooth and Wi-Fi capability [3], which allows us to expand out to wireless connections with ease; this is a positive, as we want to use wireless microphones and connect the system wirelessly to a smartphone application.

Our design will also utilize a digitally programmable analog amplifier to control the amplification of the output sound signal. We had originally planned to simply digitally scale the output signal in software running on the Raspberry Pi before the signal was output. However, as we began working on the DVC system, we discovered that digitally processing and outputting live audio through the Pi resulted in poor quality audio. This led us to rearrange the design so that the input signal is also routed straight to an analog amplifier, which will be programmed by I/O pins on the Pi itself, which we expect to result in much higher quality audio.

Our block diagram is shown above in Figure 1. As is shown, the Raspberry Pi is at the center of the design, responsible for taking in the audio source signal, the microphone signal, mobile device information, and running central computing modules to control the DVC system. The mobile device block refers to a smartphone application that will allow the user to connect to the system and control certain settings. The amplifier will be responsible for modifying the volume of the output signal.

B. Initial Calibration

The function of the initial calibration stage is to run a calibration process that will give the system a sense of the expected microphone pickup signal intensity (volume) given a certain input signal intensity (volume). This is necessary, as the DVC system is designed so that the connected microphone can be at a variable distance from the central system, and the system may be placed in a wide variety of environments, which will result in entirely different signal volume responses. The reason that we specifically want this expected microphone intensity vs. input intensity and relationship is that if we can determine the expected microphone pickup intensity over a chunk of time from an input signal, we can compare that expected intensity to the actual intensity that the microphone is observing. From that comparison, we can determine if the actual microphone intensity is significantly higher than the expected intensity (i.e., a presence of ambient noise), and if that is the case, we can increase the output signal volume to combat the ambient noise.

The user will be able to begin running the calibration process through a smartphone application, which is described in section C. Running the calibration process will play a constant tone signal from low to high volume through the output speaker. During this time, the system will record the relationship between microphone pickup intensity and input signal intensity. The intensities of each are calculated using the `rms` function available in the `audioop` library of python, which calculates the rms over a chunk of values. In the end of the calibration stage, the expected microphone intensity vs. input signal intensity function will be stored internally, so that it may be used later in the "Calculate Expected Mic Level" stage, which is defined in section D. After some testing, we discovered that this relationship forms a linear function ($y=mx+b$), as exhibited in an example below in Figure 2.

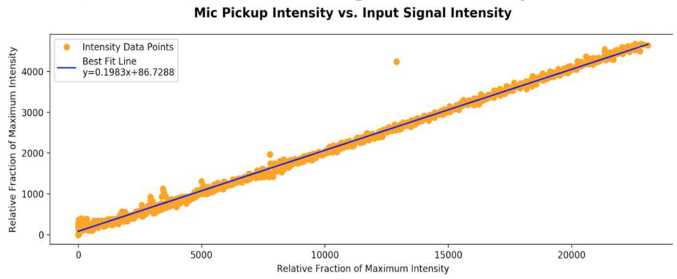


Figure 2. Microphone Pickup Intensity vs. Input signal Intensity Example

We can exploit this knowledge of a linear relationship in order to efficiently store the function and use it for later calculations. We can use a linear fit function on the data to determine the slope m and the y-intercept b, which are only 2 variables we need to keep to efficiently model the entire relationship. Then, given the intensity of an input signal, we may calculate:

$$\text{Expected Mic. Intensity} = m\bullet(\text{Input Signal Intensity}) + b \qquad (1)$$

The values of m and b are stored in variables within the main script at runtime, so that they may be used during the current run of the system. These variables are also stored externally in a JSON file. This is so that previous calibration settings may be loaded and used, so that the user does not have to re-calibrate the system each time the system is rebooted.

C. Smartphone Application

The app will be the UI that the end user interacts with to set their preferences for how the DVC system will operate. Within the app there will be three settings that the user will be able to change, the threshold, the sensitivity and also the max volume. In addition to these setting these preferences the app is where the user will start the initial calibration of DVC system. From the app these preferences will be sent to the Raspberry Pi where they will be used as variables in the main program.

After looking into what platform, the app would be developed on it was decided that the focus would be put into an iOS app. There were a few factors that went into the decision of choosing iOS vs android. One advantage of choosing iOS over android is the uniformity of iOS devices. When developing for iOS devices factors such as internal hardware and external specifications such as screen size are known and of a limited number as opposed to the hundreds of variations of physical specifications and software versions of android devices. For choice of language we will be using the Swift programming language because it was designed by Apple specifically for iOS development and will help eliminate any compatibility errors

In order to build this app, we will be drawing back on the strong programming foundation that we have built up through our years of computer science and networking courses. The development of the app will use the knowledge that we gained in programming courses such as CMPSCI 121, ECE 242 and ECE 373. We will also be utilizing information that we learned in courses such as ECE 374 to ensure that our app is interfacing with the Raspberry pi correctly.

Testing the app will be a two-part process consisting of a qualitative and quantitative test. The app will serve as the UI for the DVC system, so it needs to be clear and simple enough for an end user not be familiar with audio terminology to use. We will test the user friendliness of the app by demoing it to small focus groups and collecting feedback. The quantitative test will be much easier because, we will be able to see if the values being set in the app are being transmitted the software correctly. Once we verify that the app is interfacing with the Raspberry Pi and that all variables are being set correctly we can move to manipulating those variables in later code segments such as the calculate expected mic level and Calculate scale factor which are described below.

D. Calculate Expected Mic Level

The calculate expected mic level block will be calculating the expected microphone intensity given the input signal intensity. This calculation will happen continuously over time as the system is running as a means to live-monitor the sound level of the surrounding environment. This block is implemented entirely within software on the Raspberry Pi.

As mentioned previously in section B., the expected microphone intensity vs. input signal intensity function is effectively stored and modeled in software by the two values *m* and *b* that describe the linear relationship. As chunks of an input signal come in, the intensity over that chunk is calculated using the rms function. Next, the expected microphone intensity is calculated by using the input intensity as input to Eq. (1). This expected microphone level can then be used in further calculations to determine if any volume adjustments should be made.

E. Calculate Scale Factor

The next step in the core program of the DVC system is calculating the scale factor. The scale factor is the amount the input audio signal will be multiplied by in order to reach the desired output set by the user. In order to calculate this value the program starts with default scale factor of 1, meaning the input signal is not modified. The system then calculates a ratio of the average mic power divided by the average expected mic power, this ratio is then compared to the threshold that is set by the user in the initial setup and if the ratio is larger than the threshold the scale factor is increased slightly until it is no longer larger than the threshold. To avoid the system becoming unstable the default case for the scale factor behavior is too decrease, so that if there is no ambient noise in the room the volume of the audio signal will be as quiet as possible while still maintaining a level set by the user. Once the scale factor is calculated it will be sent over to the GPIO pins and used to set the gain of the analog amplifier that will be explained in the following section.

F. GPIO & Amplifier

This block will make adjustments to the signal sent to the speaker. The GPIO pins from the Pi will set the gain of the analog amplifier based on the scale factor calculated in the previous module. We will use the built in GPIO pins on the Raspberry Pi to send a string of bits to an amplifier with a digital potentiometer. We will use variable gain techniques learned in Electronics II to determine the resistance needed from the potentiometer to achieve a specific gain. We will also need to become familiar with how to control digitally programmable potentiometers. This will depend on the specific model we choose. For example, models may have different communication set-ups, such as, serial vs parallel bit transmission. One commercial model that seems to meet our specifications uses two wire I2C communication protocol (series bit transmission).[4]

In order to test this block, we will first create the circuit on a breadboard and manually change resistance of the potentiometer to make sure we can achieve the desired gain range for the amplifier. Once we have confirmed the range of resistance needed from the potentiometer, we will order the model that best suits this specification. Finally, we will test the circuit with the potentiometer and confirm that our speaker receives the correct range of amplitudes from this block before sending out our PCB design to the manufacturer.

III. Project Management

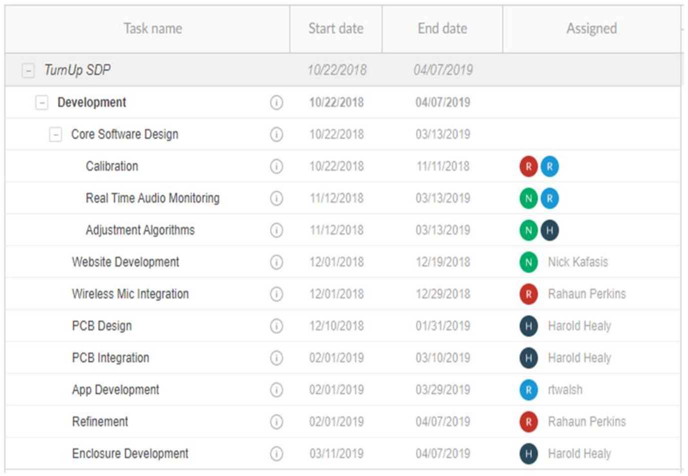
Table 2:
Deliverables

MDR Deliverables	**Status**
Show proof of captured signals and computed calibration graph	Complete
System will adjust based on threshold and sensitivity at set value.	Complete
Wired connections (no bluetooth)	Complete
Single Speaker System	Complete

Our group accomplished all MDR goals. Our system achieves the base case of increasing volume when there is more than 1.5 times the expected microphone intensity. Once the volume has increased enough so that the measured mic intensity is less than 1.5 times the expected mic intensity, the volume stops increasing and tries to decrease. With this base case working, it proves that is possible to achieve a fully functioning system by demo day.

Next semester, most of the work remaining has to do with user settings and integration of the dynamic variables sensitivity and threshold. Both of these variables are currently set on the Pi. In order to choose appropriate ranges for these variables, our group plans on testing different values by surveying a sample of friends. These surveys will be based on overall product satisfaction with regard to response time(Sensitivity) and ambient noise to speaker volume ratio(Threshold). Once the ranges for these parameters have been decided, we will develop the iOS app that allows the user to set these variables.

Our team consists of three Computer Systems Engineering students and one Electrical Engineering student. This means that our group has a strong software background. The different roles of our project have been split up according to the abilities of each member. The chart below shows the responsibilities of each team member.



Task name	Start date	End date	Assigned
TurnUp SDP	10/22/2018	04/07/2019	
Development	10/22/2018	04/07/2019	
Core Software Design	10/22/2018	03/13/2019	
Calibration	10/22/2018	11/11/2018	R R
Real Time Audio Monitoring	11/12/2018	03/13/2019	N R
Adjustment Algorithms	11/12/2018	03/13/2019	N H
Website Development	12/01/2018	12/19/2018	N Nick Kafasis
Wireless Mic Integration	12/01/2018	12/29/2018	R Rahaun Perkins
PCB Design	12/10/2018	01/31/2019	H Harold Healy
PCB Integration	02/01/2019	03/10/2019	H Harold Healy
App Development	02/01/2019	03/29/2019	R rtwalsh
Refinement	02/01/2019	04/07/2019	R Rahaun Perkins
Enclosure Development	03/11/2019	04/07/2019	H Harold Healy

Figure 3: Tasks

The core software design has been split up between all 4 group members. This is the largest, and most time consuming part of the project. PCB design will be the responsibility of Harry, as he is the electrical engineer on the team. Ryan has some experience in app development and therefore will handle the iOS app for our project. The other software related parts of the project were split up evenly between Nicholas and Rahaun, as they are computer systems engineers.

In order to keep a workflow and schedule, our team has weekly meetings amongst ourselves and with our advisor. The communication between our group consists of e-mail, text, and in person meetings. So far, we have taken a very unified approach towards completing our project. This means that we have been working on our portions of the project together rather than combining individual work every week. This has worked well so far, but we will likely need to have a more individual approach next semester once our system becomes more dynamic.

The Gantt Chart below shows what we have done so far and where we expect to be throughout next semester.

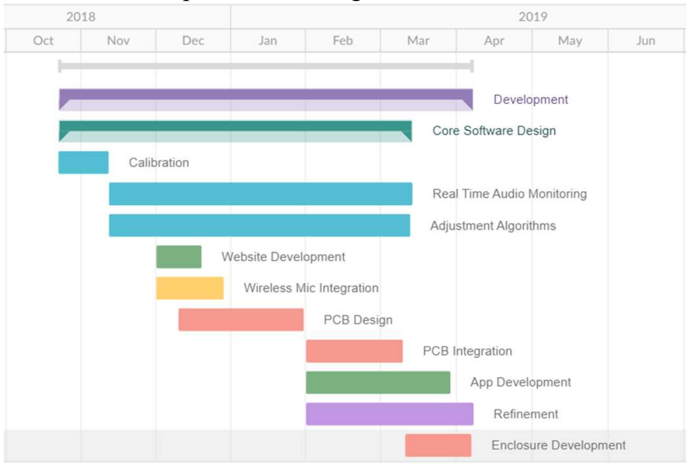


Figure 4: Gantt Chart

IV. Conclusion

So far we have met all of our proposed deliverables and deadlines for MDR including manually set variables and wired connections as well as being able to show proof of captured signals, computed calibration graph, and additional graphs such as how the scale factor changes over time. By CDR we will have a working app so variables will no longer be set in code a wireless mic will also replace the current wired version and finally our amplification will move to a an analog devices instead of being handled internally by the Raspberry Pi. One of the main challenges moving forward is making sure that our data transmission rates maintain high enough speeds to do our signal processing in near realtime after moving to wireless components. Another challenge we will face is designing a UI that can be easily understood by the a user without any knowledge of the system. Seeing as how the last problem is not quantitative with a concrete solution, it will be a unique challenge that we will have to approach differently from any traditional problem we have encountered as engineering students. At this point we need to implement a few more components such as the analog amplifier and the final system enclosure and then the project comes down to a large amount of testing and fine tuning in order to create a system that is enjoyable and convenient to use.

References

[1] Ww1.microchip.com. (2018). ATmega48A/PA/88A/PA/168A/PA/328/P Data Sheet. [online] Available at: http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf [Accessed 20 Dec. 2018].

[2] Store.arduino.cc. (2018). Arduino Uno Rev3. [online] Available at: https://store.arduino.cc/usa/arduino-uno-rev3 [Accessed 20 Dec. 2018].

[3] Raspberry Pi. (2018). Raspberry Pi 3 Model B+ - Raspberry Pi. [online] Available at: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/ [Accessed 20 Dec. 2018].

[4] C. Wells, J. Becker, *Low-Cost Digital Programmable Gain Amplifier Reference Design*. Texas Instruments, 2015.

[5] IoT Innovation. (2018). *The Impact of Smart Homes Technology | IoT Innovation*. [online] Available at: https://internet-of-things-innovation.com/insights/the-blog/smart-homes-technology-impact/#.XBw691VKjIU [Accessed 21 Dec. 2018].

[6] Toaelectronics.com. (2018). *Products - TOA Electronics*. [online] Available at: http://www.toaelectronics.com/products/audio-signal-processors/dp-l2-digital-ambient-noise-controller [Accessed 21 Dec. 2018].

[7] Amazon.com. (2018). [online] Available at: https://www.amazon.com/International-Controls-Systems-TVSR-Automatic/dp/B000Q37TBY/ref=cm_cr_arp_d_product_top?ie=UTF8 [Accessed 21 Dec. 2018].