Coresidium: a Gunshot Detection System

Brandon Cross, CSE, Valentin Degtyarev, CSE, Andrew Eshak, CSE+EE, and Andrew LaMarche, CSE

***Abstract*—Coresidium is a project that aims to detect gunshots in indoor locations in an effort to reduce school shooting response times. It uses an acoustic sensor, a thermal camera an an embedded processor to detect the gunshots locally then sends the data across the Internet for further processing. The server determines whether the event is an actual incident, stores its data and displays it to the user through a web-based dashboard. In our current attempts, the embedded module recognizes an event, stores the timestamp and send it to the server. The server determines whether the received data is an incident or a keepalive data and displays it according to the dashboard, available on dashboard.coresidium.com**

I. Introduction

PERHAPS one of the most troubling epidemics of our generation is the continuing rise of school shootings around the United States and mass shootings in general. Between January 1st, 2009 to May 21st, 2018, 288 school shooting incidents have occurred in the United States [3]. This figure is in fact 57 times that of the total number of school shootings of the six other G7 countries (United Kingdom, Japan, Italy, Germany, France and Canada) [2].

In response to this problem, Congress has remained deadlocked on a proper solution to address the number of fatalities and to stem the tide of violence occurring weekly in schools around the nation. As a result, many schools have implemented novel techniques and protocols to address this problem. For example, some schools have begun to teach their students basic defense mechanisms against intruders during 'gym' class. Other schools have introduced drills to allow students to exit the building as swiftly as possible in order to reduce the number of fatalities. Some parents have also forced their children to carry bulletproof backpacks in order to reduce the chances of their children being fatally shot [5]. Finally, on the recommendations of President Donald Trump, some schools have offered firearm training to their teachers and offered incentives to teachers who carry a firearm during the school day.

While these techniques may offer a temporary patch to the problem, they do not offer a complete solution to the problem and leave the students feeling vulnerable and unsafe during their school day. It should also be noted that since 2000, 186,000 students have been affected by school shootings and more than 1,100 students have been killed or injured [3].

This problem is made worse by the fact that when the authorities are alerted to an incident within their district, the response time is often much too great, delaying the time it takes for the victims to receive the proper attention and medical care they need. This long period is due to the fact that authorities are often unaware of the location of the shooter within the building and as a result, have to methodically search through every room in the building in order to properly declare the campus safe from any further threats. Table 1 demonstrates the duration of the period from which the shooter enters the building until the building is declared safe, during 3 separate shooting incidents. It should be noted that although the incidents happen in completely different decades, and although the response time decreases as we progress through the incidents, the duration is still much too long for those within the building.

TABLE I
SCHOOL SHOOTINGS IN THE UNITED STATES [4][9]

Incident	Date (mm/dd/yyyy)	State	Response time (minutes)	# of fatalities
Columbine	04/20/1999	Colorado	328	15
Virginia Tech	04/16/2007	Virginia	217	33
Douglas Stoneman High school	02/14/2019	Florida	198	14

This delay in the response time can increase the number of fatalities and injuries within the building as well as increase the duration of the period in which students are traumatized as their worst fears become a reality. Furthermore, this increased delay can leave anxious parents unaware of the status of their children, further increasing their unease.

Congress and schools are not the only parties that attempted to tackle such a problem. Companies have attempted to come up with novel solutions to curb this recent tide. Some of these attempts include bulletproof windows and doors as well as surveillance systems to detect intruders. Such systems are the focus of our project, as they can be installed at huge costs to the school districts, making them unaffordable to most schools [7].

It is difficult to come by exact figures for the cost of attaching such systems, since they often rely on multiple variables such as the area of the schools and the number of students within the school, as well as the reluctance of such companies to release their prices publicly. However, while talking with officer Kellogg of the University of Massachusetts Police Department, it has been mentioned that prices of such systems are extremely unaffordable for a public University with the size of the University of Massachusetts-Amherst. Through online research, we located just one school that installed such a system in North Carolina. Shooter Detection Services, the company manufacturing such a system, is Massachusetts-based and installed the system within the three buildings of the 1000 student school, to the estimated cost of $400,000[1].

Similar systems have also been deployed by state governments in order to locate gunshots on the streets of their cities. In such systems, microphones are placed on the streets and are used to triangulate the location of a gunshot when it is fired, using the speed of sound. The difficulty with such systems is that they are utilized in outdoor locations, ignoring the different parameters of indoor use such as echoes and the rate of sound travel within different materials. Most also rely on human input to be able to discern whether the alarm was based on a gunshot before notifying the authorities. This criterion was added since most systems were unable to discern between a gunshot and firecrackers. Finally, such systems are also hugely expensive, making them inapplicable for school use [7].

In response to such difficulties, we aim to design a system that would notify authorities in the event of a gunshot and provide a relative location of the shooter. We define relative location as the floor of the building as well as the direction within the building (east, west, north or south). This notification and location system will be provided to the authorities through a web-based dashboard. This system would rely on acoustic sensors as well as visual sensors to detect the gunshots and utilize embedded systems and a central server to perform the computation.

TABLE 2
SPECIFICATIONS

Specification	Value
Range	10 feet per module
Response time	<1 second
Accuracy	>80%
Cost	<$100 per module
Sensitivity range	>130 dB
Timestamp accuracy	<1 second
Location accuracy	Floor and direction within building

II. Design

A. Overview

Our solution to this problem entails a two-tiered system consisting of an embedded module to perform the sensing and signal filtering, and a central computing node to coordinate between the multiple nodes, analyze the data and output the data to the user. The computing node is implemented via an Amazon Web Services instance running windows server, ensuring high performance and availability to all the embedded modules. The embedded system communicates to the computing node via HTTP requests. The computing node cannot communicate back to the embedded modules, however.

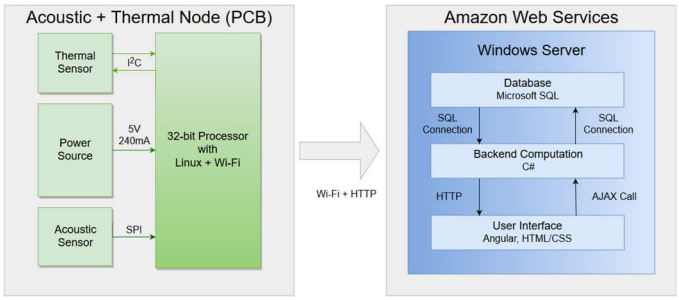


Fig. 1. The Block diagram of the coresidium system. On the left is the embedded module while on the right is the computing node that utilizes an Amazon Web Services Instance.

More specifically, the embedded module consists of an acoustic sensor to detect the sound signature of a gunshot as well as a thermal sensor to detect the muzzle flash or the heat signature of the gun. Both sensors are connected to a 32-bit microprocessor, running Linux, that performs timestamping and basic signal filtering before sending the data to the computing node over Wi-Fi. The entire module is powered by a power supply supplying five volts and a minimum of 240 milli-amps.

On the other side, the computing node is made up of an Amazon Web Services Instance running a Windows Server. It is powered by an Intel Xenon processor with 16 Gb of RAM. Within the server, a three-tiered system to perform the necessary computation. At the bottom of the system is a SQL database that stores the raw data from the server. In the middle tier, the backend controller utilizes C# to coordinate and receive data from the embedded modules. It also performs computation on the data received to check its validity, match it to a location and store the data in the database. Finally, it communicates to the dashboard in order to allow the user to view the current status of the modules and the incidents that have occurred. It is important to note that the middle tier acts as the only gateway that receives data from the embedded modules. This means that data coming from the modules cannot be stored directly in the database or be displayed directly to the user interface, providing an extra layer of computation to scrutinize the data. Furthermore, the SQL database cannot communicate to the user interface or vice versa without the use of the middle tier which performs the necessary calls and data translations for both tiers. The final tier consists of the user interface, displayed over the

internet. It utilizes HTML and CSS as well as Angular to dynamically display the data in a user-friendly manner, without the need for a data expert to perform analysis.

It is important to note at this point that other proposals have been explored in order to address this problem. In our initial proposal, the thermal sensor was replaced a traditional camera and machine vision to recognize the images of a gun. This required the use of neural nodes, high performance machines with dedicated graphics cards and thousands of training sets in order to get the needed accuracy from the network. We experimented with a neural network algorithm provided over the web, collected over 1500 images of a variety of guns as our data set and used one high performance computing machine in order to perform the necessary training. However, such endeavors were not fruitful in the sense that our accuracy was below 30% using this method. This can be attributed to the need for more training sets. It was also pointed out that our data set utilized images of guns at "convenient" angles where the gun was directly facing the camera. This contrasts with real-life situations where guns are often hidden or at awkward angles to perform accurate detection. It was also pointed out that a great amount of computational performance would need to be available in order to perform the detection in real-time. This need for performance would come at a high monetary cost, making this seem unviable for most schools and offsetting its benefits compared to the available commercial systems.

B. Acoustic Sensor

Focusing on the embedded module, our first step is to explore the acoustic sensor. In our design, we used a KY-038 sound sensor module that detects large sounds, amplifies its signal, compares it to a predetermined value and outputs it through pin one. The predetermined value to be compared to is configured through a potentiometer and varies from module to module. We relied on our experience from Computer Systems labs to receive the data from the sensor at scheduled intervals using a digital to analog converter. One piece of information that is lacking is from our knowledge is what the voltage outputs of this module represent. This means attempting to find a relation between the output voltage and the physical noise being received. We also need to learn how to perform noise reconstruction so that we can construct an audio file

In this sensor, the maximum detectable sound amplitude is 130 dB whereas gunshots range from 150 dB to 190 dB. As a result, in most cases we configure our sensor to detect the largest possible amplitude, while keeping in mind that sounds between 130 dB and 150 dB may cause a false positive in our results. It is also important to note that loud conversations have a maximum amplitude of 90 dB, a loud balloon popping has a maximum amplitude of approximately 110 dB, and an emergeny siren has a maximum amplitude of approximately 125 dB. As a result, we believe that this sound sensor will be sufficient for our purposes.

During our initial proposal, we considered using higher end microphones that can operate at higher amplitudes. For example, aersospace grade microphones, such as the HOLMCo 82-03-08274 can pick up sound in our desired range. However, the biggest disadvantage with such microphones is their price, which can range from a minimum of $150 to $500 per microphone. This would make the cost of our system highly impractical and infeasible for most schools, which is an undesired outcome that we attempted to avoid.

In order to test this module, we reduced the sensor's sensitivity, popped 25 balloons within 12 feet of 4 modules and recorded whether each individual module picked up the loud noise. We recorded the number of successfully reported incidents and divided it by the total number of attempts to receive our success percentage. The result of such a testing procedure was around 80% success with no false positives and 20% false negatives. This leads us to believe that our system was conservative in detecting such incidents. This problem can also be mitigated by the addition a supplementary dedicated microprocessor to perform the sampling on its own.

C. Thermal Camera

The thermal camera was responsible for detecting the heat signatures of a gun that has been fired. For our purposes, we chose the MLX90640 thermal sensor, that is capable of detecting temperatures ranging from -40°C to 300°C. It has a 32x24 resolution and a 110°x75° field of view. The temperature range is excellent for our purposes since gun barrels have a minimum temperature of 150°C, making it easy to detect and distinguish it from the traditional school environment.

While the field of view is excellent for our purposes, allowing us to look farther and utilize fewer modules per school, the resolution has been the most disappointing part of our attempts, not allowing us to get clearer picture of the events. The camera is also capable of recording up to 32 frames per second but transmitting at such a rate has often been difficult due to performance bottlenecks on the VoCore.

For this component, we utilized our previous knowledge of Computer Systems Labs to build this embedded module, to sample the data and to perform the necessary computation on it. However, this module has been disappointing due to its low resolution, making it difficult to distinguish objects or to view objects at a longer distance than three feet.

In order to test this module, we flickered a lighter from 3 feet away 100 times and recorded the number of successes over the total number of flickers. This resulted in a 68% success rate, a result that is both disappointing and one that we believe we may not be able to improve due to the low resolution of the camera.

Currently, we are exploring whether we should continue using this component or simply abandon it to focus on the acoustic module. We are also exploring other alternatives such as using a traditional camera to start recording the incident as soon as an event is detected by the acoustic sensor. This video recording can be used for further analysis or for law enforcement to use as security footage.

D. 32-bit processor

This component is the main processing unit in the embedded module. It performs event checking, timestamping and basic signal processing from the sensors before sending the data to the server-end of the module. In our current implementation, we use a Raspberry Pie version 3 B but we are striving to replace

this component with a VoCore embedded chip. This chip has on-board wifi as well as Linux WRT, a special version of the Linux operating system designed specifically for embedded systems. This system has two threads, one for the thermal camera and one for the acoustic sensor. Both threads run in an infinite loop and never exit until an event is detected, then the main threat timestamps the event, forms a struct with the event's data and sends it over to the server wirelessly via HTTP POST requests.

Testing this component has consisted of manually pushing write requests upon boot to the server. If the server receives the test data, then the embedded system is connected to the internet and able to push requests to the server. Secondly, we tested the threats from both the acoustic sensor and the thermal sensor by lowering the sensitivities on both and triggering the sensors using a clap of the hands or the heat of a hand. One can check if both threads work properly by checking if the server receives data for both incident reports. If the server does not receive the data, one knows that the problem is with the individual sensor computation since the code responsible for sending data over the internet has already been verified

All of the code for this module was done using the C language and relied on our previous knowledge of embedded system computing that we learned in Computer Systems labs. The next step for this module is implementing the keepalive so that the user is aware of the non-functional devices. We also hope to sample at a minimum of 40 kHz so that we can record a 30 second audio recording of the incident occurring for further human analysis. This would require the addition of dedicated memory components since the on-board memory is incapable of handling such data. Furthermore, it would require the optimization of our current code since currently, we can only sample at up to 21 kHz, meaning that we can only pick up on noises up to 10.5 kHz, much less than the maximum 20 kHz audible frequency.

It is also important to note that other options have been considered for this module. Our primary alternative was using an embedded ATmega 32-bit processor to perform the computation. This would have been much cheaper considering that the VoCore costs approximately $18 while an Atmega 32-bit processor would cost below $2. However, this was not pursued due to the superior performance of the VoCore, running at above 500 MHz compared to the Atmega's 48 MHz . This was also chosen since it is a complete module with low power use, onboard memory, a dedicated operating system, a built-in WiFi chip that does not need configuration and superior performance.

E. SQL database

The SQL database stores data regarding our system, to be used later on by the other components of the module. It only communicates to the C# based middle tier. In our current implementation, we use three tables. The first table is a user accounts table that stores authorized users' names, emails and passwords in a hashed format. The incident table is the second table and it stores data from the embedded module about whether a significant event occurred or not. More specifically, it stores the communicating device's MAC address, whether a microphone or a camera caused the alarm, whether this message is a keep alive signal or an alarm for an incident that occurred, as well as the sensor data sent by the module. The third table stores the device-to-location mappings. This means that the table stores the MAC address of each device used in our module as well as the location of each module, which we stored as a string. This enabled the middle tier to display the incident report by checking the incident table then matching each incident to a location in the location mappings table using a "left join" query statement. We relied on our previous knowledge of data structures and relations to form this relational database. However, one needs to learn how to store sound data and store it into a file for our project.

It is important to note at this point that other options have been considered, such as storing the data in a JSON file or in a text file. This approach however, seemed the most logical and the easiest to work with due to the compact and concise data storage available with SQL.

In order to test this module, we simply ran a number of insertion, join and deletion queries to make sure that the database accepts reasonable data and a reasonable number of requests. When the middle tier was developed, we manually triggered a number of incidents to view if they will be stored properly in the database. The results of such experiments were surprisingly pleasing, with no major issues due to the simplicity of this component. This leads us to believe that this component is ready for use in our system.

F. Backend computation

The middle tier of the server end of the module is based on the C# language. Its main purposes include receiving data from the embedded module and storing it in the SQL database, processing the requests made by the dashboard, and taking data from the SQL database for the frontend. It utilizes techniques from Software Engineering and relies on the model-view-controller concept.

More specifically, this tier relies on two controllers. The first controller is the read controller that reads data from the SQL database, does some basic processing on such as discarding of erroneous ones, and relays such information to the frontend. The second controller is the write controller, used by both the frontend and the embedded modules to store data into the SQL database. Its basic processing requests include authenticating users attempting to log in to the dashboard, distinguishing between events and keepalive records in the database and sending them to the appropriate page, and checking whether the keepalive messages are received from all the nodes to ensure that they are all functioning properly.

In order to check this tier, we manually pushed some write requests from the embedded modules, checking whether all the types of requests to be made are accepted without causing errors and that the data is stored properly in the database. We also pushed some manual HTTP requests simulating the frontend to see if the requested data is returned to the developer console in Google chrome, to check whether frontend information is stored properly in the database and whether users are authenticated properly. In our experiments, as soon as one request was processed properly, all the requests were processed due to the simple nature of this module.

In the future, this module will be used for other experiments such as triangulation, Fourier transforms for sound matching and reconstruction of the sound files based on the data received by the embedded module. All such experiments require a relatively high level of sophistication, something that isn't available on the VoCore. As a result, it will be implemented in this tier in the hopes of increasing the accuracy of the gunshot detection as well as the usability of the system to its users.

Testing this module has consisted of manually forcing some HTTP requests upon boot to ensure that the system can receive data, store it in the database or load it to the web-based graphical user interface.

G. *User interface*

The final component of this module is the user interface. This interface is web-based and is built based on HTML/CSS and Angular. It currently sends requests via HTTP to the C# based middle tier, that queries the database and sends the data back to be displayed to the user. This dashboard is available at dashboard.coresidium.com and currently displays the data in a table format based on the dates of the incidents. It also has a separate table for keepalive devices, demonstrating devices that do not work properly. As a result, the frontend is configured to work with keepalive messages, awaiting the message from the embedded module.

This module relies on information we learned in the software engineering course. One limitation however of the current implementation is that it does not receive the data in live time. This means that if an incident happens while the user is logged in, the incident is not displayed to the user until the time he logs out and logs in once more. Furthermore, we plan to improve this module by introducing a map of the room in which the modules are placed as well as the relative location of the event that occurred. We also plan to improve the user interface by removing any bugs and making it more intuitive.

III. Project Management

TABLE 3
MID-WAY DESIGN REVIEW GOALS

MDR Deliverables
Acoustic module identifies simulated gunshot with 65% accuracy
Thermal camera recognizes objects above 120°C for 0.5 seconds
Store data in SQL, compute location and coordinate between modules
Simple online dashboard with relative location of threat (floor & side)
Find acceptable insulation for microphone

Our first MDR goal required the acoustic sensor to work properly and detect simulated gunshots with 65% accuracy. In our tests, we were able detect a simulated gunshot (using a balloon popping at approximately 110 dB), at 12 feet away from the sensor with 80% accuracy. As a result, we exceeded our MDR goals. We also demanded that objects with temperatures above 120°C for 0.5 seconds be recognized by the thermal camera. This was simulated using the flicker of a lighter, 3 feet away from the sensor, and in our tests, we were able to achieve such a goal with 68% accuracy. It should however be pointed out, that due to the low resolution of the sensor, it is unable to detect hot objects at distances greater than 3 feet, putting into question its practicality for the goal of this project. We also required that we explore the proper insulation for the microphone. This was needed since the microphone could only recognize sounds up to 130 dB whereas gunshots range from 150 dB to 190 dB. As a result, we believed we could utilize this to reduce the amplitude of the gunshot, making it fall in the acceptable range while ignoring lower amplitude noises. This in effect would reduce our false positives and conversely increase our false negatives since it would reduce the amplitude of regular noise, making it undetectable to the microphone, while increasing the chances that a gunshot from a distance is not detected by the microphone. In our tests however, it was discovered that we did not need such insulation since if a signal above the 130dB threshold occurs, it will simply saturate the microphone and not have any detrimental effects on its components. Also, during our tests, we received no false positives and only 20% false negatives, making the use of insulation unnecessary.

For the computing node, our goal was to set up a SQL database to store the sensor data and to perform the basic computation to determine the location of the gunshot. This was achieved since our SQL database had 3 tables: User Accounts table, sensor data table, and location mapping table. The sensor data table stored the timestamp of the communication, the MAC ID of the sender, whether it was a keepalive signal or not and the type of sensor sending data. The location mapping performed the mapping between the MAC ID and the location of the device. We also utilized the middle tier to receive and coordinate the data coming from the sensors and to manage the database data. It also responded to the requests from the user interface to provide the location and statistics about the incidents that have occurred. The API for the middle tier can be found at http://backend.coresidium.com. The user interface has also been accomplished, providing the location and timestamp of the incidents in a table format to the authorized users. It can be found at dashboard.coresidium.com

Our next steps would be to improve the acoustic sensor accuracy to greater than 85%. We are also considering abandoning the thermal sensor in order to focus solely on improving the accuracy of the acoustic sensor or to perform more specific experiments such as sound signal signature matching using Fourier transforms. This type of Fourier transforms would be done on the server side. We are also exploring sampling and recording the acoustic data in an effort to create an audio recording of the incident. All such endeavors are experimental however and we cannot guarantee their completeness in May. We are also aiming to improve the user interface of the dashboard. More specifically, we are hoping to

remove the data table format from the interface and adding a more intuitive map to describe the location and timing of the incidents. Finally, we are hoping to implement “keep alive” functions on the embedded systems and displaying these device reports online so that users can view broken modules and aim to replace them.

In terms of division of responsibilities, our team divided the work equally according to each individual’s strength. Brandon Cross was responsible for managing the thermal camera, along with its code and its tests. Valentin Degtyarev was responsible for setting up the acoustic sensor module and making sure it communicates properly to the VoCore. Andrew LaMarche was responsible for the general setup of the VoCore, taking data from both the thermal camera thread and the acoustic sensor thread, and sending it over to the server side of the computation. Finally, Andrew Eshak was responsible for server-side computation, setting up the SQL database, the C# API and the Angular/HTML based frontend.

We also utilized a “buddy system” in which two people studied the same piece of code in order to offer multiple perspectives on a solution and to ensure a “backup” person in case of emergencies. Using such a system, Brandon Cross worked with Andrew LaMarche towards VoCore communications, and Andrew LaMarche worked with Valentin Degtyarev towards the acoustic sensor. Andrew Eshak worked with Brandon Cross to verify the thermal sensor and Valentin Degtyarev worked with Andrew Eshak towards ensuring proper use of the server code.

To manage communications, our team utilizes Discord and iMessage group chats in order to discuss future endeavors for the project, the status of current attempts and meeting times. In addition, biweekly meetings were set up in accordance with every person’s schedule in order to ensure that all the individual pieces of the project worked properly with each other. Finally, a weekly meeting occurred with Professor Siqueira, often on Mondays at 1:30 P.M., in order to provide a status report and to receive advise on our next steps. All such forms of communications helped ensure that the project remained on track and that all our individual modules culminated into one complete system.

IV. Conclusion

In conclusion, during the first semester of the 2018-2019 year, our team has built the embedded module utilizing both an acoustic microphone and a thermal camera as well as a Raspberry Pie Version 3. The microphone detects acoustic anomalies and the thermal sensor detects high temperatures that last for a short duration such as a muzzle flash. The Raspberry Pie was responsible for taking sensor data and detecting an event, performing the timestamp and communicating the data to the server module.

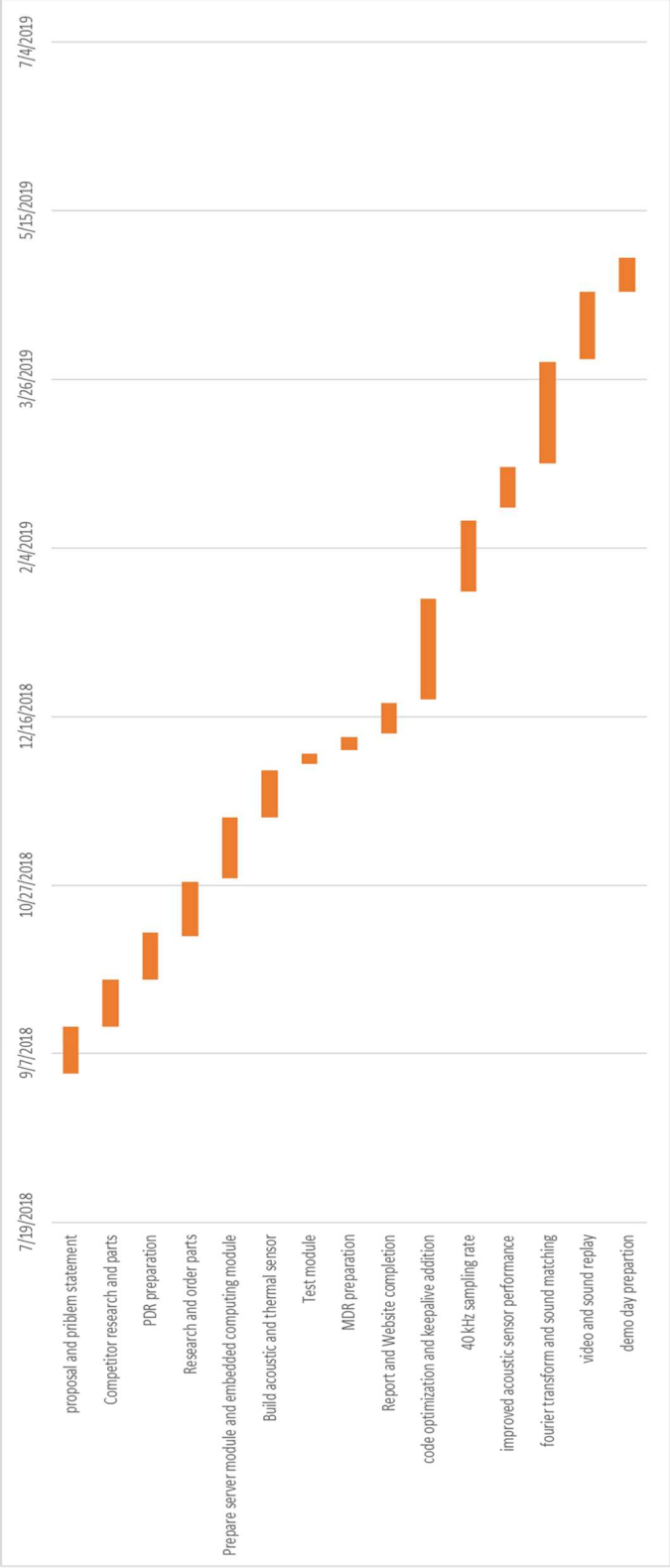
We also built the sever side of the module, utilizing a SQL database with three tables, a C# computing tier, and an Angular/HTML/CSS frontend to display the data. The SQL database consisted of a table for user accounts to remember authorized users, a table for incidents, and a table for device to location mappings. The C# middle tier receives data from the embedded module and performs the necessary computation to store it in the database or display it to the front end. The frontend displays the data for the user in a user-friendly manner, through a web-based website. Currently, this data is being displayed in a table-manner to the user.

In the future, we plan to increase the accuracy of our detection to 85% as well as utilize a VoCore embedded chip instead of a Raspberry Pie to perform the computation. We also plan to replace the table in the user interface with a map that displays the location of the gunshot and the sequence of events that occurred. We also will implement a “keep alive” function that will notify the system whether all the systems are in a functioning condition. In addition, we plan to use the keepalive data to perform averages, standard deviation and bell curves to detect true statistical anomalies and whether an alarm is merited. We will also experiment with other functions such as triangulation, and acoustic sound signature matching. The results of such attempts cannot be guaranteed and may not be employed in the final version of the project. We may also abandon the thermal camera due to its low resolution and its inability to accurately detect events from a distance greater than three feet.

In terms of difficulties, we anticipate that increasing the accuracy of the module will cause greater difficulty than expected. We also expect that tuning each individual microphone will require greater care and longer planning. We are also experiencing difficulties with our current attempts to perform sound sampling and reconstruction of sound events due to the lack of processing power on the embedded module. This limits us to matching sounds up to 10kHz, mea ning that we are unable to process all audible sounds (5 Hz to 20 kHz). We are attempting to mitigate this problem by optimizing our code to utilize less of the processing power. We are also suggesting using a separate ATmega processor for the microphone. This would offset some of the computation from the VoCore in an attempt to be able to accurately perform the sound sampling. Finally, in order to perform acoustic sound signature matching, one needs to have a prior knowledge of gunshot signatures. It is challenging however to find such data, especially ones that propagate indoors. We will continue to search for data online on such signatures or we will attempt to use an error correcting factor to perform accurate sound matching.

Appendix

The following is a GANTT chart to demonstrate the schedule for our planned activities.



ACKNOWLEDGMENT

We thank Professor Siqueira for his assistance with our project and his advice to perform Fourier transforms and sound signature matching. We also would like to thank Professor Krishna and Professor Soules for their fair criticism and their advice to implement the keep alive function and to use averages

REFERENCES

[1] "Active Shooter Detection System Launched at Triad School; A First In NC." WFMY, WFMY, 14 Aug. 2018, www.wfmynews2.com/article/news/local/active-shooter-detection-system-launched-at-triad-school-a-first-in-nc/83-583962141.

[2] Ahmed, Saeed, and Christina Walker. "There Has Been, on Average, 1 School Shooting Every Week This Year." CNN, Cable News Network, 25 May 2018, www.cnn.com/2018/03/02/us/school-shootings-2018-list-trnd/index.html.

[3] "Analysis | More than 210,000 Students Have Experienced Gun Violence at School since Columbine." The Washington Post, WP Company, www.washingtonpost.com/graphics/2018/local/school-shootings-database/?noredirect=on&utm_term=.d9f2772bfe62

[4] "Columbine High School Shootings Fast Facts." CNN, Cable News Network, 25 Mar. 2018, www.cnn.com/2013/09/18/us/columbine-high-school-shootings-fast-facts/index.html.

[5] "DeVos Gives Quiet Nod to Arming Teachers, despite Hearing from Many Who Disagree." NBCNews.com, NBCUniversal News Group, www.nbcnews.com/politics/white-house/devos-gives-quiet-nod-arming-teachers-despite-hearing-many-who-n950151.

[6] Lloyd, Whitney. "Schools Preparing for Active Shooters the Wrong Way, Experts Say." ABC News, ABC News Network, 28 Feb. 2018, abcnews.go.com/US/schools-preparing-active-shooters-wrong-experts/story?id=53360957.

[7] Smith, Ryan. "Tampa Police to Use ShotSpotter Devices in High-Crime Areas." WFTS, 19 Dec. 2018, www.abcactionnews.com/news/region-hillsborough/tampa-police-to-use-shotspotter-technology-in-high-crime-area.

[8] "The Extraordinary Number of Kids Who Have Endured School Shootings since Columbine." The Washington Post, WP Company, www.washingtonpost.com/graphics/2018/local/us-school-shootings-history/?utm_term=.ffd603c7b40f.

[9] "Timeline: How the Virginia Tech Shootings Unfolded." NPR, NPR, 17 Apr. 2007, www.npr.org/templates/story/story.php?storyId=9636137.