Coresidium: A Gunshot Detection System

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

Abstract—**School shootings have continued to plague our society, leading to unnecessary loss of life and excruciating trauma for the students. Every moment of such events is critical and could be the differentiator between life and death for those within the building. Coresidium is a project that aims to detect gunshots in indoor locations in an effort to reduce response times during school shootings. It uses an acoustic sensor, a thermal infrared camera and an embedded processor to detect gunshots locally. It then sends the sampled data to a sever for further processing. The server determines whether the event is a potential threat, stores the data and displays it to the user through a web-based interface.**

I. Introduction

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

In response to this problem, Congress has been unable to find a proper solution to address the number of fatalities and to stem the tide of violence occurring yearly 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. 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 given their children bulletproof backpacks in order to reduce the chances of them being fatally shot or injured [10]. Finally, on the recommendations of current 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 [13]. While these techniques may offer a temporary solution to the problem, they do not offer a complete one since it leaves the students feeling vulnerable and unsafe during their school day.

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, significantly increasing the time it takes for the victims to receive the proper attention and medical care they need [3]. 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 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 an academic building until the building is declared safe, during three separate shooting incidents. It should be noted that although the incidents happen in completely different decades, and although the response time decreases as time passes, 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. Furthermore, this increased delay can leave anxious parents unaware of the status of their children, increasing their unease.

Congress and schools are not the only parties that attempted to tackle such a problem. Private 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 reasoning behind our project, as they can be installed at large costs to the school districts, making them unaffordable to most schools [1].

It is difficult to come by exact figures for the cost of installing such systems, since they often rely on multiple variables including the area of the school, 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-Amherst Police Department, we were informed that prices of such systems are unaffordable for a public institution with the size of the University of Massachusetts-Amherst [September 15, 2018]. Through online research, we located just one school that

B. J. Cross from Westfield, MA (e-mail: bjcross@umass.edu)
V. Degtyarev from Westborough, MA (e-mail: vdegtyarev@umass.edu)
A. E. Eshak from Westborough, MA (e-mail: aeshak@umass.edu)
A. J. LaMarche from Southhampton, MA (e-mail: alamarche@umass.edu)

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 1,000-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 through different materials. Most systems 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 differentiate between a gunshot and firecrackers. Finally, such systems are also immensely expensive, making them inapplicable for school use [1].

In response to such difficulties, we aim to design an inexpensive 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

Requirement	Specification	Value
Functionality	Range	10 feet per module
Functionality	Response time	<1 second
Functionality	Accuracy	>80%
Price	Cost	<$100 per module
Functionality	Sensitivity range	>130 dB
Functionality	Timestamp accuracy	<1 second
Functionality	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 module to coordinate between the multiple nodes, analyze the data and output the results to the user [see Figure 1]. 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 is unidirectional, meaning it cannot communicate back to the embedded modules.

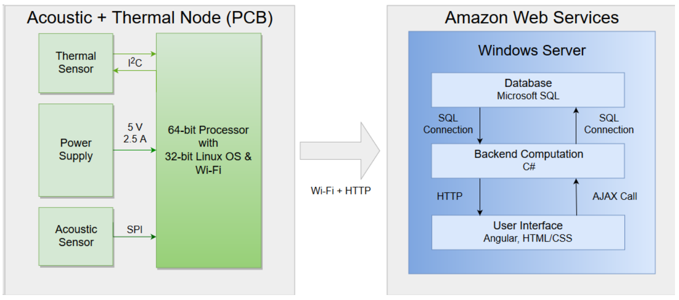


Fig. 1. The Block diagram of the Coresidium system. On the left is the embedded module while on the right is the computing node that utilizes an Amazon Web Services Instance.

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 64-bit Raspberry Pi 3B [16], running a 32 bit version of the Linux operating system, 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 traditional power brick supplying five volts and a maximum of 1.34 amps, giving a maximum use of 6.7 Watts [16].

On the server end, the computing node consists of an Amazon Web Services Instance running a Windows Server. It is powered by an Intel Xeon processor with 16 GB of RAM. Within the server, a three-tiered system is present 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 on the top level 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 [8] to dynamically display the data in a user-friendly manner, without the need for a data expert to perform analysis.

It is also important to note at this point that other proposals have been explored in order to address this problem. In our initial pitch, the thermal sensor replaced a traditional camera and machine vision to recognize the images of a gun. This required the use of neural nodes, high performance computational 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 1,500 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 hindering the machine's ability 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 original design, we used a KY-038 big sound sensor module [6]. This sensor detects large sounds, amplifies its signal, compares it to a predetermined value and outputs it through pin one. However, for our final design, we used a capacitor-based Electret microphone due to its ease of use in our PCB design [4]. This helped us eliminate the need for breakout boards and to customize the circuitry to our individual needs. Along with the microphone, we used an analog to digital converter [14] to convert the analog signal into a digital signal and a comparator chip [18] to compare the newly digital signal to a predetermined value. The predetermined value was originally configured through a potentiometer and varies from module to module. However, in our final design, we simply used a standard resistor value across all our PCBs, reducing the variation between each module and the need for individual calibration of the microphone. To conclude, the changes that were made for this module include the elimination of the breakout board, as well as the potentiometer and replacing it with a uniform resistor value used across all the modules to reduce variation and the need for calibration in the system.

For this 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 outputs of this module represent. This means we must attempt to find a relation between the output voltage and the physical noise being received.

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 emergency 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, aerospace grade microphones, such as the HOLMCo 82-03-08274 can pick up sound in our desired range [9]. 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 twenty-five balloons within twelve feet of four 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 during our midway design review was 80% success with no false positives and 20% false negatives. However, for our final review, we received more favorable results of 96% success and 4% false negatives. Such an increase in our success can be attributed to the use of standard resistor values as well as the incremental improvements to our code that have improved our detection algorithm. Finally, the fact that our system does not detect any false positives leads us to believe that our system was conservative in detecting such incidents.

C. Thermal Camera

The thermal camera is 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 [12]. It has a 32x24 resolution. 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.

In the design used for our midyear design review, we used a camera with a 110° by 75° field of view [12]. This provided us a wider image while reducing the distance that we can accurately detect. Furthermore, we were struggling to process 32 frames per second using the VoCore [11], the original low power process that we chose. Such attempts proved to be unfruitful since the camera was unable to detect hot objects at distances that were sufficient for our application and was proving to be computationally strenuous on our low-power, one core, 580 MHz VoCore [11]. It was clear at this point that this sensor was not appropriate for our project, leading to a discussion of whether we should abandon the thermal part of the project and rely solely on the microphone.

However, on the recommendations of Professor Siqueira, we replaced the original sensor which had a 110° by 75° [12] field of view, with the same model that has a 55° by 35° [12] field of view. It was pointed out that under ideal conditions, one can achieve four times the distance of the original sensor, which should make it applicable for our application. Secondly, we decided to replace our original VoCore [11] processor with a Raspberry Pi 3B [16] since it provides more computational power, a topic will be discussed in greater detail in the following section. This allowed us to process the 32 frames per second that we viewed as adequate for our purpose, and brought the thermal sensor part of the module back into contention, allowing us to have two sources of data to detect gunshots in indoor locations and providing our module with greater accuracy and redundancy.

Other improvements to this module involved using the highest pixel value above our threshold instead of using the first pixel value above our threshold when detecting an event. This allowed us to provide a datum with greater significance or meaning to the backend for computation. Secondly, we implemented the keepalive function, which allows us to know which models are offline or broken. For the keepalive function, we used an average of the pixels taken every 60 minutes and sent it to the backend to be used to calculate the percent confidence when an event is detected. Further discussion of the percent confidence calculations is discussed in the backend computation section. Finally, we eliminated the use of breakout boards for this module

In order to test this module, we flickered a lighter from three feet away one-hundred times and recorded the number of successes over the total number of flickers. Our original sensor, with a field of view of 110° by 75°[12] resulted in a 68% success rate, a result that is both disappointing and one that we believe we could not improve on with such a setup, However, with the 55° by 35° [12] we were able to achieve a 100% success rate from both 3 feet and 6 feet away from the sensor. In fact, this result continued up to 9 feet away from the sensor. With distances greater the 9 feet away, the sensor receives success rates below 30%, making its maximum usable distance between 8 and 9 feet away for our module. We consider this distance acceptable since most school hallways are between 8 and 10 feet away.

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.

D. 64-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. It is responsible for communicating with the acoustic sensor via SPI to perform the event detection via sound and the keepalive for the sound sensor. It also communicates with the thermal sensor via I2C to perform the event detection using the thermal data and the keepalive for the thermal sensor. It is this module that consumes power from the power supply and supplies it to both sensors and performs the communication between the embedded module and the cloud module.

Originally, we were aiming to use a VoCore 32-bit embedded chip [11] running Linux OpenWRT, a low power version of Linux specifically designed for embedded chips. This chip is prebuilt with WiFi and an operating system, allowing us to focus on the goals of our project. However, while interfacing with this chip, we encountered quite a few challenges that forced us to change our direction. Our first challenge is that such a chip required us to write our own drivers to store data into memory, and to communicate via SPI and I2C. Such drivers were difficult to write, time consuming and were not fully optimized for our purposes. As a result, not only were they a challenge to program but were also a performance bottleneck for our system. For example, using our customized drivers on the VoCore, we were able to perform 21,000 bit I/O writes, making it inadequate for reading both the sound sensor and the thermal sensor data. Secondly, the VoCore did not provide the necessary performance for our computation needs due to its inherent design. The VoCore is a single core, 580 MHz processor running a Linux operating system [11][5]. Adding our own drivers and our programs to this setup meant that the VoCore was simply not able to process our information at the required rates, leading in degradation of performance, such as not being able to process 32 frames per second.

As a result, we decided to transition to a Raspberry Pi version 3B. This computational module is a 64-bit processor with a 32-bit Linux operating system [16]. It has a built-in Wi-Fi chip and does not require us to write our own drivers for I/O and memory storage. This allows us to focus on the important and challenging aspects of our project and to concentrate on achieving the goals of the team [16]. Another benefit of the Raspberry Pi is its computational performance. The Raspberry Pi version 3B has a 1.2 GHz clock speed along with 4 cores which is a significant increase over the performance benchmarks of the VoCore that were mentioned above [16]. Such improvements allowed us to access sensor data from both the thermal sensor and the acoustic sensor simultaneously as well as run the keepalive with no bottlenecks. For example, we were easily able to get the 32 frames per second needed for the thermal sensor along with acoustic sensor detection and keepalives for both the acoustic and thermal sensor.

Another change that was implemented for this module in our final design was the use of four threads instead of two for our computation. Originally, we were planning on using two threads, one for the acoustic module and one for the thermal module. However, after careful consideration, we decided to use for threads, one to implement the detection of an event for the thermal sensor, one to implement the detection of an event for the sound sensor, one to implement a keepalive for the thermal sensor and a final one to implement a keepalive for the sound sensor. Such implementation was difficult to program since it required the use of locks since only one thread can access sensor data at a time. As a result, we used some of the techniques that we learned in ECE 570: System Software Design.

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 or the heat of a hand. Finally, we tested both keepalive threads by allowing them to send keepalive signals every 30 seconds instead of every 60 minutes. One can check if all the threads work properly by checking if the server receives data in the incident reports and the keepalive reports. If the server does not receive the data, one knows that the problem is with the individual sensor since the code responsible for sending data over the internet has already been verified.

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 [15]. This would have been much cheaper considering that the Raspberry Pi Version 3B costs approximately $39.99 while an Atmega 32-bit processor would cost below $2 [15] [16]. However, this was not pursued due to the superior performance of the Raspberry Pi, running at 1.2 GHz compared to the Atmega's 48 MHz [15] [16]. This was also chosen since it is a complete module onboard memory, a dedicated operating system, and a built-in Wi-Fi chip that does not need special configuration or other hardware. One disadvantage however of the Raspberry Pi is its power consumption since it uses 1.34 A compare to the VoCore 0.234 A [11][16]. This power consumption requires to use a power brick instead of a power battery, decreasing the portability of the system.

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.

It is important to note at this point that other options have been considered, such as storing the data in a JSON file , a CSV 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 several 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 side of the project is based on the C# language. Its main purposes include receiving data from the embedded module, storing it in the SQL database, processing the requests made by the dashboard, and packaging 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, performs some basic data processing such as discarding of erroneous records, 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.

Changes that were made to this module included implementing the keepalive by distinguishing them from actual events and pushing such differences to the frontend in an appropriate data structure as well as calculating the percent confidence as a measure to how confident the system is that an event has been detected. The percent confidence relies on the keepalive data of each individual sensor, taking the average of such a data and standard deviation. Followingly, when an event occurs, one can calculate how many standard deviations away from the average the event data is. For example, one can store the keepalive data sent from the acoustic sensors and calculate the averages and standard deviation of such data. Then, when an event is detected, one can use a gaussian estimation to calculate how many standard deviations away from the average the incident data is. This would ideally give us a percent confidence about whether an event that is detected is truly an alarm that needs to be considered by the user or whether it is an erroneous piece of data. One problem that we encountered with such an implementation is that our keepalive data did not resemble a gaussian function but was more of a uniform function. This made our standard deviation relatively small and our percent confidence extremely large, making it inapplicable for further application. It was however kept in the current version of our project as a proof of concept.

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 observe if the requested data is returned to the developer console in Google Chrome. This enabled us 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.

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. If the outputs of other modules function correctly, then one can assume that this module also operates correctly since all other modules are dependent on this one.

G. User interface

The final component of this module is the user interface. This interface is web-based and is based on HTML/CSS and Angular [8]. 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 has three main pages to communicate data to the user. The first page is the device report page which relies on the keepalive data from the middle tier. This page demonstrates the current status of the devices in our arsenal. If a device has not sent a keepalive signal within the last 60 minutes, it is highlighted in the page as a broken device, otherwise the devices are labeled as working. A screenshot of the Device Report page is shown in figure 2 in the appendix of our report. The second page was the events table page in which all the events that have been detected by the system are displayed to the user in a table format. This allows the user to look for a specific event and to sort events in a timely order or in accordance with a specific sensor or module. A screenshot of the events table page is shown in figure 3 in the appendix of our report. The final page was a mapper page that provided a physical mapping of the location of the sensors and highlighted which sensors were triggered by an event as well as the time the event was detected. This allows the user to view a temporal sequence of events and the actual location of the attacker within the building. A screenshot of the mapper page is shown in figure 4 in the appendix of our report.

The final change that was made to this module included using bootstrap and removing a few bugs that were present during our midyear design review. We also added a refresh button allowing the user to refresh the mapper to see if any new events have occurred since the page has been loaded. Finally, his module relies on information we learned in the software engineering course.

III. Project Management

Table 3 provides the specifications that were promised by the end of the year as well as which ones were achieved at this point. Such goals include providing greater than 80% accuracy for our acoustic sensor, as well as the ability of the thermal sensor to recognize hot objects. Furthermore, we promised to finish our SQL database design, the frontend dashboard, and the middle tier to communicate information to both modules. All the previous requirements have been satisfied.

Our first specification was to deliver a product that could provide 10 feet of range since we estimate that this is an average width of a hallway, making it our product applicable for most schools. During our testing, the acoustic sensor was easily able to achieve a range of 12 feet while maintaining accuracies of above 90% while the thermal sensor was able to pick up flames at distances of 9 feet, making it adequate for most conditions, especially considering that it often works in contagion with the sound sensor.

Secondly, we wanted our sensors to sample at approximately one sample per second since we do not believe that a shooter is capable of firing over 60 rounds per minute in one individual location. However, we were able to achieve samples of just under 1 millisecond since the Raspberry Pi often has to scan all the pixels of each individual frame of the 32 frames per second and find the average or the highest pixel depending on the functionality. As a result, we believe that we have easily overachieved for this goal and have confidence that a gunshot will not be missed due to the low sampling rate of our system.

We also wanted our system to be able to detect 80% of simulated gunshots under any condition. This meant that under both noisy and quiet conditions, if a gunshot was simulated using a handclap or a balloon pop, our system should be able to detect 80% of such cases. However, during our tests, the system was able to pick up 96% of such simulated gunshots making such a goal one that was achieved.

In terms of the acoustic sensor, we wanted to focus on sounds that were above the 130-dB noise level. To put this into perspective, a typical conversation will have a maximum of 90-dB while a balloon pop will typically be around 110-dB. Finally, a gunshot ranges between 150-dB and 190-dB. As a result, we believe that focusing on sounds that are above the 130-dB range allows us to ignore the typical noises that may occur in a school or an indoor environment in general while focusing on all the noises that could be associated with a gunshot. In our project, we were able to focus on noises that were above the 130-dB mark and this was simulated using an airhorn and measured using a decibel meter that was often saturated in our tests. Our module was easily able to detect sounds in the range while ignoring sounds that were below such a level, even in noisy conditions that were simulated using traditional conversations and speakers. As a result, our system is able to meet that requirement.

Most importantly, it is crucial to note that the original goal of our system was to notify the proper authorities and the users of our system with the relative location of the shooter within the building. The relative location was defined as the side of the building (east, west, north and south) and the floor within the building. This was achieved using the mapper page that translated the MAC ID of the triggered device into a location of the device within the building and highlights it. Therefore, using that page we were able to achieve the most important goal of providing the relative location of the shooter.

Finally, we wanted our system to cost under $100 per module. This requirement was placed as an attempt to make this system affordable for schools since our system was primarily designed with school campuses on mind. Our development costs, which is the cost of building one module came in at $91.50 while our production cost, which is the cost of building 1000 modules came in at around $72.88. The breakdowns of the development and production costs are shown in table 4 and table 5 respectively.

TABLE 3
SPECIFICATIONS AND ACCOMPLISHMENTS

Requirement	Desired	Achieved
Range	10 feet per module	10 feet per module
Response Time	< 1 second time	<1 second
Accuracy	>80%	>90%
Sound Sensitivity	>130 dB	>130 dB
Location Accuracy	Location and floor	Location and floor
Timestamp Accuracy	<1 second	$\approx 1\ ms$
Cost	<$100	$72.88

TABLE 4
DEVELOPMENT COSTS OF CORESIDIUM

Item	Development Cost
Raspberry Pi 3B+	$35
MLX90640 Thermal Sensor	$44.99
Electret Microphone	$0.95
MCP3008 ADC	$2.75
Resistors	$5.40
Capacitors	$0.50
Diodes	$0.50
LM393N	$1.0
Header	$0.01
Custom PCB	$0.4
Total	**$91.50**

TABLE 5
PRODUCTION COSTS OF CORESIDIUM

Item	Development Cost
Raspberry Pi 3B+	$34.50
MLX90640 Thermal Sensor	$34.04
Electret Microphone	$0.86
MCP3008 ADC	$1.71
Resistors	$0.20
Capacitors	$0.08
Diodes	$0.09
LM393N	$1.0
Header	$0.01
Custom PCB	$0.4
Total	**$72.88**

It is important at this point to discuss the achievements and modifications that were made for each individual component of our module. For the acoustic sensor, our first accomplishment was to eliminate the breakout board and to integrate the sensor directly into our PCB. This involved using a standard resistor value across all the sensor modules which enabled us to avoid calibration and modification of each individual sensor as well as the use of an amplifier and an Analog to Digital Converter [14] [18]. We also replaced the KY-038 acoustic sensor with an Electret Microphone due to its ease of integration into the PCB board and its availability [4][6].

For the thermal sensor module, we also eliminated the breakout board integrating the thermal sensor directly into our PCB. Secondly, we used a sensor with a small filer of view, 55° by 35° , instead of a 110° by 75°, giving us a greater range of detection [12]. We also modified the code that was used for the detection of events, so that when an event is detected, we send the highest pixel value above the threshold instead of the first value above the threshold. Finally, for the keepalive, we decided to send the average value of the pixels in the keepalive image.

For the embedded computing module, we abandoned our pursuit of using the VoCore Version 2 and continued to use the Raspberry Pi Version 3B due to its improved performance [11] [16]. We used four threads in this module, instead of two. Two threads were used for the keepalive of the acoustic module and the thermal sensor respectively, and the other two threads were used for detection of events on thermal sensor and acoustic sensor as well.

No changes were made to the SQL server. This is due to the fact the three tables that were initially constructed were flexible enough for all our future uses providing user access control, location mappings and keeping track of both keepalive and detected events.

For the middle tier, we continued to use C# but we implemented functions to check for keepalive signals and provide it to the frontend to display to the user. We also implemented the percent confidence function so that one can determine how far an event is from the average and how confident the system is that an event detected by the embedded module is an actual event worthy of alerting the user over. We also implemented a few bug fixes to ensure the robustness of the system and to prevent the system from breaking down under certain edge conditions.

Finally, for the frontend, we implemented two new pages so that users can tell which devices are broken as well as to provide a visual mapping of the devices and the current location of the shooter within the building, We also added refresh buttons for our pages so that users can update their maps or tables to keep their pages up to date without the need to restart the entire application.

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 along with its code and its tests. Additionally, Valentin focused on the hardware and PCB design of the module. 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 HTML/Angular [8] 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, 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

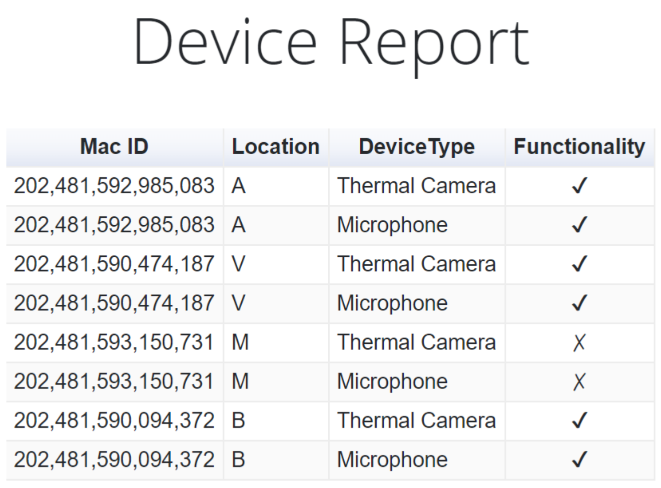
In conclusion, Coresidium is a system that aims to detect gunshots during school shootings, in an effort to reduce response times and the number of fatalities. During the first semester of the 2018-2019 year, our team has built the embedded module utilizing both an acoustic microphone and a thermal camera as well as a Raspberry Pi Version 3B [16]. 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 Pi 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 HTML/CSS/Angular [8] 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. Such data is displayed in a table manner so that the user is able to check all the detected events, as well as in a table manner so that the user can view a pictorial time sequence of the events occurring within the building.

Some of the difficulties that we encountered during this project included the need to implement locks within our threads for our embedded computing module, a topic that we were in the process of learning about it and were not entirely familiar with. We also encountered difficulty choosing a resistor value for our microphone sensor due to the variation in the internal resistance of the microphone itself. No other difficulties were encountered with our system.

Appendix

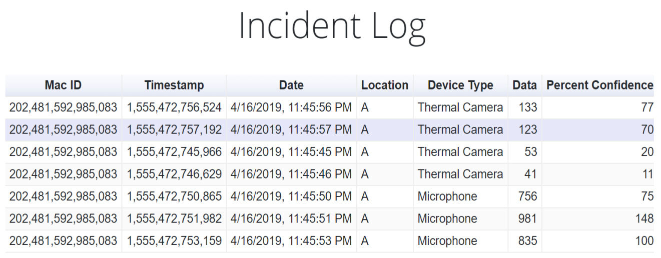
Figure 2
DEVICE REPORT PAGE



Device Report

Mac ID	Location	DeviceType	Functionality
202,481,592,985,083	A	Thermal Camera	✓
202,481,592,985,083	A	Microphone	✓
202,481,590,474,187	V	Thermal Camera	✓
202,481,590,474,187	V	Microphone	✓
202,481,593,150,731	M	Thermal Camera	✗
202,481,593,150,731	M	Microphone	✗
202,481,590,094,372	B	Thermal Camera	✓
202,481,590,094,372	B	Microphone	✓

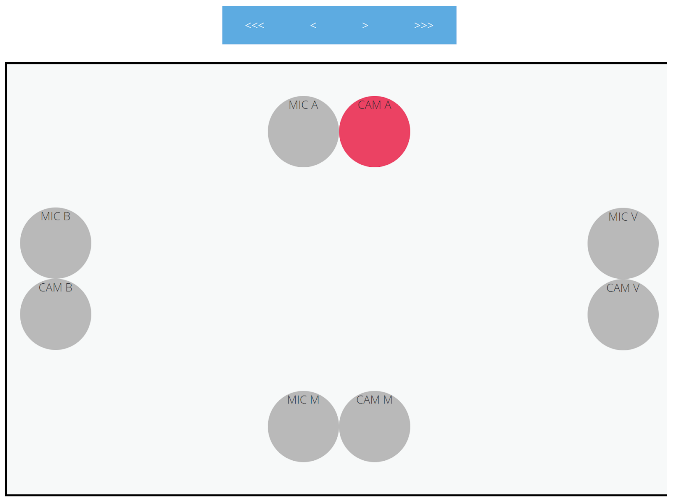
FIGURE 3
INCIDENT LOG PAGE



Incident Log

Mac ID	Timestamp	Date	Location	Device Type	Data	Percent Confidence
202,481,592,985,083	1,555,472,756,524	4/16/2019, 11:45:56 PM	A	Thermal Camera	133	77
202,481,592,985,083	1,555,472,757,192	4/16/2019, 11:45:57 PM	A	Thermal Camera	123	70
202,481,592,985,083	1,555,472,745,966	4/16/2019, 11:45:45 PM	A	Thermal Camera	53	20
202,481,592,985,083	1,555,472,746,629	4/16/2019, 11:45:46 PM	A	Thermal Camera	41	11
202,481,592,985,083	1,555,472,750,865	4/16/2019, 11:45:50 PM	A	Microphone	756	75
202,481,592,985,083	1,555,472,751,982	4/16/2019, 11:45:51 PM	A	Microphone	981	148
202,481,592,985,083	1,555,472,753,159	4/16/2019, 11:45:53 PM	A	Microphone	835	100

FIGURE 4
MAP PAGE



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 to accurately detect an event.

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] Challenge Electronics "Electret Microphone Datasheet" CEM-C9745JAD462P2.54R, 2010

[5] "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.

[6] Datasheets.com "Arduino KY-038 Microphone Sound Sensor Module"

[7] "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.

[8] https://angular.io/

[9] HOLMCO "Dynamic Microphone with Gooseneck" Series MG-50-D, Aug. 2013

[10] 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.

[11] MEDIATEK "VoCore 2 Datasheet", MT7628 Complete Datasheet, July. 2012 [Revised Sep.. 2012].

[12] Melexis "MLX90640 Datasheet", MLX90640 32x24 IR array Datasheet, July. 2016 [Revised Aug.. 2018].

[13] Merica, Dan, and Betsy Klein. "Trump Suggests Arming Teachers as a Solution to Increase School Safety." *CNN*, Cable News Network, 22 Feb. 2018, www.cnn.com/2018/02/21/politics/trump-listening-sessions-parkland-students/index.html.

[14] Microchip "2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI Serial Interface", MCP3004/3008 Datasheet [Revised. 2008].

[15] Mouser Electronics, "32-bit ATMEL AVR Microcontroller", AT32UC3B Complete Datasheet, May. 2007 [Revised Jan.. 2012].

[16] Raspberry Pi "Raspberry Pi 3 Model B+", Raspberry Pi Model 3 B Complete Datasheet [Revised Aug.. 2018].

[17] 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.

[18] Texas Instruments, "LMx93-N, LM2903-N Low-Power, Low-Offset Voltage, Dual Comparators Datasheet" [Revised Oct. 2018]

[19] "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.

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