Final Design Document
Connor Kutz, Cole Weinhauer

Introduction

<u>Description</u>**:** The desired outcome of the proposed activity will be an android application that can be used to act as an interface between users wearing headphones and incoming audio stimuli. Specific goals in terms of functionality include alerting users about upcoming heavy traffic and crime using GPS location services, changing the volume levels of your device based both on general and specific audio, like your name or screaming, honking, and other things of the nature, and finally creating customized profiles for specific locations so users can have preset conditions for the application settings that will activate whenever they enter a preset radius of that said location. To do this, we will develop our application with 6 major components: name recognition system, sound recognition system, active volume management systems, crime and stats module, weather module, and maps and overlay helper.

<u>Purpose</u>**:** The purpose of this application is to fill a societal need: it will provide a system to warn pedestrians who are listening to music about various possible hazards, including crime and traffic. Many people, especially teens and young people, become trapped in a “bubble” while listening to music while acting as pedestrians in lively cities. In a study published in *Injury Prevention*, it was found that that both the number of pedestrians killed while listening to music has nearly tripled over the years from 2004 to 2011 and that in nearly 70% of the cases, the accident was fatal.

<u>Objectives</u>**:** One big key objective our system is trying to accomplish is both the fast and correct classification of sounds. Essentially, our system gets most of its functionality from classifying sounds, so it will need to be accurate to have a successful system. Similarly, since we will try to classify every sound the microphone hears, we need to compute the Mel-Frequency Cepstrum Coefficients (MFCCs) in computationally effective way. Another such objective is a more broad one: can we effectively gather the sounds in the real-world environment. One immediate worry is that combining sounds with the general noise of cars, people, and other things will cause our sound recognition model to have problems classifying. Thus, during our next phase, we will spend time testing the system. If we have errors, the initial solution we could try is adding ambient noise to our urban sounds data and then getting the MFCCs so that it models better in the real world. Another solution we could attempt if this problem occurs is trying to remove the noise from the real-time audio we get, but this would be significantly harder. Finally, we need to see if we can accurately and quickly get our crime updates as well. For our product to be successful in keeping its users safe, the crime alerts will need to be timely. In the real world, a delay of only 1 minute could put users in a dangerous situation, so the product will be vigorously tested to ensure that our service consistently and quickly performs as expected.

<u>Technical Milestones</u>**:** For this product to be successful, we need to overcome a few technological hurdles. One obstacle is gaining the ability to constantly and seamlessly listen to and analyze the user’s surroundings. One of the challenges posed by this goal is gaining user permission for audio recording and processing. Often times, users are wary of allowing apps to use the phone’s microphone so our reasoning must be clearly explained to the user upon requesting permission. Similarly, having a constant service

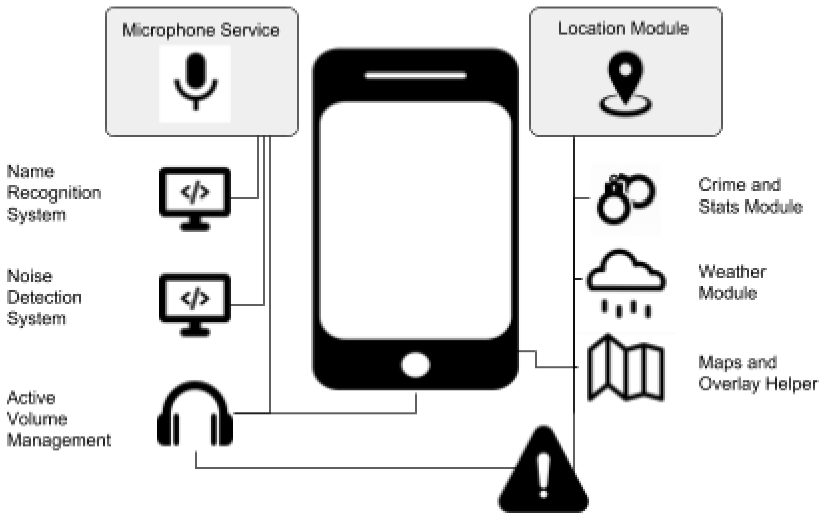
dedicated to audio processing runs the risk of being resource intensive, so it is important for us monitor CPU, memory, battery usage, etc. The only way for this app to be marketable is if it can be made to run in an unobtrusive and subtle way. We're already combating this by deploying our sound recognition model to the application after we have already trained it, and also precalculating MFCCs values.
Another milestone we need to hit is accurate classification of the real-time sounds. This is integral for our project as classifying the sounds the user hears provides most of the functionality of the application. Because of this, we want our model to correctly classify things 70% of the time, which we feel is a respectable amount and is below what our model is classifying now (without noise though, so that should cause a decrease in accuracy). Also, because of the nature of the application, we will try to have our false positives (saying there is a threat when there isn't) make up a much larger percentage of the inaccuracies as compared to our false negatives (giving no alert after a car honk, a gunshot, etc).

Users and Use Cases

The market of our application is the subset of mobile device users who walk outdoors with music playing in their headphones. This is quite a large group of users. In a 2009 survey, The National Household Travel Survey estimated that there were around 40.9 billion walking trips done each year, so many people could benefit from this app and thus we can consider them in our addressable market. Specifically, in big cities there is a very large market. It was found that over 355,000 people enter Times Square daily as pedestrians. Obviously a few things drive the market. One could be gas prices: if they were lower, then we would see a decrease in pedestrians using the app (because there would be less pedestrians) and the opposite if the prices went up. Similarly, we could consider the temperature and weather as a driver in the same way: colder, rainier, and generally worse weather causes people to walk less, and nicer, more enticing weather will make people walk more.

There are 3 major consumer demographics we expect to be interested in such an app. Firstly, pedestrians in cities that have a large amount of commuters and traffic. Due to the quickness of traffic in these areas, we expect to need to be able to respond quickly to possible auto accident stimulus. Similarly, due to typically higher amounts of traffic and crime in these areas, finding more up to date data in these areas is integral for the success of the app for these users. Another major demographic is parents who will buy this product for their children to help bring them peace of mind about their children commuting, to and from school, sports practice, etc. This requires that the user interface to be simple and easy to use. A final user demographic we are anticipating are blind/ visually-impaired people. Obviously, the reasons why they would use the application is trivial. Because of this we need audio instructions and cues to describe where certain buttons are located on the app.

Major Components



The above figure shows the 6 components of our system. Below, we will go over what they do, how they do it, the interactions this system has with the other major components, the containers for these systems, and the timeline for each system's development.

Sound Recognition System: This system is in charge of developing the model we will use to classify our real-time sounds. the sound recognition system is a Python-based machine learning algorithm. The system utilizes the UrbanSounds8K Dataset as developed by J. Salamon, C. Jacoby and J. P. Bello. This Dataset supplies 3 things for each entry: a WAV file that is composed of the sound of the event, the metadata of that file, and a classification of the sound. From here, we will get the Mel-frequency cepstral coefficients (MFCCs). A Mel-frequency cepstrum is a representation of the short-term power spectrum of a sound, based on a cosine transformation of a log power spectrum on a nonlinear mel scale of frequency, and the MFCCs are the 40 values that are used to model the Mel-frequency Cepstrum. We will then use these values to train our model. The one we will be using is the sequential neural networks model provided by the Keras library. We will use a train-test split of 80-20 on our data to validate our model. Upon running the model and validating it with the test data, the model was accurate typically in the mid-70s percentage. After developing the model, we have to freeze it and turn it into a .pb file that can be deployed on our application. We developed a small-python script to do this. Finally, we have to create .so and .jar libraries for the model, which was done by executing a .sh script, and then put the libraries within our applications gradle. After this, we have a python-developed model we can invoke on our android device to classify sound. The sound recognition system interacts with the active volume management. The mic passes the system a piece of audio and it runs the model on it to classify the sound. In the event that the system detects some sort of threatening sounds, like honks or screams, it will trigger an event and have the system sound decrease. This system began development in October and is still undergoing development and testing and is scheduled to be finalized in February.

Name Recognition System: This system's responsibilities are to 1) distinguish whether a sound is a user's name and 2) deciding what to do if it is/isn't the name. To do this, we intend to have a user enter in their name initially when first starting to use the system. We will compute the MFCCs as we did before, as well as use speech-to-text and create a list of all words that the system interprets the user as saying when we are configuring the name. Now, when a sound is encountered by the system, it will compute the MFCCs of it, get its speech-to-text result, and then pass them to the system, which will compare them to the archived values of the user's name and see if they are close enough to distinguish a match. If the system has a hard time deciding, we can use the fall-back of the speech-to-text results to decide if the user's name was said. In the instance it is found to be the user's name, we will lower the volume so the user can hear who called their name. The System interacts with the active volume management system in the same way as the sound recognition system. Mic audio is passed to it, it decides whether or not its the user's name, and if it is, then an event is triggered and the active volume management system is told to lower the volume. Unlike the sound recognition system however, it is developed entirely on the application. The system is currently not under development but will be started on sometime in December. It is intended to be finished by February.

Active Volume Management System: One feature of our design is that audio levels are actively adjusted to match your environment. This means that when in a quiet environment, the media volume level will be a quieter setting. Then, if you go into an area with a high level of ambient noise your volume will increase to remain audible. As the majority of our users will use our product while listening to music, this feature acts as a convenience to ensure that media can be heard over the ambient noise level. This feature is also a safety design to ensure that users are able to hear the alerts announced by the app. This is all done by a persistent, threaded service which gathers average noise level data. When in increase or decrease in ambient noise is detected, volume output is adjusted accordingly.

Crime Alert and Data System: The crime and safety alerts feature is one of the most important pieces of our final product. This is the part of the app which warns the user of possible dangerous situations in their vicinity, for example, a personal robbery on a nearby street. The goal of this feature is to actively warn the user of dangerous crime incidents so they can take action to stay safe and avoid the possible suspect. Another aspect of this feature is to obtain and display crime density information via a MapView to preemptively show the user dangerous areas that should be avoided. These two pieces are implemented in different ways. First, for time sensitive and location based alerts, a polling service will check a database containing rapidly updated crime reports for relevant results. When there is an alert, it will be broadcasted via text-to-speech to the user through the audio output stream along with a short notification with extra details. For aggregate data of crime incidents and density, there is a service to perform a daily update and recalculation of the most dangerous areas to avoid. This is then shown in the map activity as a heat map.

Classes, Software, APIs, Libraries

Classes:

Crime Alert Class is the class that implements real time crime alerts. Once started, this class runs on a separate helper thread meant to poll the SpotCrime API for updates and alert the user if necessary.

Crime Data Class is the class which is used to get aggregate crime data over periods of time. This can be called from anywhere within the app and returns GeoJSON data for the past 30 days of crime in DC.

Maps Class is the activity that allows the user to actually see a map with crime, traffic, and weather data.

Location Class is the class that can be called when the user's current location needs to be known. The permission checking logic is hidden within this class so all that needs to be done is to call the getLocation function.

Microphone Class is the class which houses the Active Volume Management system entirely. It is instantiated as a separately threaded service and can be started and stopped from the Main Activity.

Speech Class is the class which can provide an alert to the user from anywhere within the app. It is an AsyncTask which means it can be instantiated and executed automatically to broadcast text to speech to the user at any time.

Software: We use the Pycharm Python IDE to develop our sound recognition system. We also developed scripts to create our .so and .jar libraries that will be utilized on the application. For app development, Android Studio is used in combination with GitHub for version control.

APIs

SpotCrime provides us with real time updates of dangerous/violent crimes committed within a specified distance to the user. This is used to warn the user if there is a dangerous area or recent crime they may need to avoid. We request crimes within a recent time frame and within a small radius of the user and if it is decided to be dangerous or noteworthy, the user is alerted.

OpenData.DC provides us with aggregate crime statistics data in the DC area. This is the API that allows us to warn users when walking into a historically dangerous area. We use OpenData to make a request for the past 30 days of crimes committed in DC. With this data, a heat map can be made showing crime density to the user and providing our app with safety data.

OpenWeatherMap will allow us to display current weather conditions to the user via map and main screen. This API also gives us the ability to provide real time alerts in the case of a weather emergency such as a tornado or blizzard. This API can be queried for alerts, current data, or historical data to fit our purposes.

Libraries: For the sound recognition system, we utilize a variety of Python libraries. We use Pandas and Numpy to read in our data and to do actions on it. We use the sklearn library to encode our labels. Finally we use the Keras package to get our neural network model.