FUNCTIONAL DESIGN DOCUMENT

FOR

CI RAINBOW PROJECT:

AUDIO RECORDING & FEATURE EXTRACTION

Version *1.0*

Prepared by

Student Name: **Cameron S. Embree**

Instructor: **Prof. AJ Bieszczad**

Course: **COMP 499 Capstone**

Date: Nov 4, 2014

INDEX – TABLE OF CONTENTS

3.B) FUNCTIONAL REQUIRMENTS – **IN DEV**
3.C) PERFORMANCE REQUIRMENTS
    3.C.i) Power Consumption – **IN DEV**
    3.C.ii) Sending Data to Server
    3.C.ii) Feature Extraction Time, Memory, & CPU Requirements
3.D) DESIGN CONSTRAINTS
    3.D.i) Choice for working space limit of 2 Gigabytes
    3.D.ii) Use of 3.5mm mic and USB Sound Card
    3.D.iii) Why *Arecord* and *Yaafe* for Recording & Feature Extraction
3.E) LOGICAL DATABASE REQUIRMENTS
3.F) SOFTWARE SYSTEM ATTRIBUTES
    3.F.i) Updating recording settings files remotely
    3.F.ii) Background noise removal reliability (filtering)
    3.F.iii) Security and Data Integrity
    3.F.iv) Recording and FV Extraction Portability
3.G) ASSET REQUIRMENTS
3.H) OTHER REQUIRMENS

1. INTRODUCTION

1.A PURPOSE

This document provides background to the goals of this project and explanations of motivation for design methodology to its completion. References are provided bellow for of the behavior of the audio recording system, network communications and their form, and the audio feature extraction system. An explanation of the audio analysis system is provided with explanations of how these systems work assuming the reader has no background knowledge of these techniques. This application is a proof on concept and may eventually be used as part of the CI Rainbow project on Santa Rosa Island. Project planning details and reasoning for design choices are provided in their designated design sections.

1.B DEFINITIONS

1.B.i) Raspberry Pi (RPi)

A cheap credit card sized single board computer. It can run various Linux distributions and can perform computations and communicate with other elements in a network like a normal computer.

1.B.ii) Machine Learning

A method where a system is trained to differentiate between information passed to it. By training with various data you want, it can be used to check future data against what it learned from training.

1.B.iii) Feature Extraction

Retrieving characteristics from data, called features, can be used to discriminate and measure data. Features extracted from some data can be compared to other data to analyze similarities in feature signature.

1.B.iv) Feature Vector (FV)

An object containing numerical values representing various characteristics of something. Different measurements of an audio sample are what our feature vector will be.

1.C SYSTEM FUNCTIONAL OVERVIEW

This project will develop the code and software/hardware configurations for a Raspberry Pi to record audio and process it to alert an outside agent if certain sounds are heard. For recording audio, a Raspberry Pi will be connected to a USB Sound card for recording audio from a 3.5mm microphone. The specific type of sound card and microphone system is still in review. For analyzing recording audio, feature extraction is performed on the RPi to form a feature vector that is shipped via the network to the database. Another program analyzes the FV in the database, and if it is found to have evidence of a recording we want, will use the Meta data in the FV to alert the client of which RPi recorded the audio.

1.D REFERENCES

Hastie, T., Tibshirani, R. and Friedman, J. 2009. *The elements of statistical learning: Data mining, inference, and prediction*, New York, NY: Springer.

McKinney W. 2012. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*, New York: Springer.

2. OVERALL DESCRIPTION

2.A) PRODUCT PERSPECTIVE

This systems place in the world, including background theory and connections to other systems in the CI Rainbow Project along with other comparable solutions.

2.A.i) Place of This System in the World

This system audio recording and classification system will be used to record noised from the environment (like birds) and use feature extraction to identify the animal. There are many devices that use intelligent classification algorithms to recognize voices (iPhone, Galaxy S series, etc.) and these systems are increasingly popular. Our system is similar in classification but is more simply implemented version of this idea, as we only seeks to make a basic classification of the type of animal by analyzing noise in the nearby environment. Various groups like environmental and conservation Agencies, including the National Forest Agency, could benefit from this work.

There is a need for cheap and easily deployable systems that can be used by agencies on a budget to collect information about animal movement. These sound sensors can be deployed across a field and by recoding data about noises of a particular animal nearby, be used to track animal movements. Data about migrations or pack movement behavior could be collected. Perhaps even more significant is the ability to collect accurate data autonomously and without human presence nearby. The data collection technique for the types of sound we want to classify can be tested before deployment to ensure accurate data collection. Persons using this system will less frequently need to check the devices which will reduce the disruption to the environment compared to current in-person collection techniques.

2.A.ii) Background Theory

The process of audio analysis occurs at the point recording where a quality microphone must be used and an appropriate sampling rate chosen. Consider that noises we want to record may be at extreme frequencies that our recording system must be able to record. These noises may be fast or slow depending on the species we are searching for in our audio, so we need an appropriately chosen sample rate. After an appropriate recording has been made, we then perform audio feature extraction.

Feature Extraction from audio is a process by which various properties of some audio are mathematically analyzed. These measurements contain information about our sample, including properties like: average loudness, frequency of occurrences of peaks, and sharpness of audio frequencies over time. By using an appropriate combination of these features extracted from the audio, we can use these features as characteristics for that sample, and thereby

classify a sample based on those features. In short, analysis of features extracted from audio can be used to discriminate between audio that exists in various recordings.

A combination of features extracted can be organized together to form a Feature Vector (FV). Our FV is simply a standardized combination of features measured from our audio sample, for example, the average Loudness and Energy of our recording. The measurements that comprise our FV are carefully selected to best measure the audio we want to classify. For example, if one desired to record an animal that produces long but high frequency chirps, we might choose to include Energy. Choosing appropriate features to extract is vital as measurements that are effective at discriminating two pieces audio may work against the ability of the other effective measurements to classify a sample. After a FV has been constructed, it is then transferred to an analysis and classification system.

Sound analysis and classification is done via a Machine Learning algorithm that has been trained to classify based on the FV's of the audio we want to recognize. To train a Machine Learning Algorithm, one provides data representing the thing you want and data representing what you do not want. In audio analysis, a FV is provided and the measured information in that FV is used to help classify the noise we want. In short, the measured information for multiple recordings of the sound we want to detect can be used to classify that sound when compared to audio that is not the sound we want to collect. Machine Learning algorithms after they have been trained are also fast classifiers, which is significant when we are continuously recording audio.

2.A.iii) Connections To Other Systems

This FDD only convers in detail the purpose of the recording sound operations and the theory of moving the samples to the server. This work then has to conform to other systems in the network like: Communication of FVs to the server and Analyzing those FV's server side to find classify audio.

The goal of this project is to have a modular and well-documented set-up for recording audio and extracting a FV. Therefore, communication and FV contents are dealt with in the following way:

Communication of FV:

If a recording is made and it is deemed to contain information worthy of attempting to classify, then the recording and FV and moved to a Deployment Directory. This is simply a directory in the file system which will need to be polled to check for contents. Anything that exists in this directory still has not been sent to the server, and should be sent as soon as possible.

After a FV and audio sample have been sent to the server, they should be moved to a Cache Directory. The Cache Directory houses all files that have been sent to the server in case they need to be retrieved at a later time. Although there are no plans current plans to implement a system where the server can poll the

sound recorder to gather previously recorded audio, by saving the audio in a cache, this process could be implemented in the future.

Server-side Classification of Audio

This analysis is currently out of the scope of this project, however, it is this projects goal to send the FV and audio from a recording to the server, where it can then be processed. The implementation of the analysis via a Machine Learning process is up to the other developer.

OA&M

Although details of this process are few, there will be skeletal implementations for the ability to poll a device recording audio to either:

1. Let an outside entity confirm the microphone is recording audio
    - Make a recording and perform feature extraction to make sure there are no errors from these processes. Any errors will be sent alone in a performance status text file to the server.
2. Make recording and send it to the server with a special tag.
    - This service is largely for debugging purposes but can be used to ensure the microphone system is working when the devices are deployed in the field by letting the deployer hear the audio they record.
3. Update settings on recording remotely.
    - ssh into the device and kill the recording process. Alter the config file as desired and re-start the recording process.

2.B) PRODUCT FUNCTIONS

High-level description of all functional modules for sound recording.

2.B.i) Audio recording and analysis logic

The Sound recording and communication process is as follows:

1. Record audio clips of size *N* for *M* times. *M* is infinite but can be arbitrary.
2. Spawn a process to perform FV extraction after each time step *M*.
    a. Check if audio contained something other than background noise
        i. YES: Move FV and audio sample to a Deployment Directory
        ii. NO: Leave FV and audio in a workspace directory
3. Check there exists something in the Deployment Directory
    a. YES: Send that FV and audio to the server for classification

2.B.ii) Audio Analysis process logic

1. Retrieve a FV from the server
2. Pass this FV through the Machine Learning trained system

a. Check if FV is consistent with audio containing the sound we want
    i. YES: Move this FV and audio file to a safe place on server and alert the website of activity
    ii. NO: remove this FV and audio from the server

2.B.iii) Cleanup

Audio recordings can take up a lot of space on our memory. To keep this solution cheap, we are employing low sized flash memory between 8-16 Gigabytes. Of this 8-16Gigs, we are only going to be consuming at most 2 Gigs of it. This artificial limitation of 2Gigs is to ensure we do not consume too many resources for other projects and sets a limitation for when we must perform cleanup of our audio recordings and FV generation.

Any audio files that only contain background noise are immediately moved to a Cache Directory where they sit until they are cleaned up after some length of time or are cleaned up because the directory has become too large. Audio that is deleted is done so in reverse chronological order. The current hard-limit size of this Cache Directory is 1 Gig.

Any audio files that contain information that is not just background information is moved from the Deploy directory to another separate Cache Directory. This cache directory follows the same rules as the directory for files containing background noise, including the hard-limit size of 1Gig.

The motivation for this structure is that it is easy to test code changes to have results of the recordings and feature extractions saved for a time and it also means we only must periodically cleanup our folders and so do not need to be constantly deleting files as we create them. We can therefore create and schedule scripts that only need to be run periodically to clean our workspace along the way.

2.C) USER CHARACTERISTICS

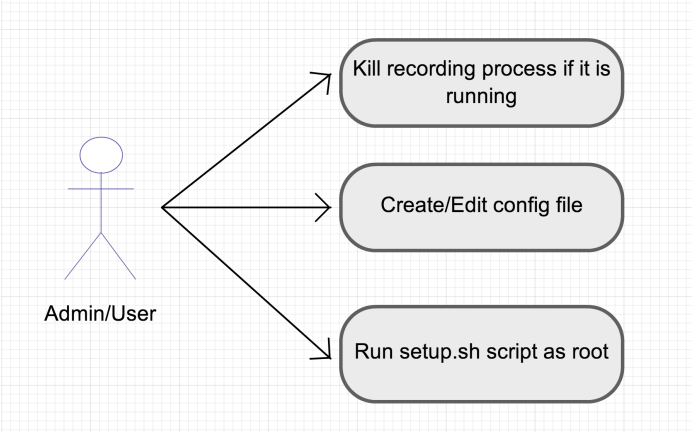
The following are various user operations explaining functionality of system.

2.C.i) Setup & Installation

The sound recording and analysis executable is installed via a bash script called *setup.sh*. The setup script does the following:

- Ensures necessary software is installed on the device (Please refer to Specific Requirements) and compiles the ‘recording and analyzing’ C language file that controls the recordings and FV extraction.
- Ensures a *config* file exists in the local directory (or $HOME directory called “cirainbow.conf”) and warns user if one is not found. The final program will not run unless a *config* file is found.

- Places an executable bash script to the local directory of the install script and copies this script to ~/bin. It is this script that calls the actual recording and FV extracting binary. In Short, the user calls this script to run the program instead of running the binary directly.



2.C.ii) Edit Recording Setting

To change recoding settings like record duration and number of recordings, change the integer numbers in the *config* file of the following settings:

- The setting for recording duration is "*recordingduration*:" followed by an integer for the seconds of recording length.
- The setting for recording number is "*recordingnumber*:" followed by an integer for the num of recordings to make, each of length *recordingduration*.
- The setting for a recording's sample rate is "*samplerate*": followed by an integer for the sampling rate.

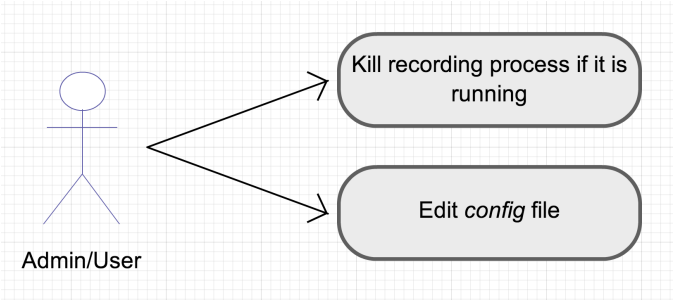
An example *config* file may look as follows:

```
[NODE_INFO]
latitude = 17.2343432
longitude = -119.23423423
raspberrypiid = 14

[SOUNDS]
background = off
analysis = on
filter = off
recordingextention = .wav
recordingduration = 3
recordingnumber = 2
recordingprefix = rec_
samplerate = 44100
datalocation = /home/pi/data/
outputform = YAAFE
```

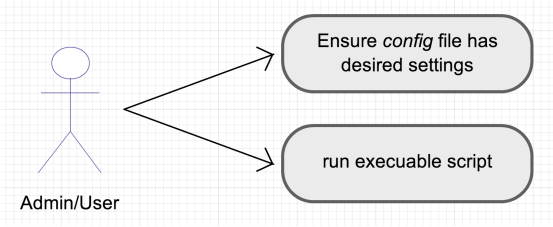
From the above *config* file, we will generate two recordings of duration 3 seconds at a sample rate of 44100Hz will be made. Each recording will be saved as a ".wav" file with the prefix "rec_" and extention ".wav". Filtering is off, so extracted audio data from Yaafe, directed by the local 'featureplan' file, and recordings are saved to the directory called "/home/pi/data/". All output is sent to stdout because background is "off".

Once desired changes have been made to the *config* file, the program is ready to run again. There is no need to run the setup script again.



2.C.iii) Recording and FV Extraction

After setup script has been run, ensure that the *config* file contains the desired recording settings. Ensure also that a feature plan file pat has been provided either in the *config* file or in the local directory of the recording binary. Assuming previous setup was successful and a *featureplan* file is provided, simply run the recording and FV extraction script. This script preps the environment for recording and analysis then runs the binary that actually performs the recording and analysis. Any settings needed by the executable at the start of execution from the *config* file.



2.C.iiii) *Config* settings

A valid config file must be in the following format:

[NODE_INFO]

...

Optional Node info, line by line

...

[SOUNDS]
...
sound config settings, line by line
...

The [NODE_INFO] section is the home for details like a raspberry pi id or latitude and longitude location. The [SOUNDS] section is the home for details about the run environment for the next execution of the sound binary. Whatever details are in the [SOUNDS] section will dictate how the sounds script will run.

The following are supported *config* settings are as follows:

Key	**Value**
recordingduration	Duration of each recoding in seconds
recordingnumber	Number of recordings to make
samplerate	Sample rate in Hz of each recording
recordingextention	Recording extension. DEFAULT is ".wav"
outputform	Either output is plan Yaafe using "YAAFE" or output is a json file using "FV". DEFAULT: string "YAAFE"
saverecording	(OPTIONAL) Save audio that is recorded DEAFULT: string "on" - recordings are saved
analysis	(OPTIONAL) Turn audio analysis (audio feature extraction via Yaafe) off. DEAFULT: string "off" - filtering is performed
background	(OPTIONAL) Turn raraa into a Daemon with "on" DEAFULT: string "off" - prints to terminal
filter	(OPTIONAL) Turn filtering on/off DEAFULT: string "on" - filtering is performed
recordingprefix	(OPTIONAL) Recording filename prefix. DEFAULT: 'rec_' at start of each audio file
featureplanpath	(OPTIONAL) Path to a feature plan file DEFAULT: local sound directory
featureplanname	(OPTIONAL) Name of a feature plan DEFAULT: string "featureplan"
filterplanpath	(OPTIONAL) Path to a filter plan file DEFAULT: local sound directory
filterplanname	(OPTIONAL) Name of a filter plan file DEFAULT: string "filterplan"
recordinglocation	(OPTIONAL) Directory path where recordings are saved. Analysis still in 'analysis' dir DEFAULT: local 'analysis' dir
datalocation	(OPTIONAL) Directory where data finished being recorded and analyzed goes.

	DEFAULT: local 'data' directory
analysislocation	(OPTIONAL) Work space for audio recording analysis and filtering. DEFAULT: local "analysis" directory
latitude	(OPTIONAL) Latitude loc, DEFAULT is zero DEAFULT: string "0.0000 W"
longitude	(OPTIONAL) Longitude location DEAFULT: string "0.0000 N"
rasberrypiid	(OPTIONAL) Custom string rasberry pi id DEFAULT: string "-1"
simulate	(OPTIONAL) flag to turn raraa into sim mode where it copies simulated output files from 'simulateiondirectory' to 'data' directory DEFAULT = string "off"
simulationdirectory	(OPTIONAL) Directory where sim output files are found. Copied to "data" directory if simulateion is "on". DEFAULT: durectory "/test/data_yaafe/"
debug	(OPTIONAL) Turn on/off debug statements

2.D) CONSTRAINTS

Constraints on the system as implemented.

2.D.i) Size Limits

An artificial limit of 2Gigs is being imposed on the file system where recordings and FV measurements are being held. Approximately 1 Gig is reserved for local caches of recordings which have yet to be analyzed while the other 1Gig is reserved for audio and FV measurements that have yet to, and have already, been sent to the server. The file system will be cleaned out after some time duration or if the imposed 2Gig directory size is being reached.

2.D.ii) Memory Degradation

Memory degradation is not a concern even though recordings and FV extraction is being may be made continuously. Recordings are made at a rate of roughly 180Kb/s. With a size limit of 1 Gig of cache space, then it would take ((1024Kb/Bb*1024Kb/Gb*2Gb)/180Kb/s)/60s/m=97mins to fill the cache space. Assuming a flash memory life cycle of 100,000 writes, it would take ((97m/write*100000write)/60min/h*24h/d)=6742.3days to fully corrupt our 2 Gig space. Our flash sticks will be >= 8Gigs in size; so the life span is years even with the high rate of write consumption.

2.D.iii) Recording Speed vs. FV Extraction Time

Feature extractions from recordings must be performed faster than recordings are made. If they were not, a backlog would form where the processor was needed to perform FV extractions while also recording and dealing with other tasks on the device. Initial testing shows that recordings should be AT LEAST 5 seconds in length (10 seconds recommended). Feature extraction becomes slower as more features are added to the *featureplan*. It is recommended to keep the number of feature vector contents low to ensure this timing constraint is maintained. For details on performance, see Section 3.C.

2.D.iv) Arecord and Yaafe Processing and Memory Usage

Arecord software for recording consumes about 2.3% of the CPU on the RPi while *Yaafe* consumes anywhere from 50-98.2% (intermittently) of the CPU and 3.5% of memory while extracting features, depending on the features.

2.E) ASSUMPTIONS AND DEPENDENCIES

Assumptions made in regards to recording and feature extraction, hardware and software, etc.

2.E.i) Feature Vector Is Arbitrarily Chosen

There is no particular insight put to what features are being extracted at this time. It is up to the user to select what features to extract by creating a *featureplan* file of their own and pointing the recording script to it by updating the *config* file. Information about *featureplan* files can be found in *Yaafe's* documentation here 2.3.2 here: (http://yaafe.sourceforge.net/manual/quickstart.html#feature-definition-format).

2.E.ii) Assume Yaafe Returns Correct Results

It is assumed that *Yaafe* is returning the correct values. It is only currently the scope of this project to make get the contents of whatever features are extracted for a recording's FV to make it to the server for analysis. Information about the quality of *Yaafe* can be found here: (http://biblio.telecom-paristech.fr/cgi-bin/download.cgi?id=10395).

2.F) APPORTIONING OF REQUIREMENTS

As needed – under advisement as progress continues.

3) SPECIFIC REQUIREMENTS

3.A) EXTERNAL INTERFACE REQUIREMENTS

Descriptions of all external interfaces include protocols.

3.A.i) RasberryPi (RPi) ModelB-Rev 1

A small computer where all hardware and software will be installed for recording and analyzing sound. This device will run the *Rasbian* UNIX operating system (see bellow). It is optimal for our implementation because it is low cost, lightweight, non-intrusive form-factor, and low power consuming.

See (http://www.raspberrypi.org/help/faqs/).

3.A.ii) Rasbian

UNIX operating system tat can be placed on the RasberryPi. It provides a fully functional linux terminal environment to code, test, and run the necessary software. *Arecord* for recoding and *Yaafe* for FV extraction can be installed here.

See (http://www.raspbian.org/).

3.A.iii) Arecord

Software for recording audio that can be used on the RPi and run as a terminal based application. It is installed on the *Rasbian* distribution natively.

See (http://linux.die.net/man/1/arecord).

3.A.iv) Yaafe

Software for measuring characteristics in audio that we organize into a feature vector and send to the server for later analysis. It is not installed natively on *Rasbian* and requires Python. It can be run as a terminal based application or integrated into another executable.

See (http://yaafe.sourceforge.net/).

3.A.v) Python

Python is a very readable programming language that the implementation of Yaafe being used requires. No code for this project was developed in Python but Yaafe needs it, so it is a requirement.

See (https://docs.python.org/3/).

3.A.vi) USB Sound Card

A USB sound card is hardware piece needed for recording audio. A sound card is necessary because the RPi does not have an on board sound card. Almost any USB sound card should do and its set-up is trivial because it will (should?) be the only sound card on the system.

We are using a “Syba SD-CM-UAUD USB Stereo Audio Adapter, S-Media Chipset”. Consider the following tutorial (http://www.g7smy.co.uk/?p=283).

See (http://www.amazon.com/Syba-SD-CM-UAUD-Adapter-C-Media-Chipset/dp/B001MSS6CS/ref=pd_sxp_grid_pt_0_1).

3.A.vii) 3.5mm Microphone

A microphone is required for recording the audio to the RPi. Almost any audio input should work, so long as it goes through the USB Sound Card. We are testing with a cheap microphone called “Neewer 3.5mm Hands Free Computer Clip on Mini Lapel Microphone” but a higher quality microphone will be employed in the future.

See (http://www.amazon.com/Neewer-3-5mm-Hands-Computer-Microphone/dp/B005DJOI8I/ref=sr_1_5?ie=UTF8&qid=1398566240&sr=8-5&keywords=lapel+microphone).

3.A.viii) Ethernet/WiFi

Internet or local network access is necessary for transferring audio recordings and Feature Vectors. Testing is being done with an Ethernet cable connected an RPi over a network where another tester ssh’s into the RPi to code. The device as deployed on the island will use WiFi with a USB like the following “Edimax”.

See (http://www.amazon.com/Edimax-EW-7811Un-Wireless-Adapter-Wizard/dp/B003MTTJOY/ref=sr_1_1?ie=UTF8&qid=1398566452&sr=8-1&keywords=USB+Wifi).

3.B) FUNCTIONAL REQUIREMENTS – **IN DEV**

Detailed description of every function, as they are implemented. Current implementation is only is proof-of-concept phase but the following methods are used. SEQUENCE DIAGRAMS NEEDED FOR EACH FUNCTION.

Building Recording System Call

Func Sig: *String makeRecordCmd(int recDur, int sampRate, String fName)*

Builds a system call for making a recording based on the date and time, length of recording, sampling rate, etc. This system call to *Arecord* is returned as a string and run as System() from the sound recording executable.

…

50+ word per function, This will eventually be the longest part of doc.

3.C) PERFORMANCE REQUIREMENTS

Detailed performance requirements including efficiency, memory, response time, etc. of various functionality.

3.C.i) Power Consumption – IN DEV

No testing has been performed on the power consumption of this system while recording and extracting FV's continuously. This testing is necessary because these devices will be deployed in an isolated environment where they are not readily accessible for battery replacement or re-charging. Consuming more energy than we generate during the day if we were using solar panels either means we would need to include functionality for not recording audio during the night as to not deplete our power supply.

3.C.ii) Sending Data to Server – IN DEV

Data to be sent to the server is placed in a "Deploy Directory" where it sits and waits to be sent. Some other entity sends data to the server. It is not the job of the sound script to send data to the server.

A Deployment directory can be specified in the *config* file but by default is the *data/* directory in the local sound directory.The Format of data to be sent can be in two ways. If the *config* file specifies *outputform = YAAFE* then the recording, whatever YAAFE features extracted, and a meta data file are moved to the data deployment directory. If the *config* file specifies *outputform = FV* then the recording and whatever YAAFE features extracted, wrapped in a JSON format, are moved to the data deployment directory.

3.C.ii) Feature Extraction Time, Memory, & CPU Requirements

Initial testing of running feature extraction of a 10 second clip of noisy audio with the features added as seen in the results in Figure 1. In summary, when extracting features the CPU is around 98% used for the duration of extraction, with only a small ramp-up time, and 3.5% of memory is used.

FIG 1. YAAFE		USEAGE			
Num. Features	AVAILBLE FEATURS	Time Dur.	Time Inc.	CPU %	MEM %
1	am: AmplitudeModulation	6.8	6.8	98.2	3.3
2	ac: AutoCorrelation	7.52	0.72	98.5	3.5
3	cdod: ComplexDomainOnsetDetection	9.14	1.62	98.2	3.4
4	e: Energy	9.3	0.16	98.2	3.5
5	env: Envelope	9.18	-0.12	98.2	3.6
6	envss: EnvelopeShapeStatistics	9.23	0.05	98.1	3.5
7	lpc: LPC	9.35	0.12	9.8	3.5
8	ld: Loudness	10.52	1.17	98.5	3.7
9	mfcc: MFCC	10.79	0.27	98.4	3.5
10	ms: MagnitudeSpectrum	15.6	4.81	9.8	3.5
11	mls: MelSpectrum	15.7	0.1	98.4	3.5
12	obsi: OBSI	16.2	0.5	98.2	3.5
13	obsir: OBSIR	16.3	0.1	98.4	3.7
14	psh: PerceptualSharpness	16.45	0.15	98.3	3.5
15	psp: PerceptualSpread	16.6	0.15	98.3	3.7
16	scfrb: SpectralCrestFactorPerBand	16.7	0.1	98.4	3.5
17	sd: SpectralDecrease	16.82	0.12	98.2	3.6
18	sf: SpectralFlatness	17.23	0.41	98.7	3.7

19	sfpb: SpectralFlatnessPerBand	17.65	0.42	98.1	3.6
20	spf: SpectralFlux	17.9	0.25	98.4	3.8
21	spr: SpectralRolloff	17.93	0.03	98.5	3.6
22	sss: SpectralShapeStatistics	18.1	0.17	98	3.8
23	sslop: SpectralSlope	18.15	0.05	98.2	3.8
24	sv: SpectralVariation	18.38	0.23	98	3.6
25	tss: TemporalShapeStatistics	18.55	0.17	98.4	3.5
26	zcr: ZCR	18.4	-0.15	98	3.4
27	acpi: AutoCorrelationPeaksIntegrator	21.15	2.75	98.1	3.5
28	der: Derivate	50.8	29.65	98.2	3.6
29	hi: HistogramIntegrator	54.12	3.32	98.1	3.5
30	si: SlopeIntegrator	55.5	1.38	98.2	3.6
31	stati: StatisticalIntegrator	56.5	1	98.2	3.6
	AVERAGES	**± 0.1**		**± 0.6**	**± 0.2**

3.D) DESIGN CONSTRAINTS

Guidelines for the software designer with justification including compliance to standards, motivation for using certain processes, employing particular constrains, etc.

3.D.i) Choice for Working Space Limit of 2Gigs

This limit is imposed to make sure enough space is left for other processes and software on the device. It can be changed at a future date as what exact processes will run on each device is solidified. It is assumed this sound recording system will never be permitted more than 2Gigs of workspace because we only need to clean up after ourselves after recording.

3.D.ii) Use of 3.5mm mic and USB Sound Card

The cheap microphone wea re using is for testing purposes and a superior one should be chosen in the future. Using this mic also gives us a worse case scenario to test and conclude the efficacy of this system. A USB Sound Card is necessary because the RPi does not have an onboard sound card, which is needed for recording.

3.D.iii) Why Arecord and Yaafe for Recording & Feature Extraction

Arecord was chosen because it can record audio, is easy to set-up and run, and is installed natively on *Rasbian*. *Yaafe* was chosen to extract features from audio because it provides fairly extensive documentation while many other services had little to no documentation.

See (http://yaafe.sourceforge.net/manual.html).

3.E) LOGICAL DATABASE REQUIREMENTS

Database requirements for sound transfer.

The exact form that FV's will take in the database is the domain of the database handler. The sound script simply provides an audio recording and a FV formatted YAAFE features extracted from that audio. Hence, everything besides the audio sent to the server can be treated as how JSON is normally formatted.

3.F) SOFTWARE SYSTEM ATTRIBUTES

Guidelines for reliability, availability, security, maintenance, and portability.

3.F.i) Updating recording settings files remotely

To update recording settings: Simply *kill* the currently recording process (if it is running), replacing the *config* file with desired changes, and start the *start* script. See Section 2.C.ii) Edit Recording Setting for more details.

3.F.ii) Background noise removal reliability (filtering)

A generic system will be in place for using some feature vector component to discriminate our certain audio recordings. For example, if there is only "background" noise in an audio sample, this may manifest itself as a low Loudness rating. This real number measurement of the Loudness could then be used to only allow data to be sent to the server that contains potentially worthwhile information for classification analysis.

The infrastructure for this system is called "filtering". There is a filtering header file in the sound directory where one can create their own filters to analyze the features extracted from *Yaafe* to filter our audio files that may not be interesting to them. How reliable a particular filter is will be determined by the choice of filter implementation.

3.F.iii) Security and Data Integrity

Although there are no plans for tracking data integrity, the device will need to ensure connection to whomever sends the FV to the server is maintained. We have a backlog local storage of 1Gig for FV and audio that we may want to send to the backend but once that is filled, the oldest will be overwritten each time a new FV and audio are recorded.

There are no plans for Security of data because the agent who sends the FV and audio to the server handles this.

3.F.iv) Recording and FV Extraction Portability

The setup/installation and recording of audio and FV extraction should be able to be done on any UNIX based OS that permits sudo privileges for installation and running of executable bash scripts.

3.G) ASSET REQUIREMENTS

System required assets like sounds, graphics, music, models, etc.

This system relies on no other assets beyond the hardware and software noted in Section 3.A External Interface Requirements.

3.H) OTHER REQUIREMENTS

As needed.
Length: 50+ words for each requirement.