Module Interface Specification for Audio360

Team #6, Six Sense
Omar Alam
Sathurshan Arulmohan
Nirmal Chaudhari
Kalp Shah
Jay Sharma

November 14, 2025

1 Revision History

Date	Version	Notes
2025-11-13	1.0	Initial write-up

2 Symbols, Abbreviations and Acronyms

symbol	description
Audio360	360 Audio analysis system on smart glasses
DOA	Direction of Arrival
FR	Functional Requirement
FFT	Fast Fourier Transform
ICA	Independent Component Analysis
M	Module
MG	Module Guide
MIS	Module Interface Specification
NFR	Non-functional Requirement
R	Requirement
SPI	Serial Peripheral Interface
SRS	Software Requirements Specification
USART	Universal Synchronous/Asynchronous Receiver/Transmitter

Table 1: Symbols, abbreviations and acronyms used in the MIS document.

See SRS Documentation at Symbols, Abbreviations, and Acronyms for a complete table used in Audio360.

Contents

3 Introduction

The following document details the Module Interface Specifications for Audio360, an assistive device system for smart glasses that provides real-time visual indications of sound source locations and classifications to aid individuals who are deaf or hard of hearing.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/Team6-SixSense/audio360.

4 Notation

The structure of the MIS for modules comes from [1], with the addition that template modules have been adapted from [2]. The mathematical notation comes from Chapter 3 of [1]. For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Audio360.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
array/list	x[]	Ordered list of type x.
negate	$\neg$	Logical math NOT.
and	$\wedge$	Logical math AND
or	$\vee$	Logical math OR
implies	$\implies$	Logical math implies
assignment	:=	For A := B. B is assigned to A.
for all	$\forall$	Referencing all items. For example: $(\forall\ variable \mid condition : statement)$

The specification of Audio360 uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Audio360 uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	Microphone Input Module USART Communication Module SD Card Module SD Card Interface Module
Behaviour-Hiding Module	Audio360 Enginer Module Visualization Module
Software Decision Module	Audio Generation Module DOA Processor Module Audio Sampling Module Audio Spectral Leakage Prevention Module Audio Normalizer Module Audio Anomaly Detection Module Fast Fourier Transform Module Logging Module Fault Manager Module Mel Filter Module Discret Cosine Transform Module Principle Component Analysis Module Linear Discriminant Analysis Module Classification Module Independent Component Analysis

Table 2: Module Hierarchy

6 Data Types

6.1 Generics

6.1.1 bool

Logical 0 or 1.

6.1.2 int16

16 bits signed integer ($\mathbb{Z}$).

6.1.3 int32

32 bits signed integer ($\mathbb{Z}$).

6.1.4 int64

64 bits signed integer ($\mathbb{Z}$).

6.1.5 uint16

16 bits unsigned integer ($\mathbb{N}$).

6.1.6 uint32

32 bits unsigned integer ($\mathbb{N}$).

6.1.7 uint64

64 bits unsigned integer ($\mathbb{N}$).

6.1.8 float32

32 bits floating point ($\mathbb{R}$).

6.1.9 float64

64 bits floating point ($\mathbb{R}$).

6.1.10 string

Character stream.

6.2 Enums

6.2.1 Audio360State

1. AudioClassificationProcess: State when audio classification is running.
2. DirectionalAnalysisProcess: State when directional analysis is running.
3. OutputProcess: State when output processing is running.

6.2.2 Audio360Status

1. Uninitialized: Audio360 Engine is not initialized.
2. Initialized: Audio360 Engine is initialized, but not ready.
3. Ready: Audio360 Engine ready for requests.
4. Running: Audio360 Engine is running. Can not accept new requests.
5. Error: Audio360 Engine is stuck at an unhandled error.

subsubsectionAudio360Status

1. NoFault: No fault state.
2. MicrophoneXFault: Fault with microphone X.
3. AudioClassificationFault: Fault with audio classification.
4. directionalAnalsysiFault: Fault with the directional analysis.

6.3 Data Structures

6.3.1 FrequencyDomain

1. N [uint16]: The number of data points.
2. frequency [float32[]]: The frequency represented in Hz.
3. real [float32[]]: The real component of thre frequency contribution.
4. img [float32[]]: The imaginary component of thre frequency contribution.
5. magnitude [float32[]]: The magnitude of the frequency component.

7 MIS of [Module Name —SS]

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

7.1 Module

[Short name for the module —SS]

7.2 Uses

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

7.4 Semantics

7.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

7.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

7.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

7.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

7.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

8 MIS of Audio Generation Module - M??

8.1 Module

This module generates audio data by simulating room acoustics and generating synthetic microphone array data from a given audio source and position. This includes room response calculations and spatial audio propagation models.

8.2 Uses

1. Python Standard Libraries

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
generateAudio	audioSource: float32[], position: float32[], outputFile: string	micrphoneData[][]: [][float32]	AudioGenerationFailure

8.4 Semantics

8.4.1 State Variables

None.

8.4.2 Environment Variables

None.

8.4.3 Assumptions

The pyroomacoustics module is assumed to be working correctly (generating realistic and reliable audio data based on configuration parameters).

8.4.4 Access Routine Semantics

generateAudio():

- transition: None
- output: microphoneData
- exception: AudioGenerationFailure

8.4.5 Local Functions

- simulateRoom(room: pra.ShoeBox) → bool

9 MIS of Direction of Arrival (DOA) Processor Module - M??

9.1 Module

This module processes audio data to estimate the direction of arrival of a sound source. This includes frequency domain analysis, signal processing, and direction estimation algorithms.

9.2 Uses

1. Audio Generation Module
2. Audio Normalizer Module
3. Audio Spectral Leakage Prevention Module
4. Audio Anomaly Detection Module

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
calculateDirection	audioData[][]: [][float32]	direction: float32	AudioProcessingFailure

9.4 Semantics

9.4.1 State Variables

None.

9.4.2 Environment Variables

None.

9.4.3 Assumptions

None.

9.4.4 Access Routine Semantics

calculateDirection():

- transition: None
- output: direction
- exception: AudioProcessingFailure

9.4.5 Local Functions

None.

10 MIS of Audio360 Engine - M??

10.1 Module

Orchestrates the overall audio processing by receiving raw input data and managing module communication.

10.2 Uses

1. Classification Module
2. DOA Processor Module
3. Fault Manager Module
4. Logging Module

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
runProgram	-	-	ProgramStartFailure, ProgramRunTimeFailure

10.4 Semantics

10.4.1 State Variables

- state [Audio360State]: Determines the current state of the the Audio360 engine. Each state will run a specific module in used modules.

10.4.2 Environment Variables

- status [Audio360Status]: Status of the module. This encapsulates initialization, running, ready, or errored.

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

run():

- transition: The state machine in figure 1 outlines the transition of this module.

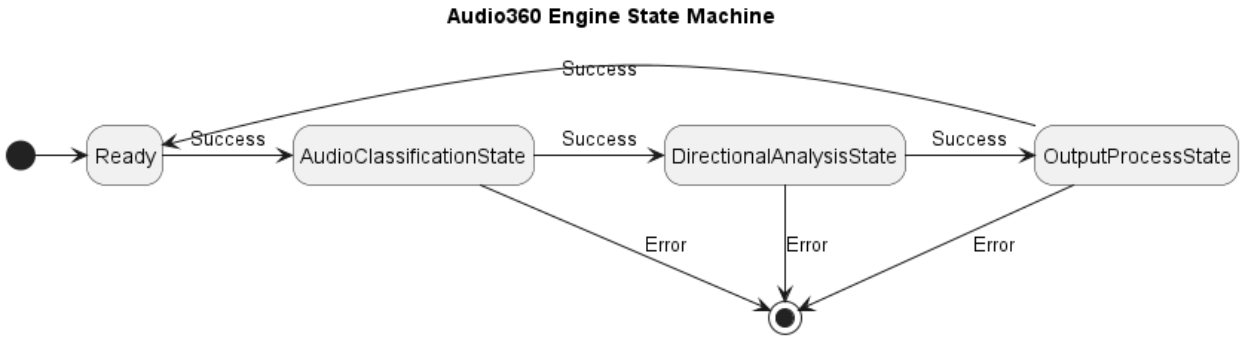

Figure 1: Internal state machine of Audio360 Engine module.

- output: None
- exception: ProgramStartFailure, ProgramRunTimeFailure

10.4.5 Local Functions

None

11 MIS of Audio Sampling Module - M??

11.1 Module

Provides sampling of audio data from a given source while preserving the order of individual samples.

11.2 Uses

- Microphone Input Module
- Microphone Diagnostic Module

11.3 Syntax

11.3.1 Exported Constants

- sampleWindowSize [uint16]: The number of samples in a window. The value is a power of 2.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
sample	inputSource	-	InputSouceError
getSampledData	-	sampledData	-

11.4 Semantics

11.4.1 State Variables

- sampledData [uint32[]]: A cyclic array of size sampleWindowSize that stored the sampled data.
- windowPosition [uint32]: The current reference index position of the array storing sampled data. This denotes the starting point of the cyclic array.

11.4.2 Environment Variables

None

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

sample():

- transition:
  1. $sampleData[windowPosition] := inputSource.newData$
  2. $((windowPosition + 1) \geq sampleWindowSize) \implies windowPosition := 0 \ \vee$
     $\neg((windowPosition + 1) \geq sampleWindowSize) \implies$
     $windowPosition := windowPosition + 1$
- output: None
- exception: InputSouceError

getSampledData():

- transition: None
- output: sampleData
- exception: None

11.4.5 Local Functions

None

12 MIS of Audio Spectral Leakage Prevention Module - M??

12.1 Module

Applying windowing on audio signal to reduce spectral leakage before frequency domain processing.

12.2 Uses

None

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
applyFilter	sampleWindow	filteredWindow	-

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

None

12.4.3 Assumptions

None

12.4.4 Access Routine Semantics

applyFilter():

- transition: None
- output:

1. $(\forall\ i \mid 0 \leq i \leq N-1 :\ filteredWindow[i] := sampleWindow[i] * \sin(\frac{\pi * i}{N-1})^2)$ such that $N =$ size of sampleWindow.

- exception: None

12.4.5 Local Functions

None

13 MIS of Audio Normalizer Module - M??

13.1 Module

Processes audio signals to maintain consistent amplitude across different sources.

13.2 Uses

None

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
normalize	sampleWindow, source	filteredWindow	-

13.4 Semantics

13.4.1 State Variables

None

13.4.2 Environment Variables

None

13.4.3 Assumptions

Normalization factor is pre-deteremined and known for all given sources.

13.4.4 Access Routine Semantics

normalize():

- transition: None
- output:

  1. $(\forall\ i \mid 0 \leq i \leq N-1 :\ filteredWindow[i] := \dfrac{sampleWindow[i]}{F_{source}})$
     such that N = size of sampleWindow and F_{source} is normalization factor of a the sampleWindow source.

- exception: None

13.4.5 Local Functions

None

14 MIS of Fast Fourier Transform - M??

14.1 Module

Computes the discrete Fourier transform of the input signal to obtain its frequency domain

14.2 Uses

- Audio Spectral Leakage Prevention Module

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
signalToFrequency	signal	frequency	-

14.4 Semantics

14.4.1 State Variables

None

14.4.2 Environment Variables

- inputsize [uint32]: The size of the signal input, as in number of samples. This is required to interface with hardware acceleration methods on the microcontroller.

14.4.3 Assumptions

None

14.4.4 Access Routine Semantics

signalToFrequency():

- transition: None
- output:
  1. $(\forall\ k \mid 0 \leq k \leq N-1 :\ frequency[k] := \sum_{n=0}^{N-1} x_n * e^{-2i\pi n \frac{k}{N}})$ such that N = size of input signal.
- exception: None

14.4.5 Local Functions

- createOutput(): returns a FrequencyDomain. This function extract features from the FFT and stores it in the data structure.

15 MIS of Logging Module - M??

15.1 Module

Provides centralized logging of data streams to designated output destinations for debugging and monitoring.

15.2 Uses

- USART Communication Module
- SD Card Interface Module

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
log	text	-	-

15.4 Semantics

15.4.1 State Variables

None

15.4.2 Environment Variables

None

15.4.3 Assumptions

None

15.4.4 Access Routine Semantics

log():

- transition: None
- output: input text is logged to output source (SD card or USART port)
- exception: None

15.4.5 Local Functions

None

16 MIS of Fault Manager Module - M??

16.1 Module

Monitors system health and manages critical faults to maintain core functionality.

16.2 Uses

1. Microphone Diagnostic Module
2. Classification Module
3. DOA Processor Module
4. Audio Anomaly Detection Module

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
runFaultAnalysis	-	-	-

16.4 Semantics

16.4.1 State Variables

- faultState [FaultState]: The current fault state of the overall software system.

16.4.2 Environment Variables

- faultState: This variable is shared with the display, to notifiy user of the internal status of the software system.

16.4.3 Assumptions

Fault Manager module itself does not run into any critical faults.

16.4.4 Access Routine Semantics

faultState():

- transition: update the value of faultState depending on the overal health of the software system.
- output: None
- exception: None

16.4.5 Local Functions

None

17 MIS of Mel Filter Module - M??

17.1 Module

Converts a Spectrogram into the equivalent Mel-Spectrogram by applying a bank of n mel-scaled triangular filters to each frequency frame.

17.2 Uses

None

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
applyMelFilterBank	spectrogram: 2D real array	mel-spectrogram: 2D real array	invalidSpectrogramDimension

17.4 Semantics

17.4.1 State Variables

None

17.4.2 Environment Variables

None

17.4.3 Assumptions

- The input spectrogram is the output of the FFT Module across a constant buffer time. This is arranged as a 2D matrix where the rows represent time frames and columns represent frequency bins. This matrix can be represented as $S(t, k)$ where k is the number of frequency bins, and t is the number of time frames.
- The mel filterbank matrix is precomputed offline and stored as a read only constant within the system. It is computed offline using pre-determined information like Sampling Rate, FFT Size, maximum and minimum frequencies and number of mel filters. This matrix can be represented as $H(m, k)$, where m is the number of mel-filters, k is the number of frequency bins and $H(m, k)$ is a weight that tells you how much frequency bin k contributes to mel band m.

17.4.4 Access Routine Semantics

applyMelFilterBank():

- **transition:** None (function doesn't have states)
- **output:** Returns the equivalent mel-spectrogram, where each time frame is converted from linear-frequency bin representation to mel-scaled frequency bins. This is done by:
  1. Multiplying each spectrogram frame by the mel filterbank matrix.
  2. Summing weighted energy contributions per mel filter.

$$E(t, m) = \sum_{k} S(t, k) \cdot H(m, k)$$

- **exception:** Invalid spectrogram dimension. If the dimension of the input spectrogram is not $S(t, k)$ and perhaps something like $S(t, p)$ where $p \neq k$, then the matrix multiplication would throw an invalid dimension error, which must be exposed from the module as invalid spectrogram dimension as well. This would only happens if the number of frequency bins in the computted FFT is not equal to the number of frequency bins in the pre-computed mel filterbank matrix.

17.4.5 Local Functions

None

18 MIS of Discrete Cosine Transform Module - M??

18.1 Module

Converts a mel-spectrogram into a 2D array containing the sequence of Mel Frequency Cepstral Coefficients (MFCC) vectors by applying the Discrete Cosine Transform (DCT) to each time frame. This reduces overlap between frequency bands that carry similar information and ensures each coefficient is uncorrelated to the other one. Pre-processing step that makes it easier to perform Principle Component Analysis later on.

18.2 Uses

None

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
applyDCT	mel-spectrogram: 2D real array	mfcc: 2D real array	invalidDimensions

18.4 Semantics

18.4.1 State Variables

None

18.4.2 Environment Variables

None

18.4.3 Assumptions

- The input mel-spectrogram is the output of the Mel Filter Module. This input will be of size $E(t, m)$, where t is the number of time frames, and m is the number of mel-frequency bins.
- The DCT transform matrix $D(c, m)$ will be computed offline, where c is the desired number of MFCC coefficients for each time frame, and m is the number of mel-frequency bands. $D(c, m)$ represents the unique cosine wave that is associated with the

c^{th} coefficient and m^{th} band. A combination of these waves will be used to uniquely represent the spectrogram at time frame t.

$$D(c, m) = \cos\Big(\frac{\pi c}{M}(m - 0.5)\Big)$$

- Only 13 MFCC coefficients will be calculated for each time frame t.

18.4.4 Access Routine Semantics

applyDCT():

- **transition:** None (function doesn't have states)
- **output:** Returns a matrix *mfcc(t,c)*, such that for each time frame t and each MFCC coefficient index c, the value of the matrix at $mfcc(t, c)$ can be computed as

$$mfcc(t, c) = \sum_{m=1}^{M} E(t, m) \cdot DCT(c, m)$$

  Where $E(t, m)$ represents the mel-spectrogram at time frame t and mel-frequency m. And $DCT(c, m)$ represents the DCT transform matrix that was computed offline.

- **exception:** Invalid spectrogram dimension. If the dimension of the input spectrogram is not $E(c, m)$ and perhaps something like $E(c, p)$ where $m \neq p$ then the matrix multiplication would throw an invalid dimension error, which must be exposed from the module as invalid spectrogram dimension as well. This would only happens if for some reason the number of mel-frequency bands returned by the Mel Filter Module is not the same as the number of mel-frequency bands in the DCT matrix computation.

18.4.5 Local Functions

None

19 MIS of Principle Component Analysis Module - M??

19.1 Module

Principle Component Analysis (PCA) highlights the features with the greatest variance from an input feature matrix. In this case, uses the MFCC feature vector for each time frame to find the direction of greatest variance. This helps with eliminating noise and unimportant information.

19.2 Uses

Non

19.3 Syntax

19.3.1 Exported Constants

None

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
applyPCA	mfcc: 2D real array	pcaFeatures: 2D real array	invalidDimensions

19.4 Semantics

19.4.1 State Variables

None

19.4.2 Environment Variables

None

19.4.3 Assumptions

- The input MFCC matrix is the output of the Discrete Cosine Transform Module. This input will be of size $mfcc(t, c)$, where t is the number of time frames, and c is the number of coefficients.
- The PCA mean vector and PCA projection matrix will be derived offline during the system training and stored as read-only constants. The PCA mean vector will be of size

$(1, c)$, and will be computed as the average MFCC vectors across all training samples.

$$\bar{x} = \frac{1}{T} \sum_{t=1}^{T} x(t)$$

The PCA projection matrix will be of size (K, c), where K is the number of eigenvectors with the maximum variance and c is the number of MFCC coefficients. These eigenvectors are extracted from the following covariance matrix.

$$C = \frac{1}{T} \sum_{t=1}^{T} (x(t) - \bar{x}) (x(t) - \bar{x})^T$$

19.4.4 Access Routine Semantics

applyPCA():

- **transition:** None (function doesn't have states)
- **output:** Returns a matrix *pca(t, K)*, where t represents the number of time frames and K represents the number of PCA components retained (proportional to the number of eignevectors). Essentially, each input mfcc matrix is projected onto the vectors of maximum variance, therefore extracting only the most important features of the MFCC based on training. This projection can be computed using the following matrix multiplication.

  $$\text{pca} = (mfcc - \mathbf{1}\bar{x})\, P^T$$

  Where P represents the projection matrix, $\bar{x}$ represents the PCA mean vector, and $mfcc$ represents the full mfcc matrix.
- **exception:** Invalid mfcc dimension. If the dimension of the input mfcc matrix is not (t, c) and perhaps something like (t, p) where $c \neq p$ then the matrix multiplication would throw an invalid dimension error, which must be exposed from the module as invalid mfcc dimension as well. This would only happens if for some reason the number of coefficients returned by the Discrete Cosine Transform is not the same as the number of coefficients in the projection matrix.

19.4.5 Local Functions

None

20 MIS of Linear Discriminant Analysis Module - M??

20.1 Module

Classifies given feature vector into one of the predefined set of sound classes. This is module is optimized for class seperation. The feature vector in this case is already narrowed down to the features with the most variance using the PCA module.

20.2 Uses

None

20.3 Syntax

20.3.1 Exported Constants

None

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
applyLDA	pcaFeatures: 2D real array	classLabels: 1D real array	invalidDimensions

20.4 Semantics

20.4.1 State Variables

None

20.4.2 Environment Variables

None

20.4.3 Assumptions

- The input pcaFeatures is the output of the Principle Component Analysis Module. This input will be of size (t, K), where t is the number of time frames, and K is number of features selected with the highest variances from PCA.
- The LDA projection matrix, and class weight parameters are computed offline using labelled training data. They are stored as read-only constants in the system. The LDA projection matrix is of size $(K, C-1)$, where C is the number of classes, and K is the number of variance eigenvectors. In essence, this matrix maps the input feature space into the discriminant subspace where class seperation is maximized.

20.4.4 Access Routine Semantics

applyLDA():

- **transition:** None (function doesn't have states)
- **output:**

  Returns a vector of class labels, one per time frame.

  Each feature vector from PCA is projected onto the discriminant space using the LDA projection matrix using the following multiplication.

  $$z(t,:) = pca(t,:) \cdot W_{\text{LDA}}$$

  Where $pca(t,:)$ represents the PCA eigenvectors at time t, and W_{LDA} represents the LDA projection matrix.

  Classification is then performed using linear discriminant functions for each class j, where w_j and b_j is the weight and bias matricies that were pre-computed offline during training for class j.

  $$g_j(t) = w_j^T z(t,:) + b_j$$

  The predicted class for that time frame is then assigned as the class with the maximum value.

  $$\text{classLabel}(t) = \arg\max_j \; g_j(t)$$

  The output of the module is then a vector of length t, representing the best classification for all time frames.

- **exception:** Invalid feature space dimension. If the dimension of the input feature matrix is not (t, K) and perhaps something like (t, p) where $K \neq p$ then the matrix multiplication would throw an invalid dimension error, which must be exposed from the module as invalid feature dimension as well. This would only happens if for some reason the number of eigenvectors returned by the PCA module is not the same as the number of eignvectors in the precomputed LDA matrix.

20.4.5 Local Functions

None

21 MIS of Classification Module - M??

21.1 Module

Orchestrates the overall audio classification flow by receiving signals in the frequency domain and encapsulating the complexity of the various modules involved.

21.2 Uses

1. Audio Sampling Module
2. Audio Normalizer Module
3. Fast Fourier Transform Module
4. Mel Filter Module
5. Discrete Cosine Transform Module
6. Principle Component Analysis Module
7. Linear Discrete Analysis Module
8.

21.3 Syntax

21.3.1 Exported Constants

None

21.3.2 Exported Access Programs

Name	In	Out	Exceptions
classify	spectrogram: 2D real array	-	ClassificationStartFailure, ClassificationRunTimeFailure

21.4 Semantics

21.4.1 State Variables

- state[classificationState]: Stores the current classification. Will continuously be updated at runtime after recieving continuous audio signals.
- state[classificationStatus]: Stores the confidence of the classification that was done. Values for status include high, medium or low confidence.

21.4.2 Environment Variables

- state[status]: Encapsulates initialization, running, ready or errored status of the module.

21.4.3 Assumptions

None.

21.4.4 Access Routine Semantics

applyLDA():

- **transition:** State machine below outlines the transition of this module.

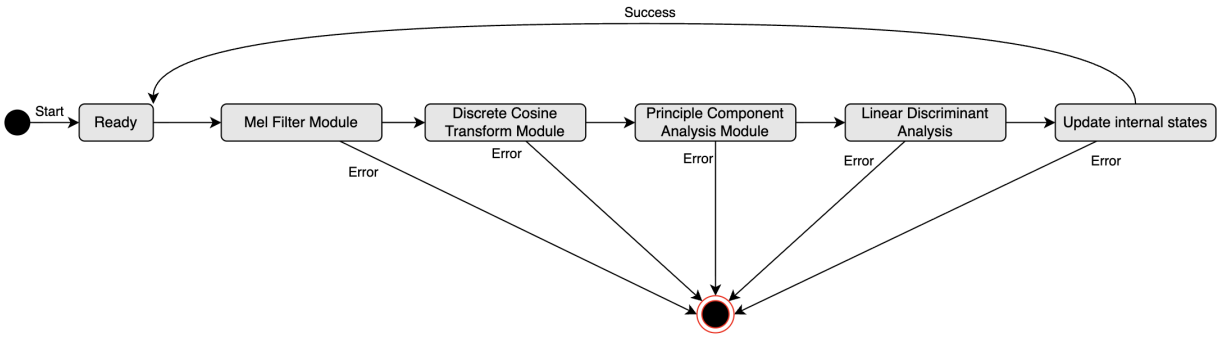

Figure 2: Internal state machine of Classification module.

- **output:** None
- **exception:** ClassificationStartFailure, ClassificationRunTimeFailure

21.4.5 Local Functions

None

References

[1] D. M. Hoffman and P. A. Strooper, *Software Design, Automated Testing, and Maintenance: A Practical Approach*. New York, NY, USA: International Thomson Computer Press, 1995. [Online]. Available: http://citeseer.ist.psu.edu/428727.html

[2] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2003.

Appendix — Reflection

1. What went well while writing this deliverable?

   Sathurshan: The system was decomposed into modules that was small enough. This allowed the team to easily split up the work without having much dependencies on each other. Furthermore, writing the MIS has helped the team further align with design decisions and formally document.

   Kalp: I think that developing the PoC (Proof of Concept) was a great way to get a better understanding of the system and the modules that are involved. Though maybe not ideal in other scenarios, getting a small headstart on the design implementation helped us really easily scope out what the modules on the system should be.

   Nirmal: Since the classification POC was already created, I feel like it was a lot easier to decompose the classification feature into distinct features. Furthermore, since the POC was created earlier, a lot of the research was already completed, which means I knew what equations I needed for each module.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   Sathurshan: The main pain point was the deliverable deadline as the team was asked to finish the first revision in a week while preparing for the proof of concept of demo. The team expressed the concerns with the professor, and fornately received an extension allowing us to put more effort into this document.

   Kalp: I think that the main pain point was just timing. With the VnV plan due shortly prior to this deliverable, and the team PoC implementation being due a week after this deliverable, there was really not much time to write out this document, especailly given the lenght and detail that it goes into. This was slightly mitigated by the provided extension and the PoC implementation, but it was still a challenge to get everything done in time.

   Nirmal: Although I mentioned I had a good idea of which modules are needed for classification, modularizing which equation goes in what module and in what order was a difficult and time consuming task. For example the POC was done as a python implementation, and as such the logic with coming up with the LDA projection matrix was encapsulated into python libraries. For this document, I had to reverse engineer how exactly this is done using equations.

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

   Sathurshan: The supervisor mentioned the difficulty of getting access to hardware that will meet our software specification. As a result, we designed the system such that the software is portable and can take input from different sources. This ensures

that the capstone project can still be successful in the case we are not able to find the right hardware within our budget.

Kalp: A lot of the hardware and software design decisions that we made was based from speaking to the supervisor and the team. Our supervisor would be able to give better insight on what would and wouldn't work for the scope of what we're trying to accomplish. We selected the pipeline design of audio to processing to output with the specific hardware, but the specific algorithm (ex. LDA for classification) or the specific hardware came through our supervisors insights.

Nirmal: The decision of using PCA and LDA for classification stems from asking our supervisor (MVM) for input on an approach that has low computation. Originally, I was using a SVM for classififying audio sources, but our supervisor mentioned this isn't really necessary, and might be overkill.

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?

   Sathurshan: SRS needed to be updated. It was updated because part of writing this document required reviewing the SRS. From this, I have found parts of the requirements that can be improved based on recent better understanding of the system.

   Kalp: The SRS document was updated to reflect the changes that we made to the system. A lot of the requirements changes revolved around the hardware and software design decisions that we made (ex. choosing to serperate a process into multiple modules leads to more specific requirements).

   Nirmal: For the classification specific modules, no parts of any other documents needed to be changed, since the other documents never mentioned any exact implementation details. Which is what this document is focused on.

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

   Sathurshan: The limitation of the solution is compute and memory power. Given further time, the team would be able to design a system that is more low level and is optimized in accomplishing the tasks that are required.

   Kalp: I think the biggest limitation that we can see right now is the hardware limitations. The software that we're using for this project is good enough to handle the tasks that we're trying to accomplish within the software requirements, but the hardware limitations are what are holding us back slightly. With just better hardware (at the expense of higher cost), a lot of the issues we're dealing with could be mitigated.

   Nirmal: Given unlimited resources, we could have a more robust classification approach. The current approach of using PCA and LDA comes from not having computation power to use resources like SVM or neural networks taking in the full audio input.

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)

   Sathurshan: Personally, there was not a lot of time to think about other designs. The team has been implementing the software before this document was created since the proof of concept is one week after this document is due. Thus, the team already considered and analyzed high level designs months ago and is too far back to document them.

   Kalp: I know for the software side, specifically the audio classification, we considered using a neural network for the classification. This was a potential solution, but we ultimately decided against it because of the hardware limitations. The neural network would require a lot of memory and processing power, which is not available on the hardware that we're using. For this reason, we decided to use the LDA for classification because it is a simple algorithm that is easy to implement and understand, and a lot less computationally expensive than a neural network.

   Nirmal: Another desgin solution we considered for classification was using SVM. Although that seemed to work really well, compared to the PCA and LDA approach, this approach would firstly take a lot of storage on the microcontroller which we didn't have. And also involved running full blackbox models. The approach now just invovles doing matrix multiplications which is what the microcontroller is optimized for.