CSSR for

Culturally Sensitive Social Robotics for Africa

D4.2.3 Sound Detection and Localization

Due date: **31/12/2023**
Submission Date: **21/02/2025**
Revision Date: **20/06/2025**

Start date of project: **01/07/2023** Duration: **36 months**

Lead organisation for this deliverable: **Carnegie Mellon University Africa**

Responsible Person: **Yohannes Haile** Revision: **1.3**

Project funded by the African Engineering and Technology Network (Afretec) Inclusive Digital Transformation Research Grant Programme		
Dissemination Level		
PU	Public	**PU**
PP	Restricted to other programme participants (including Afretec Administration)	
RE	Restricted to a group specified by the consortium (including Afretec Administration)	
CO	Confidential, only for members of the consortium (including Afretec Administration)	

Executive Summary

Deliverable D4.2.3 introduces the `soundDetection` node, a key component designed to detect and localize conspicuous sounds within a robot's hearing range, enabling enhanced interaction and responsiveness of Pepper robot. This module, implemented as a ROS node and it provides two outputs: the direction of arrival (DoA) of the sound, published as an angle in degrees, and a filtered audio signal. Designed to operate reliably in acoustically challenging environments, the module ensures robust performance in real-world conditions. It is integral to the Speech Event (D4.3.2) and Attention Subsytem(D5.3), facilitating Automatic Speech Recognition (ASR) and enabling the robot to focus its attention on sound sources. The deliverable includes a comprehensive documentation package covering the development process, functional specifications, interface design, and testing strategies. A user manual provides clear instructions for building, launching, and configuring the module for use with the Pepper robot. Testing across environments ensures reliability of the sound Detection module.

Contents

1 Introduction

The ability to detect and localize sound is fundamental for robots operating in dynamic and interactive environments. Deliverable D4.2.3 addresses this need by developing the soundDetection module, a ROS-based system that identifies conspicuous sounds and determines their direction of arrival (DoA) using interaural time difference (ITD) as the primary localization technique. By leveraging ITD, the module calculates the time delay of sound arrival between microphones, enabling precise auditory localization within the robot's hearing range. This module enhances the robot's situational awareness and facilitates more natural human-robot interactions using this auditory cues.

The soundDetection module is critical to enabling higher-level functionalities, such as Automatic Speech Recognition (ASR) and OvertAttention, where auditory input is used to trigger speech processing and direct the robot's focus toward sound sources. It is designed to work reliably and handle challenging acoustic conditions such as background noise and reverberation.

This deliverable includes the implementation of `soundDetection` ROS node, which processes multichannel audio signals to output the sound's direction and the captured audio signal. A configuration file (`sound_detection_configuration.json`) enables flexibility in band-pass filter thresholds, and intensity detection thresholds.

In addition to the software implementation, the deliverable provides a detailed report documenting the module's development process, functional specifications, and testing methodology. A user manual is also included, offering clear guidance on building, configuring, and deploying the module.

2 Requirement Definition

The `soundDetection` module is designed to enable robots to detect conspicuous sounds and determine their direction of arrival (DoA) within the hearing range. The key requirements for the module are as follows:

Sound Detection and Localization

- The module must detect conspicuous sounds, such as human voices, and ignore ambient noises or background interference.
- Localization must be limited to the azimuth (horizontal) plane and output the DoA as an angle in degrees relative to the robot's Cartesian head frame.

Configurable Parameters

- Allow customization through Configuration parameters, such as band-pass filter thresholds, and intensity detection thresholds, must be provided via a (`sound_detection_configuration.json`) file.

Input and Output

- **Input:** Multichannel audio signal from the robot's microphones.
- **Output:**
  - **Direction of Arrival:** Published on the ROS topic `soundDetection/direction`.
  - **Audio Signal:** Captured left channel audio, published on the ROS topic `soundDetection/signal`.

Integration

- Outputs must be compatible with higher-level systems, such as the `Speech Event` and `OvertAttention` packages, for Automatic Speech Recognition (ASR) and attention direction.

Verbose Mode

- Provide optional diagnostic output in the terminal

Misalignment of the Module

One of the key misalignment of the `soundDetection` module is its limited operational range in the azimuth plane. The module can only accurately localize sounds within an angle range of $-67°$ to $67°$. This limitation arises due to the smaller distance between the microphones on the robot's head, which affects the cross-correlation technique used for localization. Beyond this range, the cross-correlation results in undefined, reducing the effectiveness of the module in detecting sounds coming from wider angles.

3 Module Specifications

The sound detection module, implemented as a ROS node named soundDetection, is designed to detect sound within Pepper robot's hearing and provide a filtered audio signal and determine the direction of the sound arrival.

The input for this module is multi-channel audio from the robot's microphone. For this node we will be primarily be using the FrontLeft and FrontRight microphone to perform localization. For the sound filtering part we will be using just the FrontLeft Microphone.

The output for this module is two topics. The array of audio data will be published on the `soundDetection/signal` and the direction of conspicuous audio is published on `soundDetection/direction`.

The module employs a band-pass filtering technique to isolate audio signals within the frequency range typical of human voice. This filtering method effectively attenuates frequencies outside the desired range, ensuring that extraneous sounds are minimized. Additionally, the module utilizes spectral subtraction to remove stationary background noise by analyzing noise profiles during silent intervals. The noise signature identified is then subtracted from the overall audio signal, resulting in a cleaner output. Finally, the refined signal is published for further processing or use in downstream applications.

For the localization, the module utilizes interaural time difference (ITD) to localize sound sources. It employs the Generalized Cross-Correlation with Phase Transform (GCC-PHAT) algorithm to estimate the time delay between signals captured by multiple microphones. This method enhances the accuracy of the delay estimation by emphasizing phase information, which is less susceptible to noise interference. The computed time differences are then used to triangulate the position of the sound source relative to the microphone array. Ultimately, this localization technique enables the pepper robot to orient itself based on the direction of arrival of the sound.

If verbosemode is set to True in the configuration file, diagnostic messages and the calculated angle of arrival for the conspicuous sound will be printed out.

A unit test is developed to test the sound detection node under various conditions background noise and background chatter. The tests will be conducted using a driver-stub test platform, which utilizes recorded audio signal from pepper robot stored in the data folder. Additionally, the unit tests can be executed directly on the physical robot to validate real-world performance.

4 Module Design

Audio Input

The input for soundDetection ROS node are pepper's microphone located on the top of pepper robot head. Pepper robot has four micophone located on the top of head for this module though we will be using only the two microphone the FrontLeft (`C`) and FrontRight (`D`) microphone as on diagram below. The node expects audio to be sampled at 48 kHz and delivered in blocks of 4096 samples. For better sound localization a block of 8192 samples are used for better precision for the sound localization. The pepper robot has four microphones located at the top of the robot's head as shown in Figure 1.

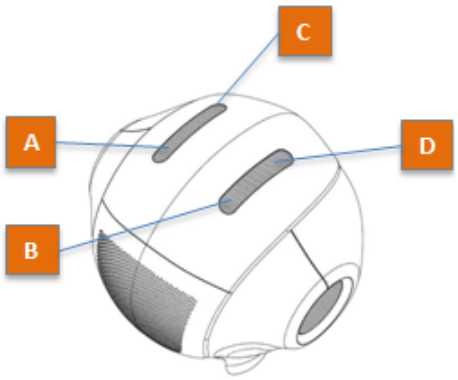

Figure 1: Pepper Microphone. Source: Aldebaran

Table 1: Pepper Microphone Part and Name.

Part	Name
A	`MicroRL_sensor`
B	`MicroRR_sensor`
C	`MicroFL_sensor`
D	`MicroFR_sensor`

Algorithms

WebRTC VAD (Voice Activity Detection)

The WebRTC Voice Activity Detector (VAD) analyzes short frames of audio, typically 10, 20, or 30 milliseconds, to determine if speech is present. It computes various features such as energy levels, zero-crossing rates, and spectral characteristics for each frame. Based on these features, the VAD compares the computed values against predefined thresholds to decide whether the frame contains speech. It offers multiple aggressiveness modes, ranging from 0 to 3, allowing it to be tuned for different noise environments. In quieter settings, lower aggressiveness modes permit more background noise to be classified as speech, while higher modes are more selective in noisy conditions. The algorithm uses a combination of time-domain and frequency-domain analysis to improve its robustness and accuracy across diverse acoustic scenarios. Ultimately, the VAD outputs a binary decision for each frame, ensuring that sound filtering and localization are only triggered when speech is detected.[1]

GCC-PHAT

The GCC-PHAT algorithm is a used method for estimating the time delay between two signals, which is crucial for sound localization. It begins by converting both signals into the frequency domain using the Fourier Transform, allowing for efficient cross-correlation. Then it computes the cross-power spectrum of the two signals and applies a phase transform by normalizing the spectrum with its magnitude, thereby emphasizing phase differences while minimizing amplitude variations. Next an inverse Fourier Transform is applied to the normalized spectrum to obtain a cross-correlation function in the time domain. The algorithm estimates the time delay by locating the peak in this cross-correlation function, which corresponds to the interaural time difference. This delay is then converted into an angle of arrival using the known geometry of the pepper's robot microphone setup(i.e the distance between the two microphone is considered to be (`0.07m`) and the speed of sound (`343m/s`) . By focusing on phase information rather than raw amplitude, GCC-PHAT remains robust in noisy or reverberant environments, ensuring accurate estimation of the sound's direction.[2]

Algorithm 1 GCC-PHAT Algorithm for Sound Localization

Require: Left-channel signal $x(t)$, right-channel signal $y(t)$, sampling frequency fs, small constant ϵ (to avoid division by zero), and optionally a maximum delay T_{max}

Ensure: Estimated time delay $\hat{\tau}$ between $x(t)$ and $y(t)$

1: Compute the Fourier transform of $x(t)$: $X(f) \leftarrow \text{FFT}(x(t))$

2: Compute the Fourier transform of $y(t)$: $Y(f) \leftarrow \text{FFT}(y(t))$

3: Compute the cross-power spectrum: $R(f) \leftarrow X(f) \cdot Y^*(f)$ $\triangleright$ $Y^*(f)$ is the complex conjugate of $Y(f)$

4: Normalize the cross-power spectrum using PHAT:

$$R_{\text{PHAT}}(f) \leftarrow \frac{R(f)}{|R(f)| + \epsilon}$$

5: Compute the inverse Fourier transform to obtain the cross-correlation function:

$$r(\tau) \leftarrow \text{IFFT}(R_{\text{PHAT}}(f))$$

6: Optionally, restrict the search for τ to the interval $[-T_{max}, T_{max}]$

7: Find the time delay $\hat{\tau}$ that maximizes $|r(\tau)|$:

$$\hat{\tau} \leftarrow \arg\max_{\tau} |r(\tau)|$$

8: **return** $\hat{\tau}$

The difference between classical cross correlation and GCC-PHAT is the classical cross correlation computes the similarity between two signals based solely on their amplitudes, which makes it sensitive to variations in signal energy and noise. In contrast, GCC-PHAT introduces a normalization step where the cross-power spectrum is divided by its magnitude, effectively stripping away amplitude information and emphasizing phase differences. This phase emphasis allows GCC-PHAT to be more robust in noisy or reverberant environments, yielding more accurate time delay estimates. By mitigating the influence of amplitude variations, GCC-PHAT can reliably identify the peak corresponding to the true time delay, even when the signals are distorted by noise.

Due to the finite resolution of the audio processing system, the estimated direction of arrival is

inherently quantized into a discrete set of values. The theoretical position of the maximum cross-correlation value can be calculated by

$$n = \frac{l \sin\theta}{c} \times F_s,$$

with our parameters $l = 0.07\,\text{m}$, $c = 343\,\text{m/s}$, and $F_s = 48000\,\text{Hz}$, we first calculate the constant factor:

$$K = \frac{0.07}{343} \times 48000 \approx 9.795.$$

Thus, the delay index becomes

$$n = 9.795 \times \sin\theta.$$

For unique discrimination, assume that the maximum distinct index is $n = 9$ (since index values are quantized). Setting

$$9.795 \times \sin\theta = 9,$$

we solve for θ:

$$\sin\theta = \frac{9}{9.795} \approx 0.9183.$$

Taking the inverse sine, we obtain:

$$\theta \approx \arcsin(0.9183) \approx 67^\circ.$$

Thus, the system can uniquely resolve angles only up to approximately 67° (and -67° on the negative side) due to the discrete nature of the cross-correlation process. Any angle beyond about 67° will produce the same index, limiting the system's angular resolution unless the sampling frequency or the interaural distance is increased.

Noise Reduction

The noisereduce package performs noise reduction by applying spectral gating techniques in the frequency domain. It begins by taking a noisy audio signal and optionally a segment containing only background noise. If no separate noise clip is provided, the function estimates the noise profile from the beginning portion of the signal. The audio is then transformed into a spectrogram using the Short-Time Fourier Transform (STFT), allowing analysis across both time and frequency. From the noise segment, an average noise spectrum is computed, which serves as a reference for identifying and attenuating noise in each time frame. In non-stationary mode, the algorithm adaptively estimates a noise threshold over time and uses a soft sigmoid function to apply a suppression mask that reduces noise while preserving speech components. For stationary noise, a single threshold is used across the entire signal. To minimize artifacts such as choppiness sounds, the mask is smoothed across both frequency and time axes. Finally, the denoised spectrogram is converted back to the time domain using inverse STFT, reconstructing a cleaner audio waveform.

Figure 2 shows how the audio flow chart starting from pepper's microphone until to get filtered audio signal and direction of arrival of the sound.

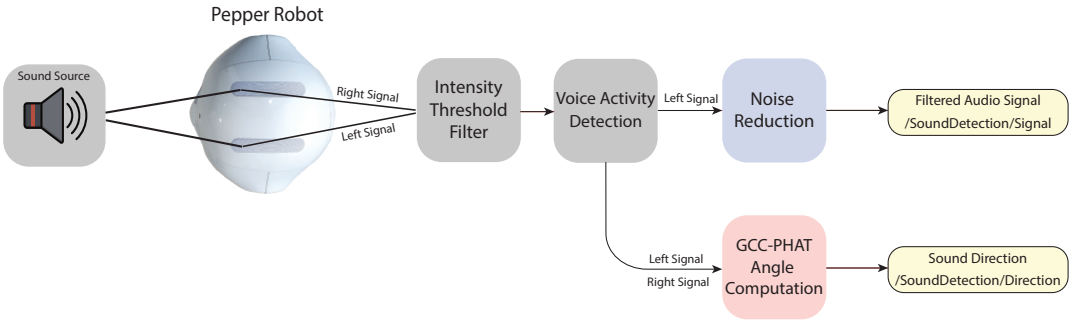

Figure 2: Sound Detection and Localization Flow Chart.

5 Implementation

File Organization

The source code for conducting sound detection and localization is structured into two primary components: `sound_detection_application` and `sound_detection_implementation`. The `sound_detection_implementation` component encapsulates all the essential functionality required for sound filtering as well as sound localization. Additionally, the sound detection system is equipped with the capability to process various files critical for testing, such as configuration files, input files, and topic files. Meanwhile, the `sound_detection_application` component serves as the entry point, invoking the main functions to run the sound detection node and executing the functions defined within `sound_detection_implementation`.

Figure 3 shows the file structure of the sound detection package.

```
sound_detection
├── config
│   └── sound_detection_configuration.json
├── data
│   └── pepper_topics.dat
├── launch
│   └── sound_detection_launch_robot.launch
├── msg
│   └── sound_detection_microphone_msg_file.msg
├── src
│   ├── sound_detection_application.py
│   └── sound_detection_implementation.py
├── sound_detection_requirements.txt
├── README.md
├── CMakeLists.txt
└── package.xml
```

Figure 3: Updated file structure of the sound detection system.

UML Diagram for the Sound Detection and Localization Module

The UML diagram provides a clear structural representation of the sound Detection and Localization Module, illustrating the various field and method present in the soundDetectionNode class.

Figure 4 shows the UML diagram of sound_detection_implementation.py.

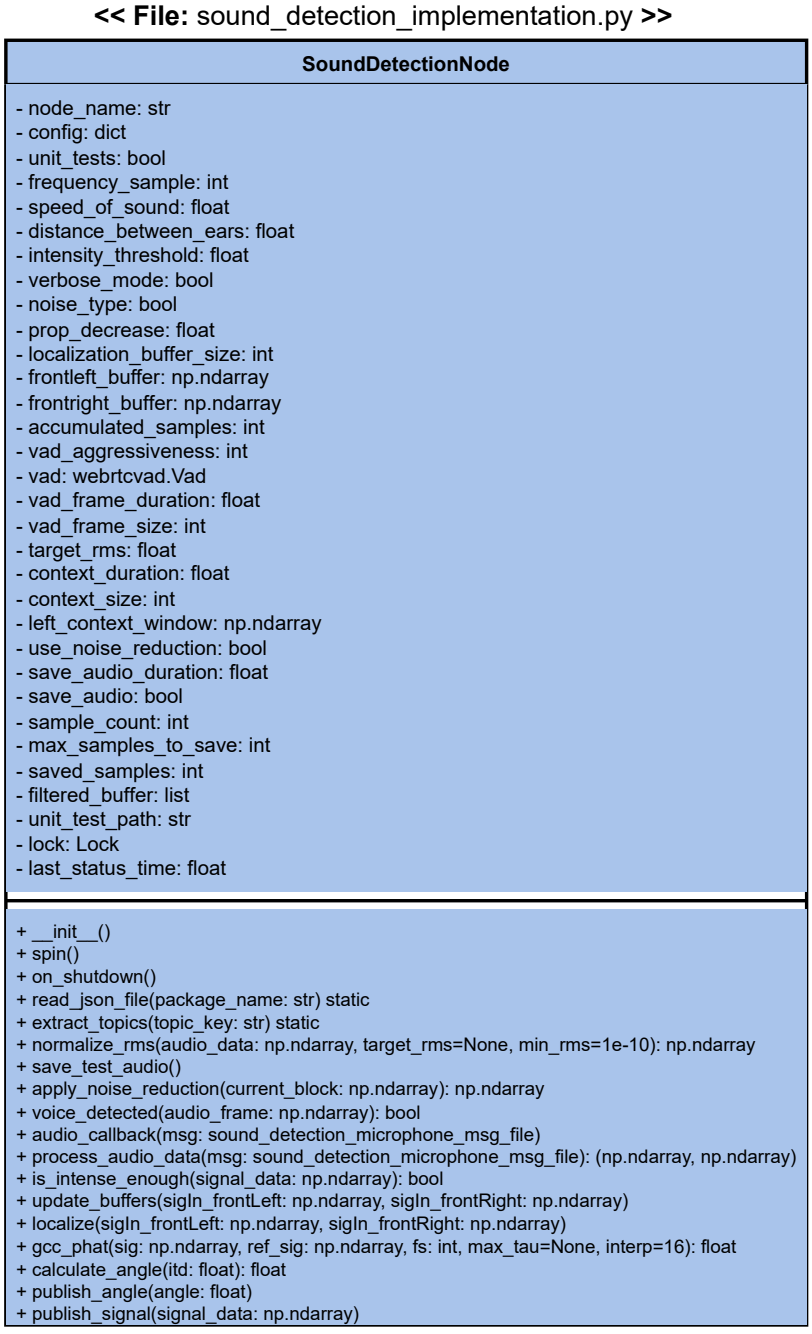

Figure 4: Sound detection implementation UML.

Configuration File

The operation of the sound detection node is determined by the contents of the configuration file that contains a list of key-value pairs shown in Table 2.
The configuration file is named `sound_detection_configuration.json`.

Key	Description
`intensityThreshold`	RMS intensity threshold for audio signal detection. Audio frames with RMS values below this threshold are filtered out as background noise.
`distanceBetweenEars`	Physical distance in meters between the left and right microphones, used in GCC-PHAT algorithm for calculating sound source angle from interaural time difference.
`localizationBufferSize`	Buffer size in samples for accumulating audio data before performing sound localization using the GCC-PHAT algorithm.
`vadAggressiveness`	WebRTC Voice Activity Detection aggressiveness level (0-3). Higher values make the VAD more aggressive in filtering out non-speech audio segments.
`contextDuration`	Duration in seconds for the rolling context window used in noise reduction processing. Longer durations provide better noise estimation but increase memory usage.
`useNoiseReduction`	If set to true, applies spectral subtraction noise reduction to the left audio channel using the noisereduce library.
`stationary`	Noise type parameter for the noise reduction algorithm. If true, assumes stationary noise characteristics for more effective filtering.
`propDecrease`	Proportional decrease factor for the noise reduction algorithm. Controls the aggressiveness of noise suppression, with values closer to 1.0 providing more conservative noise reduction.
`verboseMode`	If set to true, enables detailed logging output including RMS normalization details, noise reduction status, and processing diagnostics.

Table 2: Configuration parameters for the sound detection node.

Input File

There is no input file the sound detection node.

Output File

There is no output file the sound detection node. The node display on the terminal direction of the conspicuous sound.

Models

No models are used for this sound detection node.

Topics File

For the test, a selected list of the topics for the robot is stored in the topics file. The topic files are written in the .dat file format. The data file is written in key-value pairs where the key is the Microphone and the value is the topic. The topics file for the robot is named `pepper_topics.dat`.

Launch File

The launch file `sound_detection_launch_robot.launch` is designed to initialize pepper sensors based on the specified configuration. It declares several parameters that can be customized to match your network settings.

- `pepper_robot_ip`: specifies the IP address of the Pepper robot (default: `172.29.111.230`).
- `roscore_ip`: IP address of the ROS master (default: `127.0.0.1`)
- `pepper_robot_port`: specifies the communication port for Pepper (default: `9559`).
- `network_interface`: specifies the network interface name (default: `wlp0s20f3`).
- `namespace`: sets the ROS namespace for the naoqi driver (default: `naoqi_driver`).

Topics Subscribed

Table 3 shows the topics subscribed by sound detection node.

Sensor	Topic Name	Message Type
`Microphone`	`/naoqi_driver/audio`	`sound_detection_microphone_msg_file`

Table 3: Topics subscribed by the sound detection node.

Topics Published

Table 4 shows the topics published by sound detection node.

Topic Name	Message Type	Description
`/soundDetection/signal`	`std_msgs/Float32MultiArray`	Contains the filtered audio signal corresponding to the input audio block.
`/soundDetection/direction`	`std_msgs/Float32`	Contains the computed angle of arrival (in degrees) of the sound.

Table 4: Topics published by the sound detection node.

6 Running the Sound Detection Node

To run the sound detection node, the user must first install the necessary software packages as outlined in Deliverable 3.3. The required packages are listed in the `sound_detection_requirements.txt` file. The user can follow the README file in the sound detection package to install the required packages. Referring to the implementation section of this deliverable report, the user must set the configuration file to the desired parameters. Using the key-value pair, the user can set the intensity threshold, proportional decrease factor for noise reduction, and other parameters. The user can then run the sound detection node by executing the following command in the terminal:

```
# Launch Pepper robot sensors
$ roslaunch cssr_system sound_detection_launch_robot.launch
```

```
# Run the sound detection node
$ rosrun cssr_system sound_detection_application.py
```

7 Unit Test

The unit test is designed to validate the sound detection node's functionality under various environment including nose such as background chatter and air conditioning noise The test can be performed using a driver-stub test platform, which utilizes recorded audio data stored in the data folder as a rosbag file. The unit test can also be executed directly on the physical robot to validate real-world performance.

The sound detection unit test file structure is as follows:

```
unit_test
└── sound_detection_test
    ├── config
    │   └── sound_detection_test_configuration.json
    ├── data
    │   ├── sound_detection_test_input_sound_distance.bag
    │   ├── sound_detection_test_input_sound_angle.bag
    │   ├── sound_detection_test_input_sound_noise.bag
    │   ├── sound_detection_test_unfiltered_{timestamp}.wav
    │   ├── sound_detection_test_speech_filtered_{timestamp}.wav
    │   └── sound_detection_test_direction_data_{timestamp}.txt
    ├── launch
    │   ├── sound_detection_launch_robot.launch
    │   └── sound_detection_launch_test_harness.launch
    ├── src
    │   ├── sound_detection_test_application.py
    │   └── sound_detection_test_implementation.py
    ├── CMakeLists.txt
    ├── Package.xml
    └── README.md
```

Figure 5: File structure of the sound detection unit test.

The test cases for the sound detection node that are going to be evaluated are as follows:

Test Case	Description
Sound Distance	Verifies the sound detection node's capability to accurately measure the distance of a sound source. This bag file contains audio recordings with variations in source-to-microphone distance, allowing evaluation of distance estimation performance.
Sound Angle	Evaluates the system's ability to compute the angle of arrival of the sound. The bag file includes audio samples recorded from different azimuth angles to validate the accuracy of the GCC-PHAT based localization.
Sound Noise	Assesses the robustness of the sound detection node under noisy conditions. This bag file features audio with various levels of background noise, testing the effectiveness of filtering and voice activity detection (VAD) algorithms.

Table 5: Test cases for sound detection node evaluation using specific bag files.

Note: Valid values for `bag_file` include: `sound_distance`, `sound_angle`, `sound_noise`.

Figure 6 shows the UML diagram of sound_detection_test_implementation.py.

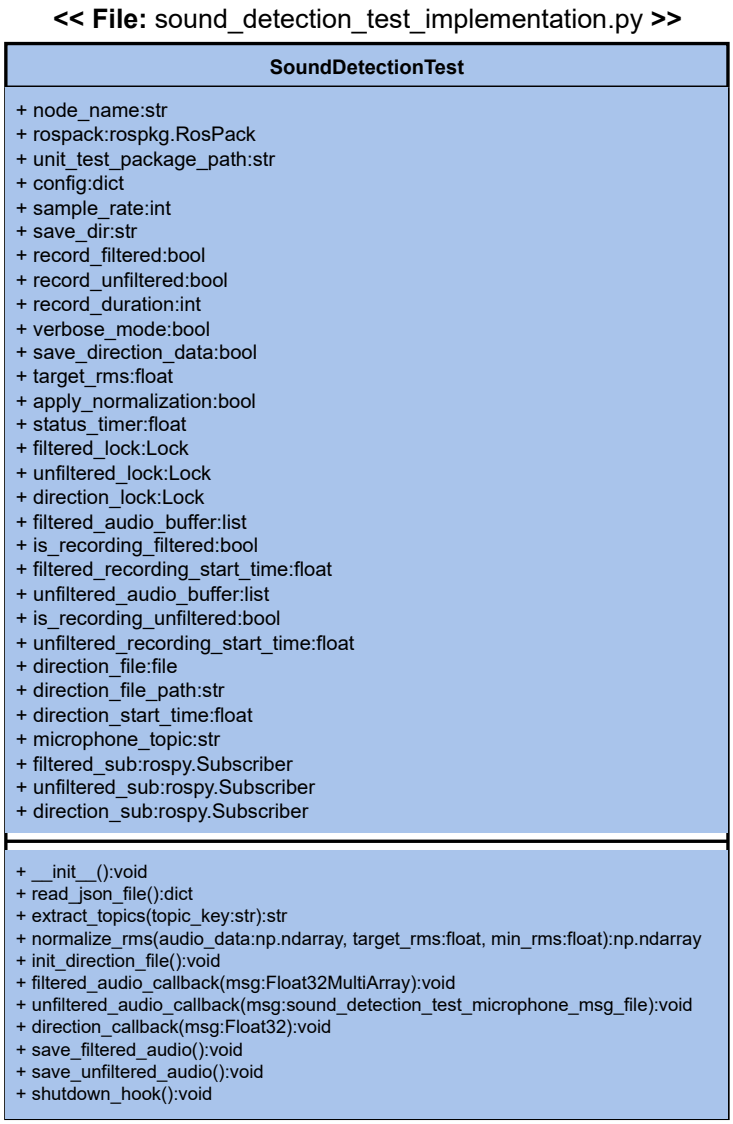

Figure 6: Sound detection implementation test UML.

Configuration File

The configuration file for the sound detection unit test is named `sound_detection_test_configuration.json` and contains the key-value pairs shown in Table 6.

Key	Description
`recordFiltered`	If set to true, records and saves the filtered audio signal.
`recordUnfiltered`	If set to true, records and saves the raw audio signal before processing.

Key	Description
recordDuration	Duration in seconds for which audio is recorded.
saveDirectionData	If set to true, saves calculated direction-of-arrival data for later analysis.
targetRMS	Target root-mean-square (RMS) level for audio normalization.
applyNormalization	If true, applies normalization to the audio signal based on targetRMS.
verboseMode	If set to true, prints debug information and logs to help with diagnostics and testing.

Table 6: Configuration parameters for the sound detection test node.

Launch File

The launch file sound_detection_test_launch_robot.launch is designed to support testing the sound detection node with a live audio input from Pepper's microphone or a recorded a rosbag audio. It provides several configurable arguments to customize the test environment.

The launch file has the parameter shown in Table 7 that can be passed.

Parameter	Description
robot_ip	IP address of the robot for establishing network connection (default: 172.29.111.230)
roscore_ip	IP address of the ROS master node for distributed system communication (default: 127.0.0.1 for localhost)
robot_port	Port number used for robot communication and data exchange (default: 9559)
network_interface	Specific network interface identifier for network communication (default: wlp0s20f3)
namespace	ROS namespace for organizing and isolating naoqi driver nodes and topics (default: naoqi_driver)
audio_file	Specifies the audio file selection for playback during testing (default: sound_distance)
use_recorded_audio	Boolean flag to enable pre-recorded audio playback instead of live microphone input (default: true)

Table 7: Launch file parameters and their descriptions.

This setup allows flexible testing of the sound detection node using live or recorded audio data sources with consistent parameters across different hardware.
The launch file sound_detection_test_launch_test_harness.launch launches the sound_detection node and sound_detection_test node that runs the unit test based on configuration file in the sound_detection_test.

Topics Subscribed

The sound detection test node subscribes to the topics shown in Table 8.

Audio Source	Topic Name	Message Type
PepperMicrophone	/naoqi_driver/audio	sound_detection_microphone_msg_file

Table 8: Topics subscribed by the sound detection test node.

In addition it publishes to `/soundDetection/signal` for processed audio data and `/soundDetection/direction` for calculated sound source angles.

Running Sound Detection and Localization Unit Test

The user can execute the following commands in the terminal to run the unit test for sound detection node.

```
# Launch unit test with live audio from Pepper robot
$ roslaunch unit_tests sound_detection_test_launch_robot.launch
  use_recorded_audio:=false
# or
# Launch unit test with recorded audio file
$ roslaunch unit_tests sound_detection_test_launch_robot.launch
  use_recorded_audio:=true audio_file:=sound_distance
```

```
# Activate the virtual environment:
source cssr4africa_sound_detection_env/bin/activate
```

```
# Run the sound detection node with custom robot IP and port
$ roslaunch unit_tests sound_detection_test_launch_robot.launch
```

The unit test will automatically save filtered audio samples to the data directory validation of the noise reduction and voice activity detection algorithms.

References

[1] WebRTC and Wiseman. Webrtc voice activity detector. https://github.com/wiseman/py-webrtcvad. Accessed: 2025-02-27.

[2] C. H. Knapp and G. C. Carter. The generalized correlation method for estimation of time delay. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(4):320–327, 1976.

Principal Contributors

The main authors of this deliverable are as follows (in alphabetical order).

Yohannes Haile, Carnegie Mellon University Africa.
David Vernon, Carnegie Mellon University Africa.

Document History

Version 1.0

First draft.
Yohannes Haile.
26 February 2025

Version 1.1

Renamed `soundDetectionConfiguration` to `sound_detection_configuration`.
Updated configuration parameters by adding new parameters and detailed description .
Replaced separate band-pass filtering and spectral subtraction sections with unified noise reduction implementation using noisereduce package.
Changed from underscore to space separation in test file names.
Expanded UML diagrams with detailed class attributes, methods, and added test implementation UML.
Updated test file naming convention, changed from single MP3 output to multiple timestamped WAV/TXT files, and redesigned test configuration parameters.
Added launch file parameters including support for both live and recorded audio testing.
Yohannes Haile.
23 May 2025

Version 1.2

Fixed typos.
David Vernon.
16 June 2025

Version 1.3

Explicit references to tables and figures.
Yohannes Haile.
20 June 2025