Culturally Sensitive Social Robotics
for Africa

D5.5.2.4 Integrated Text to Speech Conversion

Due date: **21/03/2025**
Submission Date: **21/03/2025**
Revision Date: **n/a**

Start date of project: **01/07/2023**

Duration: **36 months**

Lead organisation for this deliverable:

Responsible Person: **Richard Muhirwa**

Revision: **1.0**

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 D5.5.2.4 presents the outcomes of Task 5.5.2.4, which integrates the outputs of Tasks 5.5.2.1 (English Text-to-Speech) and 5.5.2.3 (Kinyarwanda Text-to-Speech) into a unified system. This document outlines the results of each stage of the software development process, covering requirements definition, module specification, interface design, module design, testing, and implementation. The Integrated Text-to-Speech system enables the Pepper robot to transform written text in either English or Kinyarwanda into spoken words through its internal speakers. This capability is fundamental to the robot's ability to verbally communicate with users in multiple languages, supporting a wide range of interactions from basic greetings to complex information delivery. The system accepts text input via ROS messages on the `/textToSpeech` topic and processes this text through the appropriate language-specific speech synthesis module based on configuration settings. This report details the functional requirements, interface design specifications, module architecture, testing approach, and implementation instructions for the integrated Text-to-Speech conversion system. Testing results confirm that the Pepper robot's multilingual TTS system functions correctly, producing clear speech in both English and Kinyarwanda for a wide range of inputs, and maintaining stability even under stress conditions.

Contents

1 Introduction

The integrated TTS module developed in this deliverable combines the capabilities of both English and Kinyarwanda text-to-speech conversion into a single unified system. This integration allows for seamless switching between languages, enabling the Pepper robot to communicate effectively in multilingual environments.

The basic workflow of the integrated TTS system follows this pattern: Text input is received via a ROS service call to `/textToSpeech/say_text`.

For English text, the system publishes to the `/speech`, and for Kinyarwanda text, the Coqui TTS model generates a WAV file that is transferred to the robot via SSH and played using the ALAudioPlayer topic for processing by NAOqi.

A success response is returned to the service caller

This dual-path architecture leverages the strengths of each approach: utilizing the robot's built-in capabilities for English speech while employing advanced neural TTS models for Kinyarwanda, where native support is not available in the NAOqi framework.

The implementation addresses practical challenges in multilingual robotics, such as integrating Python 3-based modern TTS frameworks with the Python 2-based NAOqi SDK, and handling the secure transfer and playback of generated audio files. This approach ensures optimal speech quality while maintaining a clean, unified interface for the rest of the robot's systems.
This deliverable is structured as follows:

Section 2 outlines the system's functional and design considerations for the dual-language text-to-speech system. It details the service-based architecture and language-specific processing approaches. **Section 3** presents the module specification, detailing the text-to-speech conversion process for both English and Kinyarwanda languages. It explains how the system leverages NAOqi's built-in capabilities for English while employing the Coqui TTS neural model for Kinyarwanda speech synthesis. **Section 4** describes the interface design, covering the ROS service definition, configuration file format, and the bridge mechanism between Python 3 and Python 2 environments. It details how the system handles text input through the service interface and manages audio output through both direct topic publishing and SSH-based file transfer. **Section 5** details the module's architecture, including component interactions between the ROS node, NAOqi framework, and Coqui TTS library. **Section 6** contains a user manual with step-by-step instructions for configuration, and usage. It provides guidance on setting up the required dependencies, configuring language preferences, and invoking the text-to-speech service programmatically. **Section 7** provides an overview of the testing procedures, including unit tests for each language path and system validation under various network conditions and text input scenarios.

2 Requirements Definition

2.1 Overview

The objective of the integrated Text-to-Speech (TTS) system is to develop a software module that enables the robot to convert text input in either English or Kinyarwanda into natural-sounding speech. The system identifies the target language, processes the text accordingly, and generates appropriate audio output for playback on the robot's speakers. This TTS functionality serves as a crucial component for robot communication, enabling verbal interaction with users in multilingual contexts.

2.2 Functional Specification

The integrated TTS system operates as a ROS node called `textToSpeech` and provides the robot with the ability to vocalize text content in multiple languages. It supports two primary modes of operation:

1. English Mode: Utilizes the NAOqi ALTextToSpeech system by publishing messages to the `/speech` topic, which is then processed by the robot's built-in speech capabilities.
2. Kinyarwanda Mode: Employs the Coqui TTS model to generate speech waveforms that are then transferred to the robot via SSH and played using the ALAudioPlayer proxy.

The system switches between these modes based on configuration settings while maintaining consistent quality standards across both languages.

2.3 Text Processing and Speech Synthesis

The system performs two key processes:

- **Text Parsing and Preprocessing**: The system receives text input through a ROS service interface (`/textToSpeech/say_text`), which accepts string messages. It processes the text according to the selected language mode.
- **Speech Synthesis**: For English, the system publishes the text to the `/speech` topic for native NAOqi processing. For Kinyarwanda, it utilizes the Coqui TTS model with conditioning audio to maintain consistent voice characteristics. The Kinyarwanda synthesis process creates a temporary WAV file, which is then securely copied to the robot via SSH and played using the ALAudioPlayer, with proper cleanup afterward.

The subsystem incorporates error handling mechanisms to provide meaningful feedback when errors occur, particularly when loading the Kinyarwanda speech synthesis components.

2.4 Configuration Management

The robot's speech system is configured through a dedicated configuration file (`textToSpeechConfiguration.ini`) that specifies:

1. Language selection (English or Kinyarwanda)
2. Verbose mode enabling or disabling for debugging purposes
3. Robot IP address for connection
4. Communication port for NAOqi services

The system falls back to default configuration values if the configuration file cannot be read or processed, ensuring continuous operation even in the event of configuration errors.

2.5 Inputs and Outputs

2.5.1 Service Provided

The textToSpeech node provides a service for text vocalization:

Service	Type	Description
/textToSpeech/say_text	text_to_speech/TTS	Service that accepts text messages and returns success status

Table 1: TTS Service Specification

2.5.2 Topics Published (English Mode)

When operating in English mode, the system publishes to:

Topic	Type	Content
/speech	std_msgs/String	Text content to be spoken by NAOqi ALTextToSpeech

Table 2: TTS Published Topics

2.6 External Dependencies and Integration

The TTS system relies on several external components for operation:

- **NAOqi Framework**: Utilized for both direct English TTS via topic publishing and for audio playback in Kinyarwanda mode via the ALAudioPlayer proxy.
- **Coqui TTS**: Python library used for Kinyarwanda speech synthesis, requiring pre-trained models and configuration files located in the `model_files` directory.
- **SSH Communication**: Secure file transfer to the robot for Kinyarwanda audio playback, requiring proper authentication and file management.

The system uses a Python 2 bridge script (`send_and_play_audio.py`) to interface with the NAOqi framework for audio playback, bridging the gap between the modern Python 3 ROS node and the Python 2 NAOqi SDK requirements.

2.7 Error Handling and Logging

The system implements several mechanisms for error management:

- **Configuration errors**: Fallback to default values with appropriate warnings
- **Model loading errors**: Critical error logging and process termination if TTS models cannot be loaded
- **Operation feedback**: Logging of text being spoken and the selected language
- **Service response**: Boolean success indication in the service response

All errors and operational messages are logged through the ROS logging infrastructure to facilitate debugging and system monitoring.

3 Function specification

3.1 Functional Characteristics

3.1.1 Text-to-Speech Conversion

The integrated TTS module consists of a unified ROS node (`textToSpeech`) that handles text-to-speech conversion for both English and Kinyarwanda languages. The language selection is determined by a configuration parameter in the `textToSpeechConfiguration.ini` file.

For English text, the system publishes messages to the `/speech` topic, which is then processed by the NAOqi framework to generate speech through the robot's internal speakers. For Kinyarwanda text, the system employs a TTS Synthesizer from the TTS library to generate a temporary WAV file, which is then transferred to the robot and played using the NAOqi ALAudioPlayer.

3.1.2 Operation Modes

The system supports two operation modes. In Normal Mode, standard operation occurs where text is converted to speech without additional output. In Verbose Mode, extended operation provides additional information logged for monitoring and debugging purposes.

3.1.3 Audio File Generation and Playback

For English TTS, the system publishes text to the `/speech` topic, which is then handled by the NAOqi driver for direct speech synthesis on the robot.

For Kinyarwanda TTS, the system follows a more complex process. First, it uses the TTS Synthesizer to generate audio data. Then, it creates a temporary WAV file. This file is transferred to the robot using SCP via a Python 2 script. Once transferred, the system plays the file using the NAOqi ALAudioPlayer, and finally removes the temporary file after playback [1].

3.1.4 ROS Integration

The module operates as a ROS node named `textToSpeech`. This node provides a service `/textToSpeech/say_text` to handle text-to-speech requests. It publishes to the `/speech` topic for English text and calls external scripts for Kinyarwanda text processing and playback.

3.2 Inputs and Outputs

The TTS module receives input text via a ROS service call, with the following specifications:

Parameter	Specification
Format	UTF-8 encoded text string
Maximum Length	1000 characters
Languages Supported	English, Kinyarwanda
Special Characters	Supported based on language requirements
Input Method	ROS service calls

Table 3: Input Text Specifications

3.2.1 Inputs

The system receives input through a ROS Service `/textToSpeech/say_text`, which uses a custom `TTS` service type with a `message` field. This service accepts text strings to be converted to speech in either English or Kinyarwanda. Additionally, the system reads from a configuration file `textToSpeechConfiguration.ini`, which contains several parameters. These include the language selection (english or kinyarwanda), verboseMode setting (True or False), the robot's IP address, and port number.

3.2.2 Outputs

The system produces several types of output. For English text, it publishes to a ROS Topic `/speech` using the `std_msgs/String` type. This contains the text to be spoken by the NAOqi framework. For Kinyarwanda, it generates an audio file - specifically a temporary WAV file that is transferred to the robot for playback. The system also provides logging through ROS logs for status information and debugging purposes.

3.3 Dependencies

3.3.1 Common Dependencies

The system relies on several common dependencies. These include ROS (with rospy), NAOqi drivers, Python 2 and Python 3.9 environments, and ConfigParser.

3.3.2 English TTS Dependencies

For English text-to-speech functionality, the system depends on the NAOqi framework's speech capabilities, which are accessed via the `/speech` topic.

3.3.3 Kinyarwanda TTS Dependencies

The Kinyarwanda functionality requires more extensive dependencies. The system needs the TTS library Synthesizer class and several pre-trained TTS model files. These include `model.pth` (main model), `config.json` model configuration, `speakers.pth` speaker information, `SE_checkpoint.pth.tar` speaker encoder checkpoint, `config_se.json` speaker encoder configuration, and `conditioning_audio.wav` reference audio for voice characteristics [2]. Additionally, it requires SSH/SCP access to the robot via sshpass and the NAOqi ALAudioPlayer.

3.4 Execution Workflow

The execution follows a defined workflow beginning with Node Initialization. During this phase, the system initializes the ROS node (`textToSpeech`), reads configuration from `textToSpeechConfiguration.ini`, and sets up appropriate resources based on the selected language. For English, it creates a publisher to the `/speech` topic, while for Kinyarwanda, it initializes the TTS Synthesizer with model files. The system then sets up the ROS service (`/textToSpeech/say_text`).

Next in the workflow is Service Request Handling. The system receives text input through service calls and logs the request information.

The third phase involves Language-specific Processing. For English, the system publishes the text to the `/speech` topic. For Kinyarwanda, it generates speech using the TTS Synthesizer, saves it to a temporary WAV file, and calls the Python 2 script to transfer and play the audio.

The final phase is Audio Playback. For English, this is handled by the NAOqi framework. For Kinyarwanda, the system transfers the file to the robot, plays it using ALAudioPlayer, and then removes the temporary file.

3.5 System Requirements

The system has several requirements for proper operation. It requires a ROS workspace with NAOqi driver installed, a Python 3.9 environment for the main node, and a Python 2 environment for NAOqi compatibility. SSH access to the robot with password authentication is necessary, as are pre-trained TTS models for Kinyarwanda. The system also requires network connectivity to the robot.

3.6 Limitations and Assumptions

There are several limitations and assumptions in the current implementation. The system assumes the NAOqi driver is properly configured and running. For Kinyarwanda TTS, SSH access to the robot with the password ”nao” is required. The robot must have sufficient disk space for temporary audio files. Audio quality may differ between the two languages due to different synthesis methods. Currently, the system supports only two languages: English and Kinyarwanda.

4 Interface Design

4.1 Directory Structure

The integrated TTS module follows this directory structure:

```
tts_combined/
├── src/
│   └── text_to_speech/
│       ├── config/
│       │   └── textToSpeechConfiguration.ini
│       ├── model_files/
│       │   ├── model.pth
│       │   ├── config.json
│       │   ├── speakers.pth
│       │   ├── SE_checkpoint.pth.tar
│       │   ├── config_se.json
│       │   └── conditioning_audio.wav
│       ├── scripts/
│       │   └── send_and_play_audio.py
│       └── textToSpeech.py
└── devel/
    └── setup.bash
```

4.2 ROS Service Definitions

The system provides two primary ROS service interfaces:

4.2.1 Text-to-Speech Service

The main text-to-speech service allows for requesting speech synthesis with explicit language specification:

```
# TTS.srv
string message  # Input text to synthesize
string language # Language model to utilize
---
bool success    # Operation status
```

This service is called as:

```
rosservice call /textToSpeech/say_text "message: 'Hello world' language:
   'english'"
```

4.2.2 Language Selection Service

The system also provides a language selection service that allows for dynamic language switching during operation:

```
# SelectLanguage.srv
string language  # Language to select (english or kinyarwanda)
---
bool success     # Operation status
```

This service is called as:

```
rosservice call /textToSpeech/selectLanguage "language: 'kinyarwanda'"
```

4.3 ROS Topic Interface

For English TTS, the system publishes to the following topic:

```
Topic: /speech
Type: std_msgs/String
```

The NAOqi driver listens to this topic and uses its built-in TTS capabilities to produce speech.

4.4 Configuration Interface

The system uses a simple INI-style configuration file with the following structure:

```
[DEFAULT]
verboseMode = False
ip = 172.29.111.230
port = 9559
```

The configuration is loaded at startup and affects the behavior of the system throughout its operation. Note that language selection is no longer handled through this configuration file but is instead managed through the ROS service interface and the culturalLanguageBase node.

4.5 External Script Interface

For Kinyarwanda TTS, a Python 2 script is called with the following arguments:

```
python2 send_and_play_audio.py <audio_file_path> <robot_ip> <robot_port>
```

This script transfers the audio file to the robot, plays it, and then removes the file.

4.6 Error Handling

The system provides error handling for the following scenarios:

1. **Service Request Errors**: Errors during processing are logged, but the service continues to operate.
2. **Language Selection Errors**: If an unsupported language is requested, an error is logged and the current language is maintained.

3. **Synthesizer Initialization Errors**: If the TTS Synthesizer cannot be initialized, an error is logged and the system falls back to alternative methods where possible.

5 Module Design

5.1 Overall Architecture

The integrated Text-to-Speech system has a modular architecture that allows for language-specific processing while maintaining a unified interface. The system consists of the following main components:

1. **ROS Node**: `textToSpeech` - Coordinates all operations and provides the service interfaces
2. **Language Manager**: Handles dynamic language selection and initialization of language-specific resources
3. **Language Handlers**:
   - **English Handler**: Publishes to the `/speech` topic
   - **Kinyarwanda Handler**: Uses TTS Synthesizer and external script for audio playback
4. **External Components**:
   - **NAOqi Driver**: Handles English speech synthesis
   - **TTS Synthesizer**: Generates Kinyarwanda speech
   - **Audio Transfer & Playback Script**: Manages file transfer and playback on the robot
   - **culturalLanguageBase Node**: Specifies the language to be used based on cultural context

5.2 Language Management

During initialization, the TTS node loads all available language models to ensure they are ready for use. The actual language selection is managed by the BehaviorController, which invokes the language selection service to set the appropriate language based on information from the culturalLanguageBase node. This approach allows for dynamic language switching during operation without restarting the node.

The language selection process is implemented as follows:

```
def select_language(self, request):
    """Handle language selection requests"""
    if request.language in self.available_languages:
        self.current_language = request.language
        rospy.loginfo(f"Switched to {request.language} language")
        return True
    else:
        rospy.logerr(f"Requested language ’{request.language}’ is not
           available")
        return False
```

This service is registered during node initialization:

```
# Register language selection service
rospy.Service('/textToSpeech/selectLanguage', SelectLanguage,
    self.select_language)
```

5.3 English TTS Implementation

For English TTS, the system uses a simple ROS publisher to send text to the `/speech` topic, which is handled by the NAOqi driver:

```
if self.current_language == 'english':
    rospy.loginfo(f"Saying '{message}' in {self.current_language}")
    self.speech_pub.publish(message)
```

The publisher is initialized during node startup:

```
# Initialize publisher for English TTS
self.speech_pub = rospy.Publisher('/speech', String, queue_size=100)
```

5.4 Kinyarwanda TTS Implementation

For Kinyarwanda TTS, the system uses a more complex approach:

5.4.1 TTS Synthesizer Initialization

```
# Initialize Kinyarwanda TTS
try:
    from TTS.utils.synthesizer import Synthesizer
    self.kinyarwanda_synthesizer = Synthesizer(
        f"{model_files_dir}/model.pth",
        f"{model_files_dir}/config.json",
        tts_speakers_file=f"{model_files_dir}/speakers.pth",
        encoder_checkpoint=f"{model_files_dir}/SE_checkpoint.pth.tar",
        encoder_config=f"{model_files_dir}/config_se.json",
        use_cuda=False
    )
    self.available_languages.append('kinyarwanda')
except Exception as e:
    rospy.logerr(f"Failed to initialize Kinyarwanda TTS: {e}")
```

5.4.2 Speech Generation and Playback

```
elif self.current_language == 'kinyarwanda':
    rospy.loginfo(f"Saying '{message}' in {self.current_language}")
    wav = self.kinyarwanda_synthesizer.tts(
        message,
        speaker_wav=f"{model_files_dir}/conditioning_audio.wav"
    )
    with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as fp:
        self.kinyarwanda_synthesizer.save_wav(wav, fp)
    subprocess.run([python2_path, python2_script, fp.name, self.robot_ip,
        self.robot_port])
```

5.5 Service Handler Implementation

The updated TTS service handler now accepts both the text message and the desired language:

```
def say_text(self, request):
    """Process text-to-speech requests with language specification"""
    message = request.message

    # If language is specified in the request, temporarily use it
    temp_language = None
    if request.language and request.language in self.available_languages:
        temp_language = self.current_language
        self.current_language = request.language

    try:
        # Process based on current language
        if self.current_language == 'english':
            rospy.loginfo(f"Saying '{message}' in {self.current_language}")
            self.speech_pub.publish(message)
        elif self.current_language == 'kinyarwanda':
            rospy.loginfo(f"Saying '{message}' in {self.current_language}")
            wav = self.kinyarwanda_synthesizer.tts(
                message,
                speaker_wav=f"{model_files_dir}/conditioning_audio.wav"
            )
            with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as fp:
                self.kinyarwanda_synthesizer.save_wav(wav, fp)
            subprocess.run([python2_path, python2_script, fp.name,
                self.robot_ip, self.robot_port])

        # Restore original language if temporary was used
        if temp_language:
            self.current_language = temp_language

        return TTSResponse(success=True)
    except Exception as e:
        rospy.logerr(f"Error processing TTS request: {e}")
        # Restore original language if temporary was used
        if temp_language:
            self.current_language = temp_language
        return TTSResponse(success=False)
```

5.6 Interaction with culturalLanguageBase

The system interacts with the culturalLanguageBase node, which provides cultural context information including the preferred language for communication. The BehaviorController subscribes to updates from the culturalLanguageBase and invokes the language selection service as needed:

```
def cultural_language_callback(self, msg):
    """Handle updates from the culturalLanguageBase node"""
    if msg.language != self.current_language:
        # Call the language selection service
        try:
            rospy.wait_for_service('/textToSpeech/selectLanguage', timeout=1.0)
            select_language =
                rospy.ServiceProxy('/textToSpeech/selectLanguage',
                SelectLanguage)
            response = select_language(msg.language)
            if response.success:
                rospy.loginfo(f"Updated language to {msg.language}")
            else:
                rospy.logwarn(f"Failed to update language to {msg.language}")
        except (rospy.ROSException, rospy.ServiceException) as e:
            rospy.logerr(f"Service call failed: {e}")
```

This callback is registered to a subscription to the culturalLanguageBase's language topic:

```
# Subscribe to cultural language base
rospy.Subscriber('/culturalLanguageBase/language', LanguageMsg,
    self.cultural_language_callback)
```

6 User Manual

```
# For Python 3.9
pip install rospkg TTS

# For Python 2
pip2 install pynaoqi

# System dependencies
sudo apt-get install sshpass
```

```
cd tts_combined
catkin_make
```

```
roslaunch naoqi_driver naoqi_driver.launch nao_ip:=172.29.111.230
    network_interface:=wlp0s20f3
```

```
cd tts_combined
source devel/setup.bash
rosrun text_to_speech textToSpeech.py
```

```
cd tts_combined
source devel/setup.bash
rosservice call /textToSpeech/say_text "message: 'muraho, murakaza neza, muri,
    robotics lab. '"
```

```
rosservice call /textToSpeech/say_text "message: 'Hello, welcome to robotics
    lab.'"
```

```
nano src/text_to_speech/config/textToSpeechConfiguration.ini
```

```
[DEFAULT]
language = english  # Change to "kinyarwanda" for Kinyarwanda
verboseMode = False
ip = 172.29.111.230
port = 9559
```

```
# Press Ctrl+C in Terminal 2 to stop the node
# Then restart it:
rosrun text_to_speech textToSpeech.py
```

6.1 Common Issues

6.1.1 NAOqi Driver Connection Failure

Check that the robot IP is correct Ensure the network interface is correctly specified Verify that the robot is powered on and connected to the network

7 Unit Testing

7.1 Introduction

The integrated Text-to-Speech (TTS) system has been tested using a rigorous unit testing approach that ensures all components function correctly in isolation and as an integrated system. This section presents the unit tests performed, including verification against module specification and validation against requirements definition. The testing framework adheres to the standard ROS testing conventions with specific launch files for different testing environments.

7.2 Unit Test Launch Files

Three launch files have been created for different testing scenarios, following the standard naming convention `<component_name>_launch<environment>.launch`:

1. `textToSpeech_launchRobot.launch`: For testing on the physical Pepper robot
2. `textToSpeech_launch_test_harness.launch`: For testing with dedicated drivers and stubs

7.2.1 Physical Robot Testing

The `textToSpeech_launchRobot.launch` file is configured to launch the TTS component on the physical Pepper robot.

The launch file connects the TTS system to the physical robot's audio output system via the NAOqi driver, allowing speech synthesis to be tested directly on the robot hardware. It also launches a mock cultural language base node to simulate language selection changes.

This launch file connects the TTS system to a simulated Pepper robot environment, allowing testing without physical hardware. Instead of outputting to physical speakers, speech is routed to the simulator's audio output, which can be monitored through ROS topics.

7.2.2 Test Harness

The `textToSpeech_launch_test_harness.launch` file provides a complete test environment with drivers and stubs:

This launch file sets up a complete test environment without requiring the robot or simulator. It uses:

- `mock_naoqi_driver.py`: A stub that simulates the NAOqi driver's behavior
- `tts_test_publisher.py`: Generates test input data for the TTS node
- `tts_test_subscriber.py`: Captures and validates output from the TTS node
- `mock_cultural_language.py`: Simulates language selection changes

7.3 Test Implementation

Validation of Communication and Computation Functionality and the expected input and output relationships:

```
# Validation of Functionality

# Input-Output Relationships

1. English Text Input:
   - Input: Service call with English text and language="english"
   - Output: Text published to /speech topic
   - Validation: The robot speaks the text in English or it appears in the
       /speech_log topic

2. Kinyarwanda Text Input:
   - Input: Service call with Kinyarwanda text and language="kinyarwanda"
   - Output: Audio file created, transferred to robot, played
   - Validation: The robot speaks the text in Kinyarwanda or it appears in the
       logs

3. Language Selection:
   - Input: Service call to /textToSpeech/selectLanguage
   - Output: Language changed in the TTS system
   - Validation: Subsequent TTS requests use the selected language

4. Request-specific Language Override:
   - Input: Service call with language parameter different from current default
   - Output: Text spoken in the specified language
   - Validation: The correct language is used for that specific request only
```

Configuration Validation and how changing configuration parameters affects behavior:

```
# Configuration Validation

The following parameters in the textToSpeechConfiguration.ini file can be
modified to validate the configuration functionality:

1. verbose_mode:
   - Default: false
   - Effect when set to true: Additional logging information is displayed in
       the console
   - Validation: Check for detailed log messages when verbose_mode=true

2. robot_ip:
   - Default: 172.29.111.230
   - Effect when changed: The system connects to a different robot
   - Validation: If set incorrectly, error logs will show connection failures

3. robot_port:
   - Default: 9559
   - Effect when changed: The system connects to a different port on the robot
   - Validation: If set incorrectly, error logs will show connection failures
```

References

[1] Edresson Casanova, Julian Weber, Christopher Shulby, Arnaldo Candido Junior, Eren Gölge, and Moacir Antonelli Ponti. Yourtts: Towards zero-shot multi-speaker tts and zero-shot voice conversion for everyone. *Proceedings of the 39th International Conference on Machine Learning*, pages 2709–2720, 2022.

[2] Jaehyeon Kim, Jungil Kong, and Juhee Son. Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech. *Proceedings of the 38th International Conference on Machine Learning*, pages 5530–5542, 2021.

Principal Contributors

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

David Vernon, Carnegie Mellon University Africa.
Richard Muhirwa, Carnegie Mellon University Africa.
Tsegazeab Tefferi, Carnegie Mellon University Africa

Document History

Version 1.0

First draft.
Richard Muhirwa.
24 March 2025.