CSSR for Africa

Culturally Sensitive Social Robotics
for Africa

D5.5.2.3 Kinyarwanda Text to Speech Conversion

Due date: **26/02/2025**
Submission Date: **02/03/2025**
Revision Date: **17/03/2025**

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

Lead organisation for this deliverable:

Responsible Person: **Richard Muhirwa** Revision: **1.1**

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 D5.5.2.3 concerns the results of Task 5.5.2.3, a task whose objective was to train, test, and deploy a text-to-speech (TTS) model for the Kinyarwanda language. This model aims to synthesize natural-sounding speech from text inputs, enabling the generation of audio output.

This report details the output of each phase of the software development process used in the fulfillment of Deliverable D5.5.2.3. The **requirements definition** section specifies the functional requirements of KinyarwandaTTS, outlining its role in delivering accurate and natural speech synthesis for Kinyarwanda-speaking users. The **module specification** section describes the core functionality of the TTS model, including its linguistic alignment and phonetic optimization for the Kinyarwanda language. The **interface design** section outlines the inputs and outputs of the TTS system, detailing how text inputs are converted into audio streams. The **module design** section delves into the architecture of the deep learning models employed, focusing on their ability to capture phonetic and prosodic features unique to Kinyarwanda. The **testing** section showcases the results and descriptions of unit and end-to-end tests conducted to evaluate the accuracy, intelligibility, and naturalness of the synthesized speech. Lastly, the **user manual** section provides step-by-step instructions on how to deploy and utilize the KinyarwandaTTS model.

Contents

1 Introduction

Text-to-Speech (TTS) systems play a crucial role in enabling natural and effective communication between humans and machines. A TTS system converts written text into spoken speech, serving as the final step in many conversational and assistive technologies.

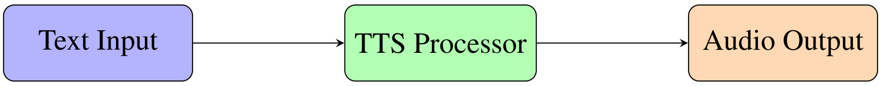

Figure 1: Basic workflow of a Text-to-Speech system, showing the conversion path from input text through processing to audio output.

A TTS system typically consists of several components: text preprocessing, linguistic analysis, prosody generation, and speech synthesis as illustrated in Figure 1. Text preprocessing involves cleaning and normalizing text inputs, linguistic analysis maps the text to phonetic representations, prosody generation determines the rhythm and intonation of the speech, and speech synthesis produces audio output that conveys the desired naturalness. Together, these components enable TTS systems to generate speech that is both accurate and contextually appropriate.

The KinyarwandaTTS module developed in this deliverable focuses exclusively on the text-to-speech component. It is designed to synthesize speech in Kinyarwanda. The model receives text inputs, processes them to capture linguistic and phonetic nuances unique to Kinyarwanda, and produces natural-sounding audio outputs.

This development is part of a broader initiative to create culturally sensitive social robotics applications for African contexts, where the ability to communicate in local languages is essential for acceptance and effectiveness. By enabling robots to speak Kinyarwanda naturally, we aim to make human-robot interaction more intuitive and accessible for Kinyarwanda speakers, supporting applications in education, healthcare, customer service, and other domains where voice interaction is valuable.

2 Requirements Definition

A running **KinyarwandaTTS** system performs one main function—converting Kinyarwanda text input into natural-sounding speech. The system receives text inputs, processes the text to account for Kinyarwanda-specific linguistic and phonetic rules, and generates corresponding audio outputs that capture the rhythm, intonation, and natural flow of the language.

In order to feasibly perform this function, the following functional requirements need to be fulfilled by **KinyarwandaTTS**:

1. **Input Text Parsing and Preprocessing** - Receive Kinyarwanda text input in UTF-8 format through an accessible interface (e.g., file input, or command-line argument). - Normalize the text by removing unnecessary symbols, handling punctuation, and resolving abbreviations to ensure proper pronunciation.

2. **Phonetic and Linguistic Analysis** - Map the normalized text to phonetic representations tailored to Kinyarwanda's unique linguistic structure. - Generate prosody features, such as pitch, duration, and stress patterns, to mimic the natural intonation and rhythm of spoken Kinyarwanda.

3. **Speech Synthesis** - Use a trained deep learning-based TTS model to synthesize high-quality audio output from the processed text and prosody features. - Ensure the audio output is natural-sounding, with minimal artifacts.

4. **Output Speech Generation** - Generate and deliver the synthesized speech as an audio file in standard formats (e.g., WAV, MP3). - Provide the option to output the audio stream directly for real-time playback.

3 Module Specification

The KinyarwandaTTS module is responsible for converting text messages into synthesized speech in the Kinyarwanda language and playing the generated audio on a remote robot, such as Pepper. The module is implemented as a ROS node that interacts with other ROS topics and external systems. Below are the detailed specifications of the module:

3.1 Functional Characteristics

3.1.1 Text-to-Speech Conversion

- The core functionality of the module is to synthesize Kinyarwanda speech from text received on the `/text_to_say` ROS topic.
- The Coqui TTS is a free text-to-speech (TTS) library that uses deep learning to convert text to audio. Coqui TTS model is used for generating speech, with pre-trained models and configuration files specified during initialization.

3.1.2 Audio File Generation

The synthesized speech is saved as a temporary `.wav` audio file for playback. The temporary file is created using Python's `tempfile` library and deleted automatically after playback.

3.1.3 Audio Playback on Remote Robot

The module uses the `send_and_play_audio.py` script to transmit the audio file to the robot and play it. Secure transfer of the audio file is achieved using `sshpass` and `scp`. Playback is managed by the NAOqi framework via the `ALAudioPlayer` proxy on the robot.

3.1.4 ROS Integration

The ROS node subscribes to the `/text_to_say` topic of type `std_msgs/String`, receiving text messages to convert to speech. It can be launched using the `rosrun` command from a properly set-up ROS workspace.

3.2 Inputs and Outputs

3.2.1 Inputs

- **ROS Topic:** `/text_to_say`
  - **Type:** `std_msgs/String`
  - **Description:** Accepts a text string in Kinyarwanda to be converted into speech.

3.2.2 Outputs

- **Audio File:** A temporary `.wav` file generated during runtime and used for playback.
- **Remote Playback:** The synthesized speech is played back on the robot via the NAOqi framework.

3.3 Dependencies

3.3.1 Python Dependencies

- `rospy` (ROS integration)
- `std_msgs` (ROS message types)
- `tempfile` (temporary file handling)
- `subprocess` (external process execution)
- `os` (file system operations)
- Coqui TTS (`TTS.utils.synthesizer.Synthesizer`)

3.3.2 NAOqi Framework

- `ALProxy` (audio playback proxy)
- Secure communication and file transfer using `sshpass` and `scp`.

3.3.3 External Files

- Model files for TTS, including `.pth` and `.json` configurations, located in `/home/muhirwa/tts_ws/src/`
- `send_and_play_audio.py` script for transferring and playing the audio on the robot.

3.4 Configuration

The module requires the following configurations for proper operation:

- **TTS Model Paths:** Paths to pre-trained TTS model files (`model.pth`, `config.json`, etc.).
- **Python Version:** The primary ROS node runs on Python 3.9, while the playback script uses Python 2.
- **Robot Connection:**
  - **Robot's IP address:** `172.29.111.230`.
  - **Robot's username:** `nao`.
  - **Robot's password:** `nao`.

3.5 Execution Workflow

3.5.1 Node Initialization

The ROS node `kinyarwandaTTS` initializes and sets up the Coqui TTS synthesizer. The `/text_to_say` topic subscriber is registered.

3.5.2 Text Reception

The node receives text input via the `/text_to_say` topic.

3.5.3 Speech Synthesis

The received text is processed and converted into a `.wav` audio file using the TTS model.

3.5.4 Audio Playback

The generated audio file is securely transferred to the robot. The playback script (`send_and_play_audio.py`) executes the file on the robot and deletes it after playback.

3.6 System Requirements

- ROS workspace configured with the NAOqi drivers.
- Coqui TTS model and required dependencies installed.
- Python 3.9 for the ROS node and Python 2 for the playback script.
- SSH connection between the local system and the robot.

3.7 Limitations and Assumptions

The current implementation assumes that the robot (e.g., Pepper) is accessible over SSH and correctly configured with NAOqi drivers. The playback script must be compatible with the target robot's operating environment. Temporary .wav files must be generated and deleted properly to avoid storage issues.

4 Interface Design

4.1 Directory Structure

The directory structure of the `kinyarwanda_tts` module is as follows:

```
kinyarwanda_tts/
├── launch/
│   └── kinyarwanda_tts.launch
├── model_files/
│   ├── model.pth
│   ├── config.json
│   ├── speakers.pth
│   ├── SE_checkpoint.pth.tar
│   ├── config_se.json
│   └── conditioning_audio.wav
├── scripts/
│   ├── __init__.py
│   └── send_and_play_audio.py
├── src/
│   └── tts_node.py
├── tests/
│   ├── __init__.py
│   └── test_tts_node.py
├── README.md
├── package.xml
├── CMakeLists.txt
└── requirements.txt
```

Explanation of File and Folder Structure

1. `launch/`
    - Contains the launch file for starting the KinyarwandaTTS ROS node.
2. `model_files/`
    - Stores the TTS model files and configurations:
        - model.pth: Pre-trained Coqui TTS model.
        - config.json: Configuration file for the TTS model.
        - speakers.pth: Speaker embeddings for voice customization.
        - SE_checkpoint.pth.tar: Speaker encoder checkpoint file.
        - config_se.json: Configuration file for the speaker encoder.
        - conditioning_audio.wav: Audio file for conditioning the model.

3. `scripts/`
   - Contains additional scripts used by the node:
     - `send_and_play_audio.py`: Python 2 script to transfer and play audio on the robot.
     - `__init__.py`: Marks the directory as a Python module.

4. `src/`
   - Contains the main source code for the KinyarwandaTTS ROS node:
     - `tts_node.py`: Core Python script implementing the node functionality.

5. `tests/`
   - Contains unit tests and integration tests for the node:
     - `test_tts_node.py`: Test suite for validating the TTS functionality.
     - `__init__.py`: Marks the directory as a Python module.

6. **Root Files**
   - `README.md`: Documentation file with instructions on setup and usage.
   - `package.xml`: ROS package metadata.
   - `CMakeLists.txt`: Build configuration for the ROS package.
   - `requirements.txt`: Python dependencies required for the node.

5 Module Design

The module design of the KinyarwandaTTS system implements an end-to-end architecture for text-to-speech synthesis for the Kinyarwanda language, as shown in the 3. The system follows a unified pipeline based on a conditional variational autoencoder with adversarial learning.

5.1 Text Encoding and Prior Encoder

This component processes input text and language information to create the foundation for speech synthesis. The prior encoder processes input text through character embedding and incorporates language ID information via language embedding. These embeddings are then transformed through a transformer-based encoder with relative positional representation instead of absolute positional encoding. A linear projection layer produces the mean and variance for constructing the prior distribution. The architecture includes a stack of affine coupling layers consisting of WaveNet residual blocks for normalizing flow, using volume-preserving transformation with Jacobian determinant of one for simplicity[1].

5.2 Neural Acoustic Modeling with Stochastic Duration Prediction

This component transforms text representations into acoustic features through interconnected neural network components. The system generates hidden text representation (htext) via the transformer encoder and predicts phoneme duration distributions through the stochastic duration predictor. It creates alignment information through linear projection and applies flow-based decoding with affine coupling layers. The stochastic duration predictor utilizes residual blocks with dilated and depth-separable convolutional layers, along with neural spline flows for improved transformation expressiveness in coupling layers. The component integrates speaker embedding from reference audio with global conditioning and supports noise input to enable variability in synthesis [1].

5.3 HiFi-GAN Decoder for Waveform Generation

This component converts the acoustic features into the final audio waveform using a high-fidelity neural vocoder. The HiFi-GAN decoder generates high-fidelity speech waveforms from the flow-based decoder output and applies transposed convolutions followed by multi-receptive field fusion modules (MRF). It follows the HiFi-GAN V1 generator architecture with stacked transposed convolutions, and the multi-receptive field fusion modules combine residual blocks with different receptive field sizes. A linear transformation of speaker embedding is added to the input latent variables (z), enabling high-quality audio synthesis that preserves the linguistic nuances of Kinyarwanda [1].

5.4 Speech Synthesis Module

This is the runtime module where the actual speech synthesis takes place using the trained end-to-end model. The module orchestrates the complete inference pipeline from text to speech and converts text input into audio waveforms following the procedure in the diagram. It utilizes the pre-trained end-to-end conditional variational autoencoder model and supports GPU acceleration for faster synthesis when available. During inference, the system omits the posterior encoder and discriminator components that are only used during the training phase. The module accepts raw text in Kinyarwanda and optional reference audio for speaker characteristics as inputs, and produces raw audio waveform in .wav format as its output. See Figure 2 and Figure 3 for training and inference procedures.

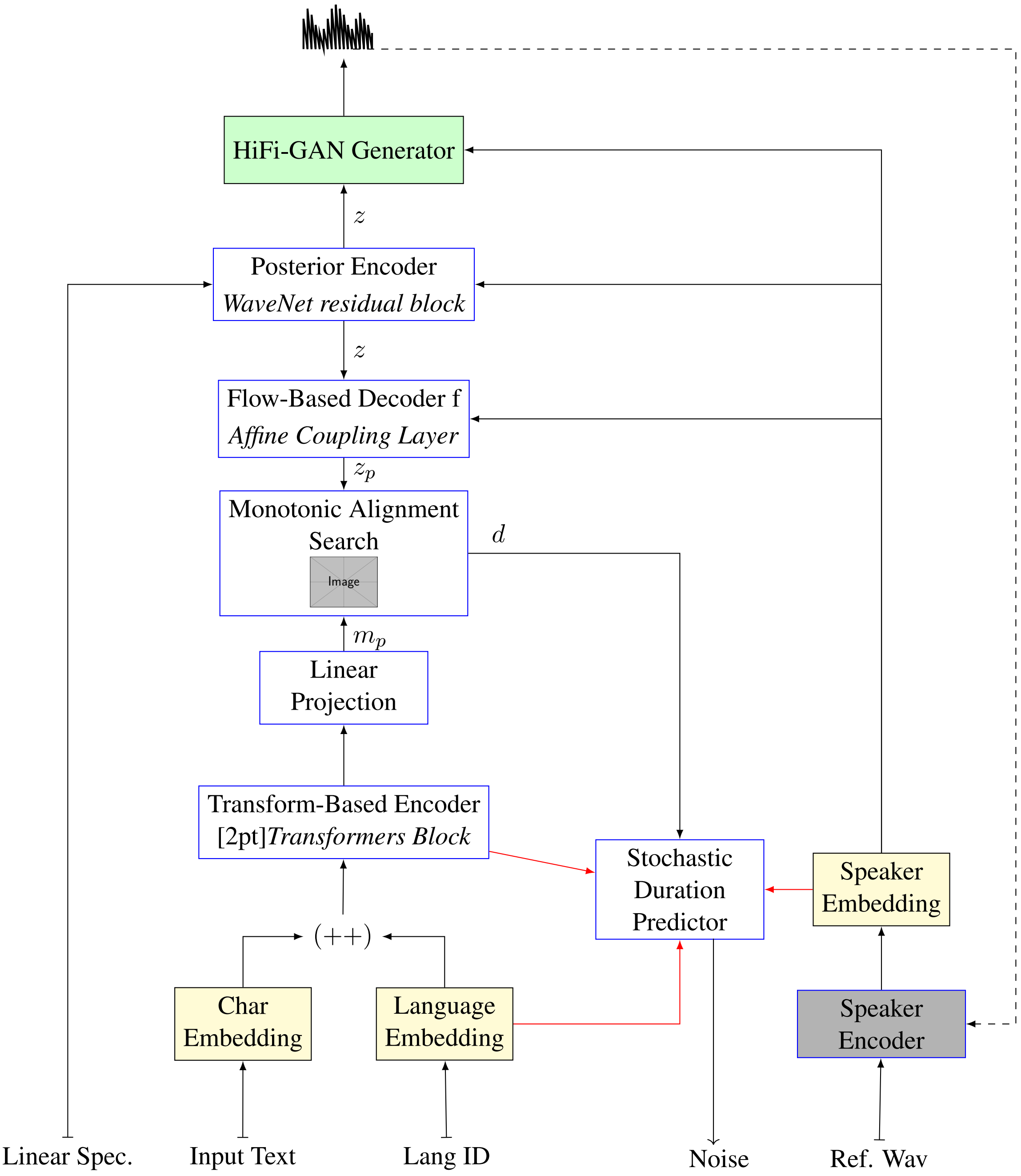

Figure 2: Kinyarwanda TTS training procedure showing the data flow between text encoder, flow-based decoder, and audio generation components. [2]

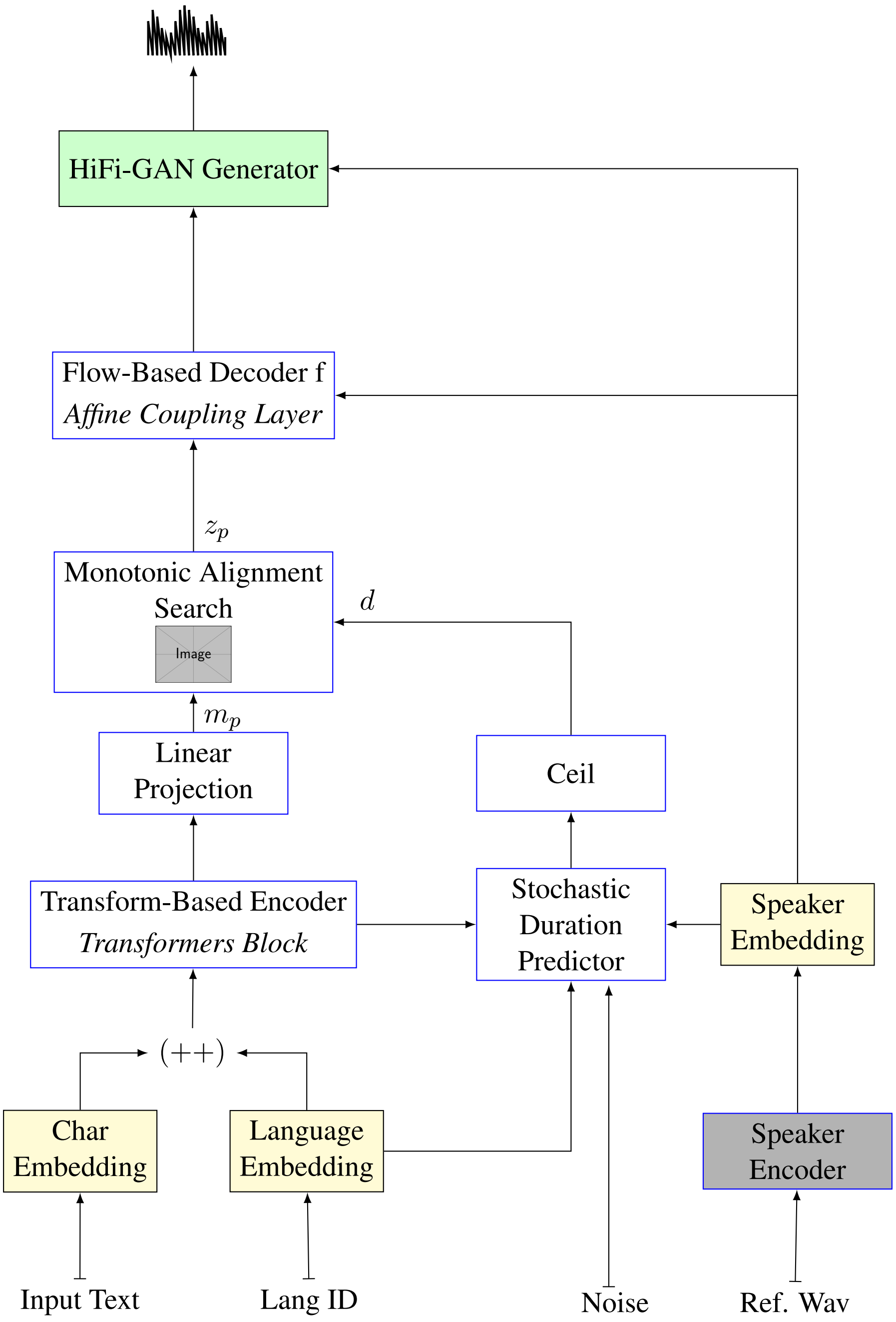

Figure 3: Kinyarwanda TTS inference procedure depicting how input text is processed and transformed into speech output. [2]

5.5 Output Audio Management Module

This module handles the generated audio output, ensuring proper storage, playback, and streaming capabilities.

Responsibilities:

- Save the synthesized audio to a specified file format (e.g., .wav, .mp3).
- Provide APIs for real-time audio playback or streaming.
- Perform post-processing, such as normalization and noise reduction.

Key Features:

- Integration with remote systems or robots for audio playback.
- Temporary file management for efficient resource utilization.

6 User Manual

6.1 Setting Up and Running KinyarwandaTTS

The KinyarwandaTTS system is implemented as a ROS node that integrates with NAOqi Framework for robotic applications. Follow these steps to set up and run the system:

6.1.1 Environment Setup

1. Navigate to the root of your ROS workspace:

```
cd /your_ros_workspace
```

2. Compile the ROS packages:

```
catkin_make
```

3. Source the setup file to ensure the environment variables are properly configured:

```
source devel/setup.bash
```

6.1.2 Launching the NAOqi Driver

Before running the KinyarwandaTTS node, you must establish communication with the robot:

1. Launch the NAOqi driver, specifying the robot's IP address and your network interface:

```
roslaunch naoqi_driver naoqi_driver.launch
nao_ip:=172.29.111.230 network_interface:=enp0s3
```

Note: Replace enp0s3 with your actual network interface name if different.

6.1.3 Running the KinyarwandaTTS Node

1. Open a new terminal tab or window

2. Source the setup file again in the new terminal:

```
source devel/setup.bash
```

3. Launch the KinyarwandaTTS node:

```
rosrun kinyarwanda_tts tts_node.py
```

6.1.4 Testing the System

To test the TTS functionality, open a third terminal and publish a text message to the `/text_to_say` topic:

1. For a simple greeting in Kinyarwanda:

```
rostopic pub /text_to_say std_msgs/String "data: 'Mwiriwe neza.'"
```

6.1.5 Troubleshooting

- Ensure the robot is powered on and connected to the same network as your computer
- Verify that NAOqi services are running on the robot
- Check network connectivity with `ping 172.29.111.230`
- Review ROS logs for error messages: `rosrun rqt_console rqt_console`

7 Kinyarwanda TTS Unit Tests

This package contains unit tests for the Kinyarwanda Text-to-Speech (TTS) component for Pepper robots.

7.0.1 Overview

The Kinyarwanda TTS component converts text in Kinyarwanda language to speech and plays it on the Pepper robot. This package provides two testing environments:

1. Physical Robot Testing
2. Test Harness Testing

7.0.2 Requirements

ROS Noetic ubuntu 20.04, Python 3.9 (for the TTS node), Python 2.7 (for the Pepper robot interface), Coqui TTS library, NAOqi SDK, and SSH access to the Pepper robot (for physical robot testing)

7.1 Testing Environments

7.1.1 Physical Robot Testing

The `kinyarwanda_tts_launch_robot.launch` file connects the TTS component to the physical Pepper robot.

Data Source: Text messages published to the `/text_to_say` topic
Data Sink: Pepper robot's speakers via NAOqi's ALAudioPlayer
Expected Behavior:

- When text is published to the `/text_to_say` topic, the TTS component should:
    1. Convert the text to speech using the Coqui TTS model
    2. Save the audio to a temporary WAV file
    3. Transfer the WAV file to the Pepper robot via SSH
    4. Play the audio on the robot
    5. Delete the temporary file after playback

Running the test:

```
roslaunch kinyarwanda_tts kinyarwanda_tts_launch_robot.launch
```

7.1.2 Test Harness Testing

The `kinyarwanda_tts_launch_test_harness.launch` file connects the TTS component to mock input and output drivers.

Data Source: Predefined test text messages from a test data publisher
Data Sink: Mock NAOqi driver that validates audio files without actual playback
Expected Behavior:

- The test data publisher should send a series of test messages
- For each message, the TTS component should:
  1. Convert the text to speech using the Coqui TTS model
  2. Save the audio to a file in the test_outputs directory
  3. The audio validator should verify:
     - Audio file is created successfully
     - Audio duration is appropriate for the text length
     - Audio file can be played successfully (without actually playing)

Running the test:

```
roslaunch kinyarwanda_tts kinyarwanda_tts_launch_test_harness.launch
```

7.2 Configuration Validation

Parameter	Default	Effect When Changed
model_path	`/home/muhirwa/tts_ws/src/kinyarwanda_tts/model_files/model.pth`	Using a different model will change the voice quality and pronunciation
speaker_wav	`/home/muhirwa/tts_ws/src/kinyarwanda_tts/model_files/conditioning_audio.wav`	Using a different conditioning audio will change the voice characteristics
use_cuda	false	Setting to true will enable GPU acceleration, resulting in faster synthesis
robot_ip	172.29.111.230	Changing this will connect to a different robot
robot_port	9559	Changing this may be necessary for different robot configurations
temp_dir	/tmp	Changing this affects where temporary audio files are stored

Table 1: Configuration Parameters and Their Effects

To test configuration changes:

- Modify the parameters in `kinyarwanda_tts.ini`

References

[1] 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.

[2] 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.

Principal Contributors

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

David Vernon, Carnegie Mellon University Africa.
Kleber Kabanda, Carnegie Mellon University Africa.
Richard Muhirwa, Carnegie Mellon University Africa.

Document History

Version 1.0

First draft.
Richard Muhirwa.
02 March 2025.

Version 1.1

Changes made in Section 2: Requirements Definition and Section 5: Module Design to incorporate the feedback received from Kleber.
Richard Muhirwa.
17 March 2025.