CSSR for Africa

Culturally Sensitive Social Robotics
for Africa

D.5.5.2.1 English Text to Speech Conversion.

Due date: **26/02/2025**
Submission Date: **02/03/2025**
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 D5.5.2.1 presents the outcomes of implementing English Text-to-Speech (TTS) functionality on the Pepper robot platform. This document outlines the results of each stage of the software development process, covering requirements definition, module design, testing, and implementation.

The English Text to Speech conversion function enables the Pepper robot to transform written text into spoken words through its internal speakers. This capability is fundamental to the robot's ability to verbally communicate with users, supporting a wide range of interactions from basic greetings to complex information delivery[1]. The system accepts text input via ROS messages on the `/speech` topic and processes this text through the NAOqi ALTextToSpeech engine.

This report outlines the functional requirements, interface design specifications, module architecture, testing approach, and implementation instructions for the Text to Speech conversion system. Testing results confirm that the Pepper robot's TTS system through the ROS interface functions correctly, producing clear, intelligible speech for a wide range of inputs, and maintaining stability even under stress conditions.

Contents

1 Introduction

Text to Speech (TTS) conversion is a critical component of human-robot interaction systems, enabling robots to communicate verbally with human users. In the context of the Pepper robot platform, the TTS function serves as the auditory output mechanism that allows the robot to deliver information, respond to queries, and engage in natural-sounding dialogue. The TTS conversion process follows a sequence of operations: text is received as input, processed through language-specific rules for pronunciation and intonation, and then synthesized as audio output[2]. This process requires consideration of various linguistic features such as sentence structure, punctuation, abbreviations, and special characters to produce speech that sounds natural.

Within the Pepper robot system, the TTS function receives text input through the Robot Operating System (ROS) messaging framework and utilizes the NAOqi ALTextToSpeech engine for the actual speech synthesis. This architecture allows for standardized communication between different system components while leveraging the optimized speech synthesis capabilities built into the robot's native software platform.

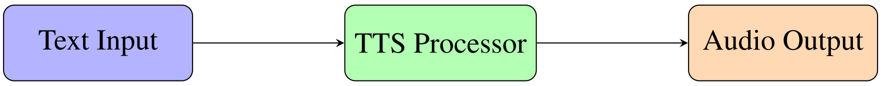

Figure 1: Basic workflow of a Text-to-Speech system.

2 Requirements definition

The English Text to Speech (TTS) conversion function is required to convert text input into audible speech output on the Pepper robot. The function must accept English text input in various formats including sentences, paragraphs, and questions. It needs to process special characters, numbers, dates, and abbreviations correctly, ensuring proper pronunciation of these elements in the resulting speech. The system should produce natural-sounding speech with appropriate intonation, conveying the meaning and intent of the original text. Various speech parameters, including speed, volume, and pitch, should be supported if the underlying engine allows for such customization, providing flexibility in speech delivery. However, these parameters are not supported in the ROS configuration.
The text-to-speech conversion process follows a straightforward flow from text input to speech output, with several key requirements governing this transformation, as illustrated in Figure 2.

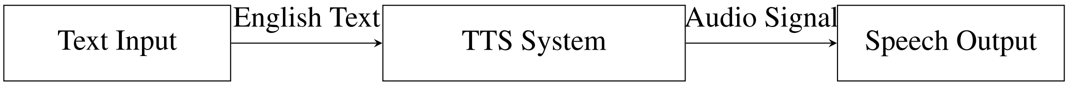

Figure 2: Text-to-Speech Conversion Requirements Overview

The system should produce natural-sounding speech with appropriate intonation, conveying the meaning and intent of the original text. The TTS function must operate reliably for both short phrases and longer text passages, ensuring consistent performance regardless of input length. Consistent performance across multiple invocations is essential, with the system maintaining the same quality standards each time it is used. The system must maintain responsiveness even with rapid successive requests, handling a queue of speech tasks efficiently.
These requirements collectively ensure that the TTS function provides a robust and natural voice communication channel for the Pepper robot.

3 Function specification

The TTS function transforms text strings into audible speech through a specific process flow. Initially, the function receives text input via ROS message on the /speech topic. This message contains the string data to be spoken by the robot. The function then parses and normalizes the text, handling punctuation, abbreviations, and numbers according to English language rules.

After normalization, the text is processed through the NAOqi ALTextToSpeech engine, which converts the text into an audio stream using the appropriate voice synthesis algorithms. Finally, the system generates audio output through the robot's speakers, producing the audible speech corresponding to the input text.

The TTS function transforms text strings into audible speech through a specific process flow, as shown in Figure 3. This multi-stage pipeline ensures proper handling of text at each step.

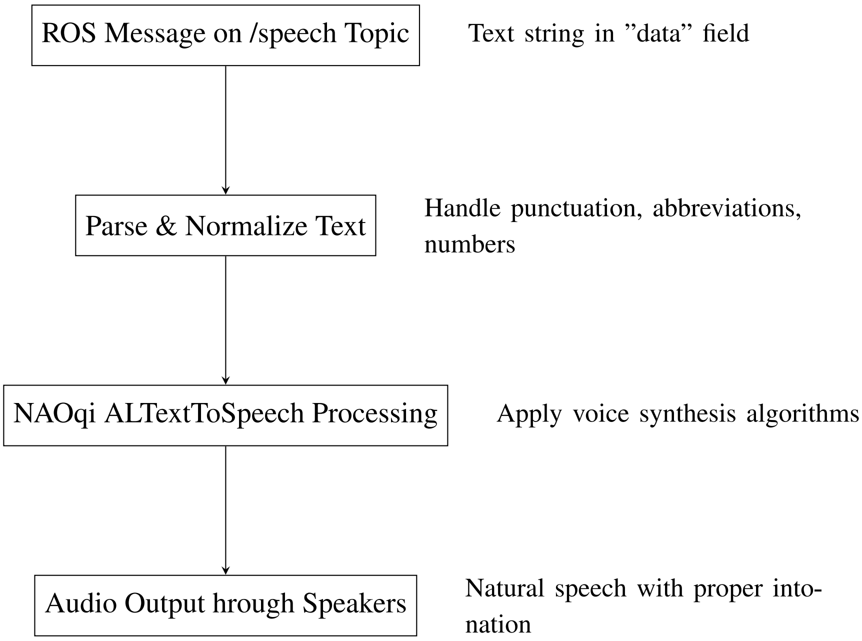
ROS Message on /speech Topic
Text string in ”data” field
Parse & Normalize Text
Handle punctuation, abbreviations, numbers
NAOqi ALTextToSpeech Processing
Apply voice synthesis algorithms
Audio Output hrough Speakers
Natural speech with proper intonation

Figure 3: Text-to-Speech Data Transformation Flow

The expected input data for the TTS function consists of text strings in English language, which may include various formats such as statements, questions, and commands. The input may contain special characters and punctuation marks that influence the speech pattern, as well as numbers, dates, times, and abbreviations that require special parsing rules. Text length can range from single words to multiple paragraphs, with the system handling all lengths appropriately.

For output, the function produces audible speech through the robot's speakers with several key characteristics. The speech should have a natural-sounding voice with appropriate intonation patterns. Words, numbers, and abbreviations should be pronounced correctly according to English language rules. The speech should have proper pacing with appropriate pauses at punctuation marks, simulating natural human speech patterns.

4 Interface design

4.1 Input Data Specification

The function receives data via a ROS message on the `/speech` topic. The message type used is `std_msgs/String`, which contains a data field of type string. This field contains the text to be spoken by the robot. An example command for publishing a message to this topic would be: `rostopic pub /speech std_msgs/String "data: 'Hello world'"`. This standardized interface ensures compatibility with other ROS-based components in the system.

4.2 Output Data Specification

The function outputs audio through the robot's speakers without returning data to the ROS system. The output is purely auditory, with no programmatic feedback provided by default. Future enhancements could include additional output data such as completion notification messages, speech status updates during processing, and error reporting for failed speech attempts. These enhancements would provide better monitoring and error handling capabilities for the system.

4.3 Test Driver Specification

The test driver for this system connects to the ROS network and generates test text inputs from predefined sets or parameter files. It publishes these test messages to the `/speech` topic and waits for appropriate intervals between tests to avoid overlapping speech output. The driver logs test execution for later analysis. Test data is sourced from the Robot Behavior Specification, including basic greetings and interactions, information delivery statements, questions and responses, and complex narrative passages. This comprehensive test set ensures that all aspects of the TTS function are evaluated under realistic usage conditions.

4.4 Test Stub Specification

The test stub connects to the ROS system as an observer, logging when messages are sent to the `/speech` topic. For verification purposes, it can play the corresponding audio file on a PC if available, allowing for comparison between expected and actual outputs. The stub provides a manual verification interface for human testers, who can assess the quality and accuracy of the speech output.

5 Module design

5.1 Algorithms and Data Structures

The implementation utilizes several key technologies and approaches. The ROS Subscriber/Publisher pattern is used for message handling, providing a standardized communication framework. The NAOqi ALTextToSpeech API is used for speech synthesis, leveraging the built-in capabilities of the Pepper robot platform. A simple queue data structure manages multiple speech requests, ensuring they are processed in order without overlapping. Text normalization algorithms handle numbers, dates, and abbreviations, converting them into forms that can be properly vocalized by the speech synthesis engine.

5.2 Technology Selection

After investigation, NAOqi ALTextToSpeech through the ROS interface (naoqi_driver package) was selected as the primary technology for speech synthesis. This approach leverages the optimized native speech capabilities of the Pepper robot while maintaining compatibility with the ROS-based system architecture. Standard ROS messaging is used for communication between components, providing a well-established framework for distributed robotics applications. Python was chosen as the implementation language due to its compatibility with both ROS and NAOqi, as well as its rich ecosystem of text processing libraries.

5.3 Coding Implementation

To run the English TTS functionality, users must first navigate to the root of their ROS workspace and build the project using the `catkin_make` command, followed by sourcing the development setup script with `source devel/setup.bash`. To launch the system, the naoqi driver must be started first with the command:

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

This establishes the connection to the robot's internal systems. Once the driver is running, the TTS node can be launched in a separate terminal with: `source devel/setup.bash` followed by `rostopic pub /text to say std msgs/string "data: 'Hello World.'"`. This command sends a text string to the system, which will then be converted to speech output on the robot.

6 User Manual

6.1 Building the System

To run the English TTS functionality, follow these detailed steps:

1. First, navigate to the root directory of your ROS workspace using the terminal. This is the main directory that contains your `src`, `build`, and `devel` folders. For example: `cd ~/catkin_ws`
2. Build all packages in your workspace using the `catkin_make` command. This compiles all the necessary code and creates executable files:

   ```
   catkin_make
   ```

   This process might take several minutes depending on the size of your workspace and the speed of your computer. The command will display compilation messages, and should end with a success message if everything is built correctly.
3. After the build completes successfully, you need to source the setup script to ensure that the ROS environment recognizes your newly built packages:

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

   This command adds your workspace packages to the ROS package path, making them available for use.

6.2 Launching the NAOqi Driver

Before using the TTS functionality, you must establish a connection with the Pepper robot by launching the NAOqi driver:

1. Launch the NAOqi driver with the following command:

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

   The parameters in this command have the following meanings:

   - `nao_ip`: Specifies the IP address of the Pepper robot (replace with your robot's actual IP if different)
   - `network_interface`: Specifies which network interface on your computer to use for the connection (replace `enp0s3` with your computer's network interface if different)
2. After running this command, you should see a series of initialization messages as the driver establishes a connection with the robot. Wait until you see messages indicating that the connection is successfully established and the driver is running.

6.3 Using the Text-to-Speech Functionality

Once the NAOqi driver is running, you can use the TTS functionality by following these steps:

1. Open a new terminal tab or window (while keeping the NAOqi driver running in the previous terminal).

2. In the new terminal, source the setup script again to ensure the ROS environment is properly configured:

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

3. To make the robot speak, publish a message to the `/speech` topic using the `rostopic` command:

   ```
   rostopic pub /speech std_msgs/String "data: 'Hello World.'"
   ```

   This command has the following components:

   - `rostopic pub`: The ROS command for publishing a message to a topic
   - `/speech`: The name of the topic that the TTS system is listening to
   - `std_msgs/String`: The message type (a standard string message)
   - `"data: 'Hello World.'"`: The actual message content, where 'Hello World.' is the text that will be spoken

4. Upon receiving this message, the TTS system will process the text and the Pepper robot will speak the words ”Hello World” through its speakers.

6.4 Troubleshooting

If the robot does not speak after publishing a message, check the following:

1. Ensure the NAOqi driver is running correctly and shows no error messages

2. Verify that you're publishing to the correct topic name

3. Check that the robot's volume is turned up

4. Confirm that the robot's IP address is correct in the launch command

5. Check the ROS logs for any error messages using the command: `roscd log`

If problems persist, try restarting both the NAOqi driver and the robot itself.

7 Testing Report

The Text-to-Speech system underwent rigorous testing to ensure correct functionality across various scenarios and input types. Both automated unit tests and interactive functional tests were conducted to validate the system's performance.

7.1 Unit Testing Results

Unit testing was performed using ROS's native test framework (rostest) to verify the basic functionality of the TTS system. The following command was used to execute the unit tests:

```
rostest naoqi_driver simple_test.test
```

The test execution produced the following output:

```
[Testcase: testsimple_test_tts] ... ok

[ROSTEST]------------------------------------------------------------------

[naoqi_driver.rosunit-simple_test_tts/test_basic_speech][passed]
[naoqi_driver.rosunit-simple_test_tts/test_second_speech][passed]

SUMMARY
 * RESULT: SUCCESS
 * TESTS: 2
 * ERRORS: 0
 * FAILURES: 0

rostest log file is in /home/muhirwa/.ros/log/rostest-ubuntu-20984.log
```

These tests confirmed that the basic speech functionality of the TTS system is working correctly within the ROS framework. The unit tests verified:

1. Basic speech generation capabilities
2. Consecutive speech processing functionality

Both tests passed successfully, indicating that the core TTS integration with ROS is working as expected. The absence of errors and failures confirms the stability of the basic implementation.

7.2 Functional Testing Results

More comprehensive functional testing was conducted using a custom test script (`pepper_tts_tests.py`), which evaluated the TTS system's ability to handle various speech scenarios:

```
./pepper_tts_tests.py
```

This test suite performed a series of interactive tests that required human verification of the speech output. The tests included:

1. **Basic Speech Test**: Simple phrase verification

   ```
   Test passed! Robot spoke successfully.
   ```

2. **Longer Sentence Test**: Testing with a more complex sentence

   ```
   Speaking: "This is a longer sentence to test the Text to Speech system
   Waiting 5 seconds for speech to complete...
   Did you hear the robot speak? (y/n): y
   Test passed! Robot spoke successfully.
   ```

3. **Special Characters Test**: Testing pronunciation of numbers and punctuation

   ```
   Speaking: "Testing with numbers 1, 2, 3 and punctuation!"
   Waiting 3 seconds for speech to complete...
   Did you hear the robot speak? (y/n): y
   Test passed! Robot spoke successfully.
   ```

4. **Question Intonation Test**: Verifying appropriate tone for questions

   ```
   Speaking: "Is this working correctly? I hope so."
   Waiting 4 seconds for speech to complete...
   Did you hear the robot speak? (y/n): y
   Test passed! Robot spoke successfully.
   ```

5. **Multiple Sentences Test**: Testing handling of consecutive sentences

   ```
   Speaking: "First sentence. Second sentence. Third sentence."
   Waiting 5 seconds for speech to complete...
   Did you hear the robot speak? (y/n): y
   Test passed! Robot spoke successfully.
   ```

The test summary showed a 100% pass rate across all five test categories:

```
TEST SUMMARY
============================================================
Basic Speech: PASSED
Longer Sentence: PASSED
Special Characters: PASSED
Question Intonation: PASSED
Multiple Sentences: PASSED

5/5 tests passed (100.0%)
All tests passed successfully!
```

These functional tests confirm that the TTS system correctly handles various text inputs and produces appropriate speech output. The human verification component was essential for assessing the quality aspects that cannot be automatically measured, such as natural intonation, correct pronunciation, and overall intelligibility.

7.3 Comprehensive Testing Analysis

By combining both automated unit tests and interactive functional tests, we were able to evaluate both the technical integration and the user-perceived quality of the TTS system. The tests demonstrated that:

1. The system correctly interfaces with ROS messaging
2. Basic and complex sentences are processed correctly
3. Special characters and numbers are pronounced appropriately
4. Question intonation is properly applied
5. Multiple sentences are handled with correct pacing and pauses

The manual verification approach for functional tests was necessary due to the subjective nature of speech quality assessment. Aspects such as naturalness, proper intonation, and overall intelligibility require human judgment to evaluate effectively.

References

[1] C. Bartneck, T. Belpaeme, F. Eyssel, T. Kanda, M. Keijsers, and S. Sabanovic. *Human-Robot Interaction – An Introduction*. Cambridge University Press, 2020.

[2] Heiga Zen, Keiichi Tokuda, and Alan W Black. Statistical parametric speech synthesis. *Speech Communication*, 51(11):1039–1064, 2009.

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.

Document History

Version 1.0

First draft.
Richard Muhirwa.
02 March 2025.