2. Offline Text-to-Speech (TTS)

---

1. Introduction

1.1 What is “TTS”?

TTS technology is a technology that converts written text into human-readable speech output. It enables computers to “read” text content and is widely used in various fields such as accessible reading, intelligent assistants, navigation systems, and educational software. Through TTS, users can hear natural and fluent human voices generated by machines, greatly improving the convenience and flexibility of information acquisition.

1.2 Brief Description of Implementation Principles

The implementation of a TTS system mainly includes the following key steps and technologies:

1. Text Analysis

- In this stage, the input text is first preprocessed, including but not limited to removing irrelevant characters, standardizing punctuation and capitalization, word segmentation, and recognizing special symbols such as numbers and converting them into their corresponding word forms.
- Simultaneously, linguistic analysis of the text is required, such as determining the pronunciation of each word (which usually requires the use of a pronunciation dictionary), stress positions, intonation patterns, and sentence structure.

2. Language Processing

- This step primarily focuses on how to correctly pronounce and adjust intonation based on context. For example, the word "read" has different pronunciations in different tenses (past tense/past participle is pronounced /red/, while in other cases it is pronounced /riːd/). Therefore, a powerful language model is needed to understand these subtle differences.
- This also involves prosodic modeling, i.e., determining which parts should be emphasized, whether the speech rate should be fast or slow, and the overall emotional tone of the sentence.

3. Speech Synthesis

- The text information processed in the previous two stages is fed into a speech synthesis engine, which is responsible for generating the actual sound waveforms.
- Traditional TTS systems use a concatenation synthesis method, which selects appropriate units from a pre-recorded speech segment database and concatenates them to form a complete sentence. While this method can produce high-quality speech, it is limited by the samples in the database.
- Modern TTS systems rely more on parametric synthesis or neural network synthesis (such as WaveNet, Tacotron, etc.). These methods can directly predict speech features from text and generate continuous sound signals. Especially deep learning-based methods can better capture subtle changes in speech, thus producing more natural and fluent sounds.

4. Sound Waveform Generation

- Finally, the generated sound waveform is further processed to ensure its quality meets expected standards, such as adjusting volume and equalizing frequency response.
- Afterward, this audio data can be played back through speakers or other audio playback devices for people to listen to.

With the development of artificial intelligence and machine learning technologies, especially the application of deep learning, TTS systems have not only significantly improved in accuracy but have also made great strides in naturalness and emotional expression, making machine-generated speech increasingly closer to real human voices.

2. Code analysis

Key code

1. TTS initialization and calling (`largemodel/largemodel/model_service.py`)

```
# From largemodel/largemodel/model_service.py
class LargeModelService(Node):
    def __init__(self):
        # ...
        self.system_sound_init()
        # ...

    def init_param_config(self):
        # ...
        self.declare_parameter('useolinetts', False)
        self.useolinetts =
self.get_parameter('useolinetts').get_parameter_value().bool_value
        if self.useolinetts:
```

```
            self.tts_out_path = os.path.join(self.pkg_path, "resources_file",
"tts_output.mp3")
        else:
            self.tts_out_path = os.path.join(self.pkg_path, "resources_file",
"tts_output.wav")

    def system_sound_init(self):
        """Initialize TTS system"""
        model_type = "oline" if self.useolinetts else "local"
        self.model_client.tts_model_init(model_type, self.language)
        self.get_logger().info(f'TTS initialized with {model_type} model')

    def _safe_play_audio(self, text_to_speak: str):
        """
        Synthesizes and plays all non-empty messages only in non-text chat mode.
        """
        if not self.text_chat_mode and text_to_speak:
            try:
                self.model_client.voice_synthesis(text_to_speak,
self.tts_out_path)
                self.play_audio_async(self.tts_out_path)
            except Exception as e:
                self.get_logger().error(f"Safe audio playback failed: {e}")
```

2. TTS backend implementation (`largemodel/utils/large_model_interface.py`)

```
# From largemodel/utils/large_model_interface.py
class model_interface:
    # ...
    def tts_model_init(self,model_type='oline',language='zh'):
        if model_type=='oline':
            if self.tts_supplier=='baidu':
                self.token=self.fetch_token()

            self.model_type='oline'
        elif model_type=='local':
            self.model_type='local'
            if language=='zh':
                tts_model=self.zh_tts_model
                tts_json=self.zh_tts_json
            elif language=='en':
                tts_model=self.en_tts_model
                tts_json=self.en_tts_json
            self.synthesizer = piper.PiperVoice.load(tts_model,
config_path=tts_json, use_cuda=False)

    def voice_synthesis(self,text,path):
        if self.model_type=='oline':
            if self.tts_supplier=='baidu':
                # ... (Baidu TTS implementation)
                pass
            elif self.tts_supplier=='aliyun':
                # ... (Aliyun TTS implementation)
                pass
        elif self.model_type=='local':
            with wave.open(path, 'wb') as wav_file:
```

```
                wav_file.setnchannels(1)
                wav_file.setsampwidth(2)
                wav_file.setframerate(self.synthesizer.config.sample_rate)
                self.synthesizer.synthesize(text, wav_file)
```

Code Analysis

The text-to-speech (TTS) function is initiated by the `LargeModelService` node, and its implementation is provided by the `model_interface` class. Its design uses parameter configuration to switch between different backend services.

1. **Initialization Process (`model_service.py`)**:

- During the initialization of `LargeModelService`, the `init_param_config` function reads the boolean value `useolinetts` from the ROS parameter server.
- The `system_sound_init` function, based on the value of `useolinetts`, decides whether to pass the string `'local'` or `'oline'` to the `self.model_client.tts_model_init` method.
- In `large_model_interface.py`, the `tts_model_init` method executes the corresponding initialization logic based on the received string parameter. If it's `'local'`, then `piper.PiperVoice.load` loads the local model file.

2. **Synthesis and Playback Process (`model_service.py`):**

- When voice playback is needed, the `_safe_play_audio` function is called.
- This function first calls the `self.model_client.voice_synthesis` method, passing in the text to be converted and the target audio path `self.tts_out_path`.
- After the `voice_synthesis` method completes and generates the audio file, `_safe_play_audio` then calls `self.play_audio_async` to play the file asynchronously.

3. **Backend Implementation Selection (`large_model_interface.py`):**

- The `voice_synthesis` method is the backend distribution center for TTS functionality. Internally, it selects the execution path by checking the value of the `self.model_type` attribute set during initialization.
- If `self.model_type` is `'local'`, the code block uses Python's `wave` library to open a WAV file, sets its header parameters (channels, sample bit width, sample rate), and then calls the `self.synthesizer.synthesize` method to directly write the text-synthesized audio stream to that file.
- If `self.model_type` is `'oline'`, it will enter the logical branch corresponding to different cloud service providers (such as Baidu, Aliyun).
- This structure will call the upper-level node (This section separates the "saying this" statement from the underlying specific synthesis techniques (which library to use, which API to call).

3. Practical Operation

3.1 Configuring Offline TTS Functionality

To enable offline TTS, you need to correctly configure `yahboom.yaml` and `large_model_interface.yaml`, and ensure that the local model is correctly placed.

1. **Open the main configuration file**:

```
vim ~/yahboom_ws/src/largemodel/config/yahboom.yaml
```

2. **Modify/confirm the following key configurations**:

```
model_service:                          #Model server node parameters
  ros__parameters:
    language: 'en'                      #Large model interface language
    useolinetts: False                   #Whether to use online speech synthesis
(True uses online, False uses offline)
```

Useolinetts must be set to False here to use the local model.

Language selection: zh for Chinese, en for English.

3. **Open the model interface configuration file:**

```
vim ~/yahboom_ws/src/largemodel/config/large_model_interface.yaml
```

4. **Confirm the offline model path:**

```
# large_model_interface.yaml
## Offline Speech Synthesis (Offline TTS)
# Chinese TTS Model
zh_tts_model: "/home/sunrise/yahboom_ws/src/largemodel/MODELS/tts/zh/zh_CN-
huayan-medium.onnx"
zh_tts_json: "/home/sunrise/yahboom_ws/src/largemodel/MODELS/tts/zh/zh_CN-
huayan-medium.onnx.json"

# English TTS Model
en_tts_model: "/home/sunrise/yahboom_ws/src/largemodel/MODELS/tts/en/en_US-
libritts-high.onnx"
en_tts_json: "/home/sunrise/yahboom_ws/src/largemodel/MODELS/tts/en/en_US-
libritts-high.onnx.json"
```

Recompile

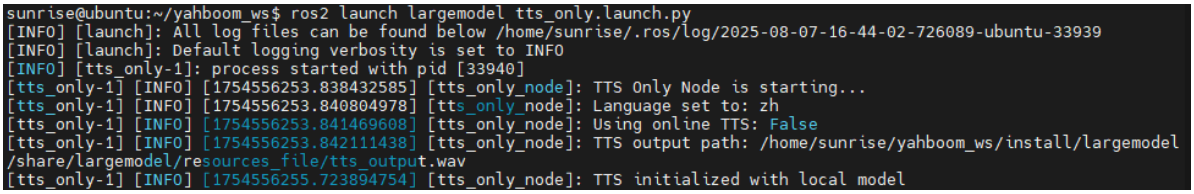
```
cd ~/yahboom_ws/
colcon build
source install/setup.bash
```

3.2 Startup and Testing

1. **Start the TTS node**:

Run the following command:

```
ros2 launch largemodel tts_only.launch.py
```



2. **Sending the Text to be Synthesized**:

Open a new terminal and run the following command to publish a voice message:

```
ros2 topic pub --once /tts_text_input std_msgs/msg/String '{data: "语音合成测试成功"}'
```

3. **Testing**:

If everything is normal, you should be able to hear the robot say "Speech synthesis test successful" in synthesized speech from your speaker.

---

4. Common Problems and Solutions

4.1 Playback Issues

Problem 1: The program runs normally without errors, but no sound is heard.

Solution:

1. **Check Audio Output:** Ensure your system's audio output device is selected correctly and the volume is not muted. You can try playing a regular music file to test the hardware.
2. In the Sound settings, keep the Balance in the middle position.