1. Offline Speech-to-Text (ASR)

---

1. Introduction

1.1 What is “ASR”?

ASR (Automatic Speech Recognition) is a technology that converts human speech signals into text. It is widely used in intelligent assistants, voice command control, automated telephone customer service, real-time caption generation, and other fields. The goal of ASR is to enable machines to “understand” human language and convert it into a form that computers can process and understand.

1.2 Implementation Principles

The implementation of an ASR system mainly relies on the following key technical components:

1. Acoustic Model

- The acoustic model is responsible for converting the input sound signal into phonemes or sub-word units. This typically involves feature extraction steps, such as using Mel-frequency cepstral coefficients (MFCCs) or filter banks to represent the audio signal.
- These features are then fed into deep neural networks (DNNs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), or more advanced Transformer architectures for training to learn how to map audio features to corresponding phonemes or characters.

2. Language Model

- A language model is used to predict the next most likely word given context, thus helping to improve recognition accuracy. It is trained on large amounts of text data to understand which word sequences are more likely to occur.
- Common language models include n-gram models, RNN-based LMs, and the recently popular Transformer-based LMs.

3. Pronunciation Dictionary

- A pronunciation dictionary provides a mapping between words and their corresponding pronunciations. This is crucial for connecting the acoustic model and the language model, as it allows the system to understand and match heard sounds based on known pronunciation rules.

4. Decoder

- The decoder's task is to find the most likely word sequence as output, given the acoustic model, the language model, and the pronunciation dictionary. This process typically involves complex search algorithms, such as the Viterbi algorithm or graph-based search methods, to find the optimal path.

5. End-to-End ASR

- With the development of deep learning, end-to-end ASR systems have emerged. These systems attempt to learn text output directly from raw audio signals without requiring explicit separate designs for acoustic models, pronunciation dictionaries, and language models. These systems are often based on sequence-to-sequence (Seq2Seq) frameworks, such as using attention mechanisms or Transformer architectures, greatly simplifying the complexity of traditional ASR systems.

In summary, modern ASR systems achieve efficient and accurate human speech-to-text conversion by combining the above components and training with large-scale datasets and powerful computing resources. With technological advancements, the performance of ASR systems continues to improve, and their application scenarios are becoming increasingly widespread.

---

2. Code Analysis

Key Code

1. Speech Processing and Recognition Core (`largemodel/largemodel/asr.py`)

```
# From largemodel/largemodel/asr.py
def kws_handler(self)->None:
    if self.stop_event.is_set():
        return

    if self.listen_for_speech(self.mic_index):
        asr_text = self.ASR_conversion(self.user_speechdir)  # Perform ASR
conversion
        if asr_text =='error':   Check if ASR result length is less than 4
characters
```

```
            self.get_logger().warn("I still don't understand what you mean.
Please try again")
            playsound(self.audio_dict[self.error_response])  #  Error response
        else:
            self.get_logger().info(asr_text)
            self.get_logger().info("okay😜, let me think for a moment...")
            self.asr_pub_result(asr_text)  # Publish ASR result
    else:
        return

def ASR_conversion(self, input_file:str)->str:
    if self.use_oline_asr:
        result=self.modelinterface.oline_asr(input_file)
        if result[0] == 'ok' and len(result[1]) > 4:
            return result[1]
        else:
            self.get_logger().error(f'ASR Error:{result[1]}')  # ASR error.
            return 'error'
    else:
        result=self.modelinterface.SenseVoiceSmall_ASR(input_file)
        if result[0] == 'ok' and len(result[1]) > 4:
            return result[1]
        else:
            self.get_logger().error(f'ASR Error:{result[1]}')  # ASR error.
            return 'error'
```

2. VAD Smart Recording (`largemodel/largemodel/asr.py`)

```
# From largemodel/largemodel/asr.py
def listen_for_speech(self,mic_index=0):
    p = pyaudio.PyAudio()   # Create PyAudio instance.
    audio_buffer = []       # Store audio data.
    silence_counter = 0     # Silence counter.
    MAX_SILENCE_FRAMES = 90  #  Stop after 900ms of silence (30 frames * 30ms)
    speaking = False  # Flag indicating speech activity.
    frame_counter = 0  # Frame counter.
    stream_kwargs = {
        'format': pyaudio.paInt16,
        'channels': 1,
        'rate': self.sample_rate,
        'input': True,
        'frames_per_buffer': self.frame_bytes,
    }
    if mic_index != 0:
        stream_kwargs['input_device_index'] = mic_index

    # Prompt the user to speak via the buzzer.
    self.pub_beep.publish(UInt16(data = 1))
    time.sleep(0.5)
    self.pub_beep.publish(UInt16(data = 0))

    try:
        # Open audio stream.
        stream = p.open(**stream_kwargs)
        while True:
            if self.stop_event.is_set():
                return False
```

```
            frame = stream.read(self.frame_bytes, exception_on_overflow=False)
 # Read audio data.
            is_speech = self.vad.is_speech(frame, self.sample_rate)  # VAD
detection.

            if is_speech:
                # Detected speech activity.
                speaking = True
                audio_buffer.append(frame)
                silence_counter = 0
            else:
                if speaking:
                    # Detect silence after speech activity.
                    silence_counter += 1
                    audio_buffer.append(frame)  # Continue recording buffer. 冲。

                    # End recording when silence duration meets the threshold.
                    if silence_counter >= MAX_SILENCE_FRAMES:
                        break
            frame_counter += 1
            if frame_counter % 2 == 0:
                self.get_logger().info('1' if is_speech else '-')
                # Real-time status display.
    finally:
        stream.stop_stream()
        stream.close()
        p.terminate()

    # Save valid recording (remove trailing silence).
    if speaking and len(audio_buffer) > 0:
        # Trim the last silent part.
        clean_buffer = audio_buffer[:-MAX_SILENCE_FRAMES] if len(audio_buffer) >
MAX_SILENCE_FRAMES else audio_buffer

        with wave.open(self.user_speechdir, 'wb') as wf:
            wf.setnchannels(1)
            wf.setsampwidth(p.get_sample_size(pyaudio.paInt16))
            wf.setframerate(self.sample_rate)
            wf.writeframes(b''.join(clean_buffer))
            return True
```

Code Analysis

The ASR (Speech-to-Text) function is provided by the `ASRNode` node (`asr.py`). This node is responsible for recording, converting, and publishing audio.

1. **Audio Recording (`listen_for_speech`):**

- This function uses the `pyaudio` library to capture an audio stream from the microphone.
- It integrates the `webrtcvad` library for Voice Activity Detection (VAD). The function loops through audio frames and uses `vad.is_speech()` to determine if each frame contains human voice.
- When speech is detected, data is written to a buffer. Recording stops when sustained silence (defined by `MAX_SILENCE_FRAMES`) is detected.
- Finally, the audio data in the buffer is written to a `.wav` file at `self.user_speechdir`.

2. **Backend Selection and Execution (`ASR_conversion`):**

- The `kws_handler` function calls the `ASR_conversion` function after successful recording.
- This function determines which backend implementation to call by reading the ROS parameter `use_oline_asr` (a boolean value).
- If `false`, the `self.modelinterface.SenseVoiceSmall_ASR` method is called, corresponding to local recognition.
- If `true`, the `self.modelinterface.oline_asr` method is called, corresponding to online recognition.
- This function takes the audio file path as an argument to the selected method and processes the returned result.

3. **Result Publication (`asr_pub_result`):**

- After `ASR_conversion` returns valid text, `kws_handler` calls the `asr_pub_result` function.
- This function wraps the text string in a `std_msgs.msg.String` message and publishes it to the `/asr` topic via the ROS publisher.

3. Practical Operation

3.1 Configuring Offline ASR Functionality

To enable offline ASR, the `yahboom.yaml` file needs to be configured correctly, and the local model needs to be placed correctly.

1. **Open the configuration file:**

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

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

```
asr:                                    # Voice node parameters
  ros__parameters:
    # ...
    use_oline_asr: False                # Critical: Must be set to False to
enable offline ASR
    mic_serial_port: "/dev/ttyUSB0"     # Microphone serial port alias
    mic_index: 0                        # Microphone device index
    language: 'en'                      # ASR language, 'zh' or 'en'
```

Make sure use_oline_asr is False here to use the local model.

In the terminal, type `ls /dev/ttyUSB*` to check if the USB device number assigned to the voice module is USB0. If not, you can modify the 0 in the configuration file to change it to your own device number.

The language selection is zh for Chinese and en for English.

Additionally, you need to specify the path to the offline model in `large_model_interface.yaml`.

Open the file in the terminal:

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

Find the configuration related to local_asr_model

```
# large_model_interface.yaml
## Offline Speech Recognition (Offline ASR)
local_asr_model:
"/home/sunrise/yahboom_ws/src/largemodel/MODELS/asr/SenseVoiceSmall"
# Local ASR model path
```

Recompile

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

3.2 Startup and Testing

1. **Startup Command**:

```
ros2 launch largemodel asr_only.launch.py
```

2. **Test**:

Speak into the microphone:"Hello yahboom", it will respond with "I'm here", and then it can start speaking. Finally, it will display the recorded audio converted into text and printed on the terminal.