Reconocimiento de voz en tiempo real

Este código Python implementa el reconocimiento de voz en tiempo real utilizando la API de Google Cloud Speech-to-Text y la biblioteca PyAudio. Captura audio del micrófono, lo transmite a la API de Speech-to-Text e imprime el texto transcrito. La clase `MicrophoneStream` maneja la entrada de audio, y la función `main` configura el cliente de reconocimiento de voz y procesa la transmisión de audio.

```
import os
import argparse
import io
import sys
import time

from google.cloud import speech

import pyaudio
from six.moves import queue


# Parámetros de grabación de audio
RATE = 16000
CHUNK = int(RATE / 10)  # 100ms


class MicrophoneStream(object):
    """Abre una transmisión de grabación como un generador que produce los fragmentos de audio."""
    def __init__(self, rate, chunk):
        self._rate = rate
        self._chunk = chunk

        # Crea una interfaz de audio usando PyAudio
        self._audio_interface = pyaudio.PyAudio()
        self._audio_stream = self._audio_interface.open(
            format=pyaudio.paInt16,
            # La API actualmente solo admite audio de 1 canal (mono)
            # https://goo.gl/z726ff
            channels=1, rate=self._rate,
            input=True, frames_per_buffer=self._chunk,
            # Ejecuta la transmisión de audio de forma asincrónica para llenar el objeto de búfer.
            # Esto es necesario para que el búfer del dispositivo de entrada no
```

```
        # se desborde mientras el hilo que llama realiza solicitudes de red, etc.
        stream_callback=self._fill_buffer,
    )
    self.closed = False
    self._buff = queue.Queue()

def _fill_buffer(self, in_data, frame_count, time_info, status_flags):
    """Recolecta continuamente datos de la transmisión de audio, en el búfer."""
    self._buff.put(in_data)
    return None, pyaudio.paContinue

def generator(self, record_seconds):
    start_time = time.time()
    while not self.closed and time.time() - start_time < record_seconds:
        # Usa un get() de bloqueo para asegurar que hay al menos un fragmento de
        # datos, y detener la iteración si el fragmento es None, indicando el
        # final de la transmisión de audio.
        chunk = self._buff.get()
        if chunk is None:
            return
        data = [chunk]

        # Ahora consume cualquier otro dato que aún esté en búfer.
        while True:
            try:
                chunk = self._buff.get(block=False)
                if chunk is None:
                    return
                data.append(chunk)
            except queue.Empty:
                break

        yield b''.join(data)

def close(self):
    self.closed = True
    # Señaliza al generador que termine para que el método
    # de reconocimiento de transmisión del cliente no bloquee la terminación del proceso.
    self._buff.put(None)
    self._audio_stream.close()
```

```
        self._audio_interface.terminate()

    def __enter__(self):
        return self

    def __exit__(self, type, value, traceback):
        self.close()


def main(record_seconds=10, language_code='en-US'):
    # Consulta http://g.co/cloud/speech/docs/languages
    # para obtener una lista de idiomas compatibles.
    # language_code = 'en-US'  # una etiqueta de idioma BCP-47

    client = speech.SpeechClient()
    config = speech.RecognitionConfig(
        encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,
        sample_rate_hertz=RATE,
        language_code=language_code,
        model="latest_long",
    )

    streaming_config = speech.StreamingRecognitionConfig(
        config=config,
        interim_results=True)

    with MicrophoneStream(RATE, CHUNK) as stream:
        audio_generator = stream.generator(record_seconds)
        requests = (speech.StreamingRecognizeRequest(audio_content=content)
                    for content in audio_generator)

        responses = client.streaming_recognize(streaming_config, requests)


        # Ahora, usa las respuestas de transcripción.
        transcript = ""
        for response in responses:
            print(response)
            # Una vez que se realiza la transcripción, imprime el resultado.
            for result in response.results:
```

```
                if result.is_final:
                    alternative = result.alternatives[0]
                    transcript += alternative.transcript + " "
        print(u'Transcripción: {}'.format(transcript))


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Reconocimiento de voz en tiempo real con duración ajustable
    parser.add_argument('--duration', type=int, default=10, help="Duración de la grabación en segundos.")
    parser.add_argument('--language_code', type=str, default='en-US', help="Código de idioma para la transcrip
    args = parser.parse_args()
    print("Por favor, hable...")
    main(record_seconds=args.duration, language_code=args.language_code)
```