Assignment 24 - Report

The Assignment purpose is to enable device that can translate between multiple languages, allowing folks who speak different languages to be able to communicate. This universal translator using 2 IoT devices here will be used to :

- Record the microphone sound and turn audio to text using function get_voice_command

() function

```python
def get_voice_command(prompt="Say what  you want translate to other person ?"):
    with sr.Microphone() as source:
        print(prompt)
        speak(prompt)
        audio = recognizer.listen(source)
    try:
        command = recognizer.recognize_google(audio)
        print(f"You said: {command}")
        return command.lower()
    except sr.UnknownValueError:
        speak("Sorry, I didn't catch that.")
        return ""
```

- Send the text receive from the previous function to send to MQTT topic
- The MQTT topic will act as a middle man to save the conversation
- Both device will continuously listen to the MQTT topic and after receive each other message through function on_message() . When a command is heard, it will then be translated from German to English (or vice versa depending on user preference).

```python
def on_message(client, userdata, msg):
    try:
        stop_flag.set()  # STOP voice input loop immediately

        german_text = msg.payload.decode()
        print(f"\n📩 Received (German): {german_text}")

        # Translate German to English
        translation = translator.translate(german_text, src='de', dest='en')
        english_text = translation.text
        print(f"🌐 Translated (English): {english_text}")

        # Speak translation
        speak(f"The translation is: {english_text}")

        stop_flag.clear()  # RESUME voice input loop after speaking
```

- The translation will be supported by API from google translation

```python
from googletrans import Translator  # Add this import

# Initialize the translator
translator = Translator()
```

- In this main function below, the MQTT client is initialized and set up with the given client name. The callbacks for connection and incoming messages are assigned before connecting to the HiveMQ broker on the specified port. Once connected, the client starts running its network loop in a background thread (loop_start()), allowing it to process incoming messages asynchronously. Meanwhile, a separate voice_loop thread continuously listens for a voice command via the microphone using get_voice_command(). If a command is detected, it is sent to the MQTT topic by calling send_to_MQTT(). The main thread remains active in an infinite loop (with sleep delays) to keep the program running and responsive to MQTT events.

```python
def main():
    client_name = id + "translate_client"
    stop_flag.clear()
    mqtt_client = mqtt.Client(client_name)
    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message
    try:
        mqtt_client.connect("broker.hivemq.com", 1883, 60)
        mqtt_client.loop_start()  # Start MQTT in background
        # Start thread for continuous voice input and sending
        def voice_loop():
            while not stop_flag.is_set():
                command = get_voice_command()
                if command:
                    send_to_MQTT(mqtt_client, command)
                time.sleep(0.5)  # Optional: short pause between messages

        voice_thread = threading.Thread(target=voice_loop)
        voice_thread.start()

        # Keep the main thread alive to receive MQTT messages
        while True:
            time.sleep(1)
    except Exception as e:
        print("❌ MQTT connection error:", e)


if __name__ == "__main__":
    main()
```

Conclusion: The Code was able to perform a universal translator, converting speech detected by one device into speech played by another device in a different language using translation from Google API