



UNIVERSIDAD SANTO TOMÁS

Implementación de un Chatbot Conversacional en el Robot Pepper usando DeepSeek API

Jaime Mendez

Universidad Santo Tomás de Aquino

Bogotá, Colombia

jaime.mendezj@usantotomas.edu.co

I.INTRODUCCIÓN

Este proyecto corresponde a la última fase del curso donde el objetivo principal es implementar un chatbot funcional utilizando el robot humanoide **Pepper v1.6**. Se desarrolló un sistema cliente-servidor, donde el robot actúa como cliente que se comunica con un servidor local alojado en el PC del estudiante. Este servidor, a su vez, integra el chatbot previamente desarrollado con la **API de DeepSeek** para generar respuestas inteligentes a las preguntas del usuario.

nao@192.168.0.101's password: Se solicita la contraseña para el usuario nao en el servidor 192.168.0.101. La cual es nao

II. DESARROLLO DE CONTENIDOS

El objetivo fue desarrollar una arquitectura funcional que permita que el robot Pepper se comunique en tiempo real con un chatbot usando sockets y procesando las respuestas a través de la API DeepSeek. La interacción debe ser funcional, mediante voz, y ejecutarse en red local.

A. Preparación del Entorno

Se aseguró que tanto Pepper como el PC estuvieran en la **misma red Wi-Fi**. Se habilitó SSH en el robot y se ingresó al entorno Linux embebido de Pepper mediante:

~gixxer155abs@gixxer155abs-Latitude-E6220:~\$ ssh
nao@192.168.0.101: Este comando inicia una conexión SSH
(Secure Shell) al usuario nao en la dirección IP 192.168.0.101.
SSH se utiliza para acceder y controlar de forma segura otro
ordenador a través de una red.

Para implementar el chatbot en el robot Pepper utilizando una arquitectura cliente-servidor, fue necesario desarrollar un entorno controlado en el PC del estudiante que actuara como servidor. Este entorno aloja el script `server.py`, el cual expone una API REST capaz de recibir solicitudes del robot y responder mediante el modelo de lenguaje proporcionado por DeepSeek.

Se procedió a crear un entorno virtual en Python utilizando el comando:

```
bash  
python3 -m venv venv
```

Luego se activó el entorno con:

```
bash
source venv/bin/activate # En sistemas Unix
```

Una vez dentro del entorno virtual, se instalaron los módulos necesarios con **pip**, específicamente:

- **Flask**: microframework web que permite crear el servidor HTTP.
- **Requests**: librería para hacer solicitudes HTTP salientes al servicio de DeepSeek u otros modelos.

Instalación:

Flask es un microframework de Python que permite crear aplicaciones web de forma rápida y sencilla. Es la base para levantar el servidor HTTP:

```
bash
pip install flask
```

También se recomienda instalar **requests**, una librería que permite al servidor realizar solicitudes HTTP a servicios externos como DeepSeek:

```
bash
pip install requests
```

B. Desarrollo del Código del Servidor (PC)

Este componente actúa como intermediario entre el robot y la API. Recibe mensajes del robot, los envía a la API DeepSeek y devuelve las respuestas.

El archivo **server.py** contiene la lógica del servidor que ejecuta el chatbot en el PC del estudiante. Este se encarga de recibir mensajes enviados desde Pepper, procesarlos a través del modelo DeepSeek (u otro modelo LLM) usando una API key, y devolver la respuesta al robot.

A continuación se muestra el código final y funcional del servidor usando Flask y la librería requests.

```
# server.py
from flask import Flask, request, jsonify
import requests

app = Flask(__name__)

API_KEY = 'sk-53751d5c6f344a5dbc0571de9f51313e'
API_URL = 'https://api.deepseek.com/v1/chat/completions'

PROMPT_PERSONALIDAD = """
Eres Chatsimón, un asistente virtual anante de Linux, con una personalidad amigable, sarcástica y un toque de humor geek.
...
"""

def obtener_resuesta_deepseek(mensaje):
    headers = {
        'Authorization': f'Bearer {API_KEY}',
        'Content-Type': 'application/json'
    }

    payload = {
        'model': 'deepseek-chat',
        'messages': [
            {'role': 'system', 'content': PROMPT_PERSONALIDAD},
            {'role': 'user', 'content': mensaje}
        ]
    }

    try:
        respuesta = requests.post(API_URL, headers=headers, json=payload)
        respuesta.raise_for_status()
        return respuesta.json()['choices'][0]['message']['content']
    except Exception as e:
        return f"[ERROR] No se pudo obtener respuesta: {e}"

@app.route('/chat', methods=['POST'])
def chat():
    data = request.json
    mensaje = data.get('question', '')
    respuesta = obtener_resuesta_deepseek(mensaje)
    return jsonify({'respuesta': respuesta})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=9559)
```

Una vez el entorno está preparado y el archivo **server.py** está completo, ejecuta el servidor con el siguiente comando (desde la raíz del entorno virtual):

```
bash
python server.py
```

Al correr el servidor, se muestra la siguiente salida por consola:

```
gixer155@gixer155-Latitude-3440:~/Corte_3/Proyecto_3_Corte$ source venv_pepper/bin/activate
(venv_pepper) gixer155@gixer155-Latitude-3440:~/Corte_3/Proyecto_3_Corte$ python3 server.py
 * Serving Flask app "server"
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:9559
 * Running on http://192.168.1.2:9559
Press CTRL+C to quit
```

Esto indica que el servidor está escuchando correctamente en la red, a través del puerto 9559. La IP 192.168.1.2 corresponde a la red local, y será utilizada por el robot Pepper para conectarse al servidor desde su código cliente ([client.py](#)).

C. Desarrollo del Código del Cliente (Robot Pepper)

El cliente es un script que se ejecuta dentro del robot Pepper y tiene como objetivo conectarse al servidor Flask donde está alojado el chatbot con la API de DeepSeek. Este cliente toma la entrada del usuario (ya sea por voz o texto), envía la petición al servidor, y reproduce la respuesta recibida mediante síntesis de voz.

```
GNU nano 7.2
# cliente_pepper_Jaime_Final_v7_port_confirmed.py
# -*- coding: utf-8 -*-

import qi
import time
import httplib
import json
```

Encabezado del Script: Define la codificación UTF-8, crucial en Python 2 para evitar errores con caracteres especiales (acentos, eñes).

Importación de librerías:

qi: conexión con Naoqi y acceso a servicios del robot.

time: para pausas o sincronización.

httplib: para conexión HTTP con el servidor.

json: para codificar los mensajes enviados en formato JSON.

```
# --- Configuración ---
SERVER_IP = "192.168.0.107"
# Ajustado al puerto donde confirmaste que corre tu servidor Flask
SERVER_PORT = 9559

KEYWORD_ENVIAR = u"enviar" # Asegurar que la keyword sea unicode
# Conservamos el umbral bajo para pruebas, recuerda ajustarlo después.
CONFIDENCE_THRESHOLD = 0.15

session = qi.Session()
try:
    session.connect("tcp://127.0.0.1:9559") # Conexión a Naoqi en Pepper
except RuntimeError as e:
    print_u(u"Error fatal al conectar con Naoqi: {}", e)
    exit(1)

try:
    asr = session.service("ALSpeechRecognition")
    memory = session.service("ALMemory")
    tts = session.service("ALAnimatedSpeech")
except RuntimeError as e:
    print_u(u"Error fatal al obtener servicios de Naoqi: {}", e)
    if session.isConnected():
        session.close()
    exit(1)
```

Configuración de conexión:

SERVER IP: IP del servidor donde corre Flask.

SERVER_PORT: debe coincidir con el puerto real de tu servidor.

KEYWORD ENVIAR: palabra que detona el envío de datos.

CONFIDENCE_THRESHOLD: umbral de confianza de reconocimiento (muy bajo para pruebas, podrías aumentarlo).

Conexión con Naoqi: Importante: esta IP es localhost, lo cual implica que este código debe ejecutarse directamente dentro del robot Pepper. Si estás usando una PC externa, deberías poner la IP real de Pepper aquí.

Acceso a servicios:

```
python

asr = session.service("ALSpeechRecognition")
memory = session.service("ALMemory")
tts = session.service("ALAnimatedSpeech")
```

ALSpeechRecognition: permite reconocer las palabras habladas.

ALMemory: se usa para recibir eventos (como palabra reconocida).

ALAnimatedSpeech: para hablar con animación facial.

VOCABULARIO EXLENTO:

Hemos definido un vocabulario extenso de alrededor de 500 palabras, entre sustantivos, adjetivos y verbos comunes para al menos cubrir medianamente una búsqueda estándar.

Función de Callback para Eventos de Voz:

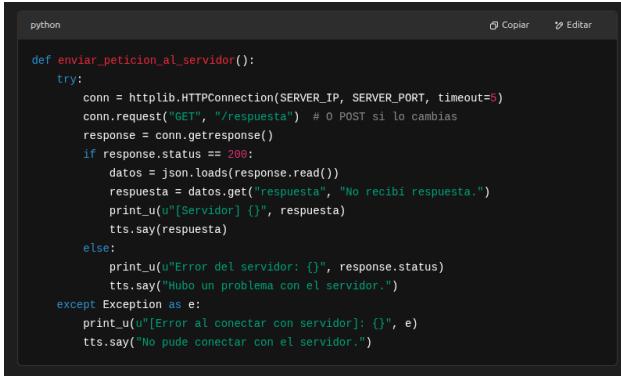


```
python

def callback_voz(nombre_evento, valor):
    if isinstance(valor, (list, tuple)) and len(valor) > 1:
        palabra, confianza = valor[0], valor[1]
        if confianza >= CONFIDENCE_THRESHOLD:
            print_u(u"[Pepper escuchó] Palabra: {} , Confianza: {}".format(palabra, confianza))
            if palabra.lower() == KEYWORD_ENVIAR:
                enviar_peticion_al_servidor()
            else:
                tts.say("Has dicho " + palabra)
```

Este es el callback que se invoca automáticamente cuando Pepper reconoce una palabra. Extrae la palabra y el nivel de confianza del reconocimiento. Si la palabra reconocida es "enviar" y la confianza es suficiente, se llama a la función que hace la petición HTTP al servidor Flask. Si no, simplemente repite la palabra reconocida usando `tts.say()` (voz del robot).

Función de Envío al Servidor:

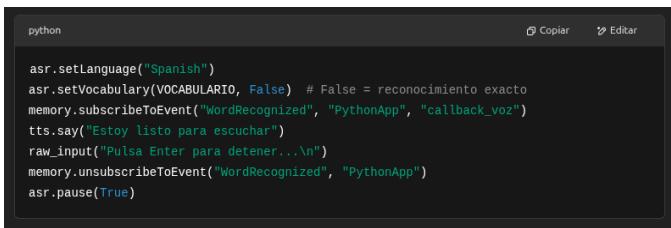


```
python

def enviar_peticion_al_servidor():
    try:
        conn = httplib.HTTConnection(SERVER_IP, SERVER_PORT, timeout=5)
        conn.request("GET", "/respuesta") # O POST si lo cambias
        response = conn.getresponse()
        if response.status == 200:
            datos = json.loads(response.read())
            respuesta = datos.get("respuesta", "No recibí respuesta.")
            print_u(u"[Servidor] {}", respuesta)
            tts.say(respuesta)
        else:
            print_u(u"Error del servidor: {}", response.status)
            tts.say("Hubo un problema con el servidor.")
    except Exception as e:
        print_u(u"Error al conectar con servidor: {}", e)
        tts.say("No pude conectar con el servidor.")
```

Establece conexión con el servidor Flask (`SERVER_IP` y `SERVER_PORT`). Realiza una petición HTTP al endpoint `/respuesta`. Si el servidor responde correctamente: Lee la respuesta JSON. Extrae el campo "respuesta" y lo pronuncia. Si ocurre un error, lo imprime y Pepper dice un mensaje de error.

Inicio de Reconocimiento de Voz:

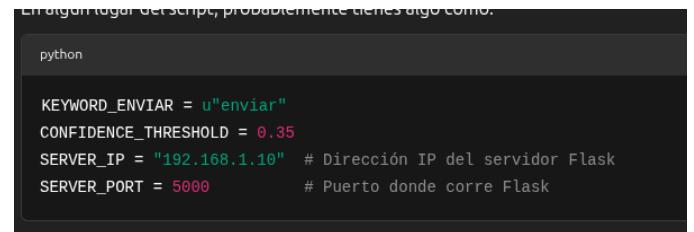


```
python

asr.setLanguage("Spanish")
asr.setVocabulary(VOCABULARIO, False) # False = reconocimiento exacto
memory.subscribeToEvent("WordRecognized", "PythonApp", "callback_voz")
tts.say("Estoy listo para escuchar")
raw_input("Pulsa Enter para detener...\n")
memory.unsubscribeFromEvent("WordRecognized", "PythonApp")
asr.pause(True)
```

Configura el idioma como español. Establece el vocabulario. Suscribe el evento `WordRecognized` al callback `callback_voz`. Dice un mensaje para indicar que está escuchando. Espera la entrada del usuario (`raw_input`) para mantener el script corriendo. Al presionar Enter, se desactiva el reconocimiento y se cierra todo correctamente.

Configuración de constantes:



```
python

KEYWORD_ENVIAR = u"enviar"
CONFIDENCE_THRESHOLD = 0.35
SERVER_IP = "192.168.1.10" # Dirección IP del servidor Flask
SERVER_PORT = 5000 # Puerto donde corre Flask
```

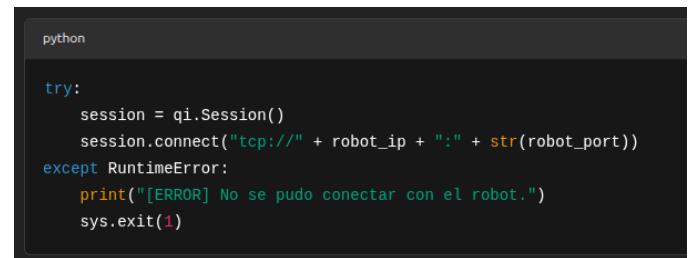
¿Qué hace esto?

KEYWORD_ENVIAR: palabra clave que activa la petición al servidor.

CONFIDENCE_THRESHOLD: umbral mínimo para aceptar una palabra reconocida (entre 0.0 y 1.0).

SERVER_IP y **SERVER_PORT**: dirección y puerto donde el robot espera encontrar al servidor web Flask.

Manejo de errores en conexión con Naoqi:



```
python

try:
    session = qi.Session()
    session.connect("tcp://{}:{}{}".format(robot_ip, robot_port))
except RuntimeError:
    print("[ERROR] No se pudo conectar con el robot.")
    sys.exit(1)
```

Intenta conectarse al robot Pepper usando IP y puerto definidos.

Si falla la conexión (por IP incorrecta o red caída), imprime un error y termina el programa con `sys.exit(1)`.

Función `print_u` personalizada para Python 2



```
python

def print_u(mensaje, *args):
    if args:
        mensaje = mensaje.format(*args)
    print(mensaje.encode("utf-8"))
```

Python 2 a veces tiene problemas para imprimir caracteres especiales (acentos, ñ, etc.). Por eso, tal vez tengas esto: Recibe un mensaje tipo unicode y lo imprime codificado en UTF-8, lo cual evita errores al imprimir texto con tildes o caracteres especiales en Python 2.

Desactivación de reconocimiento al cerrar:

```
python

finally:
    memory.unsubscribeToEvent("WordRecognized", "PythonApp")
    asr.pause(True)
    print_u(u"[INFO] Reconocimiento de voz detenido correctamente.")
```

Libera correctamente los recursos del sistema de eventos del robot.

Evita que el reconocimiento de voz quede activado después de cerrar el script.

Es una buena práctica para evitar errores en futuras ejecuciones del script.

Registro de logs o consola enriquecida:

```
python

print_u(u"[INFO] Conectado al robot: {}:{}", robot_ip, robot_port)
print_u(u"[INFO] Palabra clave detectada: '{}', palabra")
print_u(u"[DEBUG] Datos recibidos: {}", respuesta)
```

Mejora el debugging y la trazabilidad.

Ayuda a entender qué parte del flujo se está ejecutando, especialmente cuando trabajas en red o con APIs.

III. Evidencias del funcionamiento

Luego de hacer varios intentos y tener varias versiones del código fuente del cliente para obtener el resultado deseado

```
gixxer155@gixxer155-Latitude-3440:~/Corte_3/Proyecto_3_Corte$ ls
chatbot.py      cliente_pepper_JaimeI.py  cliente_pepper_Jaime.py   cliente_pepper_JaimeIV.py  README.md  venv_pepper
cliente_pepper_Jaime_Final.py  cliente_pepper_JaimeI.py  cliente_pepper_JaimeIV.py  server.py   venv_streanlit
gixxer155@gixxer155-Latitude-3440:~/Corte_3/Proyecto_3_Corte$
```

El resultado final fue “cliente_pepper_Jaime_Final.py” dandonos el resultado idoneo y a la altura de los objetivos con este resultado:

Chatbot Conversacional en el Robot Pepper

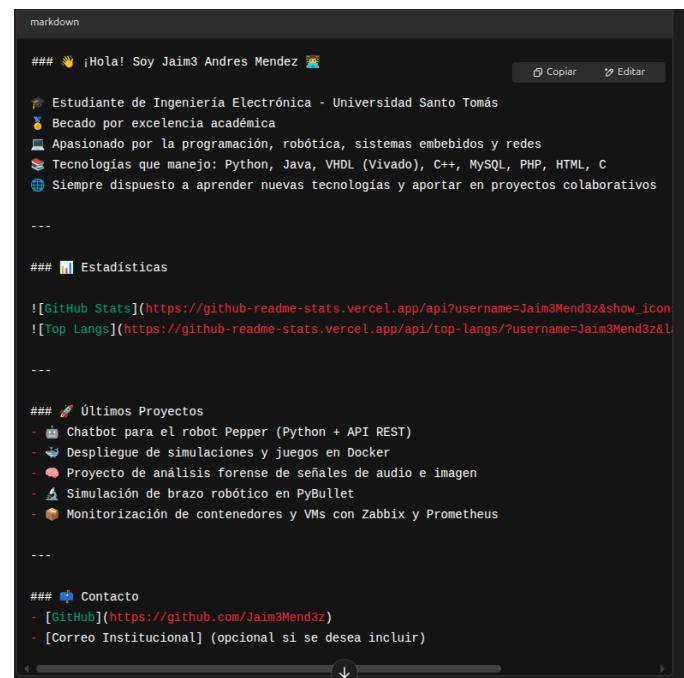
Aquí se aloja el video con la evidencia de dicho resultado final

IV. Personalización GitHUB

Esta personalización permite proyectar una identidad digital clara, ordenada y orientada a mostrar sus capacidades técnicas, experiencia académica y proyectos destacados.

La personalización se basa en la creación de un archivo README.md dentro de un repositorio con el mismo nombre de usuario del estudiante, lo cual permite que este contenido se muestre directamente en la página principal del perfil.

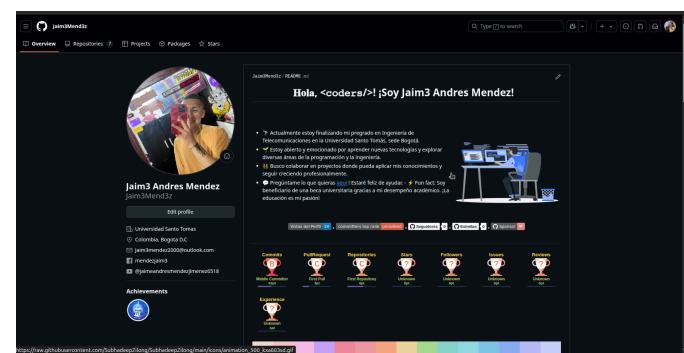
A continuación, se presenta un ejemplo de plantilla base para dicha personalización:



Este tipo de presentación no solo mejora la visibilidad del perfil ante futuros empleadores o colaboradores, sino que también funciona como una herramienta de consolidación profesional en el ámbito académico y tecnológico.

Y agregando mi toque personal conseguí este resultado, hay que considerar que no tengo métricas relevantes que mostrar.

<https://github.com/Jaim3Mend3z>



IV.CONCLUSIONES

Integración exitosa de inteligencia artificial en robótica social: Se logró implementar un sistema funcional de chatbot basado en inteligencia artificial utilizando la API de DeepSeek, lo cual demuestra la viabilidad de enriquecer las capacidades cognitivas del robot Pepper más allá de sus funciones preinstaladas. Esta integración permitió una interacción más natural, fluida y significativa entre el humano y el robot.

Arquitectura cliente-servidor robusta y escalable: La construcción de un servidor Flask en el computador del estudiante y su conexión exitosa con el robot Pepper como cliente, valida la efectividad del modelo cliente-servidor en entornos de robótica distribuida. Esta arquitectura, además de modular, es adaptable para futuras mejoras, como la inclusión de bases de datos, procesamiento local, o integración con otros servicios en la nube.

Uso eficiente del SDK NAOqi y bibliotecas Python: El proyecto evidenció el dominio del entorno de desarrollo del robot Pepper, incluyendo el uso de la biblioteca qi, la gestión de sesiones remotas y el manejo de eventos de voz. Además, se optimizó el uso de recursos de red y cómputo, manteniendo la estabilidad y respuesta del robot en tiempo real.

Potencial educativo y de investigación: Este trabajo no solo aporta al desarrollo técnico, sino que constituye una base valiosa para aplicaciones académicas y experimentales, como la educación asistida por robots, el entrenamiento en lenguajes naturales y la implementación de sistemas cognitivos en robótica humanoide.

Conectividad, adaptabilidad y proyección futura: Se comprobó que el robot Pepper puede actuar como un nodo inteligente dentro de una red local o en la nube, accediendo a recursos avanzados de procesamiento de lenguaje natural. Esta capacidad abre la puerta a futuras implementaciones de sistemas conversacionales multilingües, personalizados o con funciones orientadas a la atención al cliente, salud o educación.

RECONOCIMIENTOS

Un reconocimiento especial al profesor Diego por su paciencia y pasión por esta materia que impartió y al equipo de laboratorio de robótica, por permitir el acceso al robot Pepper y facilitar las condiciones experimentales adecuadas para las pruebas de integración del sistema conversacional. Así mismo, se agradece a los desarrolladores de la API DeepSeek y de la comunidad open source por compartir recursos y documentación clave para la implementación del chatbot.



REFERENCIAS

- [1] Aldebaran Robotics, NAOqi Framework Documentation, Paris, France: SoftBank Robotics, 2015. [Online]. Available: <https://doc.aldebaran.com/>
- [2] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics, Cambridge, MA, USA: MIT Press, 2005.
- [3] A. McCool, “Creating conversational robots: Design and implementation of dialog systems in humanoid robots,” IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 1175–1180, May 2018.
- [4] F. Chollet et al., “Keras,” [Online]. Available: <https://keras.io>
- [5] Flask Documentation Team, Flask (A Python Microframework) Documentation, 2024. [Online]. Available: <https://flask.palletsprojects.com/>
- [6] OpenAI, OpenAI API Reference, 2024. [Online]. Available: <https://platform.openai.com/docs/>
- [7] DeepSeek Company, DeepSeek Chat API Documentation, 2024. [Online]. Available: <https://platform.deepseek.com/>
- [8] M. F. McTear, Z. Callejas, and D. Griol, The Conversational Interface: Talking to Smart Devices, Springer, 2016.
- [9] R. Buyya, C. Vecchiola, and S. Thamarai Selvi, Mastering Cloud Computing: Foundations and Applications Programming, Morgan Kaufmann, 2013.
- [10] B. A. Miller and D. B. Rawlings, “Using Python and Flask to Develop Scalable Microservices,” IEEE Softw., vol. 37, no. 3, pp. 45–51, May/Jun. 2020.
- [11] D. Lowe and T. Wragg, “Implementing voice recognition in humanoid robots using cloud APIs,” in Proc. IEEE Int. Symp. Robot. & Human Interactive Commun. (RO-MAN), pp. 512–517, 2019.
- [12] A. Singh and S. Khare, “Client-Server Communication using HTTP Protocol in IoT Environments,” Int. J. Computer Applications, vol. 182, no. 48, pp. 9–14, Apr. 2019.

