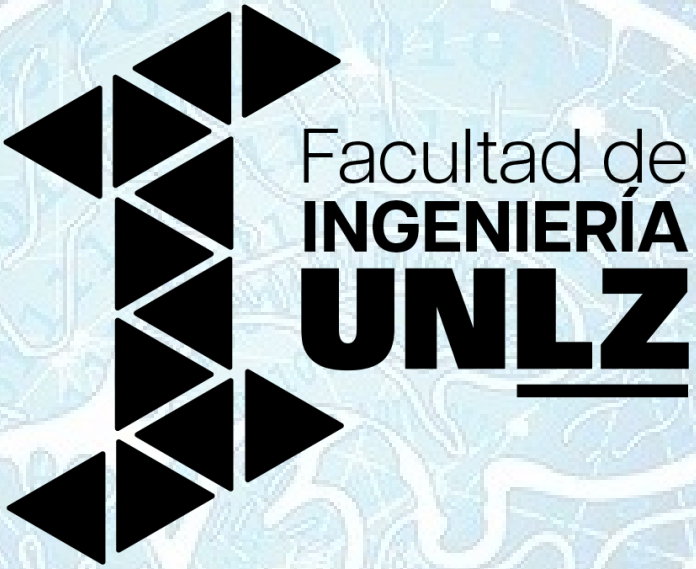
Proyecto tecnológico 3



Control de dispositivos por Voz con IA - Whisper

Ing. Lukaszewicz, Cristian

Documento Versión 1.0

Índice

1. Observación del Entorno

1.1. Disparador

En un mundo cada vez más conectado, la capacidad de interactuar con dispositivos a través de la voz se ha vuelto esencial. Nos preguntamos: ¿Cómo puede un sistema interpretar la voz humana y realizar tareas automáticamente?

1.2. Recorte del Tema

Desarrollar un sistema que utilice Whisper para el reconocimiento de voz local y permita controlar luces virtuales u otras acciones en la computadora mediante comandos de voz simples.

1.3. Objetivos

1.3.1. Objetivo General

Crear un sistema de control de dispositivos mediante comandos de voz utilizando Whisper para el reconocimiento de voz y un modelo de clasificación de texto para interpretar los comandos y ejecutar acciones en la computadora.

1.3.2. Objetivos Específicos

- Implementar el reconocimiento de voz local utilizando Whisper.
- Desarrollar una aplicación básica en Python que interprete comandos de voz.
- Controlar luces virtuales o acciones en la computadora basándose en los comandos detectados.
- Fomentar el trabajo en equipo y el desarrollo de competencias técnicas y blandas en los estudiantes.

1.4. Materiales para Ejecutar el Proyecto

- **Hardware:**
- Computadoras con Python instalado y micrófonos.
- **Software:**
- Python 3.x
- Whisper
- Transformers (Hugging Face)

- Librerías de Python: `sounddevice`, `soundfile`, `transformers`, `playsound`, `tkinter` (incluida en Python estándar)
- **Otros:**
- Conexión a internet para descargar bibliotecas.

1.5. Otros Actores Involucrados

- Docentes de programación e inteligencia artificial
- Asistentes técnicos para la configuración de software
- Coordinadores de proyectos tecnológicos

1.6. Roles y Funciones a Designar en Cada Grupo de Trabajo

1.6.1. Equipos

Cada grupo se dividirá en:

- **Líder de Equipo:** Coordina actividades y mantiene el progreso.
- **Programadores:** Desarrollan y mantienen el código Python.
- **Testers:** Realizan pruebas del sistema y reportan errores.
- **Documentalistas:** Se encargan de la documentación y reportes del proyecto.

1.7. Tiempo Total del Proyecto

El proyecto se desarrollará en cuatro semanas, cada una enfocada en diferentes aspectos del desarrollo e integración del sistema de control por voz.

1.8. Plan de Trabajo

1.8.1. Planteamiento Guía

"¿Cómo puede un sistema interpretar la voz humana y realizar tareas automáticamente usando tecnologías de inteligencia artificial como Whisper?"

1.8.2. Organización y Planificación

El proyecto se divide en cuatro etapas semanales, cada una con tareas específicas de investigación, desarrollo e integración.

1.8.3. Descripción Detallada de Actividades

2. Diagnóstico

2.1. Conocimiento Técnico

Para llevar a cabo este proyecto, es esencial que los estudiantes tengan conocimientos básicos en:

- **Programación en Python:** Para el desarrollo de scripts que manejan el reconocimiento de voz.
- **Uso de Bibliotecas de IA:** Manejo de Whisper y Transformers para el reconocimiento y clasificación de comandos de voz.
- **Procesamiento de Señales de Audio:** Entender cómo capturar y manejar audio en Python.

2.2. Aporte Curricular

Este proyecto contribuye al currículo en las siguientes áreas:

- **Programación:** Aplicación práctica de conceptos de programación en Python.
- **Inteligencia Artificial:** Implementación de reconocimiento de voz y clasificación de comandos.
- **Procesamiento de Voz:** Uso de tecnologías de procesamiento de señales de audio para interpretar comandos de voz.

2.3. Aporte al Desarrollo de Competencias Genéricas

- **Trabajo en Equipo:** Colaboración efectiva entre los miembros del grupo.
- **Resolución de Problemas:** Identificación y solución de desafíos técnicos durante el desarrollo del proyecto.
- **Motivación para Entender Aplicaciones de la IA:** Fomentar el interés en cómo la IA puede aplicarse para mejorar la interacción humano-computadora.

3. Planteo de las Metas

3.1. Selección de Contenidos

- **Procesamiento de Señales de Audio:** Fundamentos y técnicas para capturar y procesar audio en Python.
- **Manejo de Bibliotecas para IA:** Uso de Whisper para el reconocimiento de voz y Transformers para la clasificación de comandos.
- **Desarrollo de Aplicaciones Sencillas con Python:** Creación de scripts que capturen, procesen y ejecuten comandos de voz.

3.2. Producto Final

Un sistema que detecte y ejecute comandos de voz para controlar luces virtuales o acciones en la computadora, utilizando Whisper para el reconocimiento de voz y un modelo de clasificación de texto para interpretar los comandos.

3.3. Objetivos de Aprendizaje

- Comprender cómo funciona el reconocimiento de voz utilizando Whisper.
- Aplicar técnicas de procesamiento de audio en Python.
- Desarrollar habilidades en el uso de modelos de clasificación de texto para interpretar comandos de voz.

3.4. Resultados de Aprendizaje

- Crear un sistema básico de reconocimiento de voz que detecte comandos específicos.
- Implementar interacciones simples a partir de comandos de voz detectados.
- Integrar el reconocimiento de voz con acciones ejecutables en la computadora.

4. Acciones a Desarrollar

4.1. Semana 1: Introducción al Procesamiento de Voz y Whisper

Objetivo de la Etapa:
Introducir a los estudiantes en los conceptos básicos del procesamiento de voz y familiarizarlos con la biblioteca Whisper para el reconocimiento de voz local.

Actividades:

- **Conceptos Básicos de Procesamiento de Voz:**
- **Definición:** Explicación de cómo se procesa la voz humana en señales digitales.
- **Aplicaciones:** Uso de reconocimiento de voz en asistentes virtuales, control de dispositivos, etc.
- **Introducción a Whisper:**
- **¿Qué es Whisper?** Breve descripción de Whisper como modelo de reconocimiento de voz de OpenAI.
- **Instalación de Whisper:**

```
pip install git+https://github.com/openai/whisper.git
```

- **Prueba Básica de Whisper:**

```
import whisper


# Cargar el modelo Whisper
model = whisper.load_model("base")


# Transcribir un archivo de audio
result = model.transcribe("audio_sample.wav")
print(result["text"])
```

Explicación: Este script carga el modelo Whisper y transcribe un archivo de audio llamado `audio_sample.wav`, imprimiendo el texto transcrito.

- **Configuración del Entorno de Trabajo:**
- **Instalar Python y Bibliotecas Necesarias:**

```
pip install sounddevice soundfile transformers
```

- **Configuración de Micrófono:**
- Asegurarse de que el micrófono esté correctamente conectado y configurado en el sistema operativo.
- **Captura de Audio en Python:**
- **Script para Capturar Audio:**

```
import sounddevice as sd
import soundfile as sf

def capturar_audio(duracion=5, fs=44100):
    print("Capturando audio...")
    audio = sd.rec(int(duracion * fs), samplerate=fs, channels=1)
    sd.wait()
    sf.write('captura.wav', audio, fs)
    print("Audio capturado.")
    return 'captura.wav'

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

Explicación: Este script captura 5 segundos de audio y lo guarda como `captura.wav`.

- **Dinámica de Introducción:**
- **Preguntas de Guía:**
- ¿Cómo se puede usar el reconocimiento de voz para mejorar la interacción con dispositivos electrónicos?
- ¿Qué desafíos enfrentan los sistemas de reconocimiento de voz?
- **Discusión en Grupo:**
- Reflexión sobre las aplicaciones del reconocimiento de voz en la vida cotidiana.
- Identificación de posibles comandos de voz para el proyecto.

4.2. Semana 2: Implementación de Comandos de Voz para una Aplicación Básica

Objetivo de la Etapa:
Desarrollar una aplicación básica en Python que utilice Whisper para detectar comandos de voz y ejecutar acciones simples en la computadora.

Actividades:

- **Definición de Comandos de Voz:**
- "Encender luces"
- "Apagar luces"
- "Aumentar volumen"
- "Disminuir volumen"
- **Desarrollo del Script de Reconocimiento y Ejecución de Comandos:**

Código Unificado para Raspberry Pi y Windows:

```
import whisper
import time
import sounddevice as sd
import soundfile as sf
import platform
```

```
if platform.system() == "Linux":
    import RPi.GPIO as GPIO
elif platform.system() == "Windows":
    import serial
    from transformers import pipeline

# Configuración específica para cada plataforma
if platform.system() == "Linux":
    # Configuración de GPIO para Raspberry Pi
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(18, GPIO.OUT)  # Luz
    GPIO.setup(23, GPIO.OUT)  # Ventilador

    # Cargar modelo Whisper
    model = whisper.load_model("base")

elif platform.system() == "Windows":
    # Configuración de Arduino (ajustar el puerto COM según corresponda)
    ser = serial.Serial('COM3', 9600, timeout=1)

    # Cargar modelo Whisper
    model = whisper.load_model("base")
```

```
    # Cargar modelo de clasificación de texto
    classifier = pipeline("zero-shot-classification",
model="facebook/bart-large-mnli")
    candidate_labels = ["encender luces", "apagar luces",
"encender ventilador", "apagar ventilador"]


def capturar_audio(duracion=5, fs=44100):
    print("Capturando audio...")
    audio = sd.rec(int(duracion * fs), samplerate=fs, channels=1)
    sd.wait()
    sf.write('captura.wav', audio, fs)
    print("Audio capturado.")
    return 'captura.wav'


def procesar_comando(audio_path):
    # Transcribir audio con Whisper
    result = model.transcribe(audio_path)
    texto = result["text"].lower()
    print(f"Comando detectado: {texto}")


    if platform.system() == "Linux":
        # Procesar comandos en Raspberry Pi
        if "encender luces" in texto:
            GPIO.output(18, GPIO.HIGH)
            print("Luces encendidas")
        elif "apagar luces" in texto:
```

```
            GPIO.output(18, GPIO.LOW)
            print("Luces apagadas")
        elif "encender ventilador" in texto:
            GPIO.output(23, GPIO.HIGH)
            print("Ventilador encendido")
        elif "apagar ventilador" in texto:
            GPIO.output(23, GPIO.LOW)
            print("Ventilador apagado")
        elif "aumentar volumen" in texto:
            aumentar_volumen()
            print("Volumen aumentado")
        elif "disminuir volumen" in texto:
            disminuir_volumen()
            print("Volumen disminuido")

    elif platform.system() == "Windows":
        # Clasificar el texto para identificar el comando
        classification = classifier(texto, candidate_labels)
        label = classification['labels'][0]

        if label == "encender luces":
            ser.write(b'ENCENDER_LUZ\n')
            print("Luces encendidas")
        elif label == "apagar luces":
            ser.write(b'APAGAR_LUZ\n')
```

```
            print("Luces apagadas")
        elif label == "encender ventilador":
            ser.write(b'ENCENDER_VENTILADOR\n')
            print("Ventilador encendido")
        elif label == "apagar ventilador":
            ser.write(b'APAGAR_VENTILADOR\n')
            print("Ventilador apagado")
        else:
            print("Comando no reconocido")

def aumentar_volumen():
    # Implementar lógica para aumentar el volumen en Linux
    os.system("amixer -D pulse sset Master 5%+")

def disminuir_volumen():
    # Implementar lógica para disminuir el volumen en Linux
    os.system("amixer -D pulse sset Master 5%-")

while True:
    audio = capturar_audio()
    procesar_comando(audio)
    time.sleep(1)
```

Explicación Detallada:

1. **Importaciones y Configuraciones Iniciales:**

- **Bibliotecas Comunes:**
- `whisper`: Para el reconocimiento de voz local.
- `time`, `sounddevice`, `soundfile`: Para la captura y manejo de audio.
- **Bibliotecas Específicas por Plataforma:**
- **Raspberry Pi (Linux):** `RPi.GPIO` para el control de pines GPIO.
- **Windows:** `serial` para la comunicación con Arduino y `transformers` para la clasificación de texto.

2. **Configuración Específica por Plataforma:**
   - **Raspberry Pi:**
   - Configura los pines GPIO para controlar relés que manejarán las luces y ventiladores.
   - Carga el modelo Whisper para el reconocimiento de voz local.
   - **Windows (Netbook con Arduino):**
   - Establece la comunicación serial con el Arduino a través del puerto COM especificado.
   - Carga el modelo Whisper y el modelo de clasificación de texto (`facebook/bart-large-mnli`) para interpretar comandos de voz sin necesidad de APIs de pago.

3. **Funciones Principales:**
   - `capturar_audio`**:** Captura un fragmento de audio de 5 segundos y lo guarda como `captura.wav`.
   - `procesar_comando`**:**
   - **Transcripción:** Utiliza Whisper para transcribir el audio capturado a texto.
   - **Clasificación de Comandos (Windows):** Usa un modelo de clasificación de texto para identificar el comando específico.
   - **Ejecución de Comandos:**
   - **Raspberry Pi:** Controla los pines GPIO para encender o apagar luces y ventiladores según el texto transcrito.
   - **Windows:** Envía comandos seriales al Arduino para controlar los dispositivos.

- **Funciones de Volumen (Solo Linux):**
- `aumentar_volumen` **y** `disminuir_volumen`**:** Implementan el aumento y disminución del volumen del sistema utilizando `amixer`.

4. **Loop Principal:**

- Captura audio.
- Procesa el comando detectado.
- Espera 1 segundo antes de repetir el proceso.

Notas Importantes:

- **Puerto COM en Windows:**
- Asegúrate de reemplazar `'COM3'` con el puerto COM correcto asignado a tu Arduino. Puedes verificarlo en el Administrador de Dispositivos de Windows bajo "Puertos (COM & LPT)".
- **Manejo de Excepciones:**
- Para una mayor robustez, considera agregar bloques `try-except` para manejar posibles errores, como la falta de reconocimiento de voz o problemas de conexión serial.
- **Modelo de Clasificación:**
- El modelo `facebook/bart-large-mnli` es gratuito y puede ejecutarse localmente, evitando costos asociados a APIs de pago. Este modelo realiza clasificación de texto sin requerir datos adicionales de entrenamiento.

4.3. Semana 3: Optimización del Reconocimiento de Voz y Pruebas

Objetivo de la Etapa:
Mejorar la precisión del sistema de reconocimiento de voz y realizar pruebas exhaustivas para asegurar su funcionamiento correcto.

Actividades:

- **Mejoras en la Captura y Transcripción de Audio:**
- Ajustar la duración de la captura de audio según sea necesario.
- Filtrar ruido de fondo utilizando técnicas de procesamiento de audio.
- **Optimización del Modelo de Clasificación:**

- Ajustar los parámetros del modelo de clasificación de texto para mejorar la precisión.
- Experimentar con diferentes modelos de Transformers si es necesario.
- **Pruebas de Precisión:**
- Ejecutar una serie de pruebas con diferentes comandos de voz y en distintas condiciones de ruido.
- Registrar los resultados y ajustar el sistema según sea necesario.
- **Documentación de Resultados:**
- Crear tablas y gráficos que muestren la precisión del sistema en diferentes escenarios.
- **Feedback y Mejoras:**
- Recopilar retroalimentación de los estudiantes y ajustar el sistema para mejorar la usabilidad y la precisión.

4.4. Semana 4: Integración con Dispositivos IoT (Opcional) o Simulación en PC

Objetivo de la Etapa:
Integrar el sistema de reconocimiento de voz con dispositivos IoT reales o simular acciones en la computadora para controlar dispositivos virtuales.

Actividades:

- **Integración con Dispositivos IoT:**
- **Hardware:** Conectar dispositivos IoT como luces inteligentes, enchufes, etc., a la red local.
- **Software:** Modificar el script de Python para enviar comandos a los dispositivos IoT a través de protocolos como MQTT.
- **Código de Ejemplo:**

```
import paho.mqtt.client as mqtt


MQTT_BROKER = "localhost"
MQTT_PORT = 1883
```

```
MQTT_TOPIC = "home/lights"


client = mqtt.Client()
client.connect(MQTT_BROKER, MQTT_PORT, 60)


def encender_luces_iot():
    client.publish(MQTT_TOPIC, "ON")
    print("Luces IoT encendidas")


def apagar_luces_iot():
    client.publish(MQTT_TOPIC, "OFF")
    print("Luces IoT apagadas")
```

- **Simulación en PC:**
- **Acciones en la Computadora:** Controlar aplicaciones o funciones del sistema operativo basándose en comandos de voz.
- **Ejemplo:** Abrir o cerrar aplicaciones, controlar el volumen del sistema, etc.
- **Código de Ejemplo:**

```
import os
import subprocess


def abrir_navegador():
    subprocess.Popen(['start', 'chrome'], shell=True)  # Windows
    # En Linux:
    # subprocess.Popen(['xdg-open', 'https://www.google.com'])
```

```
def cerrar_navegador():
    os.system("taskkill /IM chrome.exe /F")  # Windows
    # En Linux, se puede usar pkill
    # os.system("pkill chrome")
```

- **Pruebas de Integración:**
- Ejecutar el sistema con dispositivos IoT reales o con acciones simuladas en la computadora.
- Verificar que los comandos de voz ejecutan las acciones correspondientes.
- **Optimización Final:**
- Refinar el sistema para mejorar la fluidez y la precisión en la ejecución de comandos.
- **Documentación Final:**
- Documentar todo el proceso de integración, las pruebas realizadas y los resultados obtenidos.

5. Evaluación

5.1. Instrumento de Evaluación

- **Evaluación Continua:** Revisión semanal del progreso del proyecto y cumplimiento de objetivos.
- **Evaluación Final:** Verificación de la funcionalidad completa del sistema de control por voz, incluyendo el reconocimiento de comandos y la ejecución de acciones.
- **Autoevaluación y Coevaluación:** Reflexión individual y evaluación del trabajo en equipo.

5.2. Evaluación Formativa

- **Observación Directa:** Monitoreo de la participación y compromiso de los estudiantes durante las actividades.
- **Bitácoras de Proyecto:** Registro de actividades, problemas y soluciones propuestas por los estudiantes.

5.3. Evaluación Sumativa

- **Funcionamiento del Sistema:** Evaluación de la precisión y eficiencia del reconocimiento de voz y la ejecución de comandos.
- **Presentación del Proyecto:** Exposición oral y demostración del sistema funcionando.
- **Informe Final:** Documento escrito detallando el desarrollo del proyecto, decisiones técnicas y resultados obtenidos.

5.4. Entregables Esperables

- **Bitácoras de Proyecto:** Documentación diaria de actividades y avances.
- **Programas en Python:** Scripts desarrollados para el reconocimiento de voz y control de dispositivos.
- **Documentación Técnica:** Especificaciones de hardware y software utilizados.
- **Informe Final:** Compilación de todo el trabajo realizado con análisis y conclusiones.

- **Presentación del Proyecto:** Diapositivas y demostración del sistema funcionando.

5.5. Rúbricas

Rúbrica para la Evaluación del Sistema de Reconocimiento de Voz

Criterio	Nivel 1 (Insuficiente)	Nivel 2 (Aceptable)	Nivel 3 (Bueno)	Nivel 4 (Excelente)
Reconocimiento de Voz	No funciona o es inconsistente.	Funciona con dificultades.	Funciona correctamente con algunos ajustes.	Funciona de manera fluida y precisa.
Interpretación de Comandos	No interpreta correctamente los comandos.	Interpreta algunos comandos con errores.	Interpreta correctamente la mayoría de los comandos.	Interpreta de manera precisa todos los comandos definidos.
Ejecución de Acciones	No ejecuta las acciones o las ejecuta incorrectamente.	Ejecuta acciones con errores o retrasos.	Ejecuta acciones correctamente con mínimas fallas.	Ejecuta acciones de manera eficiente y sin errores.
Interfaz de Usuario	No hay interfaz o es difícil de usar.	Interfaz básica pero funcional.	Interfaz clara y fácil de usar.	Interfaz intuitiva y altamente funcional.
Documentación	Documentación incompleta o ausente.	Documentación básica pero faltan detalles importantes.	Documentación clara y detallada con mínimos ajustes.	Documentación completa, clara y detallada.

Presentación	Presentación desorganizada y difícil de entender.	Presentación comprensible pero falta de claridad en algunos puntos.	Presentación clara y bien organizada con buena comunicación.	Presentación excepcionalmente clara, organizada y profesional.

6. Expansiones Opcionales

6.1. Integración con Dispositivos IoT Adicionales

- **Control de Electrodomésticos:**
  Implementación de comandos para controlar otros electrodomésticos como ventiladores, termostatos, etc.
- **Hardware:** Añadir relés adicionales o módulos IoT para los nuevos dispositivos.
- **Software:** Añadir nuevos comandos y acciones en el script de Python.
- **Ejemplo:**

```
candidate_labels = ["encender luces", "apagar luces", "encender ventilador", "apagar ventilador", "encender termostato", "apagar termostato"]

# Añadir lógica para termostato en procesar_comando

if label == "encender termostato":

    # Acción para encender termostato

    pass

elif label == "apagar termostato":

    # Acción para apagar termostato

    pass
```

6.2. Implementar Notificaciones Visuales y Sonoras

- **Alertas en la Computadora:**
  Configuración de notificaciones visuales o sonidos para confirmar la recepción de comandos.
- **Herramientas:** Uso de librerías como `playsound` para sonidos o `tkinter` para ventanas emergentes.
- **Implementación en Python:**

```
import tkinter as tk
from playsound import playsound


def notificacion_visual(mensaje):
    root = tk.Tk()
    root.title("Notificación")
    tk.Label(root, text=mensaje).pack()
    root.after(3000, root.destroy)  # Cierra la ventana después
de 3 segundos
    root.mainloop()


def notificacion_sonora(ruta_sonido):
    playsound(ruta_sonido)


# Ejemplo de uso
notificacion_visual("Comando recibido: Encender luces")
notificacion_sonora("sonido_alerta.mp3")
```

6.3. Mejorar la Precisión del Reconocimiento de Comandos

- **Ajustes en el Modelo de Clasificación:**
- Refinar el pipeline de clasificación o usar modelos más específicos si es necesario.
- Añadir más comandos y ajustar los parámetros del modelo para mejorar la clasificación.
- **Ejemplo:**

```
candidate_labels = ["encender luces", "apagar luces", "encender
```

```
ventilador", "apagar ventilador", "aumentar volumen", "disminuir
volumen"]

# Refinar el pipeline o ajustar threshold
```

7. Referencias

1. **OpenAI.** *Whisper: An Automatic Speech Recognition System*. 2023. https://openai.com/research/whisper.
2. **Transformers by Hugging Face.** *Transformers Documentation*. 2023. https://huggingface.co/docs/transformers/index.
3. **SoundDevice.** *SoundDevice Documentation*. 2023. https://python-sounddevice.readthedocs.io/.
4. **SoundFile.** *SoundFile Documentation*. 2023. https://pysoundfile.readthedocs.io/.
5. **Python.org.** *Python Documentation*. 2023. https://docs.python.org/.
6. **IFTTT.** *IFTTT Webhooks Documentation*. 2023. https://ifttt.com/maker_webhooks.
7. **OpenCV.** *Open Source Computer Vision Library*. 2023. https://opencv.org/.
8. **Transformers by Hugging Face.** *Transformers Documentation*. 2023. https://huggingface.co/docs/transformers/index.
9. **Paho MQTT.** *MQTT Client Documentation*. 2023. https://www.eclipse.org/paho/index.php?page=clients/python/index.php.

Anexos

Anexo A: Diagramas de Conexión de Hardware

- **Diagrama 1:** Conexión del Micrófono a la Computadora.
- **Diagrama 2:** Conexión de Relés a la Computadora (si se usa Raspberry Pi).
- **Diagrama 3:** Conexión de Arduino para Control de Dispositivos (si se usa Windows con Arduino).

Anexo B: Scripts de Python Utilizados

- **Script 1:** Reconocimiento de Voz y Control de Acciones (Unificado para Raspberry Pi y Windows).
- **Script 2:** Envío de Notificaciones Visuales y Sonoras.
- **Script 3:** Control de Dispositivos IoT (si se implementa).

Anexo C: Capturas de Pantalla de la Aplicación

- **Captura 1:** Terminal con el script corriendo.
- **Captura 2:** Ejemplo de notificación visual.
- **Captura 3:** Dashboard de cualquier integración opcional (como con IoT).

Anexo D: Manual de Usuario del Sistema

- **Instrucciones de Inicio:**
- **Instalación de Dependencias:** Cómo instalar las bibliotecas necesarias.
- **Configuración del Sistema:** Cómo configurar el puerto COM (en Windows) o GPIO (en Raspberry Pi).
- **Uso de Comandos de Voz:** Lista de comandos disponibles y cómo usarlos.
- **Solución de Problemas:**
- **Problema:** Comandos de voz no reconocidos.
- **Solución:** Verificar la configuración del micrófono y la calibración del modelo de clasificación.
- **Problema:** El sistema no ejecuta comandos.

- **Solución:** Verificar conexiones de hardware o la configuración del puerto COM/ GPIO.
- **Mantenimiento del Sistema:**
- **Actualización de Software:** Cómo actualizar los scripts y bibliotecas.
- **Revisión de Hardware:** Inspección periódica de conexiones y componentes electrónicos.

Mejoras y Recomendaciones Adicionales (Futuras)

1. **Optimización del Reconocimiento de Voz:**
    - **Filtros de Audio:** Implementar filtros de ruido en el preprocesamiento del audio para mejorar la precisión del reconocimiento.
    - **Comandos Personalizados:** Crear un diccionario de comandos personalizados para adaptarse mejor al entorno específico de uso.
2. **Seguridad del Sistema:**
    - **Protección de Datos:** Asegurar que los datos de audio y comandos no se almacenen innecesariamente para proteger la privacidad.
    - **Acceso Controlado:** Establecer mecanismos de autenticación para acceder al sistema de control de voz, evitando usos no autorizados.
3. **Interfaz de Usuario Mejorada:**
    - **Aplicación Gráfica:** Desarrollar una interfaz gráfica con `tkinter` para facilitar el uso del sistema.
    - **Feedback Visual y Sonoro:** Integrar señales visuales (como luces LED) o sonoras para confirmar la recepción y ejecución de comandos.
4. **Expansión de Funcionalidades:**
    - **Control de Otros Dispositivos:** Añadir control de dispositivos adicionales como televisores, termostatos, etc.
    - **Automatización Basada en Contexto:** Programar acciones automáticas basadas en horarios o eventos específicos.
5. **Documentación y Formación Continua:**
    - **Guías de Usuario Detalladas:** Crear manuales más extensos y detallados para facilitar el uso y mantenimiento del sistema.
    - **Capacitaciones Adicionales:** Ofrecer sesiones de formación continuas para los estudiantes sobre nuevas tecnologías y actualizaciones del sistema.

Links de actualización pequeños

Link pequeño

https://sor.bz/2xvre

QR pequeño

Links de actualización completos

Link completo

https://github.com/guideahon/Pro_tec_3_IA_Electr-nica

QR completo