Universidad Nacional de Colombia

Facultad de Ingeniería y Arquitectura

Traductor - Sistema embebido raspberry pi

Presentado por:

Juan Esteban Montoya Ramírez

UNIVERSIDAD
NACIONAL
DE COLOMBIA

Manizales - 2025

Índice

Resumen

Este proyecto, desarrollado como parte del curso ”Programación de Sistemas Linux Embebidos”, consistió en la creación de un sistema de traducción en tiempo real capaz de procesar voz, texto e imágenes. Utilizando tecnologías como Tesseract para el reconocimiento óptico de caracteres (OCR), SpeechRecognition para el procesamiento de voz y OpenCV para el manejo de imágenes, el sistema integra múltiples funcionalidades en una plataforma embebida basada en Linux.

El proyecto enfrentó desafíos significativos, como la optimización de recursos limitados, la mejora de la precisión en condiciones variables y la integración de periféricos como cámaras y micrófonos. A través de un enfoque modular y colaborativo, se logró desarrollar una solución funcional que no solo cumple con los requisitos técnicos, sino que también fomenta la innovación y el aprendizaje continuo. Este trabajo demuestra la aplicabilidad de los sistemas embebidos en soluciones prácticas y resalta la importancia de combinar teoría y práctica en la formación de profesionales en el campo de la tecnología.

1. introducción

Durante el desarrollo del curso ”Programación de Sistemas Linux Embebidos”, los participantes adquirieron competencias fundamentales en la estructura computacional y en como generar software dedicado, con un enfoque especial en el funcionamiento de sistemas operativos basados en el kernel Linux. Esta clase no solo proporcionó una base sólida en conceptos teóricos relacionados a este, sino que también fomentó la aplicación práctica de estos conocimientos a través de diversas actividades y tareas.

Cada actividad fue diseñada para estimular la curiosidad y el pensamiento crítico de los estudiantes para resolver problemas de algún grado de complejidad, permitiéndoles formular preguntas relevantes y participar en debates personales que enriquecieron su comprensión. Estos ejercicios no solo reforzaron los temas discutidos en clase, sino que también promovieron un ambiente de aprendizaje colaborativo y reflexivo y enfocado a la programación comercial.

Como culminación del curso, se presentó un reto significativo: el desarrollo de un proyecto que requería la aplicación de los recursos y competencias adquiridos durante las sesiones. Este proyecto no solo evaluó la capacidad de los estudiantes para resolver problemas complejos utilizando Linux, sino que también fomentó la innovación y la creatividad en la búsqueda de soluciones efectivas.

Además, el curso estimuló la obtención continua de conocimiento, incentivando a los participantes a explorar más allá de los límites del aula y a mantenerse actualizados con las últimas tendencias y tecnologías en el ámbito de los sistemas embebidos. En resumen, el curso ”Programación de Sistemas Linux Embebidos”proporcionó una experiencia integral que combinó teoría y práctica, preparando a los estudiantes para enfrentar desafíos reales en el campo de la programación de sistemas embebidos.

El proyecto elegido en este caso particular fue de un traductor en tiempo real de voz, texto e imagen el cual nos presta su salida en pantalla y en altavoz de ser así requerido, este proyecto conllevo retos particulares como lo fue manejo de imágenes en tiempo real, construcción de estructura de desarrollo y manejo de periféricos de voz, cámara.

2. obejtivos

- Desarrollar un sistema de traducción en tiempo real que integre funcionalidades de reconocimiento de voz, procesamiento de imágenes (OCR) y salida de texto y audio.

- Aplicar los conocimientos adquiridos en el curso ”Programación de Sistemas Linux Embebidos”para resolver problemas prácticos relacionados con el manejo de periféricos, procesamiento en tiempo real y optimización de recursos.
- Fomentar la innovación y creatividad en la implementación de soluciones técnicas, abordando desafíos como el manejo de imágenes en tiempo real, la construcción de una estructura de desarrollo modular y la integración de hardware y software.
- Promover el aprendizaje colaborativo y la resolución de problemas complejos mediante el trabajo en equipo y la aplicación de metodologías de desarrollo ágil.
- Evaluar la viabilidad y eficiencia del sistema en entornos reales, considerando limitaciones de hardware y software, así como la precisión y usabilidad del sistema

3. Marco Teórico

3.1. Sistemas embebidos

Los sistemas embebidos son dispositivos de único propósito el cual da solución a una serie de requerimientos, estos se componen de un microprocesador o un microcontrolador que aunque suenen muy parecido tiene diferencias marcadas las cuales pueden marcar la diferencia si se elige uno u otro, un microcontrolador en un empaquetado de un microprocesador con herramientas que nos ayudan a procesar entradas y salidas como lo son ADC’s, PWM, DAC’s. Los microprocesadores es un circuito integrado que contiene millones de transistores y otros componentes electrónicos, diseñado para ejecutar instrucciones de un programa informático. Es el cerebro de cualquier dispositivo electrónico que realiza tareas computacionales, como computadoras, teléfonos inteligentes, tabletas y muchos otros dispositivos electrónicos.

Los sistemas embebidos son el núcleo de muchos productos, máquinas y operaciones inteligentes, como las aplicaciones de aprendizaje automático e inteligencia artificial. Como las aplicaciones de los sistemas embebidos aparecen hoy en día en todas las industrias y sectores, los dispositivos y el software embebidos desempeñan un papel crucial en el funcionamiento de los coches, los electrodomésticos, los dispositivos médicos, los quioscos interactivos y otros equipos que utilizamos en nuestra vida cotidiana.

3.2. Linux

Linux es una familia de sistemas operativos semejantes a Unix, de código abierto y desarrollado por una comunidad, para computadoras, servidores, mainframes, dispositivos móviles y dispositivos embebidos. Es compatible con casi todas las principales plataformas informáticas, incluyendo x86, ARM y SPARC, por lo que es uno de los sistemas operativos más soportados.

3.3. Programación de Sistemas Embebidos

La programación de sistemas embebidos generalmente se hace en los lenguajes:

- C
- C++

- Python
- Rust
- Assembly

3.4. Estado del arte

Un sistema embebido con propósito de traductor en la epoca del 2010 cuando llegaron dispositivos de por ejemplo los auriculares pilot de waverly labs en 2016 que permitian la traducción en conversación con un rango de 15 idiomas, posteriormente, en 2017, google impulso esto con sus propia tecnología de traducción en tiempo real, actualmente con la llegada de nuevos avances de software y hardware como quizás uno de los impulsos mas grande de estos últimos años la cual es la llegada masiva de la IA se puede hoy en día hacer estos procesos de una forma increíblemente rápida con una respuesta de menos de un segundo con el uso de celular móviles dejando muy atrás y casi que muy poco factible la construcción de dispositivos separados a los celulares sino que el mercado se centra en darle mas poder a los smartphones, sabiendo esto se considera que este proyecto es por investigación y tiene la intención de realizar análisis y aprendizaje no de búsqueda de proyectos con salida comercial, sin des meritar que algunas herramientas aquí vistas se pueden usar para otro tipo de proyectos con posible segmento de mercado.
algunas menciones especiales a dispositivos de alto interes que realizaban tareas similares al de este proyecto:

- pilot auriculares por waverly Labs
- Google Lens Traductor
- Articulo sobre la llegada de la IA a la traduccion en tiempo real

4. Descripción del Sistema

En esta sección se abordaran algunas características cruciales en proyecto, ya que se definirá cuales son los requisitos funcionales y no funcionales y se esquematizaran para posteriormente darle una solución solida a cada uno de estos requerimientos.

4.1. Requisitos Funcionales

Listado:
Numero requerimiento:RF1
Nombre requerimiento:Menú
Descripción: El usuario debe tener la posibilidad de elegir entre un menú de opciones la cual sea conveniente para su objetivo.
Prioridad: Alta **Numero requerimiento**:RF2

Nombre requerimiento:Traducción por voz
Descripción: El usuario debe tener la posibilidad de elegir entre el menú de opciones la opción de traducción por voz dando así un indicio para que el usuario hable y al finalizar la traducción sea dada.

Prioridad: Alta **Numero requerimiento**:RF3

Nombre requerimiento:Traducción por voz
Descripción: El usuario debe tener la posibilidad de elegir entre el menú de opciones la opción de traducción por captura dando así un indicio para que el usuario acomode la cámara y al finalizar la traducción sea dada.
Prioridad: Alta **Numero requerimiento**:RF4

Nombre requerimiento:Traducción en tiempo real
Descripción: El usuario debe tener la posibilidad de elegir entre el menú de opciones la opción de traducción en tiempo real dando así un indicio para que el usuario acomode la cámara y la traducción sea impresa tanto en la pantalla de previsualización de la camara como en la salida a pantalla.
Prioridad: Alta

4.2. Requisitos no Funcionales

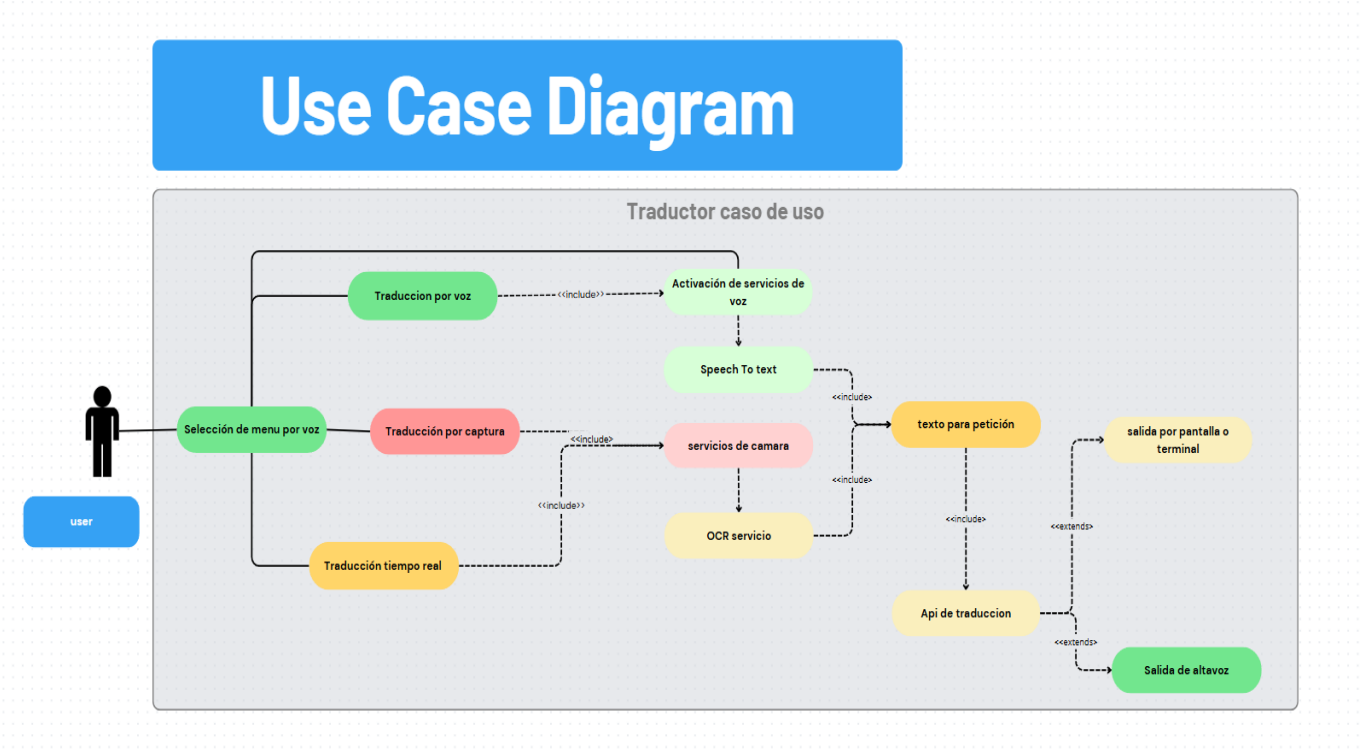
Listado: **Numero requerimiento**:RNF1

Nombre requerimiento:Respuesta a errores
Descripción: El sistema debe ser capaz de manejar errores y dar feedback de estos errores para alertar si se esta presentando un caso particular y avisar al usuario para que este tome medidas al respecto
Prioridad: Alta **Numero requerimiento**:RNF2

Nombre requerimiento:Manejo de memoria
Descripción: El sistema debe ser capaz de manejar y responder a situaciones que se den con la memoria dando así el máximo desempeño posible ayudando a un buen manejo de los recursos **Prioridad**: Alta
Numero requerimiento:RNF3

Nombre requerimiento:protección de equipo
Descripción: El sistema debe ser capaz verificar el estado de el mismo para poder tomar acciones a sus tareas.
Prioridad: media

4.3. Diagrama de uso

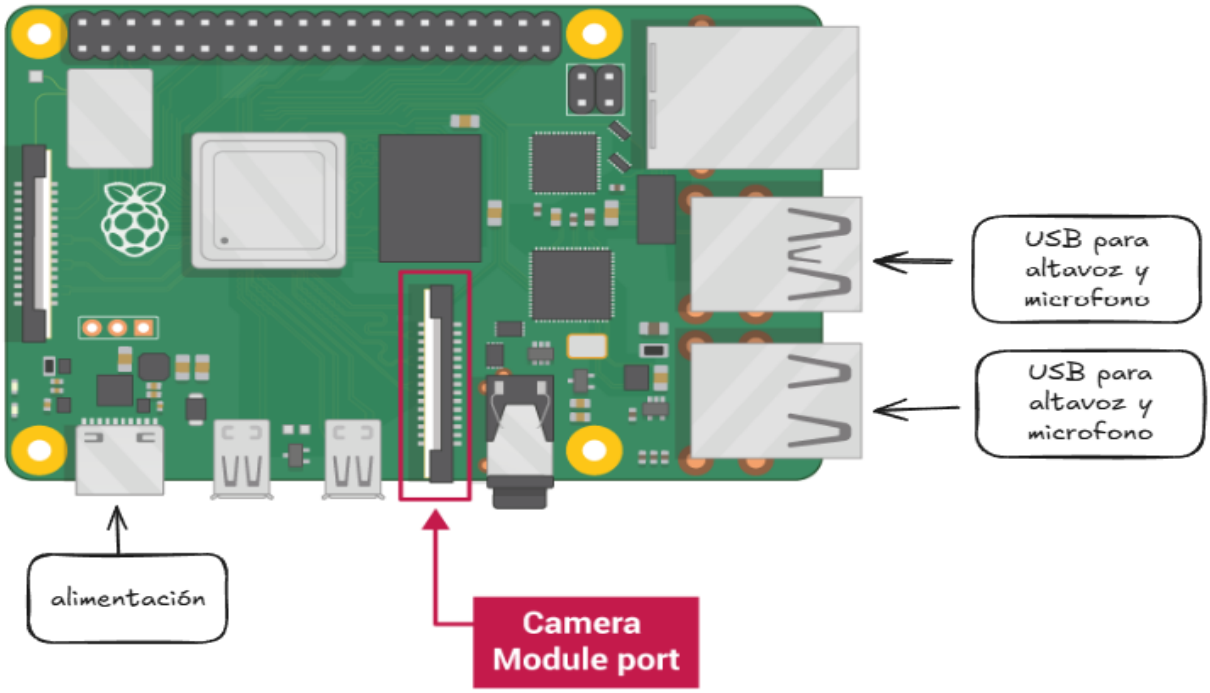


4.4. Diseño del Hardware

En este proyecto se usaron elementos que mas ser usados por necesidad del proyecto se usaron por disponibilidad prestada en presupuesto y demás factores.

- La Raspberry Pi 4 es una placa de computadora de bajo costo y alto rendimiento, desarrollada por la Raspberry Pi Foundation. Es la cuarta generación de la serie Raspberry Pi y ofrece mejoras significativas en comparación con sus predecesoras.
- La Picamera 2 es un módulo de cámara de alta definición diseñado para la Raspberry Pi, con un sensor Sony IMX219 de 8 megapíxeles que permite capturar fotos de alta resolución (3280x2464) y videos en 1080p30, 720p60 y VGA90. Es compatible con todos los modelos de Raspberry Pi y se conecta mediante un cable plano al puerto CSI. Su versatilidad la hace ideal para una amplia gama de aplicaciones, desde visión artificial y seguridad doméstica hasta fotografía time-lapse y proyectos de robótica.
- Altavoz genérico bluetooth, jack o USB
- Micrófono Genérico bluetooth, jack o USB

4.5. Diagrama esquemático del circuito



4.6. Diseño del Software

4.6.1. Traduccion

LibreTranslate es una API de traducción de código abierto que permite realizar traducciones mediante solicitudes HTTP, y lo mejor es que puede funcionar de manera offline al ser autoalojada. A continuación, se describen sus características y ventajas principales

Funcionalidad para el proyecto

- Autoalojamiento: Puedes instalar LibreTranslate en tu propio servidor o utilizar contenedores Docker. Esto te permite operar sin depender de servicios externos, lo que resulta ideal para proyectos que requieren alta privacidad y control sobre los datos.
- Independencia de Conexión: Al ejecutarse localmente, no necesitas conexión a Internet para traducir textos, lo cual es ventajoso en entornos con conectividad limitada o para aplicaciones internas.
- Interfaz RESTful: La API está diseñada para recibir y responder solicitudes HTTP. Por ejemplo, puedes enviar una solicitud POST al endpoint /translate con un cuerpo JSON que especifique el texto, el idioma de origen y el idioma de destino.
- Endpoints Adicionales: Además de la traducción, la API suele ofrecer endpoints para detectar el idioma del texto o listar los idiomas soportados, facilitando su integración en diversas aplicaciones.

Ventajas y Flexibilidad

- Código Abierto y Personalizable: Al ser open source, puedes revisar y modificar el código según tus necesidades, o incluso contribuir con mejoras a la comunidad.
- Soporte Multilingüe: LibreTranslate ofrece soporte para varios idiomas y, gracias a su naturaleza abierta, es posible ampliarlo según las demandas de la comunidad o del proyecto.
- Privacidad y Seguridad: La posibilidad de operarlo offline garantiza que los datos sensibles nunca salgan de tu entorno, lo cual es fundamental para aplicaciones que manejan información privada.

4.6.2. OCR

Tesseract es un motor de Reconocimiento Óptico de Caracteres (OCR) de código abierto que permite convertir imágenes que contienen texto en datos textuales editables. Inicialmente desarrollado por Hewlett-Packard y actualmente mantenido por la comunidad y Google, se ha consolidado como una herramienta eficaz y flexible para extraer información textual de imágenes.

Características Principales

- **Código Abierto:** Su naturaleza de código abierto permite su adaptación y personalización para diversas aplicaciones.
- **Compatibilidad Multiplataforma:** Funciona en diferentes sistemas operativos, facilitando su integración en múltiples entornos de desarrollo.
- **Soporte Multilingüe:** Tesseract soporta numerosos idiomas y puede ser entrenado para reconocer otros, ampliando su uso en contextos internacionales.
- **Precisión:** Con configuraciones y entrenamientos adecuados, ofrece una alta precisión en el reconocimiento de textos, incluso en imágenes complejas.

Aplicación en OCR Tesseract se utiliza para extraer texto de imágenes, lo cual resulta fundamental en una variedad de escenarios:

- **Digitalización de Documentos:** Permite transformar documentos físicos en archivos digitales, facilitando su almacenamiento y edición.
- **Automatización de Procesos:** Se integra en flujos de trabajo para extraer datos de facturas, recibos y formularios, reduciendo la necesidad de entrada manual.
- **Accesibilidad:** Contribuye a la creación de herramientas que convierten imágenes en texto para aplicaciones de lectura en voz alta, beneficiando a personas con discapacidades visuales.

4.6.3. voz a texto

La librería SpeechRecognition de Python es una herramienta de código abierto que facilita el reconocimiento de voz y la conversión de audio a texto. Proporciona una interfaz unificada para interactuar con diversos motores y servicios de reconocimiento de voz, permitiendo a los desarrolladores integrar capacidades de transcripción de audio en sus aplicaciones de manera sencilla.

Características Principales

- **Soporte para Múltiples APIs:** Permite la integración con varios servicios de reconocimiento de voz, tales como:
  - Google Web Speech API
  - IBM Speech to Text
  - Microsoft Azure Speech
  - Wit.ai
  - CMU Sphinx (modo offline)
- **Facilidad de Uso:** Su interfaz sencilla y bien documentada facilita la captura de audio desde archivos o directamente desde el micrófono.
- **Flexibilidad:** Ofrece la posibilidad de configurar parámetros como la sensibilidad del reconocimiento y la gestión de distintos formatos de audio.
- **Código Abierto:** Al ser de código abierto, se puede modificar y adaptar según los requerimientos específicos del proyecto.

Aplicaciones y Uso

- **Desarrollo de Asistentes Virtuales:** Integración de comandos de voz en asistentes virtuales y sistemas de control por voz.
- **Accesibilidad:** Conversión de voz a texto para mejorar la accesibilidad en aplicaciones y dispositivos, facilitando el uso a personas con discapacidades.
- **Automatización de Procesos:** Implementación de sistemas que reaccionan a comandos hablados, automatizando tareas en diversas aplicaciones.

Ejemplo de Uso A continuación se muestra un ejemplo básico de cómo utilizar la librería para capturar audio desde un micrófono y convertirlo a texto:

5. Implementación

5.1. Estructuración de carpetas

```
   /src
main.py
ApiManager.py
Realtime.py
OCR.py
VoiceRecognizer.py
requirements.txt
README.md
```

5.2. Pruebas unitarias

Las pruebas unitarias describen código modular separado en pequeñas tareas para previamente ser integrado al final en un código completo. Antes de realizar la prueba se hizo conexión de micrófono ya que como es claro el micrófono es nuestro punto de entrada.

5.2.1. Prueba unitaria voz

Untitled

```python
import speech_recognition as sr

def recognize():
    r = sr.Recognizer()
    with sr.Microphone() as source:
        print("Calibrando el micrófono para el ruido ambiente...")
        r.adjust_for_ambient_noise(source)  # Ajusta el umbral de ruido
        print("¡Di algo!")
        audio = r.listen(source)

        languages = ["es-ES", "en-US", "fr-FR", "de-DE", "it-IT", "pt-BR", "ja-JP", "zh-CN"]

        for lang in languages:
            try:
                print(f"Intentando reconocer en {lang}...")
                text = r.recognize_google(audio, language=lang)
                print(f"Idioma detectado: {lang}")
                print(f"Has dicho: {text}")
                return text, lang
            except sr.UnknownValueError:
                print(f"No se pudo entender el audio en {lang}")
            except sr.RequestError as e:
                print(f"Error en la solicitud para {lang}: {e}")
            except Exception as e:
                print(f"Ocurrió un error con {lang}: {e}")

        print("No se pudo detectar el idioma o reconocer el audio.")
        return None, None


text, detected_lang = recognize()
if text:
    print(f"Texto reconocido: {text} (Idioma: {detected_lang})")
else:
    print("No se pudo reconocer el audio.")
```

Untitled

```
Calibrando el micrófono para el ruido ambiente...
¡Di algo!
Intentando reconocer en es-ES...
Idioma detectado: es-ES
Has dicho: Hola cómo han estado hoy
Texto reconocido: Hola cómo han estado hoy (Idioma: es-ES)
```

5.2.2. Prueba unitaria OCR

Untitled

```python
import cv2
import pytesseract

def ocr_from_image(image_path):
    img = cv2.imread(image_path)
    if img is None:
        print("Error: No se pudo cargar la imagen")
        return
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _, threshold = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    text = pytesseract.image_to_string(threshold, lang='spa')
    return text

if __name__ == "__main__":
    image_path = "test.jpg"
    extracted_text = ocr_from_image(image_path)
    print("Texto extraído:")
    print(extracted_text)
```

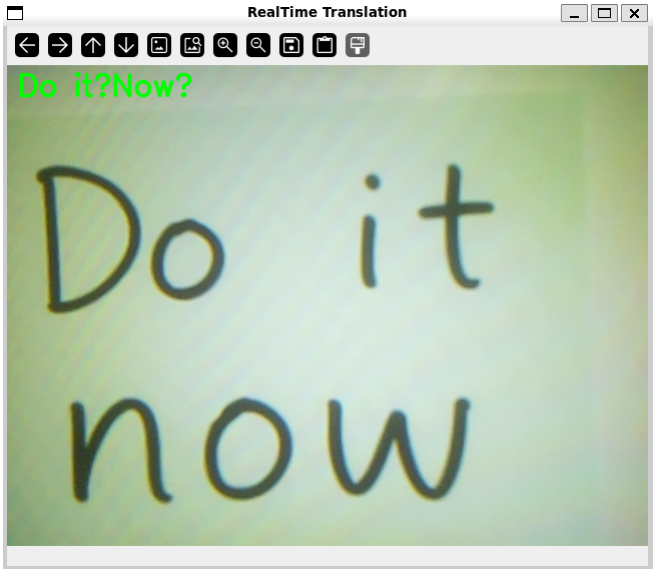
TRADUCTOR

```
Texto extraído:
I was so sad from
losing two of my

dogs and my mother.
T had this vision of all
these animals sitting
behind bars. They had
no control and were
scared. That's why 1
got into fostering and
adopting animals out.
- Linda Blair
```

5.2.3. OCR y camara tiempo real

Untitled

```python
import pytesseract
import cv2
import numpy as np
from picamera2 import Picamera2
from langdetect import detect

def take_photo():
    picam2 = Picamera2()
    picam2.start()
    image = picam2.capture_array()
    cv2.imwrite('image.jpg', image)
    return image

def preprocess(image):

    imagen = cv2.imread(image, cv2.IMREAD_GRAYSCALE)

    imagen = cv2.resize(imagen, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
    imagen = cv2.GaussianBlur(imagen, (5, 5), 0)
    imagen = cv2.adaptiveThreshold(imagen, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                   cv2.THRESH_BINARY, 11, 2)

    kernel = np.ones((2,2), np.uint8)
    imagen = cv2.morphologyEx(imagen, cv2.MORPH_CLOSE, kernel)

    cv2.imwrite("preprocesada.jpg", imagen)
    return imagen

def get_text(image):

    if isinstance(image, str):
        image = cv2.imread(image)

    try:

        multi_lang = 'eng+spa+fra+deu'
        text_inicial = pytesseract.image_to_string(image, lang=multi_lang, config='--oem 3 --psm 6')
        print("Texto inicial:")
        print(text_inicial)


        try:
            idioma_detectado = detect(text_inicial)
            print("Idioma detectado:", idioma_detectado)

            lang_map = {'en': 'eng', 'es': 'spa', 'fr': 'fra', 'de': 'deu'}
            tesseract_lang = lang_map.get(idioma_detectado, multi_lang)
        except Exception as e:
            print("No se pudo detectar el idioma, se usará OCR multi-idioma")
            tesseract_lang = multi_lang

        text_final = pytesseract.image_to_string(image, lang=tesseract_lang, config='--oem 3 --psm 6')
        print("Texto final:")
        print(text_final)
        return text_final

    except pytesseract.pytesseract.TesseractError:
        print("No se pudo extraer el texto de la imagen")
        return None
    except Exception as e:
        print(e)
        return None


take_photo()
preprocess('image.jpg')
get_text('preprocesada.jpg')
```



6. Problemas durante la implementación

Durante el desarrollo de este proyecto, uno de los mayores desafíos que enfrentamos fue equilibrar la precisión de los resultados con los recursos disponibles. En un mundo ideal, contaríamos con hardware de última generación y acceso ilimitado a servicios en la nube para procesar grandes volúmenes de datos en tiempo real. Sin embargo, la realidad es que trabajamos con limitaciones prácticas, tanto técnicas como económicas, que nos obligaron a tomar decisiones difíciles y a optimizar cada aspecto del sistema. La precisión, especialmente en tareas como el reconocimiento óptico de caracteres (OCR) y el reconocimiento de voz, fue un punto crítico. Aunque herramientas como Tesseract y SpeechRecognition son potentes y ampliamente utilizadas, su rendimiento depende en gran medida de la calidad de los datos de entrada. Por ejemplo, en el caso del OCR, si la imagen capturada tenía poca iluminación o el texto estaba distorsionado, la precisión disminuía notablemente. Esto nos llevó a implementar técnicas de preprocesamiento de imágenes, como el ajuste de contraste y el filtrado de ruido, que si bien mejoraron los resultados, también añadieron una carga adicional al procesamiento.

En cuanto al reconocimiento de voz, el ruido ambiental y los acentos regionales representaron otro obstáculo. Aunque ajustamos el sistema para que fuera más tolerante a estas variaciones, siempre hubo un margen de error que no pudimos eliminar por completo. Esto nos hizo reflexionar sobre la importancia de diseñar sistemas que no solo sean técnicamente robustos, sino también adaptables a las condiciones reales de uso, donde los usuarios no siempre están en entornos controlados.

Por otro lado, los recursos disponibles también jugaron un papel crucial en la evolución del proyecto. Trabajar con hardware limitado, como una Raspberry Pi o un ordenador portátil de gama media, nos obligó a priorizar ciertas funcionalidades sobre otras. Por ejemplo, tuvimos que decidir entre procesar todo localmente, lo que garantizaba mayor privacidad pero consumía más recursos, o enviar datos a servidores externos, lo que aumentaba la eficiencia pero introducía preocupaciones sobre la seguridad y la latencia.

Estas limitaciones no solo fueron técnicas, sino también humanas. Como equipo, tuvimos que aprender a gestionar nuestro tiempo y energía de manera más eficiente, priorizando las tareas que tenían mayor impacto en el resultado final. Esto nos enseñó una lección valiosa: en el mundo real, la perfección no siempre es alcanzable, y a veces es más importante entregar una solución funcional y útil que perseguir un ideal inalcanzable.

7. Conclusiones

- Los procesos de OCR son muy costosos computacional mente por esto se hace con modelos mas complejos desarrollados privadamente por cada empresa
- Linux es un entorno muy amplio que tiene un seguimiento por la comunidad muy grande.
- Raspberry pi ofrece un entorno muy amplio y con buen soporte para hacer prototipado.
- El proyecto se estructuró en módulos independientes (OCR, VoiceRecognizer, ApiManager, Realtime), facilitando el mantenimiento y la escalabilidad.
- Se integraron herramientas como Tesseract (OCR), SpeechRecognition (voz), OpenCV (procesamiento de imágenes) y APIs de traducción, demostrando su efectividad en aplicaciones de procesamiento de lenguaje natural.
- Se sugiere añadir soporte para más idiomas, optimizar el rendimiento en tiempo real y mejorar la precisión del OCR en condiciones de iluminación variable.