Tecnológico Nacional de México Campus

Costa Grande

Carrera

Ingeniería en Sistemas Computacionales

Plan ISIC-2010-224

Módulo de especialidad

Tecnologías e Innovación

Nombre del proyecto

Robot Asistente Educativo Digital Luminia

Alumno

César Augusto Andrade Pérez

Fecha de elaboración

22 de diciembre de 2024

ÍNDICE DE CONTENIDO

1. INTRODUCCIÓN

Este manual detalla a nivel técnico los aspectos físicos y lógicos por los cuales opera el robot asistente Luminia, mencionando las consideraciones y razones que se tomaron en cuenta para la realización de ciertos aspectos en su diseño tanto físico como lógico.

Se recomienda que se tengan **conocimientos dominados o experiencia intermedia acerca de robótica, circuitos, manejo de Linux y programación en Python como mínimo**, muchas de las cosas que se hablan en este documento son muy técnicas, puede que no entiendan nada si no poseen tales conocimientos.

El manual esta dividido en principalmente en 4 partes, la idea principal con la que se construyó el robot, el estado del arte donde se detallan todos los conceptos que involucra el proyecto, el apartado físico que es el hardware, el material que fue utilizado para su estructura, los motores, computadoras y características de estos detalladamente y el apartado lógico que es el software del robot mismo, como opera, que paquetes y recursos se tuvieron que instalar en este, así como su proceso de instalación y como este determina los movimientos del robot, etc.

Adicionalmente cuenta con un **apartado donde se detallan los problemas que se encontraron en el camino y lo que se probó con anterioridad**, pero no quedo implementado por la razón que sea correspondiente, así como **un apartado de sugerencias por la cual el proyecto podría ser mejorado en gran medida**, y que por razones ya sean económicas o técnicas no se realizaron en la versión actual del robot, también habrá **un apartado para el presupuesto del robot**, el costo de todos los materiales para el caso de que se deseen producir más.

Por último, pero no menos importante, habrá un **apartado de precauciones que hay que tener en cuenta al momento de operar o modificar el robot**, esto con tal de no dañar los componentes ya implementados en este mismo, **bajo ninguna circunstancia deberá de ignorarse este apartado de precauciones si van a modificar u operar cualquier aspecto del robot mismo como tal**, de no acatar dichas precauciones se deberá de negar el acceso a modificar u operar el robot por motivos de seguridad.

2. IDEA PRINCIPAL DEL PROYECTO

La idea por la que se desarrolló este robot, consiste de un tipo de animatrónica potenciado por inteligencia artificial para realizar movimientos con sus manos a forma de llamar la atención y dar expresión en sus respuestas generadas por la IA.

Su trabajo en cuestión consiste en ser de asistencia a los alumnos universitarios para la realización de cualquier trámite escolar, uno como estudiante va y presiona un botón para poder hablar con el robot, de ser necesario presionaría el mismo botón otra vez para hacer que el robot empiece a procesar lo que se le dijo, evitando así que ruidos externos influyan en su entendimiento, después generaría una respuesta la cual reproducirá por sus bocinas integradas mientras mueve sus manos en la respuesta.



El robot almacena todo lo que se le pregunta para tener contexto de lo que se le ha preguntado, de esta forma se puede entablar una conversación con este de un mismo tema de forma extensiva, este contexto no es permanente, hay 2 momentos en donde se borraría todo contexto para no influir en la generación de nuevas respuestas de otros temas.

Si el robot no recibe ninguna respuesta de alguien durante un tiempo determinado (configurado por defecto que sean 2 minutos), el robot generará una respuesta para llamar

la atención de quienes estén alrededor presentándose a si mismo con la información proporcionada, cuando hace eso, borrará todo contexto previo que tenga almacenado.

Si al robot se le despide o agradece por haber cumplido con su labor generará una despedida lo cual en teoría debería borrar el contexto anterior de igual forma para que se le haga una petición diferente, esto puede fallar sin embargo debido a que depende de que la IA ponga el comando en su respuesta para realizar esa acción, y eso puede variar según la generación de respuesta de la IA.

3. ESTADO DEL ARTE

Antes de empezar a detallar todo lo que consiste el robot mismo, es esencial que se conozcan las ideas y conceptos por las cuales opera el robot, en este apartado se encuentra todos las ideas y conceptos que se tomaron en consideración para la elaboración del proyecto.

3.1. Grandes Modelos de Lenguaje (LLM)

Los **Grandes Modelos de Lenguaje (LLM, por sus siglas en inglés)** son un tipo de inteligencia artificial diseñada para comprender y generar texto de manera similar a como lo haría un ser humano. Su propósito es trabajar con el lenguaje natural, es decir, con el tipo de texto que se usa en la vida cotidiana, para tareas como responder preguntas, traducir idiomas, redactar textos o incluso realizar conversaciones.

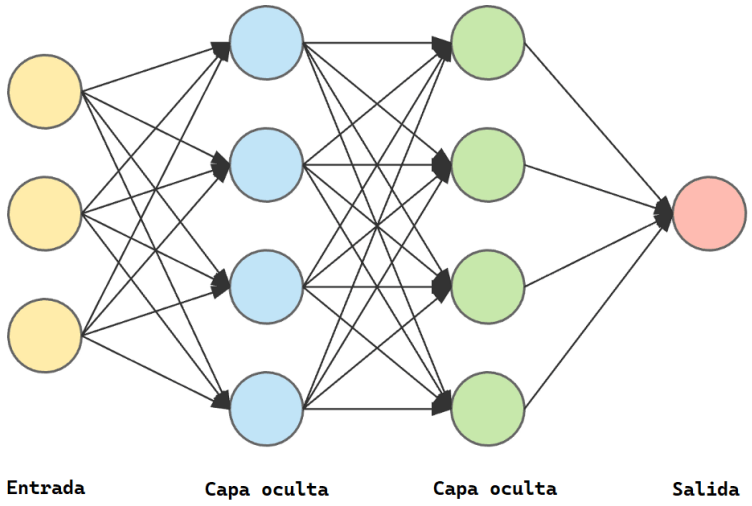
¿Cómo funcionan?

1. **Entrenamiento con grandes cantidades de datos**: Los LLM se entrenan utilizando una enorme cantidad de texto de internet, como libros, artículos, sitios web y otros recursos. Esto les permite aprender patrones del lenguaje, relaciones entre palabras, contexto y cómo construir frases coherentes. Este proceso de aprendizaje es similar a cómo los humanos aprenden un idioma: al exponerse a muchas palabras y estructuras, se va comprendiendo cómo usarlas.
2. **Modelado basado en predicción**: Los modelos funcionan prediciendo cuál será la siguiente palabra en una oración. Por ejemplo, si se les da la frase "El cielo es...",

intentarán adivinar la próxima palabra, como "azul". Al repetir este proceso una y otra vez, son capaces de generar textos largos y responder a preguntas de manera bastante coherente.

3. **Capas de redes neuronales**: En su núcleo, estos modelos usan redes neuronales profundas, que son sistemas inspirados en el cerebro humano. Cada "capa" de esta red examina diferentes aspectos del texto, como la gramática, el significado, las relaciones entre palabras y el contexto general. Cuantas más capas tenga un modelo, más complejo es su análisis y mayor su capacidad de comprensión y generación de texto.

   Esto es un ejemplo de una red neuronal chica y simple que recibe 3 entradas y da una salida:



4. **Ajuste fino**: Después del entrenamiento general, algunos LLM se "afinan" con datos específicos para que sean útiles en tareas particulares. Por ejemplo, un modelo puede ser ajustado para responder preguntas técnicas o para realizar análisis de sentimiento (es decir, identificar si un texto expresa emociones positivas, negativas o neutras).

Esto es un proceso muy extenso que puede tomar mucho tiempo en desarrollarse, pero hay soluciones al respecto para usar este tipo de tecnologías en varias aplicaciones, como el sitio web Hugging Face.

3.2. Sitio Web Hugging Face

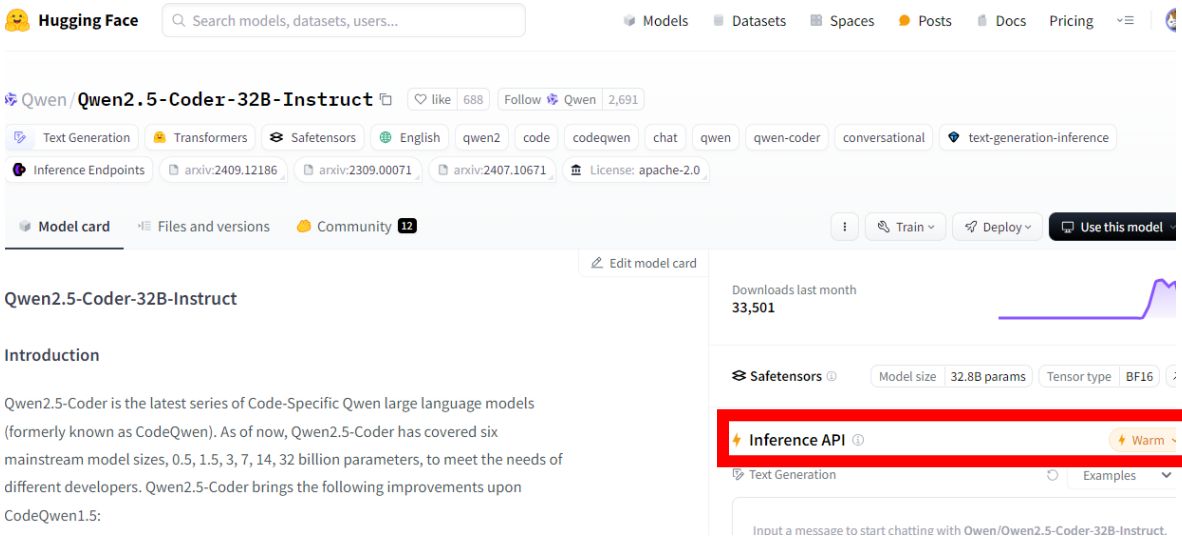
Hugging Face es una plataforma popular en el mundo de la inteligencia artificial, especializada en herramientas y servicios para el procesamiento de lenguaje natural (PLN), pero que ha crecido para ofrecer soluciones en muchas áreas relacionadas con los modelos de aprendizaje profundo. En términos simples, Hugging Face permite a desarrolladores y empresas acceder a modelos avanzados de IA sin necesidad de empezar desde cero.

HUGGING FACE

Hugging Face pone a disposición más de 1 millón de modelos entrenados para su uso a través de su servidor, pero no todos estos pueden ser accedidos a través del servidor, pues los que son más demandados y usados son los que están cargados en su servidor, por lo que es importante seleccionar el modelo que se va a utilizar tomando eso en consideración, de no estar cargado en su servidor se tendrá que descargar y manejar de forma local para realizar la **inferencia** a estos modelos IA.

¿Cómo saber si un modelo está cargado en el servidor?

Para saber si un modelo está cargado en el servidor de Hugging Face hay forma sencilla de saberlo, en su sitio web al momento que se selecciona un modelo, en la parte derecha del sitio debe haber algo que diga “Inference API” y delante puede estar “Warm”, “Cold” o “Frozen”.

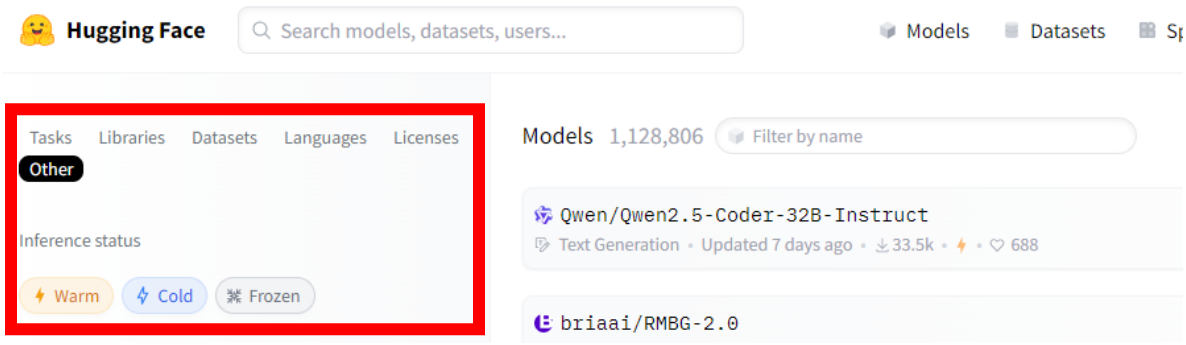


Si no está presente ese apartado al momento de visualizar un modelo, quiere decir que está en un estatus “Frozen”, que quiere decir que el modelo no está disponible para usarse en el servidor porque no tiene tanta atención, si la atención incrementa puede haber un chance de que se vuelva mínimamente “Cold”.

Si un modelo esta “Cold”, quiere decir que está disponible para usarse, pero puede que tome un tiempo generar la primera o varias respuestas debido a que debe cargar el modelo al servidor, ya que no hay tanta actividad en este modelo para que este cargado todo el tiempo en el servidor.

Cuando un modelo esta “Warm”, quiere decir que está listo para ser usado en cualquier momento por la **API de inferencia**, el único problema que puede haber al usar estos modelos es que si es demandado demasiado puede que tarde un poco en responder a la solicitud, pues mucha gente la está usando.

Se puede usar el filtrador del sitio web al buscar modelos para obtener resultados de modelos que estén listos para usarse según sean las necesidades.



¿Qué es una inferencia?

Se ha mencionado la palabra inferencia y API de inferencia con anterioridad, por lo que hay que explicar que es una inferencia.

Inferencia en el contexto de la inteligencia artificial se refiere al proceso de usar un modelo entrenado para hacer predicciones o generar resultados basados en nuevos datos. Por ejemplo, si tienes un modelo que se entrenó para responder preguntas, hacer una "**inferencia**" significa enviarle una pregunta y recibir la respuesta generada.

Servicios que ofrece Hugging Face

1. **Model Hub**:
    - Es una biblioteca gigantesca de modelos de inteligencia artificial listos para usar. Estos modelos han sido creados y compartidos por la comunidad, empresas e investigadores, cubriendo tareas como traducción de texto, análisis de sentimiento, generación de texto, clasificación de imágenes y más.
    - Los usuarios pueden realizar **inferencia en línea** directamente desde el sitio web. Por ejemplo, puedes escribir una frase para que un modelo la traduzca o la clasifique sin necesidad de descargar el modelo.
2. **Transformers Library**:
    - Hugging Face creó la biblioteca **Transformers**, que permite a los desarrolladores usar modelos avanzados de PLN y otros tipos con gran facilidad en sus aplicaciones de Python.

- Aunque es posible descargar modelos para usarlos localmente, Hugging Face también ofrece formas de probarlos en línea sin necesidad de configuración local.

3. **Spaces**:
    - Este servicio permite a los usuarios crear y alojar aplicaciones interactivas impulsadas por IA usando herramientas como Gradio o Streamlit. Por ejemplo, puedes crear un demo de un chatbot, un sistema de reconocimiento de voz o un analizador de texto, y compartirlo con otros para que lo prueben directamente desde su navegador.
4. **Inference API**:
    - Si deseas incorporar capacidades de IA en tu propia aplicación, puedes usar la **API de Inferencia** de Hugging Face. Con ella, puedes realizar consultas a modelos alojados en los servidores de Hugging Face sin necesidad de gestionar servidores o descargar archivos pesados.
    - Esto es particularmente útil para tareas como análisis de texto, generación de contenido, clasificación de imágenes y mucho más.
5. **Datasets**:
    - Hugging Face también ofrece una gran colección de conjuntos de datos que se pueden usar para entrenar o evaluar modelos. Todo está disponible en línea, y puedes explorarlos sin necesidad de descargarlos en tu máquina.
6. **AutoNLP**:
    - Un servicio que facilita a los usuarios crear y ajustar modelos de lenguaje personalizados sin necesidad de experiencia profunda en entrenamiento de modelos de IA. Los usuarios cargan sus datos y el servicio se encarga del resto.

En resumen, **Hugging Face** facilita a los desarrolladores y usuarios acceder a la inteligencia artificial avanzada sin necesidad de descargas complicadas o infraestructuras

pesadas, es decir, maquinas con alta capacidad computacional. Ya sea realizando **inferencias en línea**, integrando modelos mediante APIs, o explorando datasets, ofrece un ecosistema poderoso y accesible para todos.

3.3. Generación Aumentada por Recuperación (RAG)

La **Generación Aumentada por Recuperación (RAG, por sus siglas en inglés)** es una técnica poderosa que combina modelos de lenguaje, como los **Grandes Modelos de Lenguaje (LLMs)**, con la capacidad de buscar información relevante en bases de datos o documentos externos para proporcionar respuestas más precisas y actualizadas.

¿Qué es y cómo funciona RAG?

Cuando un LLM recibe una pregunta o solicitud, este generalmente genera respuestas basadas únicamente en el conocimiento con el que fue entrenado. Esto puede ser problemático cuando se necesita información específica, actualizada o contextualizada que no estaba presente en su entrenamiento original. Aquí es donde entra en juego la **Generación Aumentada por Recuperación**:

1. **Recuperación de información**:
    - En lugar de depender únicamente del conocimiento estático del LLM, RAG primero busca en una base de datos de documentos o información relacionada para encontrar contenido relevante.
    - Esta fase de búsqueda o "recuperación" se realiza mediante un sistema que filtra los documentos más relevantes basándose en la consulta del usuario. Por ejemplo, si preguntas sobre "las últimas regulaciones de energía renovable en 2023", el sistema buscará documentos, artículos o bases de datos que contengan esta información.
2. **Generación de respuesta con contexto**:
    - El LLM toma la información recuperada de la base de datos y genera una respuesta, usando tanto su conocimiento como el contenido adicional encontrado. Esto asegura que la respuesta no solo sea coherente, sino también precisa y específica para el contexto solicitado.

- En esencia, el LLM "se apoya" en el contenido recuperado para construir una respuesta que no podría haber dado de manera tan precisa basándose solo en su entrenamiento previo.

Tipos de bases de datos que RAG utiliza

Para proporcionar información precisa, RAG puede conectarse a diferentes tipos de bases de datos o fuentes de información, tales como:

1. **Bases de datos estructuradas**:
   - Bases de datos SQL u otros formatos relacionales que contienen registros organizados. Ejemplo: inventarios, registros de productos, bases de clientes.
2. **Índices de texto**:
   - Se utilizan sistemas de búsqueda como **Elasticsearch** para indexar y buscar rápidamente en documentos no estructurados como textos largos, artículos o documentos legales.
3. **Documentos almacenados en formatos como JSON, CSV o archivos de texto plano**:
   - RAG puede buscar en archivos con diferentes formatos para encontrar respuestas relevantes.
4. **Bases de Datos Vectoriales (Vector Store)**:
   - Almacenan representaciones vectoriales de documentos o fragmentos de texto, permitiendo búsquedas de similitud semántica. Herramientas como **Faiss** o **Pinecone** se utilizan para hacer coincidir consultas con textos similares de manera eficiente, permitiendo a los LLM encontrar el contenido que mejor se ajusta al contexto de la pregunta.
5. **Sistemas de gestión de conocimiento**:
   - Pueden integrarse con plataformas que almacenan documentación empresarial, manuales técnicos u otros recursos clave para proporcionar respuestas precisas.

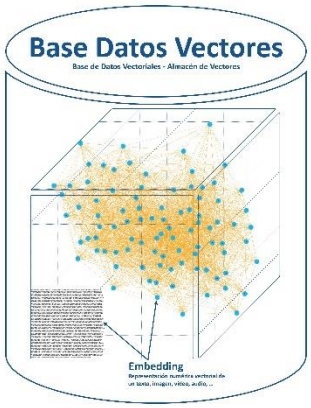
Beneficios de RAG

- **Precisión**: Las respuestas generadas por LLMs pueden basarse en datos frescos o específicos, en lugar de depender de conocimientos "antiguos" o generales.
- **Contextualización**: Los LLMs pueden proporcionar respuestas más adaptadas al contexto del usuario gracias a la información adicional.
- **Escalabilidad**: Se puede conectar a bases de datos muy grandes sin necesidad de reentrenar constantemente el modelo.

En resumen, la **Generación Aumentada por Recuperación (RAG)** combina la potencia de los **Grandes Modelos de Lenguaje** con el acceso a fuentes de datos específicas, permitiendo generar respuestas más útiles y precisas al proporcionar información complementaria en tiempo real. Esto es especialmente valioso para casos donde la precisión y el acceso al conocimiento más reciente son clave.

3.4. Bases de Datos Vectoriales (Vector Store)

Las **bases de datos vectoriales** o **"vector stores"** son un tipo de base de datos especializada para almacenar y buscar datos representados como vectores, que son listas de números que capturan características importantes de un objeto, como palabras, frases, imágenes o cualquier dato que se pueda representar en un espacio numérico. Este tipo de base de datos ha cobrado gran relevancia en el campo de la **inteligencia artificial (IA)** porque permite realizar búsquedas basadas en **similitud semántica**, lo que significa que puede encontrar elementos que tengan un significado parecido en lugar de simplemente coincidir de manera exacta.



El proyecto utiliza Base de Datos Vectoriales (Vector Store) para el método RAG en conjunto con el LLM, esto debido a que se puede usar de una IA para hacer la búsqueda en este tipo de base de datos de una forma mucho más sencilla que con textos o documentos sueltos y bases de datos relacionales.

¿Cómo funcionan las bases de datos vectoriales?

1. **Conversión de datos a vectores**:
    - Antes de almacenar datos, se convierten en **vectores** mediante un modelo de IA (como los LLMs o modelos de embeddings) que puede ser extraído de Hugging Face. Por ejemplo, si tienes un conjunto de frases, cada una se transformará en un vector que representa su significado. Palabras o frases que tienen significados similares estarán representadas por vectores cercanos entre sí en el espacio vectorial, es como calcular la distancia entre ellos y obtener los más cercanos que deberían ser los más relevantes.
2. **Almacenamiento de vectores**:
    - Los vectores resultantes se almacenan en la base de datos vectorial. Estas bases de datos están optimizadas para manejar grandes cantidades de datos en forma de vectores y permiten buscar con rapidez elementos similares.
3. **Búsqueda por similitud**:
    - Cuando un usuario busca algo (como una frase o consulta), la base de datos convierte esta consulta en un vector y compara su similitud con los vectores almacenados usando medidas como la **distancia coseno** o la **distancia euclidiana**.
    - El sistema devuelve resultados que son los "más cercanos" al vector de la consulta, es decir, aquellos que tienen mayor similitud semántica. Esto es muy útil cuando no se busca una coincidencia exacta, sino resultados que se "parezcan" a lo que se pidió.

Las **bases de datos vectoriales** son esenciales para realizar búsquedas semánticas en datos complejos, lo que significa buscar elementos que sean similares en su **significado** en lugar

de ser idénticos en su forma. Usando representaciones vectoriales y modelos de IA, estas bases de datos permiten a las aplicaciones ofrecer resultados más relevantes, personalización y comprensión más profunda del lenguaje. Esto facilita la creación de soluciones inteligentes que "entienden" mejor las consultas de los usuarios y proporcionan respuestas precisas y contextualmente significativas.

3.5. Sintetizador de Voz (TTS)

Un **sintetizador de voz**, también conocido como **TTS (Text-to-Speech)**, es una tecnología que convierte texto escrito en voz hablada. Básicamente, toma palabras escritas y las "lee" de manera que las personas puedan escucharlas, generando una salida de audio que suena como si una persona estuviera hablando. Los sintetizadores de voz son muy útiles en aplicaciones como asistentes virtuales, navegadores GPS, software de accesibilidad, dispositivos inteligentes, entre otros.

¿Qué tipo de IA utiliza el TTS?

Los sintetizadores de voz modernos emplean **inteligencia artificial (IA) basada en modelos de aprendizaje profundo**, específicamente en técnicas de **redes neuronales**. Estos modelos de IA están diseñados para aprender cómo "hablar" a partir de grandes cantidades de datos de voz grabada y texto, lo que les permite generar audio realista y de alta calidad.

¿Cómo funciona un TTS con IA?

1. **Entrada de texto**:
    - El sistema recibe un texto que debe ser convertido a voz.
2. **Análisis de texto**:
    - El TTS analiza el texto, dividiéndolo en palabras, frases y otras unidades lingüísticas. También puede analizar la sintaxis para comprender cómo debe sonar (tono, entonación, etc.).
3. **Conversión a representaciones fonéticas**:

- Las palabras se convierten en representaciones fonéticas (sonidos que representan cómo se pronuncian las palabras). Aquí entra en juego la IA para asegurarse de que la pronunciación sea precisa.

4. **Generación de audio**:

   - Usando modelos de red neuronal, el sistema sintetiza el sonido de cada palabra y las une en frases. Las redes neuronales más avanzadas pueden generar audio de manera que suene fluido, ajustando la entonación, el ritmo y otros elementos para crear una experiencia auditiva más realista.

En resumen, los **sintetizadores de voz (TTS)** modernos emplean IA avanzada para convertir texto en voz con una naturalidad y precisión sorprendentes. Utilizan redes neuronales y técnicas de aprendizaje profundo para producir un habla fluida, lo que los convierte en herramientas esenciales para aplicaciones que requieren interacción de voz humana con tecnología.

3.6. Reconocedor de Voz (STT)

Un **reconocedor de voz** o **STT (Speech-to-Text)** es una tecnología que toma el habla humana (audio) y la convierte en texto escrito. Es la tecnología detrás de comandos de voz, transcripciones automáticas, asistentes virtuales y aplicaciones que permiten la interacción mediante la voz. Un sistema de STT interpreta el sonido de una voz humana, lo procesa y genera una transcripción precisa en texto, todo en tiempo real o casi en tiempo real.

El uso de IA en el STT

Los reconocedores de voz actuales utilizan **inteligencia artificial (IA) avanzada** para procesar, comprender y transcribir el lenguaje hablado con precisión. La IA detrás de estos sistemas suele emplear **redes neuronales profundas** y otros métodos de aprendizaje automático que han revolucionado la capacidad de las máquinas para interpretar el lenguaje humano.

Cómo funciona un STT con IA

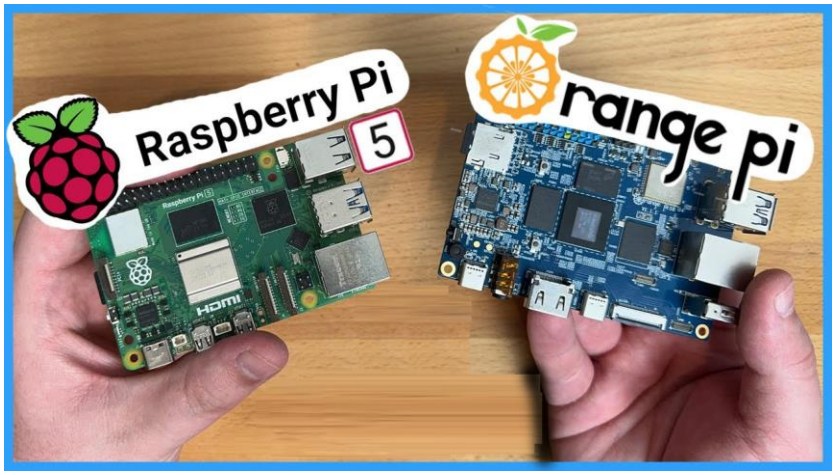
1. **Captura de audio**:
    - El sistema recibe una señal de audio a través de un micrófono o un archivo de audio. Esta señal es un flujo de datos de onda que contiene las frecuencias de sonido del habla.
2. **Conversión de audio a espectrogramas**:
    - Para que el modelo de IA pueda trabajar con el sonido, la señal de audio se convierte en un **espectrograma**, que es una representación visual de las frecuencias sonoras a lo largo del tiempo. Este proceso ayuda a "ver" las características del sonido y permite que los modelos de IA reconozcan patrones en los datos.
3. **Reconocimiento de patrones mediante IA**:
    - Los sistemas de STT modernos utilizan **redes neuronales profundas**, como las redes **recurrentes (RNN)**, redes **transformer**, o modelos como **Convolutional Neural Networks (CNN)** y **Long Short-Term Memory (LSTM)** para procesar el espectrograma. Estas redes han sido entrenadas previamente con millones de muestras de voz para reconocer cómo suenan diferentes palabras, acentos, tonos y velocidades del habla.
    - Usando estos modelos, el sistema puede identificar y distinguir palabras y sonidos del habla humana.
4. **Conversión de audio a texto**:
    - A medida que la red neuronal reconoce los patrones de sonido, convierte cada segmento de audio en una palabra o una frase. Este proceso de "decodificación" permite que el sistema transcriba el discurso hablado en texto.
    - La IA puede tener en cuenta el contexto para mejorar la precisión, asegurándose de que las palabras detectadas sean coherentes y relevantes.

5. **Post-procesamiento y ajuste contextual**:
    - El texto generado puede ser ajustado para tener en cuenta errores comunes, homófonos, nombres propios y contexto gramatical. Por ejemplo, si alguien dice "voy a casa", un sistema bien entrenado evitará transcribir "voy a caza" si el contexto no encaja.

El **reconocedor de voz (STT)** es una tecnología impulsada por **inteligencia artificial** que convierte el lenguaje hablado en texto escrito. Utiliza **redes neuronales profundas** y técnicas de aprendizaje profundo para analizar señales de audio, reconocer patrones y traducirlas a texto de manera precisa. Gracias al uso de IA, los sistemas de STT pueden comprender diferentes acentos, reconocer el contexto de una conversación y mejorar continuamente, lo que los hace cada vez más precisos y naturales.

3.7. Placas de Desarrollo Programables

Las **placas de desarrollo programables** son dispositivos electrónicos que proporcionan un entorno flexible y accesible para crear y controlar una amplia variedad de proyectos electrónicos. Están diseñadas para ser utilizadas por entusiastas de la electrónica, estudiantes, profesionales y makers en el desarrollo de prototipos de sistemas embebidos, automatización, robótica y otras aplicaciones. Ejemplos populares de estas placas son **Arduino, Raspberry Pi y Orange Pi**. Sin embargo, a diferencia de Arduino, que se enfoca principalmente en el control de hardware mediante microcontroladores, **Raspberry Pi** y **Orange Pi** son computadoras completas en miniatura con capacidades mucho más avanzadas.



Diferencias y enfoque de Raspberry Pi y Orange Pi en robótica

Raspberry Pi

- **Raspberry Pi** es una computadora de placa única (SBC, por sus siglas en inglés) que combina hardware y software para proporcionar un entorno de desarrollo potente y versátil. A diferencia de Arduino, que es un microcontrolador, Raspberry Pi funciona con sistemas operativos completos, como **Raspberry Pi OS (basado en Linux)**, lo que permite ejecutar aplicaciones complejas, scripts en diferentes lenguajes y hasta interfaces gráficas.
- En el contexto de la **robótica**, Raspberry Pi se usa ampliamente debido a su capacidad para realizar múltiples tareas, conectarse a sensores y dispositivos externos mediante sus pines GPIO, ejecutar algoritmos avanzados (como reconocimiento de imágenes y visión por computadora) y procesar datos de manera eficiente.

Orange Pi

- **Orange Pi** es otra **computadora de placa única** que comparte muchas similitudes con Raspberry Pi, pero ofrece una variedad más amplia de modelos con diferentes capacidades de hardware, como procesadores más potentes o más memoria, a menudo a precios competitivos. Orange Pi puede ejecutar sistemas operativos basados en Linux, Android y otros, lo que la convierte en una opción flexible para desarrollos avanzados.
- En **robótica**, Orange Pi se destaca por ser una alternativa potente y asequible que puede realizar tareas similares a las de Raspberry Pi, como controlar servomotores, gestionar sensores y ejecutar algoritmos complejos. Además, al soportar sistemas de inteligencia artificial (IA) y procesamiento de imágenes, Orange Pi puede ser utilizada en robots que requieren **reconocimiento de voz, procesamiento de imágenes o navegación autónoma**.

Ventajas de Raspberry Pi y Orange Pi en la robótica

1. **Capacidades de procesamiento**:
    - Ambas placas ofrecen más potencia de cómputo que microcontroladores tradicionales (como los de Arduino), lo que es ideal para ejecutar algoritmos avanzados, procesar imágenes y manejar múltiples procesos simultáneamente.
2. **Conectividad**:
    - Soporte para Wi-Fi, Ethernet y Bluetooth, lo que facilita la creación de robots conectados en red, controlables de manera remota o interactuando con otros dispositivos.
3. **Interfaz de pines GPIO (General Purpose Input/Output)**:
    - Tanto Raspberry Pi como Orange Pi tienen pines GPIO que permiten la conexión con sensores, motores, LEDs y otros componentes electrónicos, haciendo posible el control directo de hardware para tareas robóticas.
4. **Flexibilidad y compatibilidad con módulos y periféricos**:
    - Estas placas pueden interactuar con una amplia variedad de periféricos y módulos, como cámaras, pantallas, sensores ultrasónicos y módulos de comunicación (Wi-Fi, módulos Zigbee), expandiendo las capacidades de los proyectos robóticos.
5. **Soporte para lenguajes de programación populares**:
    - Puedes programar en lenguajes como **Python, C, C++, Java y más**, lo que facilita el desarrollo de software para sistemas robóticos, incluyendo algoritmos de control, lógica de navegación, sistemas de percepción, y más.

Comparación con Arduino

- **Arduino** es excelente para el control de bajo nivel y tareas sencillas que requieren control directo de pines, como encender LEDs, controlar motores de manera básica

o leer valores de sensores. Sin embargo, tiene limitaciones en cuanto a potencia de procesamiento y multitarea.

- **Raspberry Pi y Orange Pi** permiten ejecutar aplicaciones más complejas, almacenar y procesar grandes cantidades de datos y ejecutar sistemas operativos completos, lo que les brinda una ventaja significativa en la **robótica avanzada**, donde se requieren capacidades de procesamiento, visión artificial, IA y conectividad de red.

Raspberry Pi y Orange Pi son herramientas muy versátiles y potentes para la robótica debido a su capacidad para manejar tareas avanzadas, su conectividad, y su facilidad para trabajar con otros módulos y componentes. Estas placas permiten a los desarrolladores construir robots que van desde sistemas simples hasta soluciones complejas, con capacidades de procesamiento que antes solo eran accesibles en sistemas mucho más grandes y costosos.

3.8. Servo Motores

Probablemente ya conoces que es un servo motor si estas en el área de la electrónica o robótica, en caso de no conocerlo se explicará después de este párrafo, como este proyecto utiliza de ellos masivamente para la animación del robot como una animatrónica pues es de suma importancia abarcar unos temas importantes acerca de su uso, capacidades y características específicas de estos, pues no cualquier servo motor puede ser usado para el robot, se necesita que tengan capacidades específicas para su correcto funcionamiento y evitar accidentes como quemar uno de estos.

Un **servo motor** es un dispositivo electromecánico que permite controlar el movimiento preciso de rotación o el ángulo de un eje o un mecanismo. Los servos son ampliamente utilizados en proyectos de robótica, automatización, control de movimiento, y en aplicaciones donde se necesita precisión, como brazos robóticos, robots móviles, aviones a control remoto, y más. A diferencia de los motores convencionales, un servo motor no gira continuamente (a menos que sea un servo modificado); en su lugar, tiene la capacidad de moverse a una posición específica con precisión.

Características de los servo motores

1. **Control preciso de posición**:
    - Los servos tienen un sistema de control incorporado que permite mover el eje del motor a una posición angular específica. Este control se logra mediante señales de **PWM (modulación por ancho de pulso)**. La duración del pulso determina la posición del eje.
2. **Sistema de retroalimentación**:
    - Los servos suelen tener un potenciómetro o sensor de posición que envía información al controlador sobre la posición actual del eje. Esto permite que el motor ajuste su posición en tiempo real según las órdenes que recibe.
    - Aunque el servo motor posee de este sistema para ajustar su propio eje en la posición correcta, este mismo no puede ser utilizado para obtener información de la rotación actual del servo motor, ya que este siempre esta siendo rotado a la posición ordenada, por lo que no es necesario dicha funcionalidad.
3. **Velocidad y par (torque)**:
    - El **torque** es la fuerza que produce el servo para girar o mantener una posición bajo carga, y es una de las características más importantes de un servo. Se mide generalmente en **kilogramos-centímetro (kg-cm)**.
        - **1 kg-cm** significa que el servo puede aplicar una fuerza de 1 kg a una distancia de 1 cm desde el eje de rotación.

        - Por ejemplo, un servo con un torque de **5 kg-cm** puede mover o mantener una carga de 5 kg a 1 cm de distancia del eje de rotación, o una carga de 1 kg a 5 cm de distancia.

    - El torque indica la **potencia de giro** y determina la capacidad del servo para manejar cargas pesadas.

    - Muchos no poseen conocimiento de este aspecto importante de los servo motores, por ello **es importante recalcar que se debe medir bien la carga que llevará el servo motor para evitar problema alguno al manejar estos**.

4. **Rango de movimiento**:

    - La mayoría de los servos comunes tienen un rango de movimiento limitado, generalmente de **0 a 180 grados**, aunque hay servos que permiten rotar hasta 270 grados o más, e incluso algunos de rotación continua.

5. **Tipo de alimentación**:

    - Los servos suelen alimentarse con **5V a 6V**, aunque algunos pueden requerir voltajes más altos. La corriente que consumen depende del tamaño y la carga que manejen.

Pautas correctas de uso para evitar dañar un servo motor

Para usar un servo de manera efectiva y evitar dañarlo, sigue estas pautas:

1. **No exceder el torque máximo**:

    - No utilices un servo para mover o mantener cargas que excedan su capacidad de torque especificada. Esto puede causar un esfuerzo excesivo en el motor, lo que genera sobrecalentamiento y posibles daños.

2. **Proporcionar suficiente corriente**:

    - Los servos pueden consumir grandes cantidades de corriente al moverse bajo carga. Asegúrate de que la fuente de alimentación sea capaz de suministrar

la corriente necesaria. El uso de fuentes inadecuadas puede llevar a un rendimiento deficiente o al sobrecalentamiento del servo.

3. **Evitar el bloqueo del servo**:
    - No intentes forzar el eje del servo a moverse más allá de su rango permitido (normalmente 0-180 grados). Esto puede causar que el motor luche contra el límite y se sobrecaliente.
    - Asegúrese que el movimiento del servo motor se encuentre libre de obstáculos que puedan impedir el movimiento del eje principal, si el servo motor es incapaz de llegar a la posición a la que desea llegar se sobrecalentará y puede llegar a causar daños.
4. **Usar señales PWM estables**:
    - El control del servo depende de señales PWM precisas y estables. Señales ruidosas o incorrectas pueden causar movimientos bruscos o erráticos, lo que afecta su durabilidad.
5. **Mantenerlo alejado del calor**:
    - Los servos pueden generar calor durante su operación, especialmente bajo cargas pesadas. Evita colocarlos en lugares donde no puedan disipar el calor correctamente.
6. **Lubricación y mantenimiento**:
    - Si el servo se usa frecuentemente bajo carga, puede ser útil verificar si hay desgaste o necesidad de lubricar partes móviles para un funcionamiento suave.

Importante recalcar que se deben de seguir las pautas precautorias descritas con anterioridad para prolongar la duración de los servo motores y evitar dañar alguno.

4. HARDWARE

Recordemos que cuando hablamos de hardware se refiere a todo componente físico que cumplen una función computacional o mecánica en un sistema, en este caso el robot, aunque también se cubrirá el material que se usó para darle la forma que tiene actualmente al momento de redactar este manual.

4.1. Estructura de la Apariencia Física

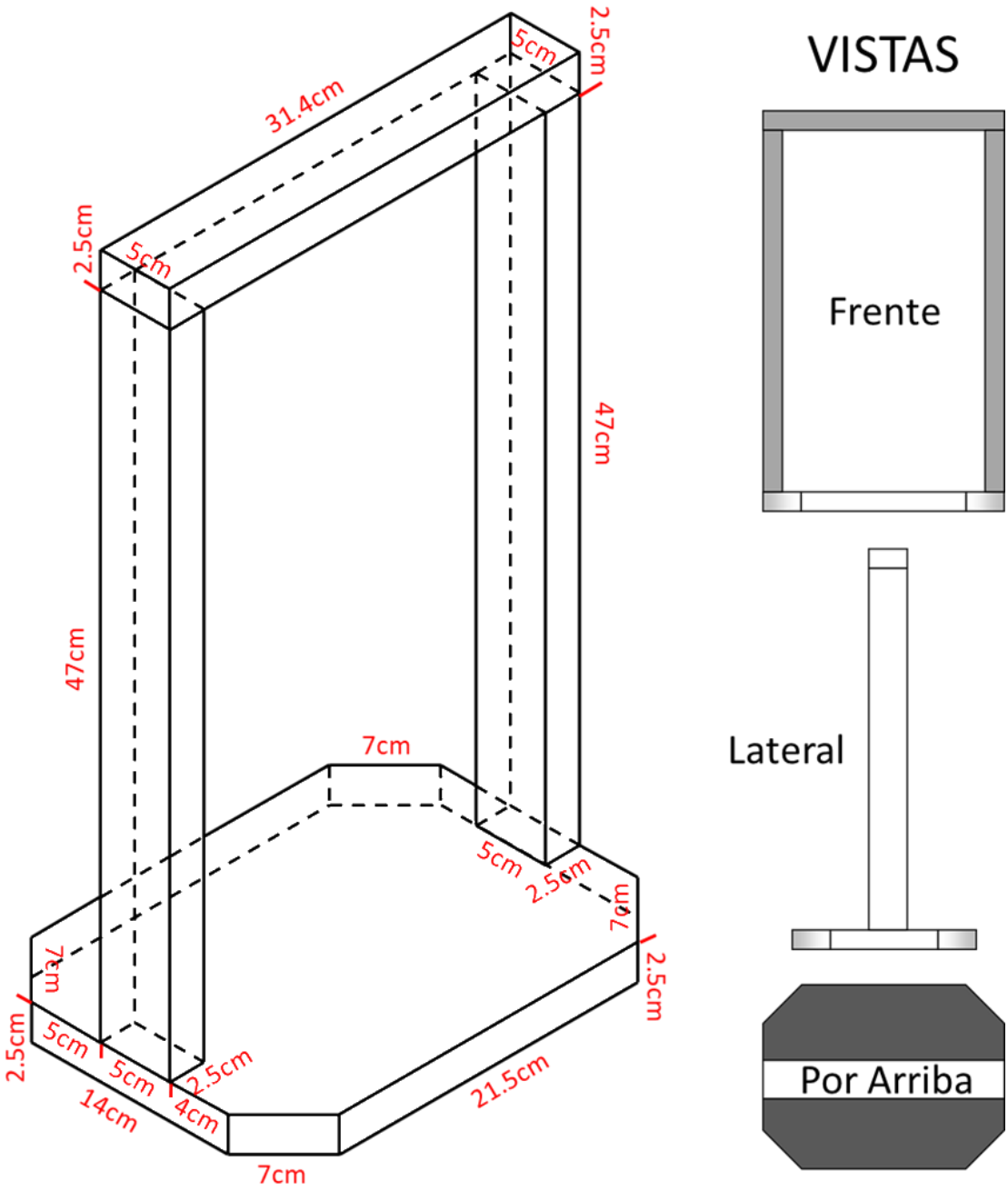
Principalmente el robot esta constituido de dos partes, la principal que es el cuerpo donde tiene las manos y donde estará todo el circuito y montaje del robot, y la otra que serían "los pies", pero como no se tiene planteado que se mueva realmente, pues en su lugar es un banco para sentarse giratorio como el que se muestra a continuación:

Este banco realmente no fue adquirido, si no obtenido del basurero, "la basura de unos es el tesoro de otros", una buena limpiada, lijada y pintada lo dejo en un estado no perfecto o nuevo pero decente.

No presenta tantos aspectos de oxidación y parece que si resistirá por unos años más, no tiene la silla suave para que uno pueda sentarse cómodamente, pero eso no es necesario, de igual forma la palanca para ajustar el nivel de altura no la tiene, por lo que probablemente no es posible ajustar la altura, aunque quedo a la altura perfecta para el robot, el espacio excedente que dejo la madera que forma el asiento del banco giratorio fue recortado para

que encajará la forma de la base del cuerpo del robot, más que nada para darle una forma estética y de persona como tal.

Justo sobre este banco improvisado descansa el cuerpo del robot, que consta de una estructura de madera simple y sencilla descrita en el siguiente dibujo:



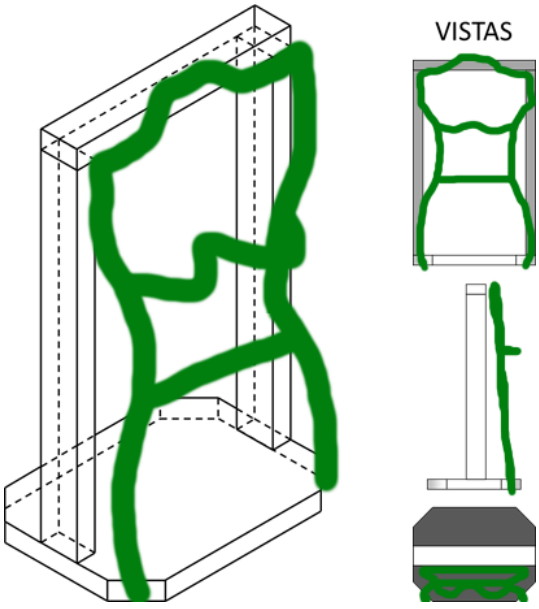
Esta estructura hecha de madera fue elaborada por un contribuyente al proyecto cuyo nombre es Gabriel Rendón Luna, es un poco pesada de transportar, pero queda

perfectamente al cuerpo del robot, una camisa del tecnológico le queda perfectamente (un poco apretada/forzada y con dificultades, pero le queda).

Dentro de esta estructura de madera se incrusto por enfrente un gancho de ropa diseñado para colocar ropa de mujer como el que se muestra a continuación:

Se pudo haber hecho un robot masculino, pero para potenciar lo que es la inclusión de genero se optó por hacer un diseño femenino, de ahí la necesidad de usar este gancho, donado para el proyecto por parte del profesor Joselito Chue Morales.

Subsecuentemente este gancho hecho de material metálico y recubierto por plástico fue cortado y ajustado para encajar en la estructura de madera de la siguiente forma:



Fue recubierto por una espuma de esponja color amarillo para darle la textura, por la parte de atrás se uso un fomi negro largo que fue usado anteriormente para un teclado gigante de colores, pero ahora queda como una capa para cubrir la parte que vendría siendo “la espalda” del robot.

También tiene unos tubos de metal en donde están las 2 maderas de las orillas de la estructura y una placa metálica que une los tubos por la parte central en altura que no se usan en lo absoluto, pero sirve para darle forma a la espalda, aunque no son realmente necesarios.

Eso sería todo lo que consiste el robot en cuanto a la estructura que le da apariencia femenina, dentro de este espacio se ubican todos los dispositivos que darían vida al robot como tal, usando la capa como una cortina que se levanta se puede acceder al circuito directamente para cualquier acción correspondiente.

En caso de requerir la estructura de madera por cualquier razón, quizás sea una buena idea **consultar a un arquitecto para que se encargue de crearla o diseñar uno diferente** según sus necesidades.

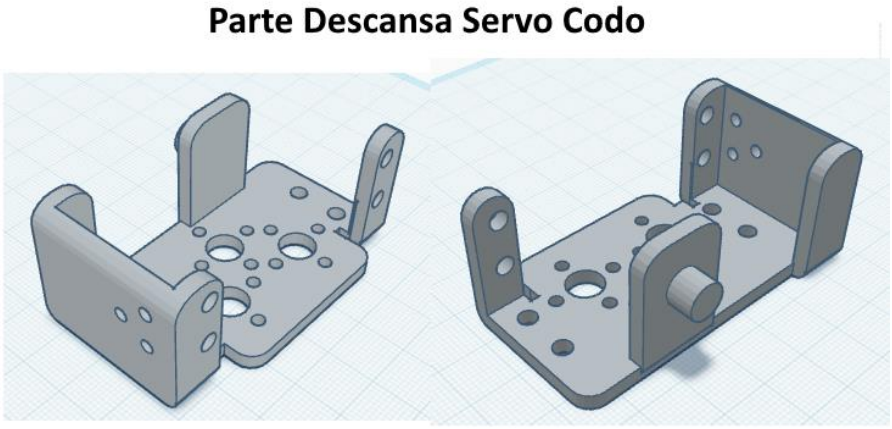
4.2. Brazos del Robot

Es necesario separar la estructura estática de la movible por motivos de ciertas especificaciones a recalcar en cuanto la forma en la que operan los brazos del robot, en sí el

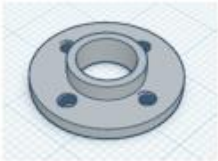
cuerpo y toda la estructura anteriormente mencionada no influyen en cuanto al comportamiento del robot mismo, es solo un soporte físico para contener y darle forma al robot.

A diferencia del cuerpo, los brazos fueron hechos con los servo motores y piezas impresas en una impresora 3D, es decir, piezas hechas de plástico, diseñadas específicamente para tener la rigidez suficiente para levantar los brazos sin que se doblen, tuerzan o rompan de alguna forma... excepto las partes donde descansan los servo motores, esas piezas quizás les haga falta un rediseño, por lo general no sufren daños si no chocan con nada o realizan movimientos bruscos, reforzarlos es recomendable para evitar roturas sin embargo.

Cada pieza será detallada a continuación en los siguientes diagramas hechos en tinkercad:

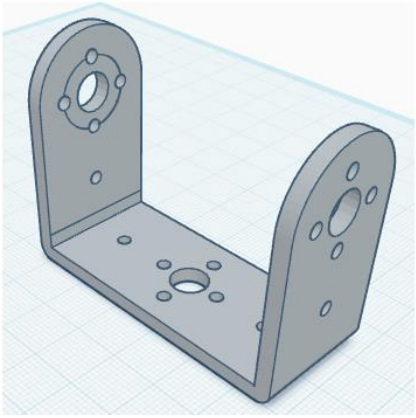


Parte Soporte Servo Codo

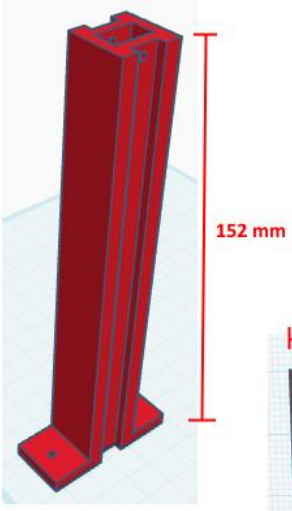


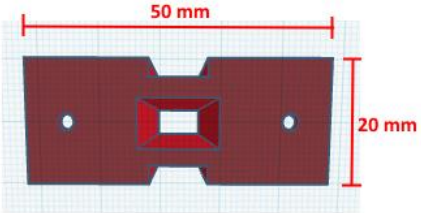
Pieza Acoplamiento Eje Servo

Parte Movible Codo

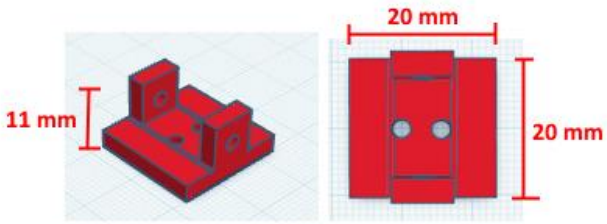


Partes Brazo Codo-Mano

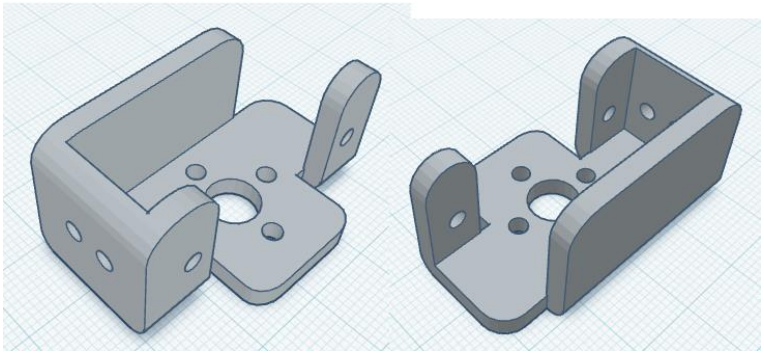




Parte Ensamblaje Brazo Codo-Mano



Pieza Descansa Servo Mano



La mayoría de los servo motores ya vienen con la pieza “Acoplamiento Eje Servo” incluida con estos mismo, por lo que no será necesario imprimir dicha pieza en esos casos.

Note que solo se dieron medidas en milímetros para las partes “Brazo X-Y”, más que nada para las longitudes de los brazos, las demás piezas no se dieron medidas ya que puede variar dependiendo los servo motores que se utilicen, por lo que es recomendable hacer los diseños considerando los servo motores que se tengan o usarlos como base para crear su propio diseño.

Quizás se están preguntando, ¿Dónde está la parte donde el servo motor del hombro descansa y la que conecta el cuerpo con dicho brazo? Muy sencillo, no existe, ya que los servo motores que se usan para el hombro vienen ya con piezas metálicas que se pueden atornillar a la madera, observe una imagen en cuestión de las piezas junto a su servo motor:

Si observa con atención de hecho, notará que la pieza “Brazo Hombro-Codo” que es la que conecta el hombro a la parte donde descansa el servo del codo tiene en su base los hoyos necesarios para atornillar la pieza con los hoyos que tiene la pieza metálica más ancha del servo motor.

La otra parte metálica que es la menos ancha de las dos, se atornilla directamente a la madera en la altura que deberían estar los brazos posicionados.

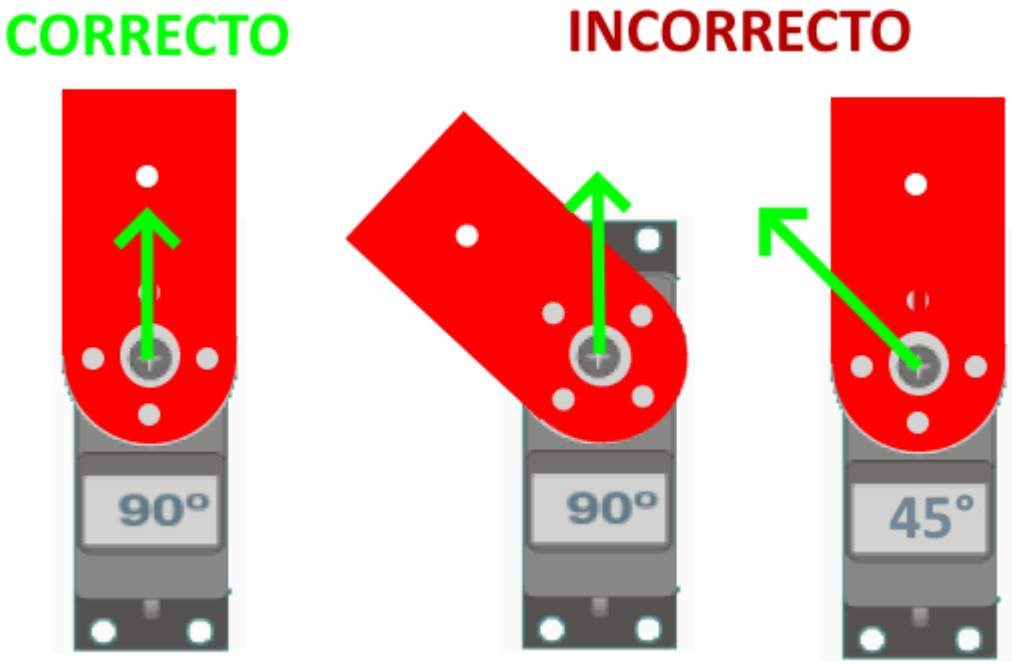
De esta forma se construyen los brazos que constituyen al robot, ensamblándolos apropiadamente uno tras otro en los hoyos correspondientes, estas piezas solo tienen una forma de ensamblarse, con excepción de las que se extienden alargadamente entre las piezas que conectan el hombro-codo y el codo-mano.

Dado lo anteriormente mencionado, es importante mencionar que de preferencia el brazo debe tener el servo motor ubicado en el codo “mirando” hacia atrás, en dirección a la espalda del robot, y el servo motor ubicado en la mano “mirando” hacia enfrente, en dirección al pecho del robot.

Todo esto para evitar recodificar las direcciones de rotación de los servo motores, pues **no es lo mismo tener un servo motor girando en sentido horario cuando esta “mirando” hacia enfrente que cuando esta “mirando” a espaldas**, antes de ensamblar el brazo, es

importante tener esto en mente para evitar el reensamblaje del brazo o la recodificación de las direcciones de rotación.

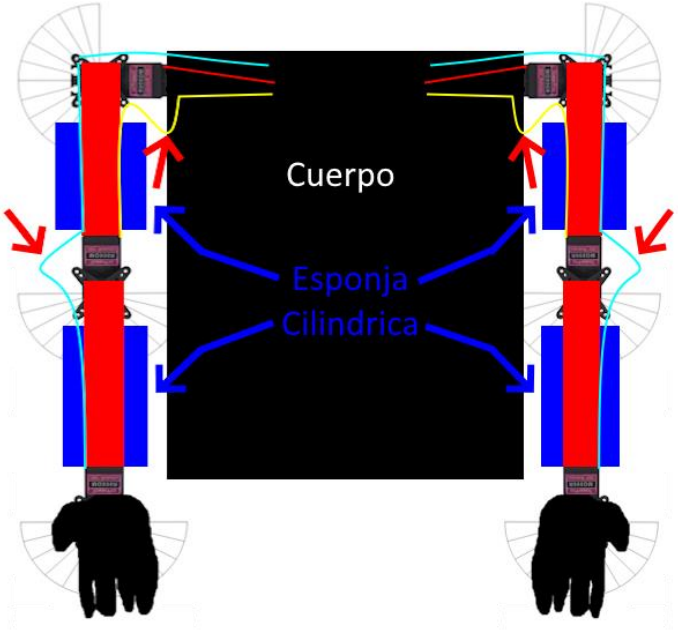
Adicionalmente, al momento de ensamblar los “ganchos” o pieza movible a cada servo motor, **asegúrese de que este orientado a 90° o conozca la orientación actual del eje del servo motor y ensamble paralelamente a esa dirección la pieza en cuestión**, de tal forma que quede de la siguiente forma:



Esto es importante, pues los servos solo tienen un rango de rotación de 0 a 180 grados, **si la pieza no esta orientada en la misma dirección de rotación que el eje del servo motor, puede haber problemas** donde choque con la estructura de madera o la cabeza que consta de un monitor, lo cual podría causar que el servo motor se dañe o dañe otros componentes **(consulte el punto 3 de las pautas de uso correcto de servo motores en la sección 3.8 del manual)**.

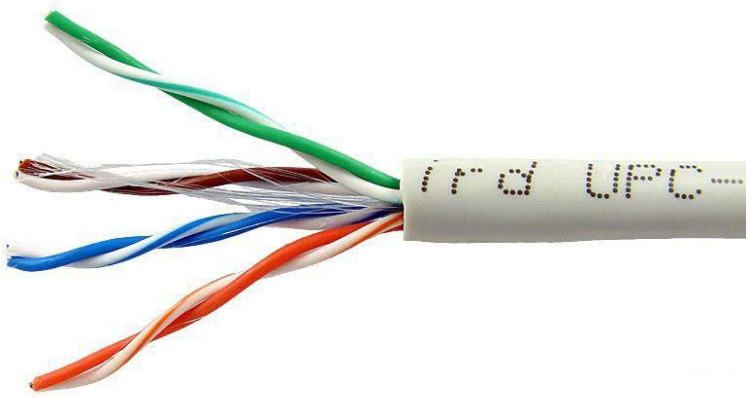
El cable de cada servo motor es pasado entre medio de las piezas que conectan los hombros, codos y manos para que puedan llegar al centro del cuerpo donde todo el circuito se encuentra alojado, por esto mismo es que tienen la ranura entre medio de su forma, por supuesto el cable mismo del propio servo motor no alcanza a llegar todo el camino hacia el centro del cuerpo, por lo que extender el alcance con cable adicional es necesario,

adicionalmente, **asegúrense de dejar cable de sobre colgando en el codo y hombro**, pues el robot moverá dichas partes y **si colocan el cable muy justo en esas partes, terminará rompiéndolo o será incapaz de mover el brazo a una posición y terminará dañando los servo motores**.



De preferencia no usen cable puente que se usa para circuitos de Arduino, hacer conexiones puente entre cables macho-hembra o macho-macho y hembra-hembra tiende a causar ruido en las señales y tiende a no conectar bien, utilicen otro medio para hacer la conexión, usualmente tendrán que cortar el cable del servo motor para remover esas entradas a los cables puente y realizar la conexión con los filamentos de cobre internos en el cable del mismo, experiencia en esto es necesario para no terminar inutilizando los servo motores, es recomendable usar estaño y cinta de aislar para conectar los cables para asegurar que la conexión sea lo más confiable posible.

Para la elaboración del robot al momento de redactar este reporte se uso cable de red conocido como UTP, aunque es usado para comunicación en redes entre computadoras, también puede ser usado para transmitir señales y corriente de casi cualquier tipo en cada uno de los cables internos.



Con todo eso cubierto acerca de aspectos importantes de los brazos del robot y su correcto ensamblaje, note que en anterior diagrama se hace mención de esponja cilíndrica para cubrir los brazos, más que nada son para dar volumen a los brazos, véalo como que las partes de plástico son “los huesos”, ya que son rígidos y la base principal de movimiento, la esponja cilíndrica entonces recubre ese “hueso” para darle volumen y se vea más como un brazo cuando este dentro de una camisa, como si fuera “la piel” o “el músculo” del brazo.

Para fácil ensamblaje, esas esponjas se abren cortándolas por un lado completamente para que se puedan abrir y meter sobre los brazos, manteniendo los cables y la pieza de plástico dentro y para mantenerlo cerrado, se usa cinta; así es como se constituyen los brazos del robot y los aspectos técnicos que se tienen que tener en cuenta al momento de armarlos.

4.3. Dispositivos Tecnológicos y sus Conexiones

Para que el robot pueda operar y gestionar la inteligencia artificial que se tiene planeada implementar, se consiguió de una **Raspberry Pi 4 Modelo B**, la cual tiene las capacidades computacionales mínimas requeridas para ejecutar un sistema operativo y, por ende, ser capaz de ejecutar el programa con la inteligencia artificial que genera respuestas, anima una interfaz gráfica simple y moverá los servo motores implementados en los brazos.

Es necesario que se cuente con un ventilador miniatura adecuado conectado en los pines de alimentación de la placa para enfriarla de su calentamiento, de lo contrario se puede llegar a calentar y dañar, el sistema se apaga para evitar esto, pero no se puede confiar todo el tiempo en este.

Para visualizar la interfaz gráfica y operar la Raspberry Pi 4 se utiliza un cable conversor de micro HDMI a HDMI, otro adaptador de HDMI a VGA y un cable VGA que conecta hasta el monitor, el monitor es uno marca ViewSonic con resolución 1440 x 900 de 19’’, adecuado para que las manos no colisiones con pantallas anchas.

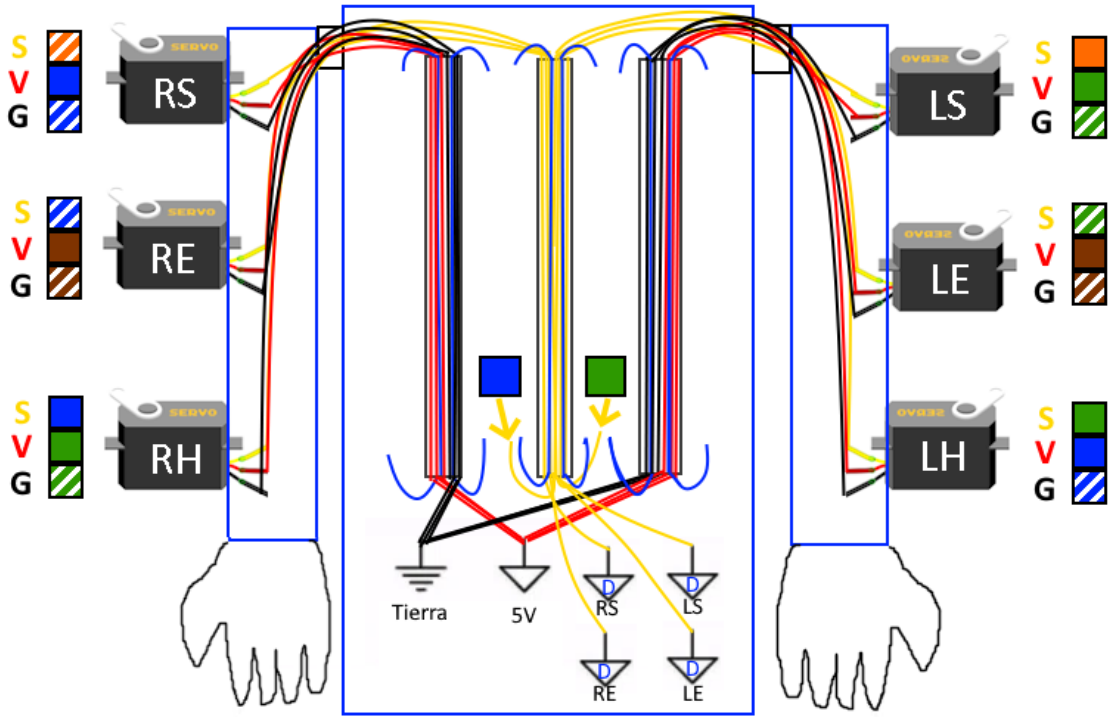
También se consiguieron **6 servo motores de diferentes capacidades** para levantar las estructuras que conforman los brazos del robot las cuales se especifican a continuación:

- 2 servo motores DSSERVO (25 kg-cm de torque)
- 2 servo motores MG995 TowerPro (10 kg-cm de torque)
- 2 mini servo MG90S (2 kg-cm de torque)

Para el cableado de los servo motores, como se mencionó en la sección anterior se utiliza cable de red UTP, cada uno de los cables internos del cable sirven para llevar una conexión para alimentar los motores tanto por señal como por alimentación, en total se usan de 3 cables UTP debido a que cada uno contiene solo 8 cables y se requieren 18 conexiones.

De los 24 cables que se obtienen de los 3 cables UTP, 6 cables salen sobrando para futuras conexiones a más componentes y mejoras o remplazo por daño del cable, mientras tanto, se encuentran aislados correctamente para evitar causar un corto o interferencia de algún tipo.

El color de cada uno de los cables facilita identificar cual cable de señal corresponde a cada servo motor, lo cual facilita su conexión, el diagrama a continuación determina como los 3 cables UTP están conectados según su color:



Note como **2 cables de uno de los cables UTP que maneja las señales PWM de los servos no conectan a ningún pin a pesar de estar conectados al servo (los cables amarillos que sobran en el cable UTP del centro)** y aun cuando estos servos están siendo alimentados, pues estos no están operando, solo los servos de los hombros y los codos.

Estos servo motores no se implementaron debido a problemas con las manos anteriormente que no podían girar, puede que se requiera un servo de torque mayor, que los servo motores en su momento no funcionaban o no recibían buena alimentación, de igual forma, su funcionamiento no fue requerido, pueden ser reconectados para hacer mover las manos, pero tienen que fijarlos a las manos mismas para que el movimiento se haga.

Adicionalmente, **en cada cable UTP sobra 2 cables (aparte de los 2 sobrantes del UTP central de señales PWM) ya que de cada uno solo se usan 6 cables (los cables azules de cada cable UTP en el diagrama)**, quedan de sobra como repuesto en caso de que el cable salga dañado o como sobrante para agregar más componentes electrónicos para mejoras o modificaciones.

Todas las señales PWM son conectadas a la Raspberry Pi 4 en los pines correspondientes donde el programa manda las señales para dicho movimiento, cabe destacar que **bajo**

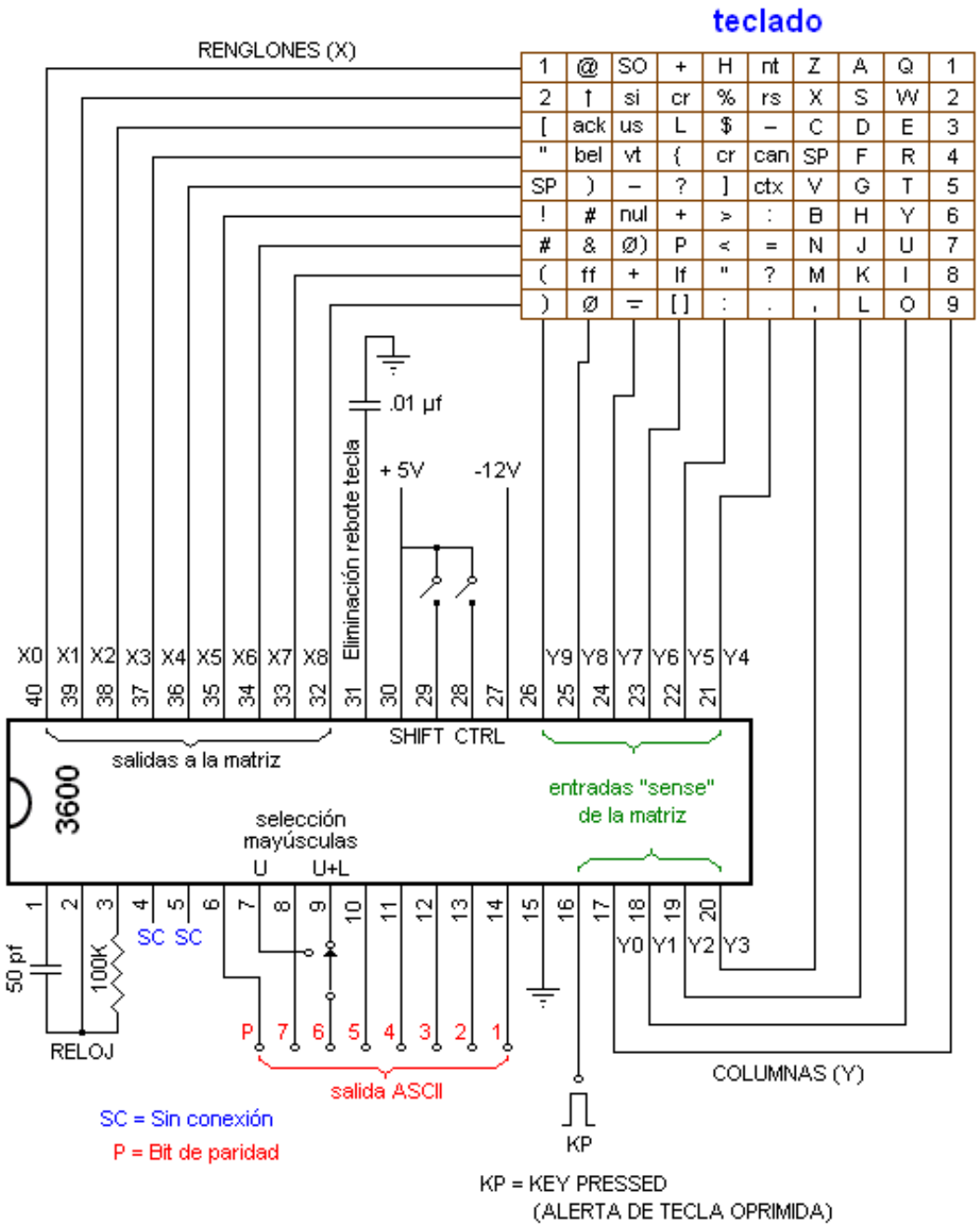
ninguna circunstancia deben alimentar los servo motores con los pines de la Raspberry Pi 4, de lo contrario podrían dañar la Raspberry Pi 4 o sus pines debido al consumo de amperaje que implican los servo motores, una fuente de alimentación externa es necesaria, **la tierra es la única que se deberá conectar a la Raspberry Pi 4 desde la fuente de alimentación externa para marcar el valor de referencia para diferencias de voltaje en todo el circuito para la comunicación**, comunicación con los servo motores no podrá establecerse apropiadamente si no se hace esto.

Para el sistema de audio, se utiliza de un set de bocinas de escritorio de los antiguos, como los que se observan a continuación:

No es exactamente ese modelo de la imagen, pero se trata de un modelo viejo donde se tenía que aun enchufar a un receptor de clavija para alimentarlos, no se utiliza USB directamente, se utiliza un plug el cual se conecta directamente a la Raspberry Pi 4.

También se usa de un teclado de entrada USB, pero no de la forma que uno pensaría, observe la imagen siguiente:

Se trata del chip del teclado, el cual se encarga de mandar señales por sus pines a una lámina no conductora con líneas de cobre la cual toca con otra lamina con cobre con diferente mapeado, donde al presionar una tecla, hacen contacto ciertos puntos de esa lamina para cerrar el circuito de una forma y mandar así la información de la tecla correspondiente a esta, un sistema que consta de aproximadamente 100 combinaciones diferentes con solo 20 pines aproximadamente formando una matriz de combinaciones:



Este tipo de técnica es abusado para realizar una sola conexión entre 2 pines del chip del teclado que corresponde a la letra R en el teclado, esos pines son conectados a través de un botón grande el cual se implementa al robot externamente para que pueda recibir la pulsación de este en el programa.

Todo esto para que, al momento de presionar el botón, los usuarios puedan empezar una rutina para que pueda recibir una entrada por voz del usuario y al presionarlo devuelta empiece a procesar la información, además, con solo un botón se puede hacer mucho más, como varias otras funciones dependiendo si se mantiene presionado un tiempo determinado y se suelta o si se pulsa sucesivamente rápidamente X número de veces, todo esto lo manejaría el software como si se tratará de la tecla R.

Y hablando de usuarios hablando al robot, se utiliza de un micrófono de entrada USB para recibir el audio, el cual el sistema operativo y el programa tienen acceso para recibir el audio del usuario y procesarla para generar una respuesta con el robot, en general cualquier micrófono que sea compatible y pueda ser detectado por el sistema Linux (sistema que manejan las Raspberry Pi) funciona para esto, siempre y cuando sea el único micrófono conectado a la placa:

Esta parte es muy fácilmente remplazable y removible por ser conexión USB, por lo que puede ser cambiado por un micrófono nuevo o diferente USB para mejorar o remplazar uno dañado.

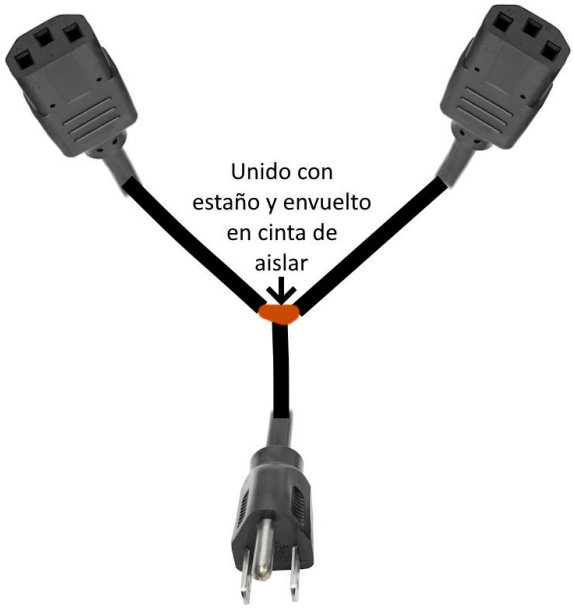
Con todo eso cubierto, un componente que no se puede olvidar mencionar y que forma parte esencial de alimentación del robot es la fuente de poder que se utiliza, que fue conseguida de una laptop como tal, se tratá de una caja metálica con varios cables de salida que mandan diferentes niveles de corriente de 3.3V, 5V y 12V además de la tierra, estos vienen en varias partes y tiene el amperaje más que suficiente para alimentar los servomotores y la placa Raspberry Pi 4.

La imagen es para que se den una idea de cual es la fuente de poder, más no representa la que se utiliza en el robot, tiene más cables la que se utiliza.

La fuente de poder es la que alimenta todos los servo motores y les da la tierra adecuada a todos, además de que en una línea separada se alimenta la Raspberry Pi 4 cortando y adaptando un cable de entrada tipo C a este, una línea de tierra es sacada y conectada a un pin de tierra de la Raspberry Pi 4 para hacer funcionar correctamente los servo motores cuando la señal está conectada a esta.

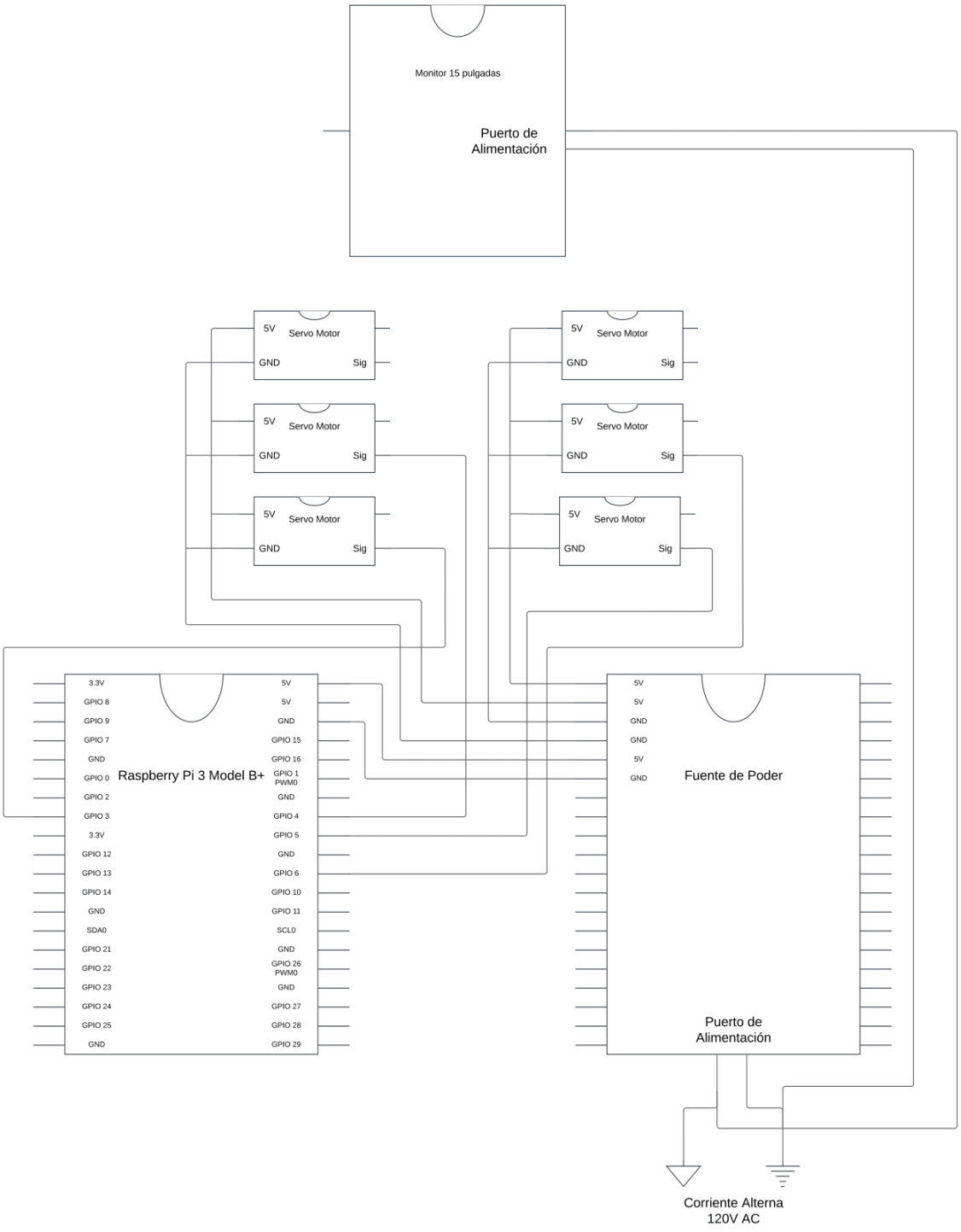
Tanto la fuente de poder como el monitor y las bocinas requieren conectarse a una clavija de corriente alterna para funcionar, sin embargo, no siempre se tendrá a disposición 3 clavijas donde enchufarlos, por lo que para simplificar la cantidad de clavijas necesarias, se cortaron los cables y unieron adecuadamente para formar un solo cable que conecta a la corriente directa de una clavija.

Observe un ejemplo con solo 2 entradas y 1 salida la cual se conecta para alimentar dos dispositivos con solo una clavija, eso se realizó para 3 dispositivos:



A continuación, observe un diagrama de conexiones completo de todo lo que constituye la alimentación del robot, tanto servo motores, fuente de poder, monitor *(denotado como 15'' pero es de 19'')* y Raspberry Pi 4 *(representado como Raspberry Pi 3 Modelo B+ por ser la placa anterior que se usó para el robot, pero las conexiones son las mismas igualmente)* en la siguiente página del manual.

El microfono, teclado y bocinas fueron ignorados en dicho diagrama debido a su simple conexión al robot, ya que el microfono y el teclado requieren una entrada USB nada mas en la Raspberry Pi 4, por lo que es sencillo, la bocina se conecta al plug de salida de audio de la Raspberry Pi 4 y se alimenta con el cable creado de 3 entradas y 1 entrada.



Si el manual fue impreso es probable que no se pueda apreciar las conexiones en su totalidad, por lo que es recomendable verlo en digital y hacer zoom dentro del documento Word o PDF.

5. SOFTWARE

El software del programa se separa en dos partes, una siendo el sistema operativo Linux de nombre Raspbian el cual se debe de configurar con los paquetes necesarios para que el programa escrito en Python funcione sin problema alguno, el cual es la segunda parte del software como tal.

5.1. Configuración del Sistema Operativo Raspbian

Para empezar, cabe mencionar que **no es necesario descargar e instalar Python en el sistema operativo ya que el programa viene con un entorno virtual de Python**, construido adecuadamente para correr el programa, **en caso de ser necesario instalar nuevos paquetes al entorno virtual de Python utilizar el pip ubicado en la carpeta del entorno virtual**, la carpeta bin y dentro con la terminal, usar los comandos python o pip para instalarlos.

En caso de ser necesario remplazar o recrear el entorno virtual de Python, vean la sección 5.3 del manual.

Lo único que se debe de instalar son los controladores de audio para que Python pueda acceder a los dispositivos de salida de audio como las bocinas y a los dispositivos de entrada de audio para el micrófono, para ello solo se corren los siguientes 4 comandos:

- `sudo apt update`
- `sudo apt upgrade`
- `sudo apt-get install portaudio19-dev`
- `sudo apt install python3-pyaudio`

Con todo esto preparado solo queda copiar el programa junto a sus archivos adicionales a una ubicación cual sea que pueda ser accedido desde la terminal y correr los siguientes dos comandos cada vez que deba ejecutar el programa con el entorno virtual de Python, usualmente son:

- **`sudo pigpiod`**
- `<ruta-entorno-virtual>/bin/python <ruta-programa>/main.py`

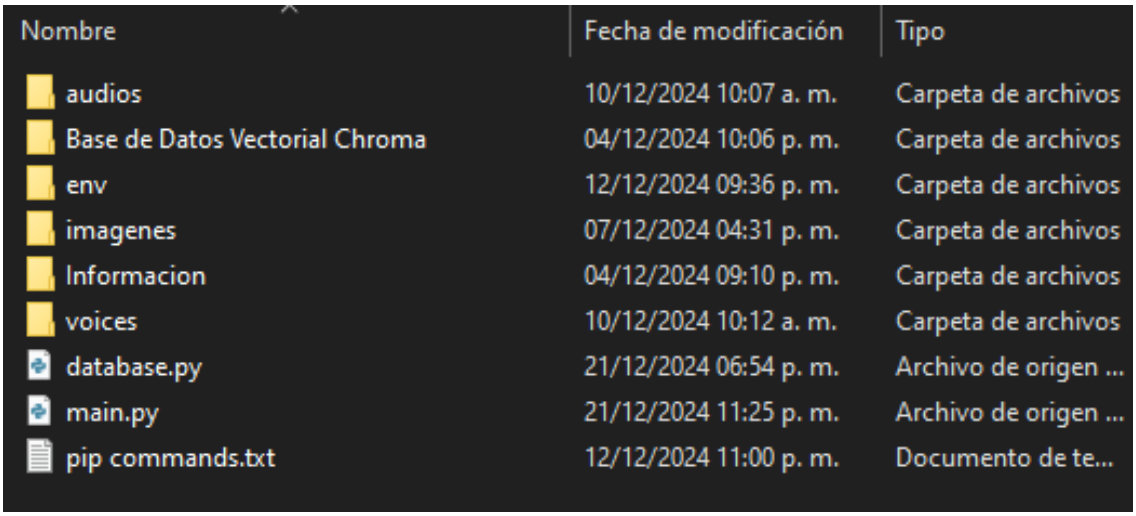
Donde <ruta-entorno-virtual> es la carpeta donde esta el entorno virtual de Python y <ruta-programa> es la carpeta en la que se encuentra el programa ubicado con sus contenidos.

El primer comando es para activar los pines de la Raspberry Pi 4 en donde corre el programa para poder mover los servo motores, es de suma importancia correr ese comando antes que el programa.

Se puede crear un script .sh para automatizar esos dos comandos y solo ejecutarlo dando doble clic.

5.2. Estructura del Programa

El programa esta conformado de unas carpetas que tienen archivos adicionales y directorios esenciales para su correcto funcionamiento, dicho directorio se encuentra de la siguiente forma:



El programa se encuentra separado en 4 partes diferentes, de las cuales 3 son el programa principal que conforman el correcto funcionamiento del robot, cada una gobieran un sistema diferente y todos estos se encuentran dentro de un solo script de Python utilizando multiprocesamiento para su ejecución a excepción de un sistema que es un script por separado.

La falla de cualquiera de los sistemas principales afectará a todo el programa entero en su mayoría, hay excepciones donde el programa aun puede funcionar aun cuando uno de los

sistemas ha fallado, para mejor funcionamiento del robot es esencial que ambos estén operacionales.

El programa principal consta de un sistema de control de servo motores, un sistema de interfaz gráfica para mostrar en la pantalla y entrada de teclado del usuario y un sistema de manejo de la IA para procesar toda la información ya sea del usuario o de la IA, el sistema donde toda la lógica se realiza en pocas palabras.

El otro sistema que se encuentra en un script separado del programa principal es el sistema de creación de la base de datos vectorial, encargado de crear la base de datos para que el programa principal tenga acceso a la información especial del Instituto Tecnológico de la Costa Grande, toda esta escrita de forma manual en archivos .txt los cuales también van como parte de los archivos del programa del robot.

5.2.1. Sistema de Creación de la Base de Datos Vectorial

Este es un script independiente que consiste en crear la base de datos vectorial de la cual la IA se enriquecerá para dar información concreta sobre cualquier petición que se pide del usuario.

El script se encarga de cargar y leer todos los datos que hay en una carpeta de información donde se encuentran varios archivos .txt, los cuales lee y parte en partes de 800 caracteres, cada parte superpone 80 caracteres de la pieza anterior, para asegurar la continuidad del contexto por donde se haya cortado, después se les asigna una ID para ser almacenado en la base de datos vectorial que tiene la siguiente forma: `<nombre del archivo>:chunk <número de chunk>`, de esta forma se puede ver qué información y parte de esta la IA esta consultado al momento de hacer una búsqueda en el programa principal.

Una vez obtenido los chunks y generado los IDs de cada chunk de información, se meten a la base de datos vectorial, sin embargo, **solo agregará aquellos cuya ID no se encuentre en la base de datos**, es decir, si dentro de la base de datos ya existe uno con ID `General.txt:chunk 1`, entonces no se volverá a agregar ese chunk en específico cada vez que se lean los documentos por si se agregaron más, evitando repetir la información así, **el programa no es capaz de detectar sí la parte en la base de datos es la misma que la que se ha generado separando el documento actual**, por lo que si se modifican los

documentos, correr el programa de nuevo no hará que los actualice, borrar la base de datos y generarla de nuevo será necesario o bien modificar el programa para eso.

Al momento de agregar los datos a la base de datos se usa de una IA para generar el vector posicional para cada nuevo chunk de datos que se agrega a la base de datos, esta IA provienen del sitio huggingface y puede ser cambiada por otra modificando la variable del modelo, **cabe destacar que el modelo IA para generar vectores posicionales al almacenar los datos en la base de datos debe ser la misma en el programa principal ya que cada IA maneja una longitud vectorial diferente y significado diferente en cada número de este vector**.

El script contiene una función adicional para borrar la base de datos, el cual consiste en borrar la carpeta donde se aloja la base de datos, se puede borrar manualmente en el sistema operativo para obtener el mismo resultado y recrear la base de datos, **por defecto cada vez que se ejecuta el script se borra la base de datos y se recrea con la información actual en la carpeta de Información**.

El código que se encarga de hacer eso es el siguiente, usualmente viene bajo el nombre database.py:

```
import os
import shutil
from langchain_chroma import Chroma
from langchain_community.embeddings import HuggingFaceInferenceAPIEmbeddings
from langchain_community.document_loaders import DirectoryLoader, TextLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_core.documents.base import Document

DIRECTORIO = os.path.dirname(os.path.realpath(__file__))
EMBEDDING_MODEL = "intfloat/multilingual-e5-large-instruct"
CHROMA_PATH = DIRECTORIO + "/Base de Datos Vectorial Chroma"
DATA_PATH = DIRECTORIO + "/Informacion"
HF_KEY = "<YOUR HUGGING FACE KEY>"

embedding_function = HuggingFaceInferenceAPIEmbeddings(
    model_name=EMBEDDING_MODEL,
    api_key=HF_KEY
)

def main():
```

```
    documents = load_documents()
    chunks = split_documents(documents)
    add_to_chroma(chunks)


def load_documents():
    document_loader = DirectoryLoader(DATA_PATH, glob="*.txt",
loader_cls=TextLoader, loader_kwargs={'autodetect_encoding': True})
    return document_loader.load()


def split_documents(documents: list[Document]):
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=800,
        chunk_overlap=80,
        length_function=len,
        is_separator_regex=False,
    )
    return text_splitter.split_documents(documents)


def add_to_chroma(chunks: list[Document]):
    # Load the existing database.
    db = Chroma(
        persist_directory=CHROMA_PATH,
        embedding_function=embedding_function
    )

    # Calculate Page IDs.
    chunks_with_ids = calculate_chunk_ids(chunks)

    # Add or Update the documents.
    existing_items = db.get(include=[])  # IDs are always included by
default
    existing_ids = set(existing_items["ids"])
    print(f"Number of existing documents in DB: {len(existing_ids)}")

    # Only add documents that don't exist in the DB.
    new_chunks = []
    for chunk in chunks_with_ids:
        if chunk.metadata["id"] not in existing_ids:
            new_chunks.append(chunk)

    if len(new_chunks):
        print(f"👉 Adding new documents: {len(new_chunks)}")
        new_chunk_ids = [chunk.metadata["id"] for chunk in new_chunks]
```

```
        db.add_documents(new_chunks, ids=new_chunk_ids)
    else:
        print("✅ No new documents to add")


def calculate_chunk_ids(chunks):
    # This will create IDs like "data/monopoly.pdf:6:2"
    # Page Source : Page Number : Chunk Index

    last_page_id = None
    current_chunk_index = 0

    for chunk in chunks:
        source = chunk.metadata.get("source")
        source = source[source.rfind('\\') + 1:]
        current_page_id = f"{source}"

        # If the page ID is the same as the last one, increment the index.
        if current_page_id == last_page_id:
            current_chunk_index += 1
        else:
            current_chunk_index = 0

        # Calculate the chunk ID.
        chunk_id = f"{current_page_id}:chunk {current_chunk_index}"
        last_page_id = current_page_id

        # Add it to the page meta-data.
        chunk.metadata["id"] = chunk_id

    return chunks


def clear_database():
    if os.path.exists(CHROMA_PATH):
        shutil.rmtree(CHROMA_PATH)


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

Dependiendo de la disponibilidad del modelo IA y conexión a internet, puede tardar el script en generar la base de datos, **el directorio donde se aloja la base de datos debe de estar creada de antemano**.

5.2.2. Sistema de Control de Servo Motores

Este sistema se ejecuta a la par con los otros dos sistemas principales, consiste en el manejo de servo motores a través de una referencia de variable, ya que la IA es la que genera los movimientos de los servo motores, por lo que es necesario usar una variable como referencia para pasar la información entre hilos, este sistema corre en un hilo del sistema de manejo de la IA.

La variable es un diccionario de 4 campos denominados “rs”, “re”, “ls” y “le”, denotando cada parte del brazo previamente descrito en este manual, esta variable es la que se encarga de rotar los servo motores en las posiciones correctas, pues es usada para el cálculo de las rotaciones actuales de los servos, es decir, no van directamente a la posición que contienen en las variables, van lentamente haciendo interpolado constante entre la posición actual en la que están los brazos y la posición a la que deberían de llegar, dando un efecto suavizado.

La función encargada en el código de hacer eso es la siguiente:

```
def control_servo_movement(servo_angles):
    try:
        import pigpio

        if (DEBUG_MODE):
            debug_timer = 0

        rs_pin = 22
        re_pin = 23
        ls_pin = 24
        le_pin = 25

        rs_angle_cur = 0
        re_angle_cur = 90
        ls_angle_cur = 0
        le_angle_cur = 90

        pi = pigpio.pi()

        pi.set_mode(rs_pin, pigpio.OUTPUT)
        pi.set_mode(re_pin, pigpio.OUTPUT)
        pi.set_mode(ls_pin, pigpio.OUTPUT)
        pi.set_mode(le_pin, pigpio.OUTPUT)
```

```
        pi.set_servo_pulsewidth(rs_pin, 2500)
        pi.set_servo_pulsewidth(re_pin, 1500)
        pi.set_servo_pulsewidth(ls_pin, 2500)
        pi.set_servo_pulsewidth(le_pin, 1500)

        if (DEBUG_MODE):
            print("Servo Loop Started")

        while True:
            rs_angle_cur = rs_angle_cur + (servo_angles["rs"] -
rs_angle_cur)/25
            re_angle_cur = re_angle_cur + (servo_angles["re"] -
re_angle_cur)/25
            ls_angle_cur = ls_angle_cur + (servo_angles["ls"] -
ls_angle_cur)/25
            le_angle_cur = le_angle_cur + (servo_angles["le"] -
le_angle_cur)/25

            if (DEBUG_MODE):
                if (debug_timer == 0):
                    print("Servo Right Shoulder Angle: " +
str(rs_angle_cur))
                    print("Servo Right Elbow Angle: " + str(re_angle_cur))
                    print("Servo Left Shoulder Angle: " + str(ls_angle_cur))
                    print("Servo Left Elbow Angle: " + str(le_angle_cur))

                    debug_timer = SERVO_ANGLES_DEBUG_TEXT_FREQUENCY*20
                else:
                    debug_timer -= 1

            pi.set_servo_pulsewidth(rs_pin, 500 + 2000*(180 -
rs_angle_cur)/180)
            pi.set_servo_pulsewidth(re_pin, 500 + 2000*(180 -
re_angle_cur)/180)
            pi.set_servo_pulsewidth(ls_pin, 500 + 2000*(180 -
ls_angle_cur)/180)
            pi.set_servo_pulsewidth(le_pin, 500 + 2000*(180 -
le_angle_cur)/180)

            time.sleep(0.05)
    except Exception as e:
        print("Servo Manager Errored: " + str(e))
        print("Servo Signal Deactivated, the servos won't move anymore")
```

Las señales mandadas por los pines de salida están dadas en valores entre 500 y 2500 que consiste en el rango de ángulo en el que giran los servo motores por PWM.

Los pines que se utilizan para los servo motores son del 22 al 25, los cuales corresponden a los siguientes según la documentación de la biblioteca pigpio:

Type 3 - Model A+, B+, Pi Zero, Pi Zero W, Pi2B, Pi3B, Pi4B

- 40 pin expansion header (J8).
- Hardware revision numbers of 16 or greater.
- User GPIO 2-27 (0 and 1 are reserved).

	GPIO	pin	pin	GPIO	
3V3	-	1	2	-	5V
SDA	2	3	4	-	5V
SCL	3	5	6	-	Ground
	4	7	8	14	TXD
Ground	-	9	10	15	RXD
ce1	17	11	12	18	ce0
	27	13	14		Ground
	22	15	16	23	
3V3	-	17	18	24	
MOSI	10	19	20		Ground
MISO	9	21	22	25	
SCLK	11	23	24	8	CE0
Ground	-	25	26	7	CE1
ID_SD	0	27	28	1	ID_SC
	5	29	30	-	Ground
	6	31	32	12	
	13	33	34	-	Ground
miso	19	35	36	16	ce2
	26	37	38	20	mosi
Ground	-	39	40	21	sclk

Pueden usarse otros pines para mandar señales a los servo motores “rs”, “re”, “ls” y “le”.

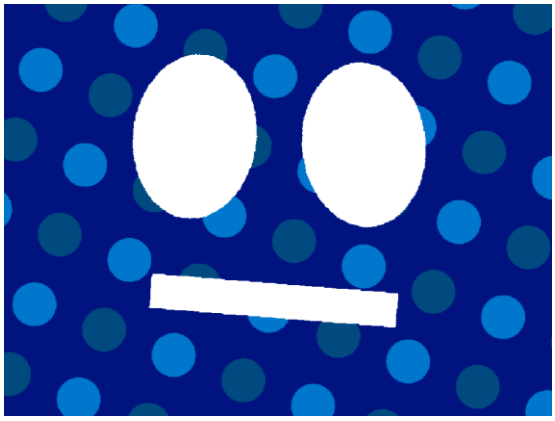
5.2.3. Sistema de Interfaz Gráfica y Entrada de Teclado

La parte de la interfaz gráfica consiste en una ventana en pantalla completa que muestra una cara la cual tiene la animación del habla para su interacción con el usuario, se utilizan de 2 variables compartidas para cambiar el estado e indicar cuando debe animarse para

hablar, esta interfaz fue hecha con pygame para recibir la entrada del teclado adicionalmente.

Hay 4 estados en la interfaz gráfica, una es donde esta la cara normal y las demás no son más que imágenes para indicar el proceso del robot, como Procesando que representa trabajo de la IA en generar respuesta y el audio correspondiente para "hablarlo", uno donde dice Te escucho, donde esta escuchando lo que dice el usuario, y el ultimo estado que dice Buscando, donde esta haciendo una búsqueda en la información de la base de datos vectorial para responder a la solicitud del usuario.

También dispone de un fondo simple animado, observe una imagen de la interfaz que constituye la cara y el respectivo código que lo compone en su totalidad con dicha funcionalidad:



```
if (__name__ == "__main__"):
    import multiprocessing, pygame, subprocess

    manager = multiprocessing.Manager()
    lock = multiprocessing.Lock()
    lock2 = multiprocessing.Lock()
    lock3 = multiprocessing.Lock()
    lock4 = multiprocessing.Lock()
    lock5 = multiprocessing.Lock()
    habla = manager.Value("d", 0)
    query = manager.Value("s", "")
    state = manager.Value("d", 2) #0 is face, 1 is listening, 2 is
processing, 3 is searching
    amount_audios = manager.Value("d", 0)
    key_pressed = manager.Value("d", 0)
    generated_audio_index = manager.Value("d", 0)
```

```
    reproduced_audio_index = manager.Value("d", 0)
    commands_queue_index = manager.list()

    proceso_ai = multiprocessing.Process(target=query_rag, daemon=True,
args=(habla, key_pressed, lock, lock2, lock3, lock4, lock5, query, state,
generated_audio_index, reproduced_audio_index, commands_queue_index,
amount_audios))
    proceso_ai.start()

    if (USE_TEXT_QUERY):
        hilo_input = threading.Thread(target=leer_input, daemon=True,
args=(query, lock2))
        hilo_input.start()

    pygame.init()

    screen = pygame.display.set_mode((WIDTH, HEIGHT), pygame.SCALED)
    pygame.display.set_caption("Asistente Luminia")
    pygame.mouse.set_visible(False)

    listening_image =
pygame.image.load(os.path.dirname(os.path.realpath(__file__)) +
"/imagenes/listening.png").convert_alpha()
    processing_image =
pygame.image.load(os.path.dirname(os.path.realpath(__file__)) +
"/imagenes/processing.png").convert_alpha()
    searching_image =
pygame.image.load(os.path.dirname(os.path.realpath(__file__)) +
"/imagenes/searching.png").convert_alpha()
    background_image =
pygame.image.load(os.path.dirname(os.path.realpath(__file__)) +
"/imagenes/bg.png").convert_alpha()
    bg_width, bg_height = background_image.get_size()

    # Reloj para controlar la velocidad de fotogramas
    clock = pygame.time.Clock()

    def draw_rotated_face(center_x, center_y, angle):
        face_surface = pygame.Surface((WIDTH, HEIGHT), pygame.SRCALPHA)

        draw_rotated_eye(face_surface, WIDTH // 2 + (face_leftX -
60*face_scale), HEIGHT // 2 - (face_leftY + 50*face_scale),
face_leftSizeX*face_scale, face_leftSizeY*face_scale, face_leftAngle)
```

```
        draw_rotated_eye(face_surface, WIDTH // 2 + (face_rightX +
60*face_scale), HEIGHT // 2 - (face_rightY + 50*face_scale),
face_rightSizeX*face_scale, face_rightSizeY*face_scale, face_rightAngle)
        draw_rotated_rect(face_surface, WIDTH // 2 + face_mouthX, HEIGHT //
2 - (face_mouthY - (60 + face_mouthSizeY/2)*face_scale),
face_mouthSizeX*face_scale, (10 + face_mouthSizeY)*face_scale,
face_mouthAngle)

        rotated_face = pygame.transform.rotate(face_surface, angle)
        face_rect = rotated_face.get_rect(center=(center_x, center_y))

        screen.blit(rotated_face, face_rect)

    # Función para dibujar y rotar ojos
    def draw_rotated_eye(surface, center_x, center_y, width, height, angle):
        # Crear superficie para el ojo
        eye_surface = pygame.Surface((width, height), pygame.SRCALPHA)
        pygame.draw.ellipse(eye_surface, FACE_COLOR, (0, 0, width, height))

        # Rotar superficie
        rotated_eye = pygame.transform.rotate(eye_surface, angle)
        eye_rect = rotated_eye.get_rect(center=(center_x, center_y))

        # Dibujar ojo rotado en la pantalla
        surface.blit(rotated_eye, eye_rect)

    def draw_rotated_rect(surface, center_x, center_y, width, height,
angle):
        # Crear superficie para el ojo
        rect_surface = pygame.Surface((width, height), pygame.SRCALPHA)
        pygame.draw.rect(rect_surface, FACE_COLOR, (0, 0, width, height))

        # Rotar superficie
        rotated_eye = pygame.transform.rotate(rect_surface, angle)
        eye_rect = rotated_eye.get_rect(center=(center_x, center_y))

        # Dibujar ojo rotado en la pantalla
        surface.blit(rotated_eye, eye_rect)

    def draw_background():
        bg_surface = pygame.Surface((WIDTH + 200, HEIGHT + 300),
pygame.SRCALPHA)

        for x in range(-bg_width, WIDTH + 200, bg_width):
            for y in range(-bg_height, HEIGHT + 300, bg_height):
```

```
                bg_surface.blit(background_image, (x + offset_x, y +
offset_y))

        rotated_bg = pygame.transform.rotate(bg_surface, -20)
        bg_rect = rotated_bg.get_rect(center=(420, 300))

        screen.blit(rotated_bg, bg_rect)

    timer = 0
    timer2 = 0
    offset_x = 0
    offset_y = 0

    running = True
    checked = False
    last_time = None
    last_time2 = None
    shutdown = True
    fullscreen = False
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                shutdown = False
                running = False

            if (event.type == pygame.KEYDOWN):
                lock4.acquire()
                key = key_pressed.value
                lock4.release()

                if (event.key == pygame.K_q):
                    shutdown = False
                    running = False
                elif (event.key == pygame.K_r):
                    if (key < 2):
                        checked = True

                        lock4.acquire()
                        key_pressed.value += 1
                        lock4.release()

                    if (last_time2 is not None and time.time() - last_time2
<= 1):
                        print("YES")
                        running = False
```

```
                    break

                if (last_time is None or time.time() - last_time > 1):
                    last_time = time.time()
                elif (time.time() - last_time <= 1):
                    last_time2 = time.time()
            elif (checked):
                checked = False
        else:
            pressed_time = None

            if (checked):
                checked = False

    if (not fullscreen):
        try:
            pygame.display.toggle_fullscreen()
            fullscreen = True
        except:
            pass

    offset_x = (offset_x + 3) % bg_width
    offset_y = (offset_y + 1.5) % bg_height

    draw_background()

    timer += 1.5

    lock.acquire()
    if (habla.value == 1 or timer2 > 0):
        timer2 += 1.5
        if (timer2%24 == 0):
            timer2 = 0
    lock.release()

    eye_width, eye_height = 60, 90
    face_scale = 2
    face_angle = 5*math.sin(math.radians(0.5*timer))
    face_leftX = 5*math.sin(math.radians(1.3*timer))
    face_leftY = 5*math.sin(math.radians(2*timer))
    face_leftAngle = 7*math.sin(math.radians(0.76*timer))
    face_leftSizeX = 90
    face_leftSizeY = 65 +
55*math.cos(math.radians(360*min(max(timer%800, 0), 15))/15)
```

```
        face_rightX = -5*math.sin(math.radians(1.1*timer))
        face_rightY = 5*math.sin(math.radians(1.76*timer))
        face_rightAngle = -7*math.sin(math.radians(0.93*timer))
        face_rightSizeX = 90
        face_rightSizeY = 65 +
55*math.cos(math.radians(360*min(max(timer%800, 0), 15))/15)
        face_mouthX = 3*math.sin(math.radians(0.73*timer))
        face_mouthY = -4*math.sin(math.radians(0.94*timer))
        face_mouthAngle = -5*math.sin(math.radians(1.2*timer))
        face_mouthSizeX = 180
        face_mouthSizeY = 30 - 15*math.cos(math.radians(15*timer2))

        lock3.acquire()
        if (state.value == 0):
            draw_rotated_face(WIDTH // 2, HEIGHT // 2, face_angle)
        elif (state.value == 1):
            timer = 0
            screen.blit(listening_image, (0, 0, WIDTH, HEIGHT))
        elif (state.value == 2):
            timer = 0
            screen.blit(processing_image, (0, 0, WIDTH, HEIGHT))
        else:
            timer = 0
            screen.blit(searching_image, (0, 0, WIDTH, HEIGHT))
        lock3.release()

        # Actualizar pantalla
        pygame.display.flip()

        # Controlar FPS
        clock.tick(60)

    # Cerrar Pygame
    pygame.quit()

    if (platform == "Linux" and shutdown):
        subprocess.call(["shutdown", "-h", "now"])
```

Note que el código hace recepción de la tecla R, una de las funcionalidades de las interfaces de pygame, la cual al pulsarse manda un cambio en la variable compartida key_pressed para hacer que se reciba el evento donde la IA empieza a escuchar audio del usuario, si la tecla o bien botón en este caso se presiona varias veces, esto hará que el programa se cierre

y apague el sistema si se trata de Linux, ya que la idea es que el robot empiece con este programa siempre.

Cabe mencionar además que **si se necesita modificar el programa se debe de pulsar la tecla Q en un teclado, esto hará que el programa se cierre y le de acceso al sistema operativo sin apagar la Raspberry Pi 4**, esta **al no estar presente en un botón alguno se deberá de conectar un teclado externo para poder realizar dicha función**.

Adicionalmente, ahí mismo en este código se empieza de hecho todo el programa, pues la interfaz no puede estar en un subproceso del principal, es necesario que la interfaz pygame sea el principal proceso, teniendo que usar multiprocesamiento para correr los demás sistemas, entre ellos el query_rag() que es el sistema de lógica de la IA, por ello es que se declaran todas las variables compartidas necesarias y los locks necesarios para asegurar que no se corrompan si se acceden al mismo tiempo las variables.

Debido a que este sistema empieza todo el programa, input() que es una función base de Python solo puede ser usado en el proceso principal, procesos hechos con multiprocessing no pueden usarlo, por lo que se tiene que hacer un hilo para recibir la entrada por teclado del usuario en caso de que se use el teclado para depurarlo, esta función se llama leer_input() y usa una variable compartida para pasar el resultado del usuario.

```
def leer_input(query, lock2):
    while True:
        leido = input("")

        lock2.acquire()
        if (leido != "" and query.value == ""):
            query.value = leido
        lock2.release()
```

5.2.4. Sistema de Manejo de la IA (Lógica del Programa)

Con todo lo anterior cubierto se habla del sistema que maneja toda la secuencia lógica de acciones, este sistema es ejecutado antes que la interfaz gráfica se manifieste.

Este sistema dispone de más pequeños subsistemas que se explicarán en detalle, a diferencia de los demás sistemas, estos funcionan en serie, es decir, el programa no

continua hasta que estos subsistemas terminen su secuencia a pesar de que algunos utilizan multiprocesamiento.

Primero los sistemas se inicializan, los modelos IA de respuesta y reconocimiento de audio son cargados para su uso en el programa, a través de este sistema el hilo del sistema de los servo motores es inicializado junto con su variable diccionario para la posición de los servo motores, se carga el modelo IA Embedding para las consultas a la base de datos vectorial, se realiza una prueba con este sencilla y se carga la misma base de datos vectorial igualmente para su posterior uso.

Se crean las bases para los chats de la IA de respuestas, donde se deja un mensaje de sistema que explícitamente indica que la IA es un asistente escolar universitario y que debe formatear las respuestas usando un formato especifico de comandos entre corchetes [] para ser ejecutados, así es como la IA indica que servo motores mover y cuando realizar una búsqueda en su base de datos.

Después de eso se carga el método de texto a voz el cual es diferente según el sistema operativo, donde Linux utiliza una biblioteca llamada Dimits, la cual utiliza el Piper-TTS para generar audio a partir de modelos sintetizadores de audio, los cuales fueron descargados manualmente desde la web y colocados junto al programa en una carpeta para su posterior uso, el que se usa es un modelo de voz llamado “es_MX-claude-high”, también se inicia un hilo para reproducir los audios que se generen mientras la IA sigue generando la respuesta, cuando se trata de Linux, se usa una librería de pygame de nombre mixer para poder redirigir el audio generado por el TTS a las bocinas conectadas a la Raspberry Pi 4 que tienen un nombre específico.

Una vez iniciado todos esos sistemas, la secuencia de eventos es como sigue:

1. Se espera a que pase IDLE_TIME_DIALOG de tiempo o que se reciba una entrada del audio, ya sea en texto (USE_TEXT_QUERY = True) o en audio (presionando la tecla R con el botón) ejecutando la función listen_microphone().
2. Se escucha por el micrófono con Pyaudio en la función listen_microphone() si es que se usa la opción de voz, utilizando la IA reconocedora de voz y se obtiene una

forma textual de la entrada de audio usando el modelo de reconocimiento de voz asignado en las variables (cuando se usa el modo textual es directo el texto).

3. Si el texto no se pudo reconocer por defecto se manda como texto “Error, did not understand what you said.”, si eso sucede, se aborta la operación y se vuelve al primer paso.
4. Se entra aun ciclo while donde se mete la solicitud del usuario en el historial de conversación en la variable “conversation” y se entrega a la IA generadora de respuestas hasta que genere una respuesta, encerrándolo en un bucle while True.
5. Despues la respuesta que vaya generando la IA se va acumulando en una variable “response_text” y una copia de este en “temp_msg”.
6. Cada vez que se agrega un carácter de la respuesta de la IA se busca que contenga un set de “caracteres terminadores de frases” para cortar en texto más corto la respuesta de la IA y se sustraen de la variable “temp_msg”.
7. Para cada trozo obtenido se pasa a la función speak() que se encarga de generar un audio para decir el texto cortado mientras la IA sigue generando la respuesta, una vez el audio es generado se incrementa un índice contador de audios generados para que el reproductor de audios pueda saber si hay un audio que reproducir y se repite hasta que la IA termine de responder.
8. Antes de generar el audio del texto cortado, se ejecutan los comandos que esten en corchetes [] que son los que la IA puede usar para mover los servo motores o realizar una búsqueda en la base de datos.
9. Si la IA ejecuta el comando RAG_SEARCH para buscar en la base de datos, todo se aborta y se repite el ciclo while en el que esta y pone las variables para ejecutar la búsqueda y formatear la respuesta para entregar a la IA y volver a ejecutar desde el paso 4.
10. Después de que la IA termina de generar la respuesta y los audios son generados con la respuesta de la IA, se queda en espera de que todos los audios hayan terminado de reproducirse y se vuelve al primer paso.

A continuación, se muestra el código de la función que realiza todo lo anterior, se utilizan de varios “locks” para asegurar que la información de variables compartidas no se corrompa por el multiprocesamiento.

```
def query_rag(habla, key_pressed, lock, lock2, lock3, lock4, lock5,
query_text, state, generated_audio_index, reproduced_audio_index,
commands_queue_index, i_amount):
    import random
    from huggingface_hub import InferenceClient
    from langchain_community.embeddings import
HuggingFaceInferenceAPIEmbeddings
    from langchain_huggingface import HuggingFaceEmbeddings
    from langchain_chroma import Chroma
    from pygame import mixer

    client = InferenceClient(model=CHAT_MODEL, token=HF_TOKEN)
    stt_client = InferenceClient(model=AUDIO_RECOGNITION_MODEL,
token=HF_TOKEN)

    servo_angles = {"rs": 0, "re": 90, "ls": 0, "le": 90}

    if (platform == "Linux" and SERVOS):
        print("Linux Platform Detected")
        print("Make sure the Servos are connected properly to the Raspberry
Pi, for Orange Pi you will need to change wiringpi library")
        print("If you are using Linux OS on a VM or physical computer,
you'll see an error with the wiringpi library most likely")
        print("Servo Signal Activated, the servos will move to AI commands")

        hilo_servo = threading.Thread(target=control_servo_movement,
daemon=True, args=(servo_angles,))
        hilo_servo.start()

    if (DEBUG_MODE):
        print("Loading Embedding Model Connection")

    if (ONLINE_RAG):
        embedding_function =
HuggingFaceInferenceAPIEmbeddings(model_name=RAG_MODEL_ONLINE,
api_key=HF_TOKEN)
    else:
        model_kwargs = {'device': 'cpu'}
        encode_kwargs = {'normalize_embeddings': False}
        embedding_function =
HuggingFaceEmbeddings(model_name=RAG_MODEL_LOCAL, model_kwargs=model_kwargs,
encode_kwargs=encode_kwargs, api_key=HF_TOKEN)

    print("Initiating Embedding Test")

    try:
```

```
        print(embedding_function.embed_query("hello world")[:10])
        print("Test Successful!")
    except Exception as e:
        print("Embedding Test Errored: " + str(e))
        print("Check if it's an internet connection problem, this affects
the performace of the program critically")

    CHROMA_PATH = directorio + "/Base de Datos Vectorial Chroma"

    PROMPT_TEMPLATE = """
Responde la anterior pregunta basado solamente en el siguiente contexto:

{context}

---

Responde la anterior pregunta basada en el contexto de arriba: {question}
"""

    db = Chroma(persist_directory=CHROMA_PATH,
embedding_function=embedding_function)

    conversation = [{"role": "system", "content":
"""Your name is Luminia and you speak only in Spanish, you're an assistant
for helping
students in the university with the information provided to you, giving
short and preccise answers.

In your answers you must add commands between brackets to move the servo
motors to give a more vibid look when
talking, you must put a number between 0 and 180 to determinate the rotation
of the motor, you only can use them
once per sentence and they must be at the beginning of each sentence, they
control parts of two arms, these commands are as follows:

[rs] - Moves the right shoulder, 0 is down, 180 is straight up.
[re] - Moves the right elbow, 90 is straight with the right shoulder
orientation, 0 moves it to the center of the body
(must have right shoulder at 55 or higher), 180 moves it away from the body.
[ls] - Moves the left shoulder, 0 is down, 180 is straight up.
[le] - Moves the left elbow, 90 is straight with the left shoulder
orientation, 0 moves it closer to the body
(must have left shoulder at 55 or higher), 180 moves it away from the body.
```

```
A sentence is counted when a dot (.), question mark, exclamation mark or
semicolon is found in the response.

Here's an example of a response:
"[rs:135][re:135]Hola estudiante, muy buenos días.
[rs:45][re:135][ls:45][le:135]¿Hay algo en lo que pueda ayudarte hoy?
[ls:90]Puedes preguntarme lo que sea."

When someone asks for information about something or the location of some
departament or places, if possible assume it's information from the
TecNM aka the Tecnologic Institute of Costa Grande, and respond only with:
[RAG_SEARCH], nothing else, otherwise respond normally if no
question is asked.

After the proper context is given to you, try to answer the request from the
student, if none of the information provided to you can
answer the question simply say that the information couldn't be found, maybe
it's not in the database and probably should ask a person from the
institute instead.

In your responses avoid at all cost placing </s> or unreadable stuff without
proper context, your responses are read aloud by a text to speech
application, so avoid placing any unreadable stuff in your response other
than the commands allowed, additionally, mathematical expressions
such as x^2 put them as x squared and such so the text to speech can
pronounce them properly."""},
    {'role': 'assistant', 'content': '[rs:135][re:135]Hola estudiante, ¿En
que puedo servirte el dia de hoy?'}]
    presentation_dialog = [{"role": "system", "content":
conversation[0]["content"]},
                           {'role': 'assistant', 'content':
'[rs:135][re:135]Hola profesor, ¿En que puedo ser de utilidad el dia de
hoy?'}]

    idle_timer = 0

    lock3.acquire()
    state.value = 0
    lock3.release()

    if (DEBUG_MODE):
        print("Voice Engine Initializing")

    if (platform == "Linux"):
        from dimits import Dimits
```

```
        mixer.init(devicename=OUTPUT_DEVICE)

        hilo_reproduce = threading.Thread(target=reproduce_audio,
daemon=True, args=(lock, lock3, lock5, reproduced_audio_index,
generated_audio_index, mixer, servo_angles, commands_queue_index, i_amount))
        hilo_reproduce.start()

        engine = Dimits(directorio + TTS_MODEL_PATH)
    else:
        import pyttsx3

        engine = pyttsx3.init()

        engine.setProperty("rate", 160)

    lock4.acquire()
    key_pressed.value = 0
    lock4.release()

    lock2.acquire()
    query_text.value = ""
    lock2.release()

    print("Ready to operate!")

    if (USE_TEXT_QUERY):
        print("Type something to the AI now")

    while True:
        if (idle_timer >= IDLE_DIALOG_TIME*20):
            idle_timer = 0

        if (USE_TEXT_QUERY):
            lock2.acquire()
            if (query_text.value == ""):
                lock2.release()
                time.sleep(0.05)

                continue

            lock2.release()
        else:
            lock4.acquire()
            key = key_pressed.value
            lock4.release()
```

```
            if (key == 0):
                idle_timer += 1

                if (idle_timer < IDLE_DIALOG_TIME*20):
                    time.sleep(0.05)

                    continue
            else:
                idle_timer = 0

            if (DEBUG_MODE):
                if (idle_timer < IDLE_DIALOG_TIME*20):
                    print("Key Detected")
                else:
                    print("Initiaing Idle Dialog")

            lock2.acquire()
            if (idle_timer < IDLE_DIALOG_TIME*20):
                try:
                    query_text.value = listen_microphone(stt_client, lock3,
lock4, state, key_pressed)
                except Exception as e:
                    print("STT errored: " + str(e))

                    query_text.value = ""
            else:
                query_text.value = "Presentaté ante los estudiantes
universitarios por favor."

                lock3.acquire()
                state.value = 2
                lock3.release()

            if (query_text.value == "" or query_text.value == "Error, did
not understand what you said."):
                lock2.release()

                lock4.acquire()
                key_pressed.value = 0
                lock4.release()

                lock3.acquire()
                state.value = 0
                lock3.release()
```

```
                time.sleep(0.05)

                continue

            lock2.release()

        if (DEBUG_MODE):
            print("User Text Received: " + query_text.value)

        do_rag = False
        loop = True

        while (loop):
            if (idle_timer < IDLE_DIALOG_TIME*20 and do_rag):
                if (DEBUG_MODE):
                    print("Searching Chroma Database Similarity")

                # Search the DB.
                while True:
                    try:
                        results =
db.similarity_search_with_score(query_text.value, k=10)
                        break
                    except:
                        pass

                if (DEBUG_MODE):
                    print("Formatting User Prompt")

                context_text = "\n\n---\n\n".join([doc.page_content for doc,
_score in results])
                prompt = PROMPT_TEMPLATE.format(context=context_text,
question=query_text.value)
                # print(prompt)
            else:
                prompt = query_text.value

            if (idle_timer < 1000):
                conversation.append({"role": "user", "content": prompt})
            else:
                presentation_dialog.append({"role": "user", "content":
prompt})

            if (DEBUG_MODE):
```

```
                print("Generating AI Response")

            lock3.acquire()
            state.value = 2
            lock3.release()

            while True:
                try:
                    if (idle_timer < IDLE_DIALOG_TIME*20):
                        completion = client.chat_completion(conversation,
stream=True, max_tokens=1000, seed=random.randint(10000000,100000000))
                    else:
                        completion =
client.chat_completion(presentation_dialog, stream=True, max_tokens=1000,
seed=random.randint(10000000,100000000))
                    break
                except:
                    pass

            if (DEBUG_MODE):
                print("AI Reponse Generating.")

            response_text = ""
            temp_msg = ""
            phrase_enders = [".", "?", "!", ";", "]"]

            lock5.acquire()
            generated_audio_index.value = 1
            reproduced_audio_index.value = 1
            i_amount.value = 1000
            commands_queue_index[:] = []
            lock5.release()

            try:
                for chunk in completion:
                    if (chunk.choices[0].delta.content is not None):
                        part = chunk.choices[0].delta.content
                        response_text += part
                        temp_msg += part
                        index = len(temp_msg)

                        for character in phrase_enders:
                            index2 = temp_msg.find(character)
                            if (index2 >= 0):
                                index = min(index, index2)
```

```
                        if (index < len(temp_msg)):
                            do_rag = speak(habla, lock, lock3, lock5,
temp_msg[:index + 1], servo_angles, engine, generated_audio_index,
commands_queue_index)

                            if (do_rag):
                                break

                            if (platform == "Linux"):
                                lock5.acquire()
                                generated_audio_index.value += 1
                                lock5.release()

                            temp_msg = temp_msg[index + 1:]

                        #print(part, end="")
                        #sys.stdout.flush()

                if (do_rag):
                    lock3.acquire()
                    state.value = 3
                    lock3.release()

                    continue
            except Exception as e:
                if (DEBUG_MODE):
                    print("AI Generation Streaming Errored: " + str(e))

            if (DEBUG_MODE):
                print("AI Generated: " + response_text)

            if ("[RAG_SEARCH]" in temp_msg):
                do_rag = True
                continue

            loop = False

        if (idle_timer < IDLE_DIALOG_TIME*20):
            if (do_rag):
                conversation.pop()
                conversation.append({"role": "user", "content":
query_text.value})
```

```
            conversation.append({"role": "assistant", "content":
response_text})
        else:
            presentation_dialog.pop()

        if (DEBUG_MODE and do_rag):
            sources = [doc.metadata.get("id", None) for doc, _score in
results]
            print(f"Context Sources: {sources}")

        lock4.acquire()
        key_pressed.value = 0
        lock4.release()

        lock2.acquire()
        query_text.value = ""
        lock2.release()

        while True:
            lock5.acquire()
            if (i_amount.value == 1000):
                i_amount.value = generated_audio_index.value
            elif (i_amount.value <= reproduced_audio_index.value):
                break
            lock5.release()

            time.sleep(0.05)
        lock5.release()

        print("Ready to operate again!")

        if (USE_TEXT_QUERY):
            print("Type something to the AI now")
```

Note que se usa de variables especiales como USE_TEXT_QUERY y DEBUG_MODE, estas variables son usadas para debugear el programa en diversas formas o cambiar el dispositivo de salida de audio, muchas de las características son solo disponibles cuando DEBUG_MODE esta activado en general.

Adicionalmente a esto, las funciones de listen_microphone() solo usan PyAudio para captar el audio del microfono como es correspondiente con la configuración apropiada y puede ser parada según el estado de presionado del botón o tecla R.

```
def listen_microphone(client, lock3, lock4, state, key_pressed):
    import speech_recognition as sr, pyaudio, wave, struct

    AUDIO_FILE = directorio + "/audios/recording.wav"
    SILENCE_TIME = 2
    LISTENING_TIME = 10
    audio_buffer = []

    if (DEBUG_MODE):
        print("Trying to open microphone")

    p = pyaudio.PyAudio()
    device_count = p.get_device_count()
    for i in range(0, device_count):
        device_info = p.get_device_info_by_index(i)
        if (MICROPHONE_DEVICE in device_info["name"]):
            device_index = i

            break

    FORMAT = pyaudio.paInt16
    CHANNELS = device_info["maxInputChannels"]
    RATE = int(device_info["defaultSampleRate"])
    CHUNK = 1024

    stream = p.open(format=FORMAT, channels=CHANNELS, rate=RATE, input=True,
input_device_index=device_index, frames_per_buffer=CHUNK)
    silence_start = None
    started_talking = False
    listening_start = time.time()

    if (DEBUG_MODE):
        print("Listening...")

    lock3.acquire()
    state.value = 1
    lock3.release()

    while True:
        data = stream.read(CHUNK)
        audio_buffer.append(data)

        if (len(audio_buffer) > 0):
            data = audio_buffer[-1]
```

```
            samples = struct.unpack(f'{len(data)//2}h', data)  # "h" para
enteros de 16 bits (paInt16)
            volume = max(abs(sample) for sample in samples)

            if (volume < 500 and started_talking):
                if silence_start is None:
                    silence_start = time.time()
                elif time.time() - silence_start >= SILENCE_TIME:
                    if (DEBUG_MODE):
                        print("User Stopped Talking.")

                    break
            elif (not started_talking):
                started_talking = True
            else:
                silence_start = None

        if (time.time() - listening_start >= LISTENING_TIME):
            if (DEBUG_MODE):
                print("Timeout.")

            break

        lock4.acquire()
        key = key_pressed.value
        lock4.release()

        if (key == 2):
            if (DEBUG_MODE):
                print("Process Stopped.")

            break

    stream.stop_stream()
    stream.close()
    p.terminate()

    if (DEBUG_MODE):
        print("Stopped listening.")

    wf = wave.open(AUDIO_FILE, 'wb')
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(pyaudio.PyAudio().get_sample_size(FORMAT))
    wf.setframerate(RATE)
    wf.writeframes(b''.join(audio_buffer))
```

```
    wf.close()

    recognizer = sr.Recognizer()
    with sr.AudioFile(directorio + "/audios/recording.wav") as source:
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.record(source)

    lock3.acquire()
    state.value = 2
    lock3.release()

    try:
        return
client.automatic_speech_recognition(audio.get_wav_data()).text
    except Exception as e:
        print("Recognizer STT Errored: ",end="")
        print(e)

        return "Error, did not understand what you said."
```

La función reproduce_audio() se encarga de esperar y reproducir los audios por el canal adecuado cada vez que se encuentra disponible, es un ciclo que esta esperando siempre que un audio este disponible para su reproducción.

```
def reproduce_audio(lock, lock3, lock5, j, i, mixer, servo_angles, i_list,
i_amount):
    while True:
        lock5.acquire()
        j_value = j.value
        i_value = i.value
        i_amount_value = i_amount.value
        lock5.release()

        if (j_value >= i_value):
            if (j_value >= i_amount_value):
                servo_angles["rs"] = 0
                servo_angles["re"] = 90
                servo_angles["ls"] = 0
                servo_angles["le"] = 90

            time.sleep(0.05)

            continue
        index = j_value - 1
```

```
        servo_angles["rs"] = i_list[index]["rs"]
        servo_angles["re"] = i_list[index]["re"]
        servo_angles["ls"] = i_list[index]["ls"]
        servo_angles["le"] = i_list[index]["le"]

        mixer.music.load(directorio + "/audios/speech" + str(index + 1) +
".wav")
        mixer.music.play()

        lock3.acquire()
        state.value = 0
        lock3.release()

        lock.acquire()
        habla.value = 1
        lock.release()

        while True:
            if (not mixer.music.get_busy()):
                break

            time.sleep(0.05)

        lock.acquire()
        habla.value = 0
        lock.release()

        lock5.acquire()
        j.value += 1
        lock5.release()
```

La función speak() es la que recibe el texto y ejecuta los comando que contenga antes de generar el audio correspondiente con el TTS correspondiente según el sistema operativo.

```
def speak(habla, lock, lock3, lock5, text, servo_angles, engine, i, i_list):
    if (DEBUG_MODE):
        print("Raw TTS: " + text)

    if (platform == "Linux"):
        lock5.acquire()
        index = len(i_list[:]) - 1
        if (index == -1):
            i_list.append({"rs": 0, "re": 90, "ls": 0, "le": 90})
        else:
```

```
            i_list.append({"rs": i_list[index]["rs"], "re":
i_list[index]["re"], "ls": i_list[index]["ls"], "le": i_list[index]["le"]})
        lock5.release()

    index = text.find("[")
    while (index >= 0):
        index2 = text.find("]")

        if (index2 >= 0):
            result = execute_command(lock5, text[index + 1:index2],
servo_angles, i_list)

            if (result == "RAG"):
                return True

            text = text[:index] + text[index2 + 1:]
        index = text.find("[")

    if (DEBUG_MODE):
        print("Speaking TTS: " + text)

    if (text == "" or text == "."):
        lock5.acquire()
        i.value -= 1
        lock5.release()

        return False

    if (platform == "Linux"):
        lock5.acquire()
        aux = i.value
        lock5.release()

        engine.text_2_audio_file(text, "audios/speech" + str(aux),
directorio, format="wav")
    else:
        engine.say(text)

        lock3.acquire()
        state.value = 0
        lock3.release()

        lock.acquire()
        habla.value = 1
        lock.release()
```

```
        engine.runAndWait()

        lock.acquire()
        habla.value = 0
        lock.release()

    return False
```

Se complementa de la función que ejecuta los comando execute_command().

```
def execute_command(lock5, text, servo_angles, i_list):
    separation = text.split(":")
    command = separation[0].replace(" ","")
    value = int(separation[1])

    if (DEBUG_MODE):
        print("Command Executed: " + command + " with value " + str(value))

    if (value < 0 or value > 180):
        return

    if (command == "RAG_SEARCH"):
        return "RAG"

    if (platform == "Linux"):
        lock5.acquire()
        index = len(i_list[:]) - 1
        servo_angles = i_list[index]
        lock5.release()

    if (command == "rs"):
        servo_angles["rs"] = value

        if (2*value + servo_angles["re"] < 90):
            servo_angles["re"] = 90 - 2*value
        elif (2*value + servo_angles["re"] > 450):
            servo_angles["re"] = 450 - 2*value
    elif (command == "re"):
        servo_angles["re"] = value

        if (value + 2*servo_angles["rs"] < 90):
            servo_angles["rs"] = (90 - value)/2
        elif (2*value + servo_angles["rs"] > 450):
            servo_angles["rs"] = (450 - value)/2
```

```
    elif (command == "ls"):
        servo_angles["ls"] = value

        if (2*value + servo_angles["le"] < 90):
            servo_angles["le"] = 90 - 2*value
        elif (2*value + servo_angles["le"] > 450):
            servo_angles["le"] = 450 - 2*value
    elif (command == "le"):
        servo_angles["le"] = value

        if (value + 2*servo_angles["ls"] < 90):
            servo_angles["ls"] = (90 - value)/2
        elif (2*value + servo_angles["ls"] > 450):
            servo_angles["ls"] = (450 - value)/2

    if (platform == "Linux"):
        lock5.acquire()
        i_list[index] = servo_angles
        lock5.release()

    return ""
```

5.2.5. Código Completo del Sistema Principal

Como en las anteriores secciones solo se mostraba lo relevante de cada sistema, aquí se puede observar la forma en la que los 3 sistemas se juntan en un solo script, usualmente el archivo esta bajo el nombre de main.py:

```
import os, math, threading, multiprocessing, platform,time

platform = platform.system()

WIDTH, HEIGHT = 800, 600
FACE_COLOR = (255, 255, 255)
DEBUG_MODE = True
USE_TEXT_QUERY = False
ONLINE_RAG = True
SERVOS = True
SERVO_ANGLES_DEBUG_TEXT_FREQUENCY = 120
IDLE_DIALOG_TIME = 90
OUTPUT_DEVICE = "bcm2835 Headphones, bcm2835 Headphones"
MICROPHONE_DEVICE = "Poly Blackwire 3320 Series: USB Audio"
CHAT_MODEL = "Qwen/Qwen2.5-Coder-32B-Instruct"
AUDIO_RECOGNITION_MODEL = "openai/whisper-large-v3-turbo"
RAG_MODEL_ONLINE = "intfloat/multilingual-e5-large-instruct"
```

```
RAG_MODEL_LOCAL = ""
TTS_MODEL_PATH = "/voices/es_MX-claude-high"
HF_TOKEN = "<YOUR HUGGING FACE KEY>”

directorio = os.path.dirname(os.path.realpath(__file__))

def query_rag(habla, key_pressed, lock, lock2, lock3, lock4, lock5,
query_text, state, generated_audio_index, reproduced_audio_index,
commands_queue_index, i_amount):
    import random
    from huggingface_hub import InferenceClient
    from langchain_community.embeddings import
HuggingFaceInferenceAPIEmbeddings
    from langchain_huggingface import HuggingFaceEmbeddings
    from langchain_chroma import Chroma
    from pygame import mixer

    client = InferenceClient(model=CHAT_MODEL, token=HF_TOKEN)
    stt_client = InferenceClient(model=AUDIO_RECOGNITION_MODEL,
token=HF_TOKEN)

    servo_angles = {"rs": 0, "re": 90, "ls": 0, "le": 90}

    if (platform == "Linux" and SERVOS):
        print("Linux Platform Detected")
        print("Make sure the Servos are connected properly to the Raspberry
Pi, for Orange Pi you will need to change wiringpi library")
        print("If you are using Linux OS on a VM or physical computer,
you'll see an error with the wiringpi library most likely")
        print("Servo Signal Activated, the servos will move to AI commands")

        hilo_servo = threading.Thread(target=control_servo_movement,
daemon=True, args=(servo_angles,))
        hilo_servo.start()

    if (DEBUG_MODE):
        print("Loading Embedding Model Connection")

    if (ONLINE_RAG):
        embedding_function =
HuggingFaceInferenceAPIEmbeddings(model_name=RAG_MODEL_ONLINE,
api_key=HF_TOKEN)
    else:
        model_kwargs = {'device': 'cpu'}
        encode_kwargs = {'normalize_embeddings': False}
```

```
        embedding_function =
HuggingFaceEmbeddings(model_name=RAG_MODEL_LOCAL, model_kwargs=model_kwargs,
encode_kwargs=encode_kwargs, api_key=HF_TOKEN)

    print("Initiating Embedding Test")

    try:
        print(embedding_function.embed_query("hello world")[:10])
        print("Test Successful!")
    except Exception as e:
        print("Embedding Test Errored: " + str(e))
        print("Check if it's an internet connection problem, this affects
the performace of the program critically")

    CHROMA_PATH = directorio + "/Base de Datos Vectorial Chroma"

    PROMPT_TEMPLATE = """
Responde la anterior pregunta basado solamente en el siguiente contexto:

{context}

---

Responde la anterior pregunta basada en el contexto de arriba: {question}
"""

    db = Chroma(persist_directory=CHROMA_PATH,
embedding_function=embedding_function)

    conversation = [{"role": "system", "content":
"""Your name is Luminia and you speak only in Spanish, you're an assistant
for helping
students in the university with the information provided to you, giving
short and preccise answers.

In your answers you must add commands between brackets to move the servo
motors to give a more vibid look when
talking, you must put a number between 0 and 180 to determinate the rotation
of the motor, you only can use them
once per sentence and they must be at the beginning of each sentence, they
control parts of two arms, these commands are as follows:

[rs] - Moves the right shoulder, 0 is down, 180 is straight up.
[re] - Moves the right elbow, 90 is straight with the right shoulder
orientation, 0 moves it to the center of the body
```

```
(must have right shoulder at 55 or higher), 180 moves it away from the body.
[ls] - Moves the left shoulder, 0 is down, 180 is straight up.
[le] - Moves the left elbow, 90 is straight with the left shoulder
orientation, 0 moves it closer to the body
(must have left shoulder at 55 or higher), 180 moves it away from the body.

A sentence is counted when a dot (.), question mark, exclamation mark or
semicolon is found in the response.

Here's an example of a response:
"[rs:135][re:135]Hola estudiante, muy buenos días.
[rs:45][re:135][ls:45][le:135]¿Hay algo en lo que pueda ayudarte hoy?
[ls:90]Puedes preguntarme lo que sea."

When someone asks for information about something or the location of some
departament or places, if possible assume it's information from the
TecNM aka the Tecnologic Institute of Costa Grande, and respond only with:
[RAG_SEARCH], nothing else, otherwise respond normally if no
question is asked.

After the proper context is given to you, try to answer the request from the
student, if none of the information provided to you can
answer the question simply say that the information couldn't be found, maybe
it's not in the database and probably should ask a person from the
institute instead.

In your responses avoid at all cost placing </s> or unreadable stuff without
proper context, your responses are read aloud by a text to speech
application, so avoid placing any unreadable stuff in your response other
than the commands allowed, additionally, mathematical expressions
such as x^2 put them as x squared and such so the text to speech can
pronounce them properly."""},
    {'role': 'assistant', 'content': '[rs:135][re:135]Hola estudiante, ¿En
que puedo servirte el dia de hoy?'}]
    presentation_dialog = [{"role": "system", "content":
conversation[0]["content"]},
                           {'role': 'assistant', 'content':
'[rs:135][re:135]Hola profesor, ¿En que puedo ser de utilidad el dia de
hoy?'}]

    idle_timer = 0

    lock3.acquire()
    state.value = 0
    lock3.release()
```

```
    if (DEBUG_MODE):
        print("Voice Engine Initializing")

    if (platform == "Linux"):
        from dimits import Dimits

        mixer.init(devicename=OUTPUT_DEVICE)

        hilo_reproduce = threading.Thread(target=reproduce_audio,
daemon=True, args=(lock, lock3, lock5, reproduced_audio_index,
generated_audio_index, mixer, servo_angles, commands_queue_index, i_amount))
        hilo_reproduce.start()

        engine = Dimits(directorio + TTS_MODEL_PATH)
    else:
        import pyttsx3

        engine = pyttsx3.init()

        engine.setProperty("rate", 160)

    lock4.acquire()
    key_pressed.value = 0
    lock4.release()

    lock2.acquire()
    query_text.value = ""
    lock2.release()

    print("Ready to operate!")

    if (USE_TEXT_QUERY):
        print("Type something to the AI now")

    while True:
        if (idle_timer >= IDLE_DIALOG_TIME*20):
            idle_timer = 0

        if (USE_TEXT_QUERY):
            lock2.acquire()
            if (query_text.value == ""):
                lock2.release()
                time.sleep(0.05)
```

```
                continue

            lock2.release()
        else:
            lock4.acquire()
            key = key_pressed.value
            lock4.release()

            if (key == 0):
                idle_timer += 1

                if (idle_timer < IDLE_DIALOG_TIME*20):
                    time.sleep(0.05)

                    continue
            else:
                idle_timer = 0

            if (DEBUG_MODE):
                if (idle_timer < IDLE_DIALOG_TIME*20):
                    print("Key Detected")
                else:
                    print("Initiaing Idle Dialog")

            lock2.acquire()
            if (idle_timer < IDLE_DIALOG_TIME*20):
                try:
                    query_text.value = listen_microphone(stt_client, lock3,
lock4, state, key_pressed)
                except Exception as e:
                    print("STT errored: " + str(e))

                    query_text.value = ""
            else:
                query_text.value = "Presentaté ante los estudiantes
universitarios por favor."

                lock3.acquire()
                state.value = 2
                lock3.release()

            if (query_text.value == "" or query_text.value == "Error, did
not understand what you said."):
                lock2.release()
```

```
                lock4.acquire()
                key_pressed.value = 0
                lock4.release()

                lock3.acquire()
                state.value = 0
                lock3.release()

                time.sleep(0.05)

                continue

            lock2.release()

        if (DEBUG_MODE):
            print("User Text Received: " + query_text.value)

        do_rag = False
        loop = True

        while (loop):
            if (idle_timer < IDLE_DIALOG_TIME*20 and do_rag):
                if (DEBUG_MODE):
                    print("Searching Chroma Database Similarity")

                # Search the DB.
                while True:
                    try:
                        results =
db.similarity_search_with_score(query_text.value, k=10)
                        break
                    except:
                        pass

                if (DEBUG_MODE):
                    print("Formatting User Prompt")

                context_text = "\n\n---\n\n".join([doc.page_content for doc,
_score in results])
                prompt = PROMPT_TEMPLATE.format(context=context_text,
question=query_text.value)
                # print(prompt)
            else:
                prompt = query_text.value
```

```
            if (idle_timer < 1000):
                conversation.append({"role": "user", "content": prompt})
            else:
                presentation_dialog.append({"role": "user", "content":
prompt})

            if (DEBUG_MODE):
                print("Generating AI Response")

            lock3.acquire()
            state.value = 2
            lock3.release()

            while True:
                try:
                    if (idle_timer < IDLE_DIALOG_TIME*20):
                        completion = client.chat_completion(conversation,
stream=True, max_tokens=1000, seed=random.randint(10000000,100000000))
                    else:
                        completion =
client.chat_completion(presentation_dialog, stream=True, max_tokens=1000,
seed=random.randint(10000000,100000000))
                    break
                except:
                    pass

            if (DEBUG_MODE):
                print("AI Reponse Generating.")

            response_text = ""
            temp_msg = ""
            phrase_enders = [".", "?", "!", ";", "]"]

            lock5.acquire()
            generated_audio_index.value = 1
            reproduced_audio_index.value = 1
            i_amount.value = 1000
            commands_queue_index[:] = []
            lock5.release()

            try:
                for chunk in completion:
                    if (chunk.choices[0].delta.content is not None):
                        part = chunk.choices[0].delta.content
                        response_text += part
```

```
                        temp_msg += part
                        index = len(temp_msg)

                        for character in phrase_enders:
                            index2 = temp_msg.find(character)
                            if (index2 >= 0):
                                index = min(index, index2)

                        if (index < len(temp_msg)):
                            do_rag = speak(habla, lock, lock3, lock5,
temp_msg[:index + 1], servo_angles, engine, generated_audio_index,
commands_queue_index)

                            if (do_rag):
                                break

                            if (platform == "Linux"):
                                lock5.acquire()
                                generated_audio_index.value += 1
                                lock5.release()

                            temp_msg = temp_msg[index + 1:]

                        #print(part, end="")
                        #sys.stdout.flush()

                if (do_rag):
                    lock3.acquire()
                    state.value = 3
                    lock3.release()

                    continue
            except Exception as e:
                if (DEBUG_MODE):
                    print("AI Generation Streaming Errored: " + str(e))

            if (DEBUG_MODE):
                print("AI Generated: " + response_text)

            if ("[RAG_SEARCH]" in temp_msg):
                do_rag = True
                continue

            loop = False
```

```
        if (idle_timer < IDLE_DIALOG_TIME*20):
            if (do_rag):
                conversation.pop()
                conversation.append({"role": "user", "content":
query_text.value})

            conversation.append({"role": "assistant", "content":
response_text})
        else:
            presentation_dialog.pop()

        if (DEBUG_MODE and do_rag):
            sources = [doc.metadata.get("id", None) for doc, _score in
results]
            print(f"Context Sources: {sources}")

        lock4.acquire()
        key_pressed.value = 0
        lock4.release()

        lock2.acquire()
        query_text.value = ""
        lock2.release()

        while True:
            lock5.acquire()
            if (i_amount.value == 1000):
                i_amount.value = generated_audio_index.value
            elif (i_amount.value <= reproduced_audio_index.value):
                break
            lock5.release()

            time.sleep(0.05)
        lock5.release()

        print("Ready to operate again!")

        if (USE_TEXT_QUERY):
            print("Type something to the AI now")

def reproduce_audio(lock, lock3, lock5, j, i, mixer, servo_angles, i_list,
i_amount):
    while True:
        lock5.acquire()
        j_value = j.value
```

```
        i_value = i.value
        i_amount_value = i_amount.value
        lock5.release()

        if (j_value >= i_value):
            if (j_value >= i_amount_value):
                servo_angles["rs"] = 0
                servo_angles["re"] = 90
                servo_angles["ls"] = 0
                servo_angles["le"] = 90

            time.sleep(0.05)

            continue
        index = j_value - 1

        servo_angles["rs"] = i_list[index]["rs"]
        servo_angles["re"] = i_list[index]["re"]
        servo_angles["ls"] = i_list[index]["ls"]
        servo_angles["le"] = i_list[index]["le"]

        mixer.music.load(directorio + "/audios/speech" + str(index + 1) +
".wav")
        mixer.music.play()

        lock3.acquire()
        state.value = 0
        lock3.release()

        lock.acquire()
        habla.value = 1
        lock.release()

        while True:
            if (not mixer.music.get_busy()):
                break

            time.sleep(0.05)

        lock.acquire()
        habla.value = 0
        lock.release()

        lock5.acquire()
        j.value += 1
```

```
        lock5.release()

def speak(habla, lock, lock3, lock5, text, servo_angles, engine, i, i_list):
    if (DEBUG_MODE):
        print("Raw TTS: " + text)

    if (platform == "Linux"):
        lock5.acquire()
        index = len(i_list[:]) - 1
        if (index == -1):
            i_list.append({"rs": 0, "re": 90, "ls": 0, "le": 90})
        else:
            i_list.append({"rs": i_list[index]["rs"], "re":
i_list[index]["re"], "ls": i_list[index]["ls"], "le": i_list[index]["le"]})
        lock5.release()

    index = text.find("[")
    while (index >= 0):
        index2 = text.find("]")

        if (index2 >= 0):
            result = execute_command(lock5, text[index + 1:index2],
servo_angles, i_list)

            if (result == "RAG"):
                return True

            text = text[:index] + text[index2 + 1:]
        index = text.find("[")

    if (DEBUG_MODE):
        print("Speaking TTS: " + text)

    if (text == "" or text == "."):
        lock5.acquire()
        i.value -= 1
        lock5.release()

        return False

    if (platform == "Linux"):
        lock5.acquire()
        aux = i.value
        lock5.release()
```

```
        engine.text_2_audio_file(text, "audios/speech" + str(aux),
directorio, format="wav")
    else:
        engine.say(text)

        lock3.acquire()
        state.value = 0
        lock3.release()

        lock.acquire()
        habla.value = 1
        lock.release()

        engine.runAndWait()

        lock.acquire()
        habla.value = 0
        lock.release()

    return False

def listen_microphone(client, lock3, lock4, state, key_pressed):
    import speech_recognition as sr, pyaudio, wave, struct

    AUDIO_FILE = directorio + "/audios/recording.wav"
    SILENCE_TIME = 2
    LISTENING_TIME = 10
    audio_buffer = []

    if (DEBUG_MODE):
        print("Trying to open microphone")

    p = pyaudio.PyAudio()
    device_count = p.get_device_count()
    for i in range(0, device_count):
        device_info = p.get_device_info_by_index(i)
        if (MICROPHONE_DEVICE in device_info["name"]):
            device_index = i

            break

    FORMAT = pyaudio.paInt16
    CHANNELS = device_info["maxInputChannels"]
    RATE = int(device_info["defaultSampleRate"])
    CHUNK = 1024
```

```
    stream = p.open(format=FORMAT, channels=CHANNELS, rate=RATE, input=True,
input_device_index=device_index, frames_per_buffer=CHUNK)
    silence_start = None
    started_talking = False
    listening_start = time.time()

    if (DEBUG_MODE):
        print("Listening...")

    lock3.acquire()
    state.value = 1
    lock3.release()

    while True:
        data = stream.read(CHUNK)
        audio_buffer.append(data)

        if (len(audio_buffer) > 0):
            data = audio_buffer[-1]
            samples = struct.unpack(f'{len(data)//2}h', data)  # "h" para
enteros de 16 bits (paInt16)
            volume = max(abs(sample) for sample in samples)

            if (volume < 500 and started_talking):
                if silence_start is None:
                    silence_start = time.time()
                elif time.time() - silence_start >= SILENCE_TIME:
                    if (DEBUG_MODE):
                        print("User Stopped Talking.")

                    break
            elif (not started_talking):
                started_talking = True
            else:
                silence_start = None

        if (time.time() - listening_start >= LISTENING_TIME):
            if (DEBUG_MODE):
                print("Timeout.")

            break

        lock4.acquire()
        key = key_pressed.value
```

```
        lock4.release()

        if (key == 2):
            if (DEBUG_MODE):
                print("Process Stopped.")

            break

    stream.stop_stream()
    stream.close()
    p.terminate()

    if (DEBUG_MODE):
        print("Stopped listening.")

    wf = wave.open(AUDIO_FILE, 'wb')
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(pyaudio.PyAudio().get_sample_size(FORMAT))
    wf.setframerate(RATE)
    wf.writeframes(b''.join(audio_buffer))
    wf.close()

    recognizer = sr.Recognizer()
    with sr.AudioFile(directorio + "/audios/recording.wav") as source:
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.record(source)

    lock3.acquire()
    state.value = 2
    lock3.release()

    try:
        return
client.automatic_speech_recognition(audio.get_wav_data()).text
    except Exception as e:
        print("Recognizer STT Errored: ",end="")
        print(e)

        return "Error, did not understand what you said."

def execute_command(lock5, text, servo_angles, i_list):
    separation = text.split(":")
    command = separation[0].replace(" ","")
    value = int(separation[1])
```

```
    if (DEBUG_MODE):
        print("Command Executed: " + command + " with value " + str(value))

    if (value < 0 or value > 180):
        return

    if (command == "RAG_SEARCH"):
        return "RAG"

    if (platform == "Linux"):
        lock5.acquire()
        index = len(i_list[:]) - 1
        servo_angles = i_list[index]
        lock5.release()

    if (command == "rs"):
        servo_angles["rs"] = value

        if (2*value + servo_angles["re"] < 90):
            servo_angles["re"] = 90 - 2*value
        elif (2*value + servo_angles["re"] > 450):
            servo_angles["re"] = 450 - 2*value
    elif (command == "re"):
        servo_angles["re"] = value

        if (value + 2*servo_angles["rs"] < 90):
            servo_angles["rs"] = (90 - value)/2
        elif (2*value + servo_angles["rs"] > 450):
            servo_angles["rs"] = (450 - value)/2
    elif (command == "ls"):
        servo_angles["ls"] = value

        if (2*value + servo_angles["le"] < 90):
            servo_angles["le"] = 90 - 2*value
        elif (2*value + servo_angles["le"] > 450):
            servo_angles["le"] = 450 - 2*value
    elif (command == "le"):
        servo_angles["le"] = value

        if (value + 2*servo_angles["ls"] < 90):
            servo_angles["ls"] = (90 - value)/2
        elif (2*value + servo_angles["ls"] > 450):
            servo_angles["ls"] = (450 - value)/2

    if (platform == "Linux"):
```

```
        lock5.acquire()
        i_list[index] = servo_angles
        lock5.release()

    return ""

def control_servo_movement(servo_angles):
    try:
        import pigpio

        if (DEBUG_MODE):
            debug_timer = 0

        rs_pin = 22
        re_pin = 23
        ls_pin = 24
        le_pin = 25

        rs_angle_cur = 0
        re_angle_cur = 90
        ls_angle_cur = 0
        le_angle_cur = 90

        pi = pigpio.pi()

        pi.set_mode(rs_pin, pigpio.OUTPUT)
        pi.set_mode(re_pin, pigpio.OUTPUT)
        pi.set_mode(ls_pin, pigpio.OUTPUT)
        pi.set_mode(le_pin, pigpio.OUTPUT)

        pi.set_servo_pulsewidth(rs_pin, 2500)
        pi.set_servo_pulsewidth(re_pin, 1500)
        pi.set_servo_pulsewidth(ls_pin, 2500)
        pi.set_servo_pulsewidth(le_pin, 1500)

        if (DEBUG_MODE):
            print("Servo Loop Started")

        while True:
            rs_angle_cur = rs_angle_cur + (servo_angles["rs"] -
rs_angle_cur)/25
            re_angle_cur = re_angle_cur + (servo_angles["re"] -
re_angle_cur)/25
            ls_angle_cur = ls_angle_cur + (servo_angles["ls"] -
ls_angle_cur)/25
```

```
            le_angle_cur = le_angle_cur + (servo_angles["le"] -
le_angle_cur)/25

            if (DEBUG_MODE):
                if (debug_timer == 0):
                    print("Servo Right Shoulder Angle: " +
str(rs_angle_cur))
                    print("Servo Right Elbow Angle: " + str(re_angle_cur))
                    print("Servo Left Shoulder Angle: " + str(ls_angle_cur))
                    print("Servo Left Elbow Angle: " + str(le_angle_cur))

                    debug_timer = SERVO_ANGLES_DEBUG_TEXT_FREQUENCY*20
                else:
                    debug_timer -= 1

            pi.set_servo_pulsewidth(rs_pin, 500 + 2000*(180 -
rs_angle_cur)/180)
            pi.set_servo_pulsewidth(re_pin, 500 + 2000*(180 -
re_angle_cur)/180)
            pi.set_servo_pulsewidth(ls_pin, 500 + 2000*(180 -
ls_angle_cur)/180)
            pi.set_servo_pulsewidth(le_pin, 500 + 2000*(180 -
le_angle_cur)/180)

            time.sleep(0.05)
    except Exception as e:
        print("Servo Manager Errored: " + str(e))
        print("Servo Signal Deactivated, the servos won't move anymore")

def leer_input(query, lock2):
    while True:
        leido = input("")

        lock2.acquire()
        if (leido != "" and query.value == ""):
            query.value = leido
        lock2.release()

if (__name__ == "__main__"):
    import multiprocessing, pygame, subprocess

    manager = multiprocessing.Manager()
    lock = multiprocessing.Lock()
    lock2 = multiprocessing.Lock()
    lock3 = multiprocessing.Lock()
```

```
    lock4 = multiprocessing.Lock()
    lock5 = multiprocessing.Lock()
    habla = manager.Value("d", 0)
    query = manager.Value("s", "")
    state = manager.Value("d", 2) #0 is face, 1 is listening, 2 is
processing, 3 is searching
    amount_audios = manager.Value("d", 0)
    key_pressed = manager.Value("d", 0)
    generated_audio_index = manager.Value("d", 0)
    reproduced_audio_index = manager.Value("d", 0)
    commands_queue_index = manager.list()

    proceso_ai = multiprocessing.Process(target=query_rag, daemon=True,
args=(habla, key_pressed, lock, lock2, lock3, lock4, lock5, query, state,
generated_audio_index, reproduced_audio_index, commands_queue_index,
amount_audios))
    proceso_ai.start()

    if (USE_TEXT_QUERY):
        hilo_input = threading.Thread(target=leer_input, daemon=True,
args=(query, lock2))
        hilo_input.start()

    pygame.init()

    screen = pygame.display.set_mode((WIDTH, HEIGHT), pygame.SCALED)
    pygame.display.set_caption("Asistente Luminia")
    pygame.mouse.set_visible(False)

    listening_image =
pygame.image.load(os.path.dirname(os.path.realpath(__file__)) +
"/imagenes/listening.png").convert_alpha()
    processing_image =
pygame.image.load(os.path.dirname(os.path.realpath(__file__)) +
"/imagenes/processing.png").convert_alpha()
    searching_image =
pygame.image.load(os.path.dirname(os.path.realpath(__file__)) +
"/imagenes/searching.png").convert_alpha()
    background_image =
pygame.image.load(os.path.dirname(os.path.realpath(__file__)) +
"/imagenes/bg.png").convert_alpha()
    bg_width, bg_height = background_image.get_size()

    # Reloj para controlar la velocidad de fotogramas
    clock = pygame.time.Clock()
```

```
    def draw_rotated_face(center_x, center_y, angle):
        face_surface = pygame.Surface((WIDTH, HEIGHT), pygame.SRCALPHA)

        draw_rotated_eye(face_surface, WIDTH // 2 + (face_leftX -
60*face_scale), HEIGHT // 2 - (face_leftY + 50*face_scale),
face_leftSizeX*face_scale, face_leftSizeY*face_scale, face_leftAngle)
        draw_rotated_eye(face_surface, WIDTH // 2 + (face_rightX +
60*face_scale), HEIGHT // 2 - (face_rightY + 50*face_scale),
face_rightSizeX*face_scale, face_rightSizeY*face_scale, face_rightAngle)
        draw_rotated_rect(face_surface, WIDTH // 2 + face_mouthX, HEIGHT //
2 - (face_mouthY - (60 + face_mouthSizeY/2)*face_scale),
face_mouthSizeX*face_scale, (10 + face_mouthSizeY)*face_scale,
face_mouthAngle)

        rotated_face = pygame.transform.rotate(face_surface, angle)
        face_rect = rotated_face.get_rect(center=(center_x, center_y))

        screen.blit(rotated_face, face_rect)

    # Función para dibujar y rotar ojos
    def draw_rotated_eye(surface, center_x, center_y, width, height, angle):
        # Crear superficie para el ojo
        eye_surface = pygame.Surface((width, height), pygame.SRCALPHA)
        pygame.draw.ellipse(eye_surface, FACE_COLOR, (0, 0, width, height))

        # Rotar superficie
        rotated_eye = pygame.transform.rotate(eye_surface, angle)
        eye_rect = rotated_eye.get_rect(center=(center_x, center_y))

        # Dibujar ojo rotado en la pantalla
        surface.blit(rotated_eye, eye_rect)

    def draw_rotated_rect(surface, center_x, center_y, width, height,
angle):
        # Crear superficie para el ojo
        rect_surface = pygame.Surface((width, height), pygame.SRCALPHA)
        pygame.draw.rect(rect_surface, FACE_COLOR, (0, 0, width, height))

        # Rotar superficie
        rotated_eye = pygame.transform.rotate(rect_surface, angle)
        eye_rect = rotated_eye.get_rect(center=(center_x, center_y))

        # Dibujar ojo rotado en la pantalla
        surface.blit(rotated_eye, eye_rect)
```

```
    def draw_background():
        bg_surface = pygame.Surface((WIDTH + 200, HEIGHT + 300),
pygame.SRCALPHA)

        for x in range(-bg_width, WIDTH + 200, bg_width):
            for y in range(-bg_height, HEIGHT + 300, bg_height):
                bg_surface.blit(background_image, (x + offset_x, y +
offset_y))

        rotated_bg = pygame.transform.rotate(bg_surface, -20)
        bg_rect = rotated_bg.get_rect(center=(420, 300))

        screen.blit(rotated_bg, bg_rect)

    timer = 0
    timer2 = 0
    offset_x = 0
    offset_y = 0

    running = True
    checked = False
    last_time = None
    last_time2 = None
    shutdown = True
    fullscreen = False
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                shutdown = False
                running = False

            if (event.type == pygame.KEYDOWN):
                lock4.acquire()
                key = key_pressed.value
                lock4.release()

                if (event.key == pygame.K_ESCAPE):
                    shutdown = False
                    running = False
                elif (event.key == pygame.K_r):
                    if (key < 2):
                        checked = True

                        lock4.acquire()
```

```
                        key_pressed.value += 1
                        lock4.release()

                    if (last_time2 is not None and time.time() - last_time2
<= 1):
                        print("YES")
                        running = False

                        break

                    if (last_time is None or time.time() - last_time > 1):
                        last_time = time.time()
                    elif (time.time() - last_time <= 1):
                        last_time2 = time.time()
                elif (checked):
                    checked = False
            else:
                pressed_time = None

                if (checked):
                    checked = False

        if (not fullscreen):
            try:
                pygame.display.toggle_fullscreen()
                fullscreen = True
            except:
                pass

        offset_x = (offset_x + 3) % bg_width
        offset_y = (offset_y + 1.5) % bg_height

        draw_background()

        timer += 1.5

        lock.acquire()
        if (habla.value == 1 or timer2 > 0):
            timer2 += 1.5
            if (timer2%24 == 0):
                timer2 = 0
        lock.release()

        eye_width, eye_height = 60, 90
        face_scale = 2
```

```
        face_angle = 5*math.sin(math.radians(0.5*timer))
        face_leftX = 5*math.sin(math.radians(1.3*timer))
        face_leftY = 5*math.sin(math.radians(2*timer))
        face_leftAngle = 7*math.sin(math.radians(0.76*timer))
        face_leftSizeX = 90
        face_leftSizeY = 65 +
55*math.cos(math.radians(360*min(max(timer%800, 0), 15))/15)
        face_rightX = -5*math.sin(math.radians(1.1*timer))
        face_rightY = 5*math.sin(math.radians(1.76*timer))
        face_rightAngle = -7*math.sin(math.radians(0.93*timer))
        face_rightSizeX = 90
        face_rightSizeY = 65 +
55*math.cos(math.radians(360*min(max(timer%800, 0), 15))/15)
        face_mouthX = 3*math.sin(math.radians(0.73*timer))
        face_mouthY = -4*math.sin(math.radians(0.94*timer))
        face_mouthAngle = -5*math.sin(math.radians(1.2*timer))
        face_mouthSizeX = 180
        face_mouthSizeY = 30 - 15*math.cos(math.radians(15*timer2))

        lock3.acquire()
        if (state.value == 0):
            draw_rotated_face(WIDTH // 2, HEIGHT // 2, face_angle)
        elif (state.value == 1):
            timer = 0
            screen.blit(listening_image, (0, 0, WIDTH, HEIGHT))
        elif (state.value == 2):
            timer = 0
            screen.blit(processing_image, (0, 0, WIDTH, HEIGHT))
        else:
            timer = 0
            screen.blit(searching_image, (0, 0, WIDTH, HEIGHT))
        lock3.release()

        # Actualizar pantalla
        pygame.display.flip()

        # Controlar FPS
        clock.tick(60)

    # Cerrar Pygame
    pygame.quit()

    if (platform == "Linux" and shutdown):
        subprocess.call(["shutdown", "-h", "now"])
```

Note que el programa esta diseñado para ser probado en Windows también aunque principalmente corre en Linux, la única diferencia cae en los comandos que se deben ejecutar con subprocess como el shutdown y el TTS.

5.3. Construcción del Entorno Virtual de Python para el Programa

En caso de ser necesario la construcción del entorno virtual de Python para el programa ya sea por daño al entorno virtual por instalación de paquetes malos, daños a los archivos en este o conflicto de paquetes, **se tendrá que descargar Python 3.11.2 (de preferencia) y crear el entorno virtual con este en Linux** ya que el entorno virtual que se crea con Windows no funciona igual para Linux.

Una vez se tenga ese entorno virtual, utilizando pip o python de ese mismo se instalan los siguientes paquetes.

- pip install speechrecognition
- pip install huggingface_hub
- pip install langchain_huggingface
- pip install langchain_chroma
- pip install langchain_community
- pip install pigpio
- pip install pyaudio
- pip install pygame
- pip install dimits

Note que todos fueron colocados con pip, hay un conflicto de paquetes entre dimits y huggingface_hub que utilizan diferentes versiones del paquete tqdm que no coinciden, pero el programa funciona aun así con esas diferencias, por lo que **para evitar el conflicto se deben de instalar los paquetes uno por uno, no coloquen más de dos paquetes en el mismo comando** de preferencia.

Una vez instalado todo eso, el entorno virtual esta listo para correr el programa.

6. OBSERVACIONES Y SUGERENCIAS DE MEJORA

Los servo motores se ven que son capaces de moverse pero tiemblan un poco al momento de realizar movimientos, puede que apenas se tenga la capacidad para realizar los movimientos y necesite de servo motores de mayor capacidad de torque para realizar los movimientos de las manos en su lugar.

La captura de audio podría ser automatizada si de alguna forma se elabora un sistema que escuche una palabra clave y empiece a escuchar, algo así como lo que hace una Alexa.

En ambientes con mucho ruido es muy probable que no capte bien la respuesta de los usuarios, por lo que se tendría que gritarle o conseguir un mejor micrófono para mitigar ruidos externos.

La IA utilizada para generar respuestas tiene una capacidad 32 billones de parámetros, si las conversaciones se alargan es posible que olvide la existencia de los comandos y no los use, por lo que se puede probar con diferentes IAs del sitio web Hugging Face o conseguir la IA uno mismo para el funcionamiento del robot, con mayor capacidad de recordar comandos o tareas.

La IA para reconocer voz al no detectar audio del usuario por defecto devuelve “Thank you”, se puede modificar el código para evitar esto de alguna forma o usar una IA diferente para reconocer la voz.

Si el usuario no es muy específico con su petición, como, por ejemplo, especificar “jefe de departamento de servicios escolares” en vez de solo decir “jefe de servicios escolares”, la búsqueda de la información puede fallar y el robot no podrá responder a la solicitud del usuario apropiadamente, es probable que se necesite organizar y reestructurar la información en diferente forma en la base de datos vectorial para mejor eficiencia en la búsqueda de información o usar una IA embedding diferente hasta obtener los resultados deseados.

Actualmente al momento de escribir el manual, el robot no inicializa el programa al momento de encender el sistema operativo, he intentado hacerlo funcionar, pero no he tenido resultado, sugiero a quien quiera intentarlo a mirar este tutorial para colocar el

programa como un servicio dentro de Linux y así inicialice el programa al encender la Raspberry Pi 4: https://github.com/thagrol/Guides/blob/main/boot.pdf. De ser necesario usar la librería de subprocesos en el programa para ejecutar el comando `sudo pigpiod`.

7. PRECAUCIONES Y ADVERTENCIAS

Al momento de modificar el hardware del robot, asegurese de que no este enchufado u operando el programa, esto para evitar que los servo motores salgan quemados o las piezas se rompan en el proceso, también para evitar cortos circuitos.

Al momento de que el robot este operando, colocarlo en un lugar donde el movimiento de sus brazos no vaya a chocar con algo o alguien que pueda impedir el movimiento de sus manos u obstruir el paso a alguien, ya que puede haber accidentes que lleven a la quemadura de lo servo motores o rotura de piezas impresas.

Para evitar dañar la Raspberry Pi 4 del robot evitar que un corto circuito se realice en su cable de alimentación, si el cable se empieza a ver deteriorado o la cinta de aislar que une los cables se empieza a deteriorar, remplazar el cable inmediatamente antes de que suceda algo de mayor escala, también **eviten desconectar el robot mientras está operando, para apagar el robot se tiene que pulsar el botón o tecla R 3 veces consecutivas rápidamente**.

Si se desea modificar al programa en cualquier medida alguna, se deberá conectar un teclado externo y pulsar la tecla Q mientras el robot esta operando para cerrar el programa pero no apagar la Raspberry Pi 4, posteriormente realizar los cambios al programa o remplazar los archivos y volver a arrancar el programa.

8. PALABRAS FINALES

Este manual si así se puede llamar fue elaborado justo después de haber finalizado el último semestre de las materias de la carrera de Ingeniería en Sistemas Computacionales, actualmente al momento de escribir esto me encuentro en las vacaciones de Diciembre del 2024, esperando a que se de la fecha en la que elaboraré mis Residencias Profesionales para Titularme de la carrera.

Este proyecto llevo 2 años y medio en desarrolló para cubrir diversas cosas, entre ellas acreditar materias, promocionar la institución, pero más que nada para participar en eventos de Innovación Tecnológica conocido como InnovaTec, solo participo una vez en el InnovaTec 2023 y llego hasta la regional, la idea tiene potencial de eso estoy certero.

Al principio la idea era que este fuera un robot de promoción de ventas, evolucionó a uno de atención al cliente y ahora es un asistente para el Institutito Tecnológico de la Costa Grande y aunque ya no tendré el tiempo para seguirlo desarrollando, espero que quien quiera que este leyendo esto, pueda hacer uso de toda la información que se coloca en este manual ya sea para uso personal o para mejorar el proyecto como tal.

Las primeras iteraciones del robot no dieron mucha impresión, pero rápidamente creció a convertirse en lo que es ahora mismo, y aunque quizás no sea la gran cosa para cualquiera que este leyendo estas palabras en algún futuro, aprendí mucho realizando este proyecto, experimentando y empujando los limites de lo que parecía que no se iba a poder conseguir, lo que me llevo con todo esto es la experiencia y agradezco el apoyo de los docentes, amigos y compañeros que también contribuyeron al desarrollo de este proyecto, en especial a Joselito Chue Morales por su apoyo constante en este proyecto, gracias.

Dejo a manos del actual jefe de carrera de Ingeniería en Sistemas Computacionales, Irving Jesús Sánchez Campos, el robot y todos sus componentes, así como este manual para el mantenimiento y mejoramiento del robot a aquellos que considere adecuado para el trabajo, no seré el mejor documentando proyectos, pero hice mi mejor intento, espero les sea de ayuda a quien sea que este leyendo esto ahorita mismo.