



Universitat d'Alacant  
Universidad de Alicante

Grado en Ingeniería Robótica  
Robótica de servicios

**Proyecto Robótica de servicios**

**NurseBot: Sistema Integral de Asistencia  
de Medicación Automatizada**

Autor:

Miguel Ferrer Castellá

# Índice

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>Simulación en RobotStudio</b>	<b>3</b>
2.1	Componentes Gráficos de la Simulación . . . . .	3
2.2	Pinza Smart_Gripper_Servo_Fingers . . . . .	6
2.3	Constantes del Programa . . . . .	7
2.4	Señales de Entrada y Salida . . . . .	8
2.5	Funciones Principales . . . . .	9
2.6	Componentes Inteligentes . . . . .	10
2.7	Main: Flujo de programa . . . . .	14
2.8	Comunicación con la Base de Datos . . . . .	15
<b>3</b>	<b>Base de Datos</b>	<b>15</b>
<b>4</b>	<b>Control por Visión</b>	<b>18</b>
4.1	Experimentación . . . . .	18
4.2	Implementación . . . . .	19
4.3	Configuración . . . . .	21
4.4	Funcionamiento en la aplicación . . . . .	22
<b>5</b>	<b>Agente conversacional</b>	<b>22</b>
5.1	Detección de habla . . . . .	22
5.2	STT (Speech-to-text) . . . . .	23
5.3	RAG - LLM . . . . .	24
5.3.1	FlowiseAI . . . . .	24
5.3.2	Ollama . . . . .	26
5.4	Modelo de LLM (Llama3.1 8B) . . . . .	27
5.5	RAG (Retrieval-Augmented Generation) . . . . .	28
5.6	Traducción . . . . .	29
5.7	XTTS (Text-to-Speech) . . . . .	30
5.8	Flujo de Control . . . . .	30
<b>6</b>	<b>Interfaz</b>	<b>34</b>
6.1	Reconocimiento Facial . . . . .	35
6.2	Disposición de la interfaz . . . . .	35
6.3	Gestión de Base de Datos . . . . .	35
6.4	Integración con Archivos . . . . .	35
6.5	Funcionalidades . . . . .	35
<b>7</b>	<b>Vídeo de la aplicación</b>	<b>37</b>
<b>8</b>	<b>Conclusiones</b>	<b>37</b>
<b>9</b>	<b>Anexos</b>	<b>38</b>
<b>10</b>	<b>Bibliografía</b>	<b>39</b>

# 1 Introducción

El proyecto NurseBot ha sido desarrollado como un sistema integral de asistencia médica automatizada, diseñado para facilitar la dispensación de medicamentos y el monitoreo de pacientes en residencias de mayores. Este sistema combina tecnologías avanzadas como el reconocimiento facial, el procesamiento de voz y una base de datos robusta para identificar y administrar medicación personalizada, asegurando una atención más eficiente y reduciendo posibles errores humanos.

El funcionamiento del NurseBot se basa en un flujo estructurado y eficiente. Los ancianos acceden de uno en uno al sistema, momento en el que el robot reconoce sus caras utilizando algoritmos de visión por computadora. Si se identifica a un paciente nuevo, NurseBot le sirve la medicación correspondiente según su plan personalizado almacenado en la base de datos. Todo este proceso puede apreciarse en tiempo real a través del entorno de simulación en RobotStudio.

Mientras el robot realiza la dispensación, permite interactuar de forma dinámica con el paciente o con el personal de enfermería a través de un sistema de conversación. Esto incluye la capacidad de responder preguntas específicas sobre un paciente o consultas más abiertas relacionadas con la administración de medicamentos o sobre cualquier ámbito de interés del usuario. Además, el sistema cuenta con una interfaz visual diseñada para que el personal médico pueda actualizar recetas, modificar información de pacientes como su número de habitación o edad, y gestionar la asignación de medicamentos de forma intuitiva y rápida.

## Diagrama de Gantt

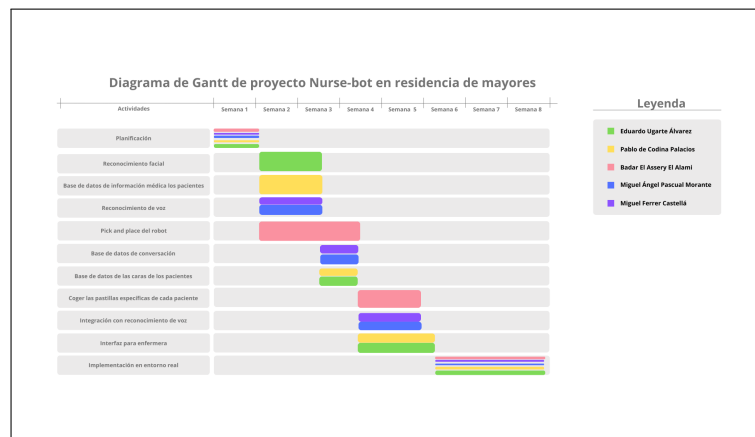


Figure 1: Diagrama de Gantt previo al desarrollo.

Durante la planificación inicial, se establecieron plazos y objetivos claros, reflejados en el diagrama de Gantt. A pesar de ciertos retrasos y ajustes derivados de la complejidad técnica, el equipo ha logrado cumplir en gran medida con los plazos previstos, destacando la capacidad de adaptación y gestión ante los desafíos encontrados.

## 2 Simulación en RobotStudio

### 2.1 Componentes Gráficos de la Simulación

#### Vista General de la Estación

Esta sección muestra una representación visual de toda la estación de trabajo, diseñada en herramientas básicas de diseño 3D como Tinkercad o desde la pestaña de modelado de RobotStudio.

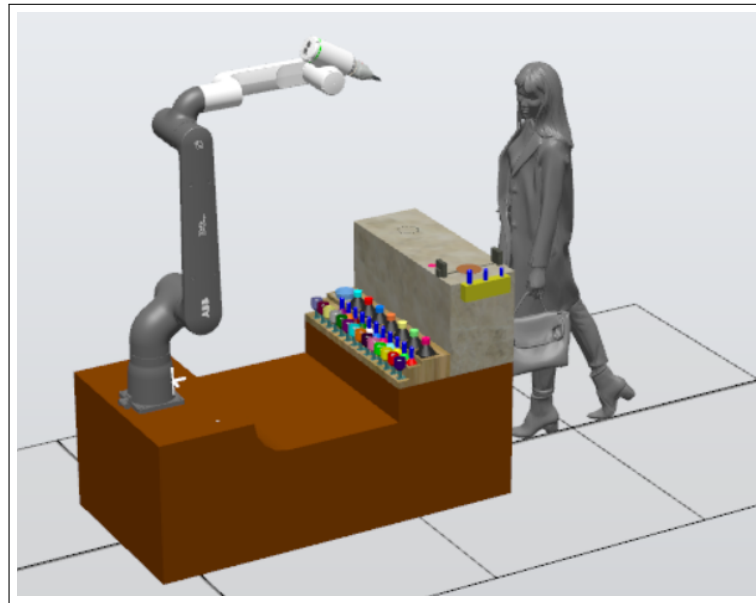


Figure 2: Escenario RobotStudio.

#### Elementos clave

- Jarabe

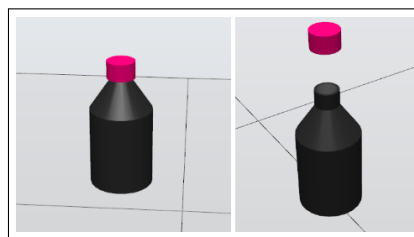


Figure 3: Jarabe.

- Goteras

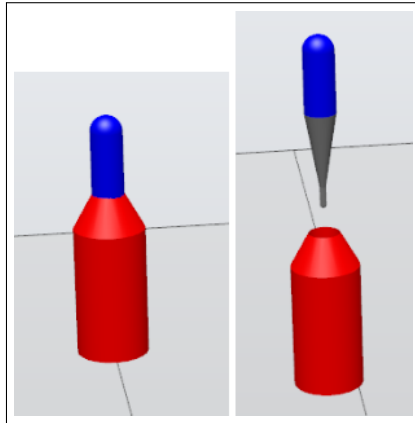


Figure 4: Goteras.

- Pastilleros



Figure 5: Pastilleros.

- **Presionador:** Componente que ejerce presión sobre los medicamentos para que el robot pueda destaparlos.

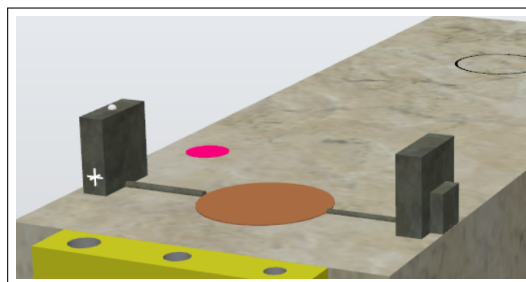


Figure 6: Presionador.

- **Cuentagotas:** Tres tipos diseñados para distintas dosis (10ml, 50ml y 100ml).

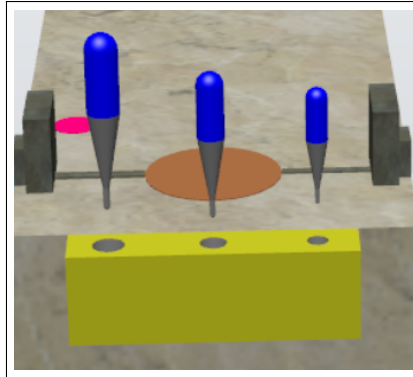


Figure 7: Cuentagotas.

- **Vasos**

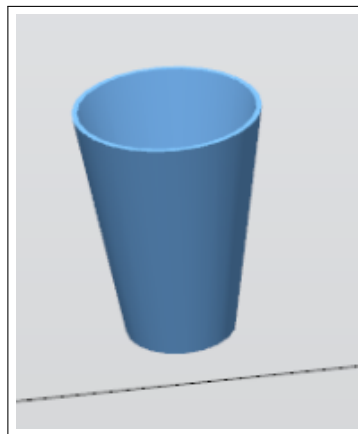


Figure 8: Vasos.

- **Estantería con Medicinas:** Ubicación organizada de los 32 medicamentos disponibles.

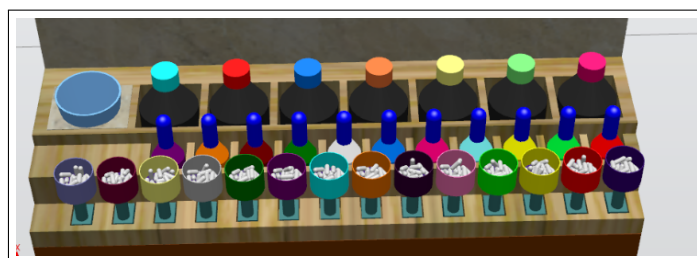


Figure 9: Estantería con Medicinas.

## 2.2 Pinza Smart\_Gripper\_Servo\_Fingers

La pinza Smart\_Gripper\_Servo\_Fingers es una pinza fácil de usar de la biblioteca de ABB, clave en NurseBot para sujetar diferentes tipos de objetos con precisión.

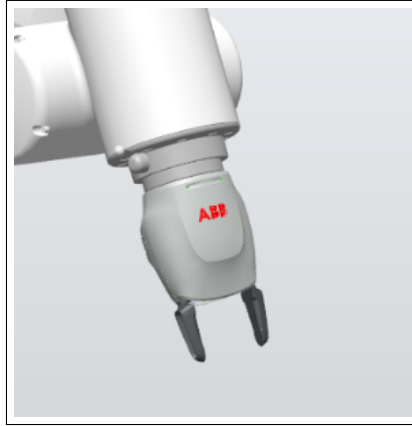


Figure 10: Smart\_Gripper\_Servo\_Fingers.

Este dispositivo cuenta con las siguientes características principales:

### 1. Diseño y Funcionalidad

- Capacidad para ajustar la apertura, actuando en su componente inteligente, según el tipo de objeto a manipular, ya sean vasos, medicamentos o tapas.
- Movimientos suaves y controlados, ideales para evitar daños a los objetos frágiles.
- Incluye sensores que garantizan un cierre óptimo para asegurar la estabilidad del objeto.

### 2. Aperturas Configurables

Las aperturas de la pinza están definidas en el programa a través de constantes como:

- **VASO\_PINZA**: Configuración para vasos.
- **GOTERA\_PINZA**: Configuración para goteras.
- **JARABE\_PINZA**: Configuración para jarabes.
- **PASTILLERO\_PINZA**: Configuración para pastilleros
- **CUENTAGOTAS\_PINZA**: Diferentes configuraciones según el tamaño del cuentagotas (10ml, 50ml, 100ml).

### 3. Señales Asociadas

- **Coger (1/0)**: Activa el cierre o apertura de la pinza.
- **apertura\_pinza**: Variable que define la distancia de cierre según el objeto.

#### 4. Funciones en RAPID Relacionadas

- **coger\_objeto(dnum objeto):** Configura la apertura de la pinza y activa la señal de cierre.
- **soltar\_objeto():** Desactiva la señal de cierre para liberar el objeto.

### 2.3 Constantes del Programa

#### MARGEN

Define la distancia al medicamento en la que se posiciona el robot antes de alcanzarlo. Es esencial para garantizar movimientos controlados, ya que algunas trayectorias no son lineales y pueden resultar impredecibles. Al estar a la distancia MARGEN, el robot realiza un movimiento lineal para evitar colisiones.

#### DISTANCIAS ENTRE ELEMENTOS

- **DISTANCIA\_ENTRE\_JARABES**
- **DISTANCIA\_ENTRE\_GOTERAS**
- **DISTANCIA\_ENTRE\_PASTILLEROS**

Estas constantes permiten calcular las posiciones de todos los medicamentos almacenando solo la ubicación del primer elemento de cada tipo.

#### Aperturas de la Pinza

- **VASO\_PINZA**
- **JARABE\_PINZA**
- **PASTILLERO\_PINZA**
- **CUENTAGOTAS\_100ML\_PINZA**
- **CUENTAGOTAS\_50ML\_PINZA**
- **CUENTAGOTAS\_10ML\_PINZA**

Definen la apertura o distancia de cierre de la pinza para manipular cada tipo de objeto.

#### Presiones

- **PRESION\_GOTERA**
- **PRESION\_JARABE**

Indican las posiciones específicas para ejercer presión sobre los medicamentos. Esto es necesario para retirar las tapas antes de dispensar.



## Posiciones Clave

El programa incluye varias posiciones importantes, entre las cuales destacan:

- **Home**
- **Goteras**
- **Jarabes**
- **Pastilleros**
- **Vasos**
- **Cuentagotas (10ml, 50ml, 100ml)**

## `posiciones_medicamentos`

Un vector que contiene las posiciones calculadas de los 32 medicamentos. Cada posición se accede mediante el identificador (*id*) del medicamento. Por ejemplo, si el *id* es 7, el programa devuelve la posición correspondiente del vector.

## 2.4 Señales de Entrada y Salida

### Señales de Salida

- **Presionar:** Activa el presionador para abrir o cerrar (1 para presionar, 0 para soltar). Conectado al componente inteligente del presionador.
- **Coger:** Activa o desactiva la pinza para cerrar (1) o abrir (0). Conectado al componente inteligente de la pinza.
- **soltar\_pastilla:** Emite pulsos para dispensar una cantidad determinada de pastillas.
- **Ready:** Indica si el robot está preparado para recibir órdenes. Cuando el robot está en proceso, el valor es 0; si está libre, es 1.

### Señales de Entrada

- **medicina\_signal:** Identificador (*id*) del medicamento a servir.
- **dosis\_signal:** Cantidad del medicamento a servir:
  - Para jarabes:
    - \* 1: Dosis baja (10ml).
    - \* 2: Dosis media (50ml).
    - \* 3: Dosis alta (100ml).
  - Para pastillas: Cantidad de unidades a dispensar.
  - Las goteras siempre tienen la misma dosis.

## 2.5 Funciones Principales

### Funciones de Control General

- `reset_signals()`: Reinicia todas las señales de salida para evitar errores al encender o apagar el sistema.
- `go_home()`: Lleva al robot a la posición inicial o de seguridad.

### Funciones de Manipulación

- `coger_objeto(dnum objeto)`:
  - Configura la apertura de la pinza para un objeto específico.
  - Activa la señal "Coger" para cerrar la pinza.
- `soltar_objeto()`:
  - Desactiva la señal "Coger" para abrir la pinza.
- `presionar(dnum medicina)`:
  - Configura la presión necesaria para el medicamento (jarabe o gotera).
  - Activa la señal "Presionar".
- `Desactivar_Presionador()`:
  - Desactiva la señal "Presionar" para liberar el medicamento.

### Funciones de Posicionamiento

- `robtarget seleccionar_posicion_medicamento(num medicamento_id)`:
  - Devuelve la posición del medicamento correspondiente al id proporcionado.

### Funciones de Dispensación

- `dispensar_pastillas(num dosis)`:
  - Emite pulsos en la señal "soltar\_pastilla" tantas veces como indique la dosis.
- `string tipo_medicina(num id)`:
  - Devuelve el tipo de medicina según su id ("JARABE", "GOTERA", "PASTILLA").
- `bool dosis_correcta(num dosis)`:
  - Verifica que la dosis del jarabe sea válida (1, 2 o 3).

## Funciones de Comprobación

- `string tipo_medicina(num id):`
  - Devuelve el tipo de medicina según su id ("JARABE", "GOTERA", "PASTILLA").
- `bool dosis_correcta(num dosis):`
  - Verifica que la dosis del jarabe sea válida (1, 2 o 3).

## Programas Específicos

- `programa_pastillas(robtargt Pastillero, num dosis):`
  - Realiza las trayectorias y acciones necesarias para dispensar las pastillas.
- `programa_jarabe(robtargt Jarabe, robtargt Cuentagotas, dnum Cuentagotas_Pr`
  - Realiza las trayectorias y acciones necesarias para dispensar el jarabe.
- `programa_gotera(robtargt Gotera):`
  - Realiza las trayectorias y acciones necesarias para dispensar las gotas.
- `poner_vaso():`
  - Recoge un vaso del dispensador y lo coloca en la mesa.

## 2.6 Componentes Inteligentes

Los componentes inteligentes en RobotStudio son dispositivos simulados preconfigurados que emulan el comportamiento de herramientas o sistemas reales en una celda robótica, como pinzas, presionadores o cintas transportadoras. Estos componentes interactúan directamente con las señales del robot, lo que permite realizar acciones específicas, como abrir o cerrar una pinza o activar un presionador, sin necesidad de programar cada detalle desde cero. Ofrecen precisión, facilidad de uso y un entorno de simulación realista, optimizando el diseño y la validación de procesos industriales.

A continuación se explican los 5 módulos de componentes inteligentes usados.

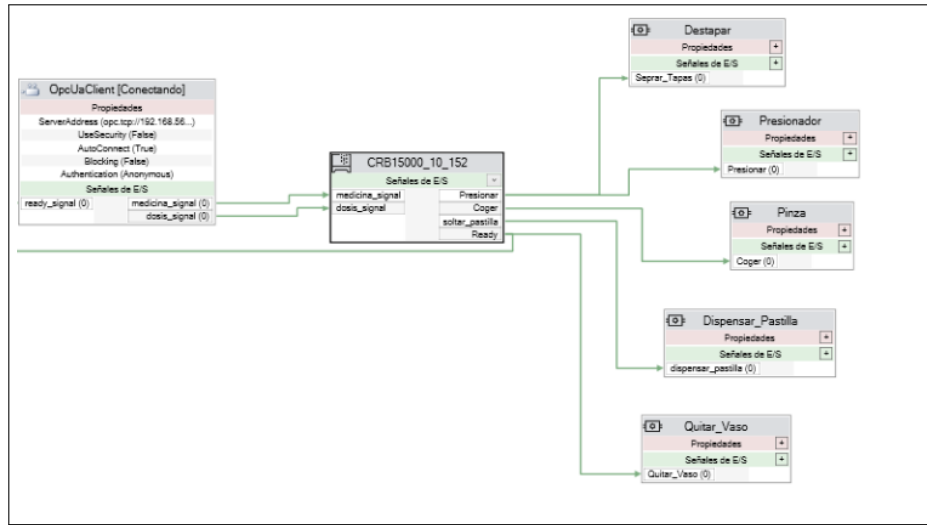


Figure 11: Componentes Inteligentes utilizados.

## Destapar

Inicialmente, las tapas de los medicamentos están unidas a las botellas para simular la realidad. Sin embargo, al servir la medicina, es necesario abrir el frasco. Con los componentes attacher y detacher, se pueden unir o separar las tapas de los medicamentos, controlado por la señal "Presionar". Cuando esta se activa, indica que hay un medicamento para presionar y posteriormente quitar la tapa. También incluye sensores para detectar la medicina que debe separarse.

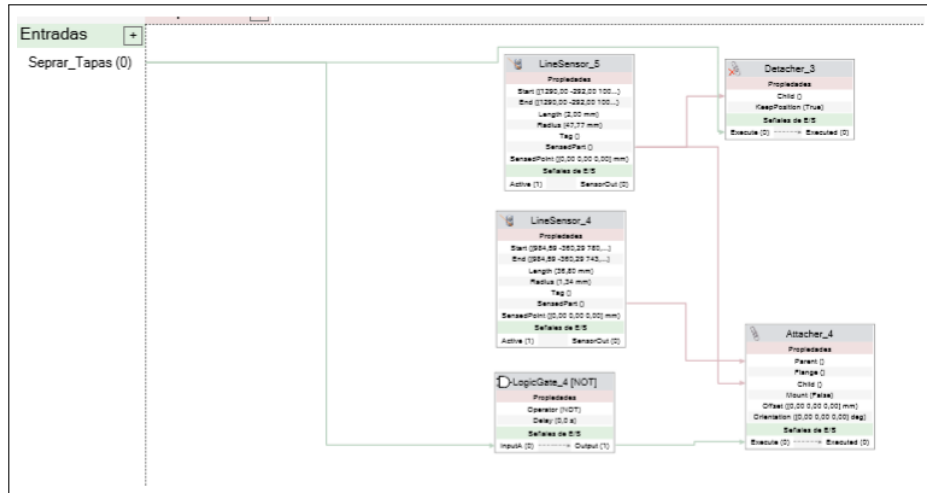


Figure 12: Destapar.

## Presionador

Utiliza el componente RapidVariable para acceder a la distancia que han de colocarse, almacenada en la variable "presión", y posicionar los presionadores con LinearMover. Cuando la señal "Presionar" está activa, se posicionan para ejercer fuerza; al estar en 0, un NOT gate activa otro movimiento para abrir el presionador.

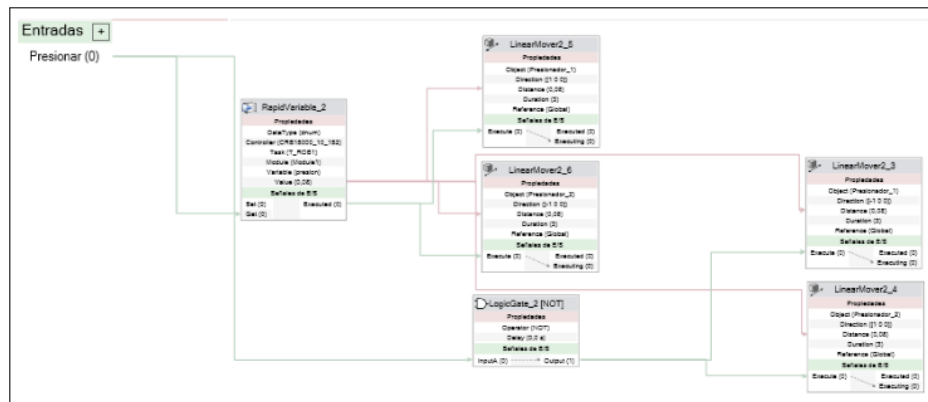


Figure 13: Presionador.

## Pinza

Controlada por la señal "Coger". Cuando está en 1, accede a la variable que define la apertura necesaria y activa el componente inteligente de la pinza. Cuando está en 0, la pinza se abre completamente. Incluye attacher y detach para simular el agarre y liberación de objetos. Un sensor lineal permite detectar los objetos a sujetar.

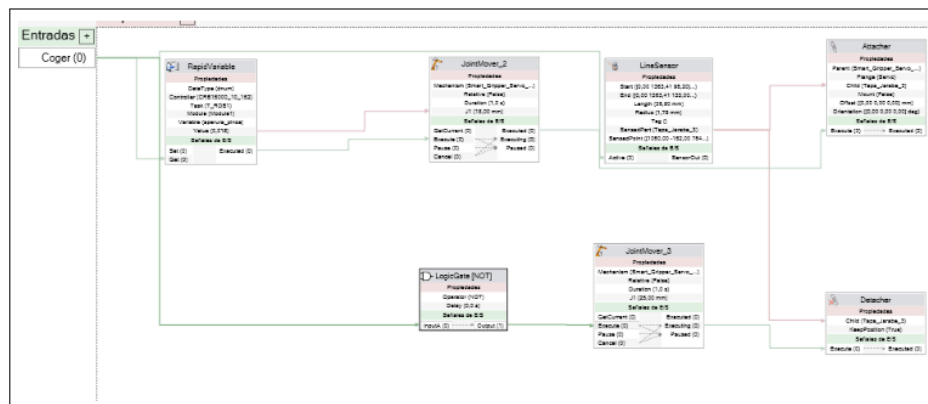


Figure 14: Pinza.

## Dispensar Pastilla

Controlado por la señal "soltar\_pastilla". Se utiliza el componente Source para crear una copia de la pastilla y MoveAlongCurve para simular su caída al vaso. Con PhysicsControl, la pastilla se vuelve dinámica para quedar en el fondo del vaso.

Recordar que la señal "soltar\_pastilla" emite pulsos, por lo que este proceso se realiza con cada pulso, simulando la dispersión de varias pastillas.

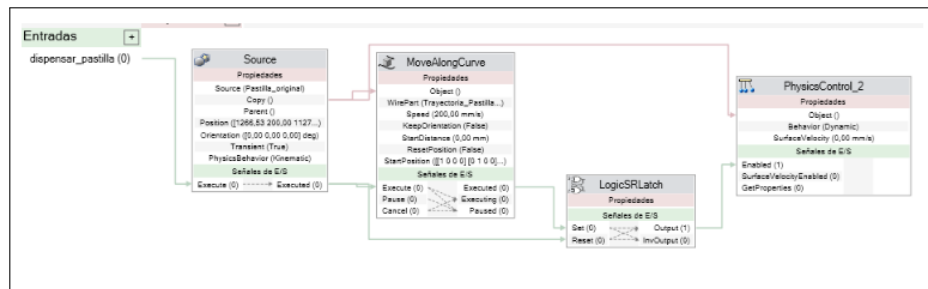


Figure 15: Dispensar Pastilla.

## Quitar Vaso

Controlado por la señal "Ready", que indica que el robot está listo para la siguiente orden. Simula que el anciano retira el vaso moviéndolo y haciéndolo desaparecer con el componente Sink.

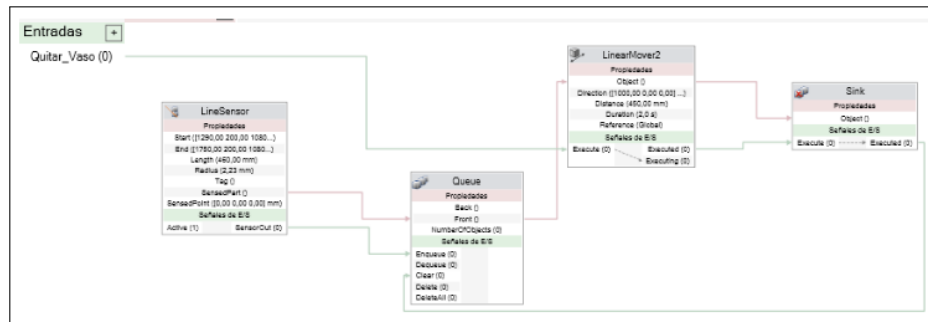


Figure 16: Quitar Vaso.

## OpcUaClient

Este componente conecta el robot a un servidor OPC UA para comunicarse con el programa Python de reconocimiento facial. Recibe las señales "medicina\_signal" y "dosis\_signal", que se envían al controlador del robot, y envía la señal "Ready".

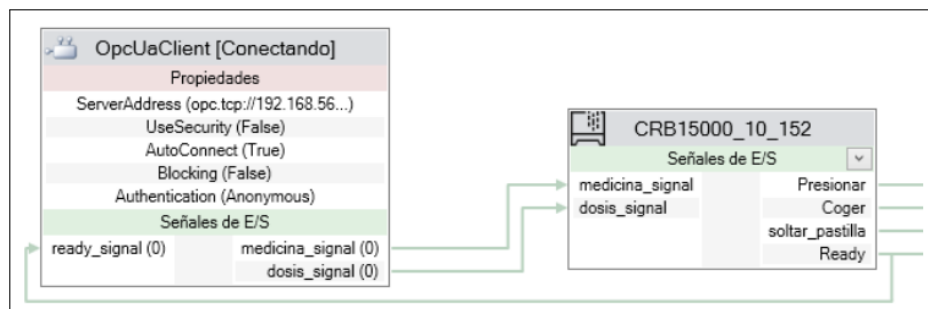


Figure 17: OpcUaClient.

## 2.7 Main: Flujo de programa

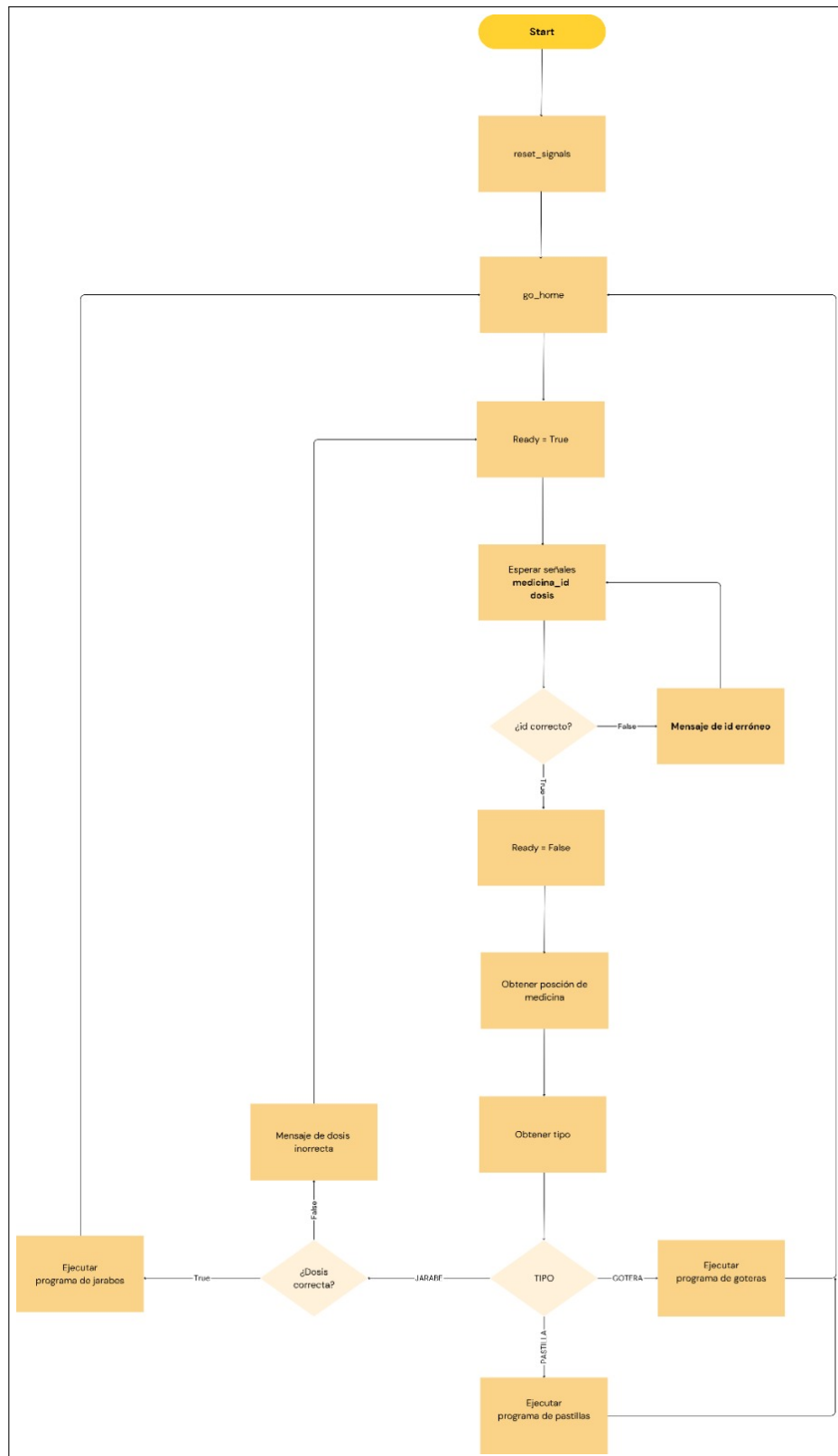


Figure 18: Flujo RobotStudio.

## 2.8 Comunicación con la Base de Datos

Se ha desarrollado un código en Python, denominado `opcua_server.py`, para la conexión entre la simulación en RobotStudio y el programa de reconocimiento facial. Este código contiene la clase `OpcuaServer`, que permite gestionar la comunicación mediante un servidor OPC UA. A continuación, se describen sus características principales:

### 1. Inicialización del Servidor

El inicializador de la clase `OpcuaServer` recibe como parámetro la dirección del servidor para establecer la conexión. Durante este proceso, se crean las tres variables principales:

- **medicina\_signal**: Identificador del medicamento.
- **dosis\_signal**: Cantidad de medicamento.
- **ready\_signal**: Indica si el robot está listo para recibir una nueva orden.

### 2. Métodos

- **start\_server()**: Inicia la conexión con el servidor OPC UA.
- **stop\_server()**: Detiene la conexión con el servidor.
- **is\_ready()**: Comprueba si el robot está ocupado o listo para recibir un nuevo medicamento.
- **enviar\_datos\_paciente(medicamento\_id, dosis)**: Envía los datos del medicamento y la dosis al robot a través de las variables definidas.

**Nota:** Este servidor actúa como intermediario entre el programa de reconocimiento facial y el controlador del robot, permitiendo la comunicación entre ambos.

## 3 Base de Datos

Para la gestión y administración de pacientes, medicamentos y recetas se ha utilizado una base de datos creada con PostgreSQL. La base sirve para la lectura de cualquier información necesaria y también es capaz de recibir modificaciones en los distintos registros de datos. La base de datos para este proyecto ha sido alojada en la nube a través de la plataforma Railway.com, la cual ofrece servicios de alojamiento para diversos tipos de bases de datos. Este enfoque resulta fundamental para garantizar que el proyecto pueda ejecutarse desde cualquier dispositivo, requiriendo únicamente una conexión a internet.





Figure 19: Tablas principales.

1. **Tabla pacientes:** Aquí se almacena toda la información de los distintos pacientes registrados en nuestro sistema. La variable clave de esta tabla es “id\_paciente” (las variables clave sirven para poder relacionar los datos entre las distintas tablas). También cuenta con las variables “nombre”, “edad” y “habitación”. Por último, cuenta con una variable de tipo foreign key que relaciona a un paciente con su receta.

id_paciente	nombre	edad	habitacion	id_receta
1	Miguel Ferrer	81	113	1
5	Badar el Asery	65	204	5
4	Pablo de Codina	80	234	4
2	Miguel Pascual	82	200	2
3	Eduardo Ugarte	74	100	3

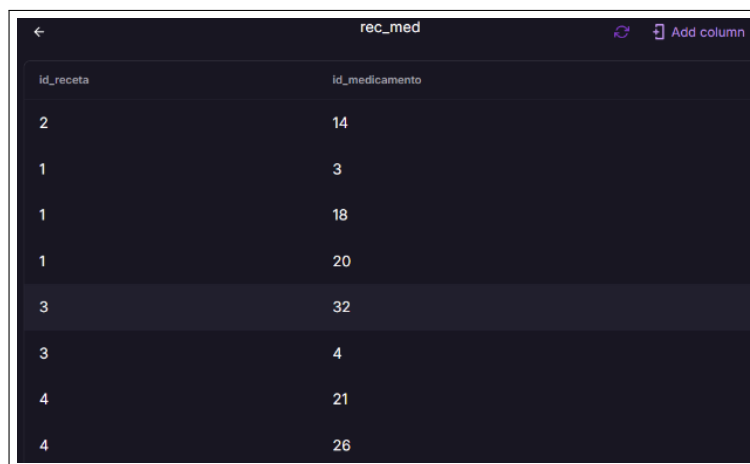
Figure 20: Tabla de Pacientes.

2. **Tabla recetas:** Esta tabla simplemente relaciona las diferentes recetas creadas, con los pacientes registrados. Su variable clave es “id\_receta” y cuenta con una variable foránea “id\_paciente”.

id_receta	id_paciente
1	1
2	2
3	3
4	4
5	5

Figure 21: Tabla de Recetas.

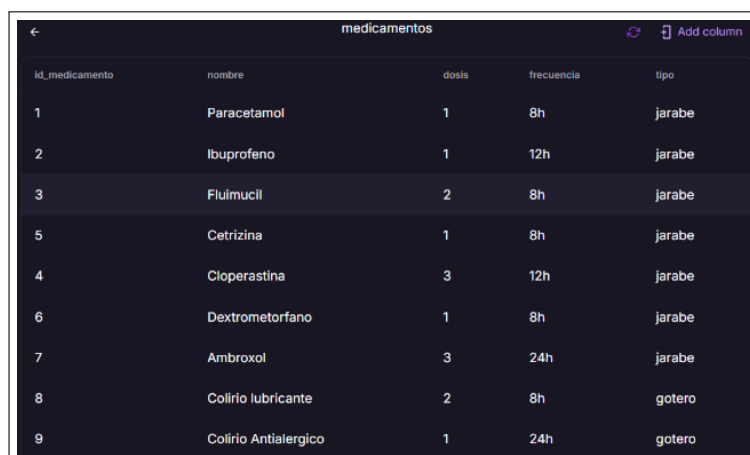
3. **Tabla `rec_med`:** En esta tabla se registran los medicamentos que hay en cada receta creada. Cuenta con las variables “`id_receta`” y “`id_medimento`”. Ambas son variables foráneas que permiten la relación entre medicamento y receta. Con esta tabla conseguimos almacenar en una receta los medicamentos necesarios.



id_receta	id_medimento
2	14
1	3
1	18
1	20
3	32
3	4
4	21
4	26

Figure 22: Tabla `rec_med`.

4. **Tabla de medicamentos:** Por último, en esta tabla se almacenan todos los registros de cada uno de los medicamentos que se encuentran en la mesa del robot. Su variable clave es “`id_medimento`”, también cuenta con las columnas “`nombre`”, “`dosis`”, “`frecuencia`” y “`tipo`”. Para nuestro proyecto hemos identificado tres tipos distintos de medicamentos que son: jarabe, gotero y pastilla; que un medicamento sea de un tipo en concreto implica que el robot diferenciará entre los distintos medicamentos con mismo nombre pero distinta forma de medicación y también cambiará el significado del valor de la variable dosis. La variable dosis reflejará el número de pastillas en caso de que el tipo sea “pastilla”, el tamaño del utensilio que suministra el jarabe para tipo “jarabe” o las gotas en medicamentos de tipo “gotero”



id_medimento	nombre	dosis	frecuencia	tipo
1	Paracetamol	1	8h	jarabe
2	Ibuprofeno	1	12h	jarabe
3	Fluimucil	2	8h	jarabe
5	Cetirizina	1	8h	jarabe
4	Cloperastina	3	12h	jarabe
6	Dextrometorfano	1	8h	jarabe
7	Ambroxol	3	24h	jarabe
8	Colirio lubricante	2	8h	gotero
9	Colirio Antialergico	1	24h	gotero

Figure 23: Tabla de Medicamentos.

## 4 Control por Visión

Debido a la necesidad de automatizar el proceso de dispensar pastillas a los pacientes. Se necesitaba, en primer lugar, saber qué paciente era el que necesitaba qué medicación, por ello se ha desarrollado un algoritmo de detección facial para permitir al programa saber cuál es la persona a medicar. De esta manera se puede controlar el fármaco y la dosis suministrada, además de evitar volver a medicar a aquellos pacientes que ya hubieran recibido lo que le correspondía.

### 4.1 Experimentación

En el desarrollo de este proyecto se tuvieron en cuenta distintas posibilidades de tecnologías a utilizar.

#### Deep Learning con TensorFlow

En primer lugar se pensó en usar modelos preentrenados implementados en frameworks como TensorFlow para extraer las características faciales para su posterior uso en el reconocimiento facial. La principal desventaja que hizo que no siguiéramos desarrollando esta aplicación con este método fue el alto coste computacional, ya que estos métodos son muy demandantes. Además de que el ordenador no solo debería estar corriendo la parte de detección, sino que también iba a estar ejercitando la parte de simulación con robotStudio y la parte de comunicación verbal con la máquina.

#### Métodos tradicionales de visión por computador

La siguiente idea que se tuvo fué usar algoritmos basados en características como Local Binary Patterns Histograms o Eigenfaces, ya que el principal problema que tuvimos con TensorFlow en principio iba a estar resuelto, debido a que estos algoritmos son muy ligeros y no requieren de muchos recursos. Sin embargo, durante su implementación y pruebas, se observó que el funcionamiento no era consistente ni fiable, ya que no era capaz de detectar correctamente a más de dos personas distintas sin fallar.

#### Dlib con face\_recognition

Dlib es una biblioteca de C++ enfocada en visión por computadora y aprendizaje automático, famosa por su detector de rostros y su modelo para extraer características faciales. face\_recognition, un proyecto en Python, se basa en Dlib para simplificar la detección y comparación de rostros.

El proceso comienza detectando el rostro (usando métodos como HOG o redes neuronales), luego se identifican puntos de referencia (ojos, nariz, boca) para alinear la cara y, finalmente, se genera un vector de características (embedding). Estos vectores permiten comparar rostros calculando su distancia: mientras más pequeña, más similitud. Además, se pueden entrenar clasificadores para identificar a múltiples

personas. Esta combinación de Dlib y face\_recognition ofrece una forma potente y accesible para implementar sistemas de reconocimiento y autenticación facial.

Finalmente se optó por utilizar face\_recognition, que tras una correcta calibración de parámetros fue el algoritmo utilizado en la aplicación debido a su balance entre el correcto funcionamiento y los recursos computacionales que requiere.

## 4.2 Implementación

### Carga de caras conocidas

En primer lugar, cada paciente o persona a detectar como enfermeros o personal de asistencia, tiene una carpeta con su nombre y fotos suyas para su reconocimiento. Para la detección de características de cada persona se ha usado un algoritmo que sigue estos pasos:

```
for name in os.listdir(KNOWN_FACES_DIR):
    for filename in os.listdir(f"{KNOWN_FACES_DIR}/{name}"):
        file_path = f"{KNOWN_FACES_DIR}/{name}/{filename}"
        print(f"Procesando {filename}...")

        # Cargar la imagen
        image = face_recognition.load_image_file(file_path)

        # Intentar codificar la cara
        encodings = face_recognition.face_encodings(image)
        if len(encodings) > 0:
            encoding = encodings[0]
            known_faces.append(encoding)
            known_names.append(name)
        else:
            print(f"No se detectó ninguna cara en {filename}. Asegúrate de que la imagen contiene una cara visible.")
```

Figure 24: Carga de Caras conocidas.

1. Se leen las imágenes almacenadas en las carpetas nombradas anteriormente que tienen el identificador de cada persona.
2. Posteriormente se usa face\_recognition para extraer las características o encodings de la cara detectada en la imagen.
3. Una vez ha sido detectada y extraída la información de las caras, se guardan los encodings junto con el nombre de la persona para ser usada posteriormente en las comparaciones pertinentes.

Se ha diseñado de esta manera para permitir añadir y quitar pacientes de una manera sencilla creando o eliminando las carpetas con las imágenes de cada persona.

## Proceso de reconocimiento

La aplicación usa este código para el proceso de reconocimiento:

```
video = cv2.VideoCapture(0)

while True:
    # Leer un frame de la cámara
    ret, frame = video.read()
    if not ret:
        break

    # Reducir tamaño del frame para procesarlo más rápido
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
    rgb_small_frame = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)

    # Encontrar todas las caras y codificarlas
    face_locations = face_recognition.face_locations(rgb_small_frame, model=MODEL)
    face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

    for face_encoding, face_location in zip(face_encodings, face_locations):
        # Comparar con caras conocidas
        matches = face_recognition.compare_faces(known_faces, face_encoding, TOLERANCE)
        name = "Desconocido"

        if True in matches:
            match_index = matches.index(True)
            name = known_names[match_index]
```

Figure 25: Reconocimiento.

Este algoritmo sigue estos pasos:

### Captura de cámara en tiempo real

Para ello se utiliza la librería de opencv, con la que se obtiene la imagen de la cámara en tiempo real. Para acelerar el procesamiento, se redimensiona la imagen con `resize`, consiguiendo una imagen de salida mucho más fluida.

### Detección

Una vez se tiene la imagen de la cámara lista y redimensionada en `rgb_small_frame`, se utiliza `face_recognition` para dos propósitos, en primer lugar guardar la localización de donde se encuentra la cara detectada en la imagen, y en segundo lugar para obtener los encodings que posteriormente se compararán con los de los pacientes.

### Comparación

Obtenidos ya los encodings, se pasa a la parte de comparación, en la que se compara usando un método de `face_recognition` la cara detectada en la imagen de la cámara con las que han sido analizadas y guardadas anteriormente en la fase de carga de caras conocidas.

Si esta función no encuentra otra cara suficientemente parecida (con la tolerancia requerida en `TOLERANCE`), califica a la persona como desconocida, y por el contrario si ha encontrado `matches` suficientes, le pone el nombre de la carpeta en la que se encuentra la imagen, que es el nombre de la persona.

## Persona detectada

Una vez se tiene el nombre de la persona detectada este se guarda en un .txt llamado persona.txt para el uso de este en las distintas partes de la aplicación y por tanto tener una comunicación sencilla y en tiempo real con las demás partes.

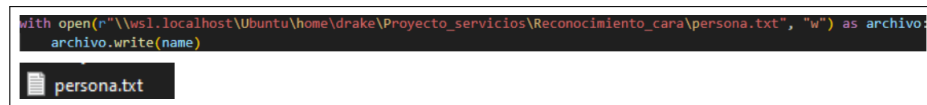


Figure 26: Detección.

## Visualización

Por último se visualiza la captura de la cámara junto al nombre, y recuadro de localización de la cara de la persona.

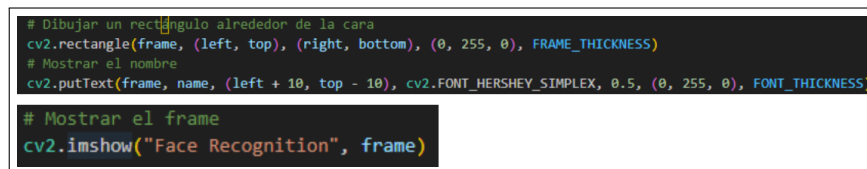


Figure 27: Visualización.

## 4.3 Configuración

A la hora de parametrizar y ajustar parámetros en el código, estos son algunos de los parámetros utilizados y el porqué de sus valores.

### TOLERANCE

Es el valor utilizado en la parte de comparación de caras, y delimita cuando se considera una coincidencia entre la cara detectada y las guardadas en la fase de carga de caras conocidas.

Se ha usado finalmente un valor de 0.4, ya que con valores altos, había veces que fallaba en la detección correcta de la persona y el nombre de la persona oscilaba entre dos o tres nombres distintos. Por el contrario, cuando se usaban valores demasiado bajos, el algoritmo no era capaz de detectar a las personas y salían mayormente como desconocidas.

### MODEL

Esta variable responde al modelo usado para la detección de caras, hoy es el más rápido mientras que cnn se supone que es más preciso pero requiere de GPU.

Se optó por usar hog ya que cnn no funcionaba correctamente además de que la imagen de salida era mucho menos fluida incluso cuando solo se estaba ejecutando esta parte de la aplicación.

## **Variables adicionales**

Además de estas variables se usaron otras como `KNOWN_FACES_DIR` o `OUTPUT_FILE` que representan respectivamente el directorio donde se almacenan las imágenes de las personas conocidas, y la ruta donde se guarda el nombre de la persona detectada en `persona.txt`.

## **4.4 Funcionamiento en la aplicación**

### **Identificación Automatizada**

El sistema permite identificar a los residentes de manera rápida al acercarse a la cámara. Esto minimiza errores humanos y garantiza que cada residente sea reconocido correctamente.

### **Asistencia en Medicación**

Al vincularse con la base de datos a través de `persona.txt`, permite que la interfaz muestre automáticamente el plan de medicación correspondiente al paciente identificado.

### **Registro de Administración**

Por último, la información de identificación es registrada en tiempo real, proporcionando un historial detallado de qué medicación fue administrada y a quién y evitar de nuevo errores humanos.

## **5 Agente conversacional**

### **5.1 Detección de habla**

Silero Voice Activity Detector (VAD)

Silero-VAD es un modelo preentrenado diseñado para identificar cuando hay habla humana en un flujo de audio, siendo capaz de ignorar ruidos, silencios y otros sonidos. Este detector es pequeño, rápido y muy preciso, lo que lo convierte en una solución ideal para aplicaciones en tiempo real como la planteada.

El modelo se basa en redes neuronales profundas y analiza fragmentos de audio en torno a 30 milisegundos cada uno. El modelo analiza cada fragmento y determina si contiene habla humana o no, proporcionando un nivel de confianza en cada detección. Esto permite a Nursebot almacenar la información necesaria para la posterior transcripción, almacenando el sonido captado en un audio desde el momento en el que se detecta habla humana hasta que se detecta un cierto tiempo de silencio, considerando como finalizado el turno de escucha.

El modelo Silero-VAD ha sido el elegido para desempeñar esta tarea, tras su análisis comparativo con otros modelos enfocados en la misma tarea, concluyendo que este ofrece una gran mejora en rendimiento respecto otras alternativas como WebRTC, librosa o SpeechPy. El modelo utilizado ofrece tanto una mayor precisión como capacidad para trabajar en entornos con ruido y menos falsos positivos que otros modelos. Además, su tamaño inferior a 1 MB y su integración con PyTorch facilitan mucho su implementación.

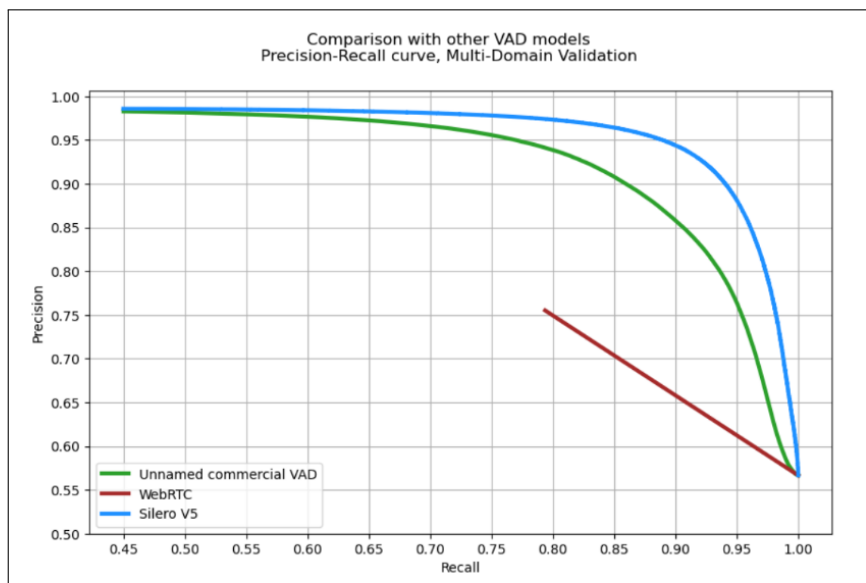


Figure 28: Comportamiento de Silero-VAD.

## 5.2 STT (Speech-to-text)

Para la transcripción del audio creado en el proceso de escucha de Nursebot intervienen dos modelos de inteligencia artificial basados en Whisper, desarrollados específicamente para convertir audio a texto. Cada uno de ellos tiene un papel fundamental en el proceso de detección y transcripción del habla.

- **Whisper**

El primer modelo, Whisper, es el modelo original desarrollado por OpenAI. Este modelo, como se ha mencionado anteriormente, es presentado para la transcripción y traducción de audio en múltiples idiomas.

El modelo está entrenado con una amplia variedad de datos lo que hace se diseño robusto frente a diferentes contextos, como ruidos de fondo, acentos u otras variaciones ambiguas durante el habla. Whisper utiliza una arquitectura basada en Transformers y procesa el audio como una entrada continua que convierte en texto mediante técnicas avanzadas de atención.



La implementación de Whisper en el sistema no viene dada por su capacidad de transcripción, si no por su característica de detección de habla en tiempos muy cortos. Al emplear Whisper para la detección del idioma, se asegura la correcta identificación de este dato en tiempo real.

- **WhisperS2T**

Variante optimizada del Whisper original de OpenAI que puede utilizar tres tipos de backends, el de Openai, CTranslate2 o TensorRT-LLM. WhisperS2T ha sido diseñado para reducir los requisitos computacionales sin sacrificar la precisión que ofrece el modelo original.

El backend escogido ha sido CTranslate2, una biblioteca diseñada específicamente para acelerar modelos de inteligencia artificial basados en Transformers. CTranslate2 está diseñado para aprovechar al máximo tanto las capacidades de CPU y GPU implicadas, permitiendo que WhisperS2T funcione con un consumo de recursos significativamente menor en comparación con el modelo original de OpenAI.

La variante WhisperS2T es más eficiente y rápida que el modelo original, y sumado al uso del backend mencionado convierten el sistema de transcripción en una elección ideal para entornos donde la velocidad y el rendimiento son críticos, como en el desarrollado.

En el proceso de transcripción del audio, Whisper se enfoca sólo en la detección del idioma y será WhisperS2T, que no tiene la característica de detectar el idioma, el que transcribe el audio almacenado anteriormente, que aporta una mayor eficiencia y un resultado prácticamente perfecto en multitud de situaciones.

## 5.3 RAG - LLM

### 5.3.1 FlowiseAI

FlowiseAI es una plataforma o framework que facilita la creación de flujos de trabajo (workflows) y agentes conversacionales. Permite “conectar” diferentes nodos o módulos que realizan tareas específicas (por ejemplo, recibir el prompt del usuario, consultar un modelo de lenguaje, etc.). Su interfaz visual de “arrastrar y soltar” hace que el diseño de flujos sea más intuitivo y modular.

- **Nodos Personalizables:** Cada “nodo” en FlowiseAI puede configurarse para un fin específico, como conectarse a Ollama, ejecutar lógica de negocio, etc.

- **Conectores y Hooks:** FlowiseAI ofrece conectores que facilitan la integración con API externas (por ejemplo, la API de búsqueda en Internet) y “hooks” para inyectar funciones personalizadas en diferentes puntos del flujo.
- **Simplicidad de Desarrollo:** Incluso sin amplios conocimientos de programación, FlowiseAI permite construir diagramas lógicos de lo que se quiere hacer paso a paso.

## Ventajas

- **Modularidad:** Agregar o cambiar componentes (ej. distintos LLM o distintas fuentes de datos) es sencillo.
- **Escalabilidad:** Puede emplearse en proyectos pequeños y luego crecer para proyectos más grandes sin rehacer toda la infraestructura.
- **Visual y Documentado:** Resulta más claro para equipos grandes o multidisciplinarios (donde no todos programan) entender y modificar los flujos.

## Integración con el Proyecto

FlowiseAI es el “cerebro” que recibe la solicitud del usuario por la API local, decide a qué módulo (RAG, LLM o Búsqueda en Internet) se debe derivar la petición y elabora una respuesta adecuada al prompt recibido por el usuario. Posteriormente, la respuesta puede pasar al módulo de traducción si es necesario y luego al módulo de texto a voz.

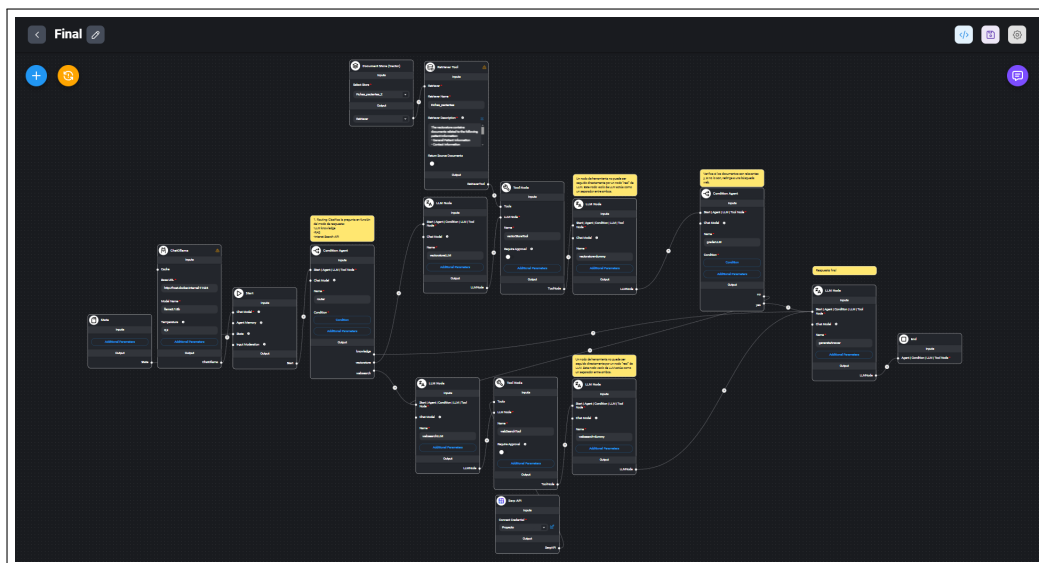


Figure 29: Flujo del agente en Flowise.

### 5.3.2 Ollama

Ollama es la plataforma que se utiliza para albergar y servir los modelos de lenguaje (LLMs). Actúa como una capa intermedia entre el modelo en sí (por ejemplo, llama3.1 8B) y las aplicaciones que hacen solicitudes de inferencia.

- **Gestión de Modelos:** Permite cargar, descargar y administrar diferentes versiones de modelos de lenguaje.
- **Endpoint Unificado:** Ollama ofrece un endpoint HTTP/REST que facilita la integración con otros servicios o plataformas (como FlowiseAI).
- **Optimización de Recursos:** Está diseñado para hacer un uso eficiente de la memoria y de las GPU/CPU, facilitando la escalabilidad y la implementación en entornos de producción.

### Ventajas

- **Centralización:** Se gestiona en un solo lugar la provisión de modelos LLM, lo que facilita actualizaciones o cambios en el modelo sin impactar directamente en la app cliente.
- **Facilidad de Integración:** La API de Ollama permite un rápido “plug and play” con distintas aplicaciones y flujos de trabajo.

### Desafíos potenciales

- **Requerimientos de Hardware:** Manejar modelos grandes (como 8B parámetros) exige un hardware robusto (memoria RAM, GPU, etc.).
- **Escalabilidad:** Si se manejan muchas solicitudes simultáneas, puede ser necesario balancear carga en varios servidores Ollama, lo cual añade complejidad al proyecto.

### Integración con el Proyecto

En FlowiseAI se emplean distintos nodos que interactúan con el modelo de texto de Ollama que clasifica la petición del usuario y decide que se requiere una respuesta del LLM o si por el contrario requiere hacer uso del RAG o de un nodo que hace búsqueda en internet. Si se hace uso del conocimiento del LLM simplemente se genera la respuesta, sin embargo, cuando se hace uso del RAG o de la búsqueda en internet lo obtenido en ambos procesos pasa por un nodo de LLM que condensa y expone la información de forma adecuada.

## 5.4 Modelo de LLM (Llama3.1 8B)

Llama3.1 8B es un modelo de lenguaje de la familia Llama que cuenta con 8 mil millones de parámetros. Este modelo se especializa en comprender y generar texto en múltiples idiomas, así como en seguir instrucciones de manera coherente.

- **Tamaño Intermedio:** Con 8B parámetros, se ubica en un rango medio para usos locales, ofreciendo un balance entre capacidad de generación y requerimientos computacionales.
- **Entrenamiento Multilingüe:** Permite procesar y responder en diferentes lenguas, sujeto a la disponibilidad de datos en su entrenamiento.
- **Rendimiento en Tareas Generales:** Es apto para consultas de cultura general, redacción de textos, corrección gramatical, traducción básica y otras tareas de NLP.

### Ventajas

- **Capacidad de Comprensión:** Entiende contextos extensos y puede retener la información del hilo conversacional, siempre que la ventana de contexto lo permita.
- **Flexibilidad:** Puede ajustarse (fine-tuning) para dominios específicos o personalizarse con prompts instructivos.

### Desafíos Potenciales

- **Alcance del Conocimiento:** Su base de datos puede estar desactualizada en cuanto a sucesos muy recientes.
- **Sesgos y Alucinaciones:** Puede generar respuestas con datos incorrectos o sesgados.

### Integración con el Proyecto

Almacenado y servido por Ollama, FlowiseAI hace llamadas al endpoint de Ollama, quien a su vez ejecuta inferencias de llama3.1 8B para responder las preguntas.

Este LLM fue escogido debido a que debido a su tamaño respondía mejor a esos prompts de sistema y cumplía adecuadamente con el rol que se le establecía. Uno de los casos en que no funcionaba en otros era debido a que cuando trataba información proveniente del RAG, al ser en su mayoría datos privados sobre personas respondía que no podía dar ese tipo de información debido a las limitaciones en materia de censura del mismo. Sin embargo, este modelo al ser "más inteligente" podía entender que se trataba de un RAG y sí proporcionaban la información tratada.

## 5.5 RAG (Retrieval-Augmented Generation)

### ¿Qué es un RAG?

Un RAG (Recuperación Aumentada de Generación, por sus siglas en inglés: Retrieval-Augmented Generation) es un enfoque que combina un modelo de lenguaje (LLM) con la recuperación de información específica desde una base de datos o conjunto de documentos. Normalmente, un LLM por sí solo tiene el conocimiento que obtuvo durante su entrenamiento. Sin embargo, existen casos en los que se necesita información adicional que no se encontraba en el corpus de entrenamiento (por ejemplo, datos privados, información muy reciente o específica, etc.). En esos casos, se añade un componente de recuperación de información (generalmente basado en embeddings y búsqueda semántica) que localiza los textos relevantes en una base de datos o repositorio y los pasa al LLM como contexto. Así, el LLM genera su respuesta final “aumentada” con la información recuperada.

### Uso de embeddings (nomic-embed-text) y vector stores

Para poder implementar un sistema RAG, se necesita un esquema que permita representar los textos en un espacio vectorial. De esta forma, es posible realizar búsquedas semánticas o similares, en lugar de buscar únicamente palabras clave.

- **Embedding:** En este proyecto, se ha empleado nomic-embed-text, que convierte el texto en vectores numéricos. Cada vector captura la semántica del texto de forma que dos textos con significados parecidos tendrán vectores similares.
- **Vector store:** FlowiseAI permite crear un almacén de vectores (vector store) con el embedding elegido. Así, cuando se hace una búsqueda de un texto (o una pregunta específica), se convierte la pregunta en un vector y se comparan las distancias o similitudes con todos los vectores almacenados.

### Descripción y Rol Principal

El RAG combina un modelo de lenguaje (LLM) con la recuperación de información específica desde una base de datos o conjunto de documentos. Esto permite al LLM acceder a datos que no se encontraban en su corpus de entrenamiento y enriquecer sus respuestas con información actualizada o específica.

### Ventajas

- **Información Actualizada o Privada:** Acceso a datos que no estaban en el entrenamiento original del LLM.
- **Reducción de Alucinaciones:** Proveer textos concretos disminuye la probabilidad de información inventada.

## Desafíos Potenciales

- **Calidad de la Base de Datos:** Documentos desactualizados o incorrectos pueden afectar la respuesta.
- **Costo Computacional:** El cálculo de similitudes en grandes bases puede requerir hardware especializado.

## Integración con el Proyecto

Cuando se necesita información específica, el RAG extrae los textos relevantes, los pasa al LLM para generar respuestas más precisas.

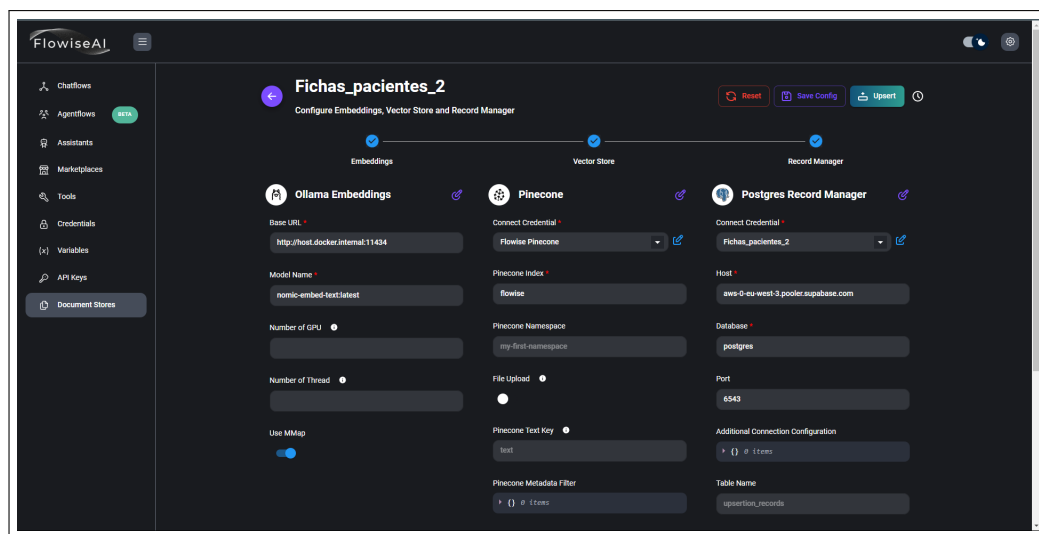


Figure 30: Document Store en Flowise.

Para crear el vector a partir del embedding comentado antes se usa la página Pinecone y como record manager se usa una base de datos PostgreSQL alojada en el servicio Supabase. El record manager se emplea para que cuando se realizan cambios o se añadan nuevos archivos al document store se sobrescriban los cambios, se actualice la información y se borren las anteriores versiones para que no se acumulen innecesariamente.

## 5.6 Traducción

Se ha empleado la biblioteca de python `googletrans` diseñada para facilitar la traducción de texto entre múltiples idiomas utilizando la API de Google Translate, cuyo propósito principal es proporcionar una manera eficiente y sencilla de traducir texto.

### Características principales:

- **Compatibilidad con múltiples idiomas:** `googletrans` soporta más de 100 idiomas, lo que permite traducir texto desde y hacia casi cualquier idioma.

- **Detección automática de idioma:** Ofrece la capacidad de detectar automáticamente el idioma del texto de origen, eliminando la necesidad de especificar manualmente el idioma de entrada.
- **Traducciones rápidas y sencillas:** La biblioteca proporciona métodos simples para realizar traducciones, como el método `translate()`, que permite obtener traducciones con solo unas pocas líneas de código.

## 5.7 XTTS (Text-to-Speech)

XTTS es un sistema de conversión de texto a voz (TTS) que permite voces personalizadas en diferentes idiomas, con variedad de géneros y acentos. Es la última etapa del flujo antes de entregar la respuesta final al usuario, en formato de audio.

- **Personalización de Voces:** Voces naturales y adaptables a diferentes necesidades.
- **Soporte Multilingüe:** Capacidad de generar habla en múltiples idiomas.
- **Parámetros de Entonación y Ritmo:** Ajustes para mayor naturalidad.

## Ventajas

- **Accesibilidad:** Facilita la interacción para personas con dificultades de lectura o visión.
- **Experiencia de Usuario Mejorada:** Voces adaptadas favorecen la usabilidad en contextos diversos.

## Integración con el Proyecto

FlowiseAI envía la respuesta final al XTTS para la generación de audio, el cual se reproduce y vuelve al nodo de detección de habla, logrando un flujo conversacional completo.

## 5.8 Flujo de Control

El flujo de interacción principal del control por voz comienza con la ejecución de un archivo principal, `NurseBot_voice.py`. Este programa se encarga de coordinar las distintas partes del sistema, integrando módulos para grabar, procesar y responder a las interacciones por voz. Antes de ejecutarlo, es necesario poner en marcha el `servidor_TTS.py`, el servidor en local encargado del TTS (Text-to-Speech), que gestiona la conversión de texto a voz utilizando un modelo avanzado de síntesis de audio llamado XTTS. Este servidor proporciona un punto centralizado para transformar las respuestas generadas en un formato audible, lo que es esencial para la interacción fluida con los usuarios.

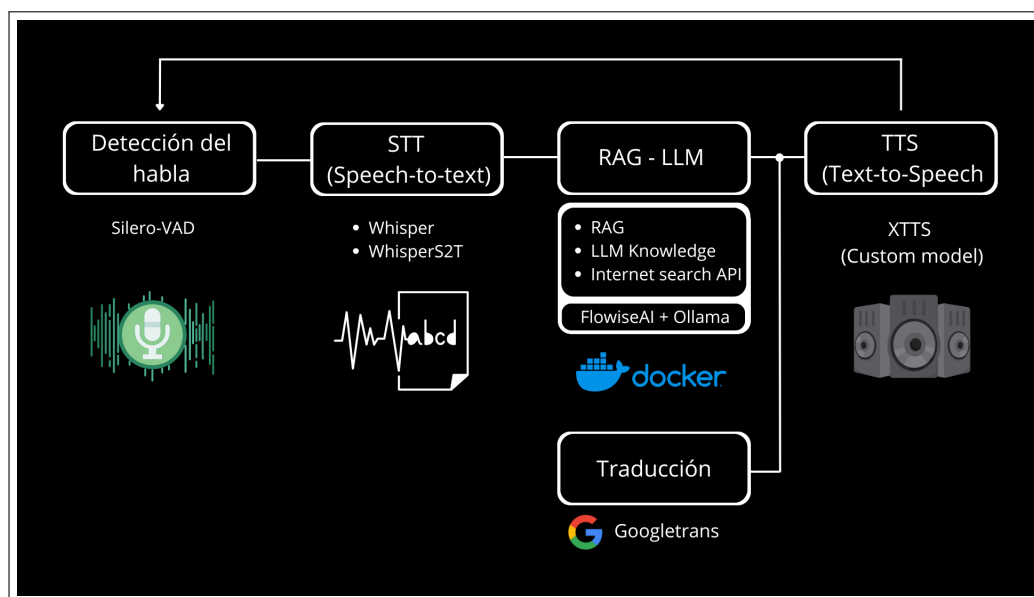


Figure 31: Flujo de Control.

## 1. Detección de habla

Para capturar el audio de las interacciones, se utiliza el módulo `silero.py`, que implementa el modelo Silero-VAD. Este modelo está optimizado para detectar la presencia de habla en tiempo real, ignorando ruidos de fondo o silencios prolongados.

El sistema permanece en un estado de escucha constante, analizando los datos del micrófono en pequeños fragmentos. Cuando detecta habla, empieza a grabar los datos en memoria. El proceso continúa hasta que se identifican 2 s de silencio consecutivos, momento en el cual el sistema detiene la grabación y guarda el audio capturado en un archivo llamado `audio.wav`. Este enfoque asegura que solo se procesen las partes relevantes de la interacción, reduciendo la carga computacional y mejorando la eficiencia del sistema.

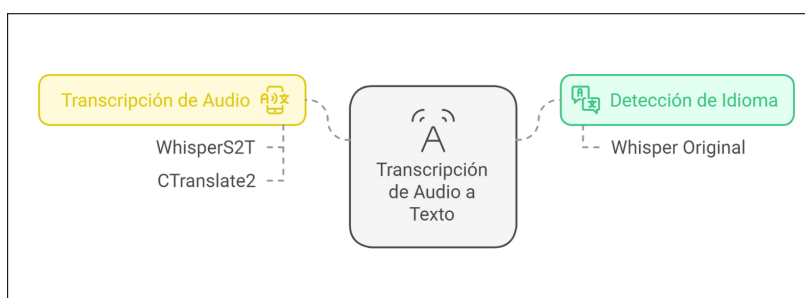


Figure 32: Proceso de detección de habla.



## Audio a Texto

El siguiente paso del flujo es convertir el archivo de audio en texto mediante el módulo `s2t.py`. Aquí se combinan dos modelos de la familia Whisper, cada uno desempeñando un rol importante.

El modelo original de Whisper, se utiliza para detectar el idioma del audio. Esta información es crucial, ya que permite adaptar el procesamiento posterior al idioma en el que se realiza la interacción. Después, el sistema emplea WhisperS2T, específicamente para transcribir al español, puesto que es el idioma usado internamente para el procesamiento de la información y la generación de la respuesta.

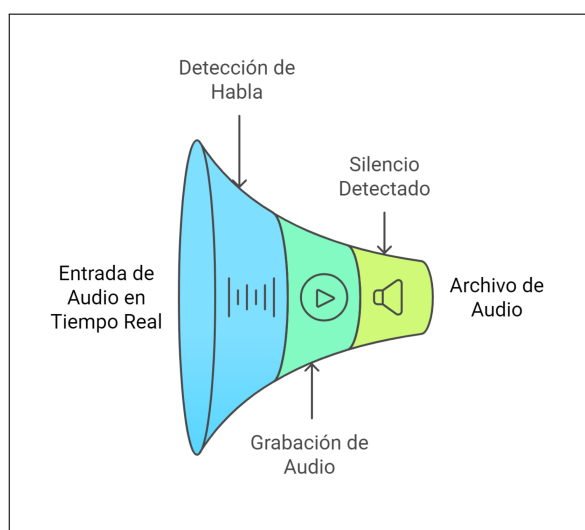


Figure 33: Audio a Texto.

## Generación de Respuesta

Una vez que el texto ha sido transcrito, el sistema lo envía al modelo de lenguaje (LLM) para generar una respuesta. NurseBot utiliza FlowiseAI, una herramienta no-code que facilita la implementación de pipelines complejos de razonamiento con capacidades de Retrieval-Augmented Generation (RAG). Este enfoque permite que el sistema acceda a bases de datos propias, proporcionando respuestas personalizadas y contextualmente relevantes, además de permitir hacer consultas a API's de buscadores en internet o usar el propio conocimiento del LLM.

La comunicación con el modelo de lenguaje se maneja mediante el módulo `api.py`, que realiza solicitudes HTTP al servidor de FlowiseAI. Este módulo se asegura de que las consultas se realicen de manera eficiente y segura, utilizando una clave de autorización para proteger las comunicaciones. La respuesta generada por el modelo se devuelve al programa principal para continuar con el flujo.

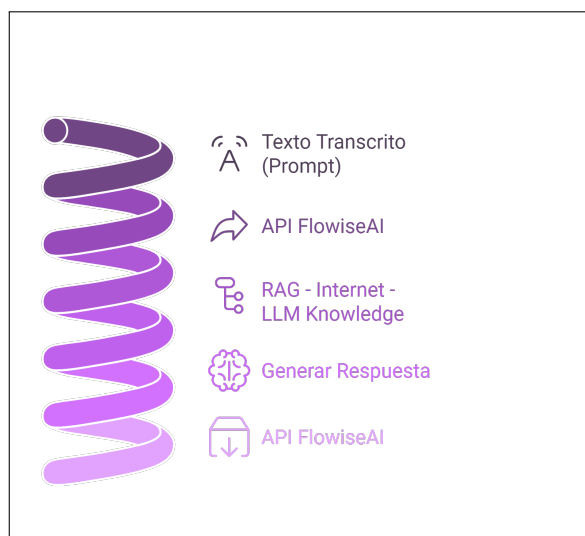


Figure 34: Generación de Respuesta.

### Traducción del texto

Una vez se ha generado la respuesta proveniente de la API de flowise, se determina si el idioma detectado en la consulta del usuario es diferente del español. En caso de que así sea se traduce la salida del LLM al idioma detectado haciendo uso de la lista de códigos ISO 639-1. Este estándar define códigos de dos letras usados para identificar los principales idiomas del mundo.

La traducción se hace empleando el módulo de `API_2.py` donde se encuentra una función especificada por la librería `googletrans` que realiza la traducción indicando el idioma de origen y el de salida.

### Texto a Audio

El texto obtenido como salida del LLM es transformado en un archivo de audio que contiene la narración con una voz natural compatible con múltiples idiomas. Esto se consigue cuando dentro del flujo principal del programa se hace una petición al servidor Flask montado (`servidor_TTS.py`) gracias al uso del módulo `API_2.py`. Este servidor local utiliza un modelo de TTS (Text-to-Speech) avanzado llamado XTTS para convertir texto a voz, creando una salida audible que puede ser fácilmente comprendida por los usuarios.

El sistema permite configurar características como el idioma, la voz seleccionada (distintos acentos disponibles), y otros parámetros de audio que son administrados desde el código en función del idioma y los requerimientos del usuario. El resultado es un archivo en formato `.wav`, que se reproduce automáticamente para que los usuarios puedan escuchar la respuesta del robot. Este paso cierra el ciclo de interacción, asegurando una experiencia fluida y eficiente.

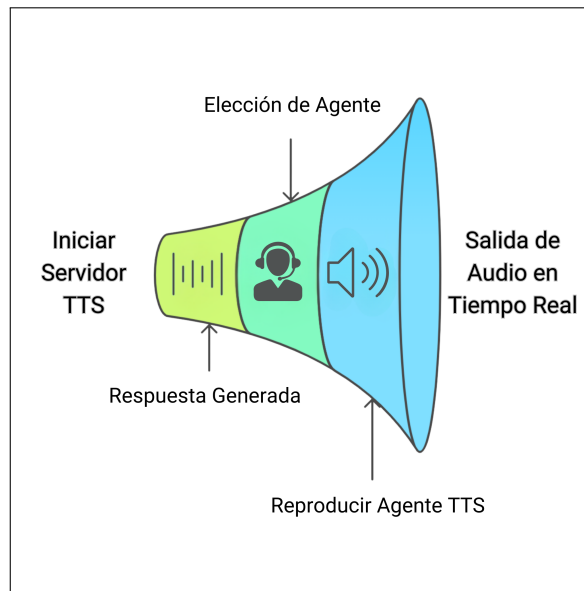


Figure 35: Texto a Audio.

### Ciclo Continuo

Una vez que el sistema está operativo, sigue este flujo de manera repetitiva. El programa principal permanece en estado de escucha, detecta nuevas entradas de voz, procesa el audio y responde en tiempo real. Esto asegura que NurseBot pueda manejar múltiples interacciones consecutivas sin interrupciones.

## 6 Interfaz

El sistema de monitoreo de pacientes está diseñado para gestionar información médica, integrar reconocimiento facial y facilitar la administración de medicamentos de forma eficiente. A continuación, se describen sus principales componentes y funcionalidades:

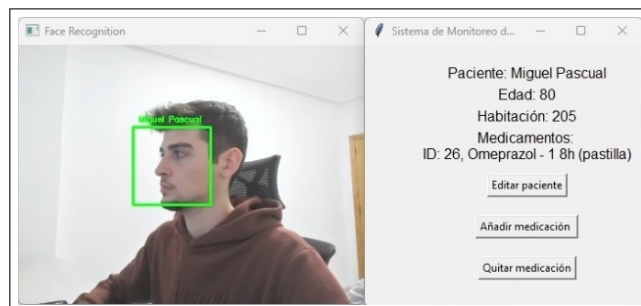


Figure 36: Interfaz global.

## 6.1 Reconocimiento Facial

El sistema utiliza OpenCV para capturar video desde la cámara, permitiendo identificar a los pacientes mediante un recuadro verde que muestra su nombre. Esta funcionalidad ha sido descrita en detalle en el apartado de reconocimiento facial, cuya finalidad es reconocer automáticamente al paciente y cargar su información almacenada en la base de datos.

## 6.2 Disposición de la interfaz

La interfaz está dividida en dos secciones principales: en el lado izquierdo, se muestra una ventana de vídeo en tiempo real que captura la imagen del paciente; en el lado derecho, se despliega un panel con la información del paciente, incluyendo nombre, edad y número de habitación. También se detalla una lista de medicamentos asignados al paciente, con información específica como ID, nombre, dosis, frecuencia y tipo. Además, se incluyen botones para editar los datos del paciente, añadir medicamentos nuevos o eliminar los existentes.

## 6.3 Gestión de Base de Datos

El sistema se conecta a la base de datos de PostgreSQL alojada en Railway mediante la biblioteca `psycopg2`. Entre las funciones implementadas se encuentran la visualización en tiempo real de medicamentos asignados a un paciente, la edición de datos como edad y habitación, y la gestión de medicamentos, ya sea añadiendo o eliminándolos de la receta.

## 6.4 Integración con Archivos

El nombre del paciente es leído desde un archivo de texto (`persona.txt`), lo que permite realizar consultas en la base de datos de manera automática para cargar y actualizar la información en la interfaz en función del paciente que se encuentra en la cámara.

## 6.5 Funcionalidades

El sistema de monitoreo de pacientes cuenta con las siguientes funcionalidades principales:

### Editar datos del paciente

Permite modificar la información básica del paciente, como su edad y número de habitación.

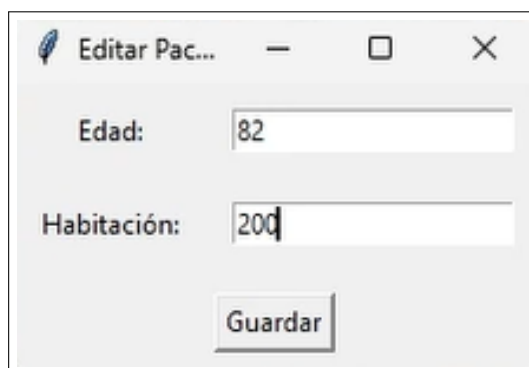
A screenshot of a software window titled "Editar Pac...". It contains two text input fields: "Edad:" with the value "82" and "Habitación:" with the value "200". Below these fields is a button labeled "Guardar". The window has standard OS window controls (minimize, maximize, close) in the top right corner.

Figure 37: Editar información del paciente.

Una vez ingresados los nuevos valores, estos se actualizan en la base de datos y se muestra un mensaje de confirmación:

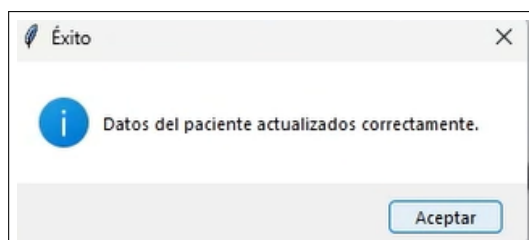
A screenshot of a message box titled "Éxito". It features a blue information icon on the left and the text "Datos del paciente actualizados correctamente." in the center. At the bottom right, there is a button labeled "Aceptar". The message box has a close button (X) in the top right corner.

Figure 38: Mensaje de éxito.

## Añadir medicación a la receta

Facilita la asignación de un nuevo medicamento a la receta del paciente. El usuario debe proporcionar el ID del medicamento, y si este es válido, se agrega a la base de datos con el mensaje de éxito.

A screenshot of a software window titled "A...". It contains a text input field labeled "ID:" with the value "3". Below the field is a button labeled "Guardar". The window has standard OS window controls (minimize, maximize, close) in the top right corner.

Figure 39: Añadir medicación.

## Quitar medicación de la receta

De la misma manera que añadir medicación, este metodo elimina una medicación de la receta del paciente mediante su ID.

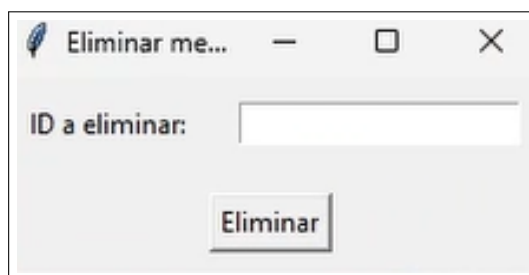


Figure 40: Eliminar medicación.

## 7 Vídeo de la aplicación

En este apartado se encuentra un enlace a un vídeo que hemos subido a YouTube donde se muestra el funcionamiento al completo de la aplicación de robótica de servicios.

NurseBot - Proyecto de robótica de servicios

## 8 Conclusiones

Tras más de un mes de trabajo intensivo, el proyecto NurseBot desarrollado ha permitido aplicar tecnologías de vanguardia y consolidar conocimientos en múltiples áreas técnicas. Entre las principales tecnologías utilizadas destacan:

- **Reconocimiento facial:** Mediante la biblioteca `face_recognition` de Python y algoritmos optimizados para entornos de recursos limitados, se logró un sistema eficiente para identificar pacientes, vinculando esta información con planes de medicación personalizados.
- **Procesamiento de voz:** El uso combinado de modelos como Whisper, Llama y XTTS o herramientas como Flowise y Ollama permitieron implementar un sistema robusto de interacción por voz, que mejora la experiencia del usuario y minimiza errores en la comunicación con el robot.
- **Gestión de datos:** Se desarrolló una base de datos en PostgreSQL alojada en la nube mediante Railway, que garantiza acceso remoto a información crítica y facilita la edición en tiempo real de los datos de los pacientes y medicamentos mediante una interfaz visual.
- **Simulación en RobotStudio:** Este elaborado entorno permitió integrar y probar todos los componentes del sistema de manera eficiente, optimizando su funcionamiento antes de una posible implementación física.

El trabajo en equipo ha sido fundamental para superar los desafíos técnicos y lograr los objetivos planteados. A pesar de los éxitos, el proyecto también ha abierto la puerta a posibles mejoras y ampliaciones futuras. Entre estas destacan:

- **Optimización de tiempos de respuesta:** Reducir los intervalos de espera en las interacciones por voz y en la ejecución de tareas complejas.
- **Implementación física:** Migrar el sistema desde el entorno de simulación a un robot físico, lo que implicaría evaluar factores como la movilidad del robot, la manipulación de objetos reales y la integración con hardware adicional.

En términos generales, NurseBot podría representar una contribución significativa al campo de la automatización médica y la robótica de servicios si se contasen con las mejoras mencionadas. El proyecto ha permitido al equipo no solo desarrollar soluciones innovadoras, sino también reflexionar sobre la viabilidad de estos sistemas en escenarios reales y cómo pueden mejorar la calidad de vida de las personas.

## 9 Anexos

- **Vídeo Demostración**
- **Guía de Usuario**
- **RobotStudio**
  - `opcua_server.py`
- **Base de Datos**
  - `main.py`
  - `persona.txt`
- **Control por Visión**
  - `orden_clean.py` (Nombrar las imágenes ubicadas en cada carpeta dentro de `know_faces` como `[nombre_carpeta]`. Además limpia los archivos que no sean imágenes de esas mismas carpetas)
  - `deteccion_facial.py`
  - `known_faces`
- **Control por Audio**
  - `silero.py`
  - `s2t.py`
  - `api.py`
  - `API_2.py`
  - `servidor_TTS.py`
  - `NurseBot_voice.py`
  - `Agente_conversacional.json`
- **Interfaz**
  - `interfaz.py`

## 10 Bibliografía

- **RobotStudio**

- Vídeo tutorial de cliente OPC UA

- **Base de Datos**

- Documentación oficial PostgreSQL
- Vídeo tutorial de Base de Datos en la nube

- **Control por Visión**

- Página Web "Reconocimiento Facial usando Face Recognition"
- Vídeo tutorial uso de librería face\_recognition en Tiempo Real

- **Control por Audio**

- Repositorio de github de Silero-VAD
- Repositorio de github de Whisper
- Vídeo explicativo del modelo Whisper
- Repositorio de github de WhisperS2T
- Vídeo explicativo del modelo WhisperS2T
- Alternativas vistas a WhisperS2T:
  - \* Vídeo explicativo del modelo Distl Whisper
  - \* Vídeo explicativo del modelo Insanely Fast Whisper
- Documentación Oficial FlowiseAI
- Vídeo tutorial base para crear flujo de FlowiseAI
- Video Explicativo de RAG en Flowise
- Repositorio github Ollama
- Documentación SerpApi
- Documentación Pinecone
- Documentación Supabase
- Video explicativo para desarrollar el TTS

- **Interfaz**

- Documentación oficial Tkinter
- Vídeo tutorial de uso de Tkinter más una base de datos
- Lista de reproducción con varios vídeos de interés