



GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE

Escuela de Ingeniería de Fuenlabrada

Curso académico 2024-2025

Trabajo Fin de Grado

Diseño de controladores gamificados
para rotot de rehabilitación

Tutor: Julio Salvador Lora Millán

Cotutor: Juan Alejandro Castaño Peña

Autor: Sofía Perales Díez



Este trabajo se distribuye bajo los términos de la licencia internacional CC BY-SA International License (Creative Commons Attribution-ShareAlike 4.0). Usted es libre de *(a) compartir*: copiar y redistribuir el material en cualquier medio o formato para cualquier propósito, incluso comercialmente; y *(b) adaptar*: remezclar, transformar y construir a partir del material para cualquier propósito, incluso comercialmente. La licenciante no puede revocar estas libertades en tanto usted siga los términos de la licencia:

- *Atribución.* Usted debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciante.
- *Compartir igual.* Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia del original.

No hay restricciones adicionales — No puede aplicar términos legales ni medidas tecnológicas que restrinjan legalmente a otras a hacer cualquier uso permitido por la licencia.

Agradecimientos

En primer lugar, me gustaría agradecer a la Universidad Rey Juan Carlos por brindarme la oportunidad de formar parte de esta institución. A todo el equipo docente que me ha formado académicamente, y en especial, a mis tutores Julio Salvador Lora y Juan Alejandro Castaño por la confianza y el apoyo depositado en mí para llevar a cabo este proyecto.

Agradecer a mis padres por todo lo que me han dado y por estar siempre a mi lado apoyándome e impulsándome a superarme y convertirme en la mejor versión de mí misma. A mi hermana, de la que aprendo todos los días y que me motiva a ser alguien en quien pueda inspirarse y confiar. Y a mi abuela, que es un referente para mí y me ha defendido siempre.

También agradecer a mis amigos, aquellos que se alegran por mis logros como si fuesen los suyos propios. Por la paciencia que tienen conmigo y por estar en las buenas y en las malas. A Nuria de la Fuente, por ser mi alma gemela desde el primer momento que nos conocimos. A Lucía Álvarez, por convertirse en mi hermana, doy gracias por habernos encontrado en la otra punta del mundo. A mis amigas del cole, quienes me han visto crecer y con las que he compartido la mayor parte de mi vida. Y a mis amigos de la universidad, gracias por las risas, los desayunos en la cafetería y sobre todo por el apoyo en momentos de estrés. Porque los ratitos con vosotros han hecho que el camino sea más fácil y ameno.

Y por último, gracias a Hispamast y todos sus empleados por darme la oportunidad de hacer las prácticas de empresa y formarme laboralmente. En especial a mis jefes y amigos Javier Trabalón y Alejandro Caballero, por cuidarme y tomarme como vuestra aprendiz. Gracias a vosotros he disfrutado de cada minuto durante las prácticas y ojalá sigamos compartiendo momentos juntos.

Todos vosotros me habeis hecho ser quien soy, gracias desde el corazón.

*A mi yo de pequeña;
quien soñaba con ser ingeniera, enhorabuena lo hemos conseguido*

Madrid, xx de xxxxxx de 2025

Sofía Perales Díez

Resumen

Abstract

Acrónimos

API *Interfaz de Programación de Aplicaciones*

GUI *Interfaz Gráfica de Usuario*

ROS *Sistema Operativo Robótico*

Índice general

1. Introducción	1
2. Objetivos	2
2.1. Descripción del problema	2
2.1.1. Objetivo general	2
2.1.2. Objetivos específicos	3
2.2. Requisitos	3
2.3. Competencias	4
2.3.1. Competencias generales	4
2.3.2. Competencias específicas	5
2.4. Metodología	5
2.5. Plan de trabajo	5
3. Plataforma de desarrollo	7
3.1. Sistema operativo	7
3.2. Entorno de desarrollo	7
3.2.1. Python	7
3.2.2. ROS 2	9
3.3. Componentes físicos	9
4. Diseño	11
4.1. Interfaz de registro de un paciente	11
4.2. Interfaz de control	13
4.3. Interfaz de visualización	16
4.4. Juego flappy	18
4.5. Esquema de nodos y topics	22
5. Conclusiones	24
5.1. Conclusiones	24
5.2. Corrector ortográfico	25

Índice de figuras

3.1. Actuador T-Series	10
4.1. Interfaz de registro de un paciente	12
4.2. Interfaz de control	15
4.3. Interfaz de configuración de los límites del brazo robótico	16
4.4. Interfaz de visualización del terapeuta	18
4.5. Nivel 2 del juego sin perturbación	21
4.6. Nivel 1 del juego con perturbación	21
4.7. Esquema de nodos y topics	23

Listado de códigos

4.1. Función para guardar los datos de un paciente	13
4.2. Encabezado del fichero de configuración	14
4.3. Cargar un sonido al juego	19
4.4. Movimiento vertical del jugador	20

Listado de ecuaciones

4.1. Cálculo del error de trayectoria en el tiempo	17
4.2. Cálculo del error de trayectoria por posición	17
4.3. Cálculo de la asistencia según el nivel de dificultad	20

Índice de cuadros

Capítulo 1

Introducción

Quizás algún fragmento de libro inspirador...

Autor, Título

Hablar del ictus como enfermedad, presencia de casos en España.

Hablar de la rehabilitación, aspectos positivos, actividades que se realizan.

Hablar de la inclusión de sistemas robóticos como herramienta terapéutica, mejoras en las sesiones.

Capítulo 2

Objetivos

Un objetivo bien definido es el punto de partida de todo logro.

W. Clement Stone, *Success Through a Positive Mental Attitude*, 1959

Después de haber establecido el marco contextual de este proyecto, continuo con la descripción del problema, los requisitos, las competencias, la metodología y el plan de trabajo empleado para su desempeño.

2.1. Descripción del problema

La recuperación motora tras un ictus requiere de terapias personalizadas y motivadoras que promuevan la participación activa del paciente. Tal como señalan [Clark et al., 2019], la ausencia de interfaces gráficas intuitivas y retroalimentación visual puede provocar una menor implicación del paciente durante el tratamiento, afectando de forma negativa los resultados terapéuticos.

En este contexto, el presente proyecto aborda dicha limitación mediante el diseño y desarrollo de un sistema de interacción gráfica que combina, por un lado, una plataforma de gamificación centrada en el paciente, con el objetivo de fomentar su participación y motivación, y por otro, una interfaz de visualización clínica centrada en el médico, que registre las métricas de desempeño y facilite el seguimiento de la evolución terapéutica.

2.1.1. Objetivo general

Diseñar una interfaz gráfica intuitiva y funcional para que un terapeuta controle y personalice las sesiones terapéuticas post-ictus, así como desarrollar un entorno gamificado que motive y facilite la participación activa del paciente durante su rehabilitación motora unilateral. De la misma manera, monitorizar el progreso y

ejecución del paciente y almacenar los datos más relevantes durante la terapia para facilitar el análisis y la evaluación de los resultados. Con el fin de mejorar la eficacia de dichas sesiones y aumentar la adherencia del paciente al tratamiento.

2.1.2. Objetivos específicos

Con el fin de alcanzar esta meta, se han definido los siguientes objetivos específicos:

- *O1.* Desarrollar entornos gamificados que incluyan estímulos visuales y auditivos, niveles de dificultad y misiones particulares, adaptadas al perfil del paciente, con el objetivo mejorar su atención y motivación.
- *O2.* Diseñar una interfaz gráfica intuitiva para que el terapeuta controle el desarrollo de la terapia y el modo de rehabilitación, permitiéndole visualizar el progreso del paciente y ajustar los parámetros del juego según su rendimiento.
- *O3.* Establecer un sistema capaz de recibir datos de sensores y controlar actuadores para coordinar estímulos visuales y físicos que el paciente experimente durante la terapia, con el fin de crear un entorno inmersivo que potencie la eficacia del tratamiento.
- *O4.* Crear una interfaz gráfica que permita definir los datos personales de un paciente y registrar su progreso después de cada sesión, con el objetivo de personalizar y ajustar de forma automática los parámetros en futuras sesiones.
- *O5.* Integrar perturbaciones controladas en el entorno de juego terapéutico, con el fin de introducir obstáculos que desafíen el control motor y la capacidad de adaptación del paciente, promoviendo una mayor atención, planificación motora y activación neuromuscular.

2.2. Requisitos

A continuación, se enumeran los requisitos que se deben satisfacer:

- Compatibilidad con la arquitectura del sistema robótico y su software de control, garantizando una integración fluida sin afectar al rendimiento del sistema.
- Diseño de una interfaz funcional y adaptable, que se ajuste a las necesidades de cada usuario.

- Integración de señales de tipo perturbación controladas, para introducir variabilidad y desafíos en a terapia.
- Incorporación de elementos de gamificación, como niveles de dificultad y retroalimentación visual, que aumente la atención del paciente.
- Capacidad para adaptar la asistencia robótica, en función del esfuerzo realizado por el paciente, de forma progresiva.
- Registro de los datos terapéuticos generados durante las sesiones, para permitir su uso y análisis posterior.
- Validación del sistema mediante la valoración de los resultados obtenidos por los usuarios, considerando métricas objetivas y subjetivas.

2.3. Competencias

Durante la realización de este proyecto, se han puesto en práctica diversas competencias generales del grado y específicas del desarrollo de interfaces y sistemas dinámicos.

2.3.1. Competencias generales

- *CG1*: Diseño software. Conocimiento de materias básicas y tecnologías, que le capacite para el aprendizaje de nuevos métodos y tecnologías, así como que le dote de una gran versatilidad para adaptarse a nuevas situaciones.
- *CE15*: Arquitectura software para robots. Capacidad de diseñar y programar aplicaciones robóticas y sistemas inteligentes en red usando middlewares, mecanismos de comunicación y estándares propios del ámbito de la Ingeniería Robótica.
- *CB2*: Planificación y sistemas cognitivos. Que los estudiantes sepan aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro de su área de estudio.
- *CB4*: Trabajo de Fin de Grado. Que los estudiantes puedan transmitir información, ideas, problemas y soluciones a un público tanto especializado como no especializado.

2.3.2. Competencias específicas

- *CE1*. Diseño e implementación de interfaces gráficas de usuario (GUI) con herramientas como Tkinter.
- *CE2*. Desarrollo de aplicaciones interactivas con elementos de gamificación, aplicando principios de diseño centrado en el usuario.
- *CE3*. Integración de interfaces con dispositivos físicos mediante comunicación en tiempo real basada en el Sistema Operativo Robótico (ROS¹).
- *CE4*. Evaluación de la usabilidad y experiencia de usuario con métricas objetivas y percepciones.

2.4. Metodología

Para llevar a cabo este proyecto, se ha optado por seguir el paradigma iterativo centrado en el usuario (User-Centered Design). Previamente, se ha realizado una fase de análisis e investigación sobre el estado del arte y necesidades específicas para comprobar la viabilidad de su desarrollo. Posteriormente, se procedió con el diseño del prototipo para validar el diseño gráfico y la navegación. Más adelante, la realización de pruebas continuas favoreció la incorporación de mejoras en el diseño visual, la lógica de juego y la retroalimentación. Una vez se desarrolló una primera versión fiable, las interfaces se conectaron con el sistema de control del robot permitiendo crear un entorno cambiante y adaptable. Por último, se llevó a cabo una evaluación de conformidad a los usuarios para comprobar la efectividad de la plataforma.

2.5. Plan de trabajo

El trabajo se ha organizado de forma progresiva y coordinada con el resto del equipo técnico. Para ello, el proyecto se ha dividido en las siguientes etapas:

1. *Investigación*. En esta fase inicial, se llevó a cabo un análisis detallado de plataformas robóticas existentes en el ámbito de la rehabilitación, como el dispositivo Armeo Spring², entre otros. El fin de este estudio fue identificar funcionalidades relevantes y enfoques innovadores que pudieran servir de inspiración para el diseño y desarrollo de la nueva plataforma.

¹<https://www.ros.org/>

²<https://www.comunidad.madrid/noticias/2019/04/16/comunidad-incorpora-robot-rehabilitacion-hospital-guadarrama>

2. *Planteamiento del software.* Una vez se establecieron los objetivos y requisitos del proyecto, se procedió con el diseño de un esquema de nodos y topics que permitió estructurar de forma organizada la arquitectura de la plataforma.
3. *Desarrollo de la parte gráfica.* Después de definir la arquitectura del sistema, se inició el desarrollo de las interfaces gráficas. A través de continuas pruebas se fueron optimizando tanto su aplicación como su uso.
4. *Conexión entre el software y hardware.* Una vez consolidada una primera versión fiable del software, se continuó con la integración de las interfaces con el sistema de sensorización del robot.
5. *Evaluación y conclusiones.* En esta etapa final, se evaluó la usabilidad y se validaron los controles de forma técnica.

Paralelamente, se ha ido completando la presente memoria a lo largo de este proceso. Asimismo, se han realizado reuniones mensuales con el equipo para compartir los progresos de cada área, proponer mejoras y resolver cuestiones, trabajando de manera coordinada hacia un objetivo común.

A continuación, se procede a tratar las plataformas de desarrollo empleadas en este proyecto.

Capítulo 3

Plataforma de desarrollo

Lo simple es mejor que lo complejo.

Tim Peters, *The Zen of Python*

A continuación, se explican las plataformas de desarrollo utilizadas a nivel de *software* y *hardware*, que han permitido alcanzar los objetivos descritos en el capítulo anterior.

3.1. Sistema operativo

Un sistema operativo puede definirse como el conjunto de programas que gestionan el hardware de un ordenador o dispositivo y permiten el funcionamiento de otros programas. Como describen [Soriano-Salvador and Guardiola Múzquiz, 2022], una distribución es una colección concreta de software de área de usuario y un núcleo del sistema operativo.

Para la realización de este proyecto se ha utilizado el sistema operativo de tipo GNU/Linux² y su distribución Ubuntu³, en concreto la versión Ubuntu 22.04. La razón de usar esta versión de Ubuntu se debe a su compatibilidad con la distribución *Humble*⁴ de ROS 2, ya que el sistema ha sido diseñado específicamente para funcionar con dicha distribución.

²<https://www.debian.org/releases/stable/i386/ch01s02.es.html>

³<https://ubuntu.com/>

⁴<https://docs.ros.org/en/humble/index.html>

3.2. Entorno de desarrollo

3.2.1. Python

Python⁵ es el lenguaje de programación utilizado para el desarrollo de este proyecto. Se trata de un lenguaje de alto nivel creado por Guido van Rossum y publicado por primera vez en 1991. Según [Stack Overflow,] es ampliamente reconocido por su simplicidad y flexibilidad, lo que hace que sea ideal tanto para principiantes como para desarrolladores expertos. Cuenta con un sistema de tipos dinámico y gestión automática de la memoria y soporta múltiples paradigmas de programación, incluyendo orientación a objetos, programación funcional y estilos procedimentales. Además, dispone de una amplia y completa biblioteca estándar, que facilita el desarrollo de aplicaciones de propósito general.

La versión de Python utilizada para el desarrollo de la GUI es Python 3.10.13. En particular, se emplearon las siguientes librerías:

- *matplotlib*: Matplotlib⁶ es una biblioteca completa para crear visualizaciones estáticas, animadas e interactivas en Python. Se ha empleado para la graficación de las señales de trayectoria y perturbación, la posición del paciente en los entornos gamificados y de visualización del médico, y los datos clínicos recogidos durante la terapia.
- *numpy*: NumPy⁷ es una biblioteca de Python que proporciona un objeto matriz multidimensional, varios objetos derivados como matrices y matrices enmascaradas y una serie de rutinas para realizar operaciones rápidas con matrices, incluidas operaciones matemáticas, lógicas, manipulación de formas, ordenación, selección, entrada/salida, transformadas discretas de Fourier, álgebra lineal básica, operaciones estadísticas simples, simulación aleatoria, entre otros. En especial se ha implementado dentro de los entornos gamificados para crear las señales límites, de trayectoria y perturbación para su posterior representación.
- *tkinter*: Tkinter⁸ es la interfaz por defecto de Python para el kit de herramientas de GUI Tk. En concreto, se han utilizado los módulos Tk y Ttk para el diseño de la interfaz de control del médico y la interfaz de registro de pacientes.
- *pygame*: Pygame⁹ es una biblioteca multiplataforma gratuita y de código abierto

⁵<https://www.python.org/>

⁶<https://matplotlib.org/>

⁷<https://numpy.org/>

⁸<https://docs.python.org/es/3.13/library/tkinter.html>

⁹<https://pypi.org/project/pygame/>

para el desarrollo de aplicaciones multimedia. Utilizada para incluir efectos de sonido dentro de un videojuego.

3.2.2. ROS 2

Como explica [Martín Rico, 2023], ROS es un middleware que aumenta las capacidades del sistema para crear aplicaciones robóticas. El número dos indica que se trata de la segunda generación de este middleware. Los paquetes de ROS 2 están organizados en distribuciones, las cuales están formadas por múltiples paquetes que aseguran su interoperabilidad. Esta arquitectura modular permite estructurar el sistema como un grafo de computación, compuesto por nodos de ROS 2 que se comunican entre sí, permitiendo así que un robot realice una serie de tareas.

Se destacan las siguientes librerías:

- *rclpy*: *rclpy*¹⁰ proporciona la Interfaz de Programación de Aplicaciones (API) canónica de Python para interactuar con ROS 2.
- *std_msgs*: *std_msgs*¹¹ define tipos de mensajes básicos y estandarizados para la publicación y suscripción entre nodos. Los tipos de mensajes utilizados en este proyecto son `Int32`, `Int32MultiArray`, `Float32` y `Float32MultiArray`.

3.3. Componentes físicos

Una vez definido el software, se especifica el hardware encargado de ejecutar las acciones en un entorno físico.

Para el movimiento del brazo robótico se utiliza un actuador de HEBI Robotics¹², más concretamente el modelo T-Series como puede observarse en la Imagen 3.1. Este actuador está diseñado para funcionar como un componente robótico con todas las funciones. La salida gira continuamente, no requiere calibración ni referencia alguna de arranque. Diseñado para funcionar en brazos robóticos con múltiples grados de libertad. Integra varios sensores fácilmente accesibles a través de varias APIs.

La selección de este modelo se debe a la incorporación de un sensor de posición, lo que permite registrar el movimiento del brazo del paciente. Esto resulta fundamental para la validación de los entornos gamificados, los cuales proponen una serie de actividades donde la posición del paciente se toma como referencia para el desarrollo de actividades interactivas y dinámicas dentro de un contexto terapéutico.

¹⁰<https://docs.ros.org/en/iron/p/rclpy/>

¹¹https://docs.ros2.org/foxy/api/std_msgs/index-msg.html

¹²<https://www.hebirobotics.com/actuators>



Figura 3.1: Actuador T-Series

Capítulo 4

Diseño

*El diseño no es solo lo que se ve y lo que se siente.
El diseño es cómo funciona.*

Steve Jobs, conferencia de lanzamiento del Apple iPod, 2001

Después de definir la plataforma de desarrollo, se procede a explicar el diseño de la aplicación.

El proyecto se divide en cuatro scripts, el primero se utiliza para crear o registrar un paciente, el segundo lanza la GUI y controla los parámetros del juego, el tercero permite visualizar el comportamiento del paciente durante la terapia, y el cuarto es el propio juego.

4.1. Interfaz de registro de un paciente

Este script permite gestionar un conjunto de datos, que registran a un paciente, mediante una GUI.

En primer lugar, se importan las bibliotecas estándar, mencionadas en el capítulo anterior, como `tkinter`, que se utiliza para crear la interfaz gráfica, `csv` para guardar los datos en un archivo CSV para su posterior uso, y `os` para interactuar con el sistema de archivos.

Se obtiene el directorio de inicio del usuario y se crea un directorio dentro de este, si no existe, bajo el nombre `database`, donde se almacenan los ficheros con los datos de registro y análisis terapéuticos de cada paciente.

Se definen tres funciones principales que gestionan las operaciones de la GUI. La función `exit()` cierra la ventana principal de Tkinter, `clear()` limpia los campos de entrada y texto, y `savedata()` guarda los valores de los campos, valida que el ID sea un número y crea un subdirectorio bajo el nombre del ID, si no existe, donde guarda los datos en un archivo CSV llamado `ID.csv`. Si el archivo no existe, se

escribe el encabezado utilizando `writer.writeheader()`. Los datos se escriben con `csv.DictWriter` y son guardados como `strings`.

Se crea una ventana principal con un título y tamaño fijo de 600×500 píxeles. Los estilos visuales se configuran con `ttk.Style` y se definen dos frames distintas, `main_frame` se utiliza como contenedor de los campos de entrada y texto y `button_frame` agrupa la lógica de los botones. Cada campo es un `Entry` enlazado a una variable de tipo `StringVar` y hacen referencia al nombre, apellido e ID del paciente, frecuencia, amplitud y perturbación de la señal que generará el brazo robótico, nivel y progreso del juego y un espacio para que el doctor incluya observaciones. Se crean dos botones, *Save* llama a `savedata()` para guardar los datos, que a su vez llama a `clear()` para limpiar los campos una vez que estos se han almacenado, y el botón *Exit* llama a la función `exit()` que cierra la aplicación. En la Imagen 4.1 puede observarse el estilo de la interfaz.

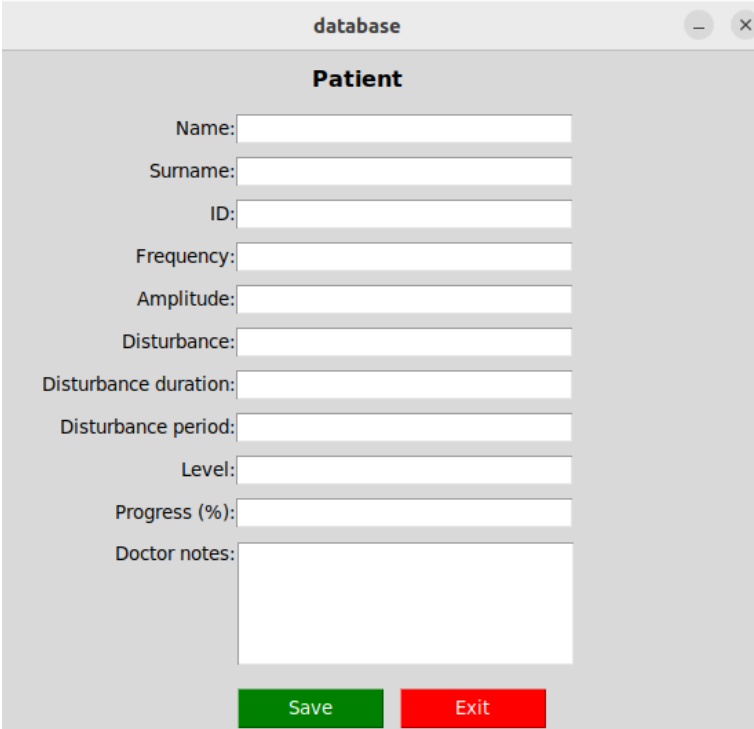


Figura 4.1: Interfaz de registro de un paciente

`root.mainloop()` inicia el bucle de eventos de Tkinter y la interfaz está activa hasta que el usuario cierra la ventana.

En el Código 4.1, escrito en `Python`, se muestra cómo se gestionan los directorios y la escritura del archivo CSV.

```
def savedata():
    ID_DIR = os.path.join(DATABASE_DIR, id)
    os.makedirs(ID_DIR, exist_ok=True)
    file_name = f"{id}{ext}"
    file_path = os.path.join(ID_DIR, file_name)

    # [...]

    file_exists = os.path.isfile(file_path)
    with open(file_path, mode="a", newline="") as file:
        writer = csv.DictWriter(file, fieldnames=patient_data.keys())
        if not file_exists:
            writer.writeheader()
        writer.writerow(patient_data)
```

Código 4.1: Función para guardar los datos de un paciente

4.2. Interfaz de control

Este script combina ROS 2 y Tkinter para crear una GUI que gestiona límites físicos (mínimo, máximo y offset) del robot, teniendo en cuenta el brazo que se va a rehabilitar, configura y publica parámetros para crear una señal de control y perturbación para un sistema robótico, parámetros de asistencia y nivel de dificultad para un videojuego terapéutico y almacena dichas configuraciones por paciente.

Al inicio del script se importan las librerías `rclpy` y `std_msgs.msg` para gestionar la comunicación entre Python y ROS 2, `tkinter` para crear la interfaz gráfica, `csv` para el manejo de archivos CSV, `os` para interactuar con el sistema y `datetime` para gestionar la fecha de creación de los archivos.

Se crea un directorio en el directorio de inicio del usuario, si no existe, bajo el nombre *database*, donde se almacenan los ficheros con los datos de configuración de cada paciente.

Se definen dos clases, `ScrollPublisherNode` y `ScrollGUI`. La primera es una clase de ROS 2 que publica los parámetros de dos señales, de tipo trayectoria y perturbación en el topic `/SliderParameters` y los datos de asistencia y nivel de juego en `/GameParameters`. Los mensajes son de tipo `Float32MultiArray` y `Int32MultiArray`, y se publican cada *100ms* y únicamente cuando se actualiza el valor de los datos, respectivamente. La decisión de publicar ambas señales en el mismo topic a dicha velocidad se debe a que la actualización de los datos debe realizarse de forma casi instantánea para permitir ajustes inmediatos en los parámetros de la terapia y evaluar la rapidez de respuesta del paciente. Y, se ha comprobado que esta estimación

es lo suficientemente rápida para satisfacer estos requisitos. Los datos necesarios para generar la señal de trayectoria son frecuencia, amplitud y tipo, mientras que para la perturbación son duración, amplitud y tipo. El tiempo de las señales se define de manera distinta para evitar redundancias, ya que la perturbación no es constante. En su lugar, se estima una duración que facilita el entendimiento del terapeuta. El tipo de señal hace referencia al modo de juego y se codifica como 1.0 (hold o mantener) y 2.0 (follow o seguir), y el tipo de perturbación como 1.0 (sinusoidal) y 2.0 (step o escalón). También se define un offset que determina la distancia entre la señal principal y los límites superior e inferior, ajustando así los márgenes del camino. La segunda clase crea la interfaz gráfica y permite ajustar los parámetros del juego en tiempo real y publicarlos a través de un nodo de ROS. Los datos se guardan en un archivo CSV bajo el nombre `ID-year-month-day-config_<index>.csv` en un subdirectorio llamado *config* dentro del directorio `home/user/database/ID/`. *index* hace referencia al número de archivo de la sesión diaria. Si el archivo no existe, se escribe un encabezado con los nombres de los campos a los que hacen referencia los datos como se observa en el Código 4.2.

```
header = ["frequency", "amplitude", "offset", "signal", "disturbance",
         "duration", "period", "mode"]
```

Código 4.2: Encabezado del fichero de configuración

Se crean deslizadores, utilizando `ttk.Scale`, para ajustar la frecuencia, amplitud, offset, duración y periodo de las señales, lo que facilita la configuración de los parámetros con mayor precisión y comodidad. Se utiliza `Combox` para permitir la selección entre los distintos tipos de señal y perturbación y niveles de asistencia (del 0 al 5, donde 0 es asistencia nula y 5 asistencia máxima) y de juego (del 1 al 10, de menor a mayor dificultad). Los botones *Update Signal* y *Update levels* se utilizan para actualizar los datos en el topic correspondiente, y el botón *Exit* para salir. En la Imagen 4.2 se observa el aspecto de la ventana principal de control.

La función `saveconfig()` guarda la configuración actual de la GUI en el archivo CSV. Las funciones `update*()` convierten el valor del deslizador de `str` a `float`. Entre ellas se destacan `updatesignal()`, que actualiza los atributos del nodo y llama a `saveconfig()`, que añade al final del archivo la nueva configuración, y `updatelevel()`, que publica un mensaje con los datos de asistencia y nivel de dificultad del juego. La función `close()` finaliza el nodo y cierra la GUI y el método `run()` inicia el bucle de eventos de Tkinter junto con `spin_once()`. `loadcsv()` lee el último registro del archivo del paciente que se pasa como parámetro y devuelve los valores de frecuencia,

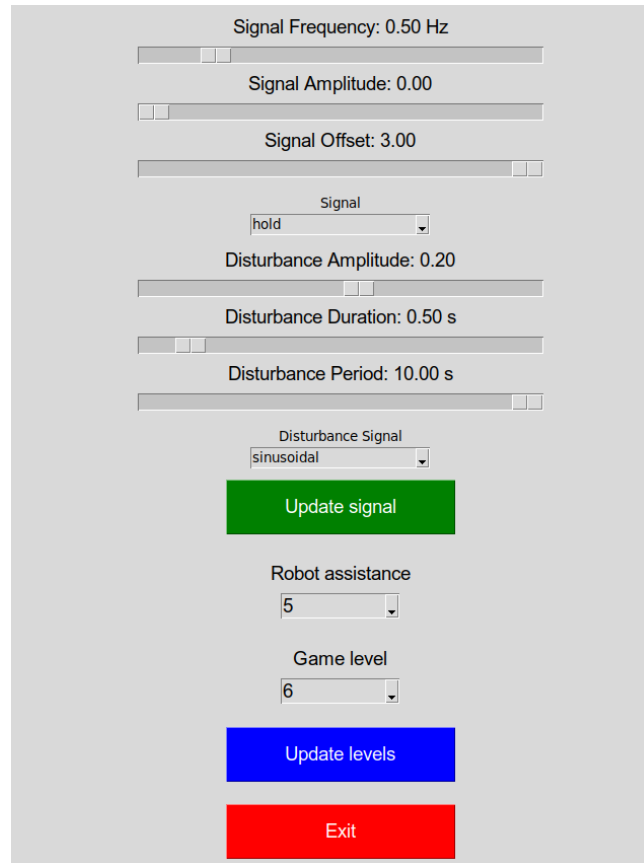


Figura 4.2: Interfaz de control

amplitud, perturbación, duración, periodo y nivel del juego. Todos son `float` excepto el último que es un `int`.

`main()` es la función principal y se encarga de extraer el ID del paciente desde la ruta al archivo de registro, cargar los datos de las señales y el juego desde el CSV, crear el nodo `ScrollPublisherNode` con dichos parámetros e iniciar la GUI. `startgui()` es la primera interfaz que se lanza y gestiona la configuración del brazo que se va a rehabilitar y de los límites físicos del robot a través de la publicación de un `Int32MultiArray` que actúa como un booleano indicando el botón que se ha pulsado (resetear, mínimo, máximo u offset). El offset corresponde a la posición inicial desde la cual comienza la terapia. Además, comprueba que se ha seleccionado o bien brazo derecho o izquierdo y que los límites mínimo y máximo se han definido correctamente antes de permitir establecer el offset o continuar a la siguiente ventana. La decisión de ajustar los límites y el offset mediante el movimiento del brazo robótico a una posición específica, y luego guardar dicha posición a través de un botón, responde a un requisito explícito del cliente. En la Imagen 4.3 se muestra el formato de la ventana de inicio.

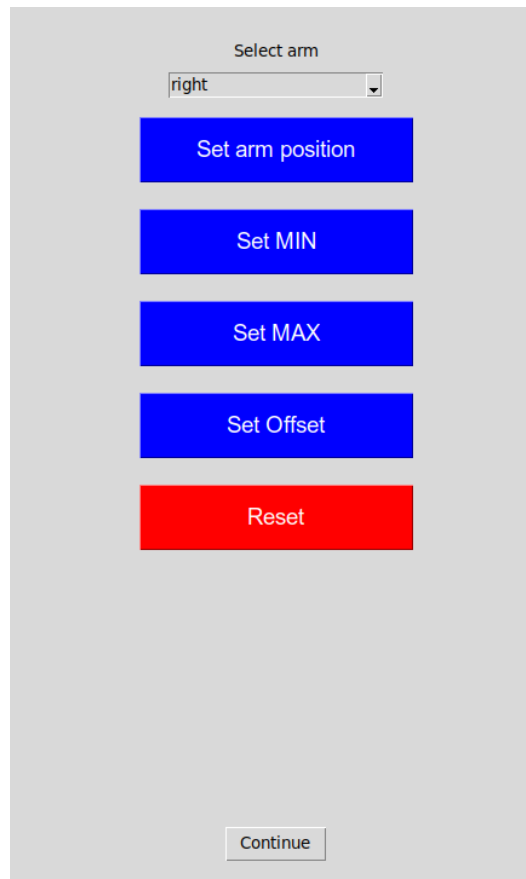


Figura 4.3: Interfaz de configuración de los límites del brazo robótico

4.3. Interfaz de visualización

Este script permite observar y registrar la ejecución de un paciente en una tarea de seguimiento de trayectoria como parte de una sesión de rehabilitación motora.

Además de incluir las bibliotecas mencionadas en la Sección 4.2, se utilizan `matplotlib` para visualizar los datos del rendimiento del jugador en tiempo real, `numpy` para manejar los datos de las señales que cambian con el tiempo y `sys` para gestionar la ruta al archivo CSV con los datos de registro del paciente.

Se utiliza el modo interactivo de `matplotlib`, `plt.ion()`, que permite actualizar dinámicamente los gráficos sin bloquear el hilo principal. Las ventanas son deslizantes respecto al eje X, no obstante debemos mantener un tamaño fijo para que la señal no se desplace hacia la izquierda, por ello se implementa `np.roll()`. El tamaño de la ventana en el eje Y se ajusta a 3 ya que es el máximo recorrido que puede hacer el brazo.

Se define la clase `FlappyBirdViewerNode` que actúa como nodo de ROS 2 y se suscribe a los topics `/CleanSignal`, donde se publica la señal de referencia que se toma como trayectoria deseada que el jugador debe seguir, `/Disturbance`, que contiene la

señal de perturbación, `/ActuatorPosition`, que publica la posición del jugador en el eje Y, y `/MotorParameters`, para obtener el tiempo del juego, la posición del jugador en el eje X y el nivel de asistencia del robot. Los mensajes de los primeros tres topics son de tipo `Float32` y el último es un `Float32MultiArray`. `signalcallback()` se activa cada vez que se recibe un valor de la señal, si el jugador está activo se actualizan los datos (tiempo, límites, posición, error de trayectoria y detección de colisiones) y se grafican. Para la detección de colisiones se implementa una lógica simple basada en la comparación de la posición del jugador con los límites de la señal. `playercallback()` actualiza la posición del jugador, el offset que separa la señal de los límites y el tiempo total y marca que el jugador está activo.

En la Imagen 4.4, se muestran dos gráficos. El que está en la parte superior muestra la señal de trayectoria (línea azul), la perturbación de tipo escalón (línea verde), los límites superior e inferior (líneas discontinuas grises) y la posición actual del jugador (punto rojo), proporcionando una visión clara y completa de la terapia. Y, la parte inferior grafica el error de posición con respecto a la trayectoria deseada en función del tiempo, lo que permite detectar fallos de control, fatiga o pérdida de atención.

En un principio, el error se calculaba en función del tiempo como se muestra en la Ecuación 4.1, pero, más adelante, se optó por calcular la interpolación (o selección por índice más cercano) del valor de la trayectoria de referencia (x, y) evaluando la posición del jugador en el eje X, x_p , como se muestra en la Ecuación 4.2. El segundo cálculo ofrece una aproximación más precisa al error, ya que está directamente relacionado con la ubicación del jugador. A diferencia del primer cálculo, que depende del instante de tiempo y puede verse afectado por variaciones en la sincronización de los datos.

$$error_t = |y_{player}(t) - y_{reference}(t)| \quad (4.1)$$

Ecuación 4.1: Cálculo del error de trayectoria en el tiempo

$$error(x_p, y_p) = |y_p - f(x_p)| \quad (4.2)$$

Ecuación 4.2: Cálculo del error de trayectoria por posición

Al finalizar la ejecución, los datos más relevantes como el tiempo, la señal, los límites mínimo y máximo, la posición del jugador, el offset, el error, las colisiones y el nivel de asistencia se almacenan en un archivo CSV bajo el nombre `ID-year-month-day-metrics_<index>.csv` en un subdirectorío llamado *metrics* dentro del directorío `home/user/database/ID/`. `index` hace referencia al número de archivo de la sesión diaria. Esto facilita un análisis posterior de la terapia.

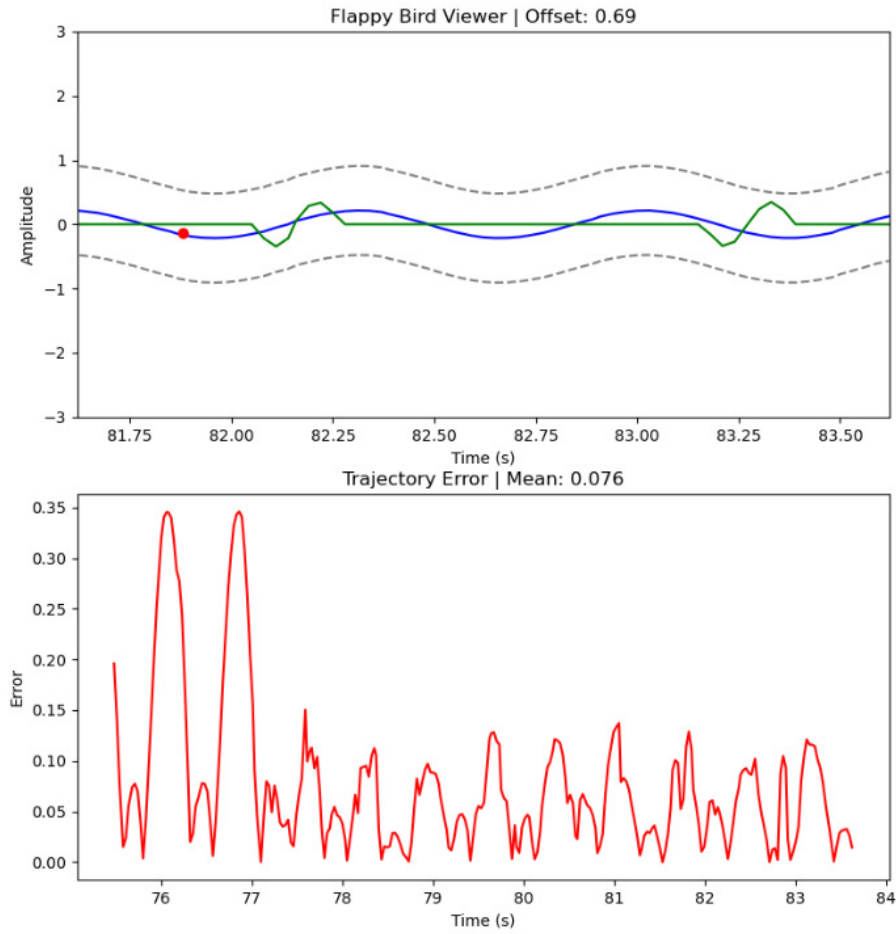


Figura 4.4: Interfaz de visualización del terapeuta

La función principal `main()` verifica que se ha pasado como único argumento un archivo CSV con los datos de registro del paciente, extrae el ID desde la ruta, crea el nodo `FlappyBirdViewerNode` y escucha de los topics.

4.4. Juego flappy

Este script implementa un juego basado en el Flappy Bird, adaptado a un entorno ROS 2 para interactuar con sensores y actuadores en tiempo real. Existen dos modos de juego distintos, *hold*, el jugador debe mantener la posición, y *follow*, el jugador debe seguir una trayectoria.

Se implementan las librerías `rclpy`, y `std_msgs` para permitir la comunicación con ROS 2, `matplotlib` y `numpy` para la visualización y el manejo de señales, y `pygame` para utilizar efectos de sonido.

Se emplean sonidos distintos cuando se sube de nivel, se consigue una recompensa o se colisiona. Esto permite que el juego sea más interactivo. En el Código 4.3, escrito en *Python*, se observa como se carga un sonido desde un directorio, *sounds*,

dentro del paquete del proyecto. Los formatos de audio más comunes soportados por `pygame.mixer` son `.wav`, `.ogg` y `.mp3`. La elección de utilizar la extensión `.wav` se debe a su simplicidad, alta calidad y compatibilidad con todos los sistemas.

```
pygame.mixer.init()
level_up_sound = pygame.mixer.Sound('../sounds/level_up.wav')
```

Código 4.3: Cargar un sonido al juego

La clase `FlappyBirdNode` engloba la lógica del programa. En `__init__` se crean los suscriptores `/CleanSignal`, de tipo `Float32`, es la señal de referencia, `/Disturbance`, también `Float32`, es la perturbación, `/GameParameters`, es un `Int32MultiArray`, contiene el nivel del juego y de asistencia del robot, `/SliderParameters`, `Float32MultiArray`, para obtener el offset entre la señal y los límites superior e inferior, y `/ActuatorPosition`, es un `Float32`, contiene la posición del jugador en el eje Y. Se crea un publicador `Float32MultiArray`, llamado `/MotorParameters`, que contiene la marca de tiempo de la ventana, las referencias de la señal y la perturbación en (x_p, y) , la posición del jugador en el eje X y la asistencia actualizada. Se crea una ventana gráfica de tamaño 2×3 , donde 3 es el recorrido máximo del brazo. En ella se representan la señal principal, los límites mínimo y máximo, las perturbaciones, la posición del jugador y objetos como estrellas y asteroides que forman parte de la lógica del juego.

La posición del jugador se representa como un punto rojo que cambia de color según la distancia a la que se encuentra entre la trayectoria deseada y los límites (negro significa que está cerca de la trayectoria, gris que está a una distancia media y rojo indica peligro de colisión con los límites). La posición se representa como una coordenada en los ejes XY, en el eje X permanece siempre a la misma distancia (a 0,25 unidades del origen), y en el eje Y se muestra la posición que se recibe del sensor de posición del motor. En un principio, para validar el movimiento del jugador, la posición en el eje Y se basaba en el manejo de las flechas arriba y abajo del teclado. Para ello se utilizó la librería `pynput`. En el Código 4.4, escrito en Python, se muestra la lógica de las teclas.

El método `updatelevel()` cambia los colores del fondo y de la línea según el nivel, reproduce el sonido de subida de nivel y disminuye la asistencia para que la dificultad sea progresiva. La asistencia del robot se disminuye utilizando la Fórmula 4.3. Se utiliza el máximo entre 0 y el valor calculado según el nivel para que la asistencia no sea un valor negativo.

Para crear un juego dinámico que capte la atención del paciente se crea una misión

```
def on_press(self, key):
    try:
        direction = -1 if self.inverted_gravity else 1
        if key == keyboard.Key.up:
            self.player_y += 0.1 * direction
        elif key == keyboard.Key.down:
            self.player_y -= 0.1 * direction
    except:
        pass
```

Código 4.4: Movimiento vertical del jugador

$$assistance = \max(0, assistance - (level - 1)/2) \quad (4.3)$$

Ecuación 4.3: Cálculo de la asistencia según el nivel de dificultad

secundaria en cada nivel. Los niveles 1, 4 y 7 tienen como objetivo permanecer dentro de los límites por un tiempo determinado. Esto permite evaluar la precisión del paciente. Además, cuando se sobrepasa uno de los límites el contador se para y se reanuda de nuevo cuando se vuelve al camino. Los niveles 2 y 6 se enfocan en el ajuste de la posición para chocar con las estrellas que están dispuestas en la señal principal. De esta forma se consigue que el paciente permanezca y siga la trayectoria deseada. Los niveles 3 y 8 pretenden ejercitar la rapidez de reacción mientras el paciente evita colisionar con los asteroides. Los niveles 5 y 9 fomentan la flexibilidad cognitiva del paciente para adaptarse a una nueva lógica de control basada en el concepto de gravedad invertida (arriba es abajo y viceversa). Y, por último, el nivel 10 permite al médico evaluar distintos escenarios límites mediante ajustes en los parámetros de la señal, perturbación y el offset. En las Imágenes 4.5 y 4.6 se comparan dos niveles diferentes del juego. La primera imagen presenta límites más estrictos que la segunda. En la segunda imagen, se observan dos perturbaciones de tipo sinusoidal con sentidos opuestos, representadas por triángulos verdes cuya punta indica el signo de la perturbación. En contraste, la primera imagen no muestra ninguna perturbación.

Se introduce un sistema de recompensas para aumentar la motivación y destreza del paciente. Esto permite reforzar comportamientos y habilidades específicas para alcanzar los objetivos propuestos. Cada vez que se consigue una estrella y se sube de nivel se suman 10 y 30 puntos, respectivamente, a un contador. Y, cuando se colisiona con un asteroide, el contador decremента en 5 puntos. Para ello se utilizan las funciones `incrementscore()` y `decrementscore()`.

Los objetos como las estrellas, los asteroides y los triángulos, que indican el sentido

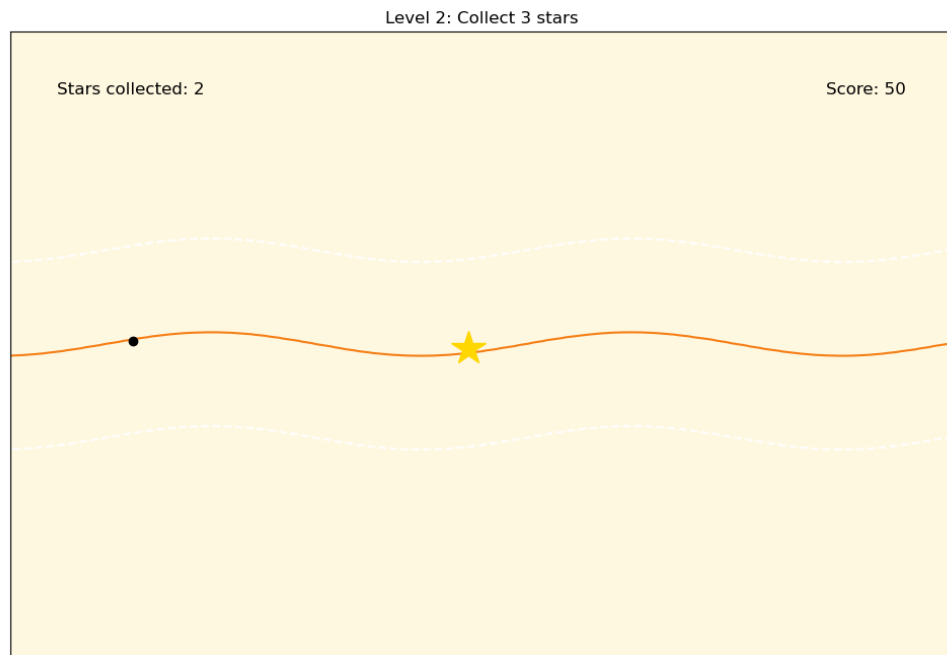


Figura 4.5: Nivel 2 del juego sin perturbación

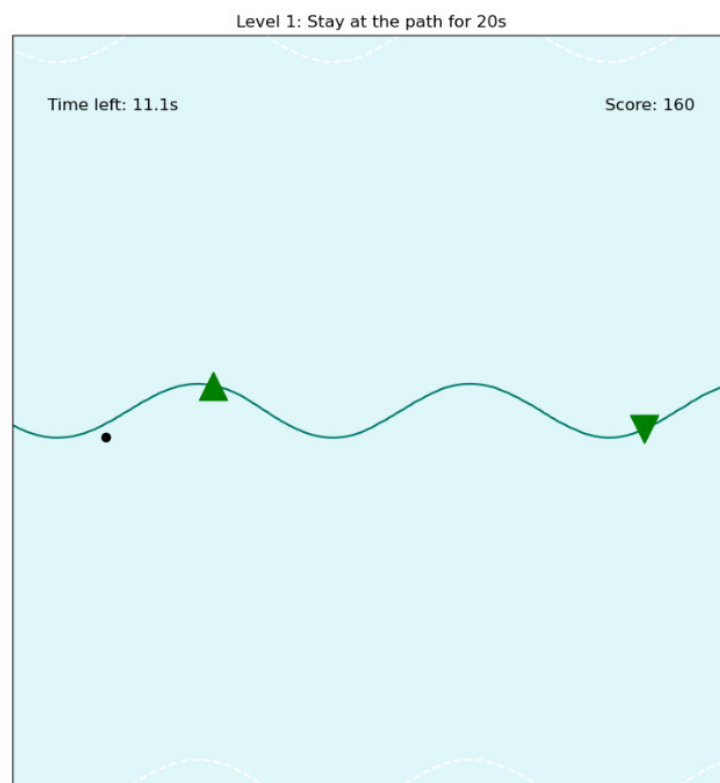


Figura 4.6: Nivel 1 del juego con perturbación

y comienzo de una perturbación, se crean con `ax.plot`. Las estrellas se generan de forma aleatoria en el eje X, y los asteroides en ambos ejes con límites definidos

entre $(0,3, offset)$ y $(-offset, -0,3)$. El máximo de objetos visuales al mismo tiempo dentro de la ventana es 2. Las funciones `generatestar()`, `generateasteroid()` y `generatedisturb()` encapsulan la lógica de creación de objetos, y la función `clearobjects()` la eliminación de estos.

El desarrollo del juego está concentrado en `listenercallback()`, que es el bucle principal y se ejecuta cada vez que se recibe un nuevo dato en el topic `/CleanSignal`. La señal y la perturbación se actualizan con `np.roll`. Como se explica en la Sección 4.3, los gráficos se actualizan en tiempo real con `plt.ion()`.

Los callbacks actualizan los datos que se reciben de los topics, estas funciones son `disturbcallback()`, `levelcallback()`, `offsetcallback()` y `positioncallback()`.

La función `main()` inicializa el nodo ROS, crea una instancia del juego y entra en un bucle hasta que el usuario detenga el programa.

4.5. Esquema de nodos y topics

En la Imagen 4.7 se contempla un esquema de los nodos, topics y tipos de datos que se utilizan para la comunicación de la plataforma software con el actuador.

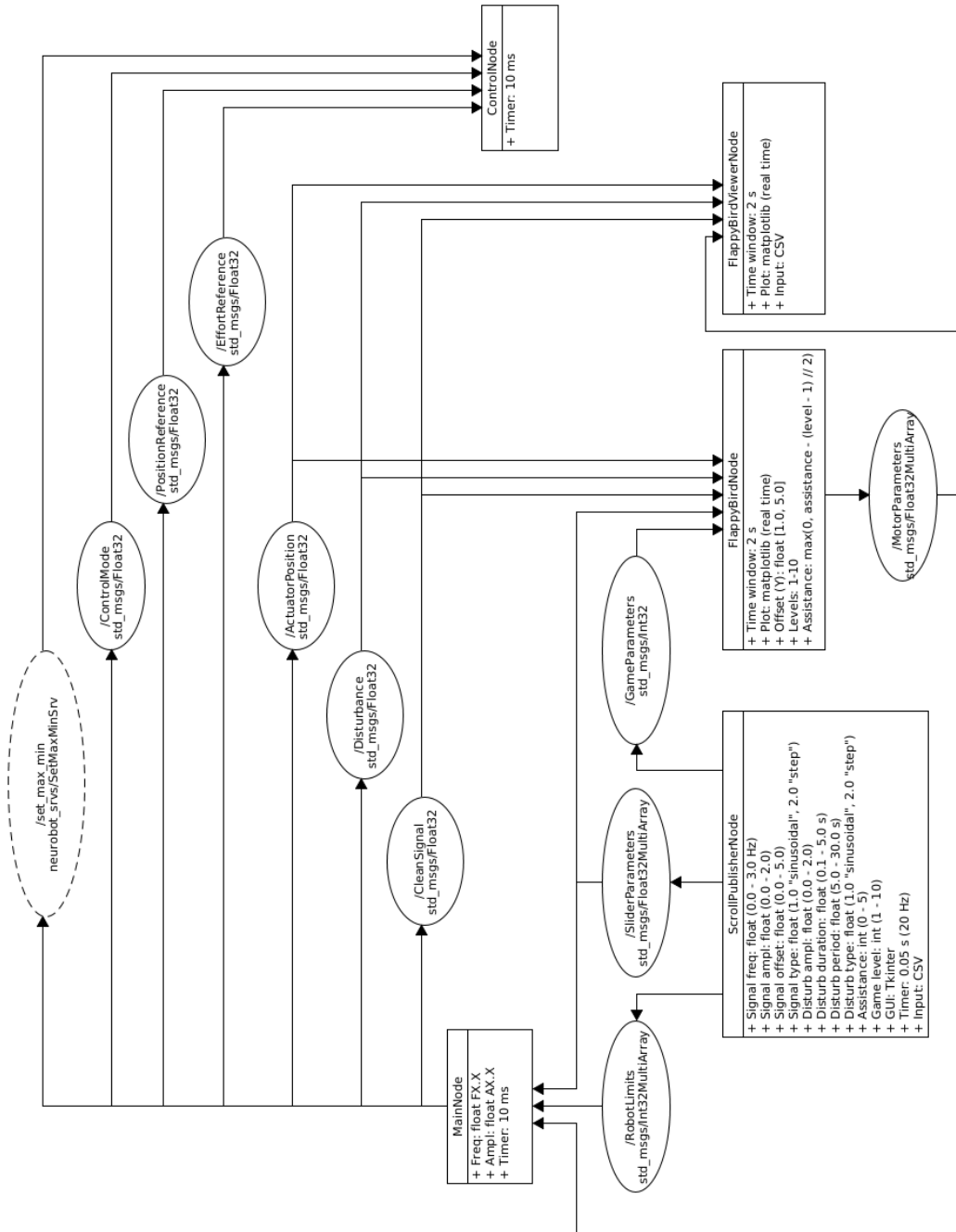


Figura 4.7: Esquema de nodos y topics

Capítulo 5

Conclusiones

Quizás algún fragmento de libro inspirador...

Autor, *Título*

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo, que básicamente será una recapitulación de los problemas que has abordado, las soluciones que has prouesto, así como los experimentos llevados a cabo para validarlos. Y con esto, cierras la memoria.

5.1. Conclusiones

Enumera los objetivos y cómo los has cumplido.

Enumera también los requisitos implícitos en la consecución de esos objetivos, y cómo se han satisfecho.

No olvides dedicar un par de párrafos para hacer un balance global de qué has conseguido, y por qué es un avance respecto a lo que tenías inicialmente. Haz mención expresa de alguna limitación o peculiaridad de tu sistema y por qué es así. Y también, qué has aprendido desarrollando este trabajo.

Por último, añade otro par de párrafos de líneas futuras; esto es, cómo se puede continuar tu trabajo para abarcar una solución más amplia, o qué otras ramas de la investigación podrían seguirse partiendo de este trabajo, o cómo se podría mejorar para conseguir una aplicación real de este desarrollo (si es que no se ha llegado a conseguir).

5.2. Corrector ortográfico

Una vez tengas todo, no olvides pasar el corrector ortográfico de L^AT_EXa todos tus ficheros *.tex*. En **Windows**, el propio editor **TeXworks** incluye el corrector. En **Linux**, usa **aspell** ejecutando el siguiente comando en tu terminal:

```
aspell --lang=es --mode=tex check capitulo1.tex
```

Bibliografía

- [Clark et al., 2019] Clark, W. E., Manoj, S., and O’Connor, R. J. (2019). Evaluating the use of robotic and virtual reality rehabilitation technologies to improve function in stroke survivors: A narrative review. *Rehabilitation and Assistive Technologies Engineering*, 6:3–5.
- [Martín Rico, 2023] Martín Rico, F. (2023). A concise introduction to robot programming with ros2. pages 1–5. Este libro se ha preparado a partir de una copia camera-ready proporcionada por los autores.
- [Soriano-Salvador and Guardiola Múzquiz, 2022] Soriano-Salvador, E. and Guardiola Múzquiz, G. (2022). Fundamentos de sistemas operativos: una aproximación práctica usando linux. pages 21–23. Para obtener la versión más reciente de este documento: <https://honecomp.github.io>.
- [Stack Overflow,] Stack Overflow, D. Python notes for professionals. page 2. This is an unofficial free book created for educational purposes and is not affiliated with official Python® group(s) or company(s).
- [Vega, 2008] Vega, J. (2008). Navegación y autolocalización de un robot guía de visitantes. Master thesis on computer science, Rey Juan Carlos University.
- [Vega, 2015] Vega, J. (2015). De la tiza al robot. Technical report.
- [Vega, 2018a] Vega, J. (2018a). *Educational framework using robots with vision for constructivist teaching Robotics to pre-university students*. Doctoral thesis on computer science and artificial intelligence, University of Alicante.
- [Vega, 2018b] Vega, J. (2018b). JdeRobot-Kids framework for teaching robotics and vision algorithms. In *II jornada de investigación doctoral*. University of Alicante.
- [Vega, 2019] Vega, J. (2019). El profesor Julio Vega, finalista del concurso ‘Ciencia en Acción 2019’. URJC, on-line newspaper interview.

- [Vega and Cañas, 2019] Vega, J. and Cañas, J. (2019). PyBoKids: An innovative python-based educational framework using real and simulated Arduino robots. *Electronics*, 8:899–915.
- [Vega et al., 2012] Vega, J., Perdices, E., and Cañas, J. (2012). *Attentive visual memory for robot localization*, pages 408–438. IGI Global, USA. Text not available.
- This book is protected by copyright.