UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Comunicación y coordinación bidireccional entre un sistema físico y un robot virtual

PROYECTO FIN DE GRADO

Alberto Quiñonero González
Grado en Ingeniería de Computadores

Madrid, 2025

UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Grado en Ingeniería de Computadores

Comunicación y coordinación bidireccional entre un sistema físico y un robot virtual

PROYECTO FIN DE GRADO

Alberto Quiñonero González
Grado en Ingeniería de Computadores

Bajo la dirección de:
Javier García Martín

Madrid, 2025

Título: Comunicación y coordinación bidireccional entre un sistema físico y un robot virtual
Autor: Alberto Quiñonero González

Grado en Ingeniería de Computadores

Dirección:

Javier García Martín

Abstract

This project focuses on the development of a robotic control system that integrates sensors, communication modules, and actuators to enable interaction between a real system and a virtual simulation. Efficient communication between both environments is a key aspect, ensuring that data is transmitted accurately and in a coordinated manner.

To achieve this, a system was developed based on the integration of a control unit and a set of sensors and actuators connected to a Raspberry Pi. This setup enables real-world data collection and its transmission to a digital twin simulated in Webots. The virtual robot also sends information to the physical robot in order to perform actions in a coordinated manner. The project was implemented using Python for the simulation environment and C for hardware-level control. The system handles inputs from components such as potentiometers, ultrasonic sensors, buttons, and joysticks, while also managing outputs like Leds and a servomotor. Structured testing methodologies—including unit and integration tests—were applied to validate the system's reliability and ensure consistent synchronization between the physical and virtual environments.

As a result, a functional system was obtained, capable of integrating the real system with its virtual environment in a synchronized manner. This process helped improve skills in programming, hardware control, and robotic system validation, reinforcing the importance of structured development and optimizing communication between devices.

Resumen

Este proyecto se centra en el desarrollo de un sistema de control robótico que integra sensores, módulos de comunicación y actuadores para permitir la interacción entre un sistema real y una simulación virtual. La comunicación eficiente entre ambos entornos es un aspecto clave, asegurando que los datos se transmitan de manera precisa y coordinada.

Para ello, se desarrolló un sistema basado en la integración de una base de mandos y un conjunto de sensores y actuadores conectados a una Raspberry Pi, que permite capturar información del entorno físico y transmitirla a un gemelo digital simulado en Webots. Por su parte, el robot virtual también enviará información al robot real para tomar acciones de forma coordinada. La programación se llevó a cabo utilizando Python para la simulación y C para el control del hardware. El sistema es capaz de gestionar entradas desde componentes como potenciómetros, sensores ultrasónicos, botones y joysticks, así como controlar dispositivos como Leds y un servomotor. Para garantizar su correcto funcionamiento, se aplicaron metodologías de prueba estructuradas, incluyendo pruebas unitarias e integradas, que permitieron validar la fiabilidad del sistema y su sincronización con el entorno virtual.

Como resultado, se obtuvo un sistema funcional capaz de integrar el sistema real con su entorno virtual de manera sincronizada. Este proceso permitió mejorar habilidades en programación, control de hardware y validación de sistemas robóticos, reforzando la importancia del desarrollo estructurado y la optimización de la comunicación entre dispositivos.

ÍNDICE

Lista de Figuras

Lista de Tablas

1. INTRODUCCIÓN

1.1. Motivación

El avance de la tecnología ha permitido el desarrollo de nuevas herramientas que integran el mundo físico con el digital, destacando entre ellas los **gemelos digitales [1]**. Un gemelo digital es una representación virtual de un sistema físico que replica su comportamiento en tiempo real, permitiendo simular, monitorear y optimizar el funcionamiento de dicho sistema. Estos modelos virtuales han encontrado aplicaciones en múltiples campos, como la ingeniería, la industria y la investigación científica, debido a su capacidad para reducir riesgos, prever fallos y optimizar procesos sin necesidad de intervenir directamente en el sistema físico.

En este proyecto de Trabajo de Fin de Grado, se busca explorar la utilidad de los gemelos digitales como una plataforma experimental capaz de replicar el comportamiento de un sistema físico. En concreto, se ha diseñado un prototipo que combina un sistema físico y su correspondiente gemelo digital en un entorno de simulación, permitiendo una **comunicación bidireccional [2]** entre ambos. Este enfoque tiene como objetivo demostrar cómo los gemelos digitales pueden ser empleados no solo para replicar movimientos, sino también para mejorar la seguridad, controlar interacciones con el entorno y facilitar la sincronización inicial entre los sistemas.

Los gemelos digitales son herramientas que, además de reducir costes y riesgos en la experimentación, ofrecen la posibilidad de trabajar en plataformas virtuales antes de llevar los resultados al entorno físico. Este aspecto resulta crucial cuando se trata de probar sistemas complejos en los que intervienen componentes mecánicos, electrónicos y de software, como el desarrollado en este proyecto.

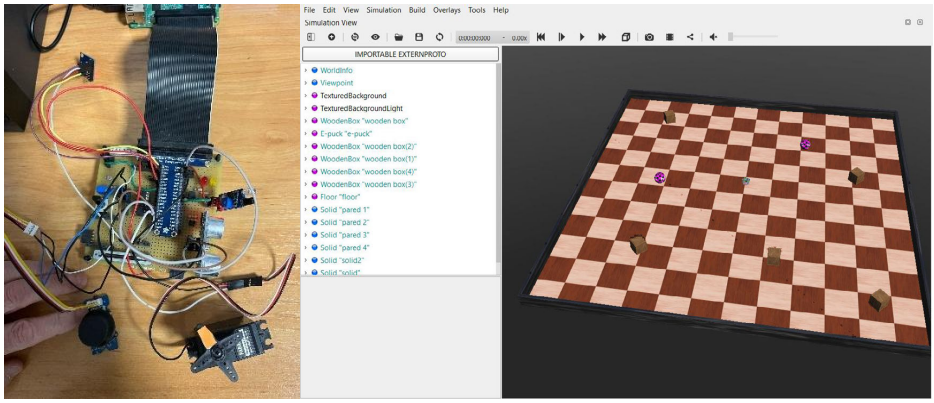
1.2. Descripción del sistema robótico físico

El objetivo principal de este proyecto es desarrollar un prototipo básico de gemelo digital que permita sincronizar el comportamiento entre un sistema físico y un modelo virtual mediante una comunicación bidireccional. Con este sistema, el robot virtual no solo replicará los movimientos del sistema físico, sino que también permitirá realizar pruebas de detección de objetos y respuestas de seguridad en el entorno digital, enviando señales al sistema físico para replicar dichas acciones.

Para ello, se han planteado los siguientes objetivos específicos:

- Diseñar y construir un sistema físico equipado con un sistema de control intuitivo que permita monitorear y ajustar su velocidad, dirección y posición inicial.
- Implementar una **plataforma virtual [3]** en **Webots [4]** que emule el comportamiento del robot físico y que permita realizar simulaciones de manera precisa.
- Establecer un sistema de comunicación bidireccional entre ambos robots, de forma que los eventos detectados en el entorno físico o virtual se reflejen en el sistema opuesto.
- Explorar la utilidad del prototipo como un ejemplo práctico de gemelo digital, validando su capacidad para replicar movimientos, detectar objetos y responder a estímulos externos de manera coordinada.

1.3.Ejemplo de caso de uso



Figuras 1a y 1b: Base de mandos y mundo

El funcionamiento del sistema comienza cuando el operador interactúa con la base de mandos (figura 1a), utilizando sus distintos componentes para dirigir el movimiento del robot físico. A través del potenciómetro, botones y otros controles, se puede ajustar la dirección, la velocidad y las pausas del robot.

Paralelamente, el gemelo digital del robot, representado en el entorno virtual de Webots (figura 1b), replica cada uno de los movimientos ejecutados por el robot real. Esta sincronización permite observar en pantalla el comportamiento del sistema como si se tratara de una simulación en tiempo real.

Si alguno de los dos robots—real o virtual—detecta un obstáculo, el sistema reacciona de forma coordinada. Por ejemplo, si el robot virtual identifica un objeto en su trayectoria, enviará esa información al robot físico, que a su vez tomará las medidas necesarias para evitar la colisión.

Gracias a este diseño, ambos robots pueden actuar de forma conjunta y coherente, compartiendo información para adaptarse al entorno y realizar movimientos sincronizados. Esto demuestra la utilidad del sistema tanto para entornos de prueba como para aplicaciones reales donde se requiere una interacción constante entre un sistema físico y su gemelo virtual.

1.4. Metodología

Para alcanzar los objetivos planteados en este proyecto, se seguirá una metodología estructurada en las siguientes etapas:

1. Diseño del sistema físico y virtual:
    - Definir los componentes del sistema físico (sensores, actuadores, controladores, etc.) y su funcionalidad específica.
    - Diseñar en Webots un robot virtual que replique la configuración del sistema físico y permita simular su comportamiento en un entorno controlado.
2. Desarrollo del hardware y software del sistema físico:
    - Diseñar el sistema físico añadiendo los componentes.
    - Programar su sistema de control para manejar la dirección, velocidad y reacciones a estímulos externos, como la detección de objetos.
3. Establecimiento de la comunicación bidireccional:
    - Implementar un sistema de comunicación que permita transmitir datos entre el sistema físico y el robot virtual.
    - Asegurar que el robot virtual reciba las señales del sistema físico y pueda replicar sus movimientos, enviando a su vez señales de vuelta al sistema físico para desencadenar respuestas, como la activación de Leds o el frenado.
4. Pruebas de sincronización:
    - Realizar pruebas iniciales para alinear la posición y estado del sistema físico con el robot virtual, asegurando una correspondencia exacta entre ambos.
    - Ajustar parámetros para garantizar una comunicación estable y en tiempo real.
5. Validación del sistema:
    - Evaluar la capacidad del sistema para cumplir con los objetivos planteados, incluyendo la replicación de movimientos, la detección de objetos y la activación de respuestas de seguridad.
    - Realizar simulaciones en el entorno virtual para observar el comportamiento del sistema en distintos escenarios y verificar que las acciones realizadas en el entorno físico y el virtual sean coherentes.
6. Documentación y análisis de resultados:
    - Recopilar los datos obtenidos durante las pruebas y simulaciones.
    - Analizar los resultados para identificar posibles mejoras o ajustes al sistema, destacando los beneficios del gemelo digital implementado.
    - Esta metodología asegura un enfoque sistemático para el desarrollo del prototipo, priorizando tanto la funcionalidad técnica como la validación experimental del sistema.

2. CONTEXTO

2.1. Gemelos digitales

- **Situación actual**

  Los gemelos digitales se han consolidado como una de las tecnologías más prometedoras en el ámbito de la simulación y el modelado de sistemas físicos. Estas representaciones virtuales permiten replicar con gran precisión el comportamiento de dispositivos reales en tiempo real, lo que ha generado un interés significativo en sectores como la industria 4.0, la aviación, la automoción y la medicina. Gracias a la conectividad entre el mundo físico y el digital, los gemelos digitales no solo reflejan el estado actual del sistema, sino que también facilitan la predicción de comportamientos futuros y la optimización de procesos.

  Actualmente, su implementación se apoya en avances como la computación en la nube, los **sistemas ciberfísicos [5]** y la inteligencia artificial. A través de sensores, actuadores y sistemas de comunicación, los datos recolectados por el sistema físico se transfieren al modelo virtual, garantizando una sincronización precisa entre ambos.

- **Ejemplos desarrollados**

  Existen diversos ejemplos destacados del uso de gemelos digitales en la práctica. En la industria aeroespacial, empresas como **Boeing [6]** utilizan gemelos digitales para simular el comportamiento de sus motores y realizar mantenimientos predictivos. En el sector de la automoción, **Tesla [7]** emplea esta tecnología para mejorar el rendimiento de sus vehículos, monitoreando el estado de los componentes en tiempo real y adaptando los modelos digitales según las condiciones del mundo real. En la medicina, los gemelos digitales han permitido el desarrollo de simulaciones de órganos humanos, lo que facilita diagnósticos más precisos y personalizados.

  En el contexto académico y experimental, los gemelos digitales son herramientas clave para la investigación y el desarrollo de prototipos funcionales, ya que permiten probar y ajustar sistemas en entornos controlados antes de aplicarlos en la práctica. Este enfoque es el que se aplica en este proyecto, donde el gemelo digital de un sistema físico actúa como una herramienta para garantizar su correcta funcionalidad y replicar su comportamiento en un entorno virtual.

2.2.Material de laboratorio

- **Raspberry pi**

  La **Raspberry Pi [8]** es una plataforma compacta y versátil que ha sido utilizada en este proyecto para gestionar la conexión y control de los elementos físicos del robot real. Gracias a su capacidad para interactuar con sensores, actuadores y otros dispositivos a través de sus **puertos GPIO [9]**, ha permitido integrar de manera eficiente los componentes como los LEDs, servomotores, y sensores, coordinando sus operaciones en tiempo real.

- **Webots**

  Webots es un entorno de simulación 3D utilizado en este proyecto para modelar y simular el comportamiento del robot virtual. Con su capacidad para replicar entornos físicos realistas y su compatibilidad con sensores y actuadores virtuales, Webots ha permitido probar y ajustar el funcionamiento del robot en un entorno controlado antes de implementarlo en el sistema físico, facilitando la validación y el desarrollo iterativo del sistema.

  Ofrece una amplia gama de modelos predefinidos y la posibilidad de personalizar las propiedades físicas y mecánicas de los robots. En este proyecto, Webots se utiliza para replicar las señales enviadas por el sistema físico, simular su respuesta a estímulos y enviar señales de retroalimentación al sistema físico.

- **Lenguajes de programación**
  - **C [10]**: C es un lenguaje de programación de bajo nivel ideal para el control de sistemas embebidos, como los microcontroladores usados en este proyecto. Su eficiencia y control directo sobre el hardware permiten gestionar los sensores y actuadores del robot real, garantizando respuestas rápidas y manejo preciso de los recursos. Además, facilita la integración de bibliotecas para el control de periféricos como ultrasonidos o motores.
  - **Python [11]**: Python es un lenguaje de alto nivel, ideal para simular y controlar el comportamiento del robot virtual de este proyecto. Su sintaxis simple y amplia gama de bibliotecas permiten una rápida implementación de lógica compleja y comunicación de datos con el robot real, facilitando la interacción entre ambos sistemas sin preocuparse por detalles de bajo nivel.

3. SISTEMA DESARROLLADO

3.1. Componentes generados

Base de mandos

La base de mandos es el centro de control del sistema, encargada de coordinar la comunicación entre el sistema físico y el robot virtual. Gestiona la recepción y el envío de datos, permitiendo una interacción fluida entre ambos entornos. A través de esta base, se ejecutan los programas necesarios para interpretar la información de los sensores y enviar comandos a los diferentes módulos del sistema.

Sistema Físico

El sistema físico está compuesto por distintos sensores y módulos electrónicos conectados a la Raspberry Pi. Su función no se limita únicamente a capturar datos del entorno, sino que también recibe y procesa información del robot virtual para ajustar su comportamiento en tiempo real. Esta comunicación bidireccional permite que el sistema físico reaccione a cambios en la simulación, garantizando una integración coherente entre ambos entornos.

Robot Virtual

El robot virtual, desarrollado en Webots, es una réplica digital del sistema físico en un entorno de simulación 3D. Recibe datos de la Raspberry Pi para ejecutar acciones y reflejar el comportamiento esperado en la simulación, pero también envía información de vuelta al sistema físico, permitiendo ajustes y validaciones en tiempo real. Esta interacción continua facilita la depuración del código y la optimización del sistema sin depender exclusivamente del hardware.

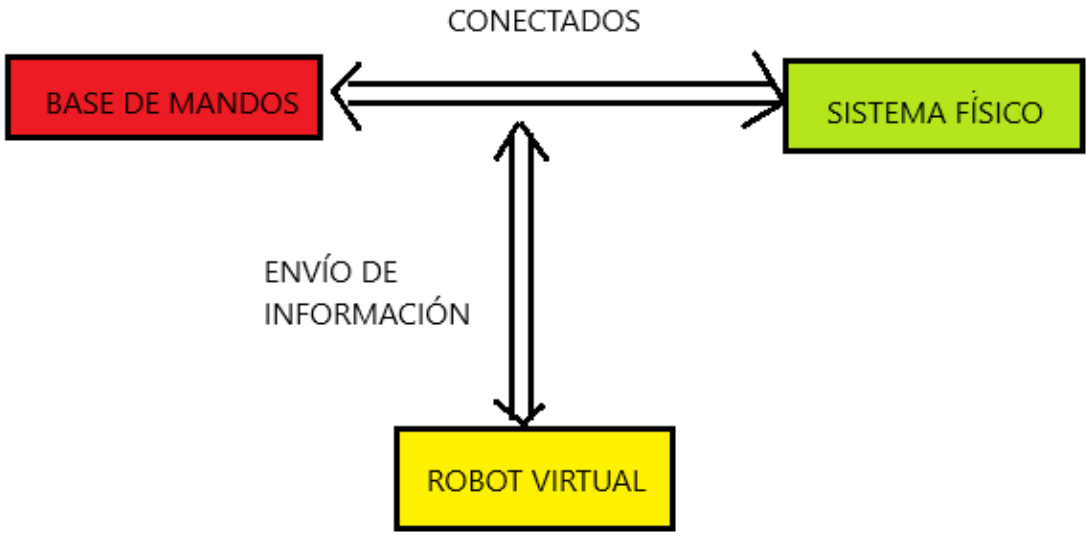


Figura 2: Componentes generados

3.2. Funcionalidades generales

El sistema desarrollado se compone de una base de mandos (BM), un sistema físico (SF) y su gemelo digital en el entorno virtual (RV), que interactúan mediante una comunicación bidireccional. Cada componente desempeña un rol específico en la simulación y el control, garantizando una sincronización precisa entre todos los elementos. Este sistema ha sido implementado previamente en la asignatura Sistemas en Tiempo Real de la ETSISI de la Universidad Politécnica de Madrid, como parte del desarrollo práctico de dicha materia. Funcionalidades generales:

1. **Base de Mandos (BM):**
    - Captura las entradas del operador a través de dispositivos de control, como el volante y el potenciómetro.
    - Envía las órdenes de movimiento y frenado al sistema físico y, posteriormente, al robot virtual.
    - Recibe información desde el robot virtual y el sistema físico para garantizar un control adecuado.
2. **Sistema físico (SF):**
    - Recoge información del entorno a través de sus sensores (volante, ultrasonidos, medidor de velocidad y sensor de objetos).
    - Responde a las órdenes de la base de mandos y transmite datos al robot virtual.
    - Gestiona los avisos de proximidad mediante los Leds amarillo y rojo, según los datos proporcionados por el sensor de ultrasonidos y el RV.
3. **Robot virtual (RV):**
    - Replica el comportamiento del sistema físico en el entorno simulado, incluyendo los movimientos y la interacción con objetos virtuales.
    - Detecta obstáculos en el entorno virtual y envía señales al sistema físico y a la base de mandos para activar los avisos correspondientes o frenar si es necesario.
    - Proporciona al operador una visualización en tiempo real del comportamiento del sistema y de las respuestas del gemelo digital.

3.3. Componentes del sistema

- **Botón**: Dispositivo empleado para dos funciones, colocar el servomotor con la misma orientación que el robot virtual, para tener una mayor orientación de este, y como freno de mano, mientras están los robots en movimiento, al pulsar el botón estos se frenaran durante 5 segundos para así tener tiempo y poder mover la posición del robot.

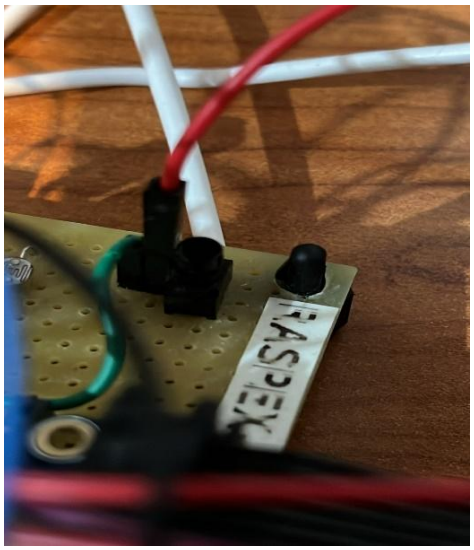


Figura 3: Botón

- **Potenciómetro rotatorio**: Dispositivo empleado como volante, este dispositivo manda valores entre 0 y 1024 y en el código se transforman a valores del 0 al 10, tras esto ese valor se manda al robot virtual y este comienza a girar hacia el lugar indicado hasta que recibe la orden de dejar de girar.

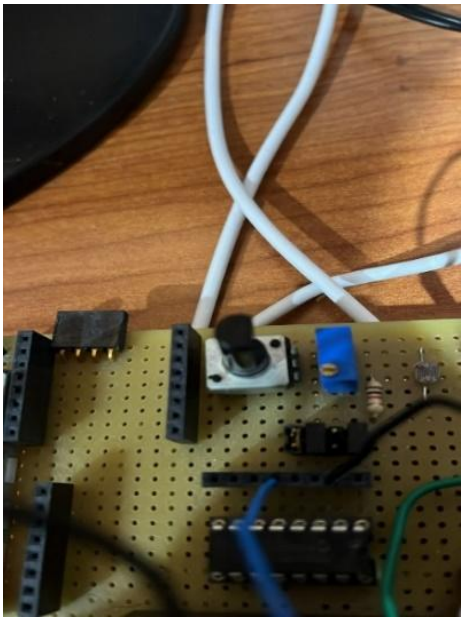


Figura 4: Potenciómetro rotatorio

- **Potenciómetro deslizable:** Dispositivo utilizado para ajustar la velocidad del robot, al igual que el potenciómetro rotatorio, este dispositivo manda valores entre 0 y 1024 y en el código se transforman a valores del 0 al 10, tras esto ese valor se manda al robot virtual y este se mueve a la velocidad indicada por este dispositivo, siendo 0, 1 y 2 valores de velocidad negativa, 3 velocidad nula, o sea que no se mueven los robots, y de 4 hasta 10 diferentes velocidades positivas.

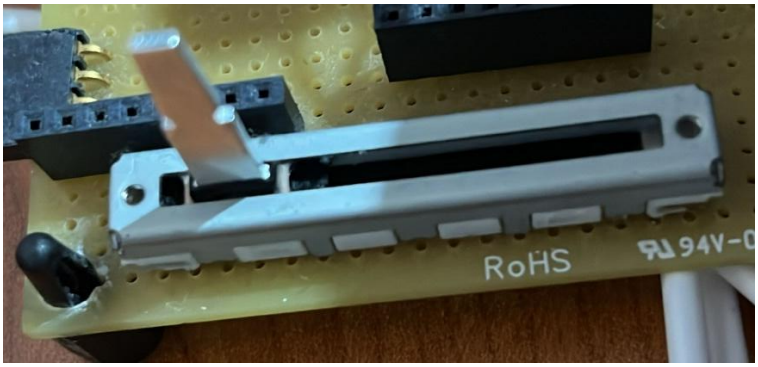


Figura 5: Potenciómetro deslizable

- **Sensor infrarrojo:** Dispositivo empleado, al igual que el botón, como un freno de mano, mientras están los robots en movimiento, al acercar un objeto o la mano, estos se frenarán durante 5 segundos para así tener tiempo y poder mover la posición del robot. Está implementado ya que tener que pulsar el robot puede resultar un poco incomodo en ciertas situaciones y este dispositivo puede resultar más sencillo.

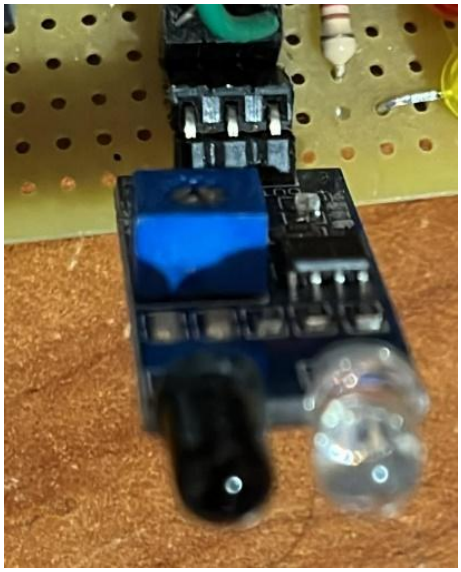


Figura 6: Sensor Infrarrojos

- **Ultrasonidos:** Dispositivo implementado para medir la distancia con un objeto frente a él, cuando detecta un objeto a menos de 15 centímetros manda un aviso para encender el led amarillo que mas adelante será explicado, y cuando detecta un objeto a menos de 8 centímetros manda otro aviso para que se encienda el led rojo y además se frene en seco de la misma forma que si detectara el freno de mano.

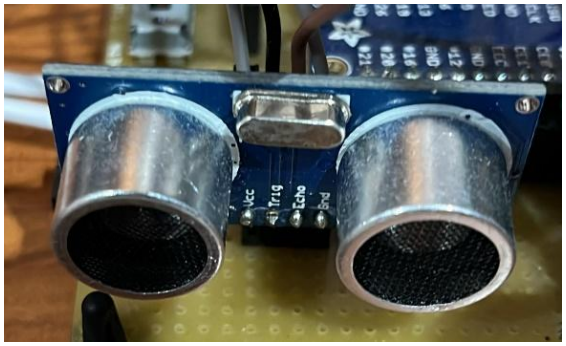


Figura 7: Ultrasonidos

- **Joystick:** Dispositivo utilizado para poder sustituir al potenciador rotatorio y al potenciador deslizante, cumple exactamente la misma función que estos dos juntos. Para poder utilizarlo hay que cambiar la configuración del webots y hacer que reciba los valores 4 y 6 recibidos de los datos en vez de los valores 0 y 2 que es como esta implementado de manera normal. Este se mueve en un plano xy bidireccional, con el cual el eje de las x se utiliza para controlar los giros del robot y el eje y para controlar su velocidad.

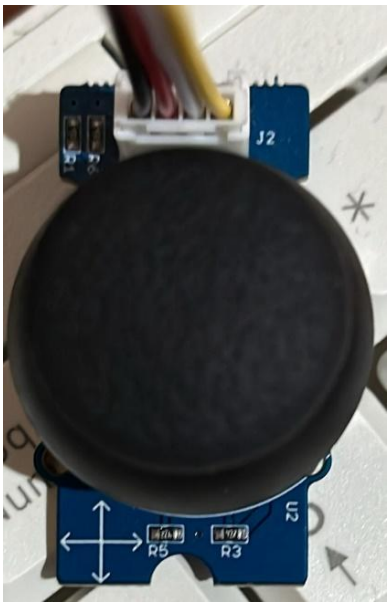


Figura 8: Joystick

- **Leds:** Dispositivos LED de colores amarillo y rojo, han sido utilizados para dar un aviso visual de los objetos cercanos, tanto en el mundo real con el ultrasonido, como en el mundo virtual con el robot de webots, en ambos cumple la misma función, cuando detecta un objeto cercano se enciende el led amarillo, y cuando detecta un objeto muy próximo (a escasos centímetros) se enciende el led rojo.

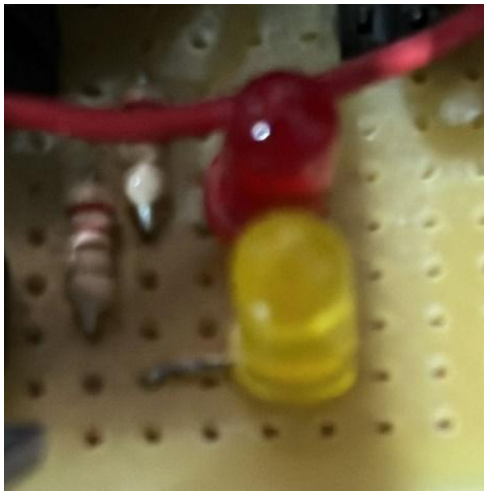


Figura 9: Leds

- **Servomotor:** Dispositivo empleado para reflejar la orientación del sistema físico, manteniendo la misma posición que el robot virtual en Webots. Cuenta con un aspa de color naranja que permite visualizar su alineación, asegurando que siga de manera síncrona todos los movimientos del robot virtual.

Figura 10: Servomotor

3.4.Requisitos

- **Requisitos funcionales**
  - **Sistema Físico:**

    1: Tanto el sistema físico como el robot virtual deberán recibir las señales del volante y ajustarse en tiempo real para orientar el servomotor según la posición indicada.

    2: Los Leds deben activarse según las lecturas del sensor de ultrasonidos:

    - LED amarillo: cuando se detecte un objeto a distancia moderada.
    - LED rojo: cuando el objeto esté muy cerca y el robot deba frenar.

    3: El botón de freno debe detener el movimiento del robot inmediatamente al ser activado.

    4: El sistema debe detectar obstáculos a diferentes distancias y reaccionar de acuerdo con su proximidad, proporcionando avisos visuales y enviando la información al robot virtual.

    5: Cuando un obstáculo se detecta a corta distancia, el sistema físico debe ejecutar una maniobra automática de giro a la derecha para evitar la colisión.

    6: El sensor de velocidad debe proporcionar datos precisos al robot virtual para replicar los movimientos.

  - **Robot Virtual**

    1: El robot virtual debe replicar fielmente los movimientos y orientación del robot real en el entorno simulado.

    2: Debe detectar obstáculos en la simulación y enviar señales al sistema físico para que éste tome medidas preventivas antes de que se produzca una colisión.

    3: En caso de detectar un obstáculo antes que el sistema físico, el robot virtual debe anticiparse y enviar una orden para que éste reaccione adecuadamente.

    4: Debe visualizar en tiempo real el estado del robot real y su entorno en la simulación, mostrando posiciones, velocidades y alertas.

  - **Interacción y comunicaciones**

    1: El robot real debe enviar datos al robot virtual sobre orientación, velocidad y estado de los sensores, de forma continua y en tiempo real.

    2: El robot virtual debe enviar retroalimentación al robot real para activar respuestas físicas (como Leds o freno) basándose en el entorno simulado.

    3: La comunicación entre ambos robots debe ser bidireccional, precisa y con mínima latencia para garantizar una sincronización constante.

- **Requisitos generales**

  1: La sincronización entre el sistema físico y el robot virtual debe mantenerse en todo momento para garantizar una simulación precisa.

  2: El sistema debe permitir una interacción intuitiva por parte del operador, asegurando facilidad de uso en la interpretación de señales y alertas.

  3: Debe implementar mecanismos de seguridad que aseguren la respuesta inmediata de los robots a posibles colisiones tanto en el entorno físico como virtual.

  4: La detección y gestión de obstáculos debe ser eficiente, asegurando una reacción oportuna para evitar impactos y mejorar la fluidez del sistema.

  5: El sistema debe ser escalable para permitir la integración de futuras mejoras o nuevos sensores.

  6: La solución desarrollada debe ser fiable, asegurando que no haya pérdida de datos o inconsistencias entre los robots en operación prolongada.

4. MODELADO

4.1. Diagrama de casos de uso

Este diagrama de casos de uso ilustra las interacciones clave entre el usuario, la Raspberry y el robot virtual, destacando las funciones principales dentro de los límites del sistema.

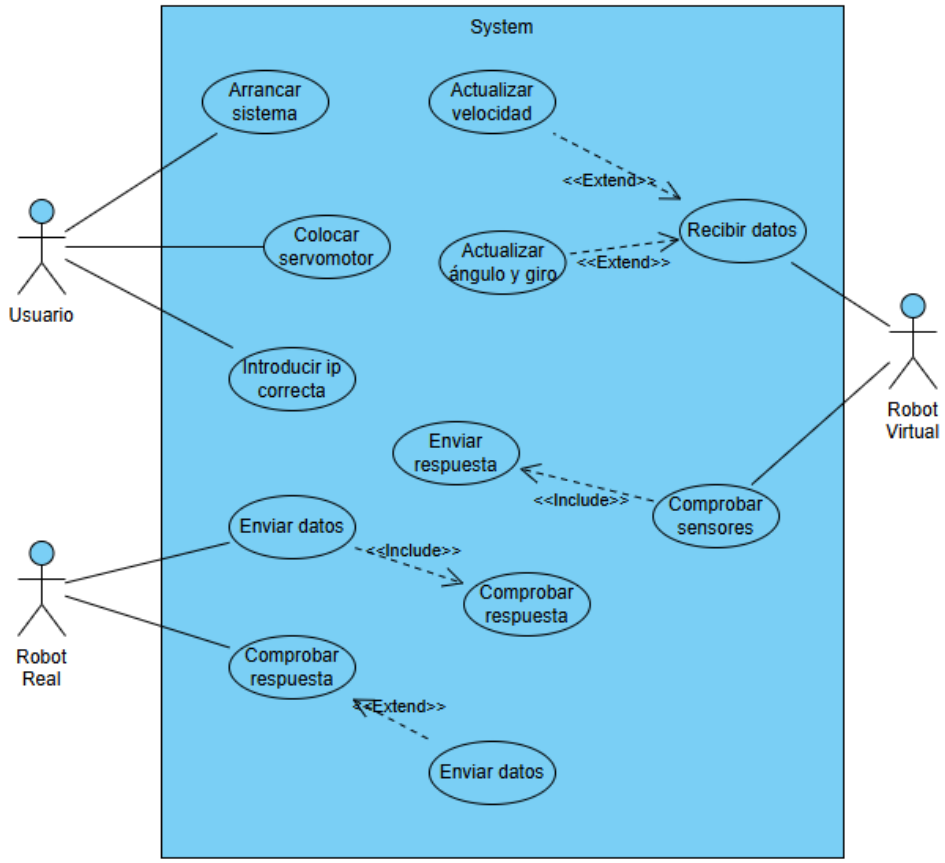


Figura 11: Diagrama de casos de uso

Actores involucrados:

- Usuario: Es quien interactúa directamente con el sistema, iniciando los procesos y verificando los datos.
- Robot Real: Se encarga de recibir, procesar y enviar datos hacia los componentes físicos del robot.
- Robot Virtual: Este actúa como una representación virtual del robot real, ejecutando comandos simulados y permitiendo ajustes en parámetros como la velocidad y la dirección.

Casos de uso principales:

- Arrancar el sistema: El usuario inicia el sistema, que conecta y activa los diferentes componentes tanto del robot real como del robot virtual.
- Colocar servomotor: El usuario posiciona el servomotor para calibrarlo y hacerlo coincidir con el robot virtual, un paso fundamental antes de continuar con el control del robot.
- Introducir **IP [12]** correcta: El sistema requiere la introducción de una dirección IP válida para garantizar la correcta conexión con el robot virtual.
- Enviar datos al Robot Virtual: La Raspberry se encarga de enviar los datos que recibe del usuario hacia el Robot Virtual, siendo este un proceso esencial para el funcionamiento del robot.
- Recibir datos del Robot Real: El robot virtual recibe la información enviada por el sistema, permitiendo que se actualicen sus parámetros en tiempo real, como la velocidad y el ángulo de giro. Este caso de uso tiene una extensión que permite actualizar la velocidad y los ángulos de giro del robot.
- Comprobar sensores: El sistema verifica que los sensores del robot estén funcionando correctamente antes de enviar una respuesta al usuario o ejecutar acciones.
- Enviar respuesta: La Raspberry envía al usuario la confirmación de que los datos fueron procesados correctamente, asegurando que el flujo de información sea continuo.
- Comprobar respuesta: El sistema realiza una verificación interna para comprobar que los datos enviados por los sensores son válidos. Si es necesario, también puede incluir la comprobación de los componentes de salida.

  Relaciones entre casos de uso:

- <<include>> Enviar respuesta: El proceso de enviar datos siempre incluye la acción de recibir una respuesta, ya que esta es fundamental para confirmar el correcto envío y ejecución de los datos.
- <<extend>> Comprobar respuesta: Este caso de uso se extiende cuando es necesario comprobar no solo los datos, sino también los componentes de salida del robot.

En resumen, este diagrama describe cómo se integran las acciones del usuario con las respuestas de los componentes del sistema, tanto físicos como virtuales, permitiendo un ciclo de control y retroalimentación que garantiza el correcto funcionamiento del robot.

4.2. Diagrama de secuencia

Este diagrama de secuencia representa las interacciones cronológicas entre el Usuario, el Robot real y el Robot virtual, mostrando cómo se comunican los distintos actores del sistema para ejecutar las funciones relacionadas con el control del robot.

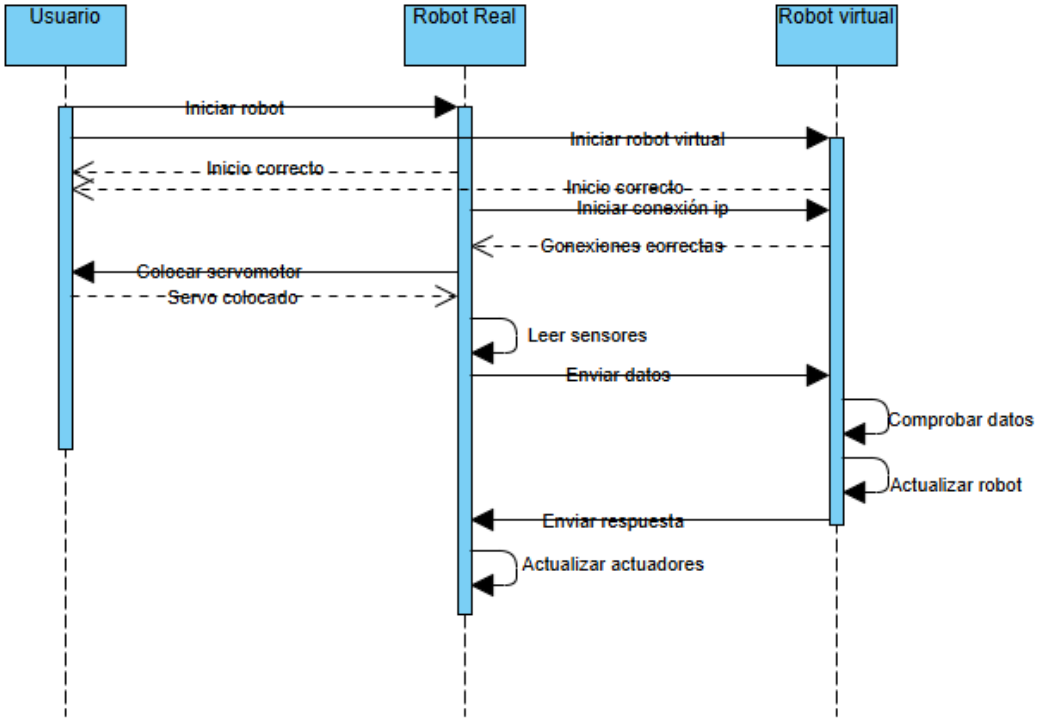


Figura 12: Diagrama de secuencia

Flujo de la secuencia.

Iniciar robot:

- El Usuario comienza el proceso enviando una solicitud para iniciar el robot. Este comando es enviado tanto al Robot real como al Robot virtual.
- El Robot real responde confirmando que el inicio ha sido correcto.
- Simultáneamente, el Robot virtual establece una conexión IP, confirmando al final si la conexión ha sido correcta.

Colocar servomotor:

- Después de la verificación de inicio, el Usuario procede a colocar el servomotor en el robot real.
- Una vez que el servomotor ha sido colocado correctamente, el robot devuelve una confirmación de que el servomotor está en la posición correcta.

Leer sensores: El Robot real realiza una comprobación de los dispositivos conectados para asegurar que todo esté en orden antes de proceder a enviar datos hacia el robot virtual.

Enviar datos: Una vez comprobados los dispositivos, el Robot real envía los datos al Robot virtual para su procesamiento. El robot virtual verifica estos datos:

Comprobar datos: El robot virtual verifica si los datos recibidos son correctos.

Actualizar robot: Si los datos son correctos, el robot virtual procede a actualizar su estado (por ejemplo, ajustando el ángulo o velocidad).

Enviar respuesta: El Robot real envía una respuesta de vuelta al Usuario confirmando que los datos han sido procesados y que los dispositivos han sido actualizados.

Actualizar actuadores: Finalmente, el Robot real actualiza los dispositivos en función de la respuesta obtenida, completando el ciclo de interacción entre el usuario, el robot real y el robot virtual.

El diagrama de secuencia describe el flujo de control y comunicación entre el usuario, el sistema físico y su contraparte virtual. El proceso comienza con la activación de ambos robots, pasa por la colocación del servomotor y la verificación de dispositivos, y culmina con el envío y actualización de datos. Esto asegura que los datos del sistema físico se reflejen correctamente en el robot virtual y que el usuario reciba una confirmación de la correcta ejecución de las tareas.

4.3. Diagrama de flujo

Este diagrama representa el proceso de activación y control de robots reales y virtuales, destacando varios puntos donde puedes decidir finalizar la prueba manualmente mediante Ctrl+C. El bloqueo no ocurre automáticamente salvo en el caso de un error en la IP, ya que tú decides cuándo interrumpir el proceso.

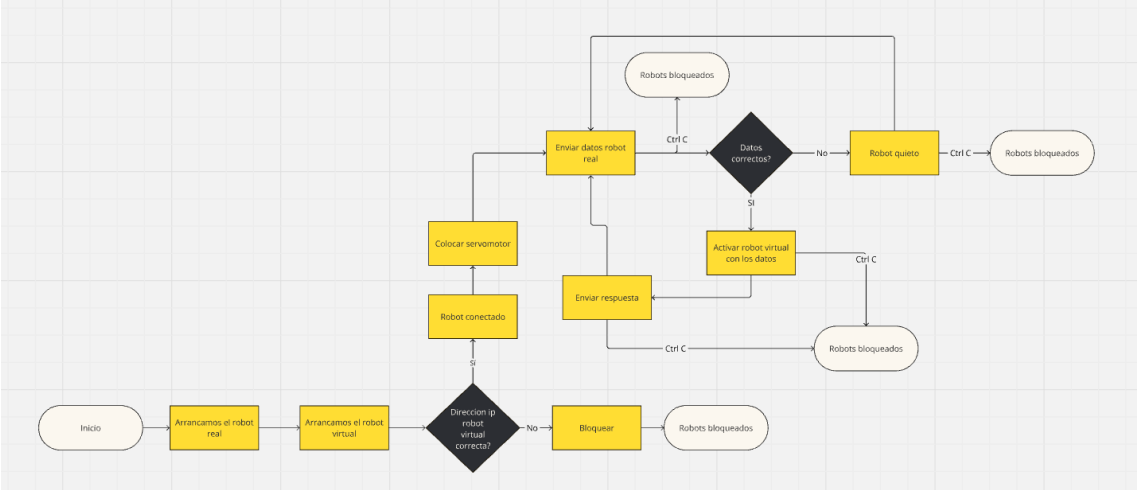


Figura 13: Diagrama de flujo

1. Inicio del proceso

El flujo comienza con dos posibles acciones:

- Arrancar el robot real.
- Arrancar el robot virtual.

Ambas acciones inicializan los robots en sus respectivos entornos.

2. Verificación de la dirección IP del robot virtual

Se verifica si la dirección IP del robot virtual es correcta:

- Si la IP es incorrecta, los robots se bloquean automáticamente y el flujo termina.
- Si la IP es correcta, el proceso continúa hacia los pasos siguientes.

3. Colocación del servomotor y conexión del robot

Aquí se realiza la colocación del servomotor y se verifica si el robot está conectado:

- Si el robot no está conectado, puedes detener el proceso con Ctrl+C.
- Si el robot está conectado, el flujo avanza y se procede al envío de datos al robot real.

4. Envío de datos al robot real

Se envían los datos al sistema físico. En este punto, se verifica si los datos son correctos:

- Si los datos no son correctos, tienes la opción de corregirlos y reenviarlos sin necesidad de bloquear el robot.
- Si decides no corregirlos o si persisten los errores, puedes finalizar el proceso manualmente con Ctrl+C.

- Si los datos son correctos, el flujo avanza y se activa el robot virtual con esos datos.

5. Activación del robot virtual

Tras la validación de los datos, se activa el robot virtual, que sincroniza su operación con el robot real. Luego se envía una respuesta confirmando la correcta operación de ambos robots.

6. Bloqueo manual de los robots

En cualquier punto del flujo, salvo en la verificación de la IP, puedes interrumpir el proceso de manera intencionada mediante Ctrl+C, bloqueando los robots y dando por finalizada la prueba. El bloqueo manual no responde a errores operativos, sino a tu decisión de concluir el experimento.

En resumen, el diagrama refleja que puedes corregir errores de datos y reenviar la información sin necesidad de detener el sistema. El bloqueo de los robots ocurre de manera manual, salvo cuando hay un fallo en la verificación de la IP.

4.4. Disposición física de los componentes

En esta configuración física, se ha conectado una Raspberry Pi con una placa de desarrollo central que controla varios componentes.

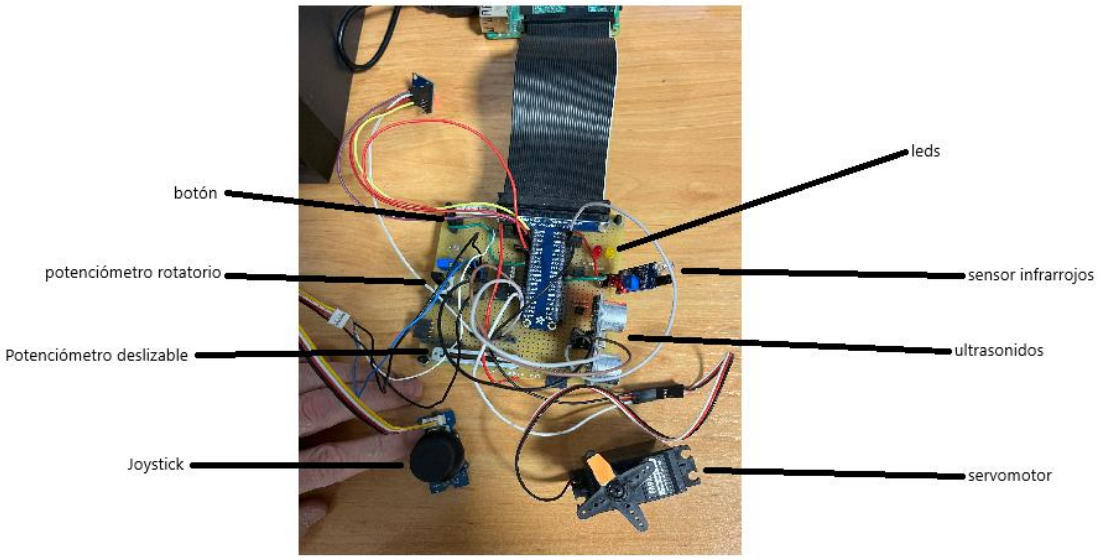


Figura 14: Disposición física de los componentes

A la izquierda, se encuentra un joystick analógico conectado a la placa para manejar el movimiento. A la derecha, se observa un servo motor, que recibe órdenes desde la placa para realizar acciones mecánicas. Los múltiples cables permiten la comunicación entre los distintos módulos, y toda la configuración está preparada para interactuar con el software de control, ya sea para simulación o tareas físicas.

4.5. Conexiones físicas entre los componentes

En este apartado, presento una descripción detallada de las conexiones del sistema físico, utilizando los GPIOs de la Raspberry Pi. Me baso en el esquema visual para ilustrar las relaciones entre los diferentes componentes. La dirección de las flechas indica si el dispositivo es de entrada (cuando la flecha apunta hacia el dispositivo) o de salida (cuando la flecha apunta hacia la Raspberry Pi).

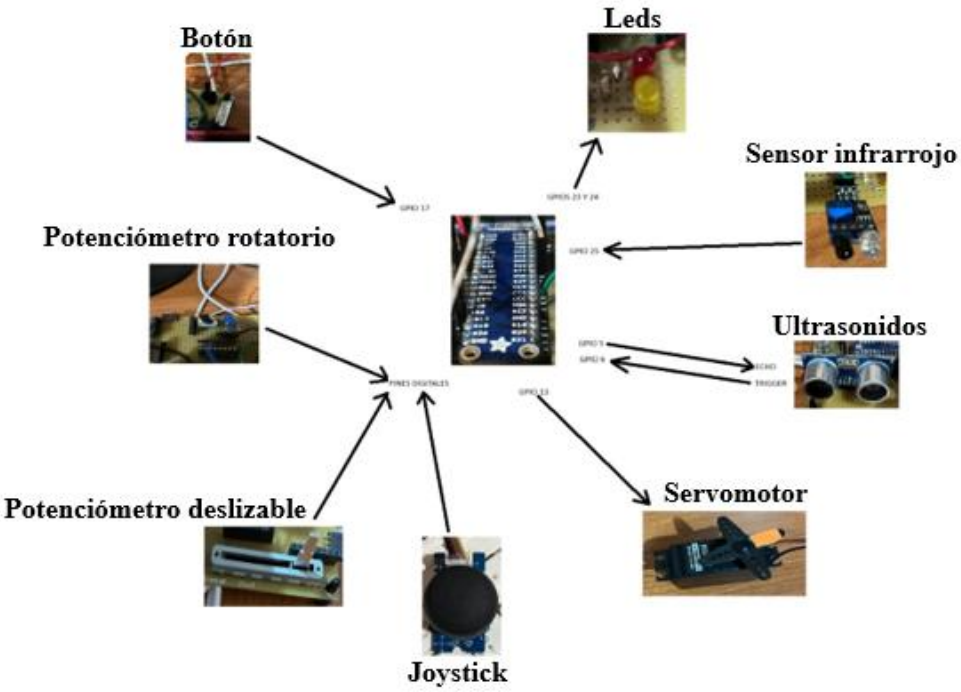


Figura 15: Conexiones físicas entre los componentes

1. Botón (GPIO 17)

El botón está conectado al GPIO 17. La flecha apunta hacia la Raspberry Pi, lo que indica que el botón actúa como una entrada. Esto significa que, al presionar el botón, se enviará una señal a la Raspberry Pi, lo que permitirá al sistema recibir comandos o ejecutar funciones específicas según el estado del botón.

2. Leds (GPIO 23 y GPIO 24)

Estos dos Leds están conectados a los GPIO 23 y GPIO 24, respectivamente. Como la flecha apunta hacia los Leds, estos funcionan como salidas. La Raspberry Pi controla los Leds, encendiéndolos o apagándolos en respuesta a objetos que se detecten, dependiendo de la distancia a la que se encuentre el objeto (tanto del robot real como del robot virtual) el sistema encenderá el led amarillo si el objeto se encuentra a media distancia (10 centímetros) y led rojo si el objeto se encuentra a muy corta distancia (1 centímetro).

3. Sensores de proximidad infrarrojos (GPIO 25)

Los sensores infrarrojos están conectados al GPIO 25. Según el diagrama, la flecha apunta hacia la Raspberry Pi, lo que indica que estos sensores son entradas. Detectan la proximidad de objetos cercanos y envían la información a la Raspberry Pi para que el sistema pueda responder en consecuencia, por ejemplo, evitando colisiones.

4. Sensor ultrasonido (HC-SR04) - GPIO 5 (Echo) y GPIO 6 (Trigger)

Este sensor de ultrasonidos tiene dos pines principales:

- Trigger (GPIO 6): Funciona como salida. La Raspberry Pi envía un pulso desde este pin para activar la medición de distancia del sensor.
- Echo (GPIO 5): Funciona como entrada. Recibe el pulso de retorno, que permite calcular la distancia a un objeto basado en el tiempo que tardó en regresar el eco.

5. Servomotor (GPIO 13)

El servomotor está conectado al GPIO 13 y la flecha apunta hacia él, lo que significa que este pin actúa como salida. La Raspberry Pi envía señales al servomotor para controlar su posición, permitiendo realizar movimientos precisos de acuerdo con las instrucciones programadas. Este dispositivo es empleado para conocer la orientación del sistema físico, que también debe seguir la misma posición del robot virtual.

6. Pines analógicos (Joystick, potenciómetro deslizable y potenciómetro rotatorio)

Tres dispositivos de entrada están conectados a los pines digitales:

- Joystick: El joystick es una entrada que permite al usuario controlar diferentes parámetros del robot, como la dirección o la velocidad.
- Potenciómetro deslizable: Este es otro dispositivo de entrada que permite ajustar valores analógicos moviendo el deslizador, es utilizado para modificar la velocidad del robot.
- Potenciómetro rotatorio: Al igual que el deslizable, el potenciómetro rotatorio es una entrada que envía valores a la Raspberry Pi en función del ángulo de rotación, este es utlizado para cambiar la posición del robot.

En este diseño, los dispositivos de entrada como los botones, los sensores (infrarrojos, ultrasonido), el joystick y los potenciómetros envían información a la Raspberry Pi, la cual procesa esos datos para tomar decisiones. Luego, los dispositivos de salida como los Leds y el servomotor responden a esos datos con acciones, como cambios de posición, iluminación de Leds, o movimientos en el entorno.

Este sistema proporciona una interacción dinámica entre el hardware y el software, permitiendo que el robot responda a estímulos externos de manera eficiente y acorde a la lógica programada.

4.6. Disposición del mundo de webots

Este “mundo” que se está presentando en Webots es una simulación que ha sido diseñada cuidadosamente, donde se pueden apreciar distintos elementos interactuando en un entorno cuadriculado, muy parecido a un tablero de ajedrez. El propósito de esta simulación es crear un escenario en el que el robot móvil "**E-puck**" **[13]** pueda navegar y sortear obstáculos mientras interactúa con el entorno.

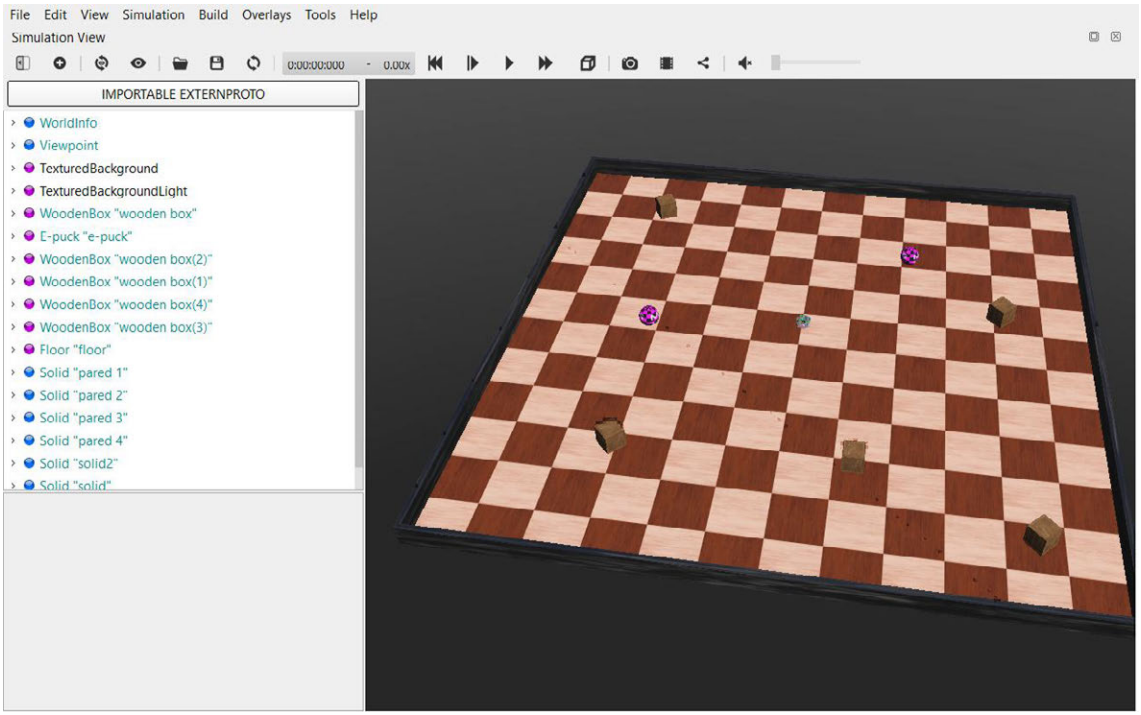


Figura 16: Disposición del mundo de webots

El suelo, que simula un tablero de ajedrez, está compuesto de cuadros marrones y claros alternados, lo que ayuda a proporcionar un contexto visual para el movimiento del robot. La cuadrícula no solo facilita la visualización del espacio, sino que también actúa como referencia para los desplazamientos y las trayectorias. El entorno cuenta con varios objetos distribuidos de manera dispersa que le añaden desafíos al robot, como cajas de madera y sólidos que forman las "paredes" del escenario, delimitando el espacio en el que se moverá el robot.

WorldInfo: Este nodo es fundamental, ya que define los parámetros globales del mundo, como el tiempo de simulación y la gravedad. En este caso, he configurado la simulación para que sea suficientemente precisa en tiempo real, permitiendo un control adecuado del robot y sus sensores.

Viewpoint: El nodo de la cámara define el punto de vista desde el que se observa la simulación. Para este mundo, la vista está ajustada desde una perspectiva ligeramente elevada y cenital, permitiendo observar de manera clara todo el entorno y el movimiento del robot a lo largo del tablero de ajedrez.

TexturedBackground y TexturedBackgroundLight: Estos dos nodos son responsables del fondo de la simulación, que utiliza texturas que simulan un entorno realista o neutro, dando la sensación de que el mundo está dentro de un espacio físico, como un laboratorio o una sala cerrada.

WoodenBox ("wooden box"): Hay varias cajas de madera en el mundo, que sirven como obstáculos estáticos para el robot. En total, he distribuido cinco cajas numeradas (1-5), que están posicionadas estratégicamente en diferentes puntos del escenario para desafiar la capacidad del robot para moverse y evitar colisiones. Estas cajas son elementos sólidos, lo que significa que no solo bloquean el paso, sino que también el robot puede interactuar físicamente con ellas, detectándolas a través de sus sensores.

Floor ("floor"): El suelo, como mencioné anteriormente, tiene un patrón de tablero de ajedrez. Esto no solo aporta estética a la simulación, sino que también es útil para visualizar el movimiento del robot y los obstáculos, ofreciendo una referencia clara de distancia y orientación en el espacio.

Solid ("pared 1", "pared 2", etc.): Las paredes (paredes 1 a 4) forman los límites del escenario, asegurándose de que el robot permanezca dentro del área de simulación. Están dispuestas alrededor del borde del tablero de ajedrez, creando una especie de recinto o arena donde el robot y los objetos interactúan. Además de estas paredes, también hay algunos sólidos adicionales que podrían representar barreras o puntos de referencia adicionales dentro del espacio.

En el centro de esta simulación se encuentra el robot E-puck, un pequeño robot móvil diseñado para simular entornos de navegación y aprendizaje en robótica. En este caso, el E-puck está equipado con sensores de proximidad que le permiten detectar objetos cercanos y ajustar su comportamiento en consecuencia. El E-puck está programado para moverse por el entorno, esquivando las cajas de madera y respondiendo a las barreras sólidas dentro del área simulada.

El E-puck es clave en este mundo, ya que todo el entorno ha sido diseñado para poner a prueba sus capacidades de navegación autónoma. A medida que avanza, puede ajustar su dirección y velocidad en función de los datos que recopila a través de sus sensores, permitiéndole tomar decisiones sobre cuándo girar o detenerse para evitar una colisión. Gracias a la implementación de su sistema de control y los sensores, el robot está programado para explorar el área, reaccionando ante los obstáculos y las señales de proximidad que detecta.

A lo largo de la simulación, el robot se enfrenta a diferentes desafíos. Las cajas de madera y los sólidos que componen las paredes le obligan a ajustar su trayectoria constantemente. Su capacidad de navegación depende de la interpretación de los datos de los sensores de proximidad, que le permiten identificar la cercanía de los objetos y ajustar la velocidad de las ruedas para moverse con precisión. Los sensores juegan un papel fundamental, ya que permiten al E-puck no solo detectar los objetos, sino también reaccionar en función de la distancia, como girar cuando está cerca de un obstáculo o detenerse si algo está demasiado cerca.

Además, en este escenario, la precisión de la simulación temporal es esencial. He establecido un paso de tiempo (TIME_STEP) lo suficientemente corto como para que el robot pueda recibir datos constantemente y hacer ajustes en tiempo real. Esto es importante en entornos dinámicos como este, donde el robot debe responder con rapidez a los objetos que encuentra a su paso.

Cajas de Madera: Las cajas de madera están dispuestas de manera estratégica en el escenario. Cada una de estas cajas representa un desafío para el robot, que debe evitarlas

mientras navega por el espacio. Al estar dispersas en diferentes ubicaciones, las cajas crean puntos de bloqueo que obligan al robot a desviarse de su curso.

Paredes: Las paredes del escenario están diseñadas para delimitar el área de trabajo del robot. Están dispuestas en los cuatro bordes del mundo, formando un recinto donde el robot opera. Estas paredes son sólidos físicos que impiden que el robot salga de los límites de la simulación.

Paredes adicionales ("solid"): Estos elementos adicionales podrían representar columnas o barreras intermedias que obligan al robot a maniobrar con mayor precisión. Estos sólidos también juegan un papel importante en cómo el robot debe ajustar su trayectoria a medida que se mueve.

En conjunto, este mundo en Webots está diseñado para simular un entorno controlado donde el robot E-puck debe navegar y tomar decisiones de manera autónoma. La disposición de los objetos y las paredes crea un espacio en el que el robot puede desarrollar habilidades de navegación, detección de obstáculos y planificación de trayectorias. Además, el patrón del tablero de ajedrez ayuda a visualizar los movimientos del robot y cómo reacciona ante las cajas de madera y otros elementos.

El E-puck, con su sistema de control avanzado, sensores de proximidad y motores, es la pieza central de esta simulación. Al enfrentarse a obstáculos y ajustarse en tiempo real, demuestra su capacidad para funcionar en entornos dinámicos, lo cual es fundamental para aplicaciones en robótica autónoma. La interacción entre el robot y el entorno muestra cómo un sistema robótico puede usar datos sensoriales para adaptarse y operar de manera eficiente en un espacio definido

Cuando el E-puck encuentra algún tipo de obstáculo en el mundo, ya sean cajas de madera o “solid” (paredes) reaccionará de manera que primero enviará un aviso al sistema físico avisando de que se encuentra frente a un objeto, pero si el sistema físico no reacciona y esta distancia se hace tan pequeña que la colisión con el objeto es inminente, este girará automáticamente a la derecha.

Este mundo ofrece un excelente punto de partida para realizar experimentos adicionales en control de robots, optimización de trayectorias o incluso la introducción de más obstáculos dinámicos para observar cómo el robot puede adaptarse a cambios en tiempo real.

5. IMPLEMENTACION

5.1. Programas del Robot Real

El código de la raspberry se conforma en dos ficheros .c y un fichero .h:

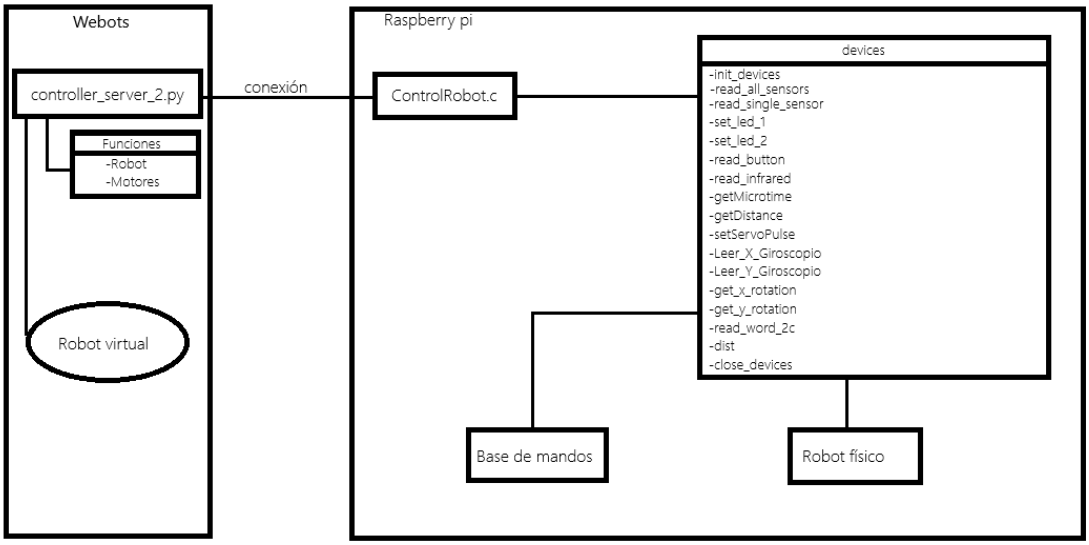


Figura 17: Programas, funciones y conexiones entre ellos

- **Devices.h**: Devices.h es un archivo de cabecera clave en este proyecto, que contiene las declaraciones de todas las funciones necesarias para interactuar con los componentes conectados a la Raspberry Pi, como sensores y actuadores. Este archivo organiza y expone las interfaces de control, permitiendo que el código principal acceda a las funcionalidades definidas en devices.c, donde se implementa la lógica específica para cada dispositivo, garantizando una estructura modular y eficiente en el manejo del hardware del robot real.

- **Devices.c**: Devices.c es el archivo de implementación donde se implementan todas las funciones necesarias para controlar los componentes físicos del robot real conectados a la Raspberry Pi. Aquí se implementa la lógica de cada dispositivo, como la lectura de sensores, el control de los servomotores y la gestión de LEDs. Este archivo ejecuta las operaciones declaradas en devices.h, permitiendo que el código principal interactúe directamente con los elementos de hardware, asegurando un control preciso y eficiente del robot en tiempo real.

- **CntrlRobot.c**: CntrlRobot.c es el archivo principal del proyecto que contiene la función main() y actúa como el punto de entrada de todo el sistema. En este archivo se maneja la lógica central, comenzando con la inicialización de los componentes físicos del robot real y estableciendo la conexión con el entorno de simulación de Webots a través de un **socket [14]**. Esta conexión permite el intercambio de datos entre el sistema físico y el robot virtual en tiempo real, facilitando una sincronización precisa de movimientos y comportamientos.

La función main() primero configura el socket, creando un canal de comunicación entre la Raspberry Pi y Webots para enviar y recibir comandos. A continuación, se realiza la inicialización de todos los dispositivos conectados a la Raspberry Pi, como sensores, servomotores y LEDs, mediante las funciones previamente definidas en devices.c.

Una vez establecidas las conexiones, el archivo entra en un bucle principal donde se monitorizan continuamente los datos provenientes de Webots y los sensores del sistema físico. Dependiendo de los datos recibidos, el robot ajusta su comportamiento, ya sea girando, cambiando de velocidad o activando dispositivos como LEDs, frenos, o servomotores. Al mismo tiempo, el archivo CntrlRobot.c envía datos a Webots para actualizar el estado del robot virtual, garantizando que ambos sistemas operen de manera coordinada.

Este archivo es crucial ya que es donde confluyen todas las conexiones y funciones, coordinando el control del robot real y la simulación en Webots de manera centralizada, gestionando tanto la lógica de control como la comunicación bidireccional entre ambos entornos.

5.2. Código Robot Real

- **Devices.h**:

  Los #include en el código incorporan diversas bibliotecas que proporcionan funcionalidades esenciales tanto del sistema como para la interacción con el hardware. Estas bibliotecas permiten la gestión de entradas y salidas (I/O), manipulación de cadenas de caracteres, manejo de errores, y control del tiempo. También incluyen librerías especializadas para la interacción con los pines GPIO de la Raspberry Pi a través de la **librería WiringPi [15]** y el control de **señales PWM [16]** para los servomotores. En conjunto, estos #include permiten la integración del sistema con el hardware, el manejo de procesos del sistema y la comunicación con dispositivos externos. Estas librerías son sacadas de las prácticas de la asignatura **Sistemas en Tiempo Real [17]**.

```
#include <stdio.h>   // Used for printf() statements
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <stdint.h>
#include <wiringPi.h> // Include WiringPi library!
#include <time.h>
#include <unistd.h>
#include <stdbool.h>
#include <sys/time.h>
#include <math.h>
```

  init_devices()

  La función init_devices() se encarga de inicializar y configurar los dispositivos conectados al sistema. Establece la comunicación con los **periféricos [18**], configura los pines GPIO para la interacción con sensores y actuadores, y ajusta los parámetros necesarios para el correcto funcionamiento del hardware.

  close_devices()

  La función close_devices() apaga y desactiva los dispositivos conectados, asegurando que los componentes como los Leds y el servomotor queden en un estado seguro antes de finalizar la ejecución del sistema.

```
// Function prototypes
int init_devices();
int close_devices();
```

  read_single_sensor(int analog_channel)

  La función read_single_sensor() obtiene el valor analógico de un sensor en un canal específico. Se asegura de que el canal sea válido y realiza la lectura a través de la **comunicación SPI [19]**, devolviendo el valor medido.

read_all_sensors(int values[])

La función read_all_sensors() realiza la lectura de todos los sensores analógicos disponibles, almacenando sus valores en un array para su posterior procesamiento.

```
// Functions to read analog channels
int read_single_sensor(int analog_channel);
int read_all_sensors(int values[]);
```

set_led_1(int Led_Value)

La función set_led_1() controla el estado del primer LED, permitiendo encenderlo o apagarlo según el valor recibido.

set_led_2(int Led_Value)

La función set_led_2() ajusta el estado del segundo LED, activándolo o desactivándolo según la señal enviada.

```
// Functions to turn on/off LEDs
int set_led_1(int Led_Value);
int set_led_2(int Led_Value);
```

read_button()

La función read_button() obtiene el estado del botón, devolviendo su valor para determinar si ha sido presionado o no.

read_infrared()

La función read_infrared() lee el estado del sensor infrarrojo, permitiendo detectar la presencia de un objeto o una señal.

```
// Functions to read digital inputs
int read_button();
int read_infrared();
```

set_trigger(int signal_value)

La función set_trigger() activa o desactiva la señal de disparo del sensor de ultrasonidos, permitiendo iniciar una medición de distancia.

read_echo()

La función read_echo() lee la señal de respuesta del sensor de ultrasonidos, indicando si se ha recibido el eco de la señal emitida.

```
// Functions to read ultrasound
int set_trigger(int signal_value);
int read_echo();
```

setServoPulse(int pulseWidth)

La función setServoPulse() envía un pulso al servomotor para ajustar su posición, variando la duración del pulso según la configuración del reloj y el rango del PWM.

```
// Function to move servomotor
void setServoPulse(int pulseWidth);
```

Inicializar_dispositivos()
La función Inicializar_dispositivos() inicializa todos los dispositivos y configura la **comunicación I2C [20]** con los dispositivos conectados, desactivando el modo de suspensión y estableciendo los parámetros iniciales.

dist(double a, double b)
La función dist() calcula la distancia entre dos puntos en un plano utilizando el teorema de Pitágoras.

read_word_2c(int addr)
La función read_word_2c() lee un valor de 16 bits desde un registro I2C, procesando los datos para interpretar valores negativos correctamente.

get_y_rotation(double x, double y, double z)
La función get_y_rotation() calcula el ángulo de inclinación en el eje Y a partir de valores de aceleración.

get_x_rotation(double x, double y, double z)
La función get_x_rotation() calcula el ángulo de inclinación en el eje X a partir de valores de aceleración.

Leer_X_Giroscopo()
La función Leer_X_Giroscopo() obtiene los valores del acelerómetro y calcula la inclinación en el eje X.

Leer_Y_Giroscopo()
La función Leer_Y_Giroscopo() obtiene los valores del acelerómetro y calcula la inclinación en el eje Y.

```
// Functions for accelerometer and gyroscope
void Inicializar_dispositivos();
int Leer_X_Giroscopo();
int Leer_Y_Giroscopo();
double get_x_rotation(double x, double y, double z);
double get_y_rotation(double x, double y, double z);
double dist(double a, double b);
int read_word_2c(int addr);
```

getMicrotime()

La función getMicrotime() obtiene el tiempo actual en microsegundos, permitiendo medir intervalos de tiempo con alta precisión.

getDistance()

La función getDistance() calcula la distancia a un objeto utilizando un sensor ultrasónico, midiendo el tiempo que tarda la señal en regresar tras rebotar en el objeto.

```
// Function to calculate distance
long getMicrotime();
float getDistance();
```

- **CntrlRobot.c**:

  Este código tiene como objetivo establecer una interacción en tiempo real entre un sistema físico y un servidor remoto (como Webots), a través de la lectura y envío de datos de sensores, el control de actuadores y el manejo de señales. La estructura principal está dividida en varias fases: primero, se establecen las conexiones mediante un socket **TCP/IP [21]** que permite la comunicación con el servidor. Luego, los dispositivos del robot se inicializan, incluyendo sensores analógicos, botones, servomotores y LEDs, para asegurar que todas las interfaces de hardware estén preparadas para la ejecución.

  El código también cuenta con varios mecanismos de control. Utiliza sensores como el ultrasonido y el infrarrojo para medir la distancia y detectar obstáculos en el entorno, mientras que el botón funciona como una especie de interruptor de seguridad o freno. Los datos de los sensores son continuamente leídos y procesados, y a partir de estos valores se determinan las acciones a seguir, como ajustar la dirección del servomotor o encender los Leds dependiendo de la proximidad de los objetos.

  Uno de los aspectos centrales es la comunicación en tiempo real con el servidor, donde el robot envía información relevante (como la posición del volante, velocidad y valores de los joysticks) y recibe instrucciones para ajustar su comportamiento. Este ciclo continuo de envío y recepción de datos asegura una constante retroalimentación y sincronización entre el sistema físico y su representación virtual, permitiendo que ambos respondan a cambios en el entorno o comandos recibidos.

  En general, el código implementa un flujo robusto de monitoreo y control, donde se equilibran las lecturas de los sensores con las respuestas físicas del robot. Los ajustes dinámicos del servomotor, el manejo de Leds como indicadores visuales, y la detección de obstáculos permiten una interacción fluida y adaptativa en un entorno compartido entre el mundo real y el simulado.

  Bibliotecas introducidas de la misma manera que en el devices.h, con la diferencia de que aquí se ha añadido el mismo devices.h. fichero donde vienen todas las cabeceras de las funciones del devices.c.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <stdint.h>
#include <softPwm.h>
#include <wiringPi.h>
#include <math.h>
#include <time.h>
#include <arpa/inet.h>
#include "devices.h"
```

Comenzamos el int main declarando todas las variables.

```
int main(void) {
//declaramos todas las variables
```

sock: Socket utilizado para la comunicación con el servidor Webots.

valread: Almacena el valor de bytes leídos tras recibir datos del socket.

valor_leds: Variable que controla el estado de los Leds (encendidos/apagados).

i: Contador utilizado para bucles, como el de espera del botón.

volante: Valor leído de los sensores analógicos que determina la dirección de giro del robot.

velocidad: Valor de los sensores que representa la velocidad del robot.

joystickx y joysticky: Valores de los sensores que capturan la posición de los ejes del joystick.

boton: Variable que detecta el estado del botón (presionado o no).

devices: Almacena el estado de inicialización de los dispositivos conectados (si han sido configurados correctamente).

infrarrojos: Variable que almacena el estado del sensor de infrarrojos.

analog_sensors: Array de valores de los sensores analógicos conectados al robot.

serv_addr: Estructura que guarda la dirección IP y el puerto para la conexión de socket.

dist: Almacena la distancia medida por el sensor de ultrasonidos.

dist_v y **dist_v_ant**: Almacenan la información recibida sobre la proximidad de objetos desde Webots.

buffer: Array que recibe los datos del servidor Webots.

combined_data: Array que almacena los datos formateados para enviar a Webots.

temp_buffer: Array temporal para construir los mensajes enviados por socket.

```
int sock = 0, valread, valor_leds, i, volante,
velocidad,   joystickx,      joysticky, boton, devices,
infrarrojos;
int analog_sensors[8] = {0, 0, 0, 0, 0, 0, 0, 0};
struct sockaddr_in serv_addr;
float dist;
char dist_v = '0', dist_v_ant = '0';
```

```
char buffer[1024] = {0};
char combined_data[256] = {0};
char temp_buffer[32];
```

Función específica para crear el socket.

```
// Crear socket
if ((sock = socket (AF_INET, SOCK_STREAM, 0)) < 0) {
printf("\n Error al crear el socket \n");
return -1;
}
```

Estas funciones piden la dirección ip del servidor con el cual se va a conectar por socket y hace una prueba para comprobar que esa ip es correcta y la conexión funciona bien.

```
//conectar con el socket de webots, importante poner la
dirección ip del ordenador destino
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(5000);
if (inet_pton(AF_INET, "192.168.1.132",
&serv_addr.sin_addr) <= 0) { //192.168.3.12
printf("\n Dirección no válida \n");
return -1;
}

// Conectar al servidor
if (connect(sock, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0) {
printf("\n Falló la conexión \n");
return -1;
}
```

Llamamos a la función init_devices definida en el devices.c

```
// Iniciamos los devices con todos los pines
printf("Testing devices ... \n");

devices = init_devices();
printf("Devices initialized: %d \n", devices);
```

Esta parte está desarrollada para poder alinear la posición del servomotor del sistema físico con la posición del robot virtual. Primero comienza a girar el servomotor a una velocidad relativamente lenta, para poder observar bien su movimiento y poder calcular bien el momento de pararlo. En el momento en el que el servomotor quede alineado con el robot de webots hay que pulsar el botón y el servomotor dejará de mover, pero habrá que mantener el botón pulsado durante 3 segundos seguidos para confirmar que esa es la posición correcta, ya que el usuario puede cometer un error y no tener la posición correcta a la primera, para ello solo hay que soltar el botón y este volverá a girar y a repetir el proceso. Una vez pasados esos 3 segundos ya se podrá soltar el botón sin peligro de que el servomotor vuelva a girar.

```
printf("\nColoque el servomotor de forma recta alineado
con el robot\n\n");
delay(2000);
i=0;
while (i < 3) {
// Le asignamos a la variable botón el valor 1 si se
está pulsando el botón y el 0 sino
boton = read_button();
if(boton){
// En esta parte el servo comenzará a girar y no parará
hasta que se pulse el botón
i=0;
setServoPulse(140);
delay(100);
}else{
// Cuando se pulsa el botón este else se activará y el
servo parará
```

```
i++;
setServoPulse(0);
delay(1000);
}
}
// Saldrá del bucle cuando se haya estado 3 segundos seguidos con el botón pulsado
```

Comenzamos calculando la distancia con el ultrasonido y escribiéndola por pantalla.

```
while (1) {//bucle principal
// Función para calcular la distancia del ultrasonido y asignársela a la variable dist
dist = getDistance();
printf("\nDistance: %.2f cm\n", dist);
```

A continuación, leemos los valores del botón y los infrarrojos y se los asignamos a sus respectivas variables.

```
// Función para utilizar el sensor de infrarrojos
infrarrojos = read_infrared();
// Volvemos a leer el valor del botón pero esta vez para utilizarlo como un freno de mano
boton = read_button();
```

Leemos también los valores de los 8 canales analógicos y se los asignamos al array de analog_sensors.

```
// Leemos los valores analógicos
read_all_sensors(analog_sensors);
```

Asignamos los valores analógicos todos a sus respectivas variables.

```
// Asignamos los valores analógicos a las variables volante, velocidad, joystickx y joysticky
// Ponemos el 10 – para darle la vuelta al valor, ya que sin esto el robot girará de forma contraria al volante, es decir, pasamos los 10 a 0, los 9 a 1, sucesivamente
```

```
// Además, tenemos que dividir entre 100 para poder mandar los números por socket y que no haya problemas de entendimiento al poner números de dos o tres cifras
volante = 10 - analog_sensors[2]/100;
velocidad = analog_sensors[3]/100;
// Este caso es el mismo que el de volante y el de velocidad, pero dividendo entre 540 ya que saca valores entre 0 y 540 en vez de entre 0 y 1000
joystickx = (analog_sensors[1] - 240)/54;
joysticky = (analog_sensors[4] - 240)/54;
```

Convertimos los 10 en 9 para que no haya problemas de lectura después al recibir los datos por el webots, ya que con un 10 mandas dos valores en vez de uno.

Además, estos valores se entenderán de la siguiente manera:

- Si recibe un 0 se entenderá como velocidad negativa máxima, equivalente a 9 pero marcha atrás.
- Si recibe un 1 se entenderá como velocidad negativa relativamente alta, equivalente a un 6 marcha atrás.
- Si recibe un 2 se entenderá como velocidad negativa relativamente baja, equivalente a un 3 marcha atrás.
- Si recibe un 3 se entenderá como velocidad nula, y el vehículo queda quieto.
- Cualquier otra velocidad superior a 3 se entenderá como velocidad positiva.

```
printf("\nVelocity: %d km/h\n", analog_sensors[3]);
// Para que no haya problemas para pasar el valor por socket, ya que 10 es un valor de 2 cifras, convertimos el 10 en 9
if(volante==10)
volante--;
if(velocidad==10)
velocidad--;
```

Comprobamos los infrarrojos y el botón y paramos el robot en caso de que detecten algo.

```
// Si recibe 1 de la variable botón o la variable infrarrojos asigna el valor 3 a la variable velocidad

// En el código del webot se sobreentiende el 0, el 1 y el 2 como valores de velocidad negativa y el robot irá hacia atrás

// El 3 en el webots se sobreentenderá como velocidad nula, y el robot se quedará quieto

if(!boton || !infrarrojos)

velocidad = 3;
```

Este bloque de código controla el giro del servomotor basado en la velocidad y dirección del robot. Si la velocidad es mayor que 3, ajusta el ángulo del servomotor en función de la dirección del volante, con pequeños retrasos para sincronizar el movimiento entre el sistema físico y el virtual. Dependiendo de la posición del volante (menor de 4 o mayor de 6), el servomotor gira hacia un ángulo específico. Si la velocidad es menor o igual a 3, el mismo proceso de ajuste ocurre, pero se enfoca en mantener el robot recto.

```
// En este if y su correspondiente else se girará el servomotor teniendo en cuenta la velocidad del robot virtual

if(velocidad>3){

if(volante<4){

setServoPulse(0);

// Los delays se utilizan para poder ajustar bien los giros

// Ya que es imposible girar el servo a la misma velocidad que el robot virtual

delay(1500);
setServoPulse(148);
}else if(volante>6){
setServoPulse(0);
delay(1500);
setServoPulse(153);
}else
setServoPulse(0);
}else{
```

```
if(volante<4){
setServoPulse(0);
delay(1500);
setServoPulse(153);
}else if(volante>6){
setServoPulse(0);
delay(1500);
setServoPulse(148);
}else
setServoPulse(0);
}
```

Este fragmento de código toma los valores de varias variables (volante, velocidad, joystickx, y joysticky) y los convierte en una cadena de caracteres mediante el uso de un array temporal llamado temp_buffer. Cada valor se añade secuencialmente al array combined_data utilizando la función snprintf para formatear los datos y strcat para concatenarlos. Al final, todos los valores se combinan en una sola cadena que puede enviarse de forma compacta a través de la comunicación por sockets.

```
// Pasamos todos los valores en un mismo array de
caracteres llamado combined_data
// Asignándolos temporalmente a otro array llamado
temp_buffer
snprintf(temp_buffer, sizeof(temp_buffer), "%d ",
volante);
strcat(combined_data, temp_buffer);
snprintf(temp_buffer, sizeof(temp_buffer),
"%d ", velocidad);
strcat(combined_data, temp_buffer);
snprintf(temp_buffer, sizeof(temp_buffer),
"%d ", joystickx);
strcat(combined_data, temp_buffer);
snprintf(temp_buffer, sizeof(temp_buffer), "%d",
joysticky);
strcat(combined_data, temp_buffer);
```

Este código analiza el primer valor recibido en el **buffer [22]** de datos, asignándolo a dist_v, que indica la proximidad de un objeto (0 = sin objetos, 1 = objeto cercano, 2 = objeto muy próximo). Si dist_v es '2', el robot ajusta el servomotor para esquivar el obstáculo. Si previamente estaba en modo de evasión (cuando dist_v_ant era '2'), el servo se detiene cuando ya no se detectan objetos cercanos. Además, dependiendo de la distancia detectada por el sensor de ultrasonidos o por dist_v, enciende luces rojas o

```
// Mandamos el array por socket
send(sock, combined_data, strlen(combined_data),0);
// Leemos la respuesta de socket
valread = read(sock, buffer, 1024);
printf("%s\n\n", buffer);
// Limpiamos el array combined_data para el próximo envío
combined_data[0] = '\0';
// Asignamos a la variable dist_v el valor de la primera posición de los datos recibidos
// Realizado porque en la primera posición siempre recibirá un 0, un 1 o un 2
// Siendo el 0 un sin objetos, el 1 un objeto próximo y el 2 y objeto muy próximo
dist_v = buffer[0];
// Si recibe un 2, significará que el robot ha encontrado un objeto y lo tiene que esquivar
// Por ello girará igual que el robot virtual
if (dist_v == '2'){
setServoPulse(0);
delay(500);
setServoPulse(148);
dist_v_ant = '2';
}else if (dist_v_ant == '2'){
// En el momento en el que deje de encontrar objetos cercanos entrará en este if, ya que dist_v dejará de ser 2
// La variable dist_ant_v ha sido creada para que solo entre en este bucle después de dejar de encontrar objetos cercanos
```

amarillas para indicar objetos cercanos o muy próximos. Si no hay obstáculos, ambas luces se apagan.

```
dist_v_ant = '0';
setServoPulse(0);
}
// Si el robot virtual encuentra un objeto muy cercano o del mismo modo lo hace el ultrasonido, se encenderá la luz roja
if(dist_v == '2' || dist < 8.00){
valor_leds = set_led_1 (1);
valor_leds = set_led_2 (0);
}else if(dist_v == '1' || dist < 15.00){
// Si el robot virtual encuentra un objeto cercano o del mismo modo lo hace el ultrasonido, se encenderá la luz amarilla
valor_leds = set_led_1 (0);
valor_leds = set_led_2 (1);
}else{
// Si no encuentra ningún objeto apagará ambas luces
valor_leds = set_led_1 (0);
valor_leds = set_led_2 (0);
}
```

Paramos el robot durante 5 segundos en caso de que se haya pulsado el robot o se hayan activado los infrarrojos para dar tiempo al usuario para reaccionar y cambiar la dirección del mismo.

```
// Si detecta que el botón ha sido pulsado o el infrarrojos activado hará un parón de 5 segundos
     if(!boton || !infrarrojos)
     delay(5000);
```

Limpiamos el buffer.

```
delay(50);
// Limpiamos el buffer para el siguiente recibimiento de datos
for(i=0;i<1024;i++)
buffer[i] = 0;
}
```

Y con esto el código de ControlRobot.c acaba.

```
return 0;
}
```

5.3. Código Robot Virtual

Este código tiene como propósito controlar un robot mediante la recepción de comandos externos, específicamente desde una Raspberry Pi, mientras se utiliza el simulador Webots como servidor. La arquitectura del sistema se basa en la creación de una conexión de red TCP/IP entre el robot virtual en Webots y el cliente (Raspberry Pi), lo que permite que ambos dispositivos intercambien datos en tiempo real. El robot virtual está equipado con varios sensores de proximidad que son usados para detectar obstáculos, y con motores para mover las ruedas de manera individual, permitiendo movimientos de avance, retroceso y giros.

La implementación comienza con la configuración básica del robot, donde se inicializan sus sensores y motores. Los sensores de proximidad (8 en total) monitorean el entorno cercano del robot para detectar objetos, mientras que los motores de las ruedas izquierda y derecha se configuran para operar en modo de velocidad controlada.

El servidor Webots actúa como el punto de conexión, esperando a que la Raspberry Pi establezca una conexión para iniciar el intercambio de información. Una vez conectados, los datos enviados desde la Raspberry Pi, principalmente valores de los canales de control (por ejemplo, channel_2 y channel_3), determinan el comportamiento del robot. El código incluye una lógica para convertir estos valores de entrada en velocidades que se aplican a los motores del robot. Además, se emplea una función de escalado (scale_value) para adaptar los valores de entrada a rangos de velocidad adecuados para las ruedas del robot.

Una parte importante del código es la gestión de la detección de obstáculos. Utilizando los valores de los sensores de proximidad, el robot evalúa si hay objetos cercanos o muy cercanos, ajustando su velocidad y dirección en consecuencia. Si se detecta un objeto muy próximo, el robot reacciona girando para evitarlo, mientras que, si no hay obstáculos, sigue las instrucciones de movimiento recibidas desde la Raspberry Pi. Esta capacidad de reaccionar dinámicamente al entorno, junto con las órdenes externas, hace que el robot sea adaptable a diferentes situaciones.

En resumen, este código crea una plataforma robusta para controlar un robot virtual de manera remota, permitiendo un control basado en comandos enviados desde un dispositivo externo. La combinación de control manual y automático basado en la detección de obstáculos hace que el sistema sea flexible y eficiente en entornos dinámicos.

Para poder introducir este código dentro de un robot E-puck debemos realizar los siguientes pasos:

1- Descargar los archivos collision_avoidance.wbt y con controller_server_2.py y la aplicación webots en el caso de no tenerla instalada.

2- Abrir el archivo collision_avoidance.wbt en webots.

3- Pulsar File/New/NewRobotController.

4- Darle a Next y ajustar Python como lenguaje.

5- Poner de nombre controller_server_2.

6- Pulsar Finish teniendo marcada la opción de open in the text editor.

7- Cuando ya se haya abierto el código a la derecha, pulsar en la carpeta encima del código y abrir el controller_server_2.py previamente instalado.

8- Copia todo el código del fichero relleno en el fichero vacío.

9- Pulsa en guardar arriba y ya empezará a correr con normalidad al darle al botón de play.

Importamos las librerías necesarias para ejecutar este código.

```
from controller import Robot, Motor
import socket
import select
import time
```

Creamos las variables globales de tiempo para ejecutar el bucle creado más adelante y para utilizar los sensores de distancia del robot, y velocidad para poder asignarle una velocidad máxima y una velocidad de giro.

```
# tiempo en [ms] de un paso de simulación
TIME_STEP = 64
MAX_SPEED = 4.00
```

Asignamos a una variable llamada robot una función creada para inicializar el robot y sus motores.

```
# Crea la instancia del robot
robot = Robot()
```

Función para activar los sensores de proximidad del robot.

```
# Inicializar sensores de proximidad
ps = []
psNames = ['ps0', 'ps1', 'ps2', 'ps3', 'ps4', 'ps5',
'ps6', 'ps7']
for i in range(8):
ps.append(robot.getDevice(psNames[i]))
ps[i].enable(TIME_STEP)
```

Iniciamos los motores y les asignamos una velocidad inicial de 0 para que el robot no se mueva hasta que reciba la primera orden.

```
# Inicializar los motores
left_motor = robot.getDevice("left wheel motor")
right_motor = robot.getDevice("right wheel motor")
left_motor.setPosition(float('inf'))
right_motor.setPosition(float('inf'))
left_motor.setVelocity(0.0)
right_motor.setVelocity(0.0)
```

Configuramos toda la conexión de socket como en el servidor, pero esta vez como receptor, de ahí el '0.0.0.0', está configurado para recibir la respuesta de cualquier envío.

```
# Configuración del servidor (Webots actúa como
servidor)
server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
server_socket.setblocking(False)  # Sockets no
bloqueantes
server_address = ('0.0.0.0', 5000)
server_socket.bind(server_address)
server_socket.listen(1)
print("Esperando conexión de la Raspberry Pi...")
```

Estas líneas son necesarias para que el robot se mueva de forma fluida, ya que creando esas variables y entrando en los condicionales que veremos mas adelante el robot no dependerá de la conexión en ningún momento para poder moverse, sino que se ejecutará de manera independiente y se irá actualizando con los datos que reciba.

```
# Esperar la conexión sin bloquear el ciclo de simulación
client_socket = None
connected = False
```

Comenzamos con estas velocidades hasta que no reciba una actualización.

```
# Variables para controlar el movimiento y los datos
left_speed = 0.0
right_speed = 0.0
```

Esta función es necesaria y se utilizará mas adelante para poder ejecutar la velocidad en unos rangos en los que el robot pueda ejecutar bien.

```
def scale_value(value, src_min, src_max, dst_min, dst_max):
"""Función para escalar un valor de un rango a otro."""
return dst_min + (float(value - src_min) / float(src_max - src_min) * (dst_max - dst_min))
```

Si no hay ningún usuario conectado comprobamos si hay alguna conexión pendiente, en caso afirmativo comprobamos si podemos conectarnos a ella, si es posible nos conectamos y nos ponemos en estado ‘conectado’, si no es posible la rechazamos y esperamos otra conexión.

```
# Loop principal
while robot.step(TIME_STEP) != -1:
# Revisar si ya hay una conexión o si estamos esperando a aceptar una
if not connected:
# Ver si hay una nueva conexión pendiente
readable, _, _ = select.select([server_socket], [], [], 0)
if readable:
```

```
client_socket, client_address = server_socket.accept()
client_socket.setblocking(False)  # Socket no bloqueante para el cliente
connected = True
print(f"Conexión establecida con {client_address}")
```

En caso de que ya estemos en estado conectado comprobamos los datos que estamos recibiendo y si los podemos leer correctamente, y en caso afirmativo se los asignamos a la variable data.

```
# Si ya hay un cliente conectado, leer datos sin bloquear
if connected:
readable, _, _ = select.select([client_socket], [], [], 0)
if readable:
# Procesar los datos recibidos
data = client_socket.recv(1024).decode('utf-8')
```

Si data son los datos que estamos esperando ya empezamos a tratar con ellos.

```
if data:
print(f"Datos recibidos: {data}")
```

Asignamos los datos a las variables channel_2 y channel_3, esta parte puede variar, ya que en caso de que queramos recibir los valores del potenciómetro rotatorio y el potenciómetro deslizable hay que dejarlos como están, con el data[0] y data[2], pero si queremos recibir los valores del joystick hay que cambiar esos valores a data[4] y data[6].

```
# Asignar datos a channels 2 y 3
channel_2 = int(data[0])
channel_3 = int(data[2])
```

En esta función comprobamos constantemente los objetos próximos que tenga el robot, importante comprobar los sensores 0, 1, 6 y 7 ya que estos son los sensores frontales del robot y el 2, 3, 4 y 5 son los traseros. En caso de que tenga un objeto cercano se activará

la variable obstacle, y en caso de que tenga un objeto muy cercano se activará la variable close_obstacle.

```
#comprobar proximidad de objetos
psValues = []
for i in range(8):
psValues.append(ps[i].getValue())
# Asignamos a los valores el peligro de proximidad
obstacle = psValues[0] > 72.0 or psValues[1] > 72.0 or
psValues[6] > 72.0 or psValues[7] > 72.0
close_obstacle = psValues[0] > 80.0 or psValues[1] >
80.0 or psValues[6] > 80.0 or psValues[7] > 80.0
```

En este apartado configuraremos la velocidad del robot, que como ya expliqué anteriormente, el 0, 1 y 2 serán velocidades negativas, el 3 nula y del 4 al 9 positivas.

```
# Ajustamos la velocidad basada en el channel_3
vel = channel_3
if vel == 0:
vel = 9
elif vel == 1:
vel = 6
elif vel == 2:
vel = 3
elif vel == 3:
vel = 0
```

Con estas funciones configuraremos dos tipos de velocidades, una en caso de que el robot vaya recto (speed) y otra en caso de que el robot este girando (speed_g), significativamente menor.

```
speed = scale_value(vel, 0, 9, 0, MAX_SPEED)
# Actualizar la velocidad del robot basado en los datos
speed_g = scale_value(vel, 0, 9, 0, 0.8)
```

En este punto comprobaremos los sensores de proximidad, y en caso de que uno de ellos detecte un objeto muy cercano haremos que el robot gire para así poder evitarlo y enviaremos una respuesta al robot real.

```
# Enviamos la informacion de la proximidad
if close_obstacle:
response = "2: Objeto MUY próximo"
left_speed = speed_g
right_speed = -speed_g
```

En estos puntos enviaremos el resto de respuestas al robot real, 1 en caso de objeto cercano y 0 en caso de que no haya objetos.

```
else:
if obstacle:
response = "1: Objeto próximo"
else:
response = "0: Sin Objetos"
# Giramos en robot en funcion de lo recibido en el
channel_2
```

En caso de recibir un valor menor que 4 en el channel_2 significara que estamos moviendo el potenciómetro rotatorio hacia la derecha o el joystick, por lo que haremos que el robot gire a la derecha.

```
if channel_2 < 4:
left_speed = speed_g
right_speed = -speed_g
```

En caso de recibir un valor mayor que 6 en el channel_2 significara que estamos moviendo el potenciómetro rotatorio hacia la izquierda o el joystick, por lo que haremos que el robot gire a la izquierda.

```
elif channel_2 > 6:
left_speed = -speed_g
right_speed = speed_g
```

En caso de que no tengamos el potenciómetro o los joysticks con intención de girar comprobaremos si es velocidad negativa o positiva.

```
else:
# Ajustamos velocidad negativa
```

Ajustamos la velocidad del robot con las velocidades actualizadas.

```
if channel_3 < 3:
left_speed = -speed
right_speed = -speed
else:
left_speed = speed
right_speed = speed
left_motor.setVelocity(left_speed)
right_motor.setVelocity(right_speed)
```

Enviamos una respuesta final vacía necesaria para que no de error el código del webots.

```
# Responder a la Raspberry Pi
client_socket.sendall(response.encode('utf-8'))
response = " ";
```

Enviamos otras velocidades finales que se ejecutaran siempre, este o no este activa la conexión.

```
# Enviar velocidades al motor (esto se ejecuta siempre)
left_motor.setVelocity(left_speed)
right_motor.setVelocity(right_speed)
```

6. PRUEBAS

6.1.Pruebas de unidad

Descripción	Pasos de Ejecución	Resultado Esperado	Resultado Obtenido
Se evalúa si el **sensor de ultrasonidos** puede medir correctamente la distancia de un objeto situado dentro de su rango operativo.	Coloca un objeto fijo a 20 cm del sensor de ultrasonidos. Inicia la lectura de datos desde el sensor mediante el software en la Raspberry Pi. Observa y registra el valor de distancia que devuelve el sensor.	El sensor devuelve un valor de 20 cm, con una tolerancia de ±1 cm (es decir, entre 19 y 21 cm).	El sensor devolvió 20 cm de distancia, cumpliendo con el resultado esperado.
Se valida si el **botón** responde correctamente a la interacción física, enviando señales para los estados pulsado y no pulsado.	Conecta el botón a la Raspberry Pi. Pulsa el botón y observa el cambio en el estado registrado por el software (de no pulsado a pulsado). Suelta el botón y verifica que el estado vuelve a no pulsado.	El sistema registra los estados de pulsado y no pulsado correctamente.	El sistema registró correctamente los estados pulsado y no pulsado en todos los intentos.
Se prueba si el **potenciómetro rotatorio** detecta correctamente las posiciones angulares en todo su rango operativo.	Conecta el potenciómetro rotatorio a la Raspberry Pi. Gira el potenciómetro a tres posiciones: mínimo (0°), medio (50% de su rango) y máximo (100%). Observa y registra los valores obtenidos en el software.	El potenciómetro devuelve valores proporcionales a las posiciones: 0%, 50% y 100%.	El potenciómetro detectó correctamente las posiciones 0%, 50% y 100%.

Descripción	Pasos de Ejecución	Resultado Esperado	Resultado Obtenido
Se valida que el **potenciómetro deslizable** registra correctamente las posiciones de desplazamiento en todo su recorrido.	Conecta el potenciómetro deslizable a la Raspberry Pi. Desliza el potenciómetro a tres posiciones: mínimo, medio y máximo. Observa y registra los valores obtenidos.	El potenciómetro devuelve valores proporcionales a las posiciones: 0%, 50% y 100%.	El potenciómetro deslizable registró correctamente los valores 0%, 50% y 100%.
Se verifica si el **sensor infrarrojo** detecta de forma precisa la presencia de un objeto dentro de su rango de operación.	Coloca un objeto dentro del rango del sensor infrarrojo. Observa si el sensor registra la detección del objeto. Retira el objeto y verifica que el sensor ya no detecta nada.	El sensor detecta correctamente la presencia o ausencia de un objeto.	El sensor infrarrojo detectó con precisión la presencia y ausencia de objetos.
Se valida si el **joystick** registra correctamente los movimientos en las cuatro direcciones principales (arriba, abajo, izquierda, derecha).	Conecta el joystick a la Raspberry Pi. Mueve el joystick hacia arriba, abajo, izquierda y derecha. Observa y registra los valores para cada dirección.	El sistema registra los movimientos en las cuatro direcciones correctamente.	El joystick detectó correctamente los movimientos en todas las direcciones probadas.
Se comprueba que los **Leds** responden correctamente a las señales enviadas desde el software.	Envía una señal desde el software para encender el LED amarillo. Observa si el LED amarillo se enciende. Envía una señal para apagarlo y luego enciende el LED rojo. Observa si el LED rojo se enciende.	Los Leds amarillo y rojo se encienden y apagan según las señales enviadas.	Los Leds amarillo y rojo se encendieron y apagaron correctamente en respuesta a las señales.

Descripción	Pasos de Ejecución	Resultado Esperado	Resultado Obtenido
Se evalúa si el **servomotor** responde de forma precisa a los comandos de posición angular enviados desde el software.	Envía comandos para posicionar el servomotor en 0°, 90° y 180°. Observa y registra si el servomotor se posiciona correctamente en cada ángulo.	El servomotor se posiciona en los ángulos 0°, 90° y 180° según los comandos enviados.	El servomotor alcanzó correctamente los ángulos 0°, 90° y 180°.
Se prueba si el **robot virtual** puede realizar giros precisos en el entorno simulado de Webots.	Envía un comando para que el robot gire 90° a la izquierda. Observa si el robot realiza el giro en el entorno virtual. Repite el procedimiento para un giro de 90° a la derecha.	El robot gira 90° en ambas direcciones correctamente.	El robot realizó giros de 90° a izquierda y derecha de manera precisa.
Se evalúa si el **robot virtual** responde a comandos de movimiento en línea recta en el entorno simulado.	Envía un comando para que el robot avance 1 metro hacia adelante. Observa si el robot realiza el movimiento. Envía un comando para que retroceda 1 metro.	El robot se mueve hacia adelante y atrás correctamente en Webots.	El robot avanzó y retrocedió 1 metro en el entorno virtual sin errores.
Se valida si el **robot virtual** detecta objetos en el entorno simulado y comunica correctamente esta información al sistema físico.	Coloca un objeto en el entorno virtual frente al robot. Observa si el robot virtual detecta el objeto y envía la señal adecuada al sistema físico. Verifica si los Leds amarillo o rojo del sistema físico se encienden según la distancia simulada.	El sistema físico responde correctamente a las detecciones del entorno virtual.	El sistema físico encendió los Leds según las detecciones realizadas en el entorno virtual.

6.2.Pruebas de Integración

Descripción	Pasos de Ejecución	Resultado Esperado	Resultado Obtenido
Asegurarse de que el **servomotor** responde correctamente a las lecturas del sensor de ultrasonido.	Coloca un objeto a 15 cm del sensor de ultrasonido. Configura el sistema para que el servomotor gire a 90° cuando el sensor detecte una distancia menor a 20 cm. Observa el comportamiento del servomotor tras la detección.	El servomotor se mueve a 90° cuando el sensor detecta el objeto.	El servomotor se mueve correctamente a 90°.
Verificar que la **Raspberry Pi** envía los datos correctos al robot virtual para simular sus movimientos.	Conecta la Raspberry Pi al software del robot virtual. Inicia una sesión de prueba en la que la Raspberry envía datos de movimiento (p. ej., avance y giro). Observa el comportamiento del robot virtual en pantalla.	El robot virtual se mueve y gira según los comandos enviados por la Raspberry Pi.	El robot virtual responde correctamente a los comandos.
Se prueba si el sistema enciende las luces **LED** según las detecciones realizadas por el sensor infrarrojo.	Coloca un objeto frente al sensor infrarrojo dentro de su rango operativo. Configura el sistema para que el LED amarillo se encienda al detectar un objeto cercano. Retira el objeto y verifica que el LED se apague.	El LED amarillo se enciende cuando se detecta un objeto y se apaga al retirarlo.	El LED amarillo respondió correctamente a las señales del sensor infrarrojo.

Descripción	Pasos de Ejecución	Resultado Esperado	Resultado Obtenido
Se evalúa si los datos del **sensor infrarrojo** son enviados correctamente a la Raspberry Pi y si se procesan adecuadamente.	Conecta el sensor infrarrojo a la Raspberry Pi. Coloca un objeto delante del sensor infrarrojo dentro de su rango operativo. Observa en el software de la Raspberry Pi si se registra la detección del objeto. Retira el objeto y verifica si el software indica la ausencia del mismo.	La Raspberry Pi detecta correctamente la presencia y ausencia del objeto según las lecturas del sensor infrarrojo.	La Raspberry Pi procesó correctamente las lecturas del sensor infrarrojo, detectando la presencia y ausencia del objeto de manera precisa.

6.3.Pruebas de Flujo Completo

Descripción	Pasos de Ejecución	Resultado Esperado	Resultado Obtenido
Se comprueba si el sistema arranca correctamente desde la **interfaz de usuario** y todos los componentes inicializan adecuadamente.	Enciende la Raspberry Pi y accede a la interfaz de usuario. Verifica si todos los componentes (sensores, actuadores y comunicaciones) son reconocidos. Confirma que no se reportan errores durante el arranque.	El sistema arranca correctamente y todos los componentes se inicializan sin errores.	El sistema arrancó correctamente y todos los componentes se inicializaron sin errores.
Se asegura de que la **Raspberry Pi** se conecta correctamente al entorno simulado mediante la dirección IP configurada.	Introduce la dirección IP del robot virtual en la configuración del sistema. Envía un paquete de prueba desde la Raspberry Pi al robot virtual. Observa si el robot virtual confirma la recepción del paquete.	La Raspberry Pi establece correctamente la conexión con el robot virtual y este confirma la recepción del paquete.	La conexión entre la Raspberry Pi y el robot virtual se estableció correctamente, y el paquete de prueba fue recibido.
Se verifica que los datos recogidos por los sensores conectados a la **Raspberry Pi** son enviados y reflejados correctamente en el entorno del robot virtual.	Activa los sensores del sistema (ultrasonido, infrarrojo, potenciómetros, etc.). Introduce un objeto o manipula los sensores para generar datos. Observa en la interfaz del robot virtual si los datos enviados se reflejan correctamente.	Los datos recogidos por los sensores se envían correctamente y se reflejan en tiempo real en el robot virtual.	Los datos de los sensores fueron enviados correctamente y reflejados en tiempo real en el robot virtual.

Descripción	Pasos de Ejecución	Resultado Esperado	Resultado Obtenido
Se valida que el **robot virtual** ejecuta correctamente las acciones derivadas de los comandos enviados desde el sistema físico.	Configura comandos básicos (p. ej., movimientos o cambios en el estado de los Leds del entorno virtual). Observa si el robot virtual ejecuta los comandos de manera precisa. Registra cualquier discrepancia entre el comando enviado y la acción realizada.	El robot virtual ejecuta correctamente todos los comandos enviados sin discrepancias.	El robot virtual respondió correctamente a todos los comandos enviados desde el sistema físico.
Se comprueba si el **sistema** permite finalizar la ejecución de manera controlada mediante un bloqueo manual.	Activa el sistema y ejecuta un flujo de prueba en el robot virtual. Desde la interfaz de usuario, selecciona la opción para bloquear el sistema. Observa si todos los procesos del sistema se detienen de forma segura.	El sistema se bloquea correctamente al ejecutar la opción de bloqueo, y todos los procesos finalizan de manera segura.	El sistema se bloqueó correctamente y todos los procesos finalizaron sin errores.

6.4.Pruebas de Interfaz de Usuario

Descripción	Pasos de Ejecución	Resultado Esperado	Resultado Obtenido
Comprobar que la **interfaz** permite introducir la IP del robot virtual y la valida correctamente.	Accede a la interfaz de usuario. Introduce una IP en el campo correspondiente. Observa la respuesta del sistema.	El sistema acepta la IP y procede al siguiente paso de conexión.	La IP es aceptada y el flujo de conexión continúa.
Verificar que el usuario puede bloquear el **robot** desde la interfaz al finalizar la prueba.	Ejecuta el flujo completo del sistema. Al finalizar, selecciona la opción de bloqueo en la interfaz. Observa que el robot se bloquea.	El sistema bloquea el robot y finaliza el flujo de trabajo.	El robot se bloquea al seleccionar la opción en la interfaz.

6.5. Pruebas de Comunicación

Descripción	Pasos de Ejecución	Resultado Esperado	Resultado Obtenido
Asegurarse de que la **Raspberry Pi** envía correctamente los datos de movimiento al robot virtual.	Inicia el sistema y conecta la Raspberry Pi al robot virtual. Envía datos de movimiento desde la Raspberry Pi. Observa el comportamiento del robot virtual.	El robot virtual recibe y ejecuta los comandos.	El robot virtual responde correctamente a los datos enviados.
Verificar que la **Raspberry Pi** recibe los datos de distancia desde el sensor de ultrasonido.	Coloca un objeto a una distancia conocida del sensor (por ejemplo, 25 cm). Observa la recepción de datos en la Raspberry Pi.	La Raspberry Pi recibe correctamente el valor de 25 cm.	La Raspberry Pi muestra el valor correcto de distancia.

7. ASPECTOS ÉTICOS Y SOCIALES

El desarrollo y uso de sistemas robóticos, como el de este proyecto, plantean varios aspectos éticos y sociales que es necesario considerar de manera profunda, especialmente en un contexto donde la automatización y la inteligencia artificial están cada vez más presentes en la vida diaria. Este análisis examina temas clave como la privacidad, la seguridad, el impacto en el empleo, la sustentabilidad ambiental, la responsabilidad ética en la experimentación y las posibles desigualdades de acceso que puede generar esta tecnología. Cada uno de estos temas está interconectado y plantea desafíos que requieren un enfoque ético responsable y un compromiso con el bienestar de la sociedad en su conjunto.

Uno de los aspectos éticos fundamentales en este tipo de proyectos es la privacidad y la protección de datos. Los sistemas robóticos, especialmente aquellos conectados a redes o a internet, recogen y procesan información de diferentes tipos. Aunque en este proyecto específico los datos pueden no ser sensibles, la posibilidad de que en un futuro el robot pueda equiparse con sensores o cámaras plantea la necesidad de asegurar que esta información sea manejada de manera segura y responsable. Proteger la privacidad de los datos no solo evita riesgos de seguridad, sino que también establece una relación de confianza entre los usuarios y el sistema. Esta confianza se convierte en un valor ético fundamental, ya que los usuarios deben sentirse seguros de que su información no será utilizada sin su consentimiento o caerá en manos equivocadas.

La seguridad en el uso de robots es otro punto de gran relevancia ética. Un sistema robótico como este, diseñado para operar en distintos entornos, debe contar con protocolos de seguridad que garanticen su correcto funcionamiento y que eviten poner en riesgo a personas o instalaciones en caso de fallos. Esto es particularmente importante, ya que un mal funcionamiento en el sistema podría llevar a situaciones peligrosas, tanto para los operadores como para el entorno físico. Diseñar el robot con medidas de seguridad, como bloqueos automáticos en caso de error o la posibilidad de ser desconectado de manera remota, es una responsabilidad ética clave para evitar riesgos. Asegurarse de que el sistema esté equipado con salvaguardias que minimicen el daño potencial, incluso en casos de fallo, es parte de una ética de diseño que prioriza la seguridad de las personas.

El impacto en el empleo y las condiciones laborales es otra dimensión social importante. La automatización de tareas mediante sistemas robóticos puede mejorar la eficiencia y reducir la carga de trabajo en tareas repetitivas o peligrosas. Sin embargo, esto también implica la posibilidad de que ciertas ocupaciones se vuelvan obsoletas, lo cual representa un desafío para los trabajadores. La ética en el desarrollo de tecnología robótica implica reconocer y abordar estas consecuencias, trabajando para que los beneficios de la automatización no solo sean económicos, sino también sociales. Esto podría incluir, por ejemplo, iniciativas de formación y capacitación que permitan a los trabajadores afectados adaptarse a nuevos roles en un entorno donde la tecnología es cada vez más predominante. Además, es crucial promover la creación de nuevos empleos relacionados con la robótica, como el mantenimiento y la programación de estos sistemas, y asegurar que los beneficios de la automatización lleguen a toda la sociedad.

La sustentabilidad ambiental también debe considerarse. La producción y operación de robots tiene un impacto en el medio ambiente, no solo en términos de consumo de energía,

sino también en la generación de residuos electrónicos. Para que un proyecto de este tipo sea verdaderamente responsable, es necesario que adopte prácticas sostenibles, como el uso de componentes de bajo consumo y materiales duraderos que minimicen la generación de desechos. La ética ambiental en el desarrollo de robótica implica que el proyecto debe comprometerse a reducir su huella ecológica mediante un diseño eficiente y sostenible. Además, los desarrolladores deben estar atentos a la vida útil del robot y planificar opciones de reciclaje o reutilización de sus componentes, fomentando un modelo de economía circular que limite la necesidad de nuevos recursos y reduzca los desechos.

La experimentación y las pruebas del sistema también plantean aspectos éticos significativos. En este proyecto, las pruebas deben realizarse de manera transparente y documentada, con protocolos que garanticen la seguridad en todo momento. Esta transparencia no solo favorece el desarrollo del sistema y permite su mejora continua, sino que también establece un compromiso ético con los futuros usuarios del robot, quienes deben ser conscientes de las capacidades y limitaciones del sistema. Documentar los errores y fallos de manera honesta es esencial para construir un proyecto confiable y creíble. Además, si en el futuro se realizaran pruebas en entornos donde haya personas presentes, se debe asegurar que se sigan todos los protocolos de seguridad, anticipando y minimizando cualquier riesgo que pueda surgir.

Finalmente, es importante considerar el acceso a la tecnología y las posibles desigualdades que puede generar. La robótica avanzada no está al alcance de todas las personas o empresas, lo que puede crear una brecha entre quienes tienen acceso a la tecnología y quienes no. Desde una perspectiva ética, los desarrolladores de tecnología tienen la responsabilidad de trabajar para reducir esta desigualdad y asegurar que los beneficios de la robótica estén disponibles de manera más amplia. Esto podría implicar diseñar sistemas accesibles y asequibles o compartir el conocimiento técnico para que más personas puedan beneficiarse de estas tecnologías, en lugar de reservar sus beneficios solo para unos pocos.

En conclusión, el proyecto tiene un amplio impacto ético y social que debe ser gestionado con responsabilidad y conciencia. La robótica tiene el potencial de transformar muchos aspectos de la vida humana, desde el trabajo hasta la interacción con la tecnología, pero esto solo será positivo si se realiza de manera ética y responsable. El respeto a la privacidad, la seguridad en la operación, el compromiso con la sustentabilidad y la equidad en el acceso a los beneficios de la tecnología son pilares fundamentales para asegurar que la robótica sea una herramienta de avance social y no una fuente de nuevos problemas o desigualdades.

8. CONCLUSION

Con este proyecto he conseguido desarrollar un sistema funcional que integra un conjunto de sensores y actuadores físicos con un gemelo digital en un entorno de simulación virtual. A lo largo del proceso, he logrado implementar una comunicación bidireccional eficiente entre ambos entornos, diseñar y programar la lógica de control del sistema, y realizar pruebas exhaustivas que han permitido asegurar su correcto funcionamiento. Cada parte del sistema, desde la captación de señales físicas hasta su reflejo en el entorno virtual, ha sido diseñada, validada y ajustada a lo largo de múltiples iteraciones.

Durante el desarrollo, he trabajado con diversas tecnologías que han sido fundamentales para alcanzar estos resultados. Python ha sido especialmente útil para gestionar la simulación en Webots, destacando por su claridad, simplicidad y versatilidad. Me ha permitido estructurar de forma eficiente el comportamiento del robot virtual y controlar su interacción con el entorno simulado. Webots, por su parte, ha demostrado ser una herramienta muy potente para validar el comportamiento del sistema sin depender del hardware físico en todo momento, facilitando pruebas rápidas y fiables.

En cuanto al sistema físico, la Raspberry Pi ha sido el eje central de la gestión de entradas y salidas, permitiendo coordinar todos los componentes conectados. El uso del lenguaje C para controlar directamente el hardware me ha permitido reforzar conceptos de bajo nivel, así como valorar la importancia de la eficiencia, la precisión y la optimización en sistemas de control. La combinación de ambos lenguajes ha resultado muy enriquecedora, aportando una perspectiva completa del flujo entre el control físico y su representación virtual.

Me siento muy satisfecho con el trabajo realizado. Este proyecto ha supuesto un reto importante tanto a nivel técnico como personal. A medida que avanzaba, cada prueba superada, cada error resuelto y cada pequeño ajuste implementado me acercaban a un sistema más robusto, preciso y fiable. He podido comprobar cómo la planificación, el análisis detallado y la perseverancia son elementos clave para el éxito en un entorno tan exigente como la robótica.

Además, este trabajo me ha permitido no solo reforzar habilidades en programación y control de hardware, sino también aprender a enfrentar los problemas de forma iterativa y flexible. La necesidad constante de adaptar soluciones, depurar código y entender cómo cada componente afectaba al sistema global me ha hecho crecer profesionalmente.

Más allá de lo técnico, también ha sido una oportunidad para reflexionar sobre el papel de la tecnología en la sociedad. Comprender que un desarrollo no solo debe ser funcional, sino también responsable y orientado al bienestar común, ha añadido una dimensión ética muy valiosa al proyecto.

En resumen, este proyecto no solo ha sido un reto técnico, sino una experiencia de aprendizaje integral. Me ha permitido afianzar conocimientos en programación con Python y C, profundizar en el uso de plataformas como Webots y Raspberry Pi, y desarrollar una visión más completa de lo que implica diseñar, probar y validar un sistema tecnológico real. Termino esta etapa no solo con más conocimientos, sino también con más motivación, confianza y entusiasmo por seguir explorando el mundo de la robótica y sus posibilidades.

9. TRABAJOS FUTUROS

Este proyecto representa una base sólida para seguir desarrollando sistemas de control sincronizados entre un entorno físico y uno virtual. A partir del trabajo realizado, existen varias líneas de mejora y ampliación que permitirían aumentar la complejidad, la precisión y la utilidad del sistema.

Una de las principales mejoras futuras sería añadir capacidades de aprendizaje automático o adaptación dinámica, permitiendo que el sistema ajuste su comportamiento en función del entorno detectado o del historial de interacciones. Esto abriría la puerta a simulaciones más inteligentes, capaces de anticipar y reaccionar de forma autónoma ante patrones conocidos o situaciones cambiantes.

También sería relevante ampliar el número y tipo de sensores, incorporando por ejemplo cámaras, sensores de temperatura o GPS, lo que permitiría aplicar este sistema en contextos más realistas o específicos, como navegación en interiores, monitorización ambiental o pruebas en entornos industriales. La integración con otros protocolos de comunicación más avanzados o más rápidos que el actual también podría mejorar la sincronización y escalabilidad del sistema.

Desde el punto de vista de la interfaz, se podrían incorporar sistemas de visualización más avanzados o entornos interactivos que permitan al operador recibir información más detallada y tomar decisiones en tiempo real a través de herramientas gráficas o dispositivos de control más sofisticados.

En términos sociales, el enfoque basado en gemelos digitales y comunicación bidireccional tiene un gran potencial en campos como la formación técnica, la simulación de sistemas críticos, el mantenimiento predictivo o incluso en aplicaciones médicas o de asistencia. Desarrollos como el que se ha llevado a cabo pueden ser la base para sistemas accesibles, seguros y precisos que repliquen en un entorno virtual lo que ocurre en tiempo real en el mundo físico, reduciendo riesgos y facilitando el aprendizaje, el análisis y la toma de decisiones.

En definitiva, este trabajo abre la posibilidad de continuar desarrollando soluciones técnicas con un alto grado de aplicabilidad práctica, manteniendo siempre como eje la coordinación entre sistemas físicos y virtuales. La arquitectura modular utilizada, junto con el uso de tecnologías abiertas y lenguajes versátiles, lo convierten en un punto de partida ideal para futuras investigaciones o aplicaciones reales en distintos ámbitos de la ingeniería y la tecnología.

10. REFERENCIAS

[1] Gemelos digitales:

https://www.repsol.com/es/energia-futuro/tecnologia-innovacion/gemelos-digitales/index.cshtml#:~:text=Los%20gemelos%20digitales%20o%20%E2%80%9Cdigital%20twins%E2%80%9C%20son%20un%20modelo%20virtual,real%20de%20una%20manera%20eficiente. (Último acceso: 21 de abril de 2025)

[2] Comunicación bidireccional:

https://www.inboundcycle.com/blog-de-inbound-marketing/comunicacion-bidireccional-que-es-y-como-conseguirla-en-redes-sociales (Último acceso: 21 de abril de 2025)

[3] plataforma virtual:

https://www.padcelona.com/blog/plataforma-virtual/ (Último acceso: 21 de abril de 2025)

[4] Webots:

https://cyberbotics.com/ (Último acceso: 21 de abril de 2025)

[5] Sistemas ciberfísicos:

https://syvalue.com/los-sistemas-ciberfisicos/#:~:text=Los%20sistemas%20ciberf%C3%ADsicos%20(CPS)%20son,afectan%20los%20c%C3%A1lculos%20y%20viceversa. (Último acceso: 21 de abril de 2025)

[6] Boeing:

https://www.boeing.es/ (Último acceso: 21 de abril de 2025)

[7] Tesla:

https://es.wikipedia.org/wiki/Tesla,_Inc. (Último acceso: 21 de abril de 2025)

[8] Raspberry pi:

https://raspberrypi.cl/que-es-raspberry/#:~:text=Es%20un%20peque%C3%B1o%20computador%20que,lenguajes%20como%20Scratch%20y%20Python. (Último acceso: 21 de abril de 2025)

[9] Puertos GPIO:

https://www.imaginart.es/faqs/que-es-el-puerto-gpio/#:~:text=Un%20puerto%20GPIO%20(General%20Purpose,o%20para%20cualquier%20otro%20prop%C3%B3sito. (Último acceso: 21 de abril de 2025)

[10] Lenguaje C:

https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n) (Último acceso: 21 de abril de 2025)

[11] Python:

https://es.wikipedia.org/wiki/Python (Último acceso: 21 de abril de 2025)

[12] IP:

https://www.kaspersky.es/resource-center/definitions/what-is-an-ip-address#:~:text=Una%20direcci%C3%B3n%20IP%20es%20una,Internet%20o%20una%20red%20local. (Último acceso: 21 de abril de 2025)

[13] E-puck:

https://e-puck.gctronic.com/ (Último acceso: 21 de abril de 2025)

[14] Socket:

https://www.ibm.com/docs/es/aix/7.2?topic=concepts-sockets (Último acceso: 21 de abril de 2025)

[15] librería WiringPi:

https://docs.sunfounder.com/projects/raphael-kit/es/latest/c/check_the_wiringpi_c.html (Último acceso: 21 de abril de 2025)

[16] Señales PWM:

https://solectroshop.com/es/blog/que-es-pwm-y-como-usarlo--n38?srsltid=AfmBOoocs2Nps1dS-7b0Ui44d0dXWqBIDPUXrx7xTHgvI3R_a0tSph_w (Último acceso: 21 de abril de 2025)

[17] Asignatura Sistemas en Tiempo Real:

https://www.upm.es/comun_gauss/publico/guias/2023-24/1S/GA_61CI_615000492_1S_2023-24.pdf (Último acceso: 21 de abril de 2025)

[18] Periféricos:

https://concepto.de/perifericos-informatica/#:~:text=Los%20perif%C3%A9ricos%20o%20dispositivos%20son,permitir%20la%20transmisi%C3%B3n%20de%20informaci%C3%B3n (Último acceso: 21 de abril de 2025)

[19] Comunicación SPI:

https://es.wikipedia.org/wiki/Serial_Peripheral_Interface (Último acceso: 21 de abril de 2025)

[20] Comunicación I2C:

https://hetpro-store.com/TUTORIALES/i2c/?srsltid=AfmBOorSPg2vPHigHFrcePtRPxdaAmSlw1A40nUKy0KDzeA7bzwlvWBz (Último acceso: 21 de abril de 2025)

[21] TCP/IP:

https://es.wikipedia.org/wiki/Modelo_TCP/IP (Último acceso: 21 de abril de 2025)

[22] buffer:

https://es.wikipedia.org/wiki/B%C3%BAfer_de_datos (Último acceso: 21 de abril de 2025)

11. ANEXO

Para poder probar todo el sistema se deben seguir los siguientes pasos:

1- Encender la Raspberry pi y dejar preparado para ser lanzado el programa ControlRobot.c, para ello entrar en la terminal de esta y navegar hasta la carpeta donde se encuentren el devices.c, devices.h y ControlRobot.c (paso obvio pero necesario para poner en funcionamiento el sistema). Para poder lanzar el sistema se debe poner en el fichero ControlRobot.c la dirección ip del ordenador receptor que estará con la aplicación webots y tanto este ordenador como la raspberry estar conectados a la misma red wifi.

2- Encender el programa webots y abrir el mundo collision_avoidance.wbt (en caso de que no esté abierto) desde la opción File/Open World y navegando hasta la carpeta donde se encuentre el fichero del mundo.

3- Cargar el programa controller_server_2.py en el robot e_puck del mundo (en caso de que no esté cargado) explicado en el punto 5.3 de esta documentación.

4- Teniendo ya todo listo pulsar el botón del play situado encima del mundo con forma de triangulo (se comprobará que se ha lanzado correctamente porque comenzará a correr un contado encima y en la parte de debajo aparecerá un mensaje de Esperando conexión de la Raspberry Pi...), tras esto lanzar también el programa ControlRobot.c escribiendo en la terminar de la raspberry make y ./ControlRobot.

5- Ahora aparecerá en la pantalla de la Raspberry el mensaje “Coloque el servomotor de forma recta alineado con el robot”, y para hacer esto se debe pulsar el botón y mantenerlo 3 segundos presionado cuando el aspa naranja del servomotor se encuentre mirando al mismo punto que el robot e_puck (en caso de fallar se puede dejar de presionar el botón y el servomotor volverá a moverse para volver a pulsarlo para volver a encontrar la correcta posición del servomotor).

6- A continuación, el robot comenzará a moverse y debe controlarse con potenciómetro rotatorio, controlar su velocidad con el potenciómetro deslizable y sus pausas con el botón y el sensor infrarrojos, también se podrán observar las interacciones entre ambos sistemas (físico y virtual) haciendo que el robot e_puck choque con algún objeto y la pared y utilizando el ultrasonidos.

7- Para parar el sistema simplemente hay que pulsar el botón de pausa (localizado donde se encontraba el botón del play con dos barras) y el de reinicio (situado dos posiciones a la izquierda del de pausa respectivamente y con dos triagulos y una barra).