Laboratorio de Sistemas Embebidos

Trabajo Final

- Plataforma de Atención Automatizada/Interactiva para la Optimización de Servicios mediante una Red de Área Local -

Alumnos: Juan Avilés, Gonzalo Bravo, Sandy Perez Rosa

Ingeniería en Computación. Escuela de Producción y Tecnología UNRN

Año 2025

Indice

1. Introducción

Los sistemas embebidos están presentes en la mayoría de los dispositivos con los que interactuamos en la vida cotidiana: teléfonos celulares, electrodomésticos, juguetes, aplicaciones aeroespaciales, industriales, instrumental médico, equipamiento de automóviles, etc. Por la naturaleza de las aplicaciones y sus inherentes ventajas, existe un creciente interés en utilizar redes inalámbricas (WLAN/WPAN) para la comunicación en redes de sistemas embebidos.

En este contexto se busca la utilización de una plataforma flexible que se ejecute mediante un sistema con conectividad de red entre los diferentes esquemas utilizados y que permita la comunicación, intercambio de datos con servidores u otros dispositivos. Para esto se utilizan mecanismos tales como Wifi, Bluetooth, Ethernet.

1.1 Objetivos

Poner en práctica los conocimientos adquiridos sobre sistemas embebidos de propósito específico, electrónica digital, placas de desarrollo y lenguajes de programación orientados al desarrollo de sistemas embebidos.

1.2 Alcances del proyecto

Este proyecto ofrece una plataforma que se puede adaptar a diferentes contextos sociales, como comercios, consultorios médicos, instituciones educativas, etc., brindando un servicio eficiente de atención automatizada/interactiva. Está dirigido a instituciones u organizaciones que desean automatizar y brindar un servicio eficiente para sus clientes, mediante la automatización de procesos básicos de solicitud y atención.

- **Límites del proyecto:**

  Implementación de una botonera física para el llamado de los clientes y una aplicación web para la gestión de pedidos o solicitudes, así como el control básico de un robot camarero. La funcionalidad se limita a la comunicación y gestión de pedidos básicos y el robot camarero que tendrá autonomía limitada con desplazamientos simples y la ejecución de comandos básicos de navegación. La plataforma está diseñada para operar en una red local WiFi.

- **Restricciones y dependencias:**

  El proyecto depende del correcto funcionamiento de los dispositivos físicos (botoneras, luces LED, servomotores) y su integración con la Raspberry Pi. El tiempo de desarrollo y pruebas está limitado por recursos humanos y materiales disponibles, y la estabilidad de la comunicación depende de la red WiFi local.

- **Gestión de expectativas:**

  El proyecto está planteado como una base modular que puede extenderse en el futuro para incluir funcionalidades adicionales, como múltiples robots controlados desde un único servidor, integración con sistemas externos o mejoras en la interfaz de usuario.

2. Desarrollo

Para el desarrollo de un prototipo funcional utilizamos los siguientes componentes de software y hardware:

2.1 Componentes de hardware

- Raspberry Pi 400:

  Unidad central en la botonera del sistema. Permite la activación de los botones pulsadores y controlar el encendido de luces LED correspondientes a cada mesa. También cumple el rol de cliente UDP.

- Raspberry Pi 3 Model B +:

  Ejecuta lel código en Python para recibir las señales desde el servidor y controlar los motores del robot camarero directamente mediante sus pines GPIO.

- Protoboard:

  Donde estarán ubicados los botones pulsadores y las luces LED, permitiendo realizar pruebas de conectividad y diseño.

- 3 botones pulsadores:

  Cada botón representa una mesa. Al ser presionado, genera una señal que es capturada por la Raspberry Pi 400 y se traduce en un mensaje enviado por la red. Este mecanismo simula una llamada de cliente al sistema.

- 3 luces led:

  Representan visualmente cuál mesa ha realizado una solicitud. Se encienden automáticamente al presionar un botón.

- Kit de Robot Boe-Bot (motores, sensores, batería, etc.):

  Posee motores de rotación continua utilizados para el movimiento del robot conectados directamente a la Raspberry Pi 3 para permitir el control desde Python, utilizando trenes de pulsos.

- Teléfono móvil o tablet (contiene la aplicación web):

Como dispositivo ubicado en el robot, permite a los clientes visualizar el menú y realizar pedidos mediante una aplicación web.

- Notebook (Servidor):

Dispositivo central encargado de coordinar la lógica del sistema. Alberga el servidor Flask, que recibe las solicitudes recibidas desde la botonera y enviadas hacia el robot (a través de peticiones HTTP y UDP).

- Fuentes de alimentación externa / Baterías:

Utilizadas para alimentar tanto los servomotores como las placas Raspberry Pi.

2.2 Componentes de software

2.2.1 Sistema Operativo

Para el desarrollo de este proyecto se utilizó el Sistema Operativo Linux basado en la distribución Debian **Raspberry Pi OS**, el cual proporciona un entorno estable, ligero y compatible con Python y bibliotecas específicas para manejo de hardware embebido. La instalación se realizó a través de la máquina virtual **VirtualBox** de Oracle.

Imagen. Sistema Operativo Raspberry PI OS

Luego creamos un **entorno virtual aislado de Python (venv)** para instalar las librerías y dependencias específicas del proyecto sin afectar al resto del sistema ni modificar el entorno global.

Imagen. Entorno Virtual de Python activado

2.2.2 Lenguajes de Programación y Frameworks

El software fue desarrollado en el Lenguaje de Programación **Python**, tanto para la comunicación entre dispositivos como el comportamiento del robot.

Para implementar el servidor web y gestionar los pedidos enviados desde la aplicación. Utilizamos el Framework de Desarrollo Web **Flask**.

Para diseñar la interfaz de la aplicación web que utilizan los clientes para seleccionar los pedidos utilizamos **HTML, CSS y JavaScript**.

Para establecer la comunicación entre procesos utilizando protocolos de red se utilizó el módulo **Socket (comunicación UDP y TCP)**.

2.2.3 Librería Rpi.GPIO

Esta librería específica de la Raspberry Pi, permite controlar los **pines GPIO** de la placa. Fue utilizada para gestionar la lectura de los botones físicos, el encendido y apagado de los LED y el envío de trenes de pulsos a los servomotores.

3. Funcionamiento del sistema y requisitos específicos

El proyecto consiste en una colección de sistemas con capacidad de cómputo independientes interconectados en red que aparentan ser un único sistema coherente para los usuarios.

3.1 Requisitos funcionales

- **Envío de mensajes a través de una red WiFi** local desde los dispositivos externos (botonera con Raspberry Pi 400) hacia un servidor intermedio, mediante el protocolo UDP/IP.

- **Iluminación de LED de estado**, indicando qué mesa ha realizado un pedido.
- Recepción de mensajes por parte del robot (Raspberry Pi 3), que actúa como servidor UDP y reacciona ante cada llamado de mesa.
- Interacción del cliente-robot mediante una aplicación web, que le permite seleccionar una comida. Al confirmar el pedido, se envía un mensaje al servidor principal (notebook) mediante el protocolo HTTP (POST), que utiliza TCP como protocolo de transporte.
- **Procesamiento de pedidos en el servidor web (notebook)**, donde se registran los pedidos recibidos y se coordina la preparación de los mismos.
- **Desplazamiento físico del robot en el entorno**, desde su posición actual hasta la ubicación de la mesa que realizó la llamada. Este movimiento está diseñado para seguir un algoritmo de búsqueda de caminos A*, adaptándose a obstáculos o cambios en el entorno.
- **Escalabilidad del sistema**, permitiendo la incorporación de más mesas (botones) o más robots sin necesidad de rediseñar toda la arquitectura.

3.2 Requerimientos no funcionales

• Requisitos de rendimiento:

Tiempo de respuesta inmediato, baja latencia en la red WiFi local y carga ligera del servidor Flask.

• Requisitos de seguridad

Red WiFi protegida por contraseña, restricciones y validaciones de entrada y acceso al servidor web.

• Requisitos de fiabilidad

Robustez en la comunicación UDP, monitoreo del estado de los dispositivos y tolerancia a fallos.

• Requisitos de usabilidad

Interfaz web intuitiva y accesible, mensajes de confirmación en pantalla.

• Requisitos de escalabilidad

Posibilidad de agregar más mesas (botoneras), posibilidad de controlar múltiples robots y realizar un desarrollo modular del software para posibles ampliaciones y funcionalidades.

3.3 Requisitos de interfaz externa

• Interfaz de Usuario

Interacción del cliente mediante una botonera física, interfaz web responsiva alojada en el robot y notificaciones visuales tanto en la botonera (luces LED) como en la aplicación web.

• Interfaz de Hardware

Botones y LEDs conectados a la Raspberry Pi 400, servomotores conectados a la Raspberry Pi 3 controlados por GPIO y conexión de red para la comunicación entre todos los dispositivos.

• Interfaz de Software

Protocolo UDP para comunicación entre botonera y robot, protocolo HTTP REST (POST/GET) entre aplicación web y servidor Flask y librerías RPi.GPIO para manejo de hardware, socket para comunicación UDP, Flask y Flask-CORS para el servidor web

• Interfaz de Comunicaciones

Todos los dispositivos están conectados a una red WiFi con direcciones IP y puertos específicos

3.4 Interfaz de usuario y casos de uso

Se pueden presentar diferentes escenarios en donde los usuarios interactúan con el sistema, mediante acciones de la interfaz y las respuestas esperadas del software:

• Llamado de cliente/mesa mediante botonera física

• Selección y envío de pedido mediante aplicación web en robot

• Respuesta del robot ante el pedido recibido

4. Implementación

La implementación integra los diversos componentes de software y hardware del sistema. Se establecen interfaces definidas entre los dispositivos y se incorporan librerías y protocolos estándares.

4.1 Implementación a nivel de hardware

- Armado de componentes y conexiones entre dispositivos.
- Tipo de arquitectura distribuida (varias unidades independientes que se comunican entre sí a través de una red) y asincrónica (sistema asíncrono con funcionamiento independiente y paralela entre sus componentes).
- Tareas específicas de las Raspberry Pi:
  -Raspberry Pi con botonera que detecta llamados (cliente UDP).

  -Raspberry Pi que actúa como robot camarero (servidor UDP y cliente HTTP).
- Notebook servidor que recibe los pedidos y los muestra en pantalla (servidor HTTP con Flask). Luego envía mensajes al robot camarero.
- Robot móvil con tracción diferencial, equipado con dos ruedas motrices controladas mediante señales GPIO desde una placa Raspberry Pi, posee autonomía de navegación empleando el algoritmo de búsqueda A* implementado en Python.

4.2 Implementación a nivel de software

- Código en lenguaje de programación Python, utilizando un Modelo “Cliente-Servidor”. Librerías específicas para el desarrollo de sistemas embebidos.
- Código en HTML, CSS, Javascript para la implementación de una aplicación web interactiva que permite el envío de mensajes al servidor por parte del cliente.
- Servidor Web desarrollado en Python con el Framework Flask, el cual provee implementaciones de clientes y servidores HTTP, utilizados para construir la funcionalidad de cada endpoint.
- Comunicación con servidores externos, empleando el protocolo REST. Secciones de código en las cuales se realizan peticiones GET y POST a un servidor dentro de la red local.
- Sockets orientados a conexión que permiten la comunicación entre los procesos clientes y procesos servidor.
- Librerías utilizadas RPi.GPIO, socket, Flask, Flask-CORS.

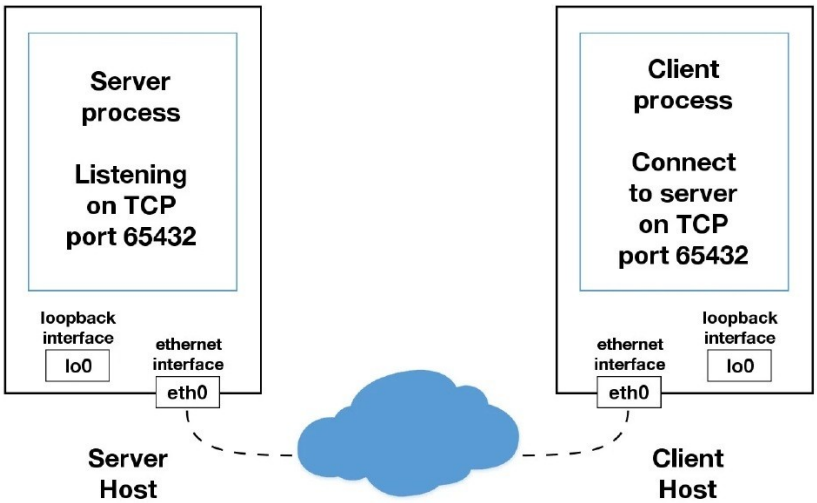


Figura. Conexión a través de una red

4.3 Protocolo UDP/IP

- El **protocolo UDP (User Datagram Protocol)** se utiliza para la comunicación entre la **botonera (cliente UDP)** y la **Raspberry Pi que controla al robot (servidor UDP)**. Esta elección se basa en la rapidez del protocolo y en que no se requiere una conexión persistente ni confiabilidad garantizada, ya que los mensajes son simples.
- Se utilizan **direcciones IP** dentro de la red WiFi local para identificar cada dispositivo (botonera, robot y servidor Flask).
- La comunicación basada en UDP es adecuada para mensajes de control o eventos breves generados al presionar un botón físico.

4.4 Protocolo TCP/IP

El protocolo TCP/IP (Transmission Control Protocol / Internet Protocol) es una arquitectura de protocolos que permite la comunicación entre dispositivos en redes distribuidas, como la red WiFi utilizada en este proyecto. En particular, el protocolo TCP (Transmission Control Protocol) proporciona una transmisión de datos fiable, ordenada y libre de errores, lo cual es fundamental para asegurar la entrega correcta de pedidos dentro del sistema.

En el contexto del proyecto:

- **TCP/IP es utilizado por la aplicación web** que se ejecuta en la pantalla del robot (por ejemplo, un smartphone o tablet). Cuando el cliente realiza un pedido seleccionando una opción del menú, la solicitud se envía al **servidor Flask** alojado en la notebook mediante el método **HTTP POST**, que se basa en TCP como protocolo de transporte.
- TCP garantiza que la información del pedido llegue completa y en orden al servidor, sin pérdidas, lo cual es esencial para evitar errores en la preparación y entrega de los pedidos.
- Esta comunicación se realiza dentro de la **red WiFi local**, utilizando direcciones IP privadas para identificar cada dispositivo conectado.

4.5 Algoritmo A*

El prototipo funcional del robot camarero es desarrollado con un robot móvil con tracción diferencial, equipado con dos ruedas motrices y servomotores controlados mediante señales GPIO desde una placa Raspberry Pi. Su diseño está orientado a tareas de navegación autónoma en entornos delimitados, utilizando sensores y controladores de bajo nivel típicos de los sistemas embebidos.

El algoritmo implementado implementado en Python para las acciones es de búsqueda **A*** (A estrella), para calcular rutas con obstáculos simulando la planificación de trayectorias en tiempo real. La lógica de navegación se integra con funciones de movimiento específicas (adelante, atras, izquierda, derecha, detener) que controlan los motores según el resultado del algoritmo, permitiendo al robot alcanzar un objetivo definido evitando obstáculos.

Este algoritmo también contempla un escenario de detección de obstáculos, en cuyo caso se modifica dinámicamente la matriz de navegación para recalcular la ruta.

La arquitectura del robot, compuesta por hardware básico de movilidad, controladores GPIO, y una lógica embebida autónoma, lo clasifica como una plataforma robótica móvil autónoma con navegación planificada.

4.6 Implementación del proyecto en contexto real

Contexto: simulación de un restaurante

1. Se presenta una botonera para cada una de las mesas. Los botones y luces LED, están ubicados en un protoboard.

2. Cuando un cliente presiona un botón se enciende una luz LED (correspondiente a ese botón) y se envía un mensaje a través de la red WiFi local desde los dispositivos externos hacia un servidor alojado en una notebook.

3. Cuando los mensajes llegan al servidor, un "robot camarero" responde a las peticiones recibidas a través de la red WiFi y se dirige hacia el lugar desde donde fue llamado para interactuar con el cliente a través de una aplicación web.

4. El robot tiene una ubicación espacial y desplazamiento a través del entorno para la realización de las órdenes recibidas, además lleva una pantalla táctil (aplicación web en teléfono móvil) en donde los clientes podrán realizar su selección a través de un menú de opciones, y enviar mensajes al servidor de la notebook. Al confirmar la orden, se envía un mensaje mediante **HTTP (POST)** al **servidor web (Flask)** alojado en la notebook, utilizando **TCP** como protocolo de transporte, lo cual garantiza la entrega confiable del pedido.

5. Luego el robot vuelve hacia el lugar en donde está ubicada la notebook para recibir el pedido y entregarlo a los clientes.

La complejidad se encontrará directamente en el cliente del robot. Las instrucciones enviadas por el servidor serán simples y el robot tendrá su propia autonomía. Una ventaja de esta implementación es la posibilidad de que un único servidor pueda controlar múltiples robots de darse el caso.

4.7 Antecedentes de proyectos similares

Como ejemplo de un proyecto similar ya implementado, hemos encontrado el siguiente ejemplo de un café en Shibuya Tokio, Japón, atendido por robots humanoides. En este comercio los robots colaboran con el personal humano, creando un ambiente de trabajo dinámico. Su función es atender a los clientes y al mismo tiempo ofrecer servicios de entretenimiento, permitiéndoles experimentar un nuevo nivel de interacción con la robótica.

Enlace al sitio web:

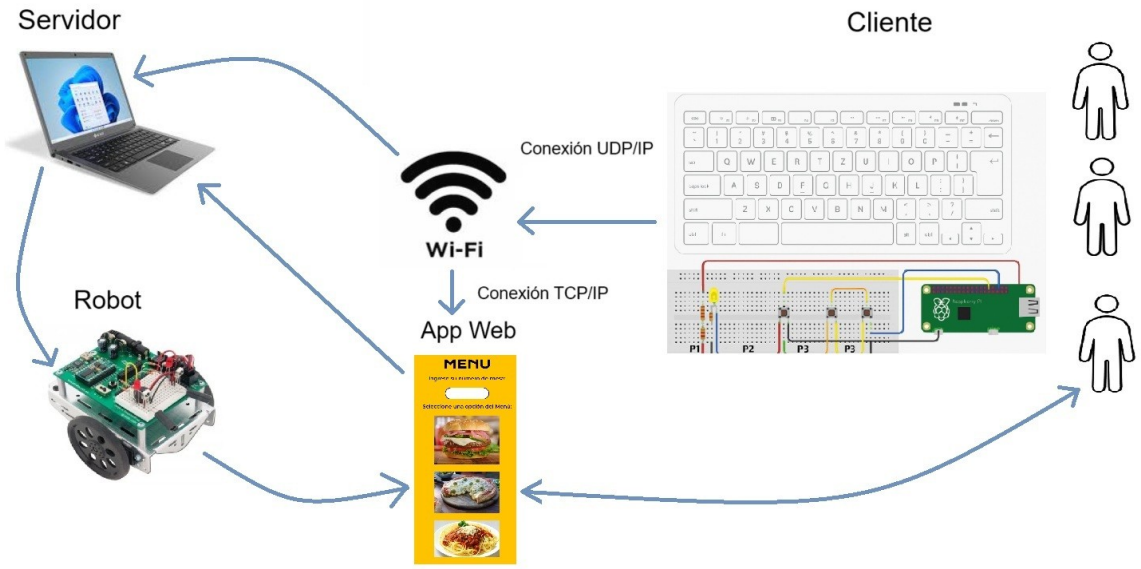
https://www.pepperparlor.com/

Enlaces con especificaciones técnicas de los robots:

https://www.robotlab.com/higher-ed-robots/store/pepper-robot-for-research-and-coding
https://www.wevolver.com/specs/softbank-robotics-pepper

5. Diagramas

5.1 Diagrama esquemático



5.2 Diagrama de conexiones

- Para el conexionado del circuito botones pulsadores y luces led utilizamos una placa de desarrollo Raspberry Pi 400.
- Para el conexionado del robot camarero utilizamos una placa de desarrollo Raspberry Pi 3 Model B +.

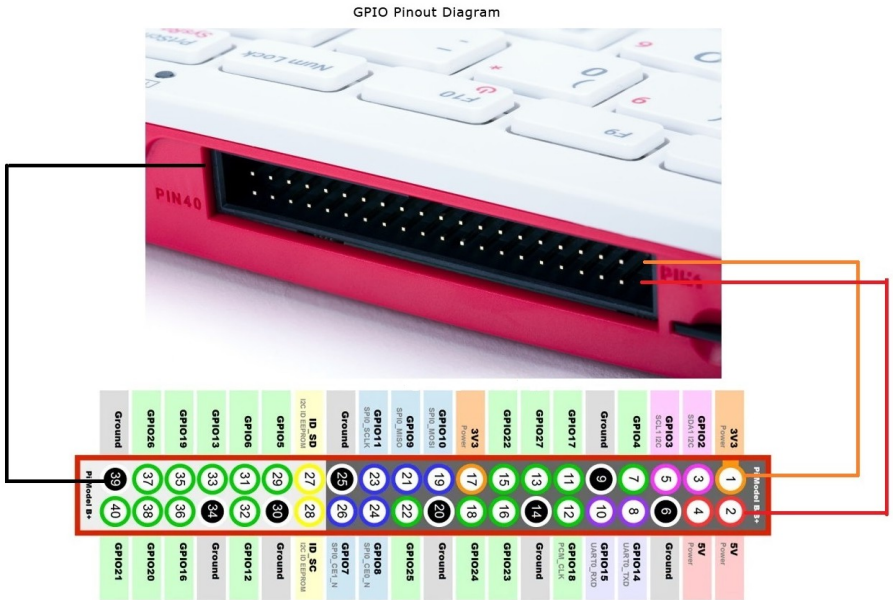


Imagen. Diagrama de Pines Rasperry Pi 400

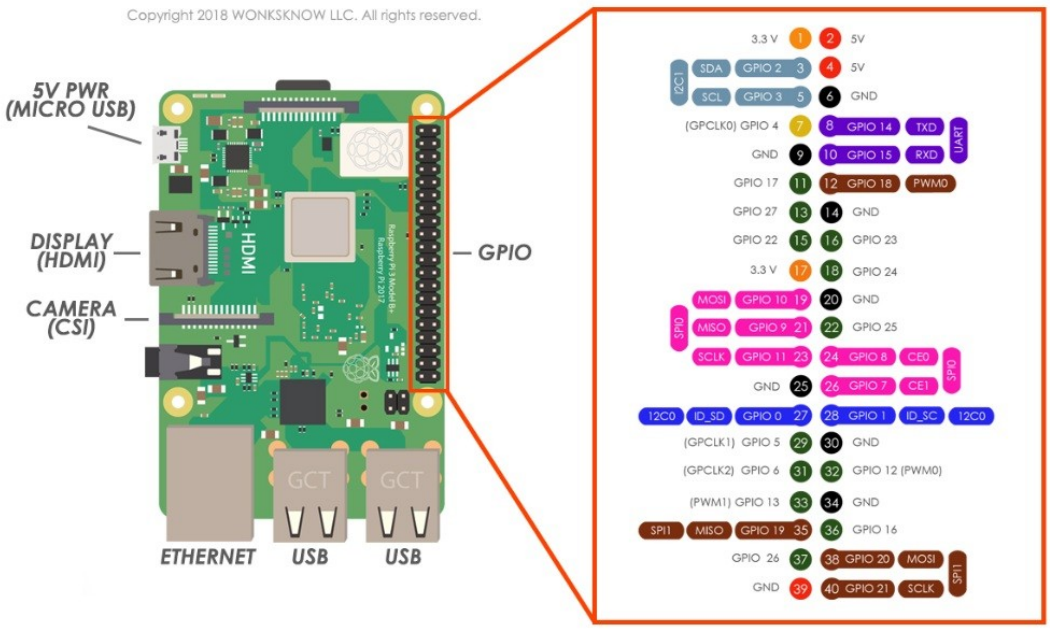


Imagen. Diagrama de Pines Rasperry Pi 3

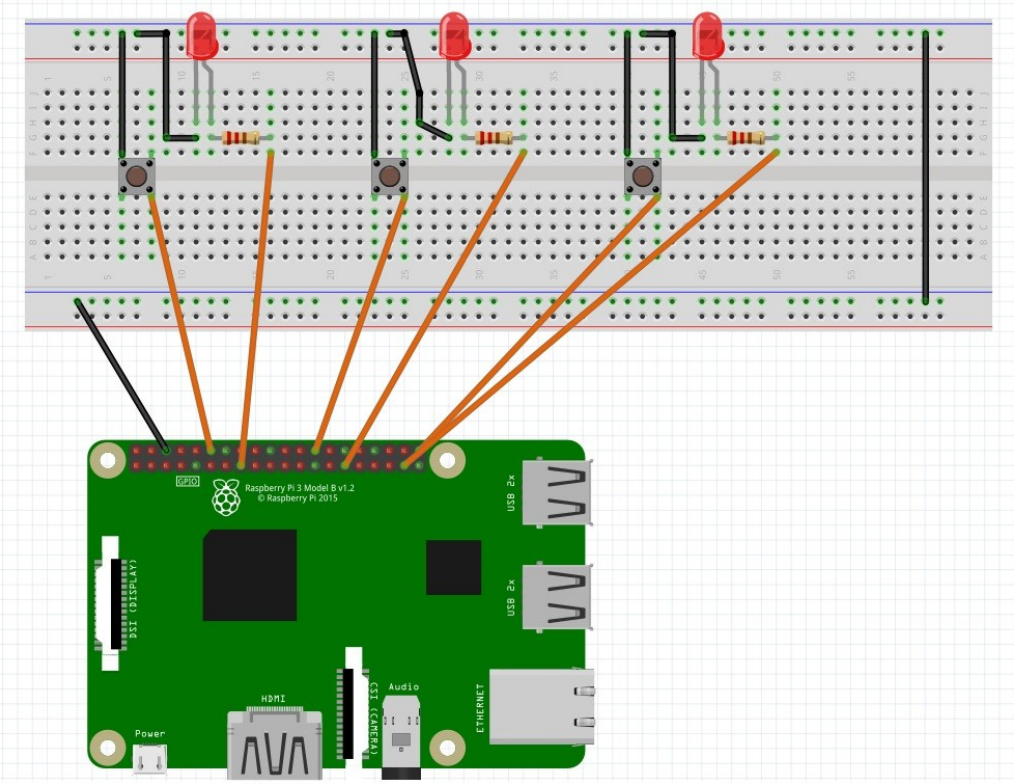


Imagen. Diseño de placa de pruebas

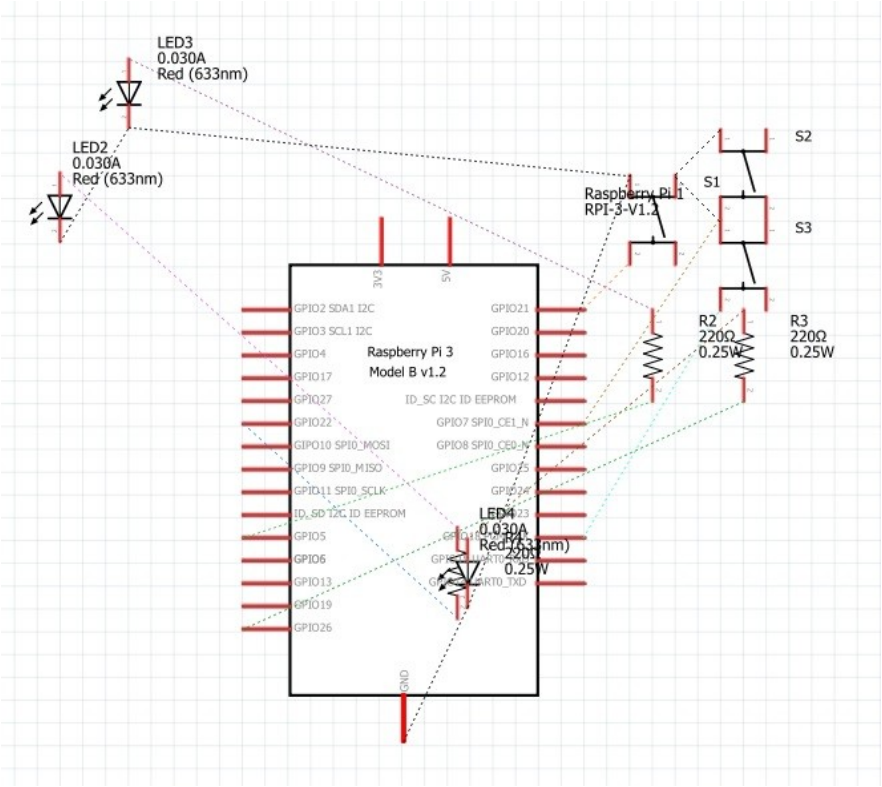


Imagen. Diseño esquemático

6. Análisis en tiempo de ejecución

6.1 Análisis de pruebas

Dimos inicio a la ejecución del proyecto según la planificación realizada con anterioridad:

- Como primer punto a resaltar, se observan resultados positivos en la interacción entre los diferentes dispositivos: botonera, luces led indicadoras de cada mesa y el servidor alojado en la notebook. Los tiempos de respuesta muestran un funcionamiento correcto, los botones envían una señal al servidor como se había planeado en un principio, enviando el mensaje correspondiente hacia el servidor principal.

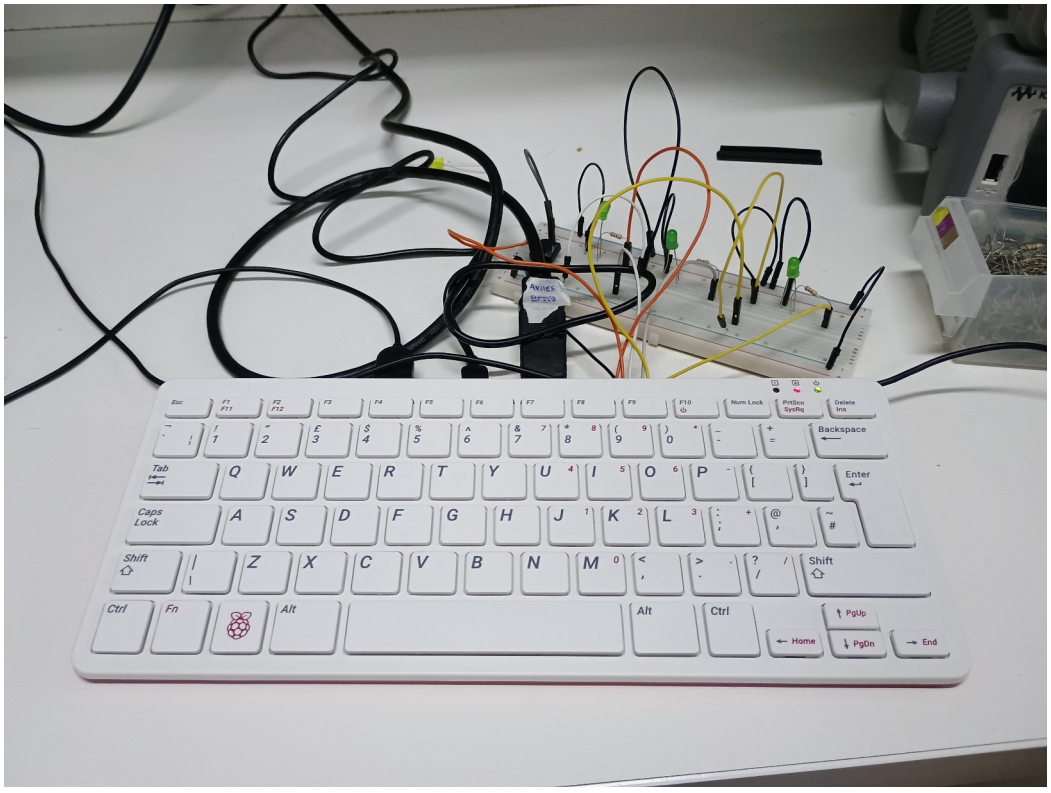


Imagen. Conexión con Raspberry Pi 400

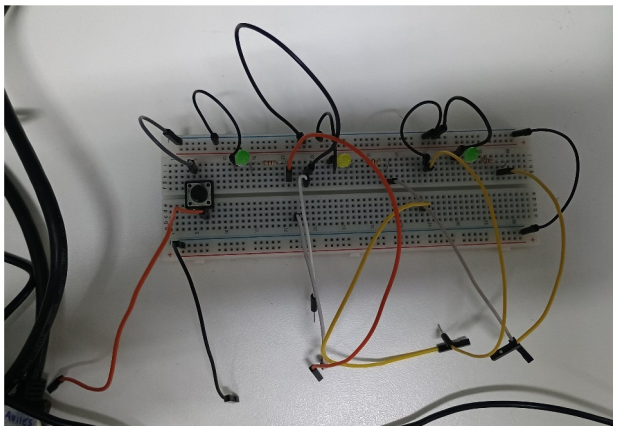


Imagen. Conexión protoboard, botones pulsadores y luces led

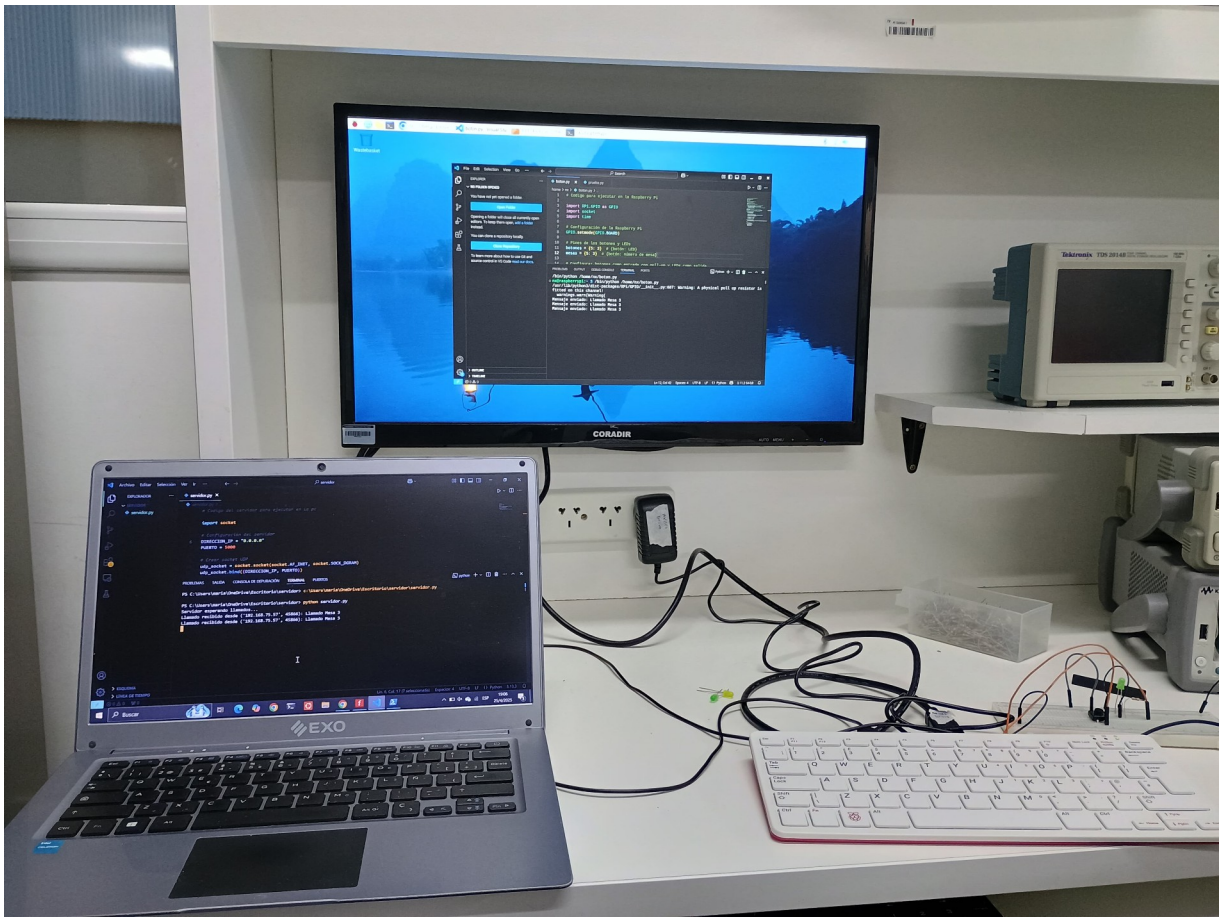


Imagen. Envío y recepción de mensajes desde los botones pulsadores hacia el servidor

- La aplicación web para alojada en un dispositivo móvil montado sobre el robot muestra un funcionamiento correcto en todos los casos de prueba, enviando los pedidos hacia el servidor.

Imagen. Selección de pedidos en app web

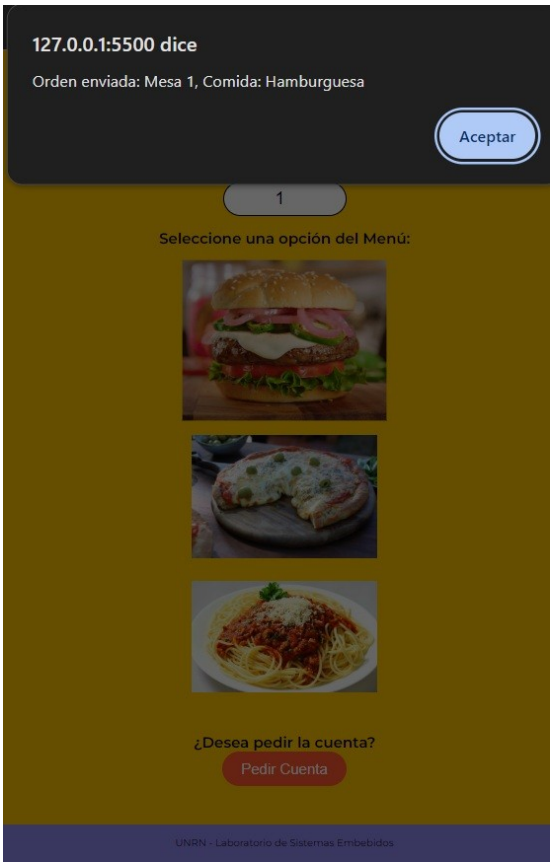


Imagen. Envío de pedidos en app web

```
Servidor esperando llamados...
 * Serving Flask app 'servidor'
 * Debug mode: on
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.1.103:5000
Press CTRL+C to quit
192.168.1.103 - - [16/Jun/2025 22:19:24] "OPTIONS /orden HTTP/1.1" 200 -
Orden recibida: Mesa 1, Pedido: Hamburguesa
192.168.1.103 - - [16/Jun/2025 22:19:24] "POST /orden HTTP/1.1" 200 -
192.168.1.103 - - [16/Jun/2025 22:19:39] "OPTIONS /orden HTTP/1.1" 200 -
Orden recibida: Mesa 2, Pedido: Pizza
192.168.1.103 - - [16/Jun/2025 22:19:40] "POST /orden HTTP/1.1" 200 -
Orden recibida: Mesa 3, Pedido: Fideos
192.168.1.103 - - [16/Jun/2025 22:19:43] "POST /orden HTTP/1.1" 200 -
192.168.1.103 - - [16/Jun/2025 22:19:52] "OPTIONS /orden HTTP/1.1" 200 -
Orden recibida: Mesa 1, Pedido: Cuenta
192.168.1.103 - - [16/Jun/2025 22:19:52] "POST /orden HTTP/1.1" 200 -
```

Imagen. Recepción de pedidos en el servidor

- El algoritmo de búsqueda de caminos es funcional y se adapta al encontrar obstáculos imprevistos por el camino.

6.2 Problemáticas, obstáculos y posibles soluciones implementadas

El desafío principal con el que nos encontramos durante el desarrollo del proyecto fue durante el desarrollo del hardware del robot (en este caso el modelo Boe-Bot de Parallax), específicamente con la configuración de *los servomotores*.

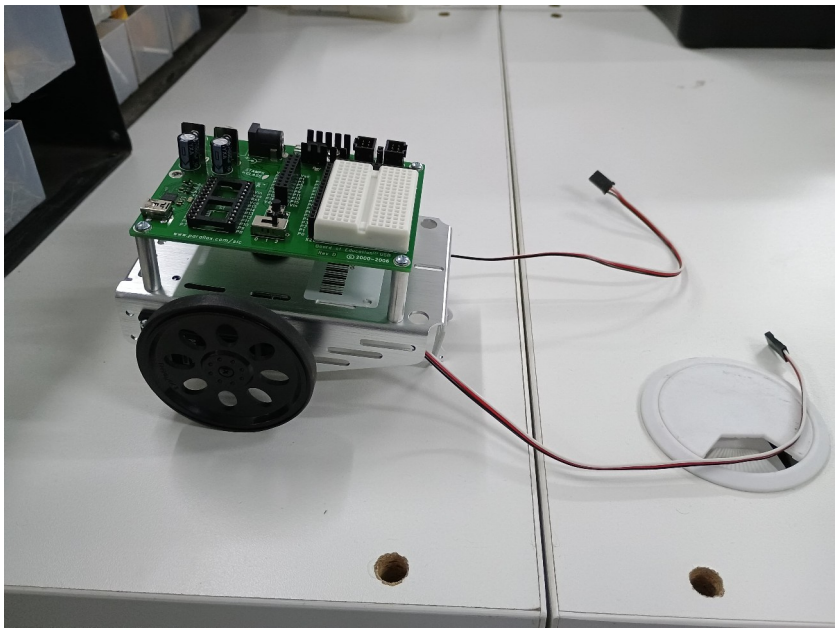


Imagen. Robot con los servomotores ensamblados

A continuación una explicaremos las características de la placa del robot, servomotores, las dificultades que se presentaron y cómo intentamos resolverlas:

La placa de este modelo de robot en específico construído con fines didácticos, fue pensada para ser utilizada con un chip de memoria propio, el cual funcionaba con lenguaje de programación PBASIC. Mediante la programación en PBASIC, logramos que el robot se moviera. El siguiente paso fue implementar el algoritmo de búsqueda desarrollado en Python. Puesto que determinamos que el microcontrolador de la Basic Stamp no era lo suficientemente potente para ejecutarlo y a este punto la interconexión con una Raspberry Pi 3 era inviable, además de que iban surgiendo problemas frecuentes en la conexión con la placa, decidimos descartamos del uso del lenguaje de programación PBASIC y el microcontrolador de la placa para mover los motores y apostamos por Python, utilizando la Raspberry Pi 400.

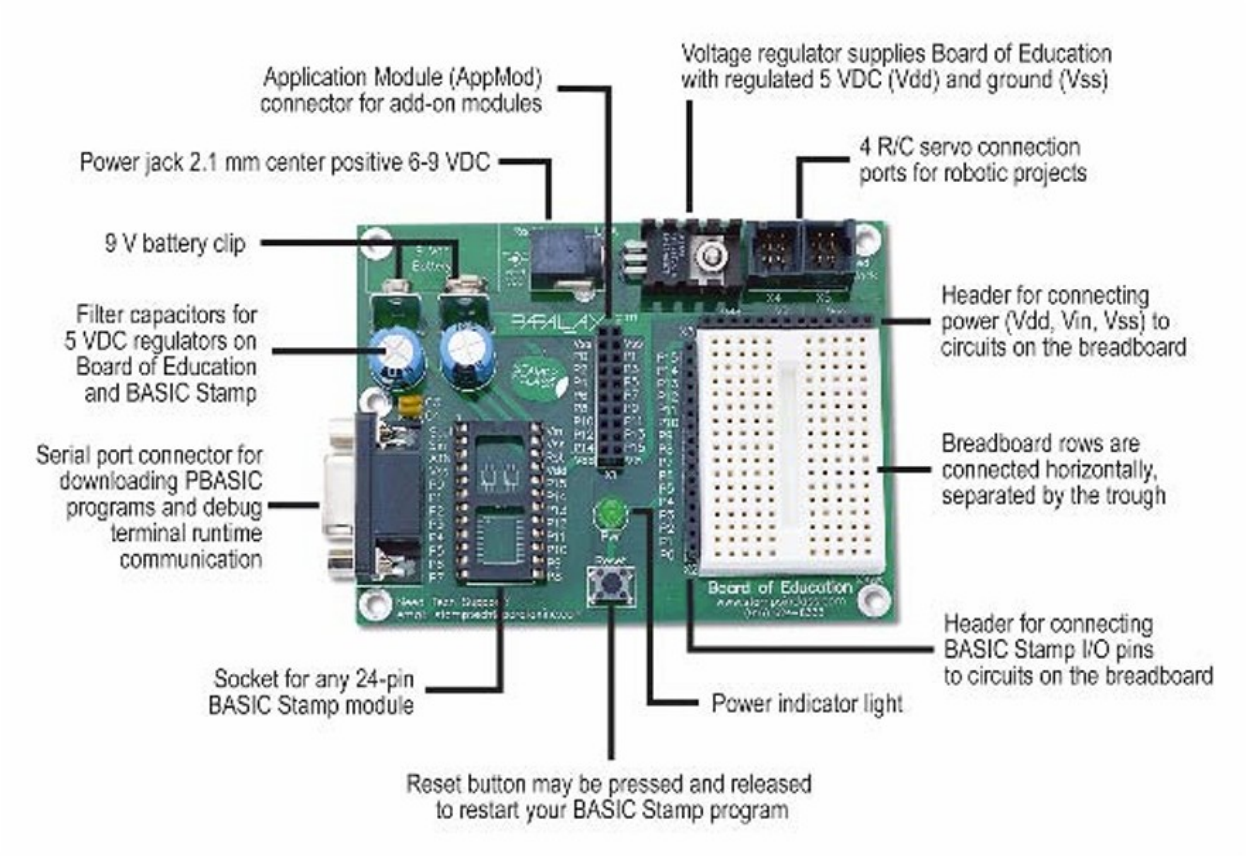
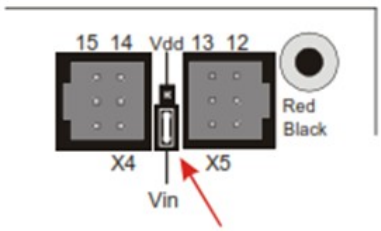


Imagen. Entradas del Application Module

Como se ve en la imagen, decidimos trabajar con las entradas del Application Module. En esa sección de la placa están numerados todos los pines y el lugar donde se conectan. Los pines más importantes (P12 y P13) son los que operan los motores y por otro lado tenemos los de Vss que están marcados como Tierra. Lo principal era conectar el GND de la Raspberry al Vss para que lo tome como referencia y las salidas GPIO a utilizar en los pines P12 y P13 para controlar los motores. A partir de estas cuestiones, entraremos más en detalle a continuación:

Inmediatamente el funcionamiento de los motores demostró ser distinto a HIGH y LOW para avanzar y detenerse.

En la siguiente imagen se explica cómo funciona la placa para cambiar su configuración de alimentación si se estarían utilizando baterías o una fuente de alimentación. Los servos estabán conectados a X5 que corresponden a P12 Y P13:



Seleccione Vin si está usando el paquete de batería que viene con los kits Boe-Bot.

Seleccione Vdd si está usando una fuente CD alimentada por un contacto CA (adaptador CA).

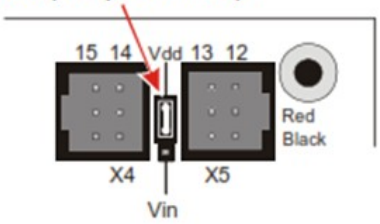
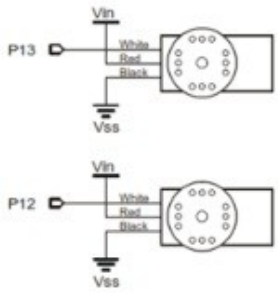


Figura 2-12
Seleccione la fuente de energía de sus servo puertos en el Board of Education

Aquí se pueden ver los tres cables que corresponden con fuente, tierra y el I/O:



El control de los servos:

Los motores deben conectarse a una corriente continua de 5V para poder funcionar y responden a comandos en forma de trenes de pulso. Por ejemplo el comando de trenes de pulso que le indica que debe detenerse es un pulso de 1,5 milisegundos, seguido de 20 milisegundos de bajo para luego repetirse como se muestra en la siguiente figura:

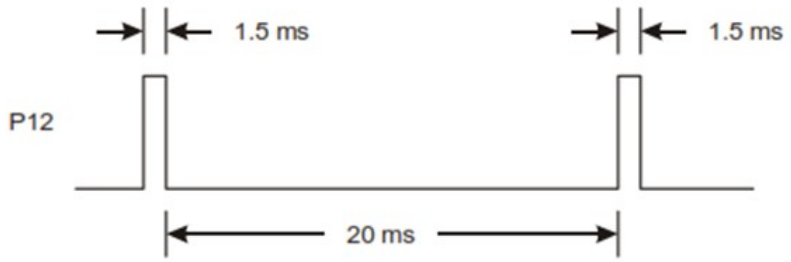


Figura 2-22
diagrama de tiempo para CenterServoP12.bs2

Los pulsos de 1.5 ms instruyen al servo a quedarse quieto.

De igual manera requiere de otro tren de pulsos para moverse en sentido de las agujas del reloj o contra este. Por ejemplo los pulsos menores a 1,5 milisegundos causarán que gire a la derecha y los pulsos mayores hacia la izquierda.

El ancho de Pulso Controla Velocidad y Dirección

Al centrar los servos aprendimos que una señal con un ancho de pulso de 1.5 ms causó que los servos se quedaran quietos. Esto fue hecho usando un comando **PULSOUT** con una ***Duration*** de 750. ¿Qué habría pasado si el ancho de pulso no hubiera sido 1.5 ms?

En la sección Su Turno de la Actividad #2, programó el BASIC Stamp para enviar una serie de pulsos de 1.3 ms a un LED. Veamos más de cerca a la serie de pulsos y averigüemos cómo puede usarse para controlar un servo. La Figura 2-25 muestra como un servo de Rotación Continua Parallax gira a velocidad plena hacia la derecha cuando le envía pulsos de 1.3 ms. La velocidad plena varía entre 50 a 60 RPM.

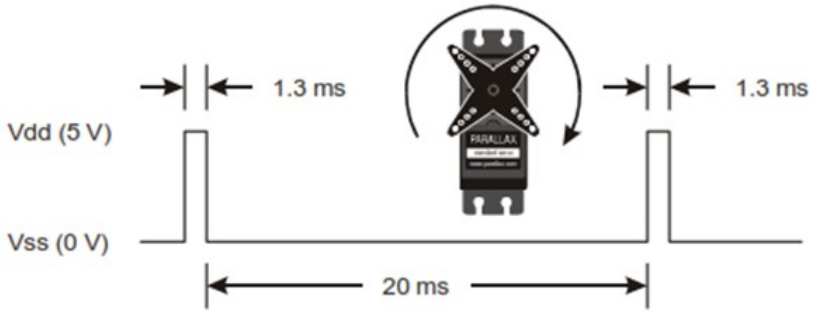


Figura 2-25
Un tren de pulsos de 1.3 ms hace girar al Servo a la derecha a velocidad plena

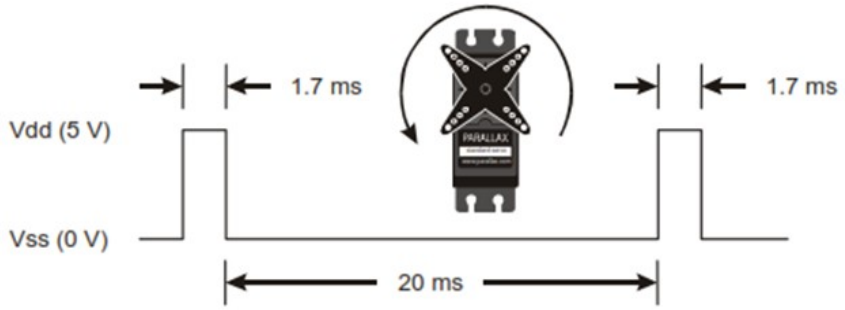


Figura 2-26
Un Tren de Pulsos de 1.7 ms hace que el Servo gire a velocidad plena hacia dla izquierda

Con esto en mente, se realizaron las conexiones y el programa de pulsos en Python usando la librería RPI.GPIO, y se implementó el tren de pulsos correspondiente. Sin embargo, la respuesta de los motores no fue la esperada. Cuando era conectada la referencia de tierra a Vss, los servos no respondieron a ningún tren de pulsos. Tampoco cuando se probó con ambos motores por separado. Lo extraño fue que al desconectar la referencia del GND los motores comenzaron a moverse pero no seguían las instrucciones del tren de pulsos como era debido.

Se verificó con un osciloscopio si el tren de pulsos enviado era correcto y efectivamente pudimos comprobar que lo eran. Por último, se decidió conectar directamente los motores a una

fuente y a la Raspberry, sospechando que tal vez esto sucedía porque la placa en la que estaban conectados deformaba la señal de alguna manera, pero el resultado fue el mismo.

Finalmente, no encontramos la manera de hacer que los motores sigan las instrucciones. Llegamos a la conclusión que: o bien los motores funcionaban mal o hay una parte importante del funcionamiento de los mismos que no está especificada en el manual.

Si bien no llegamos a realizar una prueba física del robot debido al problema con los motores, no es difícil creer que de haberlos hecho funcionar se podría haber ajustado el código para que se muevan como era esperado, e implementar el algoritmo de búsqueda A* para que se mueva de un punto a otro sin problema. Con unos motores más simples de 2 cables donde la dirección depende del lado de la entrada, la propuesta planteada pudo haber funcionado.

7. Conclusión

El desarrollo de este proyecto permitió aplicar de manera integral los conocimientos adquiridos sobre sistemas embebidos, microcontroladores, redes de comunicación, programación en Python y diseño de interfaces web. La propuesta de una plataforma de atención automatizada/interactiva demostró ser viable en entornos simulados, mostrando tiempos de respuesta adecuados y una correcta interacción entre los distintos módulos distribuidos en red.

A pesar de las dificultades encontradas, especialmente con los servomotores del robot y la compatibilidad con la placa original del Boe-Bot, se logró adaptar el sistema mediante el uso de una Raspberry Pi, ampliando así las posibilidades de programación y control del hardware. Este obstáculo permitió profundizar en el análisis de señales PWM, trenes de pulsos y diagnóstico de funcionamiento de servomecanismos.

El sistema demostró eficiencia en la comunicación asincrónica entre cliente y servidor mediante el protocolo UDP/IP, así como confiabilidad en el envío de datos críticos mediante HTTP sobre TCP/IP. La arquitectura modular del sistema lo convierte en una solución escalable, adaptable a múltiples contextos, como restaurantes, clínicas o instituciones.

Como aprendizaje principal, se destaca la importancia de elegir componentes de hardware plenamente compatibles con los entornos de desarrollo y protocolos requeridos, así como la necesidad de documentación técnica clara para su integración.

Finalmente, si bien no fue posible implementar la navegación autónoma del robot por completo debido a las limitaciones técnicas encontradas, se dejó planteada la base funcional del sistema con su infraestructura de comunicación lista para futuras mejoras.

8. Anexos de código

El código fuente del proyecto puede encontrarse en el siguiente enlace de GitHub: https://github.com/GonzaloBravo/TP-Final-Laboratorio-Sistemas-Embebidos-2025.git

9. Referencias

Documentación oficial y técnica

- Python Software Foundation. (s. f.). *Documentación de Python 3*. https://docs.python.org/es/3/
- Python Software Foundation. (s. f.). *socket — Network socket interface. Python documentation*. https://docs.python.org/3/library/socket.html
- Python Software Foundation. (s. f.). *venv — Creation of virtual environments*. https://docs.python.org/3/library/venv.html
- Read the Docs. (s. f.). *Documentación de Flask en español*. https://flask-es.readthedocs.io/
- Drogon. (s. f.). *Raspberry Pi – WiringPi – Pinout*. https://projects.drogon.net/raspberry-pi/wiringpi/pins/
- YoungWonks. (s. f.). *Raspberry Pi 3 Pinout*. https://www.youngwonks.com/blog/Raspberry-Pi-3-Pinout

---

Manuales y PDFs técnicos:

- TecnoEdu. (s. f.). *Montaje y programación del Boe-Bot (Parte 1)* [PDF]. https://tecnoedu.com/Download/01MontajeYProgramacionBoeBot.pdf
- TecnoEdu. (s. f.). *Montaje y programación del Boe-Bot (Parte 2)* [PDF]. https://tecnoedu.com/Download/02MontajeYProgramacionBoeBot.pdf
- Raspberry Pi Foundation. (2019). *BCM2711 ARM Peripherals* [Datasheet]. https://datasheets.raspberrypi.com/bcm2711/bcm2711-peripherals.pdf
- Microchip Technology Inc. (2021). *Curiosity HPC Board Schematics Rev. 2* [Esquema PDF]. https://ww1.microchip.com/downloads/aemDocuments/documents/MCT08/ProductDocuments/BoardDesignFiles/Curiosity_HPC_Schematics_rev2.pdf
- Microchip Technology Inc. (2020). *PIC16(L)F18855/75 Data Sheet* [Datasheet PDF]. https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/PIC16%28L%29F18855-75-Data-Sheet-40001802H.pdf

---

Hardware y componentes:

- Parallax Inc. (s. f.). *Boe-Bot Robot*. https://www.parallax.com/boe-bot-robot/
- Parallax Inc. (s. f.). *Parallax Continuous Rotation Servo Downloads*. https://www.parallax.com/package/parallax-continuous-rotation-servo-downloads/

Estándares IEEE:

- IEEE. (1998). *IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications*. https://doi.org/10.1109/IEEESTD.1998.88286
- IEEE. (2009). *IEEE Std 1016-2009: IEEE Standard for Information Technology—Systems Design—Software Design Descriptions*. https://doi.org/10.1109/IEEESTD.2009.5353440