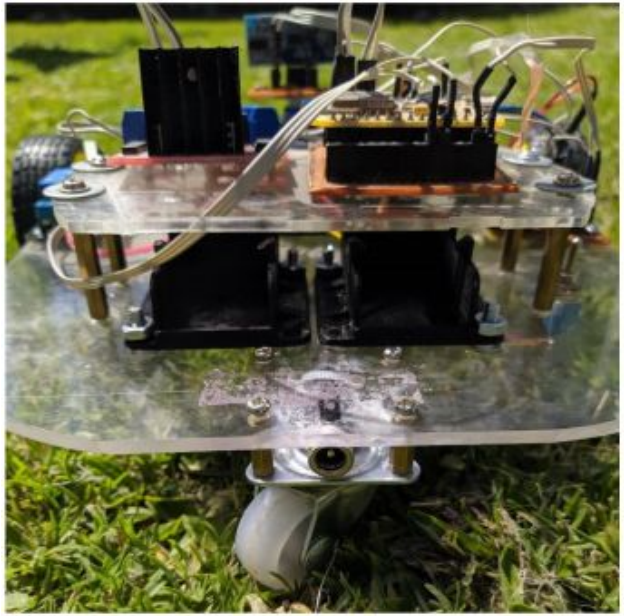
Taller de Proyecto II

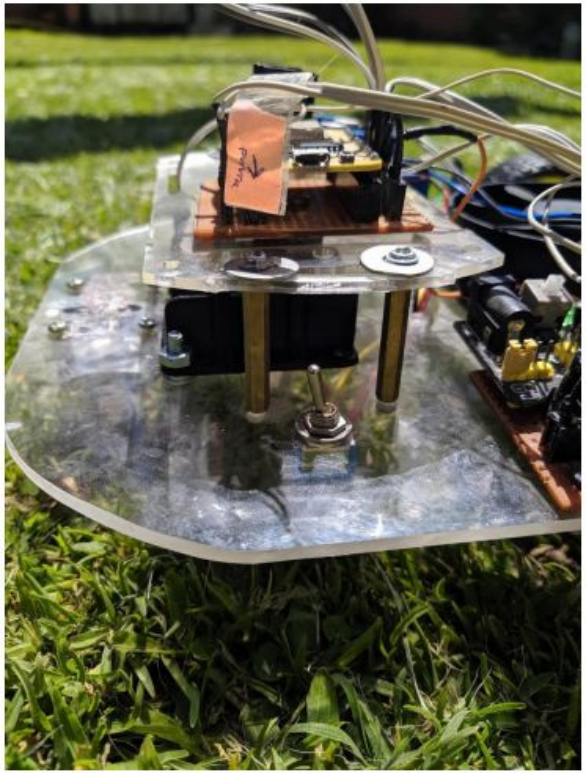
Muestra Final P.C. 2 - Robot aspiradora

Integrantes del grupo de desarrollo:

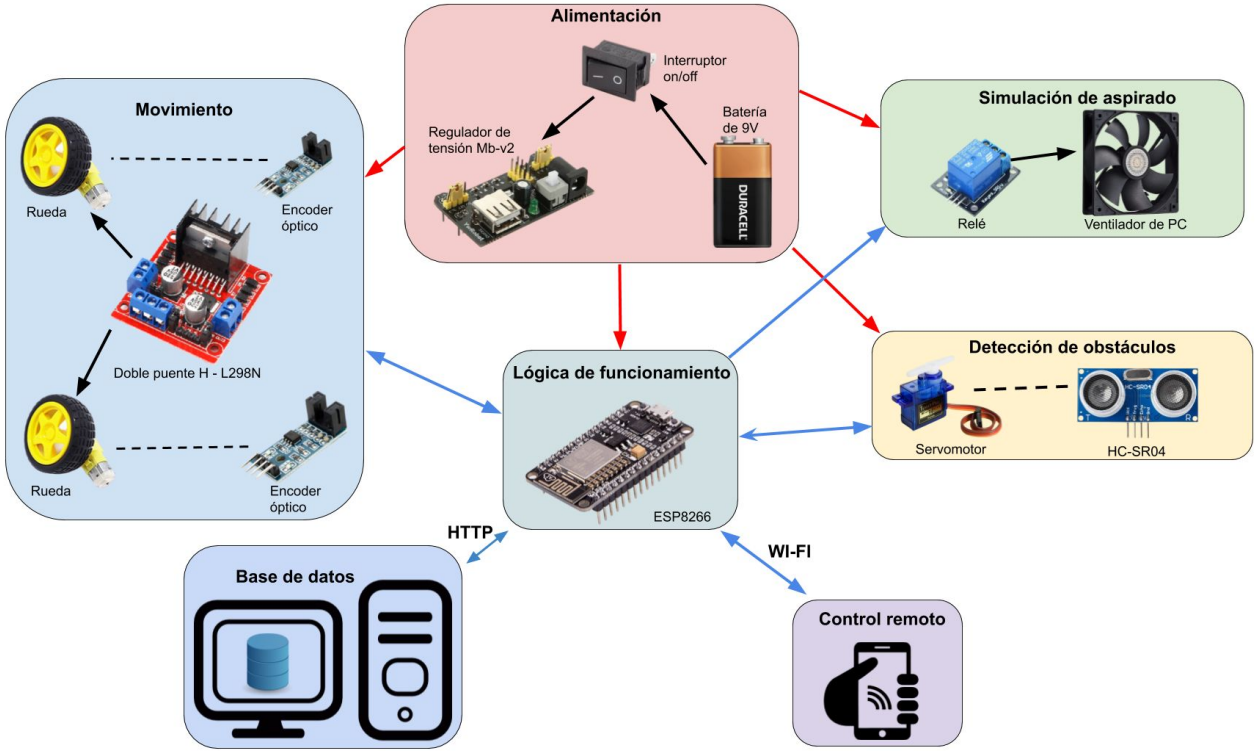
- Ezequiel Humar - 1297/4
- Juan Manuel Gómez - 1356/7

Robot aspiradora





Partes funcionales del robot



Estado inicial

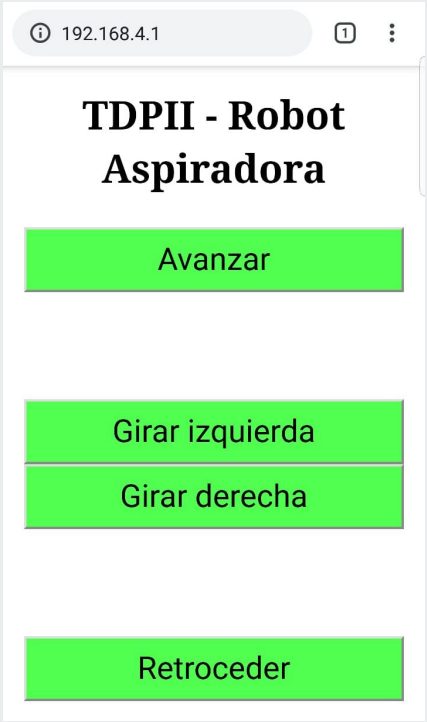
- Robot aspiradora ensamblado según lo mostrado.
- Modo de avance manual (adelante, atrás, giro izquierda y giro derecha).
- Lógica básica de movimiento en modo automático.
- Desviación durante movimiento en línea recta.

Estado actual

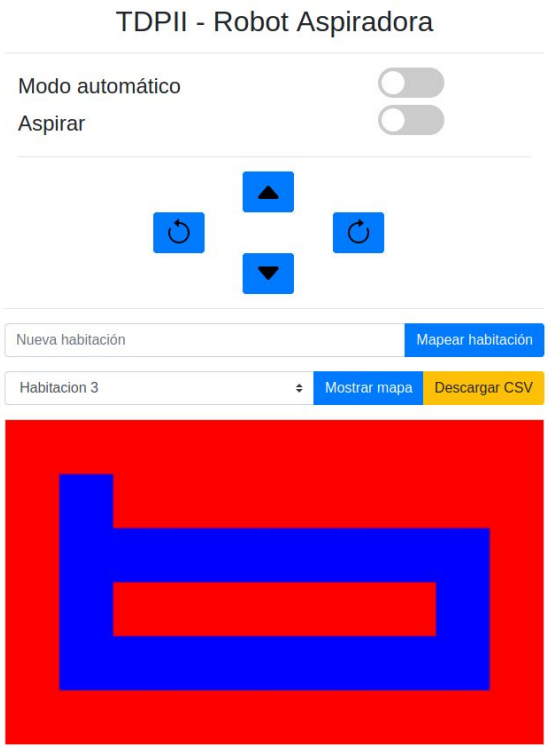
- Algoritmo de inicio mejorado.
- Diseño de un mapa automáticamente del escenario donde funciona el robot.
- Página web mejorada.
- Visualización del mapa en la página web.
- Base de datos con información del recorrido.

Interfaz Web

Estado inicial



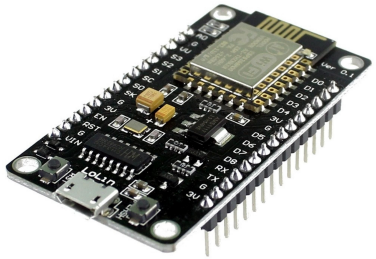
Estado actual

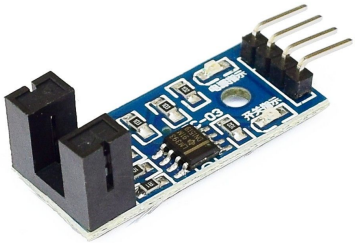


Simulación

Simulación realizada en C:

- NodeMCU
- Servomotor
- Ruedas con motores de CC
- Pthreads para sensores
  - Encoders
  - Sensor de ultrasonido





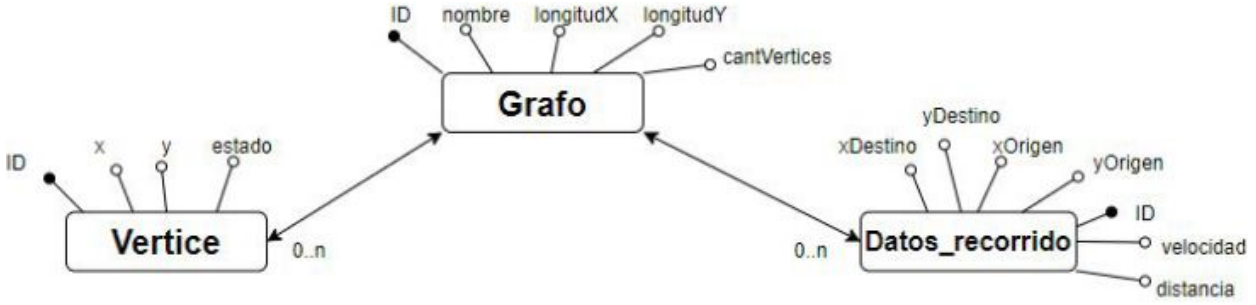
Servidor y base de datos

Servidor:

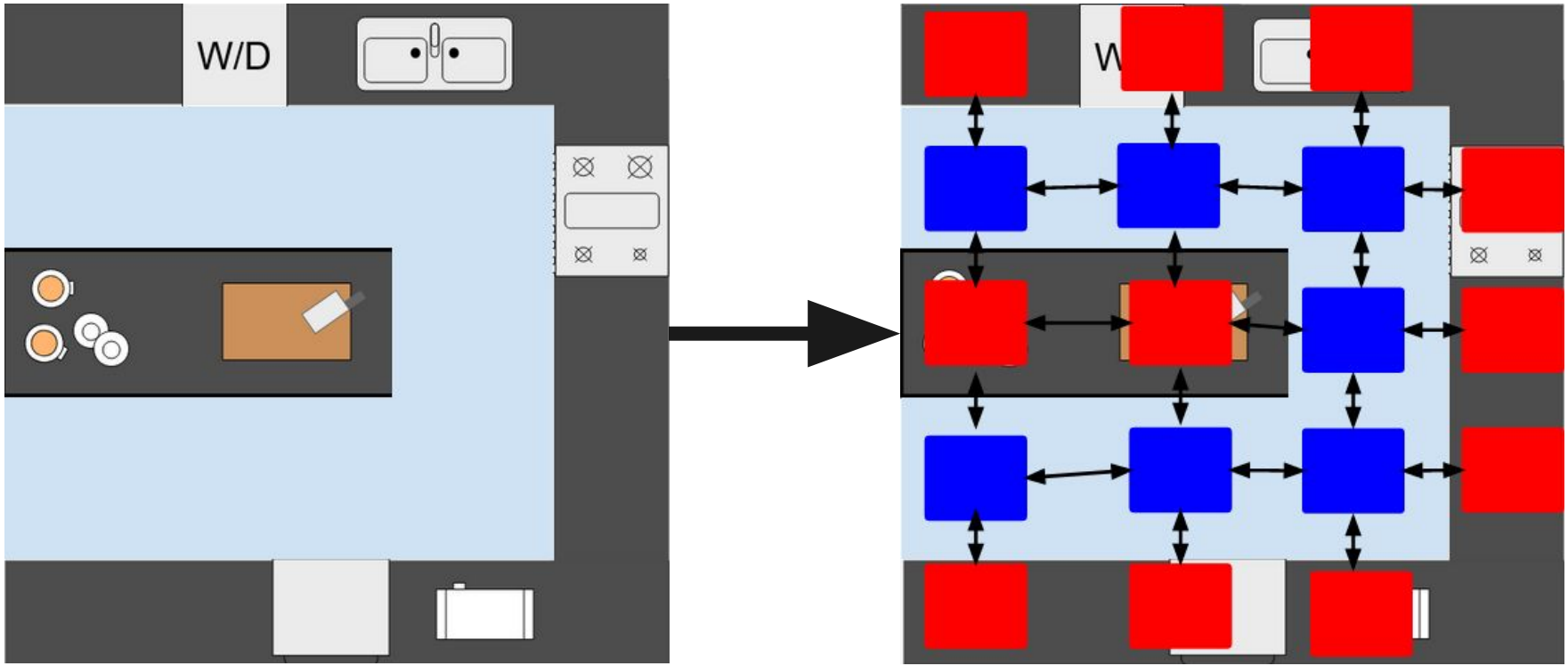
- Corre en una computadora personal
- Fue implementado con Node.js y el paquete Express.js
- API -> consulta, escritura y modificación de datos de recorridos y estado del robot

Base de datos:

- Implementada con MySQL
- Sequelize -> ORM utilizado para almacenar los objetos Javascript en la base de datos.
- Se guarda **mapa** e **información del recorrido**.



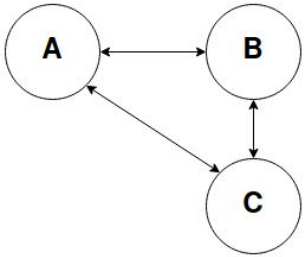
Grafo para almacenamiento de mapa

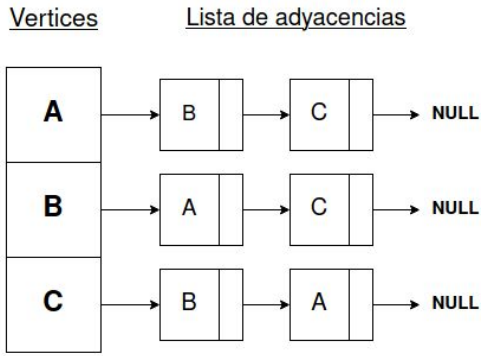


Grafo para almacenamiento del mapa

Implementación con listas de adyacencias:

- Colección de vértices
- Cada vértice tiene una lista con las referencias a sus adyacentes





Implementación propia:

- Cada vértice tiene siempre 4 adyacentes
- Vectores en lugar de listas enlazadas
- Cada vértice representa un cuadrado de 25 cm de lado en la habitación

Estrategia de simulación

Sensor ultrasonido:

- Hilo con Pthread
- Matriz de ceros (camino libre) y unos (obstáculos), para la habitación
  - Cada posición representa un cuadrado de 25 cm de lado
- Variable global ‘trigger’ dispara el cálculo de distancia
- Cuando termina de calcular libera un semáforo de sincronización

```
int habitacion[10][10] =
    {
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        {1, 0, 1, 1, 1, 1, 1, 0, 0, 1},
        {1, 0, 0, 0, 1, 1, 1, 0, 0, 1},
        {1, 0, 1, 0, 1, 1, 1, 0, 0, 1},
        {1, 0, 0, 0, 1, 1, 1, 0, 0, 1},
        {1, 1, 1, 1, 1, 0, 0, 0, 0, 1},
        {1, 0, 1, 0, 0, 0, 1, 1, 0, 1},
        {1, 1, 1, 1, 0, 0, 1, 1, 0, 1},
        {1, 1, 1, 1, 0, 0, 0, 0, 0, 1},
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}};
```

Encoder óptico

- Hilo con Pthread
- Se alterna una variable entre 0 y 1 representando la salida del encoder
- La alternancia se realiza con un periodo de 8,9 ms -> valor aprox. del encoder real a 2,05 km/h

Ruedas y servomotor

- Variables globales e impresiones en pantalla simulando su activación

Creación del grafo

Para la creación y llenado del grafo se implementó un algoritmo similar a una búsqueda en profundidad (DFS).

- Máquina de estados: Para controlar los movimientos del robot

- Algoritmo recursivo: Para asegurarse que se recorran todos los caminos posibles
  - Se avanza hasta encontrar un obstáculo
  - Cuando se encuentra obstáculo se mira a izquierda y derecha
  - Cuando no hay salida o se visitaron todos los caminos en un vértice, se retorna y se siguen visitando los posibles caminos del resto de vértices.

Comunicación simulación en C - servidor Node

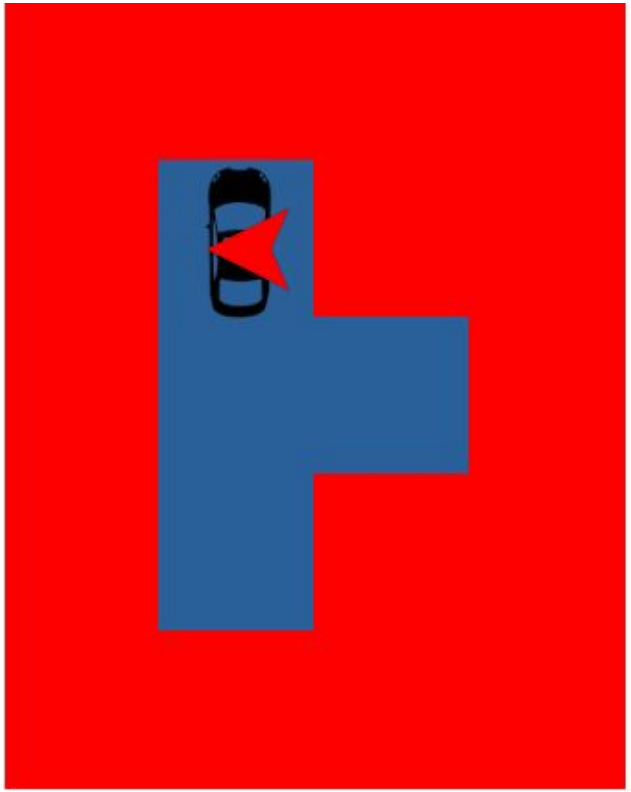
Lógica de movimiento del robot y servidor web se ejecutan en ambientes diferentes

- Transferencia de datos vía http -> librería CURL para C:
  - Curl permite enviar peticiones HTTP como un cliente, desde el código C.
  - Simulación envía POST Request con contenido en formato 'urlencoded' -> Vértices a almacenar en la BD

    Cuerpo del request: *campo1=valor1&campo2=Valor2&campo3=valor3*
  - Simulación envía GET Request al servidor para consultar el estado de la aspiradora robot (automático, manual, aspirado activado o desactivado, etc.) y los datos de la habitación a mapear.

- Servidor NodeJS
  - API para recibir las peticiones desde la simulación.
  - Se guarda el estado del robot.
  - Se convierten los datos recibidos desde la simulación a JSON y ser almacenan en la base de datos.
  - Las respuestas a la simulación se convierten a Strings antes de enviarse -> Problemas para interpretar JSON en C.

Ejemplo de simulación



¡Gracias por su atención!

¿Consultas?

Contacto:

- Juan Gómez: juanman.g97@gmail.com
- Ezequiel Humar: humarezequiel@gmail.com