PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA
INGENIERÍA CIVIL INFORMÁTICA

Proyecto Final

Docente responsable:

Sandra Cano

Integrantes:
Nidia Bugueño
Javiera Cabrera
Luciano Cubillos
Sebastián Jeria
Vicente Montiel

ICI 4150-1 ROBÓTICA Y SISTEMAS AUTÓNOMOS (1S-2025)

26 de junio 2025

Índice

Lista de figuras

1. Introducción

Este proyecto consiste en el desarrollo e implementación de un sistema de navegación autónoma para un robot móvil dentro de un entorno simulado en Webots. El objetivo principal es permitir que el robot se desplace desde un punto de origen hasta una meta determinada, evitando obstáculos y adaptándose al entorno en tiempo real. Para lograrlo, se integraron diversos sensores como LIDAR, GPS y sensores de distancia, además de motores controlados mediante velocidades diferenciadas para generar movimiento.

El sistema se basa en una arquitectura modular que incluye algoritmos para la detección de obstáculos, mapeo del entorno en una grilla lógica, y planificación de rutas mediante A*. Esta planificación permite al robot calcular caminos óptimos hacia su objetivo, mientras que el sistema de evasión reactiva garantiza la seguridad ante obstáculos no esperados. El entorno utilizado es una arena rectangular con dimensiones conocidas, dividida en celdas, lo que facilita el procesamiento espacial del movimiento del robot.

En las siguientes secciones se detallan las características del robot, el diseño del entorno simulado, la arquitectura del software implementado, los algoritmos utilizados, los resultados obtenidos durante las pruebas y posibles mejoras para futuras versiones del sistema.

2. Diseño del proyecto

En esta sección se detalla el diseño del proyecto en cuanto al robot móvil utilizado, sus características, además de la descripción del entorno simulado en Webots, la arquitectura del software, algoritmos que se utilizaron y diagramas de flujo y pseudocódigo del funcionamiento lógico del robot seleccionado.

2.1. Descripción del robot móvil y sus características.

El robot que fue utilizado para las pruebas en webots es el robot “4 wheels”, el robot consta de un cuerpo y 4 ruedas, además de dos sensores de distancia. Tanto las ruedas como el cuerpo del robot constan de la misma orientación. Por otro lado, en la figura 1 podemos observar los vectores del robot, donde el de color rojo permite mover al robot hacia adelante o atrás, el de color verde para la izquierda y derecha, finalmente el de color azul nos permite mover el robot hacia arriba o abajo.

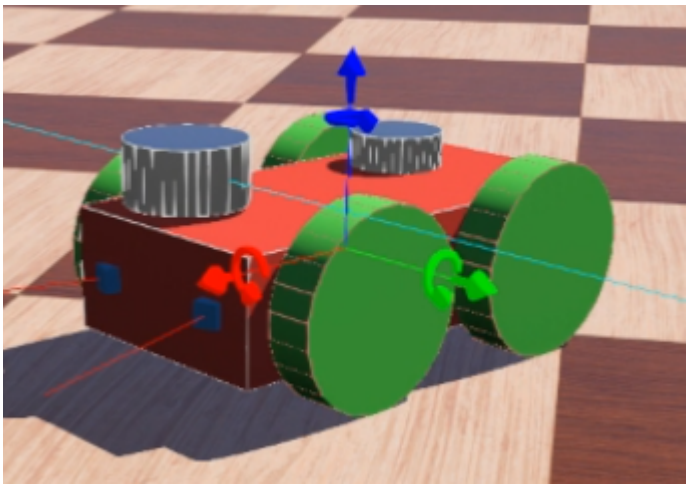

Figura 1. Robot móvil

Para explicar el funcionamiento del robot “4 Wheels”, el diagrama descriptivo del robot de la figura 2 muestra cómo está organizada su estructura interna a nivel de simulación en Webots. El robot tiene un cuerpo central con forma de caja, al que se conectan las ruedas mediante articulaciones que les permite girar. Cada rueda tiene su propio motor y parámetros físicos, lo que permite controlarlas individualmente. Todo esto está organizado en una jerarquía donde cada elemento (el cuerpo, las ruedas, los motores), está definido con precisión, para que el comportamiento del robot sea realista en la simulación.

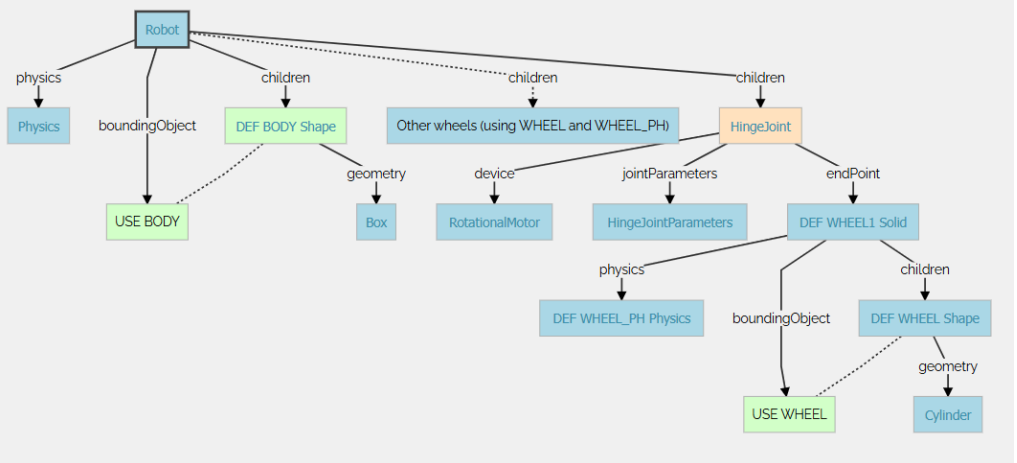

Figura 2. Diagrama descriptivo del robot

2.2. Explicación del entorno simulado en Webots

El entorno utilizado para la simulación en Webots corresponde a una arena rectangular llamada “RectangleArena”, la cual constituye el espacio de desplazamiento del robot móvil. Esta arena tiene una superficie de 4 metros por 4 metros, lo que permite contar con un área suficientemente amplia para la navegación. Además, el suelo está dividido en baldosas de 0.5 metros por lado, facilitando la representación del espacio mediante una grilla regular.

En términos de posición dentro del mundo simulado, la arena está ubicada prácticamente en el centro del sistema de coordenadas de Webots. Su posición está especificada por un vector de traslación muy cercano a cero ($x \approx 0$, $y = 0$, $z \approx 0$), lo que indica que se encuentra centrada. No presenta rotación significativa, ya que está alineada con el eje Y, lo que confirma que el suelo está orientado de forma horizontal sin giros.

Para trabajar con esta arena desde el código, se define una grilla lógica de 8 por 8 celdas, donde cada celda representa una superficie de 0.5 x 0.5 metros, coincidiendo con el tamaño de las baldosas visuales del entorno. Esta grilla es utilizada para planificar rutas, detectar obstáculos y determinar la posición del robot en términos discretos. La planificación se realiza mediante una matriz (grid[8][8]) en la que se identifican las celdas libres y ocupadas.

Debido a que las coordenadas entregadas por el GPS del robot están en valores reales (en metros), se definió un origen personalizado para mapear esas coordenadas a la grilla. En este caso, el punto de origen tiene valores x = -1.88569 y z = 0.0695482, lo que permite calcular en qué celda de la matriz se encuentra el robot o un obstáculo determinado.

2.3. Arquitectura del software

Para el robot de 4 ruedas usado en webots, utilizamos dos tipos de sensores: primero tenemos el GPS, el cual nos va ayudar en conocer la posición global del robot en las

coordenadas del entorno de trabajo, esto será usado para la planificación de rutas. Por otro lado, tenemos LIDAR, el cual nos va a permitir detectar los obstáculos que haya cerca del robot en el entorno, esto lo hace a distintas distancias y ángulos alrededor del robot, construyendo un mapa del entorno y así evitar colisiones, también tenemos dos sensores de distancia (izquierdo y derecho), los cuales detectan objetos cercanos frente al robot, esto representa sensores infrarrojo o ultrasonido. En cuanto a los actuadores que se utilizaron, se hizo uso de cuatro motores que van conectados a las ruedas, los cuales se controlan mediante velocidades, esto para generar movimiento y dirección al robot. Pasando a los módulos de control, el sistema cuenta con un módulo de planificación de rutas que implementa el algoritmo A*, el cual se encarga de guiar al robot hacia cada punto de la ruta calculada ajustando las velocidades, también consta de un módulo que se encarga de la evitación de obstáculos el cual si detecta un obstáculo cercano, se asegura de buscar una navegación segura y adaptativa en el entorno simulado.

2.4. Algoritmos a utilizar

En la implementación del sistema de navegación autónoma para el robot móvil, se utilizaron los siguientes algoritmos principales:

1. Mapeo del entorno (Grilla): El entorno se representa mediante una grilla, donde cada celda indica si está libre (0) u ocupada por un obstáculo (1). Esta grilla se actualiza dinámicamente en cada iteración utilizando:
    a. Sensor LIDAR: Se obtienen las distancias a los obstáculos y se proyectan al plano X-Z, generando coordenadas relativas de obstáculos.
    b. Conversión a celdas de grilla: Las coordenadas reales se transforman a índices de la grilla mediante la fórmula:
    $celda_x = \left[\frac{x-ORIGINX}{CELLSIZE}\right]$, $celda_y = \left[\frac{z-ORIGINZ}{CELLSIZE}\right]$
2. Planificación de rutas (Algoritmo A*): Se utiliza el algoritmo A* para planificar la ruta más corta desde la posición actual del robot hasta la celda objetivo, evitando las celdas marcadas como ocupadas.
    a. Se utiliza la función heurística distancia Manhattan entre dos puntos de la grilla.
    b. Se mantiene una lista abierta (nodos por explorar) y una lista cerrada (nodos ya evaluados).
3. Evitación de obstáculos reactiva: Se implementó un mecanismo reactivo local de evitación para situaciones de emergencia.
    a. Si el sensor LIDAR detecta un obstáculo a menos de 0.3 metros o los sensores de distancia laterales detectan un objeto muy cercano (valor < 950), se activa un comportamiento evasivo.
    b. Este comportamiento consiste en girar sobre su eje hasta que se despeje el camino.

2.5. Diagramas de flujo y pseudocódigo de la solución

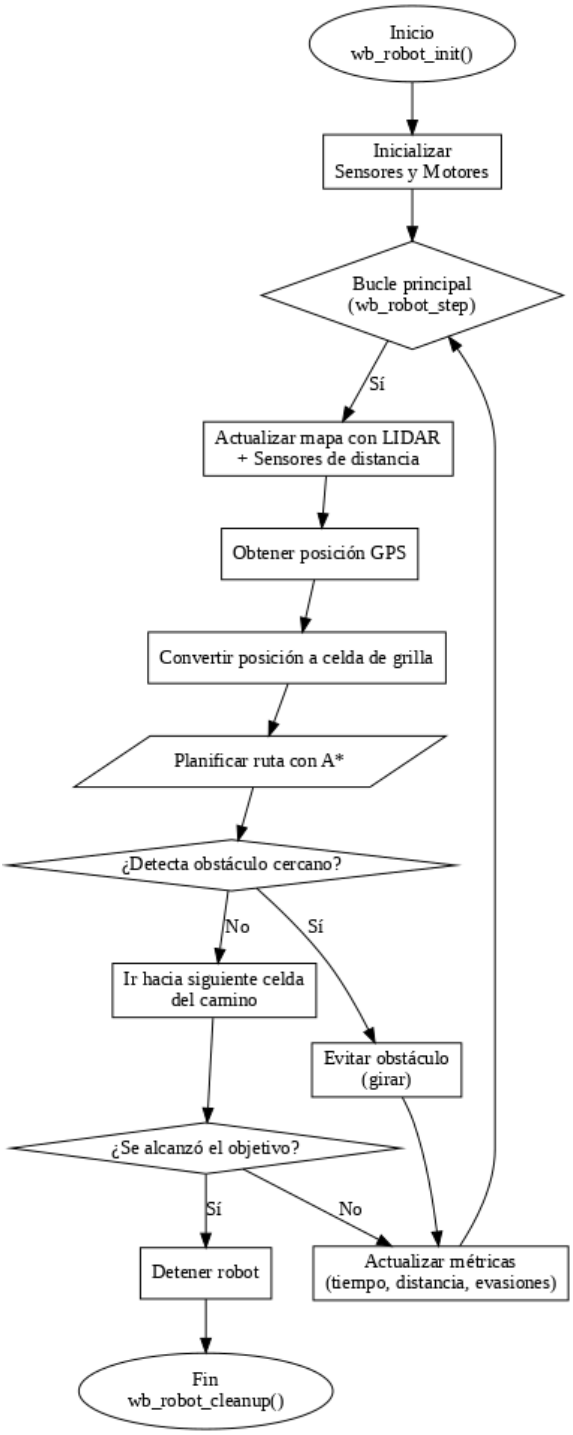

Figura 3. Diagrama de flujo

El diagrama de flujo visualiza paso a paso el funcionamiento del robot. Comienza con la inicialización (wb_robot_init) y la configuración de sensores y motores. Luego, entra en el bucle principal, donde primero se actualiza el mapa con los datos del LIDAR y sensores de distancia. A continuación, obtiene la posición GPS y la

convierte en coordenadas de grilla para planificar la ruta hacia la meta usando el algoritmo A*. Luego, evalúa si hay obstáculos cercanos: si los hay, ejecuta una maniobra de evasión; si no, avanza hacia la siguiente celda del camino. Después, verifica si se ha alcanzado el objetivo. Si es así, detiene el robot; si no, actualiza las métricas y continúa el ciclo. El diagrama resalta las decisiones clave (como detección de obstáculos o llegada a la meta) y muestra claramente el flujo de control iterativo que guía al robot durante toda su operación.

```
INICIAR robot

INICIALIZAR motores y sensores (ultrasónicos, LIDAR, GPS)

CREAR una grilla vacía de obstáculos (GRID_SIZE x GRID_SIZE)
CREAR una lista de puntos para la ruta (path)
INICIALIZAR variables: posición inicial, bandera de inicio, bandera de meta alcanzada
INICIALIZAR registro de giros

MIENTRAS la simulación está activa:

  OBTENER posición del robot con GPS
  CALCULAR si está en la zona objetivo (coordenadas específicas)

  SI es la primera iteración:
    GUARDAR posición como origen
    Marcar como inicializado

  SI NO ha llegado a la meta:
    IMPRIMIR posición del robot

  SI llegó a la meta Y no se había detectado antes:
    IMPRIMIR "¡Llegó a la meta!" y la secuencia de giros
    DETENER todos los motores
    SALIR del bucle

  LEER sensores ultrasónicos
  SI alguno detecta distancia menor a umbral:
    MARCAR detección cercana

  LEER datos del LIDAR
  PARA cada punto del LIDAR:
    CALCULAR ángulo y distancia
    SI hay obstáculo cercano:
      CALCULAR posición X,Y del obstáculo
      CONVERTIR a coordenadas de grilla
      MARCAR celda como obstáculo en la grilla
    SI el obstáculo está muy cerca:
      MARCAR detección cercana por LIDAR

  CONVERTIR posición actual del robot a coordenadas de grilla
  DEFINIR posición meta como una celda cercana a la esquina inferior derecha
```

```
  LLAMAR A* planificador (plan_path) con el mapa, inicio y meta
  GUARDAR ruta calculada

  DEFINIR velocidades por defecto (avance)
  SI se detectó obstáculo cercano:
    AJUSTAR velocidades para girar (izquierda)

  SI el robot está girando Y es un nuevo giro:
    REGISTRAR dirección del giro (izquierda o derecha)

  ACTUALIZAR velocidades anteriores
  ENVIAR velocidades a los 4 motores

TERMINAR simulación y liberar recursos
```

El pseudocódigo describe el comportamiento de un robot móvil en un entorno simulado, controlado con Webots. Al iniciar, el robot inicializa sus motores y sensores (LIDAR, GPS, y sensores ultrasónicos). Luego, entra en un ciclo principal donde, en cada iteración, obtiene su posición mediante GPS y detecta obstáculos cercanos usando el LIDAR y los sensores ultrasónicos. La información del entorno se actualiza en una grilla que representa el mapa del espacio. Si el robot aún no ha llegado a la zona objetivo, calcula una ruta óptima mediante el algoritmo A* desde su posición actual hasta la meta. Dependiendo de si hay obstáculos cercanos, decide si avanzar o girar, ajustando las velocidades de sus motores. Además, registra los giros que realiza. El ciclo se repite hasta que el robot alcanza el objetivo, momento en el cual se detiene y muestra la secuencia de giros realizada.

3. Implementación del Robot en Webots

En esta sección se detalla la implementación en el proyecto del robot móvil descrito en la plataforma Webots, así como su código funcional, configuración e integración de los sensores utilizados, algoritmos y pruebas en escenarios simulados.

3.1. Código funcional del robot en Webots

A continuación, se hace referencia al código funcional desarrollado para el robot en el simulador Webots, el cual integra sensores (GPS, LIDAR, sensores de distancia), motores y algoritmos de navegación, mapeo y evitación de obstáculos. Este código, implementado en lenguaje C, permite al robot planificar rutas mediante el algoritmo A* y desplazarse de forma autónoma en un entorno simulado, respondiendo en tiempo real a la presencia de obstáculos. El código completo se incluye en el Anexo A de este informe.

3.2. Configuración e integración de sensores

El sistema desarrollado integra 3 tipos de sensores los cuáles son GPS, sensores de distancia y LIDAR. Estos sensores cumplen un rol específico dentro de la arquitectura del robot y su configuración se realiza al inicio del programa dentro de la función principal, **main()**, donde se referencian a los dispositivos mediante el comando **wb_robot_get_device**.

El sensor GPS permite obtener la posición global del robot en el entorno simulado. Este se habilita mediante la instrucción **wb_gps_enable** y entrega coordenadas para los ejes X e Y en forma de vector con mediciones en metros respecto al sistema de coordenadas del sistema en Webots. El sensor GPS es fundamental para entender la posición del robot dentro de una grilla para la planificación de rutas y mapeos de entornos.

Se incorporan 2 sensores de distancia ultrasónicos ubicados en la parte frontal del robot por su periferia izquierda y derecha. Estos sensores se utilizan para la detección de obstáculos en el área inmediata frente al robot en una corta distancia. Estos sensores se activan mediante la instrucción **wb_distance_sensor_enable** y si detectan un objeto a una distancia específica, el robot actuará con una maniobra para evadir al obstáculo reactivamente.

El sensor LIDAR, es clave para el cumplimiento de los objetivos de este proyecto mediante la percepción del entorno desde el punto de vista del robot. Este funciona emitiendo pulsos láser y calculando la distancia a partir del tiempo que tarda en impactar en un objeto. El sensor LIDAR se activa con **wb_lidar_enable** y **wb_lidar_enable_point_cloud**, y su configuración incluye el campo de visión percibida o FOV, que determina el ángulo del escaneo hacia la parte delantera del robot, la resolución horizontal, la cuál apunta al número de puntos que genera el sensor en la creación de la nube de puntos o point cloud donde se puede obtener

información sobre cada punto en el espacio, y la frecuencia de actualización en la medición del sensor.

Gracias a la correcta configuración e implementación de estos sensores, el sistema es capaz de realizar una navegación autónoma dentro del entorno simulado en tiempo real, detectando obstáculos e impulsando acciones para esquivarlos.

3.3. Algoritmos implementados para navegación, detección de obstáculos y mapeo

Para lograr que el robot navegue de forma autónoma en un entorno con obstáculos, se implementó una combinación de algoritmos que actúan en conjunto: navegación planificada, detección de obstáculos en tiempo real y mapeo dinámico del entorno.

En cuanto a la navegación, se utilizó el algoritmo A*, ampliamente conocido por su eficiencia en la búsqueda de caminos óptimos en espacios discretos. El entorno se representa como una grilla de tamaño fijo, donde cada celda puede estar libre u ocupada. A partir de la posición actual del robot y la celda objetivo, A* calcula la ruta más corta evitando obstáculos, utilizando una heurística basada en la distancia Manhattan. El camino resultante es una lista de puntos intermedios que el robot sigue de manera secuencial.

Para la detección de obstáculos, se integraron dos tipos de sensores: LIDAR y sensores de distancia laterales. LIDAR proporciona un escaneo angular del entorno y permite detectar obstáculos en 360°, mientras que los sensores de distancia permiten una detección rápida de objetos cercanos a los costados. Si alguno de los sensores detecta un objeto a una distancia crítica, se activa un comportamiento reactivo simple: el robot detiene su avance y comienza a girar sobre su eje, evitando colisiones mientras espera que el camino se despeje o se replantee la ruta.

Finalmente, se implementó un algoritmo de mapeo local, que permite construir dinámicamente una representación del entorno en forma de grilla. Los datos del LIDAR se proyectan desde la posición actual del robot hacia coordenadas reales (X, Z) y se convierten en celdas de la grilla. Si un obstáculo es detectado dentro del rango definido, la celda correspondiente se marca como ocupada. Este mapa se actualiza en cada ciclo de control, permitiendo que el algoritmo A* planifique rutas con información actualizada y adaptativa según los cambios del entorno.

3.4. Pruebas en escenarios simulados

Las pruebas que se realizaron en el entorno donde se simuló el movimiento del robot consistía en la incorporación de objetos para que el robot pudiera esquivarlo y seguir moviéndose hacia la meta, en nuestro caso hicimos uso de paredes para obstaculizar el camino del robot, además de incorporar algunos animales como un perro y un gato. En un principio no logramos hacer que el robot llegara a la meta y solo se movía libremente por el escenario esquivando los objetos sin detenerse, luego logramos hacer que el robot esquivara los obstáculos hasta llegar a la meta y detenerse. Se estuvieron modificando parámetros del código para lograr hacer que el robot llegara a

la meta sin dificultades. Finalmente, se logró hacer que el robot se moviera libremente por el escenario hasta que llevara a la meta y se detuviera mientras esquivaba los obstáculos presentados en el escenario.

4. Resultados

En esta sección se presentan los resultados obtenidos tras la implementación del controlador en el simulador Webots. Se describe el comportamiento del robot al explorar un entorno tipo laberinto, planificar rutas hacia una meta, registrar sus giros y detenerse correctamente al llegar a una zona objetivo. Además, se analizan los algoritmos utilizados, se reflexiona sobre posibles mejoras al sistema, y se destacan lecciones y proyecciones prácticas del proyecto.

4.1. Resultados obtenidos

- El robot fue evaluado en una arena plana de 4 × 4 metros, libre de estructuras predefinidas como laberintos o muros permanentes. Su comportamiento se desarrolló en un entorno dinámico, donde debía identificar y evitar obstáculos en tiempo real, sin un conocimiento previo del espacio que lo rodeaba. Durante su funcionamiento, el robot construyó un mapa binario del entorno, es decir, una grilla de celdas marcadas como libres u ocupadas. Esta representación fue generada a partir de las lecturas obtenidas por el sensor LIDAR, combinadas con la posición geográfica del robot proporcionada por el GPS. La información fue convertida a coordenadas de grilla, permitiendo que el robot interpretara con precisión la presencia de obstáculos en su radio de acción, actualizando constantemente su percepción del espacio.
- Con la información recopilada, el robot ejecutó el algoritmo A* para planificar rutas óptimas desde su posición actual hasta una esquina definida como destino. Este punto objetivo se encontraba dentro del cuadrante inferior izquierdo de la arena, aproximadamente en las coordenadas GPS X = –1.6 y Y = –1.6. El algoritmo evaluaba las celdas disponibles y evitaba aquellas que habían sido marcadas como ocupadas, asegurando una navegación eficiente y adaptable ante nuevos obstáculos. A lo largo del recorrido, el sistema fue capaz de identificar los giros realizados por el robot, registrando cada cambio de dirección en consola. Estos eventos eran detectados mediante la comparación de velocidades entre los motores laterales, permitiendo distinguir movimientos hacia la izquierda o la derecha. Esta funcionalidad evidenció una correcta interpretación del comportamiento de desplazamiento en función de los comandos de velocidad. Finalmente, la llegada a la meta fue determinada mediante una zona de tolerancia definida por coordenadas GPS, considerando rangos entre X = –1.66 y –1.58, y Y = –1.66 y –1.58. Esta área permitió compensar pequeñas variaciones en la ubicación del robot, producto del movimiento continuo y las posibles imprecisiones del sensor. Al ingresar en esta región, el sistema detuvo el movimiento del robot y desplegó en consola la secuencia de giros realizada, indicando que la tarea había sido completada exitosamente.

4.2. Análisis de los algoritmos utilizados

El robot y el entorno modelado siguió los siguientes algoritmos para el mapeo del entorno y planificación de rutas:

- (A estrella): El entorno se modeló como una grilla de celdas ortogonales, donde cada celda representa un nodo del grafo. El costo real g corresponde al número de pasos desde el origen, y la heurística h es la distancia de Manhattan hasta la meta. El valor total f = g + h permite guiar la búsqueda de manera eficiente, combinando la precisión del recorrido con la rapidez en la toma de decisiones. Gracias a este enfoque, el robot logra planificar rutas óptimas incluso en presencia de obstáculos detectados dinámicamente.
- Mapa de ocupación con LIDAR y GPS: Cada medición del sensor LIDAR es transformada en coordenadas absolutas utilizando funciones trigonométricas. La posición global del robot, obtenida vía GPS, se utiliza como punto de referencia para ubicar los obstáculos dentro de la grilla. Este sistema asegura que la planificación de rutas se base en un mapa coherente con el entorno percibido.
- Control de motores: El movimiento del robot se realiza a velocidad constante mientras no se detectan obstáculos cercanos. En caso de que un objeto esté a menos de 0.3 metros, se activa una maniobra evasiva invirtiendo la dirección de las ruedas. Por último, una condición de parada supervisa continuamente si el robot se encuentra dentro de la zona objetivo, deteniendo completamente su avance.

Resultados de salida del robot por consola:

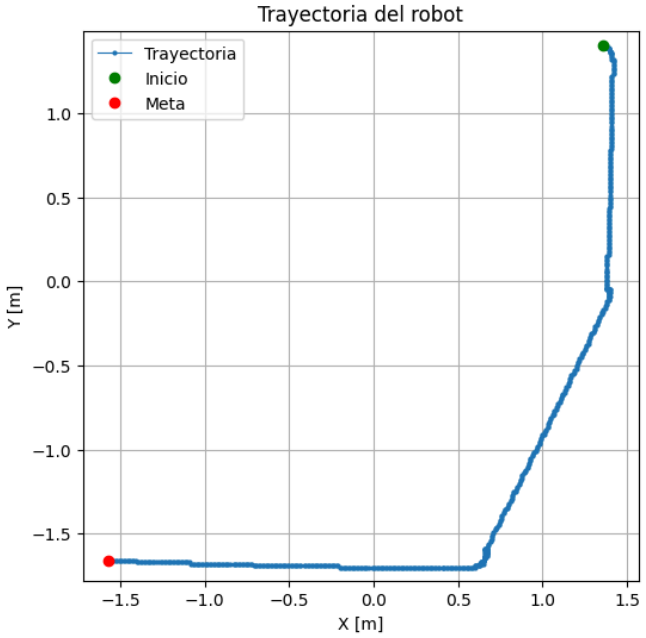

Figura 4. Resultados de la trayectoria del robot

La gráfica muestra la trayectoria que siguió un robot móvil en un entorno simulado, desplazándose desde el punto de inicio (marcado en verde, en la parte superior derecha) hasta la meta (marcada en rojo, en la parte inferior izquierda). A lo largo del trayecto, representado por una línea azul, se observa que el robot comenzó moviéndose hacia abajo en el eje Y, luego giró hacia la izquierda y continuó avanzando hasta alcanzar su destino. Esta trayectoria sugiere que el robot fue capaz de navegar correctamente en el entorno, probablemente evitando obstáculos o siguiendo una ruta planificada, lo que indica que cumplió exitosamente su objetivo de llegar a la meta desde el punto de partida.

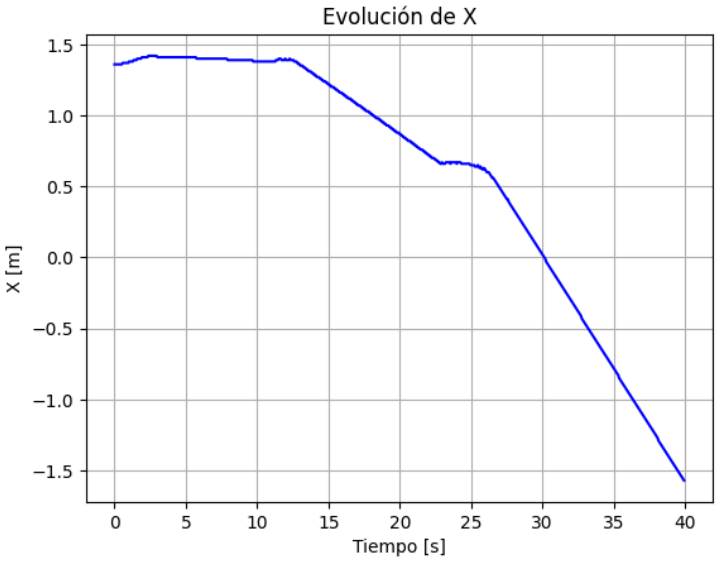

Figura 5. Resultados del eje X

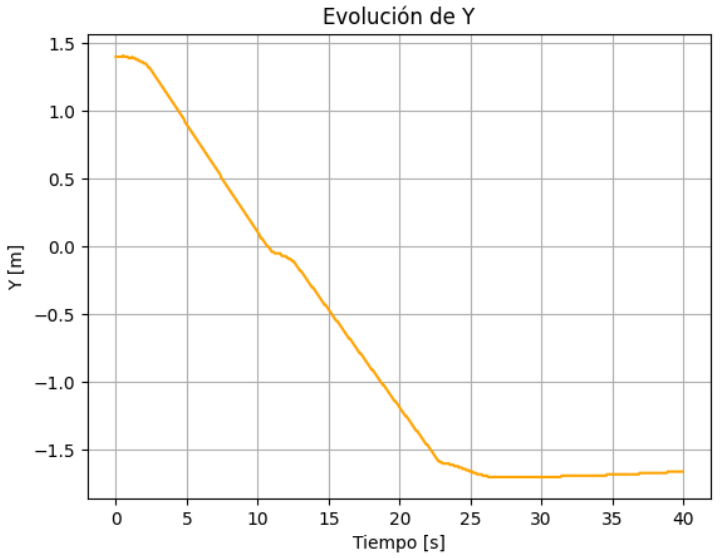

Figura 6. Resultados del eje Y

En los gráficos se observa la evolución temporal de las coordenadas X y Y del robot durante su trayectoria. En la gráfica de X, al inicio el valor se mantiene cerca de 1.4 m, lo que indica que el robot parte desde una posición positiva en el eje X. Luego, se aprecia una disminución continua en esta coordenada, lo que significa que el robot se desplaza hacia la izquierda (hacia valores negativos de X) hasta alcanzar aproximadamente -1.5 m. En cuanto a la evolución de Y, el gráfico muestra una disminución constante desde alrededor de 1.4 m hasta llegar a aproximadamente -1.7 m, reflejando un movimiento descendente en el plano. Esta información, combinada con la trayectoria observada en el primer gráfico, confirma que el robot se desplazó desde el extremo superior derecho hasta el inferior izquierdo, siguiendo una trayectoria continua sin retrocesos significativos.

4.3. Reflexión sobre mejoras y optimización del sistema

Algunas mejoras que se proponen luego de la implementación del robot:

- Rutas alternativas y replanteo dinámico: Actualmente el algoritmo A* se ejecuta en cada paso de simulación, lo que puede resultar ineficiente. Una mejora sería re planificar únicamente cuando se detecten cambios significativos en el entorno. Además, extender el modelo de movimiento a una grilla 8-conectada permitiría generar rutas más cortas y realistas, al considerar desplazamientos en diagonal.
- Sensorización extendida: Integrar sensores adicionales, como cámaras RGB o detectores de color, permitiría que el robot reconozca zonas diferenciadas en el entorno, como áreas de carga o zonas de peligro. También, la incorporación de una unidad de medición inercial (IMU) o una brújula digital proporcionaría datos de orientación, mejorando la precisión en la transformación de las lecturas del LIDAR.
- Control de trayectoria avanzado: En lugar de mantener una velocidad fija, podría utilizarse un controlador PID para ajustar la orientación del robot en función del siguiente punto de la ruta, logrando movimientos más suaves y precisos, especialmente en trayectorias curvas o con obstáculos cercanos.

4.4. Lecciones aprendidas y posibles extensiones del proyecto

El proyecto demostró que una integración básica entre sensores LIDAR y GPS es suficiente para implementar navegación autónoma en entornos estáticos. El uso de A* en grillas discretas permitió una planificación eficiente y fácil de interpretar. Además, la arquitectura modular del entorno de simulación Webots facilitó el desarrollo incremental de los distintos componentes del sistema.

Algunas posibles extensiones prácticas para este tipo de enfoque puede aplicarse en escenarios reales como:

- Logística y gestión de inventarios, con robots que recorren almacenes para realizar conteos o entregar paquetes.
- Sistemas de reparto en interiores, capaces de adaptarse dinámicamente a obstáculos móviles como personas u otros robots.
- Entornos educativos, donde los conceptos de navegación, percepción y control pueden ser enseñados mediante simulaciones interactivas que integren visión por computadora, SLAM o cooperación multi-robot.

5. Conclusión

Tras haber repasado el diseño del proyecto en Webots, la implementación del robot móvil y los resultados obtenidos, se puede concluir que en este proyecto se desarrolló e implementó el sistema de navegación autónoma de manera exitosa, cumpliendo el objetivo principal al permitir que el robot móvil se desplace desde un punto de origen hasta una meta determinada. La integración de sensores de distancia, LIDAR, GPS junto a la incorporación modular de distintos algoritmos para la detección de obstáculos, mapeo del entorno y planificación de rutas contribuyeron a lograr el objetivo propuesto.

La realización de este proyecto permitió poner en práctica y experimentar algoritmos complejos, la configuración de sensores, el control de motores y conceptos fundamentales de la robótica en un entorno simulado sin las limitaciones económicas y logísticas que implica el trabajar con hardware físico, lo que brindó la posibilidad de explorar funcionalidades y mejorar el diseño del entorno sin poner en riesgo nuestra integridad.

Esta experiencia dejó lecciones de navegación autónoma en entornos estáticos, lo que nos demostró que existen diversas implementaciones de este sistema al aplicarlo en entornos personales o profesionales como la logística y gestión de inventarios, sistemas de reparto en interiores y la integración en sistemas educativos.

En definitiva, este proyecto fortaleció nuestros conocimientos en robótica y evidenció que esta es una área poderosa para la creación de sistemas inteligentes e innovadores, que se dirijan a la aplicación en entornos profesionales inteligentes.

Anexos

A. Código funcional del controlador

```
#include <webots/distance_sensor.h>
#include <webots/lidar.h>
#include <webots/gps.h>
#include <webots/motor.h>
#include <webots/robot.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define TIME_STEP 64
#define GRID_SIZE 8
#define CELL_SIZE 0.5
#define MAX_PATH_LEN 100
#define MAX_OPEN_NODES 1000
#define SPEED 4.0
#define MAX_TURNS 100

typedef struct {
  int x, y;
} Point;

typedef struct {
  int x, y;
  int g, h, f;
  int parent_index;
} Node;

int heuristic(int x1, int y1, int x2, int y2) {
  return abs(x1 - x2) + abs(y1 - y2);
}

int plan_path(int grid[GRID_SIZE][GRID_SIZE], Point start, Point
goal, Point path[], int max_path_len) {
  Node open_list[MAX_OPEN_NODES];
  int open_count = 0;
  Node closed_list[GRID_SIZE * GRID_SIZE];
  int closed_count = 0;

  Node start_node = {start.x, start.y, 0, heuristic(start.x,
start.y, goal.x, goal.y), 0, -1};
  start_node.f = start_node.g + start_node.h;
  open_list[open_count++] = start_node;

  while (open_count > 0) {
    int best_index = 0;
    for (int i = 1; i < open_count; i++) {
      if (open_list[i].f < open_list[best_index].f)
        best_index = i;
```

```
    }

    Node current = open_list[best_index];
    for (int i = best_index; i < open_count - 1; i++)
      open_list[i] = open_list[i + 1];
    open_count--;
    closed_list[closed_count++] = current;

    if (current.x == goal.x && current.y == goal.y) {
      int length = 0;
      Node n = current;
      while (n.parent_index != -1 && length < max_path_len) {
        path[length++] = (Point){n.x, n.y};
        n = closed_list[n.parent_index];
      }
      path[length++] = (Point){start.x, start.y};
      for (int i = 0; i < length / 2; i++) {
        Point temp = path[i];
        path[i] = path[length - i - 1];
        path[length - i - 1] = temp;
      }
      return length;
    }

    const int dx[4] = {0, 1, 0, -1};
    const int dy[4] = {1, 0, -1, 0};
    for (int d = 0; d < 4; d++) {
      int nx = current.x + dx[d];
      int ny = current.y + dy[d];
      if (nx < 0 || ny < 0 || nx >= GRID_SIZE || ny >=
GRID_SIZE)
        continue;
      if (grid[nx][ny] == 1)
        continue;

      int in_closed = 0;
      for (int i = 0; i < closed_count; i++) {
        if (closed_list[i].x == nx && closed_list[i].y == ny) {
          in_closed = 1;
          break;
        }
      }
      if (in_closed) continue;

      int g = current.g + 1;
      int h = heuristic(nx, ny, goal.x, goal.y);
      int f = g + h;

      int in_open = -1;
      for (int i = 0; i < open_count; i++) {
        if (open_list[i].x == nx && open_list[i].y == ny) {
          in_open = i;
          break;
        }
      }

      if (in_open != -1) {
```

```
        if (f < open_list[in_open].f) {
          open_list[in_open].g = g;
          open_list[in_open].h = h;
          open_list[in_open].f = f;
          open_list[in_open].parent_index = closed_count - 1;
        }
      } else if (open_count < MAX_OPEN_NODES) {
        Node neighbor = {nx, ny, g, h, f, closed_count - 1};
        open_list[open_count++] = neighbor;
      }
    }
  }
  return 0;
}

int main() {
  wb_robot_init();
  int i;

  // Motores
  WbDeviceTag wheels[4];
  char wheels_names[4][8] = {"wheel1", "wheel2", "wheel3",
"wheel4"};
  for (i = 0; i < 4; i++) {
    wheels[i] = wb_robot_get_device(wheels_names[i]);
    wb_motor_set_position(wheels[i], INFINITY);
    wb_motor_set_velocity(wheels[i], 0.0);
  }

  // Sensores ultrasónicos
  WbDeviceTag ds[2];
  char ds_names[2][10] = {"ds_left", "ds_right"};
  for (i = 0; i < 2; i++) {
    ds[i] = wb_robot_get_device(ds_names[i]);
    wb_distance_sensor_enable(ds[i], TIME_STEP);
  }

  // LIDAR
  WbDeviceTag lidar = wb_robot_get_device("lidar");
  wb_lidar_enable(lidar, TIME_STEP);
  wb_lidar_enable_point_cloud(lidar);

  // GPS
  WbDeviceTag gps = wb_robot_get_device("gps");
  wb_gps_enable(gps, TIME_STEP);

  // Variables
  int grid[GRID_SIZE][GRID_SIZE] = {0};
  Point path[100];
  int path_length = 0;

  double origin_x = 0.0, origin_y = 0.0;
  bool initialized = false;
  bool reached_goal = false;

  // Registro de giros
  char turn_log[MAX_TURNS][20];
```

```
  int turn_count = 0;
  double last_left_speed = 0.0, last_right_speed = 0.0;

  while (wb_robot_step(TIME_STEP) != -1) {
    const double *pose = wb_gps_get_values(gps);
    double robot_x = pose[0];
    double robot_y = pose[1];  // plano X-Y

    // Definir zona meta
    bool in_goal_area = (robot_x >= -1.66 && robot_x <= -1.58 &&
                         robot_y >= -1.66 && robot_y <= -1.58);

    if (!initialized) {
      origin_x = robot_x;
      origin_y = robot_y;
      initialized = true;
    }

    // Mostrar posición solo si no está en la meta
    if (!in_goal_area)
      printf("Posición GPS - X: %.2f, Y: %.2f\n", robot_x,
robot_y);

    // Si llegó a la meta
    if (in_goal_area && !reached_goal) {
      printf("\n¡Llegó a la meta!\nSecuencia de giros:\n");
      for (i = 0; i < turn_count; i++)
        printf("%s\n", turn_log[i]);
      reached_goal = true;

      // Detener robot
      for (i = 0; i < 4; i++)
        wb_motor_set_velocity(wheels[i], 0.0);
      break;  // salir del bucle principal
    }

    // Evaluar sensores
    bool ds_detect_near = false, lidar_detect_near = false;
    double ds_values[2];
    for (i = 0; i < 2; i++) {
      ds_values[i] = wb_distance_sensor_get_value(ds[i]);
    }
    if (ds_values[0] < 950.0 || ds_values[1] < 950.0)
      ds_detect_near = true;

    // LIDAR
    const float *ranges = wb_lidar_get_range_image(lidar);
    int resolution = wb_lidar_get_horizontal_resolution(lidar);
    double fov = wb_lidar_get_fov(lidar);

    for (int i = 0; i < resolution; i++) {
      double angle = -fov / 2 + i * (fov / resolution);
      double dist = ranges[i];
      if (isinf(dist)) continue;

      if (dist < 1.0) {
        double obs_x = robot_x + dist * cos(angle);
```

```
        double obs_y = robot_y + dist * sin(angle);

        int cell_x = (int)((obs_x - origin_x + GRID_SIZE *
CELL_SIZE / 2) / CELL_SIZE);
        int cell_y = (int)((obs_y - origin_y + GRID_SIZE *
CELL_SIZE / 2) / CELL_SIZE);

        if (cell_x >= 0 && cell_x < GRID_SIZE && cell_y >= 0 &&
cell_y < GRID_SIZE)
          grid[cell_x][cell_y] = 1;
      }

      if (dist < 0.3)
        lidar_detect_near = true;
    }

    // Planificación
    int robot_cell_x = (int)((robot_x - origin_x + GRID_SIZE *
CELL_SIZE / 2) / CELL_SIZE);
    int robot_cell_y = (int)((robot_y - origin_y + GRID_SIZE *
CELL_SIZE / 2) / CELL_SIZE);
    Point start = {robot_cell_x, robot_cell_y};
    Point goal = {GRID_SIZE - 2, GRID_SIZE - 2};
    path_length = plan_path(grid, start, goal, path, 100);

    // Movimiento
    double left_speed = SPEED;
    double right_speed = SPEED;

    if (lidar_detect_near || ds_detect_near) {
      left_speed = 1.0;
      right_speed = -1.0;
    }

    // Detectar giros y registrar
    if (turn_count < MAX_TURNS) {
      if (left_speed == 1.0 && right_speed == -1.0 &&
!(last_left_speed == 1.0 && last_right_speed == -1.0)) {
        strcpy(turn_log[turn_count++], "izquierda");
      } else if (left_speed == -1.0 && right_speed == 1.0 &&
!(last_left_speed == -1.0 && last_right_speed == 1.0)) {
        strcpy(turn_log[turn_count++], "derecha");
      }
    }

    last_left_speed = left_speed;
    last_right_speed = right_speed;

    wb_motor_set_velocity(wheels[0], left_speed);
    wb_motor_set_velocity(wheels[1], right_speed);
    wb_motor_set_velocity(wheels[2], left_speed);
    wb_motor_set_velocity(wheels[3], right_speed);

    fflush(stdout);
  }

  wb_robot_cleanup();
```

```
  return 0;
}
```

B. Video

videoPrototipo.mp4