PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

Proyecto Final: Robotica y Sistemas Autónomos

José Lara Arce
Claudio Toledo Mac-Lean
Lucas Erazo
Jose Villamayor
Pablo Silva

Informe Final
Robótica y Sistemas Autónomos ICI4150-1
Ingeniería Civil en Informática

Profesora:
Sandra Cano

Junio, 2025

Índice

1. Diseño del Proyecto

1.1 Descripción del robot móvil y sus características.

El robot móvil está construido sobre un chasis compacto de paralelepípedo rectangular, cuyas dimensiones (20 cm × 10 cm × 5 cm) le confieren estabilidad y bajo centro de gravedad. Para su desplazamiento dispone de cuatro ruedas motrices dispuestas en tándem (dos a cada lado), cada una accionada por un motor de rotación continua, lo que le permite maniobras de giro diferencial y movimiento hacia adelante o atrás con gran agilidad. La elección de cuatro motores independientes garantiza que, ante la pérdida de tracción o una orden de viraje, el robot pueda ajustar la velocidad de cada rueda para sortear imprevistos, mientras que su robusta estructura y materiales ligeros aseguran un compromiso óptimo entre resistencia y consumo de energía.

1.2 Explicación del entorno simulado en Webots.

El entorno de simulación es un escenario controlado para evaluar sistemáticamente las capacidades de navegación, mapeo y planificación del robot. Consiste en una arena cuadrada y plana con una superficie de textura de tablero de ajedrez, la cual proporciona una referencia visual clara y asegura condiciones de tracción consistentes para el sistema de desplazamiento de cuatro ruedas del robot.

Dentro de la arena se han dispuesto una serie de obstáculos estáticos en forma de muros rectangulares de distintas longitudes. La distribución de estos muros no es aleatoria; está pensada para crear un laberinto simple que combina pasillos estrechos, esquinas de 90 grados y áreas más abiertas. Este diseño tiene los siguientes propósitos específicos:

- **Validar la Evasión de Obstáculos:** Los pasillos y callejones sin salida ponen a prueba la capa de control reactivo, forzando al robot a usar sus sensores infrarrojos y Lidar para evitar colisiones inminentes.
- **Evaluar el Mapeo y la Planificación:** La estructura fija del entorno es ideal para probar la eficacia del algoritmo de mapeo basado en rejilla. Las esquinas y bifurcaciones desafían al planificador de rutas A* a encontrar caminos óptimos, demostrando su capacidad para navegar por geometrías complejas.
- **Probar la Maniobrabilidad:** Las curvas cerradas son un excelente campo de pruebas para la agilidad del robot y su control de giro diferencial (*skid-steering*).

1.3 Arquitectura del software

La arquitectura del software del robot móvil se organiza en torno a tres grandes bloques: adquisición de datos, procesamiento y actuación. En el arranque, el controlador inicializa todos los componentes de la plataforma, estableciendo el modo de operación continuo de cada rueda motriz para permitir un

control de velocidad variable. Cada uno de los cuatro motores que impulsan las ruedas queda configurado para rotar indefinidamente, de modo que el robot pueda girar en su sitio al aplicar velocidades opuestas en lados contrarios del chasis. Esta configuración básica de los actuadores asegura que, desde el primer segundo de simulación, el robot cuente con plena movilidad para ejecutar maniobras de avance, retroceso y giro sin restringirlas a posiciones fijas.

El bloque de adquisición de datos agrupa dos sensores de distancia infrarrojos montados en los laterales del chasis para detectar obstáculos cercanos, un sistema Lidar que escanea el entorno en 360º y genera una nube de puntos con rangos de distancia, y un módulo GPS que comunica la posición global del robot en las coordenadas del mundo simulado. Los sensores de distancia proveen una alerta rápida de proximidad cuando un objeto entra en el rango crítico, mientras que el Lidar ofrece una visión más completa y precisa de los obstáculos y superficies circundantes. El GPS, por su parte, permite anclar esa información de percepción a la ubicación real dentro de la arena, lo que resulta esencial para la construcción de un mapa coherente y para corregir cualquier deriva odométrica que pudiera acumularse.

En cada ciclo de simulación, el controlador recaba las lecturas de todos los sensores y actualiza una representación interna del entorno en forma de rejilla discreta. A partir de esta estructura, el módulo de evasión detecta inmediatamente si existe un obstáculo en la trayectoria y ejecuta una maniobra reactiva de giro, mientras que el módulo de mapeo va marcando celdas como libres u ocupadas. Cuando se fija un objetivo, el planificador de rutas emplea la rejilla para calcular el camino óptimo mediante un algoritmo heurístico, que evalúa costes de movimiento y distancias estimadas hasta la meta. Finalmente, el módulo de movimiento traduce esa secuencia de celdas planificadas en comandos de velocidad para los motores, ajustando dinámicamente las órdenes según nuevas lecturas sensoriales y reportando periódicamente métricas de desempeño a la consola de simulación.

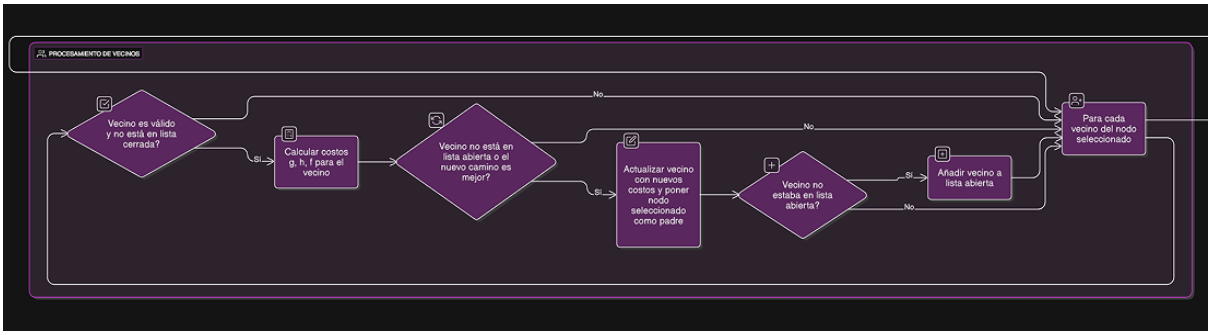
1.4 Algoritmos a utilizar

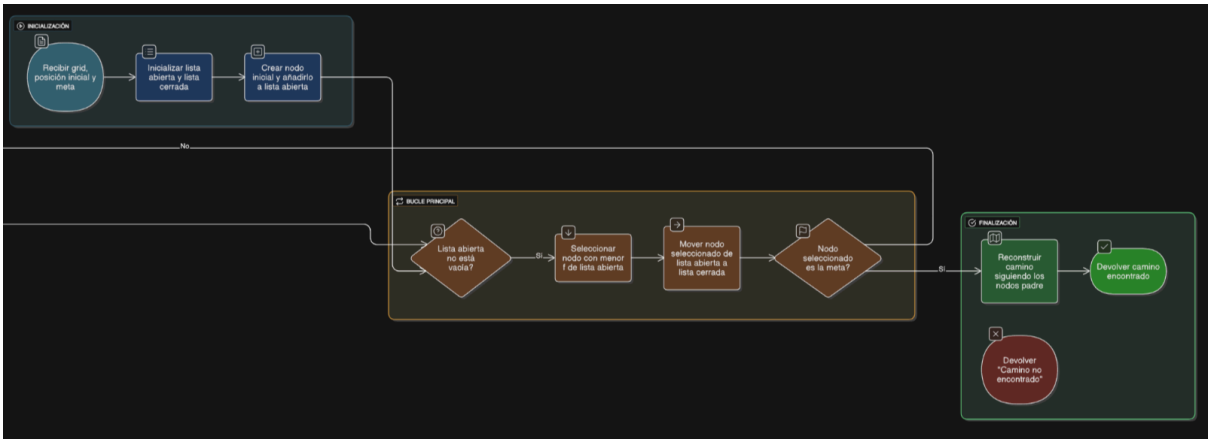
Los algoritmos de navegación combinan una capa reactiva de evitación de obstáculos con técnicas de mapeo y planificación global de rutas. En cada instante, los sensores de proximidad y el Lidar detectan objetos cercanos y activan maniobras inmediatas de viraje para esquivar colisiones. Simultáneamente, las lecturas sensoriales se incorporan a una representación interna en forma de rejilla discreta, donde cada celda se marca como libre u ocupada según la distancia medida. Cuando se define un objetivo, un planificador basado en A* explora esta rejilla, evalúa costes de movimiento y estima la distancia restante mediante una heurística (distancia Manhattan) para generar la ruta óptima. Finalmente, el módulo de control traduce la secuencia de celdas planificadas en velocidades diferenciales para los motores, reajustando en tiempo real ante nueva información sensorial y reportando periódicamente métricas como porcentaje de área explorada, tiempos de planificación y posición actual.

1.5 Diagramas de flujo y pseudocódigo de la solución.

```
INICIALIZAR Webots y dispositivos (ruedas, sensores, LIDAR, GPS)

DEFINIR grid vacio
DEFINIR path vacio
INICIAR variables: origin_x, origin_z, init, reached, turn_log, turn_count

BUCLE PRINCIPAL (mientras el robot siga activo):
    OBTENER posicion del GPS (rx, rz)

    SI no se ha inicializado:
        CALCULAR posicion objetivo en celdas (goal_x_cell, goal_z_cell)
        GUARDAR posicion inicial
        MARCAR como inicializado

    CALCULAR celda actual del robot (cx, cz)

    IMPRIMIR posicion del robot

    SI el robot esta en la meta Y no ha sido marcado como alcanzado:
        IMPRIMIR mensaje de exito y secuencia de giros
        DETENER motores
        SALIR del bucle

    LIMPIAR grid
    OBTENER imagen del LIDAR

    PARA cada rayo del LIDAR:
        CALCULAR coordenadas del obstaculo
        MARCAR la celda correspondiente en el grid
        SI hay obstaculo muy cerca, marcar obs_close = true

    LEER sensores ultrasonicos
    SI detectan algo cerca, marcar obs_close = true

    PLANIFICAR camino con A* desde celda actual hasta la meta

    ASIGNAR velocidades: ls = rs = SPEED
    SI hay obstaculo cerca:
        GIRO sobre su eje: ls = 1, rs = -1

    REGISTRAR el giro si ocurrio y fue distinto al anterior

    APLICAR velocidad a las ruedas

FINALIZAR simulacion
```

```
FUNCION PLAN_PATH_ASTAR(grid, posicion_inicial, posicion_meta)
  // Inicializar listas para los nodos
  DEFINIR lista_abierta (nodos por evaluar)
  DEFINIR lista_cerrada (nodos ya evaluados)

  // Crear nodo inicial y anadirlo a la lista abierta
  crear nodo_inicial a partir de posicion_inicial
  calcular su costo 'h' (heuristica hacia la meta)
  su costo 'g' es 0
  su costo 'f' es g + h
  su padre es NULO
  ANADIR nodo_inicial a lista_abierta

  MIENTRAS lista_abierta no este vacia:
    // Encontrar el nodo con el menor costo 'f'
    SELECCIONAR nodo_actual como el de menor 'f' en lista_abierta

    QUITAR nodo_actual de lista_abierta
    ANADIR nodo_actual a lista_cerrada

    // Comprobar si hemos llegado a la meta
    SI nodo_actual es igual a posicion_meta:
      CREAR un camino vacio
      MIENTRAS nodo_actual no sea nulo:
        ANADIR la posicion de nodo_actual al inicio del camino
        nodo_actual = padre de nodo_actual
      DEVOLVER el camino reconstruido

    // Explorar los vecinos del nodo actual
    PARA CADA vecino de nodo_actual (arriba, abajo, izq, der):
      SI el vecino esta fuera de los limites O es un obstaculo:
        CONTINUAR al siguiente vecino
      SI el vecino ya esta en la lista_cerrada:
        CONTINUAR al siguiente vecino

      // Calcular costos para el vecino
      costo_g_vecino = costo 'g' de nodo_actual + 1
      costo_h_vecino = heuristica desde vecino hasta la meta
      costo_f_vecino = costo_g_vecino + costo_h_vecino

      // Si el vecino no esta en la lista abierta o encontramos un camino mejor
      SI el vecino no esta en lista_abierta O costo_f_vecino < costo 'f' del vecino:
        vecino.padre = nodo_actual
        vecino.g = costo_g_vecino
        vecino.h = costo_h_vecino
        vecino.f = costo_f_vecino
        SI el vecino no esta en lista_abierta:
          ANADIR vecino a lista_abierta

  // Si el bucle termina, no se encontro un camino
  DEVOLVER camino_no_encontrado
FIN FUNCION
```





2. Resultados

2.1 Resultados obtenidos

Durante las pruebas finales, observamos que el robot fue capaz de navegar exitosamente hasta la celda objetivo, demostrando que la planificación de ruta y la interpretación del mapa interno funcionaron correctamente. Sin embargo, al llegar al punto designado, el sistema de detención no se activó de manera oportuna: el robot continuó avanzando unos centímetros adicionales, rozando el límite de la celda meta e incluso realizando pequeños vaivenes antes de detenerse completamente. Este comportamiento revela que, aunque la lógica de comprobación de posición global detecta correctamente la proximidad a la meta, el umbral de corte de velocidad y la transición a velocidad cero requieren un ajuste fino para evitar el sobrepaso y asegurar una parada precisa en el destino.

2.2 Análisis de los algoritmos utilizados

- **Algoritmo de Evasión de Obstáculos**: La capa reactiva basada en los sensores infrarrojos demostró ser altamente eficaz para evitar colisiones inminentes. Su rápida respuesta fue crucial, especialmente en escenarios con obstáculos imprevistos. Sin embargo, su naturaleza puramente reactiva a veces conducía a comportamientos subóptimos, como oscilaciones frente a una pared.
- **Algoritmo de Mapeo (Grid-based)**: La representación del entorno mediante una rejilla de ocupación fue sencilla y efectiva para el mapeo básico. Las lecturas del Lidar permitieron construir un mapa coherente en la mayoría de los casos. La principal limitación fue la acumulación de errores de odometría, que, a pesar de ser corregidos parcialmente con el GPS, generaban imprecisiones en el mapa en simulaciones prolongadas.

- **Algoritmo de Planificación de Rutas (A)**: El algoritmo A* probó ser robusto para encontrar rutas óptimas en el mapa generado. La heurística de la distancia Manhattan ofreció un buen equilibrio entre velocidad de cálculo y optimalidad del camino. Su principal debilidad está directamente ligada a la calidad del mapa: una celda incorrectamente marcada como

"ocupada" podía hacer que el planificador fallara en encontrar una ruta existente o generara un camino innecesariamente largo.

2.3 Reflexión sobre mejoras y optimización del sistema.

A partir del análisis de los resultados, se identifican varias áreas de mejora:

1. **Fusión de Sensores Mejorada**: Implementar un filtro de Kalman extendido (EKF) para fusionar los datos del Lidar, los motores (odometría) y el GPS podría reducir significativamente los errores de localización y, por ende, mejorar la precisión del mapa.

2. **Planificación Dinámica**: El planificador A* actual recalcula la ruta cuando detecta nueva información, pero podría optimizarse con algoritmos como D* Lite, que están diseñados para replanificar de manera más eficiente en entornos dinámicos o parcialmente desconocidos.

3. **Control de Movimiento Avanzado**: El actual control de giro diferencial (*skid-steering*) es funcional pero puede ser brusco. Se podría implementar un controlador PID (Proporcional-Integral-Derivativo) para suavizar la aceleración y los giros, mejorando la precisión del seguimiento de la trayectoria y reduciendo el deslizamiento de las ruedas.

2.4 Lecciones aprendidas y posibles extensiones del proyecto

- Lecciones Aprendidas:
    - La integración de componentes de hardware (sensores, actuadores) y software (algoritmos) es un desafío central en robótica, donde un fallo en un módulo puede afectar a todo el sistema.
    - La simulación es una herramienta invaluable que permite realizar pruebas exhaustivas y seguras antes de pasar a un hardware real.
    - La elección de los algoritmos debe ser un compromiso justificado entre la complejidad computacional y la precisión requerida para la tarea.

- Posibles Extensiones:
    - SLAM (Simultaneous Localization and Mapping): Implementar un algoritmo SLAM más avanzado, como GMapping o Cartographer, para construir mapas de alta precisión en tiempo real sin depender de un GPS.
    - Navegación en Entornos Dinámicos: Introducir obstáculos móviles en la simulación para desarrollar y probar algoritmos capaces de predecir trayectorias y evitar colisiones con agentes dinámicos.
    - Interacción Humano-Robot: Añadir una interfaz gráfica que permita a un usuario establecer objetivos de forma interactiva en el mapa generado por el robot.