DESDE 2013
https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI
ISSN: 2007-6363

Publicación Semestral Pädi Vol. 12 No. Especial 4 (2024) 332-344

Comparación de algoritmos de planeación de ruta para un robot móvil tipo Ackermann mediante ROS
Comparison of path planning algorithms for an Ackermann-type mobile robot using ROS

Ángel M. Valencia-Mendieta[a, *], Joahan Pacheco-Hernández[a], Javier Salas-Meneses[a], Rosalba Galván-Guerra[a], Silvestre A. García-Sánchez[a]

[a] *Instituto Politécnico Nacional, UPIIH, 42162, San Agustín Tlaxiaca, Hidalgo, México.*

Resumen

Este trabajo se centra en la comparación de tres algoritmos de planeación de ruta en un robot móvil tipo Ackermann. Los algoritmos evaluados son A*, Dijkstra y Campos Potenciales Artificiales, los cuales fueron implementados en un robot comercial. La evaluación se realizó en un entorno controlado, usando visión por computadora para localizar el robot y los obstáculos. Los algoritmos fueron comparados considerando: tiempo de ejecución, error de seguimiento de trayectoria del robot móvil y longitud del camino. Se utilizó el controlador Stanley para el seguimiento de trayectoria. Además, se adaptaron los algoritmos para abordar las limitaciones geométricas del robot Ackermann, como su radio máximo de giro. La implementación se realizó tanto en simulación como en un entorno real utilizando ROS (Robot Operating System). Se analizaron las trayectorias generadas por cada algoritmo y se evaluó su desempeño en condiciones reales y simuladas.

Palabras Clave: navegación autónoma, planeación de ruta, comparación, ROS.

Abstract

This work focuses on comparing three path planning algorithms in an Ackermann-type mobile robot. The evaluated algorithms are A*, Dijkstra, and Artificial Potential Fields, which were implemented in a commercial robot. The evaluation was conducted in a controlled environment, using computer vision to locate the robot and the obstacles. The algorithms were compared considering execution time, trajectory tracking error of the mobile robot, and path length. The Stanley controller was used for trajectory tracking. Additionally, the algorithms were adapted to address the geometric limitations of the Ackermann robot, such as its maximum turning radius. The implementation was conducted both in simulation and in a real environment using ROS (Robot Operating System). The trajectories generated by each algorithm were analysed and their performance was evaluated in real and simulated conditions.

Keywords: autonomous navigation, path planning, comparison, ROS.

1. Introducción

De acuerdo con una publicación de Amazon (Staff, A., 2022), el traslado de paquetes pesados y movimientos de torsión repetitivos son áreas en las que se busca automatizar con el fin de reducir el riesgo de lesiones en los trabajadores y, a su vez, aumentar la productividad. Normalmente se tiene un entorno controlado con rutas que permiten al trabajador llegar a puntos específicos para recoger la mercancía. Los trabajadores siguen rutas predefinidas y evaden los posibles obstáculos que se encuentren en la ruta como pueden ser cajas, personas, monta cargas, entre otros.

Los robots móviles y vehículos autónomos han sido parte de la solución adoptada por Amazon y la mayoría de las empresas de paquetería. Para lograr la navegación autónoma, el robot móvil debe ser capaz de percibir su entorno, tomar decisiones y moverse de forma autónoma. Estas tareas se conocen como el ciclo mira, piensa y actúa; que se muestra en la Figura 1. Este cuenta con las etapas de percepción, cognición y control de movimiento.

*Autor para la correspondencia: avalenciamendieta@gmail.com
Correo electrónico: avalenciamendieta@gmail.com (Ángel Moisés Valencia-Mendieta), jpachecoh1901@alumno.ipn.mx (Joahan Pacheco-Hernández), jsalasm1901@alumno.ipn.mx (Javier Salas-Meneses), rgalvang@ipn.mx (Rosalba Galván-Guerra), sagarcias@ipn.mx (Silvestre Ascención García-Sánchez).

Historial del manuscrito: recibido el 28/06/2024, última versión-revisada recibida el 26/08/2024, aceptado el 29/07/2024, publicado el 30/11/2024. **DOI:** https://doi.org/10.29057/icbi.v12iEspecial4.13324

En la etapa de percepción el robot móvil adquiere los datos de los sensores para así poder construir el mapa del entorno y localizarse, después este mapa y la localización del robot pasa a la etapa de cognición donde se planifica la trayectoria que esquiva los obstáculos estáticos para llegar al punto deseado, la última etapa de control de movimiento es donde se le provee la trayectoria a un controlador de seguimiento de trayectoria capaz de ejecutarla (Dan *et al,* 2018).

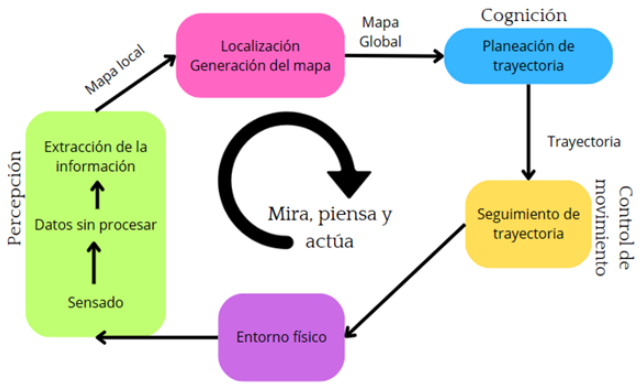

Figura 1: Componentes necesarias para la navegación autónoma. La imagen fue traducida al español. Imagen recuperada de (Dan *et al*., 2018).

En la navegación autónoma es fundamental el uso de sensores para reconocer el entorno, estos pueden ser cámaras, tecnología LiDAR, radares etc., (Figura 2). Estos proveen al vehículo de información de los obstáculos que se presenten en su entorno, esta etapa es conocida como percepción. También, es necesario contar con sensores que permitan conocer la localización del vehículo, estos pueden ser: sistemas GPS, sensores inerciales, encoders, unidades de medición inercial, etc. (Nahavandi *et al.*, 2022).

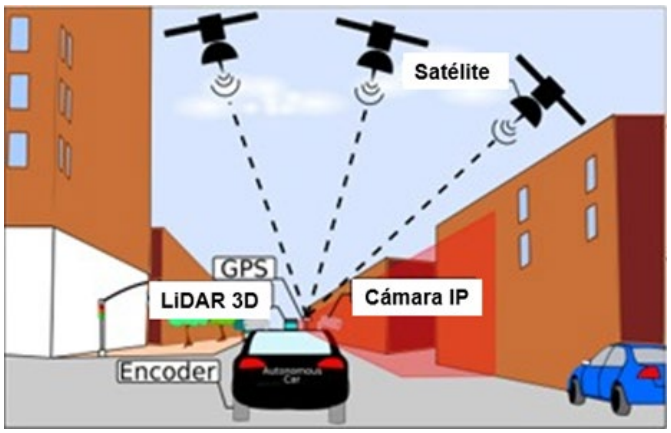

Figura 2: Sensores más utilizados para la navegación autónoma. La imagen fue traducida al español. Imagen recuperada de (Nahavandi et al., 2022).

Con los datos del entorno y de localización recuperados es necesario obtener un camino de un punto inicial hasta un punto final, esto sin colisionar (Nahavandi *et al.*, 2022). A esta etapa donde se planea esa trayectoria se le conoce como cognición. Existe una larga lista de algoritmos disponibles para resolver este problema en robots móviles, algunos ejemplos son: A*, Campos Potenciales Artificiales, algoritmos genéticos, Dijkstra, etc. (Zhang *et al.*, 2018). Cada uno de estos algoritmos tiene ventajas y desventajas, y en muchas aplicaciones son usados de forma colaborativa.

Los robots móviles deben estar instrumentados para poder ejecutar las trayectorias que proporcionan los algoritmos antes mencionados. Para resolver esta problemática a lo largo de los años se han desarrollado diversos métodos y algoritmos de seguimiento de trayectoria desde los más simples hasta los más complejos. (Nahavandi *et al.*, 2022). Algunos ejemplos de estos desarrollos serían Pure Pursuit, Stanley, métodos cinemáticos, dinámicos, etc. (Liu *et al.*, 2021).

En este trabajo se realiza una comparación de algunos de los algoritmos de planeación de ruta que se pueden encontrar en la literatura implementándolos en un robot móvil tipo Ackermann. Para ello primero se resuelve el problema de seguimiento de trayectoria, identificación de obstáculos mediante sensores y la localización del robot respecto a un marco de referencia fijo. Se presentan las desventajas de llevarlos a la práctica y las situaciones bajos las cuáles cada algoritmo es superior a los demás. Para realizar la comparación se considera el tiempo de cómputo, la longitud del camino generado y el error de seguimiento de la trayectoria.

El artículo está organizado de la siguiente manera: La sección 2 ejemplifica el problema a resolver en el trabajo. La sección 3 muestra la clasificación de algoritmos de planeación de trayectoria y como se escogieron. En la sección 4 se aborda la implementación de dichos algoritmos, con todos los componentes necesarios para llevar a cabo la navegación autónoma. La sección 5 presenta los resultados estadísticos de las pruebas físicas y simuladas, la sección 6 las conclusiones del trabajo y finalmente la sección 7 es la interpretación de los resultados estadísticos obtenidos con las pruebas.

2. Planteamiento del problema

Cuando se quiere utilizar un robot autónomo para realizar tareas de navegación en un entorno controlado como puede ser logística, almacenamiento, recolección, se conoce a priori el punto final al que quiere llegar el robot que está dado por las coordenadas de la mercancía a recoger o las coordenadas para dejar dicha mercancía (Peyas *et al.*, 2022).

Para resolver la navegación autónoma el robot debe ser capaz de, a partir de su posición inicial, llegar a una posición específica reconociendo la presencia de obstáculos, generando una ruta que evite las colisiones y siguiendo la ruta generada de forma adecuada (Dan *et al.*, 2018).

Existen diversos algoritmos de generación de rutas que se pueden implementar para resolver esta problemática, es por ello que surge la necesidad de compararlos para conocer las ventajas y desventajas al llevarlos a la práctica, y así escoger el más adecuado dependiendo de la aplicación y de las métricas que sean más importantes para la aplicación, si el tiempo de cálculo es un factor decisivo o la longitud del camino o incluso la suavidad y la fidelidad al seguir la trayectoria de acuerdo a la aplicación.

En los últimos años se han desarrollado herramientas computacionales que conforman un ambiente operativo denominado ROS, soportada por una gran comunidad de usuarios y desarrolladores, una plataforma sólida y muy atractiva para la programación de vehículos autónomos (Koubaa *et al.*, 2022), por lo cual se considera que la implementación en dicha plataforma permite evaluar el desempeño de los algoritmos en tiempo de cómputo, la

longitud del camino y qué tan viable es para el vehículo seguir dicha ruta considerando el error de seguimiento.

3. Selección de los algoritmos

La clasificación de los algoritmos de planificación de ruta se muestra en la Figura 3. Se seleccionan los algoritmos Dijkstra y A* clasificados como planificadores globales con un enfoque heurístico y el algoritmo de Campos Potenciales Artificiales clasificado como un algoritmo local (Zhang *et al.*, 2018). La selección de los algoritmos se realiza así para comparar Dijkstra y A* que son dos algoritmos clásicos basados en grafos y evaluarlos frente a métodos basados en fuerzas y potenciales.

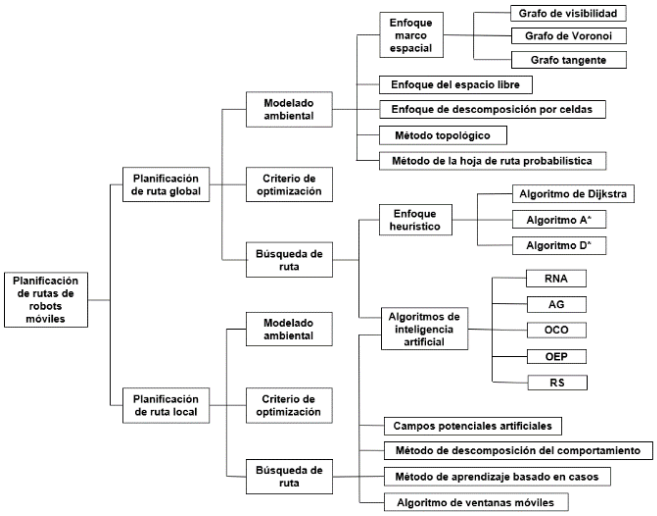

Figura 3: Clasificación de los algoritmos de generación de camino y sus componentes esenciales. La imagen fue traducida al español. Imagen recuperada de (Zhang *et al.*, 2018).

Debido a la naturaleza exhaustiva del algoritmo de Dijkstra (Zhang *et al.*, 2018), el costo computacional puede ser alto. Para verificar esto y evaluar la pertinencia de usar dicho algoritmo se analizó su tiempo de cómputo. Además, se modificó el algoritmo para que la búsqueda del camino dentro del grafo no se realice buscando en todos los nodos, sino que al encontrar el nodo objetivo el algoritmo finalice la búsqueda.

Así en la Tabla 1, se muestran los tiempos de cómputo del algoritmo original y el adaptado en el entorno de simulación, considerando el mismo punto de inicio y meta. Con la adaptación se disminuyó casi a la mitad el tiempo de cómputo del algoritmo Dijkstra. Es claro, que dicha adaptación puede afectar el desempeño del algoritmo en términos de la optimalidad del camino encontrado, pero permite que este algoritmo pueda ser comparado con algoritmos más eficientes en términos del tiempo de cómputo.

A pesar de que campos potenciales es mayormente usado como un algoritmo local se desea observar su rendimiento siendo utilizado para resolver el problema de la generación de camino de una forma global, realizando los ajustes y adaptaciones necesarias.

Tabla 1:Comparativa del tiempo de cálculo de Dijkstra original y adaptado.

Versión del algoritmo de Dijkstra	Tiempo de cálculo [s]
Original	0.09265661239624023
Adaptada	0.057588815689086914

En el apéndice A se muestran los pseudocódigos de cada algoritmo y en el apéndice B se anexa un enlace a un video demostrativo del funcionamiento de estos algoritmos.

4. Diseño del experimento

El experimento consiste en hacer pruebas de manera repetida con el robot móvil seleccionado en el entorno controlado.

4.1. Robot móvil

Para llevar a cabo la implementación y comparación de los algoritmos de evasión de obstáculos se utiliza el robot móvil comercial Pi-Car de la marca Sunfounder. Figura 4. El robot cuenta con la geometría Ackermann, que es la que se busca estudiar en este trabajo. También con una tarjeta de desarrollo Raspberry PI de 2gb de memoria RAM, además de 2 baterías 18650 de 9000mah.

Figura 4: Pi-Car de Sunfounder. Imagen recuperada del sitio web de Sunfounder www.sunfouder.com

En la Tabla 2 se muestran las características físicas identificadas del robot móvil. En la Figura 5 se muestran las medidas del robot, además en la Figura 6 se muestra el ángulo máximo y mínimo δ que puede girar cada rueda debido a la limitante física del mecanismo de la dirección.

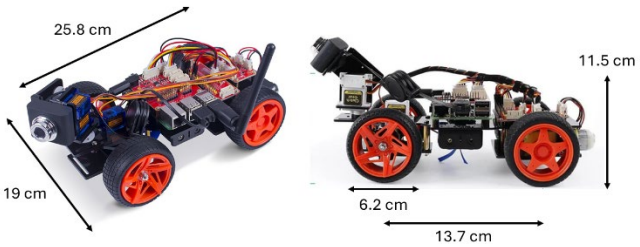

Figura 5: Medidas del robot móvil. Imagen recuperada del sitio web de Sunfounder www.sunfouder.com

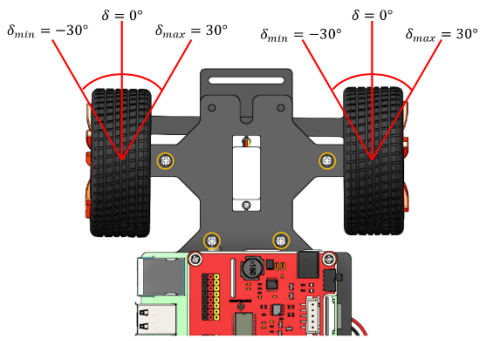

Figura 6: Ángulo máximo y mínimo δ que puede girar cada rueda delantera. Imagen recuperada del sitio web de Sunfounder www.sunfouder.com

Tabla 2: Características físicas del robot móvil comercial a utilizar: Pi-Car.

Característica	Unidad	Magnitud
Diámetro de las ruedas	cm	6.2
Velocidad lineal máxima	m/s	0.25
Medidas	cm	25.8 (largo) x 19 (ancho) x 11.5 (altura)
Distancia de eje trasero a delantero	cm	13.7
Ángulo de giro máximo de la dirección	grados °	-30 a 30

4.2. Entorno controlado

El entorno controlado consiste en una tabla MDF de $1.52m\ x\ 1.83m$ pintada de color blanco para así obtener un mayor contraste con los obstáculos favoreciendo a los algoritmos de visión por computadora. Una cámara se encuentra montada sobre un soporte que la mantiene a una altura de 1.91m y a un ángulo de 45° (Figura 7) para poder observar toda el área de la tabla.

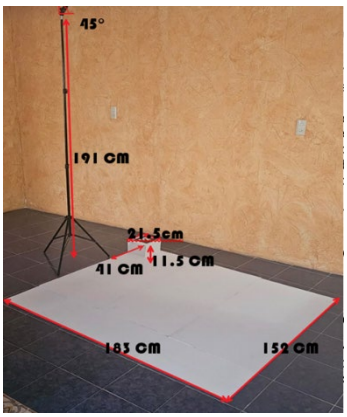

Figura 7: Entorno controlado del experimento.

4.3. Abstracción virtual del entorno controlado

Se considera un mapa formado por celdas cuadradas de forma que componen un grafo ponderado no dirigido, los pesos se asignan como se describe a continuación:

- El mapa se descompone en celdas cuadradas, y se considera un peso de 1 para el movimiento a las aristas y un peso de $\sqrt{2}$ para los movimientos en diagonal, considerando entonces un desplazamiento de 8 puntos como se muestra en la Figura 8.

- El camino generado es en 2 dimensiones, únicamente contiene las coordenadas (x, y) a las cuales el robot móvil debe ir en cada momento hasta llegar al objetivo, esto es posible ya que el robot se desplaza con un perfil de velocidad conocido.

- Los nodos de inicio y meta se indican a través de pares de coordenadas (x_i, y_i) y (x_m, y_m) respectivamente, donde (x_i, y_i, x_m, y_m) son enteros positivos, que se encuentren dentro del mapa y que representan nodos que estén libres.

Comprendidas las condiciones físicas del entorno de pruebas, se establecen las siguientes condiciones para el mapa virtual utilizado para la generación del camino:

- Cada celda será de 5cm x 5cm para simplificar la navegación del robot considerando su anchura, esto implica que el mapa completo se comprenderá de una dimensión de 30x37 casillas de forma redondeada.

- Para facilitar la navegación del robot móvil dentro del mapa se engrosan los obstáculos de manera virtual el equivalente a 15 cm o 3 celdas.

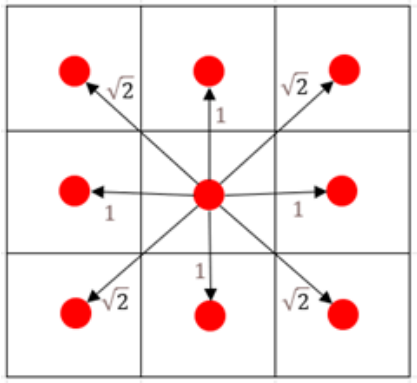

Figura 8: Tipos de movimientos permitidos dentro del mapa.

4.4. Localización del robot móvil

La localización es el proceso de determinar la posición y orientación del robot con respecto a su entorno (Huang *et al.*, 2016). Esta información es crucial para planear una trayectoria que evada los obstáculos y tome en cuenta la posición y orientación actual del robot. En este trabajo se utiliza una cámara para visualizar el entorno sobre el que el robot debe moverse (Figura 9), para ello se emplean marcadores Aruco de la librería OpenCV de Python, uno sobre el robot móvil y otro colocado en la esquina inferior izquierda del entorno cuya función es ser el origen del sistema de referencia del plano donde el robot se mueve. Se obtiene la posición y orientación de ambos marcadores con respecto al foco de la cámara para luego obtener la posición y orientación de marcador del robot con respecto al marcador del origen.

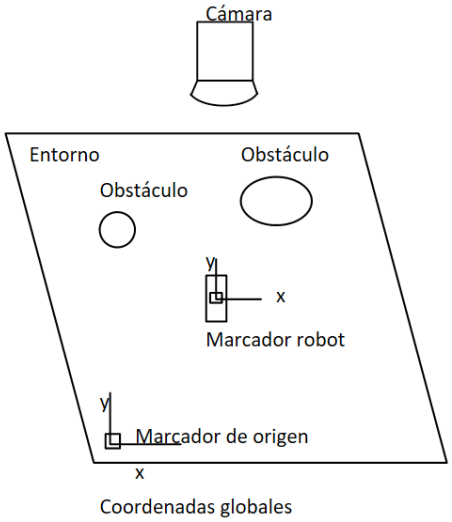

Figura 9: Elementos que componen el entorno controlado.

4.5. Detección de obstáculos

Los obstáculos se definen como cajas cuadradas, las cuales tienen una tapa de color azul que contrasta con el color blanco del piso, se utiliza visión máquina para identificar los obstáculos mediante un proceso de segmentación de color. Una vez identificados los obstáculos se aproximan como círculos, posteriormente se realiza una estimación de la posición y tamaño de cada uno para generar un mapa virtual.

5. Implementación

Los algoritmos de planificación de ruta se programan en el lenguaje Python utilizando ROS (Robot Operating System).

5.1. Seguimiento de trayectoria

Existen diversos algoritmos de seguimiento de trayectoria que se diferencian principalmente de acuerdo con el modelo matemático que utilizan para representar al robot móvil: geométrico, cinemático y dinámico (Snider *et al.*, 2016), Snider realizó una comparativa de algunos de los algoritmos de seguimiento de trayectoria más utilizados.

Los resultados muestran que a bajas velocidades los algoritmos geométricos ofrecen muy buenos resultados, Pure pursuit y Stanley obtienen rendimientos comparables, aunque el autor resalta que Stanley supera a Pure Pursuit en la mayoría de las ocasiones, por tal razón y por su simplicidad en la implementación, se eligió a Stanley como el algoritmo a utilizar en este trabajo.

El controlador de seguimiento de trayectoria Stanley es un algoritmo geométrico que consiste en reducir la distancia lateral del vehículo y un punto en la trayectoria y al mismo tiempo orientarlo de manera que siga el mismo rumbo que dicha trayectoria (Hoffman *et al.*, 2007), esto se ilustra en la Figura 10.

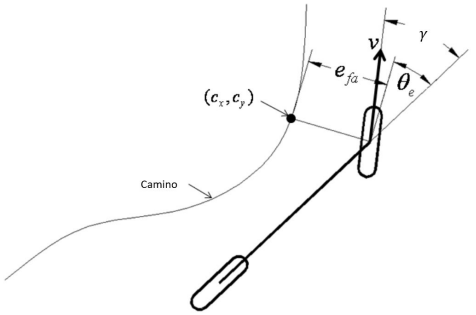

Figura 10: Geometría del algoritmo de seguimiento de trayectoria Stanley. Imagen recuperada de (Snider, 2016).

La ley de control se presenta en (1) (Snider, 2016), donde k es la ganancia del controlador, k_{soft} es una ganancia que evita que el denominador se vuelva cero y sea muy sensible al ruido y $v_x(t)$ es la velocidad lineal del vehículo que para este trabajo se supone constante.

$$\gamma(t) = \theta_e(t) + tan^{-1}\left(\frac{ke_{fa}(t)}{k_{soft}+v_x(t)}\right), \quad (1)$$

Donde:

- $\theta_e = \theta - \theta_p$ es el error de orientación del vehículo entre el ángulo de la orientación del vehículo respecto al eje z (θ) y el ángulo deseado en un punto en la trayectoria c_x, c_y (θ_p).
- e_{fa} denota el error lateral del eje delantero.
- γ es el ángulo de giro del eje de dirección.
- k_{soft} es una ganancia de suavizado, para este trabajo se utilizará un valor de 0.1m/s como en el utilizado en (Snider *et al*, 2016).
- k es la constante de control y se sintoniza de acuerdo con el rendimiento deseado.

La sintonización de la ganancia del controlador se realiza siguiendo el método propuesto en (Seiffer *et al*, 2023) donde se propone un método para sintonizar esta constante en el que se aumenta de manera gradual obteniendo el error cuadrático medio hasta que los efectos dinámicos del robot ya no son perceptibles. En este vehículo los efectos dinámicos no están presentes debido a la velocidad a la cual se puede mover que es muy baja (alrededor de $0.2\frac{m}{s}$), entonces se propuso una trayectoria de prueba arbitraria donde en simulación el robot móvil recorrió dicha trayectoria hasta obtener un error cuadrático medio menor a $1cm$ con $k = 4.0$.

5.2. Sistema en ROS

La implementación del sistema en ROS se ilustra en el diagrama de flujo presentado en la Figura 11.

Inicialmente, deben especificarse las condiciones iniciales de posición y orientación del robot móvil. Posteriormente, se indica el punto objetivo en el mapa al que se desea que el robot se dirija. Estas especificaciones constituyen las entradas del

algoritmo de planificación de ruta, el cual puede ser A*, Dijkstra o Campos Potenciales Artificiales. Una vez generada la ruta, se inicia el seguimiento de la trayectoria, entrando en un bucle en el que se verifica en cada iteración si la ruta ha sido completada para entonces finalizar el programa. En caso contrario, se envían señales de control al robot en forma de un valor de PWM y un ángulo de dirección. Debido a las limitantes de procesamiento del sistema embebido (RaspBerry PI) y a que el vehículo cuenta con una capacidad limitada de energía otorgada por baterías, se utiliza la característica de cómputo distribuido de ROS para dividir las tareas del proceso entre una computadora y la RaspBerry PI montada en el vehículo, a esta característica se le denomina ROS Network (ROS team, s.f.).

Tabla 3: Condiciones iniciales y meta para cada mapa en físico.

Mapa	Condiciones iniciales			Meta	
Mapa 1	$x_0 = 0.28\,m$	$y_0 = 0.22\,m$	$\theta_0 = 0°$	$x_f = 0.9\,m$	$y_f = 1.5\,m$
Mapa 2	$x_0 = 0.28\,m$	$y_0 = 0.22\,m$	$\theta_0 = 0°$	$x_f = 0.9\,m$	$y_f = 1.5\,m$
Mapa 3	$x_0 = 0.7\,m$	$y_0 = 0.1655m$	$\theta_0 = 90°$	$x_f = 0.9\,m$	$y_f = 1.5\,m$

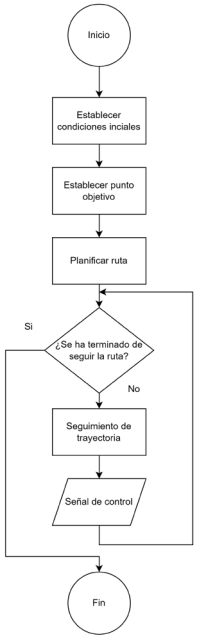

Figura 11: Diagrama de flujo general del sistema de planeación de ruta.

En la Figura 12 se muestran los elementos de la red de ROS, se utiliza una computadora como cliente, la cual ejecuta ROS y se encarga del procesamiento principal ejecutando los algoritmos de generación de camino, los algoritmos de visión por computadora y se encarga además de la visualización con el software nativo de ROS RVIZ.

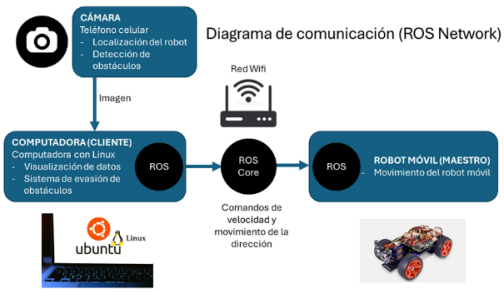

Figura 12: Diagrama de comunicación (ROS Network).

Resultados

Se definen tres mapas con distintas distribuciones de obstáculos, y se procede a colocar al vehículo en una posición y orientación inicial, dándole un punto de destino sobre el cuál cada algoritmo generó una ruta. Se realiza una serie de pruebas simuladas donde se recolectan las variables de: longitud de camino y tiempo de cálculo para cada algoritmo. Además, se procede a validar los algoritmos en el entorno controlado, donde se obtienen, además de las variables de las pruebas simuladas, el error de seguimiento de trayectoria. En la Figura 13, Figura 14 y Figura 15 se muestran la distribución, forma, tamaño y ubicación de los obstáculos en cada mapa.

5.3. Resultados de las pruebas físicas

Se realizan 12 pruebas por cada algoritmo (Campos Potenciales Artificiales, Dijkstra y A*) en cada mapa, dando un total de 108 pruebas. Se coloca al vehículo en una posición y orientación inicial sobre el mapa físico y se establece un punto de destino.

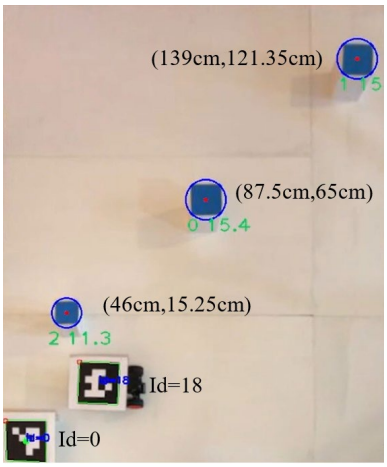

Figura 13: Configuración del mapa 1. Se muestran las coordenadas (x, y) de cada obstáculo en color negro y en verde el número id asignado y el diámetro en centímetros del círculo que lo encierra.

Figura 14: Configuración del mapa 2. Se muestran las coordenadas (x, y) de cada obstáculo en color negro y en verde el número id asignado y el diámetro en centímetros del círculo que lo encierra.

En la Tabla 3 se definen las condiciones iniciales para cada mapa, donde (x_0, y_0, θ_0) son las coordenadas y la orientación inicial del vehículo y (x_f, y_f) son las coordenadas del punto meta.

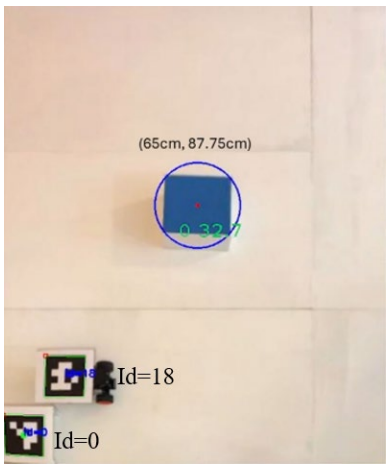

Figura 15: Configuración del mapa 3. Se muestran las coordenadas (x, y) de cada obstáculo en color negro y en verde el número id asignado y el diámetro en centímetros del círculo que lo encierra.

5.3.1. *Evasión de obstáculos en el mapa 1*

En el mapa 1 el robot comienza con una orientación paralela al eje horizontal mientras que el punto meta se encuentra en la parte superior del mapa, esto hace que el robot deba hacer un cambio de dirección de 90° en relativamente poco tiempo, este es uno de los desafíos más grandes para la geometría Ackermann.

En la Figura 16 se muestra el resultado de la generación de ruta y el seguimiento de trayectoria para cada algoritmo en el mapa 1. Se toma la primera de las 12 pruebas realizadas en este mapa. Se nota que a pesar de que las 12 pruebas son ejecutadas bajo las mismas condiciones, las rutas y el seguimiento no son los mismos debido a la presencia de ruido en la localización del robot y de los obstáculos lo que implica que cada prueba tiene un mapa y condiciones iniciales con pequeñas variaciones.

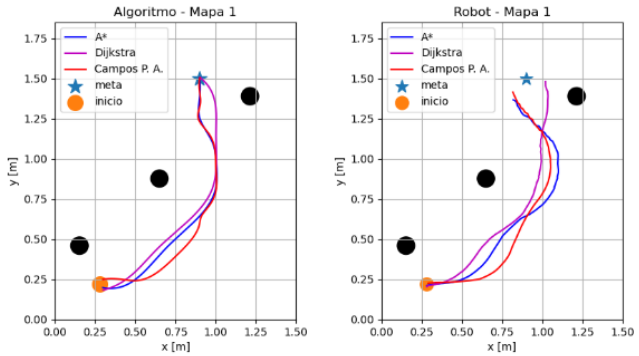

Figura 16: Resultados de la evasión de obstáculos en el mapa 1, a la izquierda se muestra la gráfica comparativa del camino suavizado de cada algoritmo, mientras que a la derecha está el seguimiento realizado por el robot.

Como se puede notar, a pesar de obtener un camino suavizado para el vehículo, este aún toma trayectorias con una curvatura mayor, incapaz de seguir los segmentos similares a una línea recta. Aunque se observa un cierto error de seguimiento, es posible para el robot llegar al punto meta sin colisionar en todas las pruebas y para todos los algoritmos demostrando que el factor de engrosamiento virtual de los obstáculos es adecuado.

5.3.2. *Evasión de obstáculos en el mapa 2*

El mapa 2 es muy similar al mapa 1 con la diferencia de que se agregan dos obstáculos extra para formar una segunda diagonal. Las condiciones iniciales y la meta son las mismas que en el mapa 1.

En la Figura 17 se muestran los resultados de la evasión de obstáculos en el mapa 2, se observa que el camino generado por los 3 algoritmos es muy similar al primer mapa, en algunas de las 12 pruebas se nota que el obstáculo de la parte inferior derecha contribuye a hacer más cerrada la curva inicial que debe tomar el vehículo.

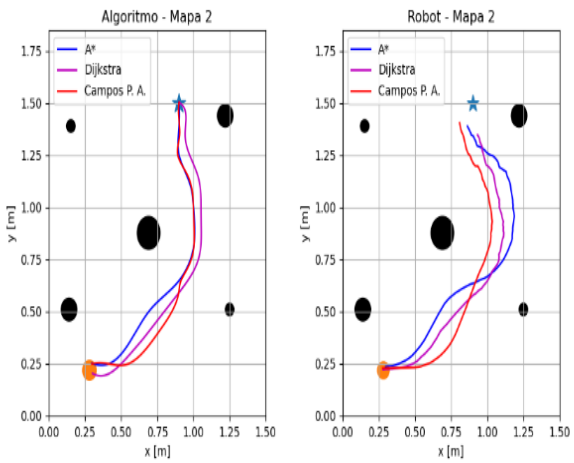

Figura 17: Resultados de la evasión de obstáculos en el mapa 2.

Es importante mencionar que en este mapa Dijkstra arroja dos rutas diferentes a lo largo de las 12 pruebas, una que es muy similar a los otros dos algoritmos y otra que en lugar de

evitar al obstáculo central por la derecha lo hacía por la izquierda (Figura 18), esto es un problema para la geometría Ackermann ya que esta ruta implica hacer un giro mucho más cerrado y que debido a las limitaciones físicas del vehículo hace que pase muy cerca del obstáculo central, incluso puede observarse que después de evitar el obstáculo central le es difícil al vehículo recuperar la trayectoria inicial.

Es posible observar que hay una zona de los mapas 1 y 2 donde parece que el seguimiento de trayectoria empeora a simple vista, esta zona es la parte superior derecha del mapa que va desde la zona a la derecha del obstáculo central hasta la zona del punto meta, se identifica que en esta área el sistema de visión máquina deja de detectar por pequeños instantes de tiempo al marcador del robot, lo que afecta de forma directa el rendimiento del seguimiento de trayectoria.

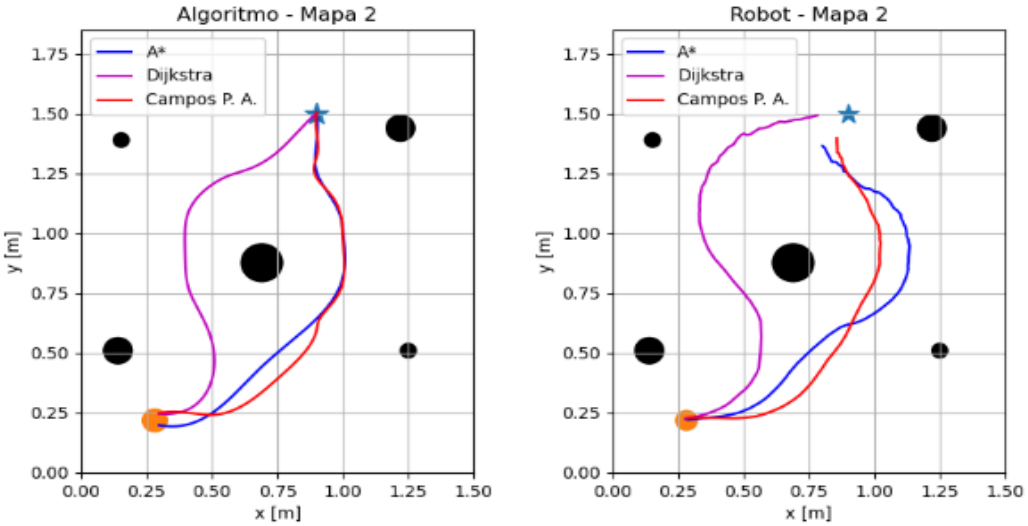

Figura 18: Resultados de la evasión de obstáculos en el mapa 2, se muestra el camino alternativo tomado por Dijkstra en ciertas ocasiones.

Tabla 4: Condiciones iniciales y meta para cada mapa en simulación.

Mapa	Condiciones iniciales			Meta	
Mapa 1, 2 y 3	$x_0 = 0.75\,m$	$y_0 = 0.45\,m$	$\theta_0 = 0°$	$x_f = 0.9\,m$	$y_f = 1.5\,m$
	$x_0 = 0.75\,m$	$y_0 = 0.45\,m$	$\theta_0 = 90°$	$x_f = 0.9\,m$	$y_f = 1.5\,m$
	$x_0 = 0.75\,m$	$y_0 = 0.45m$	$\theta_0 = 180°$	$x_f = 0.9\,m$	$y_f = 1.5\,m$
	$x_0 = 0.75\,m$	$y_0 = 0.45m$	$\theta_0 = 270°$	$x_f = 0.9\,m$	$y_f = 1.5\,m$

5.3.3. *Evasión de obstáculos en el mapa 3*

En el mapa 3 las condiciones iniciales del robot cambian, ahora se encuentra frente a un obstáculo de mayor tamaño, este es un escenario común en la navegación autónoma donde el robot debe evitar colisionar con un obstáculo singular. Esta configuración es más cómoda de afrontar para un robot tipo Ackermann ya que la curvatura del camino es menor.

En la Figura 19 se muestran los resultados de la evasión de obstáculos en el mapa 3, a grandes rasgos se observa que los 3 algoritmos tomaron rutas similares evadiendo al obstáculo por la derecha. Por su parte al robot no le fue difícil seguir las curvas generadas.

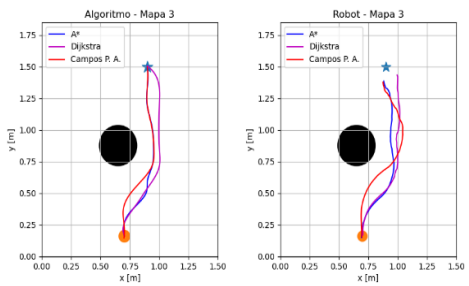

Tabla 4: Condiciones iniciales y meta para cada mapa en simulación.

5.3.4. *Resultados cuantitativos*

Para cada una de las pruebas se recaba la información de las variables seleccionadas: longitud del camino y tiempo de cómputo, y en el caso de las pruebas en físico, el error de seguimiento. A continuación, se presentan los resultados cuantitativos con enfoque en la métrica.

En la Figura 20 se muestra el diagrama de caja y bigotes para la longitud del camino. Cabe resaltar que la longitud del camino se obtiene sobre la ruta suavizada. De esta gráfica se puede deducir lo siguiente:

- En la Figura 20 se observa que la posición de la mediana es menor en el caso de A* por lo que se puede deducir que obtuvo rutas con una menor longitud. Campos potenciales por otra parte tiende a generar las rutas de mayor longitud.

- Las cajas de A* tienen una menor amplitud con respecto a los otros algoritmos lo que indica una menor

dispersión en los datos. Lo contrario se observa con Dijkstra cuyos resultados se encuentran más dispersos.

- El diseño del mapa 3 permite a los algoritmos tener una menor dispersión en los datos, esto debido a que solo había un único obstáculo que además estaba colocado en la posición central del mapa con mejor iluminación donde el ruido en su posición era menor.

- Los mapas 1 y 2 requieren de rutas con formas más curvas que con el mapa 3. Este tipo de geometría hace que Campos potenciales genere caminos más largos con respecto a los otros algoritmos.

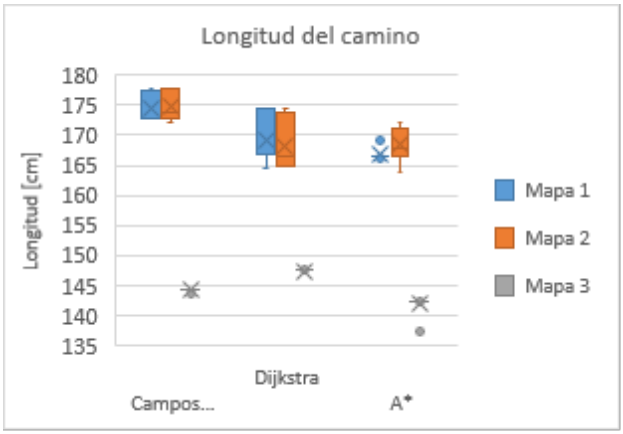

Figura 20: Resultados de la longitud del camino para cada mapa y algoritmo en las pruebas físicas.

En la Figura 21 se muestra el diagrama de caja y bigotes para el tiempo de cálculo, este se midió desde el momento en el que se pasaba el punto de destino al algoritmo hasta el momento en el que se genera por completo la ruta. De esta gráfica se puede deducir lo siguiente:

- Se observa por la ubicación de la mediana que A* tiene un mejor rendimiento respecto al tiempo de cálculo en los 3 mapas. Campos potenciales muestra los tiempos de cálculo más altos de entre los 3 algoritmos.

- A* y Dijkstra tienen cajas con una menor amplitud lo que indica una menor dispersión en los datos. Por su parte Campos potenciales llega a presentar valores atípicos.

En la Figura 22 se muestra el diagrama de caja y bigotes para el error lateral cuadrático medio, este error se obtuvo como la distancia medida desde el centro del eje delantero hacia la ruta en cada iteración del controlador de seguimiento de trayectoria cuya frecuencia está establecida en $100Hz$.

El error lateral es una variable que se ve afectada por las condiciones del entorno, depende principalmente de la posición y orientación actual del robot que son obtenidas mediante algoritmos de visión máquina, por lo tanto, la iluminación que es el principal factor que afecta a la visión también hace que el error lateral tienda a aumentar. Se nota que

Tabla 5: Resultados de la comparación.

Algoritmo	Longitud de camino	Tiempo de cálculo	Error de seguimiento	Ventajas	Desventajas	Posibles aplicaciones
A*	Corto	Bajo	Medio	La función de heurística encuentra el camino más corto	Las rutas generadas requieren ser suavizadas	Navegación local y global donde se requiera una respuesta rápida y una ruta corta
Dijkstra	Moderado	Medio	Bajo, aunque depende de la variabilidad al generar rutas	Encuentra el camino más corto, siempre que se use la versión completa	Puede tomar demasiado tiempo de cálculo si no se limita	Planificación de rutas globales donde el tiempo de cálculo no sea crítico
Campos P. A.	Considerable	Alto	Bajo	Genera rutas suaves	Puede caer en mínimos locales	Planificación de rutas locales a bajas velocidades

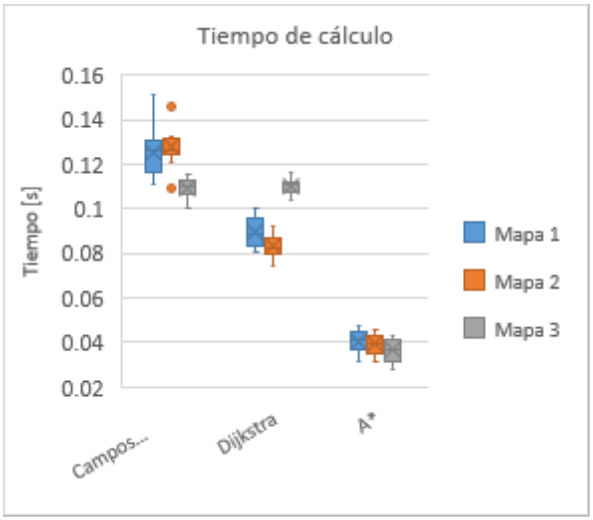

Figura 21: Resultados del tiempo de cálculo para cada mapa y algoritmo en las pruebas físicas.

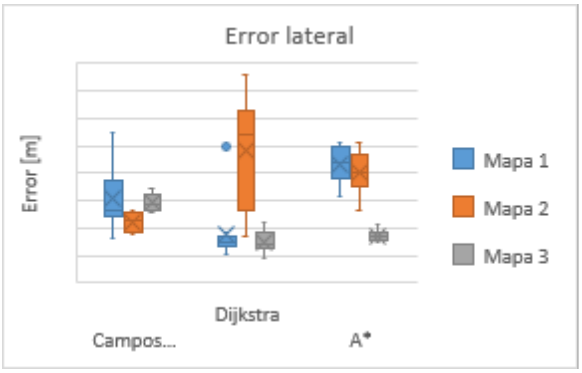

Figura 22: Resultados del error lateral para cada mapa y algoritmo en las pruebas físicas.

las condiciones de iluminación hacen que el robot deje de ser detectado por ciertos instantes de tiempo lo que ocasiona que el controlador no conozca la información necesaria para hacer la corrección en la dirección del robot a tiempo.

El sistema de evasión de obstáculos funciona a través de una red inalámbrica, por lo que el rendimiento de esta es clave para su correcto funcionamiento, se requiere de tener una cantidad adecuada de ancho de banda que permita la transferencia de la imagen con muy poco tiempo de retraso. Si esto no se consigue, el controlador no puede conocer la ubicación actual del robot y no hace una corrección de la dirección adecuada.

Los factores mencionados anteriormente afectan a los 3 algoritmos, sin embargo, se nota que algunos son más tolerables a este tipo de eventos.

- Un caso especial es el algoritmo de Dijkstra que en el mapa 2 que obtiene un error lateral con una dispersión considerable, esto debido a que generó 2 rutas distintas para llegar al mismo punto. Se observa que el ruido en la posición de los obstáculos sumado al ruido presente en la posición inicial del robot antes de comenzar a generar la ruta, hacen que el camino generado no sea el mismo usando cualquiera de los 3 algoritmos, sin embargo, el más sensible a este tipo de perturbaciones es Dijkstra.

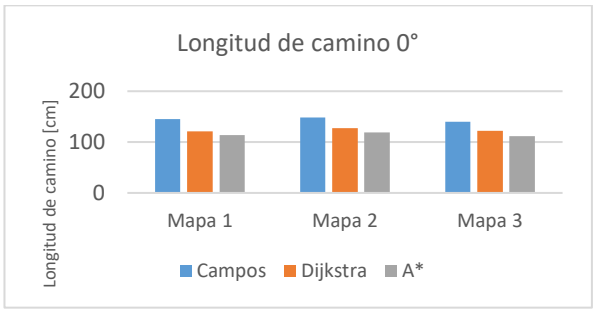

Figura 23: Longitud de camino generada en cada mapa con una condición inicial $\theta = 0°$.

5.4. Resultados de las pruebas simuladas

Se realizaron 349 viajes por cada algoritmo (Campos Potenciales, Dijkstra, A*) probando 4 condiciones iniciales diferentes por cada mapa, dando un total de 4188 pruebas por mapa. En la Tabla 4 se definen las condiciones iniciales para los tres mapas.

5.4.1. Resultados en la longitud de camino generado

En simulación los algoritmos de generación de ruta no se ven afectados por el ruido, permitiendo de este modo que la longitud de camino se mantenga constante en cada prueba, en la Figura 23 se muestran los resultados en longitud de camino de cada algoritmo, solamente haciendo uso del resultado para un ángulo de $\theta = 0°$ dado que las demás pruebas arrojan resultados similares.

Como se puede apreciar, A* es el algoritmo que siempre generó la ruta más corta mientras que campos potenciales siempre devolvió la ruta más larga.

5.4.2. Resultados en los tiempos de cálculo de cada algoritmo

El tiempo que se considera comprende desde el instante en el cual se manda el punto de origen hasta que cada algoritmo devuelve el camino generado. En la Figura 24, Figura 25 y Figura 26 se presentan la desviación estándar y la mediana del tiempo de cálculo de cada algoritmo para cada mapa y para la condición inicial $\theta = 0°$.

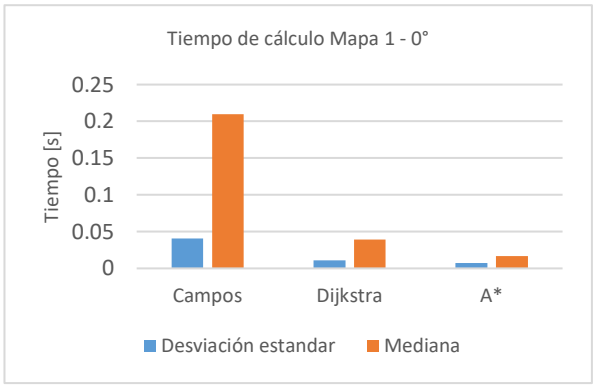

Figura 24: Desviación estándar y mediana del tiempo de cálculo de cada algoritmo en el mapa 1, $\theta = 0°$.

Se puede observar, A* es el algoritmo que siempre mantiene los menores tiempos de cálculo y el de tiempos más prolongados es campos potenciales.

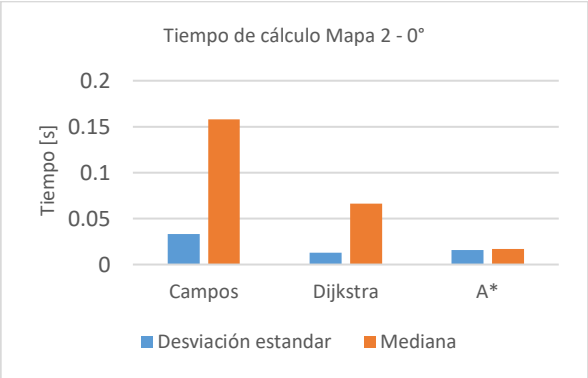

Figura 25: Desviación estándar y mediana del tiempo de cálculo de cada algoritmo en el mapa 2, $\theta = 0°$.

6. Discusión

Con los experimentos realizados tanto en el entorno físico como en el simulado se obtuvo información que permite identificar algunas de las ventajas y desventajas de los algoritmos de planificación de ruta seleccionados frente a diversas situaciones.

En la Tabla 5 se muestra un resumen de los resultados observados en las pruebas físicas y simuladas. Se observa que A* es el algoritmo que genera las rutas más cortas en el menor tiempo, su función heurística que es su principal diferenciador es de gran ayuda en el tiempo de cómputo, con respecto a Dijkstra, debido a su modificación no se aprovecha todo su potencial para conseguir siempre el camino más corto dentro de un grafo, esta modificación le sirve para reducir su tiempo de cálculo, sin embargo no fue suficiente para alcanzar el rendimiento de A*, por su parte, Campos Potenciales tuvo un rendimiento inferior con respecto a la longitud del camino y el tiempo de cálculo, sin embargo se observa que debido a la naturaleza de las rutas generadas con este algoritmo, se obtiene un error de seguimiento menor lo que indica que este algoritmo favorece a la geometría Ackermann que se ve bastante limitada por incorporar el eje delantero.

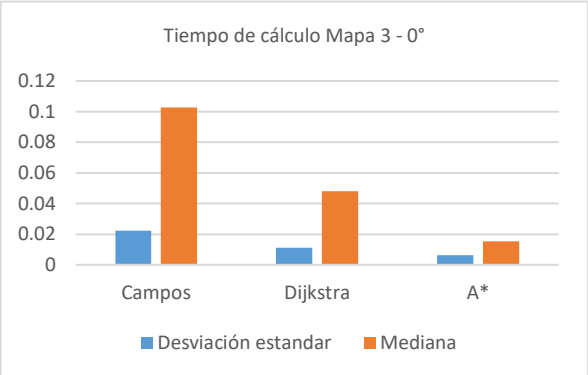

Figura 26: Desviación estándar y mediana del tiempo de cálculo de cada algoritmo en el mapa 3, $\theta = 0°$.

La principal desventaja en la implementación de los algoritmos basados en grafos como A* y Dijkstra es que tienden a generar rutas con giros bruscos que son difíciles de conseguir en una situación real por lo que es necesario hacer un suavizado de la ruta que termina inevitablemente cambiando de forma.

Campos Potenciales Artificiales es un algoritmo que puede caer en mínimos locales bajo diversas situaciones, esto es un factor importante para tener en cuenta debido a que es necesario compensar este hecho de alguna manera, ya sea incorporando algún otro algoritmo como apoyo o robusteciéndolo para lidiar con este tipo de situaciones.

Con esta información es posible identificar algunas aplicaciones de los algoritmos seleccionados dentro de la navegación autónoma. A* es un algoritmo rápido y que es capaz de generar un camino corto por lo que puede ser aplicado tanto en navegación local donde se requiere de un tiempo de respuesta rápido y en la navegación global donde es importante obtener un camino fiable y donde una ruta corta puede ser importante para mantener un consumo energético bajo. Por otro lado, Dijkstra es un algoritmo adecuado para la navegación global ya que es capaz de encontrar el camino más corto a costa de un tiempo de cálculo mayor.

Campos potenciales destaca principalmente por generar caminos suaves que fueron más fáciles de seguir para la geometría Ackermann por lo que es una buena opción si el robot móvil tiene este tipo de geometría.

7. Conclusiones

Se compararon tres algoritmos de planeación de ruta implementados en un robot móvil tipo Ackermann mediante ROS: Campos Potenciales Artificiales, Dijkstra y A*, usando las métricas de tiempo de cálculo, longitud de camino y error lateral. Con los resultados se identificaron ventajas y desventajas de cada algoritmo tanto en las pruebas físicas como en las simuladas, para luego hacer estadística con todos los datos.

Sin importar el entorno A* es el algoritmo que siempre genera la ruta en el menor tiempo de cálculo y en su mayoría con la menor longitud, mientras que Campos Potenciales es el algoritmo que tarda más tiempo en hacer la misma tarea.

Agradecimientos

Queremos expresar nuestro más sincero agradecimiento a nuestras familias, cuyo apoyo incondicional y comprensión fueron fundamentales durante todo el proceso de investigación y redacción de este artículo. Los autores agradecen el apoyo económico de la Secretaría de Investigación y Posgrado del Instituto Politécnico Nacional SIP-IPN, proyectos: 20241197 y 20242437

Referencias

C. Mesquita. (2015). UPNA. https://academica-e.unavarra.es/handle/2454/18665

Dan, R. T., Shah, U., & Hussain, W. (2018). Development Process of a Smart UAV for Autonomous Target Detection. Proceedings Of The 16th LACCEI International Multi-Conference For Engineering, Education, And Technology: "Innovation In Education And Inclusion". DOI: 10.18687/laccei2018.1.1.480

Hoffmann, G. M., Tomlin, C. J., Montemerlo, M., & Thrun, S. (2007). Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing. Proceedings Of The . . . American Control Conference/Proceedings Of The American Control Conference.
DOI: 10.1109/acc.2007.4282788

Huang, S., & Dissanayake, G. (2016). Robot Localization: An Introduction. Wiley Encyclopedia Of Electrical And Electronics Engineering, 1-10. DOI: 10.1002/047134608x.w8318

Koubâa, A., Bennaceur, H., Chaari, I., Trigui, S., Ammar, A., Sriti, M., Alajlan, M., Cheikhrouhou, O., & Javed, Y. (2019). Robot Path Planning and Cooperation: Foundations, Algorithms and Experimentations. https://link.springer.com/content/pdf/10.1007/978-3-319-77042-0.pdf

Koubaa, A. K. (2015). ROS As a Service: Web Services for Robot Operating System. JOSER - Journal Of Software Engineering For Robotics, https://aisberg.unibg.it/retrieve/handle/10446/87694/159561/97-477-1-PB.pdf. https://aisberg.unibg.it/retrieve/handle/10446/87694/159561/97-477-1-PB.pdf

Liu, J., Yang, Z., Huang, Z., Li, W., Dang, S., & Li, H. (2021). Simulation Performance Evaluation of Pure Pursuit, Stanley, LQR, MPC Controller for Autonomous Vehicles. RCAR. https://doi.org/10.1109/rcar52367.2021.9517448

Nahavandi, S., Alizadehsani, R., Nahavandi, D., Mohamed, S., Mohajer, N., Rokonuzzaman, M., & Hossain, I. (2022). A Comprehensive Review on Autonomous Navigation. arXiv (Cornell University). DOI: 10.48550/arxiv.2212.12808

Peyas, I. S., Hasan, Z., Tushar, R. R., Musabbir, A. M., Mehjabin Azni, R. M. A., & Siddique, S. (2022). Autonomous Warehouse Robot using Deep Q-Learning. ResearchGate. https://www.researchgate.net/publication/358762766_Autonomous_Warehouse_Robot_using_Deep_Q-Learning

ROS team. (s. f.). ROS/Tutorials/MultipleMachines - ROS Wiki. http://wiki.ros.org/ROS/Tutorials/MultipleMachines

Seiffer, A., Frey, M., Gauterin, F. (2023). Pragmatic and Effective Enhancements for Stanley Path-Tracking Controller by Considering System Delay. Vehicles 2023, 5, 615–636. DOI: 10.3390/vehicles5020034

Snider, J. (2019). Automatic Steering Methods for Autonomous Automobile Path Tracking.

Staff, A. (2022). Look back on 10 years of Amazon robotics. https://www.aboutamazon.com/news/operations/10-years-of-amazon-robotics-how-robots-help-sort-packages-move-product-and-improve-safety

Zhang, H. Z., Lin, W. L., & Chen, A. C. (2018). Path Planning for the Mobile Robot: A review. Symmetry. https://doi.org/10.3390/sym10100450

Apéndice A. Descripción del funcionamiento de los algoritmos seleccionados

A continuación, se presentan los pseudocódigos de los algoritmos implementados.

Procedimiento algoritmo de Dijkstra

1. S = conjunto de nodos visitados
2. g (nodo inicial) = 0 #Distancia del nodo inicial
3. g (demás nodos) = infinito #Distancia de los demás nodos al inicio
4. predecesor (todos los nodos) = nulo
5. Nodo actual = nodo inicial
6. U = Conjunto de no visitados
7. U se inicializa con todos los nodos
8. Mientras el conjunto U no este vacío hacer:
    a. Nodo actual = nodo $n \in U$ con $f(n) < f(n')$ #Costo menor dentro de la vecindad de n
    b. Si nodo actual = nodo meta entonces:
        i. Devolver el camino encontrado
    c. Si no, agregar el nodo actual al conjunto S y quitar del conjunto U
    d. Para cada nodo n' vecino del nodo n actual hacer:
        i. Si ocupación(n') =libre y S(n') =nulo, entonces:
            1. Costo g(n') = criterio de costos #Calcula nuevo costo o distancia
        ii. Si g(n') < g(n') anterior, entonces:
            1. g(n') anterior = g(n')
            2. predecesor(n') = n
            3. f(n') = g(n') #Actualizar costo de cada vecino
            4. Regresar al paso 8
9. Si S=nulo:
    a. Imprimir: "No se encontró ningún camino posible"

Fin algoritmo de Dijkstra

*Procedimiento algoritmo A**

1. S = conjunto de nodos visitados
2. g (nodo inicial) = 0 #Distancia del nodo inicial
3. g (demás nodos) = infinito #Distancia de los demás nodos al inicio
4. predecesor (todos los nodos) = nulo
5. Nodo actual = nodo inicial
6. U = Conjunto de no visitados
7. U se inicializa con todos los nodos
8. Mientras el conjunto U no este vacío hacer:
    a. Nodo actual = nodo $n \in U$ con f(n)<f(n') #Costo menor dentro de la vecindad de n
    b. Si nodo actual = nodo meta entonces:
        i. Devolver el camino encontrado
    c. Si no, agregar el nodo actual al conjunto S y quitar del conjunto U

d. Para cada nodo n' vecino del nodo n actual hacer:

i. Si ocupación(n') = libre y S(n') =nulo, entonces:

1. Costo g(n') = criterio de costos #Calcula nuevo costo o distancia

ii. Si g(n') < g(n') anterior, entonces:

1. g(n') anterior = g(n')

2. h(n') = función de heurística empleada

3. predecesor(n') = n

4. f(n') = g(n') + h(n´) #Actualizar costo de cada vecino

5. Regresar al paso 8

9. Si S=nulo:

a. Imprimir: "No se encontró ningún camino posible"

*Fin algoritmo A**

Procedimiento algoritmo campos potenciales artificiales

1. posición = posición inicial.

2. tamaño_paso = 1.

3. mientras gradiente > ε

a. grad = gradiente(posición) #calcular el gradiente

b. Siguiente posición = posición – tamaño_paso * grad #Avanza a la siguiente posición en #dirección opuesta al crecimiento del gradiente

c. posición = Siguiente posición #se actualiza la posición actual con la siguiente posición

Fin algoritmo campos potenciales artificiales

Apéndice B. Enlace a video demostrativo

En el siguiente enlace se puede ver un video demostrativo: https://youtu.be/aA__u1j6Fbw?si=9qi9-4hhz_qGk6FZ