Número Especial de la Revista Aristas: Investigación Básica y Aplicada. ISSN 2007-9478, Vol.10, Núm. 18. Año 2023

Implementación de algoritmos A* y Dijkstra para planificación de trayectorias en la navegación autónoma de un robot diferencial

Luis Rodolfo Macias, Ulises Orozco-Rosas, Kenia Picos*

CETYS Universidad, Av. CETYS Universidad No. 4. El Lago, C.P. 22210, Tijuana B.C., México
rodolfo.macias@cetys.edu.mx, ulises.orozco@cetys.mx, kenia.picos@cetys.mx

Resumen

En este trabajo se busca implementar una solución eficiente al problema de navegación y planificación de trayectoria para la robótica autónoma a través de los algoritmos A* y Dijkstra en un robot con tracción diferencial. El presente artículo documenta todo el proceso de la implementación, así como los resultados de los algoritmos previamente mencionados en un ambiente simulado para su comparación. La implementación consiste de manera general en conseguir ciertas trayectorias a través de modificaciones en los algoritmos A* y Dijkstra, posteriormente procesar esas trayectorias conseguidas para convertirlas en el movimiento que tiene que seguir el robot simulado. El proceso de conversión de trayectoria a movimiento consiste en dos controladores P que controlan las velocidades, tanto angulares como lineales, dependiendo de la posición del robot en el ambiente. El robot calcula su posición mediante algoritmos que incluyen cálculos estadísticos y sensores que le permiten conocer con un cierto nivel de exactitud en que posición esta con referencia al ambiente. La implementación fue realizada en el simulador para robótica Gazebo junto con la integración del *framework* especializado para robótica ROS el cual tiene conjuntos de librerías y herramientas que ayudan a construir cualquier tipo de aplicación para robótica.

Palabras clave— Algoritmo A*, Algoritmo de Dijkstra, Planificación de trayectorias, ROS, Gazebo, Turtlebot.

Abstract

This paper presents a solution to the problem of navigation and trajectory planning for autonomous robotics through the A and Dijkstra's algorithms in a robot with differential traction. This article documents the entire implementation process, as well as the results of the previously mentioned algorithms in a simulated robotics environment for comparison. The implementation generally consists of obtaining certain trajectories through slight modifications in the A* and Dijkstra's algorithms and later processing these trajectories to convert them into the movement that the simulated robot has to follow. The trajectory-to-motion conversion process consists of two P controllers that control both angular and linear speeds depending on the robot's position in the environment. The robot calculates its position using statistical algorithms and sensors that allow it to know with a certain level of accuracy which position it is concerning the environment, the implementation was carried out in a robotics simulator known as Gazebo along with the integration of the specialized framework. for robotics ROS which has several sets of libraries and tools that help build any kind of robotics application.*

Keywords*— A* algorithm, Dijkstra's algorithm, Path planning, ROS, Gazebo, Turtlebot.*

1. INTRODUCCIÓN

Los robots autónomos recientemente representan una de las tecnologías emergentes con más aplicaciones comerciales y futuro tecnológico para la sociedad, con los avances de la robótica y computación cada vez los problemas y retos se vuelven más complejos y recurren a otras áreas, una de esas áreas son las ciencias computacionales, específicamente en problemas de optimización [1].

Uno de los problemas más comunes en la robótica, es encontrar una ruta adecuada al ambiente en el que se ubica el agente de una manera eficiente; Actualmente sigue siendo un problema que se sigue investigando y aplicando distintas metodologías de solución [2].

En el presente trabajo se muestra el desarrollo e implementación de los algoritmos Dijkstra y A* para la planificación de trayectoria en el robot Turtlebot 2 simulado en Gazebo y controlado por medio del *framework* especializado en robótica ROS. Para el ambiente simulado se utilizaron distintos mapas en Gazebo de una cuadricula de 10x10 metros, dicha cuadricula es representada en un grafo para el posterior procesamiento de los algoritmos de planificación de trayectoria y finalmente el controlador fue implementado para desplazarse en esa misma trayectoria generada por el algoritmo (A* o Dijkstra).

2. FUNDAMENTOS

A continuación, se presenta una descripción general de los conceptos utilizados en este trabajo.

2.1 ROS

ROS (*Robot Operating System*), consiste en un *framework* especializado para el desarrollo de software enfocado al área de la robótica, en dicho *framework* se cuenta con diversas herramientas de código abierto para su aplicación directa a ambientes y sistemas robóticos [3].

* Autor para la correspondencia: *ulises.orozco@cetys.mx*

2.2 Controlador proporcional

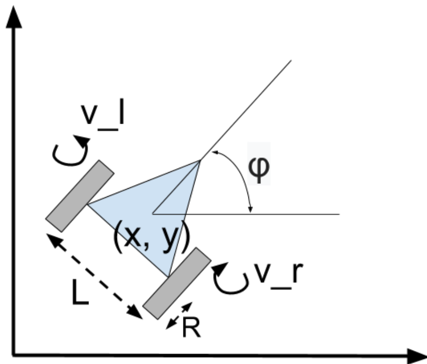
El controlador proporcional es un amplificador con ganancia ajustable. Este control permite reducir el tiempo necesario para que la salida pase del 0% al 100% de su valor final, incrementa el sobretiro y reduce el error de estado estable. Matemáticamente, la relación entre la salida del controlador y su respectivo error es expresada por $u(t)$ como se muestra en la Ecuación 1 [4].

$$u(t) = k_p e(t) \tag{1}$$

2.3 Robot diferencial

Un robot de tracción diferencial consiste en aquel donde dos de sus ruedas se encuentran montadas sobre un eje único, estas son propulsadas y controladas de manera independiente, brindando tracción y direccionamiento al robot [5]. El direccionamiento se da por la diferencia de velocidades de las ruedas laterales, por otro lado, la tracción también se consigue con estas ruedas [6]. De igual manera suele agregarse una rueda giratoria para brindarle estabilidad al sistema al momento de ejecutar la serie de movimientos. En la Figura 1 se muestra un ejemplo del sistema cinemático que se utilizó para determinar el sistema de localización, donde L es el ancho de la base, v_l y v_r son las velocidades de las ruedas, φ es el ángulo con respecto a su posición x-y [7].

Figura 1. Sistema cinemático de un robot diferencial



2.4 Odometría

La odometría es parte importante de los sistemas de navegación que refiere al estudio de la estimación de la posición y orientación de un robot móvil a partir de alguna referencia [7], el caso más común es obtener el número de pulsos obtenidos al momento que giran las ruedas. Para ello, la distancia recorrida se puede determinar por medio de *encoders* acoplados a las ruedas del robot [8]. Una ventaja es que es relativamente simple y de bajo costo, sin embargo, requiere de calibración constante por el desgaste del robot [9]. Por otro lado, existen desventajas que de manera general se pueden denominar como errores que se clasifican en dos categorías: errores sistemáticos que van desde diferencias de diámetros de las ruedas y resolución de *encoder,* y los errores no sistemáticos, de los cuales pueden ser deslizamientos en la superficie de contacto, así como posibles fuerzas internas o externas aplicadas al sistema [10].

Para el cálculo de la estimación de la posición de un robot de tracción diferencial se utilizan las Ecuaciones (2, 3, 4) [11].

$$x(t + dt) = x(t) + \frac{\Delta d_R + \Delta d_L}{2} \cos\, \theta(t) \tag{2}$$

$$y(t + dt) = y(t) + \frac{\Delta d + \Delta d_L}{2} \sin\, \theta(t) \tag{3}$$

$$\theta = \frac{\Delta d_R - \Delta d_L}{b} \tag{4}$$

Donde b es la distancia entre las dos ruedas, Δd_R y Δd_L son las respectivas diferencias de distancia de ambas llantas. x e y son la posición global con respecto al inicio y θ es la orientación.

2.5 Gazebo

Gazebo es un simulador que nos permite imitar un entorno tridimensional con aplicaciones al campo de la robótica. Este software nos permite diseñar robots con sus respectivos sensores, se puede crear diversos entornos en 3D que asemejaran las características de entornos tanto interiores como exteriores, y una característica muy importante para el presente trabajo es que Gazebo nos permite implementar sistemas de control con el *framework* de ROS [12].

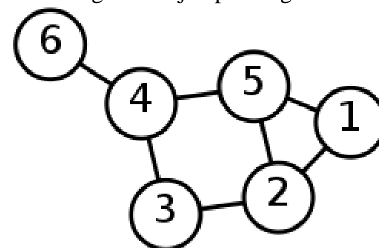
2.6 Turtlebot

Es un sistema robótico diferencial comercial que cuenta ya con un sistema de control de motores y diversos sensores para aplicaciones de investigación y desarrollo en el área de robots autónomos, el modelo que se utiliza en el presente trabajo es el Turtlebot 2, el cual cuenta con una cámara RGB-D para calcular su posicionamiento por medio de un algoritmo con cálculos estadísticos [13].

2.7 Grafo

Un grafo es una estructura de datos no lineal que consta de vértices y aristas, cada vértice se tiene un identificador el cual puede ser algún número, símbolo o texto, en la Figura 2 se utilizan números como identificadores de cada vértice. Los vértices también se denominan nodos y los bordes son líneas o arcos que conectan dos nodos en el grafo, de manera general estos grafos sirven para representar muchos tipos de problemas y por ende encontrar una solución adecuada, en este caso la aplicación que se utiliza en el presente trabajo es utilizar un grafo para representar el ambiente en el cual se encuentra el robot y por medio de este grafo aplicar un algoritmo de búsqueda para encontrar una ruta eficiente [14].

Figura 2. Ejemplo de grafo



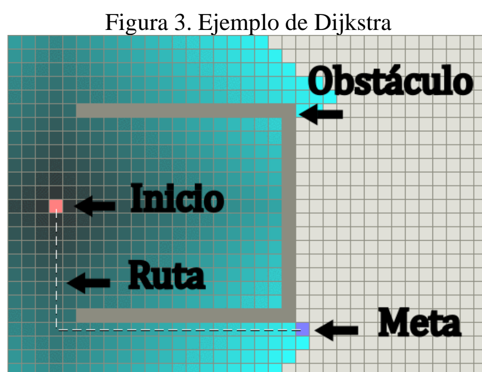
2.8 Algoritmo de Dijkstra

El algoritmo de Dijkstra básicamente comienza en el nodo que se elija (el nodo de origen) y analiza el grafico para encontrar la ruta más corta entre ese nodo y todos los demás nodos del grafo.

El algoritmo realiza un seguimiento de la distancia más corta actualmente conocida desde cada nodo al nodo de origen y actualiza estos valores si encuentra una ruta más corta. Una vez que el algoritmo ha encontrado la ruta más corta entre el nodo de origen y otro nodo, ese nodo se marca como "visitado" y se agrega a la ruta.

El proceso sigue de igual manera hasta que todos los nodos del grafo se hayan agregado a la ruta. De esta forma, tenemos una ruta que conecta el nodo fuente con todos los demás nodos siguiendo la ruta más corta posible para llegar a cada nodo [15]. En la Figura 3 se muestra un ejemplo gráfico de la exploración del algoritmo de Dijkstra, donde mientras más brillante el color azul significa mayor costo para llegar a tal nodo.

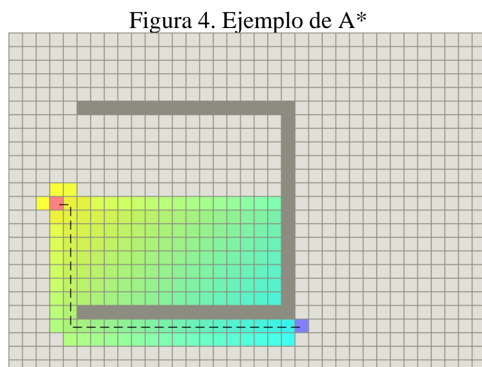
Figura 3. Ejemplo de Dijkstra



2.9 Algoritmo A*

El algoritmo de A* es muy similar al de Dijkstra, sin embargo el mismo algoritmo guía su exploración por medio de una heurística la cual consiste que en vez de explorar los nodos vecinos en todas las direcciones, busca los nodos que tengan un menor costo, este cálculo de costo se calcula tal que $f(n) = g(n) + h(n)$, siendo $g(n)$ el costo exacto del nodo y $h(n)$ representa que tan lejos está el nodo del final, gracias a esta aproximación, la búsqueda de A* es más voraz que la de Dijkstra y generalmente se obtienen resultados precisos para este tipo de problemas [15].

Figura 4. Ejemplo de A*



En la Figura 4 se muestra un ejemplo gráfico de la exploración del algoritmo de A*, Los colores son una escala de colores gradientes y significan el valor del costo calculado por A* siendo azul el costo más bajo ya que está en la meta y amarillo para el inicio ya que se encuentra a una distancia promedio de la máxima.

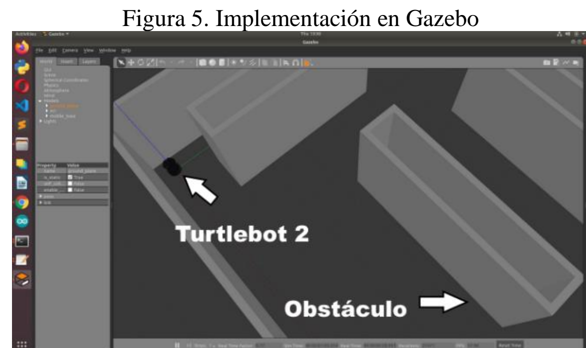
3. METODOLOGÍA

Para la elaboración del presente trabajo, se realizaron diversas tareas en una computadora con sistema operativo Ubuntu 18.04 debido a que ROS trabaja nativamente en Linux y existe una amplia documentación.

3.1 Implementación de un ambiente de trabajo en ROS para el Turtlebot 2

Para la implementación del ambiente de trabajo se elaboró desde inicio una instalación nueva de Ubuntu 18.04 en una computadora, posteriormente se instaló el *framework* de ROS junto con librerías y paquetes especiales para la simulación del Turtlebot 2. Después de la instalación de ROS y sus paquetes, se elaboró un espacio de trabajo virtual, el cual contiene los archivos necesarios para inicializar la simulación en Gazebo con el Turtlebot 2 junto con un ambiente de ROS el cual permite la comunicación con nodos personalizados. En la Figura 5 se muestra un ejemplo de una implementación en Gazebo sencilla con obstáculos.

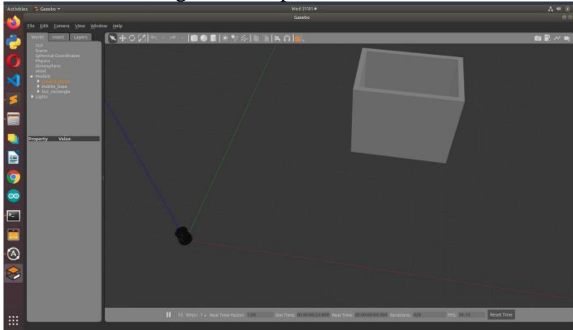
Figura 5. Implementación en Gazebo



3.2 Creación de mapas en Gazebo

La manera en la que se implementó el trabajo fue cargar distintos mapas a partir de distintos archivos de tipo CSV (*comma-separated values*), estos archivos fueron utilizados para la creación de mapas en un ambiente simulado como Gazebo. Los mapas fueron recreados utilizando distintas herramientas de edición con las que ya cuenta el software de Gazebo. En la Figura 6 se muestra la simulación del Turtlebot 2 con el mapa 1 en Gazebo.

Figura 6. Mapa 1 en Gazebo



3.3 Implementación de los algoritmos Dijkstra y A*.

El Código 1 y 2 describen las implementaciones propias de los algoritmos Dijkstra y A* para encontrar las rutas de un punto "A" hacia un punto "B" dentro de cualquier grafo. Ambos códigos están desarrollados en el lenguaje de Python y se utilizan diccionarios para almacenar la información de cada nodo.

Código 1. Código para implementación del algoritmo Dijkstra

```
def dijkstra(graph,start,goal):
 dist = {}
 from_nodes = {}
 visitedNodes = []
 unvisitedNodes = graph
 inf = 9999999
 path = []
 for node in unvisitedNodes:
  dist[node] = inf
  dist[start] = 0

 while unvisitedNodes:
  smNode = min(unvisitedNodes, key=lambda
node: dist[node])    #Smallest node

  for nbNode, weight in
graph[smNode].items():    #nb = Neighbor
   if weight + dist[smNode] < dist[nbNode]:
    dist[nbNode] = weight + dist[smNode]
   from_nodes[nbNode] = smNode
   unvisitedNodes.pop(smNode)
   visitedNodes.append(smNode)
   if smNode == goal:
    break

   C_Node = goal
  if dist[goal] == inf:
    print("The path doesn't exist")
    return None
  while C_Node != start:
   path.insert(0, C_Node)
   C_Node = from_nodes[C_Node]

  path.insert(0,start)
```

Código 2. Código para implementación del algoritmo A*

```
def a_star(graph,start,goal):
    dist = {}
    dist_goal = {}
    dist_total = {}
    from_nodes = {}
    visitedNodes = []
    unvisitedNodes = graph
    inf = 9999999
    path = []
    for node in unvisitedNodes:
        dist[node] = inf
    dist[start] = 0

    for node in unvisitedNodes:
  dist_goal[node] = euclidian(node, goal)

    #update total distances
    for node in unvisitedNodes:
        dist_total[node] = dist[node] +
dist_goal[node]

    while unvisitedNodes:

        smNode = min(unvisitedNodes,
key=lambda node: dist_total[node])
#Smallest node

        for nbNode, weight in
graph[smNode].items():    #nb = Neighbor
            if weight + dist[smNode] <
dist[nbNode]:

            dist[nbNode] = weight +
dist[smNode]
            dist_total[nbNode] = dist[nbNode]
+ dist_goal[nbNode]
            from_nodes[nbNode] = smNode

        unvisitedNodes.pop(smNode)
        visitedNodes.append(smNode)

        if smNode == goal:
            break
    C_Node = goal
    if dist_total[goal] == inf:
        print("The path doesn't exist")
        return None

    while C_Node != start:
        path.insert(0, C_Node)
        C_Node = from_nodes[C_Node]

    path.insert(0,start)
```

3.4 Implementación del controlador para el Turtlebot 2

Para hacer que el robot pueda desplazarse, se implementó un controlador proporcional para lograr determinar las velocidades lineales y angulares del robot, dependiendo del error determinado por la posición que se encuentra con respecto a la posición deseada hasta llegar a esa misma posición, este controlador se utiliza para poder desplazar el robot en alguna de sus vecindades ya sea Moore o Von Neumann. Y junto a este controlador se le añadió una función para que pueda recibir un arreglo de coordenadas con las

cuales puede seguir una trayectoria dada ya sea por el algoritmo de Dijkstra o el algoritmo de A*.

3.5 Implementación final con el simulador Gazebo

Se implementó una integración final de todas las tareas que realiza el robot por medio de un nodo de ROS desarrollado en Python. El nodo se encarga de recibir el mapa, encontrar una ruta y después utilizar el controlador para poder desplazar al robot en el simulador. Para cada mapa se crearon sus respectivos archivos y nodos en los cuales se especifica que mapa se va a utilizar, que algoritmo se va a utilizar y de que punto a que punto tiene que desplazarse el Turtlebot. Se le agrego al nodo una interfaz gráfica que muestra la ruta a seguir del Turtlebot para mostrar el comportamiento gráfico de ambos algoritmos. En la Figura 7, 8, 9 y 10 se muestran los cuatro mapas que se diseñaron.

Figura 7. Mapa 1

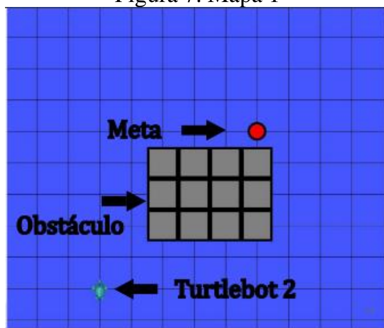


Figura 8. Mapa 2

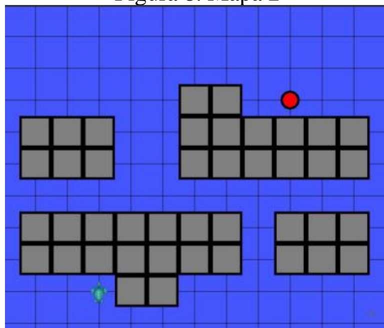


Figura 9. Mapa 3

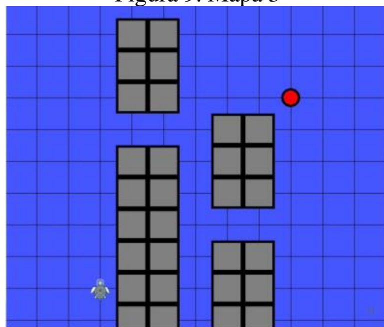
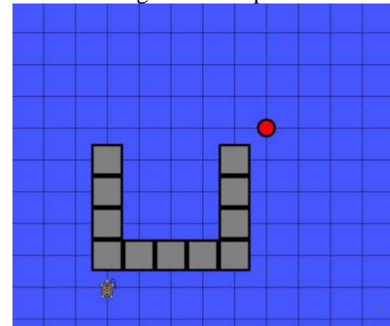


Figura 10. Mapa 4



4. RESULTADOS

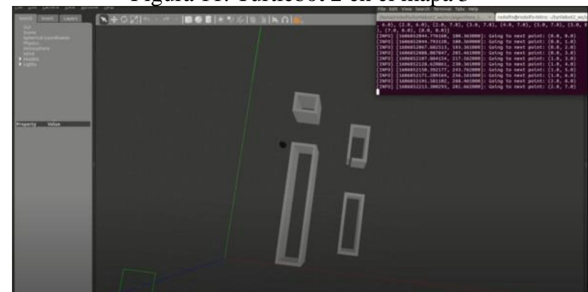
En la Tabla 1 se presentan los tiempos de ejecución de los algoritmos empleados al momento de realizar la implementación en los distintos espacios de trabajo tridimensionales.

Tabla 1. Tiempo de ejecución de los algoritmos en cada mapa

Mapa / Tiempo	Dijkstra	A*
Mapa 1	3’ 01”	2’ 44”
Mapa 2	6’ 34”	6’ 26”
Mapa 3	5’ 48”	5’ 35”
Mapa 4	5’ 09”	4’ 57”

Tal como se puede apreciar en la tabla anterior, se puede identificar que, en relación al tiempo transcurrido, el algoritmo que presento mejores resultados fue el algoritmo de A* presentando un tiempo menor en cada uno de los mapas probados. La Figura 11 muestra la simulación del Turtlebot 2 siguiendo una ruta en el mapa 3 dentro de Gazebo.

Figura 11. Turtlebot 2 en el mapa 3



5. CONCLUSIONES

La implementación de este trabajo se realizó utilizando el *framework* de ROS, que es una de las tecnologías actuales más importantes en el campo de la investigación y el desarrollo de la robótica.

Los resultados que se obtuvieron de las pruebas representan que la implementación del algoritmo A* fué más eficiente y es debido a la naturaleza de su comportamiento en este tipo de problemas que lo hace un método ligeramente superior. Esto no significa que Dijkstra sea un algoritmo inferior, sin embargo, debido a la naturaleza del problema en sí, hace que A* sea más adecuado gracias su función de costo sobre que nodos explorar sin utilizar recursos adicionales explorando nodos innecesarios.

La plataforma de ROS tiene una curva de aprendizaje bastante inclinada debido a que es una forma muy distinta de desarrollar un sistema robótico, sin embargo, debido que cuenta con varias herramientas listas para el desarrollo, ocasiona que tareas complejas de realizar, se vuelvan mucho más sencillas de implementar en ROS.

Esta implementación motiva la investigación de distintas aplicaciones de robótica, ya que se están desarrollando en un sistema el cual puede funcionar en la vida real y mostrar resultados fuera de una simulación. La robótica es un campo el cual ha estado tomando mayor importancia en el área tecnológica gracias al desarrollo avanzado de la computación y sus distintas aplicaciones en este campo por lo que es importante seguir demostrando e implementando distintas alternativas para solucionar problemas en este campo.

REFERENCIAS

[1] U. Orozco-Rosas, O. Montiel y R. Sepúlveda, «Parallel Evolutionary Artificial Potential Field for Path Planning—An Implementation on GPU,» de *Design of Intelligent Systems Based on Fuzzy Logic, Neural Networks and Nature-Inspired Optimization*, Studies in Computational Intelligence, vol 601, Springer, 2015.

[2] U. Orozco-Rosas, K. Picos, J. J. Pantrigo, A. S. Montemayor y A. Cuesta-Infante, «Mobile robot path planning using a QAPF learning algorithm for known and unknown environments,» *IEEE Access,* vol. 9, nº 1, pp. 101217-101238, 2022.

[3] ROS, «Core Components,» [En línea]. Available: https://www.ros.org/core-components/.

[4] F. Núñez Enríquez, *Control de movimiento empleando Labview, un enfoque didáctico,* Cholula, Puebla, México: Universidad de las Américas Puebla, 2007.

[5] U. Orozco-Rosas, K. Picos y O. Montiel, «Hybrid path planning algorithm based on membrane pseudo-bacterial potential field for autonomous mobile robots,» *IEEE Access,* vol. 7, nº 1, pp. 156787-156803, 2019.

[6] M. F. Rodríguez Borja y S. D. Sandobalin Guamán, *Diseño y Construcción de Tres Mini Robots Exploradores Cooperativos,* Quito, Ecuador: Escuela Politécnica Nacional, 2013.

[7] T. Olvera, U. Orozco-Rosas y K. Picos, «Mapping and navigation in an unknown environment using LiDAR for mobile service robots,» de *Proceedings of SPIE - The International Society for Optical Engineering*, San Diego, Californi, 2020.

[8] L. R. Macias, U. Orozco-Rosas y K. Picos, «Simultaneous localization and mapping using an RGB-D camera for autonomous mobile robot navigation,» de *Proceedings of SPIE - The International Society for Optical Engineering*, San Diego, California, 2021.

[9] U. Orozco-Rosas, K. Picos, O. Montiel y O. Castillo, «Environment Recognition for Path Generation in Autonomous Mobile Robots,» de *Hybrid Intelligent Systems in Control, Pattern Recognition and Medicine*, Studies in Computational Intelligence, vol 827, Springer, 2020.

[10] J. M. Armingol Moreno, *Localización geométrica de robots móviles autónomos,* Madrid, España: Universidad Carlos III de Madrid, 1997.

[11] A. Filipescu, V. Minzu, B. Dumitrascu, A. Filipescu y E. Minca, «Trajectory-tracking and discrete-time sliding-mode control of wheeled mobile robots,» de *IEEE International Conference on Information and Automation*, Shenzhen, China, 2011.

[12] Gazebo, «Gazebo,» [En línea]. Available: https://gazebosim.org/home.

[13] Turtlebot, «Turtlebot 2,» [En línea]. Available: https://www.turtlebot.com/turtlebot2/.

[14] U. Orozco-Rosas, O. Montiel y R. Sepúlveda, «Mobile robot path planning using membrane evolutionary artificial potential field,» *Applied Soft Computing,* vol. 77, pp. 236-251, 2019.

[15] S. M. LaValle, Planning Algorithms, Cambridge University Press, 2006.