ISSN 1870-4069

Navegación de robots móviles utilizando algoritmos Bugs extendidos

Mario Serna H., Abraham Sánchez L., Martín Estrada A.,
Ma. Beatríz Bernábe L., Elberfeld E. Pérez G.

Benemérita Universidad Autónoma de Puebla, Facultad de Ciencias de la Computación,
Laboratorio MOVIS, Puebla, Pue., México
mario_sh95@hotmail.com, abraham.sanchez@correo.buap.mx,
mestrada@cs.buap.mx, beatriz.bernabe@gmail.com,
enrique5001@hotmail.com

Resumen. Los algoritmos de navegación le permiten al robot móvil tener un cierto grado de autonomía en una tarea que requiere de un desplazamiento a un punto específico, planificación punto a punto. En este trabajo se realizó un estudio comparativo de los algoritmos Bug0, Bug1, Bug2, DistBug, Intelligent-Bug, Intensity-based Bug y Tangent Bug. Los algoritmos fueron implementados y probados tanto en ambientes reales como simulados en el robot Pioneer P3-DX con las herramientas provistas por ROS (Robot Operating System). Se presenta un estudio comparativo del rendimiento de los algoritmos, tomando como referencia el tiempo de ejecución y el tamaño total de la trayectoria.

Palabras clave: navegación, algoritmos bug, robots móviles.

Navigation of Mobile Robots Using Extended Bugs Algorithms

Abstract. The navigation algorithms allow the mobile robot to have a certain degree of autonomy in a task that requires a displacement to a specific point, i.e., point-to-point planning. In this work a comparative study of the Bug0, Bug1, Bug2, DistBug, Intelligent-Bug, Intensity-based Bug and Tangent Bug algorithms was carried out. The algorithms were implemented and tested in real and simulated environments in the Pioneer P3-DX robot with the tools provided by ROS. A comparative study of the performance of the algorithms is presented, taking as a reference the execution time and the total size of the trajectory.

Keywords: navigation, bug algorithms, mobile robots.

1. Introducción

Uno de los aspectos más relevantes en la robótica móvil es la navegación. Un robot móvil debe resolver en todo instante las preguntas: ¿Dónde estoy?, ¿Dónde voy? y ¿Cómo llego? La primera pregunta se resuelve con los sensores instalados en el robot y/o con sensores

ubicados en el ambiente de operación [1,2]. Existen muchas técnicas para hacerlo, como utilizar sensores isométricos, sensores de proximidad, cámaras de video o marca en el camino. La segunda pregunta se responde con una estrategia global de navegación y la tercera pregunta se responde con una estrategia de navegación local, que puede corresponder a una acción puramente reactiva o incorporar capacidades deliberativas que optimicen la evasión de obstáculos.

Para una navegación, con robots móviles, segura y efectiva es necesario un algoritmo de planificación de movimientos eficiente debido a que la calidad de la trayectoria afecta de gran manera a la aplicación robótica. Comúnmente, el principal objetivo de esto ha sido minimizar la distancia recorrida en el proceso de la navegación influenciando de esta forma a otras métricas como el tiempo de procesamiento y el consumo de energía. El problema de la planificación de movimientos puede ser clasificado de acuerdo con la naturaleza del entorno, el conocimiento del robot sobre este y el enfoque usado para resolverlo, es decir, de acuerdo con la naturaleza del entorno, conocimiento del mapa y completitud.

2. Planificación de movimientos con información local y algoritmos Bug

La planificación de movimientos puede ser divida de acuerdo con la información que el robot posee de su entorno. La planificación de movimientos con información local, o navegación reactiva, se destaca por la suposición de que la información sobre el espacio de trabajo es incompleta, es decir, la localización de los obstáculos se desconoce. El robot necesita contar con sensores (contacto, ultrasónico, láser, etc.) que le permitan la detección de los obstáculos y de esta forma evitar su colisión mientras avanza hacia el objetivo. En las primeras propuestas de este método [1], se considera al robot como un punto que se mueve en una escena bidimensional, en la cual existen obstáculos desconocidos con una forma arbitraria y de tamaño finito. El robot cuenta con información limitada a sus coordenadas actuales y a las coordenadas de la posición de objetivo. Para poder detectar los obstáculos en su camino, el robot hace uso de un sensor de contacto. Con esta información es suficiente para que el robot determine si es capaz de llegar a la posición de objetivo o en caso contrario, reportar que dicha posición no es alcanzable. Para lograr lo anterior, se definen dos comportamientos básicos que debe realizar el robot dependiendo de la situación que se presente:

1. Avanzar con dirección a la posición de objetivo.
2. Seguir la frontera del obstáculo encontrado hasta que una condición de abandono se cumpla.

El robot inicia la navegación en el ambiente ejecutando el primer comportamiento. Este comportamiento le permite llegar robot desde su posición inicial hasta la posición objetivo siempre y cuando no se encuentre obstáculos que le impidan continuar con su recorrido. Cuando el robot, por medio del sensor de contacto, detecta que hay un obstáculo entre él y la posición de objetivo procede a cambiar su comportamiento actual por el de seguir la frontera de dicho obstáculo encontrado. La condición de abandono se encarga de decidir cuando el robot tiene que dejar de seguir el obstáculo actual y retomar su camino con dirección a la posición objetivo. Los algoritmos Bug se propusieron por varios investigadores desde los años 80s para resolver el problema de la planificación de

movimientos. Estos algoritmos están enfocados en el sensado minimalista, es decir, necesitan poca información para poder dotar a los robots móviles de la capacidad de navegar en entornos con obstáculos.

La familia de algoritmos Bug se caracteriza por tener dos modos principales de movimiento: seguir la frontera de los obstáculos y moverse hacia la meta. A partir de estos dos modos se han derivado varias propuestas en las que se incluyen mejoras para poder reducir la distancia recorrida por el robot o disminuir la cantidad de información necesaria para lograr la meta principal, que es, ir de un punto inicial a otro punto en el entorno. En [3] se resume la evolución de los algoritmos de la familia Bug, en [4] se presenta una comparación de su rendimiento y en [5] se realiza un estudio más a fondo de estos algoritmos. Para este trabajo se consideraron los algoritmos Bug1, Bug2, DistBug y TangentBug por ser los más representativos en la familia Bug. Además de los anteriores, se incluyen los algoritmos:

- Bug 0, o también conocido como Com [5], el cual es usado para resaltar por qué es importante definir una condición de abandono para asegurar la terminación de los algoritmos.
- Intelligent-Bug [6], el cual es una versión orientada a la meta del algoritmo DistBug.
- I-Bug (Intensity-Based-Bug) [7], es una propuesta que utiliza la intensidad de una señal para permitirle al robot navegar sin necesidad de saber su localización en el entorno.

Las notaciones utilizadas en los en los trabajos presentados por los investigadores suelen ser similares, pero en algunos casos esto no es así, por tal motivo, se utiliza la siguiente notación para generalizar las descripciones y propiedades de los algoritmos Bug. La siguiente sección presenta a detalle algunos aspectos generales de los algoritmos BUG.

- X – la posición actual del robot.
- S – la posición inicial del robot.
- G – la posición meta, también llamada objetivo o final.
- H_i – el i-ésimo punto de contacto. Cada vez que el robot encuentra un obstáculo en su trayectoria y pasa del modo "moverse hacia la meta" al modo "seguir la frontera del obstáculo".
- L_i – el i-ésimo punto de abandono. Cuando se cumple alguna condición para dejar de seguir la frontera del obstáculo y pasar al modo "moverse hacia la meta".
- $d(a, b)$ – la distancia Euclidiana entre dos puntos arbitrarios a y b.
- $d_{trayectoria}(a, b)$ – el tamaño de la trayectoria entre dos puntos arbitrarios a y b.
- R– el rango máximo de detección del sensor.
- $r(\theta)$ – la distancia proporcionada por un sensor de distancia, en dirección de θ.
- F – el espacio libre en la dirección de la meta. Cuando θ es igual a la dirección de la meta, $F = r(\theta)$

3. Algoritmos Bug en detalle

Algoritmos Bug0, Bug1 y Bug2. En [8] se presentaron los algoritmos Bug1 y Bug2 como estrategias para la planificación de trayectorias de robots móviles en entornos con

obstáculos desconocidos. Estos algoritmos utilizan dos modos simples de movimiento para lograr su objetivo: moverse hacia la meta y seguir la frontera de los obstáculos. El robot tiene conocimiento de su posición X (coordenadas x, y) y su orientación (ángulo θ) en el entorno, así como de la localización del punto al que debe dirigirse G. Además, cuenta con un sensor de contacto que le permite detectar la presencia de un obstáculo. Esta es toda la información que necesita el algoritmo para funcionar.

El algoritmo Bug1 parte de la idea de moverse en línea recta desde S hacia G hasta que encuentre un obstáculo i. Cuando el obstáculo es detectado se define H_i, el robot ejecutará la rutina de seguir el borde hasta completarlo, mientras realiza esto, se calcula el punto L_i más cercano a la meta y se almacena temporalmente. Una vez finalizado el seguimiento del borde, el robot se dirige al punto Li almacenado siguiendo nuevamente el borde. Al posicionarse en dicho punto, el robot cambia al modo de moverse hacia la meta. Este procedimiento se repite si un nuevo obstáculo es encontrado hasta que el robot llegue a G o se complete de rodear la frontera del obstáculo actual, es decir, el punto H_i es encontrado nuevamente.

El algoritmo Bug2 funciona de forma similar que el algoritmo Bug1, pero presenta algunas variaciones. La trayectoria que se genera desde S hasta G se basa en la idea de seguir una línea recta imaginaria, llamada línea-m, entre estas dos posiciones. Si un obstáculo es encontrado se define H_i, el robot seguirá la frontera del obstáculo hasta encontrarse nuevamente sobre la línea-m. Para esto se define temporalmente un punto P, que representa a X sobre la línea-m, y se comprueba que $d(P, G)$ sea menor a $d(H_i, G)$. De ser esto así, se define L_i y se continua el trayecto cambiando al modo moverse hacia la meta; en caso contrario, se ignora la línea-m hasta que la condición anterior se cumpla [1].

El algoritmo Bug0 [5], también llamado Com, no pertenece oficialmente a la familia de algoritmos, pero es utilizado como base para el desarrollo de los algoritmos Bug. Esto se debe a que es una variación de los algoritmos originales anteriormente descritos y tiene como propósito ilustrar qué sucede si no se define una condición de abandono adecuada que asegure la terminación del algoritmo.

Algoritmo DistBug. El algoritmo DistBug [9], utiliza un sensor de distancia. La idea general de este algoritmo parte de lo siguiente: el sensor de distancia que proporciona los datos tiene una detección máxima cuyo rango es R. Los dos modos básicos de movimiento se mantienen. Inicialmente, el robot se mueve a hacia G hasta que se encuentra con un obstáculo. En esa posición se define H_i y se cambia al modo de seguir la frontera del obstáculo. Durante este modo, se registra la distancia mínima hacia la meta, denominada $d_{min}(G)$, alcanzada desde el punto H_i y se censa F a partir de las lecturas del sensor de distancia. F es la distancia en el espacio libre desde X hasta el obstáculo más cercano con dirección a G, si no se detecta un obstáculo entonces F es igual a $r(\theta)$. El robot deja de seguir la frontera del obstáculo sólo cuando el camino a G está despejado o la ecuación, $d(X, G) - F \leq d_{\min}(G) - Paso$, es satisfecha, donde $d(X, G)$ es la distancia de X a G y Paso es una constante predefinida. La variable Paso es un factor muy importante a la hora de decidir cuándo abandonar la frontera del obstáculo. Kamon y Rivlin asumen dos casos particulares para la elección del valor de esta variable. En el primer caso se asume el conocimiento de la distancia mínima entre los obstáculos, definida como M. El valor de Paso es establecido de acuerdo con el valor mínimo entre M y R.

En el segundo caso se asume que M es desconocida, por lo tanto, establecer un valor muy grande a Paso puede ocasionar que el robot no pueda dejar de seguir la frontera del

obstáculo y mucho menos alcanzar la meta. Para solucionar esto, los autores propusieron una versión modificada de la condición de abandono anterior donde se incorpora la condición del algoritmo Bug2 y es definida como CROSS (cruzar la línea, del inglés, line crossing). Mediante una relación booleana "O" se define la nueva condición de abandono de la siguiente manera: sea $C1 = CROSS$ y $d(X,G) < d(H_i,G)$, $C2 = d(X,G) - F \leq d_{min}(G) - Paso$, tal que, C1 o C2 se cumpla.

Algoritmo 1: DistBug con conocimiento a priori del entorno.

Entrada: Un robot en forma de punto con un sensor de distancia
Salida: Una trayectoria a G o la conclusión de que tal trayectoria no existe.

1. $L_0 \leftarrow S; Paso \leftarrow \min(M,R); i \leftarrow 1$
2. **mientras** Verdadero **hacer**
3. **repetir** moverse en línea recta desde L_{i-1} hacia G
4. **hasta** G es alcanzada o un obstáculo es encontrado en H_i
5. **si** G es alcanzada entonces salir con éxito
6. **repetir** seguir la frontera
7. Almacenar la distancia mínima hacia G $d_{\min}(G)$
8. Censar la distancia en el espacio libre F
9. **hasta**
10. (a) G es visible, $d(X,G) - F \leq 0$ o
11. (b) La condición de abandono basada en el rango se cumple, $d(X,G) - F \leq d_{min}(G) - Paso$ o
12. **si** G es alcanzada **entonces** salir con éxito
13. **sino** H_i es reencontrado **entonces** concluir que G no es alcanzable y salir con fallo
14. **sino** $L_i \leftarrow X; i \leftarrow i + 1$

El algoritmo Intelligent-Bug (IBA [11] es una versión orientada hacia la meta del algoritmo DistBug. En [12] se resalta que el algoritmo DistBug considera el costo de la trayectoria para la toma de decisiones, lo cual puede provocar que el robot se aleje de la meta aún en presencia de un camino libre hacia esta. Lo anterior es tomado como referencia para la mejora de este algoritmo. El procedimiento que sigue este algoritmo es el mismo que el presentado para DistBug. La diferencia radica en que el punto de abandono L es seleccionado con base en el camino libre hacia la meta.

Algoritmo I-Bug. Los algoritmos Bug, anteriormente citados, persiguen un enfoque donde se trata de reducir la trayectoria que sigue un robot desde una posición S hasta una posición G. Mientras que el algoritmo I-Bug [7] parte de la idea de reducir la cantidad de información que necesita el robot para navegar exitosamente hacia una fuente de intensidad en un entorno desconocido. El robot usa un sensor de contacto, un sensor de intensidad y un sensor de alineamiento para lograr la tarea de alcanzar la meta, la cual representa a la fuente de intensidad. El robot no tiene acceso a ninguna información que le permita inferir cualquier coordenada, su orientación o la distancia total recorrida. Un punto llamado torre existe en algún lugar del entorno. La torre emite una señal que es modelada como una función de intensidad. Además, se asume que la intensidad máxima es 1, cuando la torre es alcanzada. La localización de la torre es irrelevante para realizar los cálculos porque cualquier coordenada puede ser trasladada a la posición actual sin cambiar los valores de intensidad. Por lo tanto, se asume que la localización de la torre es (0, 0). Una restricción importante es

que únicamente se permite un máximo local, que es la torre. Las señales emitidas por la torre tienen una forma de círculos concéntricos para el caso de intensidad radialmente simétrica. La intensidad decae cuadráticamente con la distancia, sin importar la dirección. Si los conjuntos de nivel no son círculos concéntricos, entonces este caso es llamado asimétrico.

Anteriormente se hacía mención de los sensores con los que cuenta el robot, a continuación, se da una breve explicación de cada uno:

- Sensor de contacto: indica si el robot está tocando el borde del entorno.
- Sensor de intensidad: indica la intensidad de la señal emitida por la torre relativa a la posición del robot.
- Sensor de alineación: indica si el robot está mirando hacia la torre.

Generalmente para los algoritmos Bug se definen dos modos de movimiento: moverse hacia la meta y seguir la frontera del obstáculo. Sin embargo, para este algoritmo se agrega otro modo: alinearse a la torre. Para cualquier otro algoritmo Bug, este modo se puede ver como una subrutina del modo moverse hacia la meta, debido a que el robot conoce en todo momento su posición, su orientación y la posición de la meta. Por tal motivo, no es necesario utilizar un sensor externo para realizar esta tarea y se asume que el robot tiene que alinearse hacia la meta antes de dirigirse hacia ella, cosa que es diferente en este caso. Dicho lo anterior, a continuación, se describen los tres modos de movimiento adaptados para este algoritmo:

- Moverse en línea recta. Sólo se detiene si hace contacto con algún obstáculo, alcanza la torre o detecta un máximo local a lo largo de su línea de movimiento.
- Alinearse a la torre. El robot rota en dirección contraria a las manecillas del reloj, deteniéndose cuando esté alineado con la torre.
- Seguir la frontera del obstáculo. El robot mantiene contacto con la frontera del obstáculo a su izquierda todo el tiempo, sólo deteniéndose si alcanza un máximo local en la intensidad obtenida desde la torre.

Antes de mencionar el proceso de funcionamiento para este algoritmo se definen las siguientes intensidades:

- i_X es la intensidad actual recibida por el robot.
- i_H es la intensidad observada cuando el robot hace contacto con el obstáculo y cambia al modo de seguir la frontera.
- i_L es la intensidad obtenida justo antes de cambiar al modo moverse en línea recta.

De acuerdo con lo anteriormente mencionado, el procedimiento general del algoritmo es el siguiente: inicialmente se asigna a i_L el valor de i_X. Seguido de esto se ejecuta el modo de alinearse a la torre hasta que el sensor de alineación reciba la señal de que el robot está orientado hacia la torre. Se cambia al modo de moverse en línea recta hasta que i_X sea igual a 1, esto quiere decir que el robot alcanzó la torre, o hasta que el sensor de contacto detecte un obstáculo bloqueando el paso. Cuando el sensor de contacto detecta dicho obstáculo se asigna a i_H el valor de la intensidad obtenido en ese momento desde la torre y se cambia al modo de seguir la frontera del obstáculo. Mientras el robot está ejecutando este modo se comprueba que i_X sea igual a 1, indicando que se llegó a la torre, y en caso de que esto no ocurra, se busca detectar un máximo local mediante el muestreo de las tres últimas

intensidades de i_X de tal forma que $i_{X-2} > i_{X-1} \geq i_X$. Si se detecta este cambio en las intensidades, entonces se repite nuevamente el proceso desde el inicio.

Algoritmo TangentBug. El algoritmo TangentBug [10], al igual que el algoritmo DistBug, utiliza un sensor de distancia para permitirle al robot navegar en el entorno de una forma más ágil y recorriendo una trayectoria más corta. En la propuesta original se propone el uso de un grafo que representa los alrededores del robot para minimizar el tamaño de la trayectoria mediante una búsqueda realizada en él. Este grafo es llamado Grafo Tangente Local (LTG, Local Tangent Graph), el cual es una versión que utiliza la información local disponible para ser construido, en comparación a su contraparte global llamada Grafo Tangente (TG, Tangent Graph). El grafo LTG es generado a partir de la información obtenida del sensor de distancia, representada por la función $\boldsymbol{r}(\boldsymbol{\theta})$ que devuelve la distancia en dirección de θ desde la posición X al punto más cercano de un obstáculo. Al procesar $\boldsymbol{r}(\boldsymbol{\theta})$ se tienen en cuenta los siguientes casos:

- Se crea un nodo, llamado nodo-T, cuando G es visible o no hay obstáculos visibles en dirección de G, para esto se comprueba que $d(X,G) - F \leq 0$ o $F \geq R$, respectivamente.
- Se crea un nodo si es detectada una discontinuidad en la dirección de θ.

Se crea un nodo si $r(\theta) = R$ y $r(\theta)$ comienza a disminuir subsecuentemente. De la misma forma, si $r(\theta) \neq R$ y $r(\theta)$ comienza a incrementar tal que $r(\theta) = R$.

Al tener los nodos identificados, se determina cuál de estos tiene la distancia y dirección óptima hacia G y es seleccionado como el objetivo temporal mientras el robot se mueve en dirección a él y actualiza continuamente el grafo tangente local en busca de un nuevo nodo más cercano. Regresando al proceso que sigue el algoritmo, el robot sólo cambia al modo de seguir la frontera del obstáculo cuando detecta un mínimo local. Mientras se encuentra en este modo, se actualizan dos variables que permiten determinar la condición de abandono. $d_{seguimiento}$ registra la distancia mínima hacia G lograda por el robot durante este modo. Mientras $d_{alcanzada}$ es la distancia mínima desde un punto m, que se encuentra dentro del rango de detección, hasta G. Cuando $d_{alcanzada} < d_{seguimiento}$ el robot retoma su trayectoria hacia G. Si el robot completa un ciclo mientras recorre la frontera del obstáculo y encuentra nuevamente el punto m, se determina que la meta no es alcanzable. Simplificando el proceso del algoritmo, utilizando la forma heurística en lugar del grafo tangente local, puede ser expresado de la siguiente manera:

Algoritmo 2: TangentBug

Entrada: Un robot en forma de punto con un sensor de distancia
Salida: Una trayectoria a G o la conclusión de que tal trayectoria no existe.

1. **mientras** Siempre **hacer**
2. **repetir**
3. Tomar lecturas del sensor de distancia y determinar los puntos de discontinuidad
4. Moverse hacia el punto $p \in \{G, P1, P2, \dots\}$ que minimiza $h(X,p) = d(X,p) + d(p,G)$
5. **hasta**
   (a) G es alcanzada o
   (b) El valor que minimiza $h(X,p)$ empieza a incrementar $d(X,G)$, es decir, se detecta un "mínimo local"

6. Seguir la frontera eligiendo la dirección que continúe en la misma dirección del más reciente moverse hacia la meta
7. **repetir** actualizar continuamente $d_{alcanzada}$ y $d_{seguimiento}$
8. **hasta**
   (a) G es alcanzada
   (b) El robot completa un ciclo alrededor del obstáculo, determinar que G es no alcanzable
   (c) $d_{alcanzada} < d_{seguimiento}$

El rango máximo del sensor de distancia es un factor muy importante para este algoritmo. Debido a que puede afectar en el comportamiento del robot a la hora de elegir la trayectoria a seguir, causando que el tamaño final del recorrido aumente o disminuya.

4. Resultados experimentales

En esta sección presentamos los resultados experimentales, en simulación y en el robot móvil Pioneer P3-DX. Las herramientas que se utilizaron fueron: el sistema operativo ROS (Robot Operating System), las librerias Gazebo, el visualizador de datos rviz, y un sensor láser Hokuyo modelo URG 04LX-UG01.

Con el objetivo de tener más control a la hora de ejecutar los algoritmos se optó por una estructura que implementa la lógica del modelo cliente-servidor. Mediante el uso de las características de ROS se crearon los nodos correspondientes y se agruparon como se muestra en la Figura 1. En la parte superior se encuentra el nodo llamado Servidor Bug responsable del control de los nodos de los algoritmos. Para esto se utilizan los servicios de ROS, permitiendo así que un cliente externo envíe una petición para la ejecución de algún algoritmo en concreto o para cambiar su estado actual. El nodo Servidor Bug recibe estas peticiones y se encarga de validar los parámetros enviados en ellas. Después, envía un mensaje al nodo correspondiente para modificar su estado.

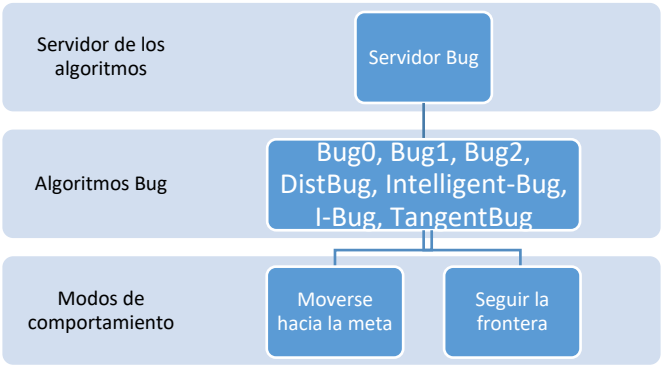

Fig. 1. Diagrama general de la propuesta para el control de los algoritmos.

Para que el nodo del Servidor Bug conozca el estado de los nodos de los algoritmos, se publican mensajes relacionados al estado interno de cada nodo en un tema de ROS. Los publicadores de estos mensajes sólo tienen permitido publicar el estado de su nodo

correspondiente cuando se produce un cambio en el estado del algoritmo, por ejemplo, cuando se pasa del modo de comportamiento "moverse hacia la meta" al modo de "seguir la frontera del obstáculo".

De esta forma se evita que los nodos que se encuentran en el estado de en espera sigan publicando su estado mientras otro nodo se encuentre en funcionamiento. El nodo Servidor Bug se encuentra suscrito a este tema y cada vez que se publica un nuevo mensaje se actualiza el estado global con la información del estado interno. El estado global es usado por el nodo Servidor Bug para comunicar de forma constante cada segundo qué es los que está ocurriendo a los clientes externos.

De la misma forma en que el Servidor Bug se comunica con los nodos de los algoritmos, estos se comunican con los nodos de los modos de comportamiento. Esto ocurre cuando hay un cambio en el estado del algoritmo, indicando que es necesario utilizar otro modo de comportamiento en ese preciso momento. Existieron otros pequeños inconvenientes para implementar los algoritmos, que se resolvieron con éxito y se pueden consultar con detalle en [13].

Las pruebas de los algoritmos se realizaron en el simulador Gazebo y para esto se construyeron algunos mapas diferentes (ver la Figura 2), haciendo uso del editor de construcciones. Esta herramienta permitió crear distintos tipos de obstáculos para su posterior uso en los mapas. La Tabla 1 presenta un resumen de la ejecución de los algoritmos en el mapa denominado "normal".

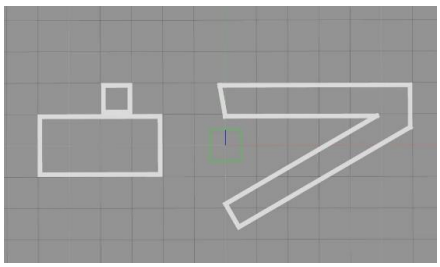

Fig. 2. Mapa tipo "normal".

Tabla 1. Resumen de la ejecución de los algoritmos implementados para el mapa tipo normal.

Algoritmo	Trayectoria final (Metros)	Tiempo (Segundos)	Moverse hacia la meta (seg)	Seguir la frontera del obstáculo (seg)
Bug0	21.041	81.927	22.276	59.651
Bug0 (E, R)	25.683	104.260	33.811	70.449
Bug1	82.382	318.491	21.791	296.700
Bug2	33.183	134.785	27.533	107.252
Bug2 (E)	33.124	135.502	27.603	107.899
Bug2 (R)	26.783	115.196	25.946	89.250
Bug2 (E, R)	26.983	116.751	25.602	91.149
DistBug	19.941	77.016	21.615	55.401
DistBug (E)	30.942	117.276	29.612	87.664
DistBug (R)	20.100	77.265	21.865	55.400
DistBug (E, R)	25.841	103.107	35.407	67.700
I-Bug	21.300	136.611	60.860	75.751
TangentBug	18.783	68.191	60.440	7.751

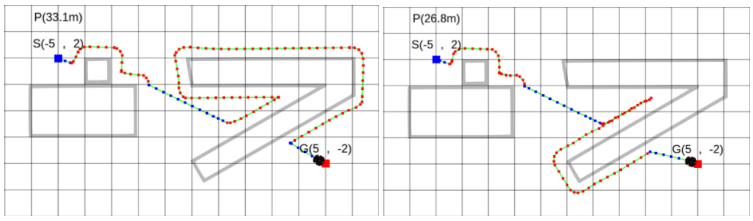

Fig. 3. Bug2 (E) y Bug2 (R) en el mapa de tipo normal.

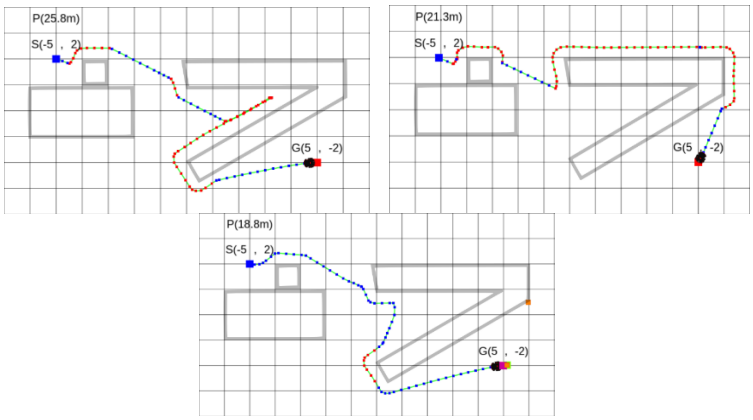

Fig. 4. DistBug (E, R), I-Bug y TangentBug en el mapa de tipo normal.

Fig. 5. Mapa del entorno real creado con el paquete Gmapping de ROS.

Las figuras 3 y 4 muestran las simulaciones realizadas con los diferentes algoritmos.

Para la realización de las pruebas con el robot real se adecuó el espacio en el laboratorio de Movis de la FCC-BUAP, creando de esta forma el entorno que se aprecia en la figura 5.

Fig. 6. Bug1 y Bug2 en el mapa del laboratorio de Movis.

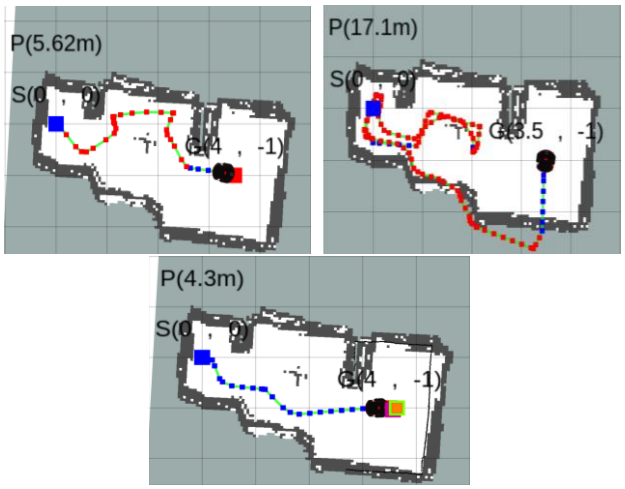

Fig.7. DistBug (E,R), I-Bug y TangentBug en el mapa del laboratorio de Movis.

Tabla 2. Resumen de la ejecución de los algoritmos implementados para el mapa real.

Algoritmo	Trayectoria final (mts)	Tiempo (seg)	Moverse hacia la meta (seg)	Seguir la frontera del obstáculo (seg)
Bug0	-	-	-	-
Bug0 (E)	5.466	55.637	22.587	33.050
Bug0 (R)	6.007	79.459	25.359	54.100
Bug0 (E, R)	5.683	54.238	23.588	30.650
Bug1	22.531	225.255	6.605	218.650
Bug2	10.083	108.254	7.754	100.500
Bug2 (E)	4.241	30.003	13.753	16.250
Bug2 (R)	6.683	92.060	13.059	79.001
Bug2 (E, R)	4.224	30.532	13.782	16.750
DistBug	13.900	152.188	20.288	131.900
DistBug (E)	3.920	29.130	14.988	14.149
DistBug (R)	6.600	84.140	12.640	71.500
DistBug (E, R)	5.620	41.850	5.050	36.800
I-Bug	17.107	227.307	80.675	146.650
TangentBug	4.300	32.080	32.080	0.000

Igualmente, las figuras 6 y 7 muestran las ejecuciones de los algoritmos. La tabla 2 presenta los resultados en el ambiente real.

De acuerdo con los resultados presentados, se resaltan los siguientes puntos:

- DistBug (E) fue el algoritmo que generó la trayectoria más corta, seguido de los algoritmos Bug2 (E) y Bug2 (E, R). El algoritmo TangentBug también realizó un buen trabajo en este escenario.
- De igual forma, los algoritmos anteriores presentaron los menores tiempos de ejecución para llegar de la configuración inicial a la final.
- Bug1 fue el algoritmo que presentó el peor rendimiento en el tamaño de la trayectoria y I-Bug fue el que requirió del mayor tiempo de ejecución para completar la trayectoria.
- I-Bug no obtuvo un buen tiempo de ejecución debido a que necesita dar un giro casi completo sobre su eje de rotación para volver a estar orientado hacia la meta.
- En la Figura 7 arriba se puede observar cómo afectan las rotaciones que realiza el robot a su odometría, haciendo que poco a poco su sistema de coordenadas iniciales pierda precisión con respecto al entorno.
- El algoritmo Bug0 fue el único que no pudo alcanzar la meta.
- Los comportamientos de elegir (E) y revertir (R) la dirección, tuvieron un impacto positivo en el rendimiento de los algoritmos originales.
- A pesar de que el algoritmo TangentBug consiguió un buen resultado, durante su ejecución se pudo observar que presentaba algunas dificultades a la hora de elegir el punto de discontinuidad a seguir, provocando que su avance hacia la meta se viera afectado un poco (Figura 7 abajo).

5. Conclusiones y trabajo futuro

A pesar de que en la actualidad existen técnicas muy avanzadas que resuelven el problema de la planificación de movimientos, por ejemplo, algunas de ellas utilizan un mapa para generar la trayectoria óptima o lo crean mediante la exploración del entorno, los algoritmos Bug siguen siendo una opción viable para resolver este problema, ya que combinan la planificación local con un criterio de convergencia global. En algunos casos, lo que se busca es minimizar el tamaño de la trayectoria final y en otros, el de simplificar el algoritmo de tal forma que requiera la mínima información para funcionar correctamente. Los algoritmos considerados para este trabajo permiten observar el panorama general de este enfoque y pueden ser vistos como una buena forma de introducirse al área de la robótica, especialmente para entender las bases de la navegación en entornos poblados de obstáculos por parte de robots autónomos.

El objetivo principal de la realización de este trabajo fue el de desarrollar estrategias para la navegación de robots móviles utilizando el enfoque de los algoritmos pertenecientes a la familia Bug. Lo anterior fue conseguido de forma exitosa al implementar los algoritmos más emblemáticos de esta familia. A través de su simulación fue posible pulir aspectos para su posterior utilización en un entorno real usando como sujeto de pruebas al robot móvil Pioneer P3-DX. Trabajar con ROS facilitó en gran medida el desarrollo y la implementación de los algoritmos, permitiendo que se distribuyeran las diferentes tareas y comportamientos

haciendo uso de sus nodos. De esta forma se tuvo un mejor control sobre lo que estaba pasando mientras los algoritmos se estaban ejecutando.

Existen muchas ideas para el trabajo futuro, pero creemos que el refactorizar el código es de los más importantes. La forma en que se realizó la implementación perseguía los propósitos de comprobar que todo funcionara correctamente y que cada archivo de código fuera entendible por sí solo, pero esta forma no es eficiente cuando se requieren incorporar nuevas funcionalidades o realizar cambios, ya que es necesario modificar la mayoría de los archivos. Por tal motivo, se plantea que el control de los nodos sea implementado en una clase que posea todas las propiedades y características propuestas en este trabajo. De igual forma, la implementación de la clase para los algoritmos Bug. Esto facilitará la inclusión de nuevas mejoras y el desarrollo de otros algoritmos.

Referencias

1. Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., Thrun, S.: Principles of robot motion: Theory, algorithms, and implementations. A Bradford Book (2005)
2. Thrun, S., Burgard, W., Fox, D.: Probabilistic robotics. The MIT Press (2005)
3. Buniyamin, N., Wan Ngah, N.W., Mohamad, Z.: Points Bug versus TangentBug algorithm: A performance comparison in unknown static environment. In: IEEE Sensors Applications Symposium, pp. 278–282 (2014)
4. Yufka, A., Parlaktuna, O.: Performance comparison of the BUG's algorithms for mobile robots. In: INISTA 2009 International Symposium on INnovations in Intelligent SysTems and Applications, pp. 416–421 (2009)
5. Ng, J.: An analysis of mobile robot navigation algorithms in unknown environments. PHD Thesis University of Western Australia (2010)
6. Zohaib, M., Mustafa, P., S., Javaid, N., Iqbal, J.: IBA: Intelligent Bug algorithm – A novel strategy to navigate mobile robots autonomously. In: IMTIC 2013: Communication Technologies, Information Security and Sustainable Development, pp. 291–299 (2014)
7. Taylor, K., LaValle, S.: Intensity-based navigation with global guarantees. Autonomous Robots 36, pp. 349–364 (2014)
8. Lumelsky, V.J., Stepanov, A.A.: Path planning strategies for point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica 2, pp. 403–430 (1987)
9. Kamon, I., Rivlin, E.: Sensory-based motion planning with global proofs. IEEE Transactions on Robotics and Automation 13(6), 814–822 (1997)
10. Kamon, I., Rimon, E., Rivlin, E.: TangentBug: A range sensor-based navigation algorithm. The International Journal of Robotics Research (17)9, 934–953 (1998)
11. Zohaib, M., Iqbal, J., Mustafa Pasha, S.: A novel goal-oriented strategy for mobile robot navigation without sub-goals constraint. Revue Roumaine des Sciences Techniques - Serie Électrotechnique et Énergétique 63(1), 106–111 (2018)
12. Serna Hernández, M.: Navegación de robots móviles utilizando los algoritmos Bug extendidos. Tesis de Licenciatura – Facultad de Ciencias de la Computación – BUAP (2018)