uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Electrónica Industrial y Automática
2017-2018

Trabajo Fin de Grado

"Planificación de trayectorias para brazos robóticos basada en RRT"

Manuel Moral Navarro

Tutor

David Álvarez Sánchez

Leganés, 2018

RESUMEN

El objetivo general de este trabajo de fin de grado es la planificación de trayectorias para brazos robóticos. En concreto se usará el algoritmo RRT destinado a este objetivo, a partir de la herramienta MATLAB. En concreto se utilizará el manipulador MANFRED-2 desarrollado por el departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid.

Todo se llevará a cabo a partir de diversas toolboxes para MATLAB, encargadas de trasladar un brazo robótico a un entorno de simulación en el que poder realizar la planificación de una trayectoria desde un punto inicial a otro punto objetico, pudiendo existir entre ambos obstáculos los cuales deberá evitar el brazo.

Para ello se adaptará un algoritmo de planificación RRT a las condiciones dada por el brazo robótico, a partir de sus parámetros de Denavit-Hartenberg y la cinemática directa; se utilizarán diversos objetos a modo de obstáculos en el entorno del brazo; se harán uso de los modelados STL del manipulador o brazo robótico para una simulación más realista.

Finalmente, con el modelo obtenido se procedieron a realizar diversas simulaciones con el fin de obtener mayor conocimiento acerca de cómo el funcionamiento del sistema se ve afectado en función del valor de los distintos parámetros del mismo.

Palabras clave: Brazo robótico, planificación de trayectorias, RRT, Denavit-Hartenberg, obstáculos, cinemática directa.

ABSTRACT

The general objective of this end-of-degree project is the path planning for robotic arms. In particular, the RRT algorithm, intended for this goal, used from the MATLAB tool. Specifically, it will be used the MANFRED-2 manipulator, developed by the Department of Systems Engineering and Automation of the Carlos III University of Madrid.

Everything will be carried out from various toolboxes for MATLAB, responsible for moving a robotic arm to a simulation environment h from an initial point to another objective point, being able to exist between both obstacles that must be to avoid by the manipulator.

To do this, a RRT planning algorithm was adapted to the conditions belonging to the robotic arm, based on the parameters of Denavit-Hartenberg and the use of forward kinematics; it will be used objects as obstacles in the environment of the arm; STL models of the manipulator or robotic arm are used for a more realistic simulation.

Finally, with the model obtained, several simulations were carried out in order to obtain more knowledge about how the functioning of the system is affected depending on the value of the different parameters of the system.

Key wrods: Robotic arm, path planning, RRT, Denavit-Hartenberg, obstacles, forward kinematics.

AGRADECIMIENTOS

Me gustaría comenzar este documento agradeciendo a aquellos que me han acompañado y ayudado durante este viaje que me ha llevado a la elaboración de este trabajo y por consiguiente a la finalización de este grado. A mis padres, Manuel y Maria del Pilar; a mi pareja, Carla, por estar ahí brindándome el apoyo necesario en todo momento, y sin los cuales no habría sido posible, y a los que dedico este documento.

Agradecer a mis amigos y compañeros de grado la inestimable ayuda que me han dado durante todos estos años y que me han facilitado enormemente.

Mi agradecimiento también a todos mis profesores durante mi paso por la UC3M y en concreto a David, mi tutor, por ofrecerme la oportunidad de trabajar en este proyecto, estando siempre disponible y dispuesto a prestar su ayuda, orientándome en la realización del trabajo de fin de grado.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE TABLAS

1. INTRODUCCIÓN

La planificación de trayectorias consiste en encontrar un camino o una trayectoria para unir dos puntos, a saber, el punto de partida y el objetivo o posición final del robot. Durante dicha planificación, se tienen en cuenta las restricciones del robot, la evasión de las colisiones con obstáculos, etc. Para ello, se hace uso de alguno de los numerosos algoritmos de planificación existentes hasta la fecha.

El objetivo del estudio es obtener un método de planificación de trayectorias para el brazo robótico MANFRED-2, un manipulador robótico de 6 grados de libertad desarrollado por el departamento de robótica de la UC3M, mediante el uso de Matlab y distintas *toolboxes* para el mismo, principalmente la *toolbox* de Peter Corke *"Robotics toolbox"*.

Se busca crear una trayectoria que el brazo pueda seguir para alcanzar un punto final u objetivo que se encuentre en el entorno de acción del robot.

Esto presenta en principio la existencia de múltiples posiciones posibles del brazo o configuraciones para un solo punto alcanzable, definido por las tres coordenadas cartesianas (x, y, z). Deberá, por tanto, buscar entre ese conjunto de posibles configuraciones (posición y orientación de cada una de las articulaciones del robot), alguna de aquellas que satisfagan todas y cada una de las condiciones y restricciones del sistema, como por ejemplo la orientación del brazo a la hora de alcanzar el objeto.

Se usará el algoritmo de planificación RRT (Rapidly-exploring Random Tree o o Arboles aleatorios de exploración rápida), para el problema a resolver. Este algoritmo, a diferencia de otros, no se ve excesivamente afectado por el número de dimensiones o grados de libertad a utilizar [1]. Para nuestro caso particular, nos permite tomar los 6 grados de libertad del brazo y transformarlos en un sistema de 3 grados de libertad. En concreto, para una configuración del brazo dada por sus 6 grados de libertad, podemos operar con ellos y pasar a trabajar con un punto en 3 dimensiones o coordenadas cartesianas (x, y, z). Además, nos permite realizar la planificación mediante el uso de cinemática directa y la representación de Denavit-Hartenberg, permitiendo así, evitar la cinemática inversa y su gran complejidad. Otros algoritmos, como D* o cualquier algoritmo de cuadrícula [2], no nos permitirían el funcionamiento en "n" dimensiones, debido a la alta complejidad que añadirían al algoritmo, así como el tiempo que comprendería su utilización, no como el RRT, que destaca entre otros factores, por su velocidad de ejecución para cualquier número de dimensiones.

Otra ventaja es la posibilidad que aporta el uso de un algoritmo RRT, es la posibilidad de inclusión de obstáculos en el entorno de funcionamiento, así como la detección de colisiones con estos.

El robot MANFRED-2 se encuentra en un entorno en el cual existen diversos obstáculos, los cuales se deberán sortear durante el movimiento del brazo hacia el objetivo o punto final. Para la cuestión relacionada con las colisiones se utilizarán diversas *toolboxes* que nos permitirán realizar el correcto modelado del robot y del entorno, así como la detección de colisiones de este con el entorno.

Adicionalmente, finalizando el documento se expondrá en el **apartado 5: "Marco regulador"** de esta memoria, en el cual se analizará la legislación aplicable al documento y a los recursos utilizados para la elaboración del proyecto.

2. CONOCIMIENTOS PREVIOS

En los siguientes apartados, se expondrán distintos conocimientos y explicaciones que permitirán poner en situación el trabajo realizado. En ellos se abarcará el manipulador a utilizar, sustento teórico, así como distintos algoritmos de planificación.

2.1 MANFRED-2

El Robot MANFRED-2 es un manipulador móvil con 8 grados de libertad (DOF) compuesto por un brazo robótico y una base. Ha sido diseñado en la Universidad Carlos III de Madrid por el departamento de Ingeniería de Sistemas y Automática (*Robotics Lab*) y sus componentes/piezas fueron mecanizadas y montadas por una empresa externa española.

Su nombre corresponde con la abreviaturización de MANipulator FRiEnDly mobile manipulator, Manfred. En su caso nos encontramos en la segunda versión de este, la cual proviene del MANFRED original, creada con el objetivo de reducir el peso del brazo y mejorar su movilidad. Su proceso de creación comenzó en 2001 que duró en torno a 2 años hasta 2003. A partir de este momento los recursos se dedicaron a las cuestiones relacionadas con el aprendizaje y la localización del mismo. En cuanto a sus dimensiones, nos encontramos con un robot de 1.70m de altura y 120 kg de peso, únicamente 18kg correspondientes al brazo, siendo el peso restante correspondiente a baterías motores y la base, en definitiva.

FIGURA 1: MANFRED-2

El MANFRED-2 fue concebido para manipular objetos, agarrar objetos y poder moverse por entornos interiores. Para ello hace uso de una pinza en el extremo del brazo que se convertirá en el futuro en una mano mecánica y de un láser con el que consigue calcular las distancias a las distancias de los distintos objetos que se encuentran a su alrededor creando así un mapa del entorno. Además, se encuentra en un proceso de aprendizaje continuo en el que irá aprendiendo el entorno de alrededor, para poder elegir entre un gran número de opciones a los problemas que se le plantearan, dando así la mejor solución posible a estos. El objetivo final no es para uso industrial, sino puramente enfocado a la investigación.

2.2 Diferentes algoritmos de cálculo/planificación de trayectorias

En la actualidad existen diferentes formas de realizar una planificación de trayectoria. Estos métodos de planificación de trayectorias pueden englobarse o clasificarse de distintas formas al tener entre ellos puntos en común. Sin embargo, se ha decidido dividirlos en dos grandes grupos. El primer gran grupo es el que engloba a la planificación de trayectorias mediante métodos geométricos, y el segundo grupo aquel que se basa en métodos con grafos/esquemas y árboles [3]. A continuación, se procede a explicar algunos de estos métodos de planificación.

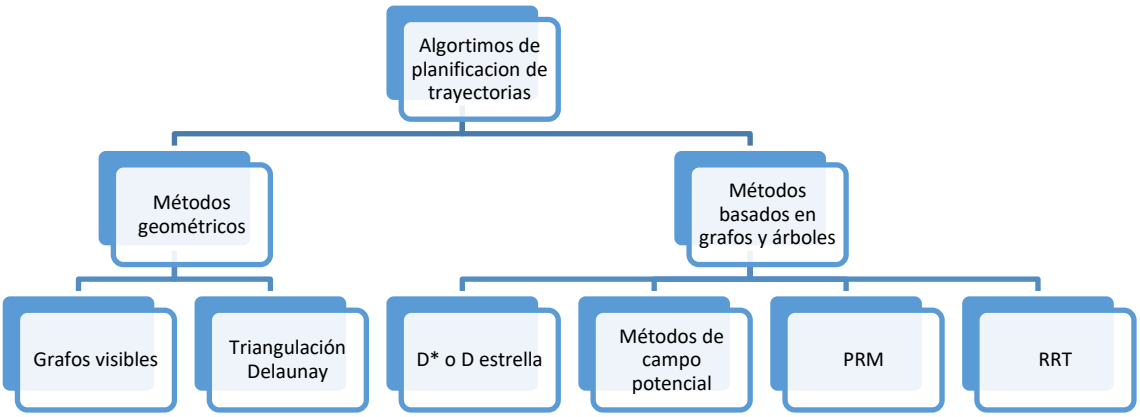

FIGURA 2 CLASIFICACIÓN ALGORITMOS DE PLANIFICACIÓN

2.2.1 Métodos geométricos

Estos métodos hacen uso de estructuras geométricas básicas para representar el entorno. Algunos se basan en el uso de líneas o segmentos para la representación del espacio, mientras que otros, más recientes, hacen uso de triángulos, para representar estructuras en tres dimensiones del entorno. Los más utilizados habitualmente son los métodos de *Grafos de Visibilidad* y *Triangulación Delaunay* [3].

- Grafos Visibles

En este método de planificación de trayectorias, se realiza una poligonización de los obstáculos del entorno. Con todos, los obstáculos poligonizados, se trazan segmentos que unen cada vértice de obstáculos, con todos los vértices de los demás obstáculos que tenga en visión directa y, por tanto, al trazar el segmento, éste no atraviese obstáculos. Una vez definido el entorno, se añaden el punto objetivo y el punto inicial, los cuales se unen con los vértices de los obstáculos, como se ha comentado, es decir, con aquellos vértices en visión directa [5].

De este modo, basta elegir la trayectoria más óptima, la cual permita unir los dos puntos (origen y destino), pudiendo esta trayectoria asemejarse a una curva simple.

Este método permite una forma intuitiva de realizar una planificación de trayectorias en dos dimensiones, sin embargo, se complicaría en el caso de añadir dimensiones adicionales, al no ser trivial y contener un alto coste computacional.

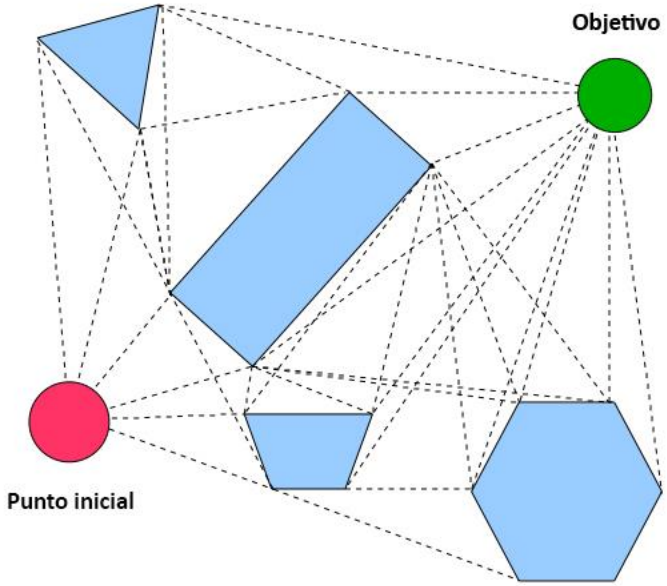

FIGURA 3: GRAFOS DE VISIBILIDAD **[17]**

En la imagen superior, se puede observar una representación del método basado en grafos de visibilidad. En él, se representan de color azul, los obstáculos del entorno; con líneas discontinuas, los segmentos que unen los vértices de estos obstáculos con otros vértices visibles de otros obstáculos o con los puntos objetivo y final.

A partir de este planteamiento se pueden encontrar diversos caminos o trayectorias que unan nuestros dos puntos.

- Triangulación Delaunay

Dado un entorno en el que se desea realizar una planificación de trayectorias, se definen por varios puntos distribuidos por el espacio a ocupar. La triangulación Delaunay se encarga de unir estos puntos, formando el mayor número de triángulos sin que se crucen sus aristas entre sí, de forma que todos los puntos del entorno acaben conectados los unos con los otros. En el proceso de construcción de la triangulación, los triángulos se definen de forma que se unan mediante una arista los puntos que estén más próximos entre sí, además de intentar que el triángulo sea lo más regular posible [5].

Adicionalmente, podría partirse del diagrama de Voronoi, que, para los mismos puntos usados para el entorno, en lugar de conectarlos, realiza una subdivisión del entorno, mediante regiones creadas por segmentos que equidistan de dos puntos del entorno [6].

A continuación, se muestra una representación de la triangulación Delaunay (segmentos con línea continua) junto con un diagrama de Voronoi (segmentos con línea discontinua). Se utilizan círculos para representar los nodos utilizados en la planificación.

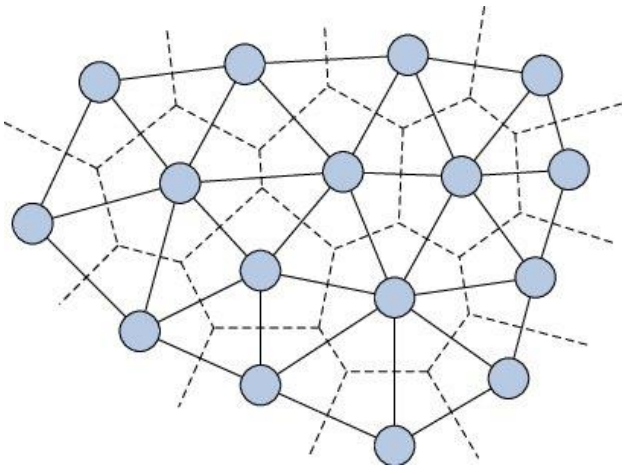

FIGURA 4: TRIANGULACIÓN DELAUNAY Y DIAGRAMA DE VORONOI **[18]**

En la imagen puede apreciarse la utilización de triángulos lo más regulares posibles en el caso de la Triangulación Delaunay, y la equidistancia existente entre cada segmento y los dos nodos adyacentes en el caso del diagrama de Voronoi.

Este método posee múltiples aplicaciones, una de ellas en concreto, la planificación de trayectorias y es ampliamente usada. Esto es porque minimiza el total de ángulos de los triángulos usados para la triangulación. Su complejidad es media y permite una gran velocidad para un número de puntos. El inconveniente principal es que la triangulación no es única y puede dar lugar a trayectorias extrañas que dificulten el movimiento o sean inválidas.

2.2.2 Métodos basados en grafos y arboles

Son los métodos más utilizados actualmente. Se caracterizan por que el espacio en el que planificar la trayectoria, está definido mediante el estado del robot respecto al entorno que lo rodea. Cada nodo o punto de futura conexión está definido por una configuración del robot.

Además, cada paso entre un nodo y el siguiente está basado en costes, siendo la trayectoria o camino más optimo, aquella que tiene menor coste. Siendo el coste, una unidad determinada en cada uno de los distintos métodos.

Esta clase puede dividirse en dos tipos de métodos principalmente, en función de cómo se definen el coste y cómo son construidos o asignados los nodos [3].

- D* o D estrella

Tras varias modificaciones a partir de dos algoritmos como *A** y *Dijkstra*, se ha desarrollo el principal algoritmo basado en una cuadricula en la que cada celda tiene asignado un coste en función de las características del entorno. Este es el algoritmo D* o D estrella.

Es un algoritmo popular, bastante útil para aplicaciones reales. Este convierte la cuadrícula de ocupación en un mapa de coste, que representa el coste 'c' de atravesar cada celda de manera horizontal o vertical, costando de manera diagonal $c\sqrt{2}$. Las celdas correspondientes a obstáculos tienen un coste infinito. [7]

Cada celda tiene un coste, una distancia al objetivo y un enlace hacia la celda vecina que está más cerca del objetivo. Adicionalmente, tiene un estado asociado, pudiendo ser 'Nuevo', 'Abierto' o 'Cerrado'. Inicialmente las celdas se encuentran en el estado 'Nuevo', salvo la final o celda objetivo, que permanece como 'Abierto'. Conforme se va generando la trayectoria, las celdas alcanzables pasan a estado 'Abierto' y una vez el algoritmo decide que celda selecciona de entre las abiertas para el siguiente paso, la celda del paso anterior se elimina de la lista de las celdas en estado 'Abierto' y pasa a 'Cerrado'.

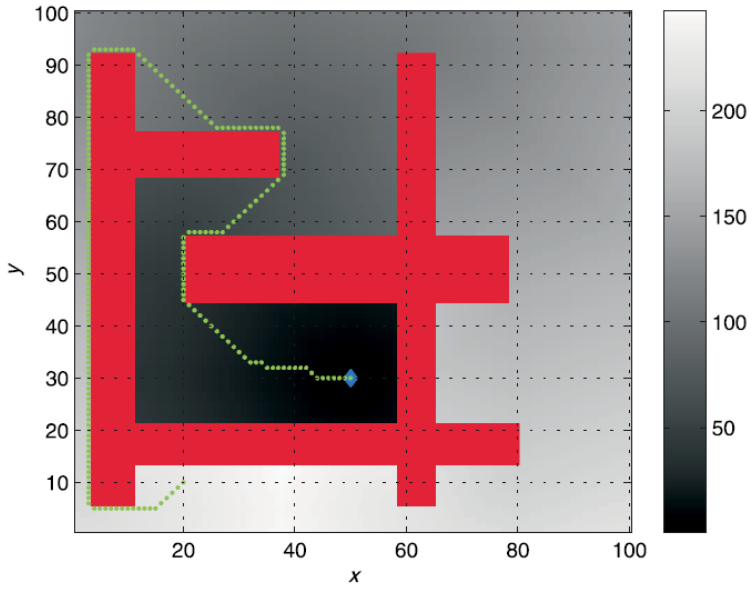

FIGURA 5: MOVIMIENTO D* [7]

En la figura superior, puede verse una cuadrícula de costes, en la cual se establece un punto objetivo, representando con color azul, al que se quiere llegar desde la posición x,y=20,10. A partir de este punto, se establecen los costes de forma que, a más cercanía de este, menos coste, y a más lejanía, mayor coste. Esta lejanía y coste asociado, se puede observar en la barra que se adjunta a la cuadricula, en la que, para ambas, a más oscuro, menor coste. En rojo están representados los obstáculos a evitar por la trayectoria. Como se puede observar, al ser el coste de una celda igual a "c" y el de una diagonal $c\sqrt{2}$, siempre que puede realizar la trayectoria por una diagonal para reducir coste en su camino hacia el objetivo, la realiza.

- Métodos basados en campo potencial

Estos métodos podrían clasificarse también en el grupo de aquellos que usan una cuadricula, sin embargo, pueden separase debido a que su concepto es distinto a estos.

En los *Campos Potenciales*, el robot es considerado como una carga eléctrica y el punto objetico como otra carga eléctrica de signo opuesto, de forma que se atraigan entre sí. A su vez, los obstáculos se definen con el mismo signo que el robot para que ambos se repelan y así evitar colisiones [8].

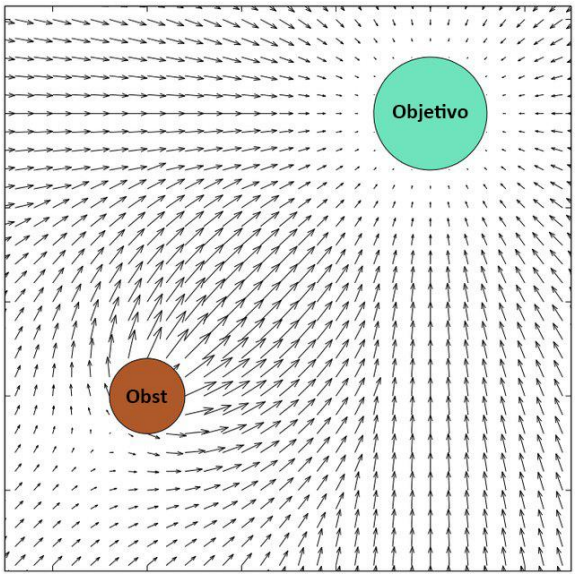

FIGURA 6: CAMPO POTENCIAL **[19]**

Sobre estas líneas se puede observar un esquema de cómo funcionaría el sistema de cargas del campo potencial. En él, se pueden observar dos círculos que representan el objetivo y un obstáculo. Alrededor suyo se pueden apreciar flechas, que indican la dirección de atracción desde ese punto concreto, siendo más pequeñas, cuanta más atracción (hacia el objetivo) o repulsión (desde el obstáculo) tengan.

- PRM (Probabilistic Road Map)

El siguiente método a tratar, sería el PRM (*Probabilistic Road Map*) o *Mapa de ruta probabilístico* [9]. Funciona de manera similar al RRT, el cual se expondrá a continuación, pero presentan algunas diferencias.

Estos métodos están basados en el muestreo aleatorio de puntos a los que se irán conectando otros nodos partiendo del punto inicial en búsqueda del punto objetivo. Además de la capacidad de detección de colisiones.

Tienen como problema principal, la aleatoriedad, pues puede devolver como resultado de la planificación, una trayectoria la cual no sea la más apropiada en términos de evasión de colisiones. Sin embargo, la rapidez de estos algoritmos en la búsqueda de una trayectoria compensa estos defectos.

En concreto, en este método (PRM), podría decirse que existen dos fases. En primer lugar, existiría la fase de planificación para definir las posibles conexiones y en segundo lugar la fase de consulta, la cual servirá para la definición de la trayectoria que unificara el punto inicial y final.

En la fase de planificación se crea una red de puntos aleatorios por todo el espacio libre, conectándolos entre sí sin que la conexión provoque colisión con el entorno. El punto origen y objetivo, no se añaden hasta la segunda fase, pudiendo estos dos ser modificados sin cambiar la trama ya creada en la primera fase, ahorrando por tanto tiempo y carga computacional. Esto significa, que una vez creada y finalizada la primera fase, se pueden dar valores origen y destino cuales quiera y se intentaran conectar con lo ya creado. Esto lleva una carga de computación muy inferior a otros casos, lo cual es una gran ventaja cuando en el problema a tratar, el espacio es grande, necesitando un gran mapa de trayectoria.

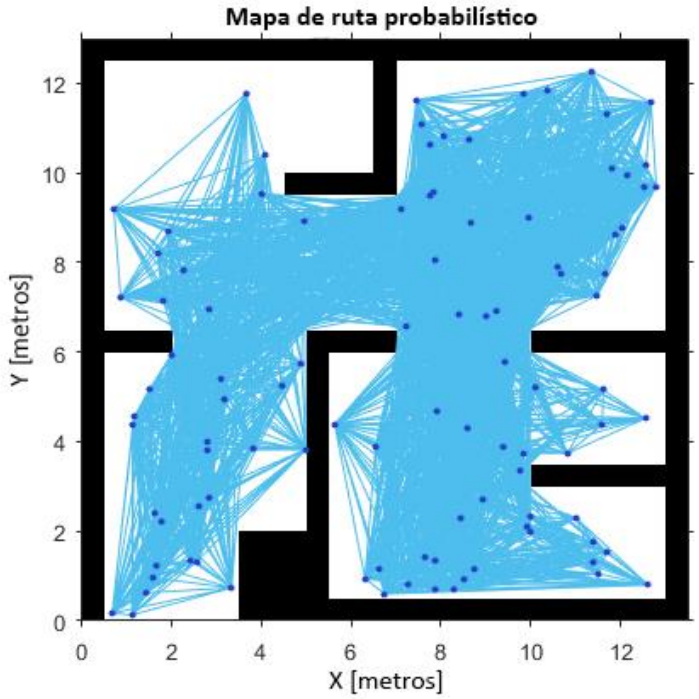

FIGURA 7: PRM [20]

En la imagen superior puede verse el mapa de ruta probabilístico. En concreto la fase de planificación, interconectada, generando por tanto múltiples caminos y nexos disponibles, los cuales serán usados posteriormente en función del punto inicial y final definidos. Puede verse, que la exploración del entorno no es homogénea, existiendo partes del espacio con saturación de puntos y/o caminos disponibles, mientras otras permanecen totalmente vacías.

- RRT

El ultimo planificador que se va a tratar dentro de esta sección es el RRT (*Rapidly-exploring random tree*), o *Arboles aleatorios de exploración rápida*, el cual es capaz de tener en cuenta que el movimiento de un vehículo o brazo robótico puede no ser omni-

direccional [24]. Al igual que el método PRM, es un algoritmo probabilístico. Este, sin embargo, asegura una exploración equiprobable de todo el entorno disponible para la generación de la trayectoria.

El algoritmo RRT se desarrollará más extensamente en apartados posteriores, al ser el elegido para la realización del proyecto [16].

2.3 Método Denavit-Hartenberg

Jackes Denavit y Richard Hartenberg introdujeron este método en 1955 con la intención de estandarizar la obtención de coordenadas para enlaces espaciales [11]. Posteriormente, en 1981, Richard Paul demostró su validez para el análisis cinemático de sistemas robóticos. Pese a que se han desarrollado nuevos métodos en el ámbito referido, el método de *Denavit-Hartenberg* sigue siendo una de las propuestas más populares. Para la explicación de dicho método se va a usar la información extraída del libro *"Fundamentos de Robótica"* de Antonio Barrientos [12].

La resolución del problema cinemático directo se resuelve encontrado la relación que permite conocer la localización del extremo del robot a partir de las coordenadas articulares. Para un robot con 6 grados de libertad, se escogerían las coordenadas cartesianas y los ángulos de Euler para así poder representar orientación y posición, quedando:

$$x = f_x(q_1, q_2, q_3, q_4, q_5, q_6)$$

$$y = f_y(q_1, q_2, q_3, q_4, q_5, q_6)$$

$$z = f_z(q_1, q_2, q_3, q_4, q_5, q_6)$$

$$\alpha = f_\alpha(q_1, q_2, q_3, q_4, q_5, q_6)$$

$$\beta = f_\beta(q_1, q_2, q_3, q_4, q_5, q_6)$$

$$\gamma = f_\gamma(q_1, q_2, q_3, q_4, q_5, q_6)$$

Generalmente, un robot de n grados de libertad posee el mismo número de eslabones unidos por el mismo número de articulaciones, de modo que el par eslabón-articulación constituye un grado de libertad. Asignando ahora a cada eslabón un sistema de referencia y utilizando las transformaciones homogéneas, se puede obtener las rotaciones y translaciones relativas a los eslabones que componen el robot.

En cuanto a las matrices de transformación homogéneas, aquella que representa la posición y orientación relativa se suele denominar ${}^{i-1}A_i$, representando esta la posición y orientación del sistema de referencia correspondiente al eslabón i con respecto al sistema de referencia de i-1.

Finalmente obtendríamos ${}^{0}A_n$, también denominada T, correspondiente a la matriz de transformación homogénea de toda la cadena cinemática, siendo esta el resultado del producto de cada una de las matrices homogéneas asociadas a cada grado de libertad. Quedando, por ejemplo, para 6 grados de libertad:

$$\mathbf{T = {}^{0}A_6 = {}^{0}A_1\,{}^{1}A_2\,{}^{2}A_3\,{}^{3}A_4\,{}^{4}A_5\,{}^{5}A_6}$$

A pesar de que se puede usar cualquier sistema de referencia ligado a cada componente, la forma habitual en robótica consiste en utilizar la representación de Denavit-Hartenberg. Según esta, eligiendo adecuadamente los sistemas de referencia correspondientes a cada eslabón, es posible pasar de uno al siguiente usando 4 transformaciones, las cuales solo dependen de las características geométricas del eslabón.

Estas transformaciones básicas, serian:

1. Rotación alrededor del eje z_{i-1} un ángulo θ_i.
2. Traslación a lo largo de z_{i-1} una distancia d_i; vector d_i (0,0, d_i).
3. Traslación a lo largo de x_i una distancia a_i; vector a_i (a_i,0, 0).
4. Rotación alrededor del eje x_i un ángulo α_i.

Debido a que el producto de matrices no es conmutativo, el producto de estas ha de hacerse en el orden indicado para aplicar la transformación comentada más arriba, correspondiente a ${}^{i-1}A_i$, de modo que tendríamos:

$$\mathbf{{}^{i-1}A_i = T(z, \theta_i)\,T(0,0, d_i)\,T(a_i,0, 0)\,T(x, \alpha_i)\ =>}$$

$$\mathbf{T= {}^{i-1}A_i} = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & 0 \\ S\theta_i & C\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha_i & -S\alpha_i & 0 \\ 0 & S\alpha_i & C\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

El método de *Denavit-Hartenberg* se basa en una serie de pasos que, al igual que el resto de este apartado se extraen de [12], y se enuncian a continuación:

1. Se numeran los eslabones comenzando con 1, correspondiente el primer eslabón móvil del brazo de la cadena cinemática en general o brazo robótico en particular y terminando con n, correspondiente al último eslabón móvil. La base corresponderá con el 0.

2. Numerar cada articulación comenzando por 1 y terminando en n, desde la articulación perteneciente al primer grado de libertad hasta la última respectivamente.
3. Localizar el eje de cada articulación. Si es rotativa, el eje será su propio eje de giro. En cambio, si es prismática, el eje será aquel a lo largo del cual se produce el desplazamiento.
4. Situar el eje z correspondiente a cada articulación en la siguiente (P.E.: el eje z_0 en la articulación 1, así hasta el eje z_{n-1} en la articulación n).
5. Situar el sistema de referencia de la base $\{S_0\}$ en cualquier punto del eje z_0 y posteriormente los ejes x_0 e y_0 de forma que conformen un sistema dextrógiro.
6. Situar para i de 1 a n, el sistema $\{S_i\}$ (solidario al eslabón i) en la intersección del eje z_i con la línea perpendicular común a z_{i-1} y z_i. Si ambos ejes se cortasen se situaría $\{S_i\}$ en el punto de corte. Si fueran paralelos $\{S_i\}$ se situaría en la articulación i+1.
7. Situar x_i en la línea perpendicular común a z_{i-1} y z_i.
8. Situar los ejes y_i de forma que formen un sistema dextrógiro con x_i y z_i.
9. Situar el sistema $\{S_n\}$ en el extremo del robot de modo que z_n coincida con la dirección de z_{n-1} y x_n sea normal a ambos
10. Obtener θ_i, como el ángulo que hay que girar en torno a z_{i-1} para que x_{i-1} y x_{i-} queden paralelos.
11. Obtener d_i como la distancia, medida a lo largo de z_{i-1}, que habría que desplazar el sistema de referencia anterior $\{S_{i-1}\}$ para que x_i y x_{i-1} queden alineados.
12. Obtener a_i como la distancia medida a lo de x_i (coincidente ahora coincidiría con x_{i-1}), que habría que desplazar el sistema de referencia anterior $\{S_{i-1}\}$ para que su origen coincidiese totalmente con el sistema de referencia siguiente $\{S_i\}$.
13. Obtener α_i como el ángulo que habría que girar en torno a x_i (coincidente ahora coincidiría con x_{i-1}), para que el nuevo sistema de referencia $\{S_{i-1}\}$ coincidiese totalmente con el sistema de referencia siguiente $\{S_i\}$.
14. Obtener las matrices de transformación definidas anteriormente.
15. Obtener la matriz de transformación que relaciona el sistema de la base con el del extremo del robot $T= {}^0A_1, {}^1A_2 \ldots {}^{n-1}A_n$.
16. La matriz T obtenida en el apartado anterior, define la orientación y posición del extremo referido a la base, en función de las coordenadas correspondientes a las articulaciones existentes.

2.4 Cinemática de un brazo robótico

La cinemática del robot, en nuestro caso un brazo robótico, se encarga de estudiar su movimiento respecto a un sistema de referencia. En particular por la posición y la orientación del extremo del robot, en función de sus articulaciones y sus coordenadas. De igual forma que con el método de *Denavit-Hartenberg*, se ha utilizado para la explicación de este apartado el libro *"Fundamentos de Robótica"* de Antonio Barrientos [12].

A la hora de este estudio cinemático entorno a un robot existen dos opciones. En primer lugar, tenemos la cinemática directa, que permite determinar cuál es la posición y orientación del extremo del robot en función del valor de las articulaciones y parámetros

geométricos del robot. Todo ello respecto a un sistema de referencia. Por el otro lado, tenemos la cinemática inversa, que posibilita obtener los parámetros de posición y orientación de la totalidad del brazo en función de la posición y orientación del extremo del mismo.

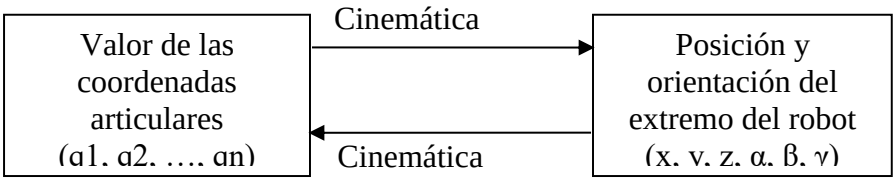

FIGURA 8: CINEMÁTICA ROBOT **[12]**

Para describir y poder representar la geometría espacial de una cadena cinemática, y en particular de un brazo robótico, con respecto a un sistema de referencia fijo, Denavit y Hartenberg desarrollaron un método. Para ello se valieron de una matriz de transformación homogénea que describiera la relación espacial entre dos elementos adyacentes, reduciendo el problema de la dinámica directa a encontrar una matriz de transformación homogénea 4x4 que relacionase la ubicación del extremo del brazo con el sistema de referencia de su base, tal y como se describe en el apartado relacionado con el método de *Denavit-Hartenberg*.

2.4.1 Cinemática Directa

Como se ha podido observar en el apartado anterior, un problema de cinemática directa se puede resolver encontrando las relaciones entre las articulaciones y eslabones del robot con el extremo de este. Para ello se puede asociar un sistema de referencia a la base del robot y referenciar a este el resto de sistemas de referencia del robot, es decir, el resto de articulaciones y eslabones.

Para llevar esta solución a cabo mediante matrices de transformación homogénea, haciendo uso del mencionado método de *Denavit-Hartenberg*. Reduciéndose el problema, por tanto, a la búsqueda de una sola matriz que relacione el extremo del robot con su base.

Este método es válido para cualquier manipulador independientemente del número o tipo de ligaduras que posea [21].

2.4.2 Cinemática Inversa

Ya hemos visto cual es el objetivo de la cinemática inversa. A diferencia del problema de la cinemática directa, el problema cinemático inverso no se puede resolver de manera sistemática y sin tener en cuenta la configuración del robot.

Por ello, se han ido desarrollando algunos métodos genéricos, los cuales puedan ser realizados por métodos computacionales, que consisten en métodos iterativos, que no tienen garantizada su efectividad. Por ello, para resolver un problema de cinemática inversa es mejor encontrar una solución cerrada, es decir, encontrar una relación matemática explicita tal como:

$$q_k = f_k(x, y, z, \alpha, \beta, \gamma)$$

Siendo k el número de grados de libertad del robot a resolver.

Este método tiene varias ventajas, como la posibilidad de resolver el problema en tiempo real, cuestión que mediante el modelo iterativo no garantizaría conseguir. Además, a diferencia del problema de cinemática directa, cabe la posibilidad de que existan varias posiciones que solucionen el problema. En estos casos, el uso de una solución cerrada permite añadir reglas o restricciones que permitan obtener la mejor y más adecuada de entre las soluciones disponibles. Sin embargo, esta solución cerrada es extremadamente difícil de encontrar, razón por la cual se ha optado por no seguir el camino de la cinemática inversa.

Existen varios métodos que pueden usarse para la resolución del problema cinemático inverso, como, por ejemplo: Por métodos geométricos (para robots de pocos grados de libertad), a través de matriz de transformación homogénea (con diferencias respecto a la cinemática directa), o por desacoplo cinemático.

2.5 Robotics Toolbox

Para la realización del proyecto se han usado diversas *toolboxes* de Matlab. Entre ellas, la principal es la *Robotics Toolbox* de Peter Corke, así como el manual que incluye en su propia página web, que además incluye el enlace de descarga de la propia *Robotics Toolbox* [13].

Esta *toolbox* contiene funciones que son útiles para el estudio y simulación de brazos robóticos incluyendo, entre otras cuestiones, la cinemática, dinámica y generación de trayectoria de estos manipuladores robóticos.

Robotics Toolbox contiene funciones y clases para representar la orientación y posición tanto en dos como en tres dimensiones como matrices, cuaternos, matrices exponenciales, etc.

Usa un método muy general de la representación cinemática y dinámica de manipuladores *serial-link* o de eslabones en serie, los cuales pueden ser creados por el usuario para cualquier manipulador.

Adicionalmente incluye soporte para la planificación de trayectorias en dos dimensiones, con diferentes algoritmos como los comentados anteriormente (D*, RTM, RRT).

Algunas de las ventajas de esta *toolbox* son que el código permite un punto de comparación con otras implementaciones de estos algoritmos, así como que las rutinas se encuentran definidas de manera que permite un fácil entendimiento de las mismas. Estas rutinas permiten, además, comprender acerca de la eficiencia computacional de los

procesos utilizados, pudiendo modificarse para mejorarse haciendo uso del compilador de Matlab.

La *toolbox* viene complementada además por un manual, que incluye todas las funciones y una breve descripción de las mismas, proporcionadas por el propio autor de la misma.

Complementariamente a la *toolbox* comentada en el apartado anterior, se harán uso de dos *toolboxes* que finalmente fueron integradas en la principal, pero que merecen mención aparte, las cuales se comentaran en los apartados siguientes.

2.5.1 pHRIWARE

Provee herramientas para analizar, investigar y evaluar interacciones físicas entre humanos y robots. Las herramientas principales que incorpora son cinemáticas como las de un brazo humano y comprobación de colisiones.

Nos interesa principalmente la herramienta de comprobación de colisiones. Dicha herramienta, basándose en el uso de matrices de transformación, genera una nube de puntos, los cuales se usan para generar el entorno y comprobar que unos objetos no invadan el espacio o volumen de los demás existentes en el entorno.

Para ello hace uso de la función *CollisionModel*, la cual, apoyándose en *SerialLink* (función que genera las características de un brazo robótico y que se encuentra contenida en la *toolbox* de Peter Corke), se encarga de analizar la interacción entre las matrices transformadas correspondientes a los objetos del entorno.

Esta *toolbox*, fue añadida a la de Peter Corke, pudiendo encontrarse más información en su propio sitio web [14].

2.5.2 ARTE

ARTE es una *toolbox* creada por Arturo Gil de la Universidad Miguel Hernández [15], la cual se centra en la manipulación robótica, para mecanismos tanto en serie como en paralelo.

Contiene varias funcionalidades:

- La simulación de cualquier robot industrial mediante el uso del entorno de Matlab.
- Representar y visualizar el sistema de *Denavit-Hartenberg* del robot.
- Contiene la representación gráfica en 3D de un gran número de robots industriales.
- Permite la visualización a través de Matlab de la posición, velocidad y aceleración de las articulaciones de un robot en movimiento, además de las fuerzas correspondientes a cada una de las articulaciones.

- La inclusión de nuevos modelos de robot.

En el caso actual y como se indicará posteriormente, se hará uso principalmente de la importación de un modelo nuevo de robot, el MANFRED-2, así como de la representación gráfica en 3D de este, complementando de este modo a las dos *toolboxes* comentadas anteriormente.

3. SOLUCIÓN AL PROBLEMA

Expuesta la cuestión, se va a proceder a exponer en detalle el algoritmo elegido para la planificación de la trayectoria, así como las adaptaciones del mismo para el caso concreto de MANFRED-2.

3.1 RRT

Como se ha comentado anteriormente, el algoritmo a utilizar para nuestro problema en concreto va a ser el RRT (*Rapidly Exploring Random Trees)* o Arboles aleatorios de exploración rápida [16].

El algoritmo RRT se basa en la construcción de un árbol de configuraciones que se expande a partir de un punto de origen, buscando nuevos puntos, hasta un punto destino.

Se llama configuración al conjunto de las posiciones y orientaciones de las articulaciones del brazo robótico que definen la posición de este en el espacio de trabajo. Definiéndose dicho espacio como aquel accesible por el brazo o lo que es lo mismo, el espacio en el cual pueden existir configuraciones del brazo robótico.

Durante el desarrollo de dicho algoritmo se usarán los siguientes conceptos que merece la pena destacar:

- C es el conjunto de todas las configuraciones posibles en el espacio de trabajo utilizado y que son, además, accesibles al robot.
- C_{free} es el subconjunto de C correspondiente a las configuraciones que no están en contacto con ninguno de los obstáculos existentes en el entorno.
- R es la métrica que se define para el sistema, definiendo ésta la orientación o dirección que tomara el árbol a la hora de extenderse y, por tanto, la dirección en la que buscara los nodos a los que conexionar.
- q_{ini} es el punto correspondiente al extremo del robot en la configuración inicial.
- q_{fin} es el punto que se desea alcanzar, siendo por tanto el punto que debe ocupar el extremo del robot al final de la trayectoria.
- q_{ran} es el punto aleatorio que genera el algoritmo, y que marcará la dirección en la que se definirá q_{new}.
- q_{near} es el punto más próximo a q_{ran}, en el sentido definido por R, de entre los existentes en un árbol.
- q_{new} es el punto que se va a añadir al árbol (Entre q_{near} y q_{rand}, a una distancia *nd* de q_{near}).
- nd es la longitud máxima que se define como la que debe haber entre un punto del árbol y el siguiente al que está conectado.

El objetivo de RRT consiste en la construcción de un árbol de exploración que abarcara de manera uniforme todo el espacio alcanzable por el robot y que no se encuentre en

colisión con el mismo. Dicha construcción, desarrollada a modo de algoritmo por LaValle en 1998, consiste en seleccionar un punto de forma aleatoria y extender hacia el mismo, el árbol de configuraciones/puntos. A continuación, se puede ver una simplificación del algoritmo:

```
1. Algoritmo RRT(qini)
2. {
3.     Arbol[0] = qini;
4.     para k = 1 hasta Kmax
5.     {
6.           qrand = ConfiguracionAleatoria();
7.           Extiende(Arbol, qrand);
8.     }
9.    devolver Arbol;
10.}
```

FIGURA 9 ALGORITMO RRT BÁSICO **[16]**

Se puede ver como se usa la función *Extiende*, la cual se encarga de ampliar el árbol en el sentido que le indica q_{rand} determinando si existe un camino libre de colisión.

El algoritmo parte desde la configuración inicial asociada al árbol. A continuación, entra en un bucle (limitado por el número de iteraciones a realizar en el proceso), para completar todas las configuraciones posibles del árbol, atendiendo, además, a las restricciones debidas a obstáculos comentadas. El número de iteraciones a realizar, serán importantes a la hora de detener el bucle o ejecución del algoritmo, en caso de que no se alcance el objetivo. Este número dependerá de las restricciones añadidas al sistema, ya sean en forma de obstáculos o de tiempo límite de realización del algoritmo.

Dentro del algoritmo RRT, nos encontramos con dos instrucciones principales. En primer lugar, se obtiene un punto aleatorio dentro del espacio definido (C_{free}). En segundo lugar, expande el árbol hacia la posición del punto aleatorio generado en la instrucción anterior.

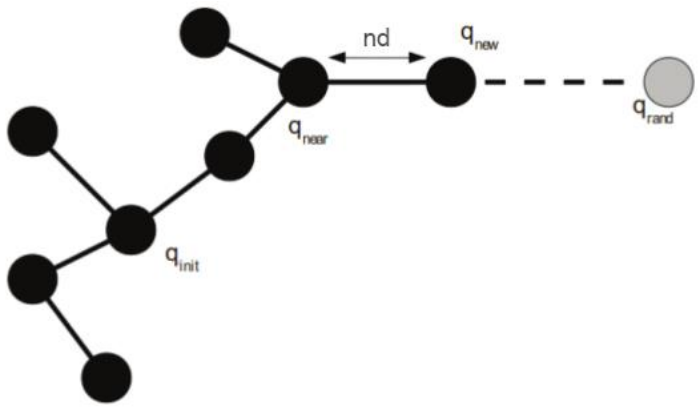

FIGURA 10: FUNCIONAMIENTO ALGORITMO RRT **[16]**

La expansión del árbol se consigue, como se comentó anteriormente, mediante la función *Extiende*. Dicha función, como se puede ver, comienza con la obtención de q_{near} (cuál de los puntos cercanos, se convertirá en el nodo de unión). Este punto configuración se genera gracias a la función *VecinoMasProximo*, que haciendo uso de "R" (definida en los parámetros a utilizar), aplica esta métrica a todos los puntos del árbol, obteniendo el punto más próximo a q_{rand}. Posteriormente, *NuevaConfiguración* se encarga de calcular q_{new}, correspondiente al nuevo punto que definitivamente se añadirá al árbol, con una distancia respecto a q_{near} de magnitud *nd*, partiendo de q_{near} en dirección a q_{rand}. Básicamente, q_{rand} puede haberse creado aleatoriamente a una distancia mayor de *nd* desde q_{near}, razón por la cual, q_{new} se define en la recta que une q_{near} y q_{rand} a una distancia *nd*, asegurando asi la distancia *nd* máxima. Nuevamente, para calcular dicha dirección se tiene en cuanta que no existe ningún obstáculo en el desplazamiento desde q_{near} a q_{new}. En caso de existir alguna colisión que impidiera realizar este movimiento, se devolvería un *falso*, volviendo a repetir el proceso y no añadiendo q_{new} al árbol.

```
1. Función Extiende(Arbol, q_rand)
2.{
3.      q_near= VecinoMasProximo(q_rand, Arbol);
4.      si (NuevaConfiguracion(q_rand, q_near, q_new) )
5.      {
6.         AñadeVertice(Arbol, q_new);
7.         si (q_new=q_rand)
8.         {
9.            devolver alcanzado;
10.        }
11.         de otro modo
12.        {
13.           devolver avanzado;
14.        }
15.     }
16.     de otro modo
17.     {
18.        devolver rechazado;
19.     }
20.}
```

FIGURA 11 FUNCIÓN EXTIENDE [16]

En caso de ser posible el movimiento, devolvería un *verdadero*, añadiendo este nuevo punto al árbol de configuraciones/puntos. Una vez se añade este nuevo punto, existen dos posibilidades. La primera, si este punto se encuentra dentro en el interior de una circunferencia de radio "nd" y centro q_{fin}, se considera que se ha alcanzado q_{fin} (punto final de la trayectoria), y el algoritmo devolverá *alcanzado*. En caso contrario, la segunda

posibilidad, más probable que ocurra que la primera, devolverá el valor *avanzado*, añadiendo esta "q" al árbol de exploración, terminando una iteración y procediendo a repitir el proceso, generando nuevos nodos hasta que se alcance q_{fin}.

Una vez alcanzado q_{fin} se considerará que se ha alcanzado el objetivo, y se procederá a generar una trayectoria. Esta trayectoria se definirá siguiendo el camino desde q_{fin} hasta q_{ini}. Para ello, se ira "esaclando" por el árbol desde el punto q_{near} que conectó finalmente con q_{fin}, pasando de esté a su nodo-padre (nodo desde el cual se creó el q_{near} anterior, acercándose hacia el punto inicial) hasta llegar a q_{ini}, completando así la trayectoria.

El algoritmo RRT se diferencia respecto a otros en la exploración más homogénea del entorno libre de obstáculos. Además, debido a sus características, le permite avanzar con mayor rapidez en aquellas zonas donde exista mayor especio libre y por tanto más posibilidades de encontrar puntos q_{rand} alcanzables. Por esta razón, también existe menos densidad de ramas en el origen del algoritmo, al intentar siempre extenderse, de manera uniforme por todo el espacio disponible.

A continuación, se muestran varias imágenes mostrando la expansión del árbol, en función del número de iteraciones realizadas:

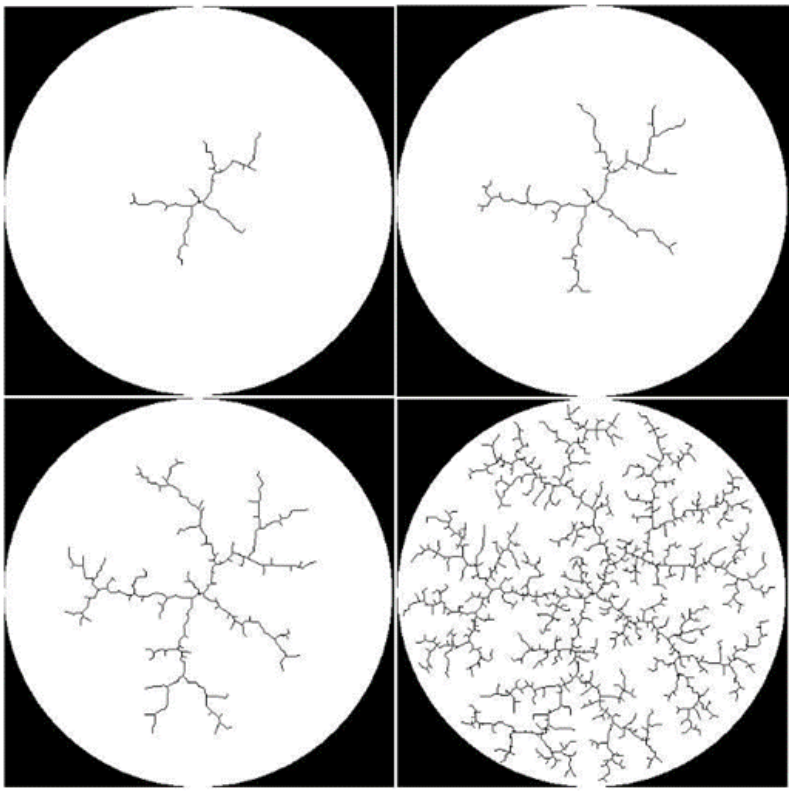

FIGURA 12 EVOLUCIÓN ALGORITMO RRT EN ESPACIO HOMOGÉNEO [16]

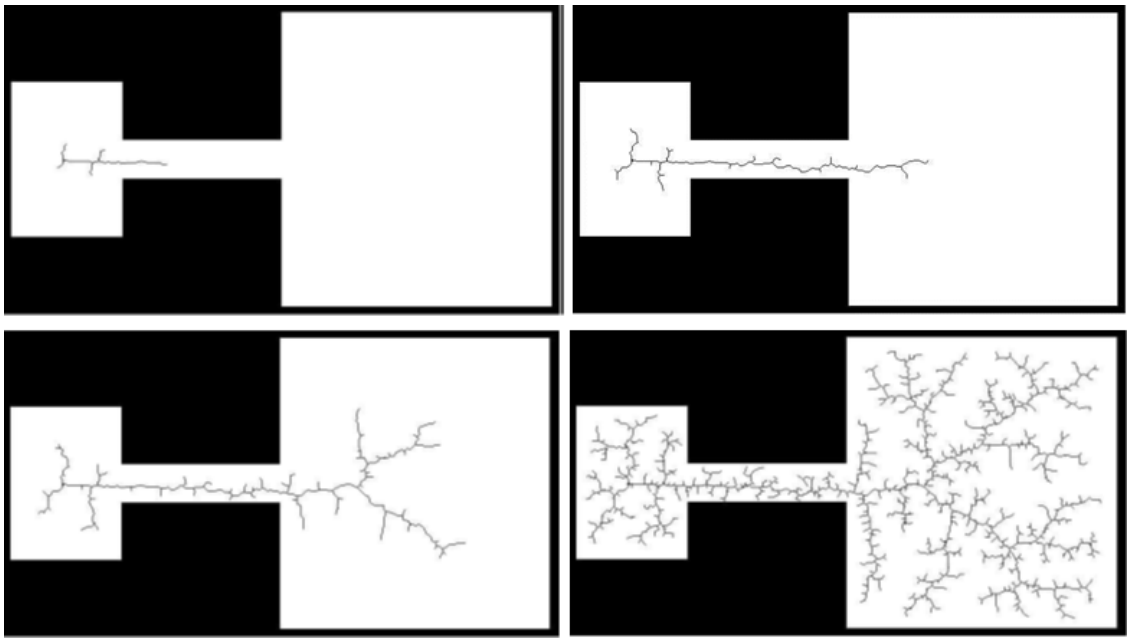

FIGURA 13 EVOLUCIÓN ALGORITMO RRT EN ESPACIO HETEROGÉNEO **[16]**

Como se puede apreciar en la imagen anterior, el algoritmo RRT parte desde el centro del espacio circular. Según va avanzando el proceso, se empieza a extender de manera uniforme por todo el espacio disponible, principalmente intentando ocupar el mayor espacio libre posible, sin superpoblar de nodos la zona central o de partida, por encima del resto de espacio (como si ocurría en el PRM visto anteriormente).

En el caso de un espacio no homogéneo, se puede apreciar como igual que en el caso homogéneo anterior, el árbol empieza a extenderse hacia el espacio más grande disponible, incluso pasando por el conducto que une los dos espacios principales. Mantiene en este caso la no sobre población de la zona inicial de crecimiento por encima de otras zonas del espacio.

3.2 Definición MANFRED2

Como se ha comentado al inicio, el MANFRED2 es un manipulador con 6 grados de libertad. Para poder realizar una planificación de trayectorias haciendo uso del algoritmo RRT, necesitamos trasladar el manipulador al entorno de Matlab, usando para ello *Robotics Toolbox*, la cual nos permitirá manejar los aspectos necesarios para la configuración del brazo robótico.

En concreto se utilizará la función *SerialLink*. Esta función, permite representar un brazo robótico de eslabones en serie, como objeto robótico en Matlab. Cada unión o articulación y enlace de la cadena se describe usando los parámetros de *Denavit-Hartenberg*.

La función *SerialLink* requiere para su uso la definición de cada uno de los parámetros de los enlaces o grados de libertad. Estos parámetros serán, el ángulo theta, el vector distancia para ubicarlo en el espacio (formado por dos parámetros a y d, correspondientes con las coordenadas x e y si se pusiera el enlace en un plano), el ángulo alpha y por último el tipo de articulación, es decir, si es prismática o rotacional.

Una vez definidos los parámetros de todos los enlaces se puede crear el objeto robot haciendo uso de la mencionada función *SerialLink*, pudiendo representar el mismo en una figura simplemente añadiendo una posición "q" para el brazo y sus articulaciones, quedando como se puede apreciar en la siguiente imagen.

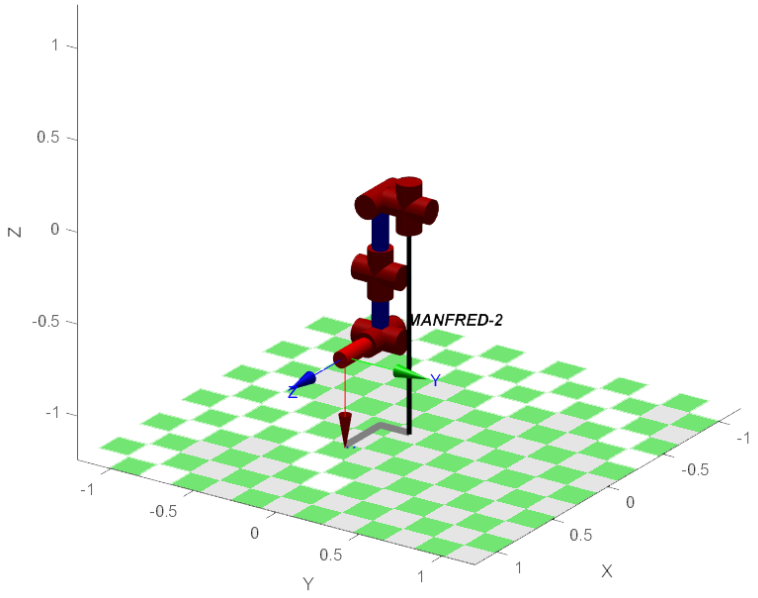

FIGURA 14 DEFINICIÓN MANFRED-2 EN MATLAB

Con el objeto robot ya creado, es necesario definir los límites de giro para cada una de las articulaciones del brazo robótico, para impedir que estas tomen posiciones que no serían posibles en el uso real del brazo. Para ello, se hace uso de la función *qlim*, perteneciente al objeto mencionado, añadiendo para cada articulación un rango definido por el ángulo de giro mínimo que al que puede rotar la articulación y el ángulo de giro máximo.

El MANFRED-2 está definido por 6 grados de libertad, y por tanto 6 componentes que conforman la cadena de eslabones, los cuales poseen las siguientes dimensiones:

Articulación	Tamaño (m)	Parámetros de Denavit-Hartenberg			
		a (m)	d (m)	Theta (rad)	Alpha (rad)
q1	0,175	0	0,175	0	pi/2
q2	0,4	0,4	0	0	-pi/2
q3	0	0	0	0	-pi/2
q4	0,35	0	0,35	0	pi/2
q5	0	0	0	0	-pi/2
q6	0,316	0	0,316	0	0

TABLA 1 PARÁMETROS MANFRED-2

En la tabla superior se exponen los tamaños de las 6 articulaciones delrobot. Se puede observar que tanto *q4* como *q5*, tienen un tamaño nulo; esto es debido a que son dos articulaciones puramente rotacionales y no eslabones como las otras cuatro. Adicionalmente, se añaden los parametros de Denavit-Hartenberg a continnuacion del tamaño.

3.3 Planificación de trayectoria

El objetivo principal de este trabajo es la planificación de trayectorias para unir un punto inicial y uno final orientado al MANFRED-2. Para ello, como se ha comentado se hará uso del algoritmo RRT.

Utilizando de base un planificador de trayectorias RRT, creado por Gavin Paul y Matthew Clifton [10], se desarrollarán las modificaciones pertinentes para adaptarlo a las necesidades propias del MANFRED-2. En él, se han añadido diferentes funciones y modificado otras para la adaptación

A continuación se muestra el funcionamiento del algortimo RRT defindo anteriormente durante la explicación del mismo, modificado añadiendo en azul, las variaciones resepto a este necesarias para el funcionamiento con MANFRED-2.

```
1. Algoritmo RRT(q_ini)
2. {
3.   robot = manfred8 (SerialLink())
4.  collisionMod (CollisionModel())
5.  Arbol[0] = q_ini;
6.  para k = 1 hasta Kmax
7.   {
8.   NewPoint (q_ini)
9.       {
10.              while Comprobacion_limites = 1 o Comprobación_colisiones = 1
11.              {
12.                       qrand = ConfiguracionAleatoria(q_ini);
13.                       nuevo punto = fkine(qrand, robot)
14.                       Comprobacion_limites = islimit(qrand,robot)
15.                       Comprobacion_colisiones = collisions (qrand, robot, entorno)
16.              }
17.              q_ini = qrand;
18.              nPoint=nuevo_punto;
19.      }
20.  Extiende(Arbol, nPoint);
21.   }
22. devolver Arbol;
23.}
```

FIGURA 15 ALGORTIMO RRT MODIFICADO

En primer lugar, se ha añadido una función *manfred8*, la cual se encarga de la definición descrita en el apartado anterior. Esta función genera el objeto robot, correspondiente a nuestro manipulador, y se asignara a una variable, la cual será usada siempre que sea requerida durante la ejecución.

El algoritmo utilizado, es capaz de hacer uso de múltiples arboles de puntos, así como planificar trayectorias en 2 o 3 dimensiones. Para definir el entorno en el que se van a buscar los puntos para las posibles trayectorias, se utilizarán ejes de coordenadas tanto para el espacio como para las figuras añadidas al mismo, es decir, coordenadas cartesianas y distancias euclídeas.

Como la planificación de trayectorias se realiza en un entorno de 3 dimensiones de coordenadas cartesianas, es necesario realizar una transformación de las 6 coordenadas necesarias para definir la posición del extremo del brazo robótico.

Para dicha transformación se hará uso de la cinemática directa, así como de los parámetros (definidos para cada articulación durante la definición de MANFRED-2 como objeto de Matlab) y método de Denavit-Hartenberg. Esta tarea puede realizarse mediante la función *fkine*, incluida en *RoboticsToolbox*, la cual nos devuelve como resultado la matriz

homogénea de transformación correspondiente, pudiendo extraer de dicha matriz, las tres coordenadas cartesianas del punto extremo.

```
ans =
         0         0         1     0.316
         0         1         0    -0.175
        -1         0         0     -0.75
         0         0         0         1
>>
```

FIGURA 16 MATRIZ TRANSFORMACIÓN PARA CINEMÁTICA DIRECTA

En la imagen superior permite ver el resultado de la transformación mediante la función *fkine*, en la cual los 3 primeros valores de la cuarta columna corresponden con los valores cartesianos x, y, z de la posición del extremo del robot.

Una vez encontrada la relación entre los 6 grados de libertad pertenecientes a las 6 articulaciones del brazo robótico y los 3 grados de libertad correspondientes al punto x, y, z que ocupa el extremo del robot, el siguiente paso consiste en la definición y búsqueda de un nuevo punto o nodo de conexión, mediante la función *NewPoint*, la cual se ha rediseñado por completo. Este nuevo punto se define como el siguiente posible punto a añadir dentro del entorno de búsqueda del algoritmo RRT. Pese a poder transformar el valor de 6 a 3 coordenadas, necesitamos indicar este nuevo punto en 6 grados de libertad, pues la cinemática directa es capaz de determinar la posición del extremo del robot a partir de la configuración de sus articulaciones, pero no al revés. Por lo tanto, se deben dar 6 valores aleatorios para los grados de libertad 1 a 6, pudiendo obtener así configuraciones aleatorias del brazo. Para definir la aleatoriedad de cada grado de libertad, se parte de una posición inicial del brazo definida en la creación del brazo robótico, mediante la función *Manfred8*, comentada anteriormente, a partir de la cual se van realizando pequeños movimientos aleatorios de cada una de las articulaciones dentro de un rango de búsqueda, permitiendo así controlar de mejor manera las posiciones que alcance el robot. Esto nos permite aumentar la probabilidad de encontrar puntos que son válidos, si el punto de partida ya era válido.

Para cada configuración, se hará uso de la función *fkine* explicada unos párrafos atrás, permitiendo obtener ese punto en coordenadas cartesianas. Una vez obtenido el valor en coordenadas cartesianas, este debe ser comprobado. Debe comprobarse que, para el punto elegido, la configuración del robot es correcta, es decir, que las articulaciones se encuentren dentro de los límites y que además el brazo no esté colisionando con ningún objeto del entorno de búsqueda. Estas dos condiciones se llevarán a cabo a través de principalmente 2 funciones.

La primera de ellas comprueba, a partir de los límites de articulación añadidos durante la definición del brazo, que la posición que corresponde con la configuración aleatoria es

posible y alcanzable por el robot. Para ello, se usa la función *islimit,* que se encarga de comprobar que la configuración del robot dada cumple con las condiciones establecidas anteriormente.

La segunda de ellas se encargará, haciendo uso de las *toolboxes* o complementos pHRIWARE y ARTE, las cuales se explican en apartados posteriores. Estas, nos permitirán comprobar si para una configuración aleatoria dada, el robot se encuentra en colisión, siendo por tanto esta configuración o posición del brazo inviable.

Para ello, se hará uso de una función nueva llamada *collisionMod*, la cual se encarga de generar al inicio de la ejecución del algoritmo todo el entorno, así como el modelado de MANFRED-2, para que, cuando se vaya a realizar la comprobación de colisión dentro de la función *NewPoint*, baste con llamar a la función *collisions*, perteneciente a la *toolbox* pHRIWARE, que se encarga de controlar la condición de colisión. Tanto *collisionMod* como *collisions* serán explicadas en el apartado dedicado a las colisiones.

En caso de incumplir alguna de las 2 condiciones de validez (que alguna de las dos de un valor de uno booleano), ese punto se elimina y se procede a la búsqueda de otra configuración, hasta encontrar una opción válida. Al encontrarse una configuración dentro de los límites y libre de colisión, esta se añade al árbol de búsqueda de trayectoria, para continuar con el proceso. Además, esta configuración válida pasa a sustituir a la configuración inicial, de tal forma que en la siguiente búsqueda se parta de esta última.

La repetición de este proceso va produciendo el árbol de puntos o configuraciones que permite explorar el espacio en busca de una trayectoria. Como se ha comentado en la definición del algoritmo RRT, una vez se alcanza el objetivo, se traza un camino entre los puntos que unen el inicio y fin de la trayectoria, correspondiéndose estos puntos intermedios con las posiciones que deberá adoptar el extremo del brazo robótico en su movimiento hacia el objetivo, las cuales tienen asociadas las configuraciones ya comprobadas durante la ejecución.

El número de repeticiones que se realicen durante la ejecución vendrá dado mediante el paramento de *Iteraciones máximas* incluido dentro del algoritmo.

En las imágenes inferiores se puede obserbar al robot durante la ejecuión del algortimo.

En la imagen de la izquierda, se aprecia cómo va generando el arbol de puntos, intentando conexionar los dos puntos rojos, correspondientes a inicio y objetivo, representads como asteriscos.

En la imagen de la derecha, se puede apreciar la ejecución terminada tras encontrar una trayectoria (linea roja) y el suavizado realizado a esta trayectoria (linea verde), el cual como se ha comentado anteriormente, en el caso de un entorno con obstáculos, debe tratarse con cautela, debido a los posibles saltos bruscos, pudiendo haber en dicha trayectoria posibles colisiones. También se puede apreciar, representados como puntos azules, los nodos generados durante la ejecución para la búsqueda de la trayectoria.

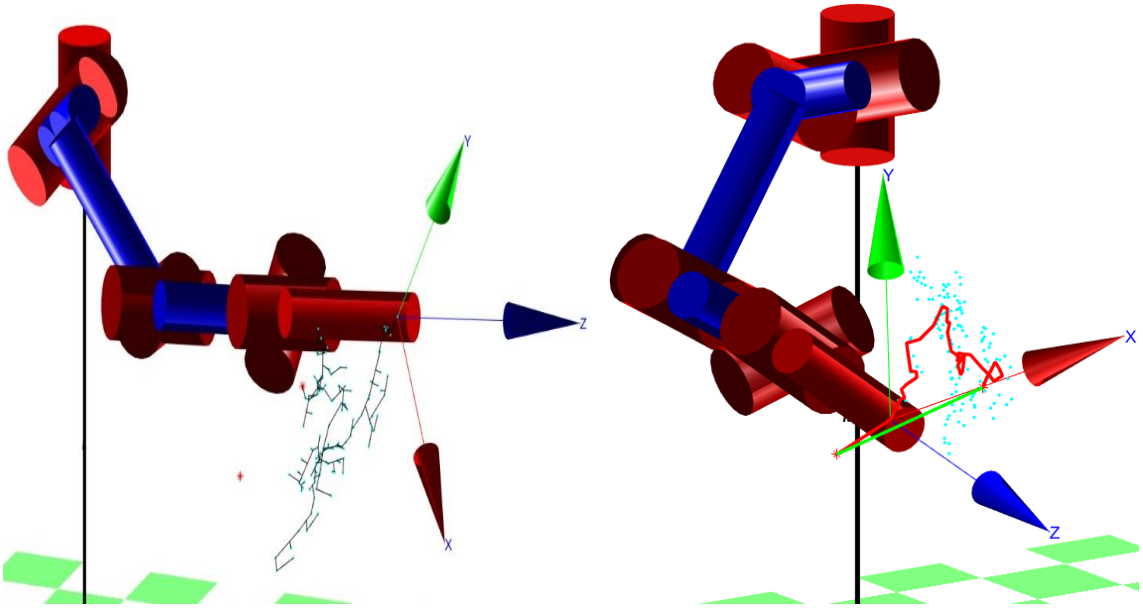

FIGURA 17 EJECUCIÓN ALGORITMO PARA MANFRED-2

3.4 Colisiones

Al ser el MANFRED2 un manipulador pensado para ser usado en un entorno real y con obstáculos, es necesario que es capaz de manejar los posibles obstáculos que pueda encontrarse a la hora de la realización de movimientos y de esta manera evitar colisiones que puedan dañar el mismo brazo robótico o impedir que alcance en el objetivo.

Para ello, como se ha comentado antes, se hará uso del complemento pHRIWARE para la *Robotics Toolbox*. En concreto las herramientas *CollisionModel* y *collisions*, la cual nos permitirá definir los objetos del entorno de trabajo del robot, que se convertirán en obstáculos a la hora de planificar las trayectorias a realizar.

Para generar ese entorno, la herramienta hará uso de matrices primitivas en las cuales estarán definidos todos los objetos a incluir en el espacio de trabajo. Estas primitivas se definen con la forma del objeto a añadir. Ya sea a partir de la unión de varias figuras simples (cubo, cono, cilindro…) o a partir de la carga de un STL (*Standard Triangle Language*) o Lenguaje Estándar de Triangulación, que contenga el modelado del objeto en cuestión.

Sin embargo, tanto en un caso como en otro, ambas deben estar regidas acorde a unas funciones definidas en la figuras contenidas en la propia *toolbox* y las cuales son necesarias para *CollisionModel*, que se encargan de convertir las figuras en una nube de puntos que se utilizará posteriormente con la herramienta/función *collisions*.

Una vez definidas todas las primitivas, se hace uso de la función *CollisionModel* que nos agrupa todas ellas en un solo objeto que sería el entorno, el cual ya estaría preparado para ser usado en la detección de colisiones, habiendo sido procesado y convertido en la nube de puntos comentada anteriormente.

Entorno = CollisionModel(prim1, prim2, ..., primN)

Todo esto, estará contenido en la función *collisionMod,* añadida en algoritmo y la cual se comentó en el apartado anterior.

Teniendo ya el modelo de colisiones correspondiente al entorno de trabajo, se utilizará la función *collisions*. Esta función o herramienta permite relacionar el movimiento del robot, o la posición actual del robot, con el entorno que lo rodea, ya sea estático o dinámico.

Colisiones = robot.collisions(q, cmdl, dyn_cmdl)

Correspondiendo:

- **robot**: definición de los parámetros del robot, definidos mediante la función comentada anteriormente *SerialLink* (comentar función serial link en RRT), y que además contiene los STL correspondientes a cada uno de los segmentos y articulaciones del MANFRED2. Estos, serán cargados y definidos mediante la herramienta/*toolbox* ARTE, la cual se explicará en el apartado siguiente.
- **q**: Posición y orientación de las articulaciones del brazo en un instante dado.
- **cmdl**: Modelo de colisión de objetos que se encuentran estáticos en el entorno de trabajo.
- **dyn_cmdl**: Modelo de colisión de objetos que varían de posición durante el movimiento del robot, es decir, para cada posición "q" del brazo, en el entorno de trabajo.

Con estos 3 componentes dados, además del robot y su modelado, la función *Collisions* se encarga de comprobar que (mediante la nube de puntos que configuran todos los elementos del sistema en conjunción), para cada configuración del brazo, este no se encuentre contenido en el volumen de alguno de los diferentes objetos del entorno, ya sea dinámico o estático, devolviendo un valor afirmativo o negativo para cada posición del brazo.

Esta función, puede realizarse independientemente, tanto para objetos solo estáticos, solo dinámicos o con objetos de las dos clases.

Una vez ejecutada, ésta devolverá un valor booleano, el cual, si es cero, querrá decir que está libre de colisiones y si es uno, que el robot se encuentra en colisión con algún elemento del entorno.

A continuación, se muestra una imagen que muestra el robot, con los diversos obstáculos añadidos mediante la las funciones definidas en este apartado.

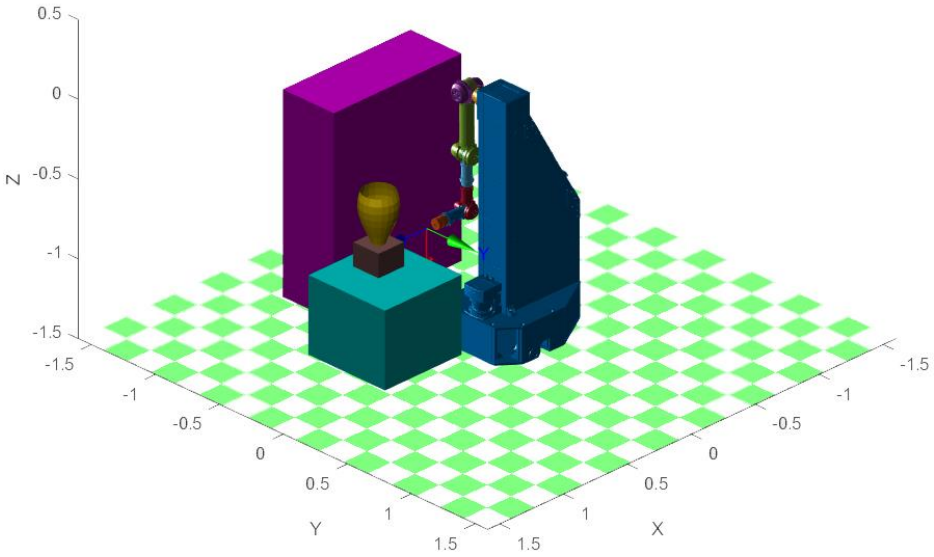

FIGURA 18 ENTORNO CON OBSTÁCULOS

3.5 Modelado en Matlab, STL

ARTE (A Robotic Toolbox for Education) es una *toolbox* pensada para manipuladores robóticos. Como se ha comentado anteriormente, se usará para la definición del modelado del MANFRED2 y cada uno de sus elementos.

Contiene un gran número de modelados de robots industriales, ya incluidos en la *toolbox*, a partir de los cuales se pueden importar modelados correspondientes a manipuladores nuevos.

En nuestro caso, al ser un robot único, el MANFRED2 requiere de esta utilidad. Para ello se necesitarán los STL, con los modelos 3D de los componentes del manipulador, los cuales serán cargados mediante la *toolbox*.

Adicionalmente, será necesaria la definición del brazo como objeto en Matlab, a través de la función *SerialLinked* comentada anteriormente, que nos defina las condiciones y características del manipulador a usar en cuestión, pudiendo ser estas cinemáticas o dinámicas.

Con estos elementos, ARTE, consigue unir el modelado de una articulación a su homólogo en el objeto definido del robot en Matlab, teniendo finalmente nuestro objeto *SerialLinked* con el modelado real de MANFRED2.

Esto nos permite, al incluirlo en la función *Collisions* comentada anteriormente, poder estudiar de manera más real las colisiones que se puedan producir entre el robot y el entorno, para cada posición durante la simulación o búsqueda de trayectoria entre el punto de origen y el punto de destino.

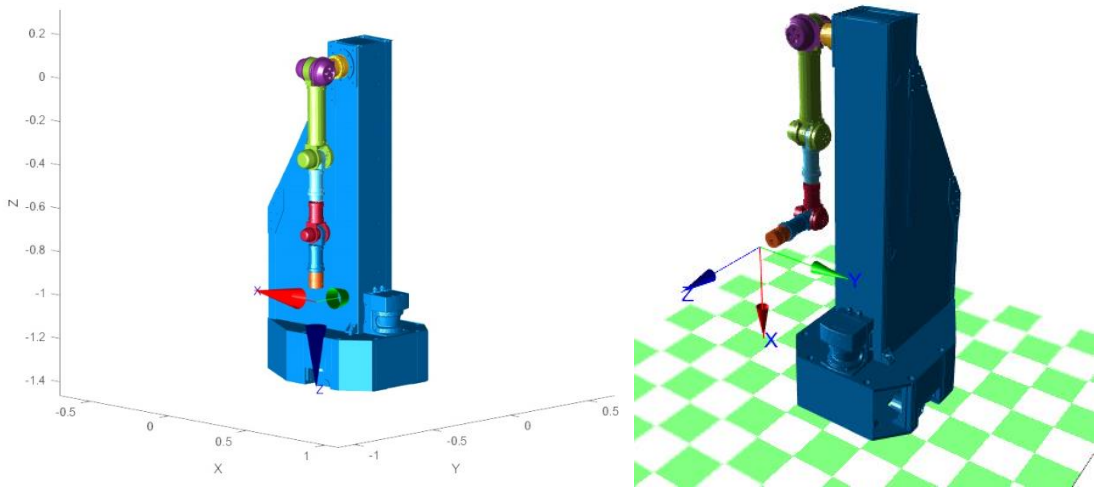

FIGURA 19 MODELADO MANFRED-2

En las imágenes superiores se puede apreciar el modelado del robot con su base, a partir de sus STL desde diferentes ángulos de vista y para diferentes configuraciones del brazo robótico.

4. RESULTADOS EXPERIMENTALES

A continuación, se van a desarrollar las distintas pruebas o experimentos realizados con el algoritmo. Para ello se ha hecho uso de un ordenador personal con las siguientes características: Procesador Intel Core i5 de 6 núcleos a 4.2 GHz y 16 Gb de RAM.

Por tanto, el tiempo empleado para cada una de las pruebas es exclusivo de este ordenador y por ello, se han realizado la totalidad de las pruebas en él, haciendo uso de Matlab, como se ha comentado durante este documento.

4.1 Sin obstáculos

Las pruebas sin obstáculos se basan en el uso exclusivo del propio brazo robótico intentando alcanzar un punto del espacio en el que se encuentra ubicado. Para ello se tendrán en cuenta únicamente las restricciones propias a los límites de cada una de las articulaciones del manipulador, para cada una de las configuraciones usadas en la búsqueda de nodos o puntos de conexión, para generar la trayectoria.

El objetivo de estas pruebas es obtener información acerca del porcentaje de éxito del algoritmo utilizado en la planificación de trayectorias, así como el tiempo empleado en la búsqueda del punto objetivo.

Para ello se tendrán en cuenta diferentes parámetros principales. Estos son:

- D: Distancia existente entre punto inicial y punto final.
- i: Número de iteraciones máximas permitidas para alcanzar el objetivo.
- ε: Distancia máxima permitida para la conexión de un nodo con el siguiente.

Las pruebas han sido realizadas para cada uno de los casos principales relacionando los parámetros principales de estos con los de los demás casos. Para cada caso se ha obtenido una muestra de 200 simulaciones.

La posición inicial para la que se han realizado las pruebas es la correspondiente a la siguiente configuración del robot y el punto que ocuparía el extremo de este:

$$q\_ini = [-pi/2\ 0\ 0\ 0\ 0\ 0]$$

$$punto\ inicial = [0.666\ -0.175\ -0.4]$$

En cada uno de estos escenarios que se describirán a continuación se tienen en cuenta tanto aquellos resultados en los cuales se ha conseguido alcanzar el objetivo, como aquellos en los cuales se ha llegado al número máximo de iteraciones sin encontrar una trayectoria que una los puntos deseados.

Adicionalmente, se incluyen otros parámetros como son la variación del punto inicial hacia una zona límite o el número de árboles a utilizar durante el funcionamiento del algoritmo.

4.1.1 Diferente número de iteraciones para una misma distancia entre puntos y de conexión.

Elegida una distancia dada por la separación entre el punto de partida del brazo robótico y el punto objetivo o final, se procede a aplicar el algoritmo para diferentes iteraciones. Además, también se fija la distancia máxima de conexión entre nodos. El número de iteraciones elegidas han sido 200, 1000, 5000, 10000 y 20000.

A continuación, se podrán observar distintos escenarios, como se comenta en el párrafo anterior, en los que se fija una misma distancia entre inicio y objetivo, así como la distancia de conexión entre todos. Se tratarán distintos escenarios en los que se tendrá en cuenta el porcentaje de éxito, el tiempo medio y la relación entre estos. Se incluyen tanto los casos en los que se ha logrado alcanzar el objetivo como los que no, para hacer una estimación de las características del experimento en una situación real.

Para tener diferentes fuentes de cómo se comporta el sistema en función del número de iteraciones, se van a analizar 3 casos distintos. Para D = 0,5 y nd = 0,1, D = 0,3 y nd = 0,15, D = 0,95 y nd = 0,2.

- Porcentaje de éxito

En las figuras que siguen a este párrafo reflejan como afecta el número de iteraciones respecto al número de veces que el algoritmo consigue conectar el punto inicial y el punto objetivo.

Como se puede observar en los 3 casos, pese a que dependen de otros parámetros y existen ligeras variaciones, la tendencia es la misma. A mayor número de iteraciones, el algoritmo dispone de más nodos con los generar una trayectoria, aumentando, como cabría esperar, el porcentaje de acierto.

Puede apreciarse también, el salto en cuanto a porcentaje, que existe entre un primer grupo formado por 20000, 10000, 5000 iteraciones y un segundo grupo formado por 1000 y 200 iteraciones. Esto nos muestra que el segundo grupo, no es el más interesante para obtener un buen ratio de alcance de objetivo a priori.

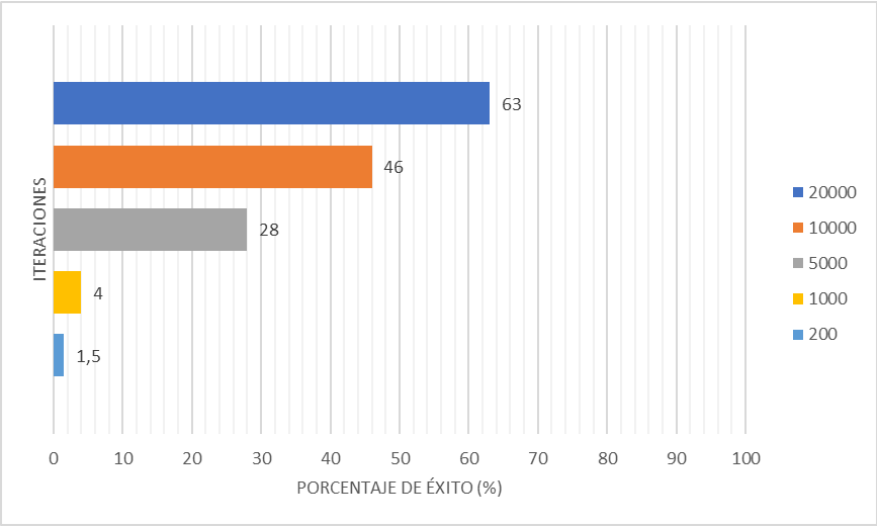

FIGURA 20 PORCENTAJE ÉXITO (NO OBST) D = 0.5 ND = 0.10

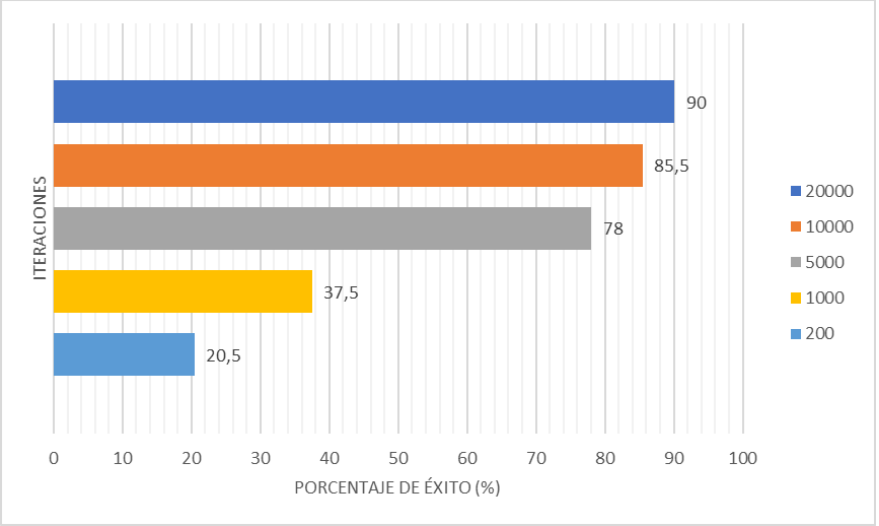

FIGURA 21 PORCENTAJE ÉXITO (NO OBST) D = 0.3 ND = 0.15

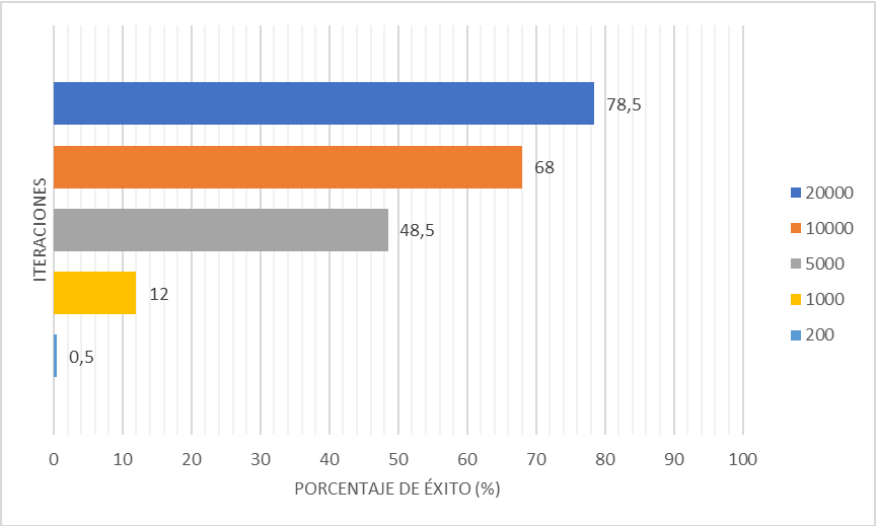

FIGURA 22 PORCENTAJE ÉXITO (NO OBST) D = 0.95 ND = 0.2

- Tiempo medio por prueba

A continuación, se muestra el tiempo medio que tarda el algoritmo en completar una ejecución para el distinto número de iteraciones.

A diferencia del caso anterior, para el porcentaje de éxito, en este caso a mayor número de iteraciones obtenemos un valor peor o menos interesante para nuestro sistema. Esto era de esperar, pues como se acaba de decir, al añadir más nodos al árbol durante la ejecución estamos generando más carga de trabajo que se traduce en un incremento temporal.

En cuanto al tiempo se refiere, se mantiene también el salto que encontrábamos en el caso anterior entre los grupos formados por 20000, 10000, 5000 iteraciones y 1000, 200 iteraciones. Por tanto, en función de los parámetros que se elijan para la ejecución del algoritmo, dependiendo de las necesidades de cada caso particular, convendrá tener un menor porcentaje de acierto, pero a la vez una ejecución más rápida o viceversa.

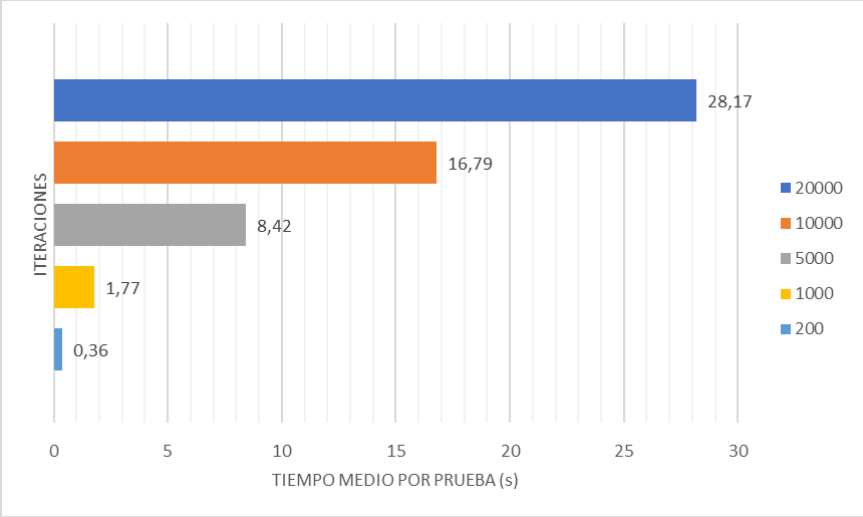

FIGURA 23 TIEMPO MEDIO (NO OBST) D = 0.5 ND = 0.1

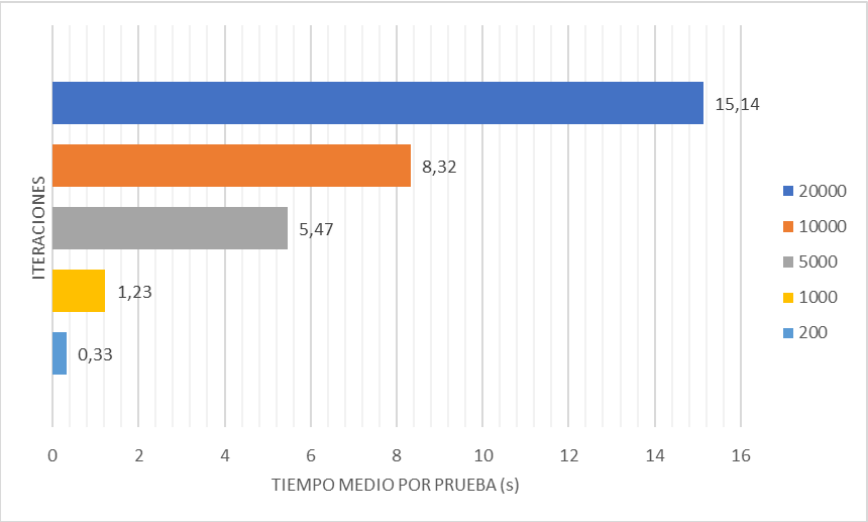

FIGURA 24 TIEMPO MEDIO (NO OBST) D = 0.3 ND = 0.15

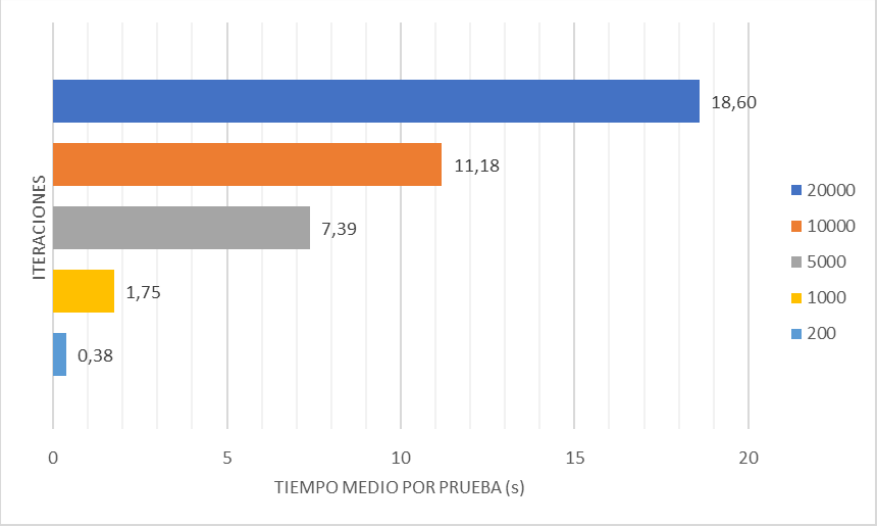

FIGURA 25 TIEMPO MEDIO (NO OBST) D = 0.95 ND = 0.2

Se puede apreciar que, a mayor distancia de conexión máxima entre nodos, se obtienen mejores resultados. Esto se detallará más adelante en el apartado dedicado a este parámetro.

- Relación entre pasos durante la ejecución y tiempo empleado

Como se ha podido observar en este apartado el porcentaje de existo crece en una proporción similar a la que crece el tiempo en función del número de iteraciones usadas durante la ejecución del algoritmo.

Se puede ver en las siguientes figuras la relación entre el tiempo y el porcentaje de éxito. Para estos experimentos se ha mantenido fijo el número de iteraciones en 20000 y se ha variado la distancia entre objetivo y la distancia máxima de conexión, observándose en los tres casos una pendiente muy similar y una serie de puntos que pueden asemejarse perfectamente una recta y por tanto a una relación lineal entre ambos parámetros.

Como nota, los valores mínimos de las 3 gráficas, debido a la escala del eje vertical y su nivel de incremento, parecen nulos. Sin embargo, no hay ningún valor igual a cero ni en lo referente a los pasos dados, ni en el tiempo utilizado. En el caso de los pasos se encuentran en torno a los 200 pasos y en el del tiempo en torno a los 0.3 segundos.

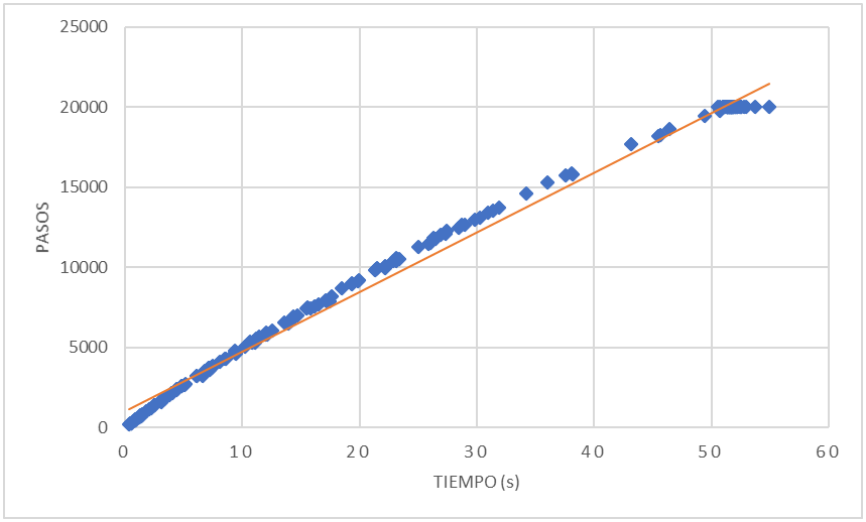

FIGURA 26 RELACIÓN PASOS/TIEMPO (NO OBST) I = 20000 D = 0.5 ND = 0.1

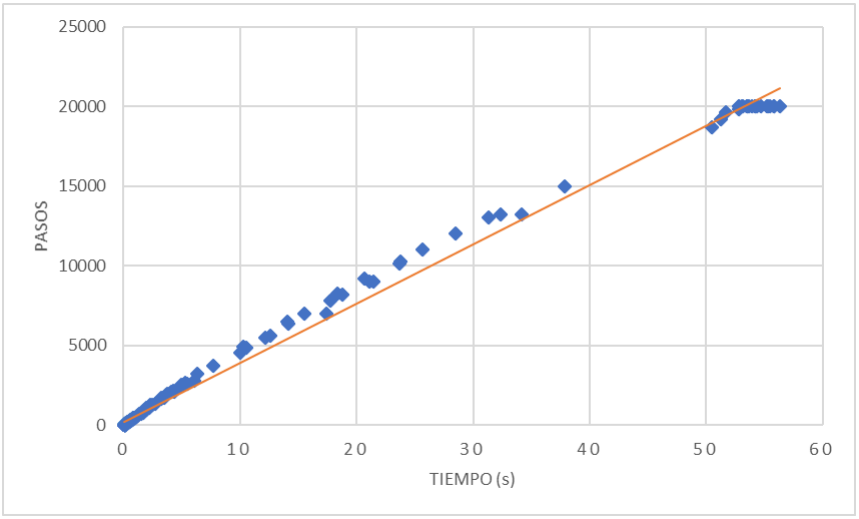

FIGURA 27 RELACIÓN PASOS/TIEMPO (NO OBST) I = 20000 D = 0.3 ND = 0.15

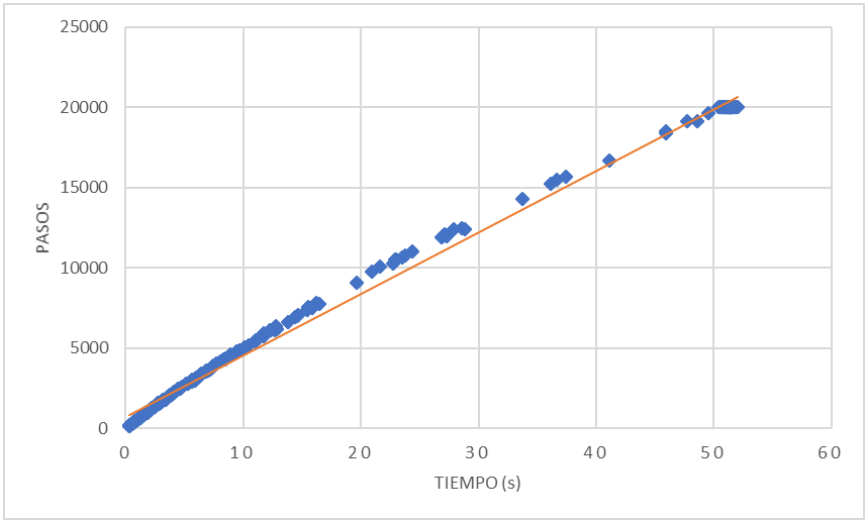

FIGURA 28 RELACIÓN PASOS/TIEMPO (NO OBST) I = 20000 D = 0.95 ND = 0.2

4.1.2 Diferente distancia para un mismo número de iteraciones y/o distancia de conexión.

En esta ocasión se va a proceder a ver cómo afecta el cambio de la distancia entre punto inicial y objetivo, fijando el resto de los parámetros. Para ello, se van a usar las mismas distancias entre que se utilizaron en el apartado anterior, es decir, 0,3, 0,5 y 0,95 metros.

Se va a tener nuevamente en cuenta como resultado, el porcentaje de éxito alcanzado con esta configuración de parámetros, además, del tiempo de ejecución.

Para la realización de estas pruebas o simulaciones, se fijarán las iteraciones y la distancia de conexión, para diferentes separaciones entre punto inicial y final u objetivo.

- Porcentaje de éxito

En las figuras que acompañan a este experimento, se han utilizado las 3 distancias entre punto inicial y objetivo, agrupándolas para cada una de las iteraciones usadas anteriormente, es decir, 20000, 10000, 5000, 1000 y 200 iteraciones.

Como se puede observar, se aprecia un comportamiento prácticamente idéntico entre todos los casos. Salvo en el caso de 20000 iteraciones en el que el porcentaje entre las tres distancias esta escalonado en incrementos similares, en el resto casos, se puede apreciar cómo se forman 2 grupos.

El primero está formado por la distancia corta (D = 0,3m), en el cual el porcentaje de éxito es mayor y disminuye en menor medida, incluso para el caso más desfavorable (200 iteraciones).

El segundo grupo está conformado por la distancia media y larga, las cuales, podría decirse que van de la mano, aportando un porcentaje de éxito similar en todos los casos. Además, se aprecia como el porcentaje desciende bastante más y más pronunciadamente que con el grupo compuesto por la distancia corta (D = 0.3m).

Adicionalmente se puede observar que, a menor distancia, en general, mayor porcentaje de éxito.

Se puede concluir, por tanto, que, para distancias medias y largas, el algoritmo funciona mejor a partir de las 5000 iteraciones en el caso de distancia de conexión entre nodos de 0.15 y 0.2 m (se procederá a analizar este parámetro en apartados posteriores), donde alcance porcentajes de entre el 40-60% hasta el 90% y a partir de 10000 iteraciones en el caso de nd = 0,1, donde el porcentaje parte desde el 40% hasta el 63%.

Para distancias cortas, sin embargo, puede resultar interesante cualquiera de las iteraciones, teniendo hasta en el caso más desfavorable un 21%, que pese a poder parecer un porcentaje malo, en el siguiente apartado, en el estudio del tiempo de ejecución, se verá si el tiempo medio de este, en concreto su velocidad, compensa la pérdida de eficiencia.

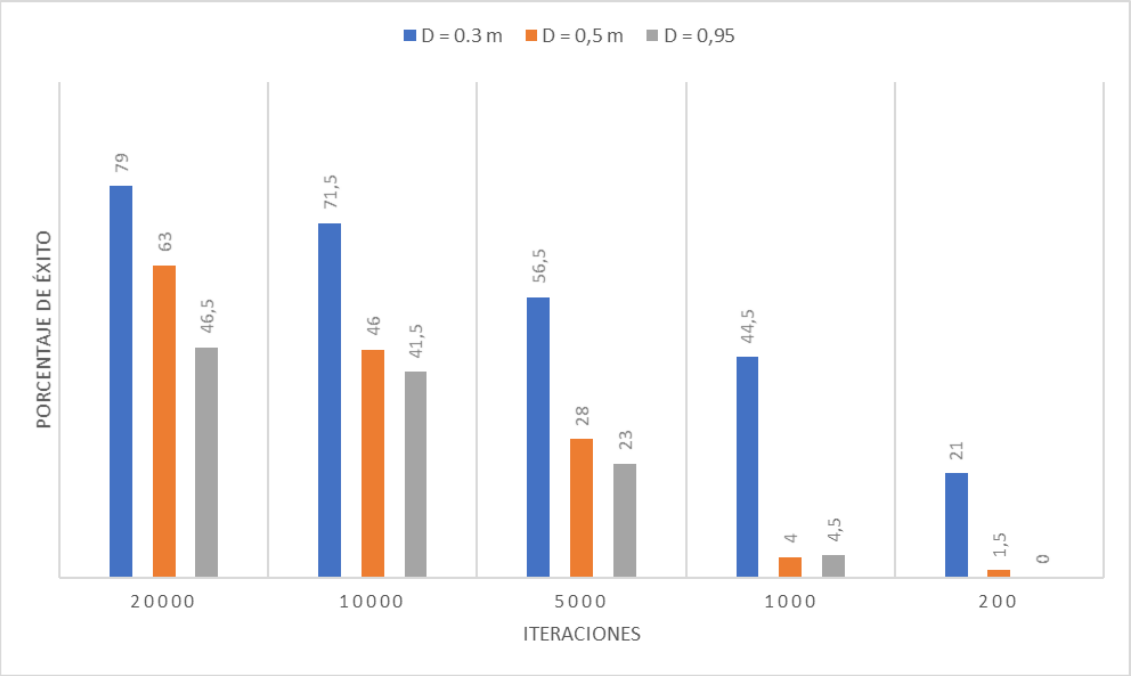

FIGURA 29 PORCENTAJE ÉXITO (NO OBST) ND=0.1

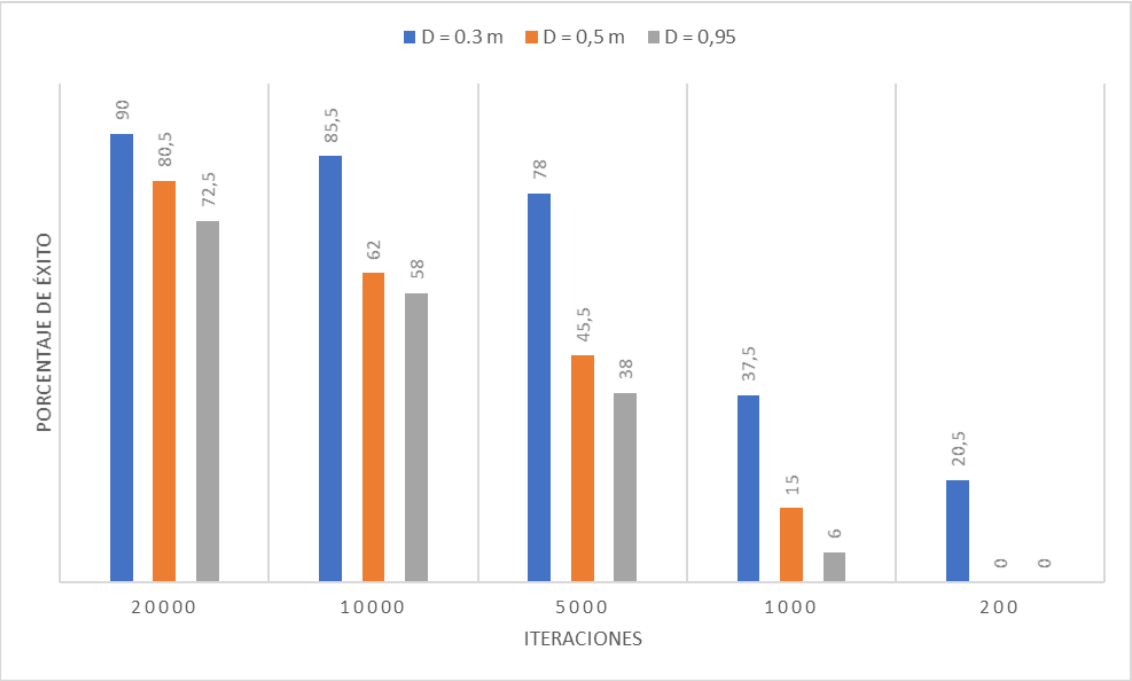

FIGURA 30 PORCENTAJE ÉXITO (NO OBST) ND=0.15

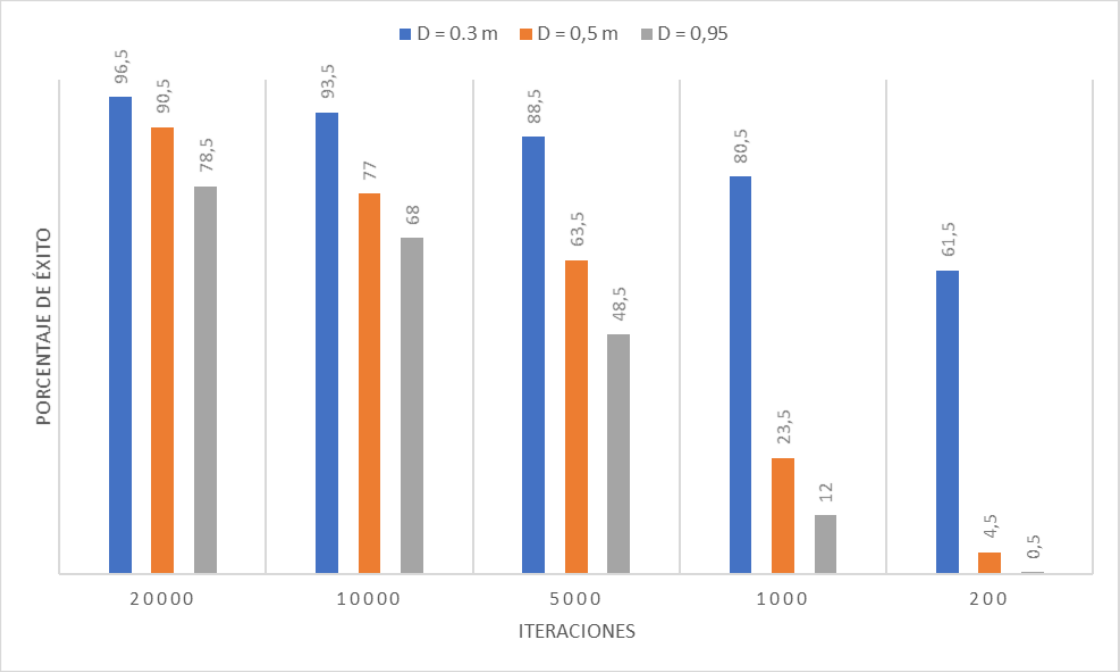

FIGURA 31 PORCENTAJE ÉXITO (NO OBST) ND=0.2

- Tiempo medio por prueba

Se va a proceder de igual manera que en el apartado anterior, cuando se trató el porcentaje de éxito en función de la distancia. Se agruparán las distancias para el distinto número de iteraciones, comprobando como afecta esto al tiempo medio de ejecución.

En las gráficas inferiores (figuras 32 a 34), se puede apreciar que ocurre el mismo suceso que nos encontrábamos en el porcentaje de éxito, es decir, la distancia media y larga, poseen unos tiempos medios de ejecución similares, siendo en ocasiones prácticamente idénticos y además una variación del tiempo más drástica entre iteraciones de la que se puede apreciar para la distancia corta. Igualmente, en la distancia corta se aprecia que actúa de modo independiente, siendo en la práctica totalidad de los casos, tiempos bajos de ejecución.

Podemos, por tanto, decir que, al igual que en el apartado del porcentaje de éxito, a partir de una determinada distancia, al pasar de distancias cortas o muy cortas a distancias medias o largas, el tiempo llega a doblarse o triplicarse para un número de iteraciones altas.

En cuanto al escenario óptimo, se puede ver, si atendemos al apartado anterior junto con este, que, para el caso, de distancias medias y largas, aunque a iteraciones bajas el tiempo es bastante menor, no compensa el porcentaje de éxito tan bajo que tiene. Sin embargo, a partir de las 5000 iteraciones puede verse, que el tiempo crece en mayor medida que el porcentaje, conforme pasamos de 5000 a 20000 iteraciones. Por tanto, podría decirse que el caso que mejor cumple el compromiso entre tiempo tardado y porcentaje en función de la distancia que separa el punto inicial y final de la trayectoria, es el de 5000 iteraciones.

Para distancias cortas, en el apartado anterior, se pudo observar que para cualquier número de iteraciones a partir de 200 podía ser interesante su uso. Si atendemos ahora al tiempo medio de ejecución, se puede ver que la diferencia entre 200 y 1000 iteraciones es del orden de 3-4 veces superior, siendo su porcentaje aproximadamente el doble. Por tanto, interesarían las 200 iteraciones por encima de las 1000. De igual modo ocurre con el resto de las iteraciones, es decir, 20000, 10000 y 5000, de forma incluso más clara, puesto que en comparación la relación entre incremento de tiempo y de porcentaje es mayor.

Por tanto, se puede concluir que el caso más optimo atendiendo a las diferentes distancias entre inicio y objetivo, en el caso de distancias cortas (0,3 metros),es el de 200 iteraciones, puesto que aunque en el peor de los casos (para nd = 0,1 y nd = 0,15), está en torno al 20%, su gran velocidad nos permite repetir el experimento y obtener, estadísticamente hablando, un resultado favorable en menor tiempo que el resto de casos, pese a tener estos mejores porcentajes de éxito.

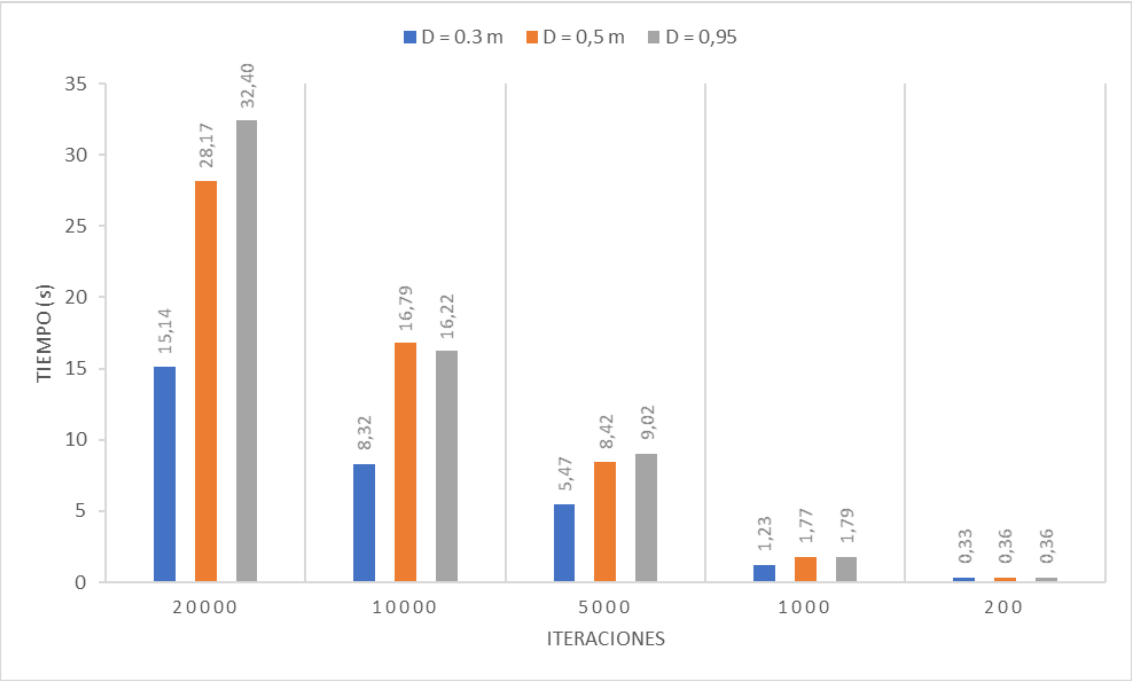

FIGURA 32 TIEMPO MEDIO (NO OBST) ND=0.1

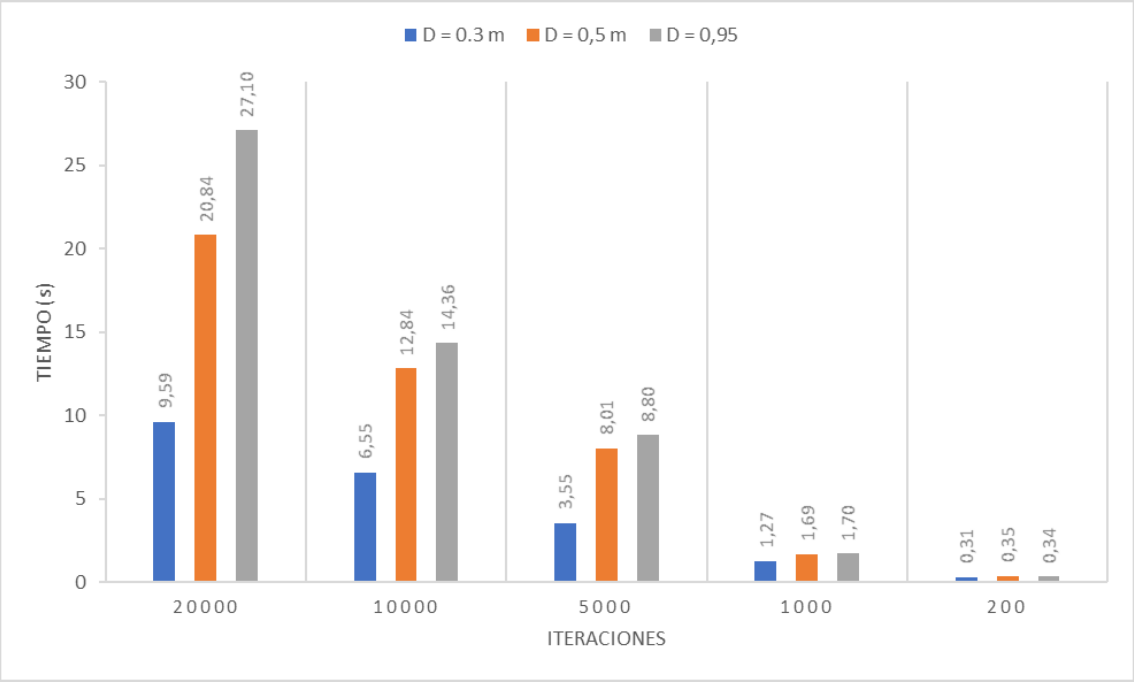

FIGURA 33 TIEMPO MEDIO (NO OBST) ND=0.15

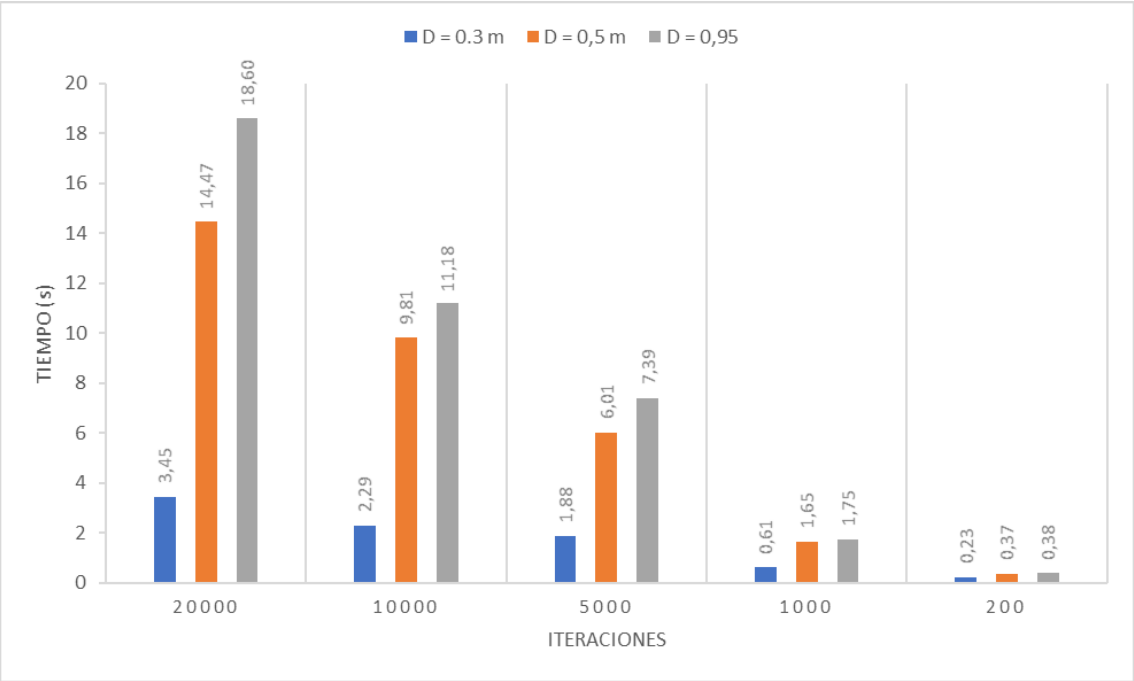

FIGURA 34 TIEMPO MEDIO (NO OBST) ND=0.2

- Relación entre tiempo durante la ejecución y tiempo empleado

A continuación, se va a mostrar en las gráficas que acompañan a este apartado, como se relacionan el porcentaje de éxito y el tiempo de ejecución, en función de la distancia y el número de iteraciones.

Se va a dividir este apartado en dos puntos de vista. El primero agrupará por número de iteraciones y el segundo por distancias.

Si **agrupamos por iteraciones**, se puede apreciar que, aunque en mayor o menor proporción, todos los casos siguen la misma tendencia dentro de su entorno. Para un mismo número de iteraciones, se puede observar lo que se ha venido viendo en apartados anteriores, es decir, para distancias más bajas, obtenemos más porcentaje de éxito en menor tiempo. Sin embargo, superada cierta distancia, y para número de iteraciones más bajos, estos valores se igualan enormemente, no encontrando diferencias prácticamente entre dos distancias largas o medias.

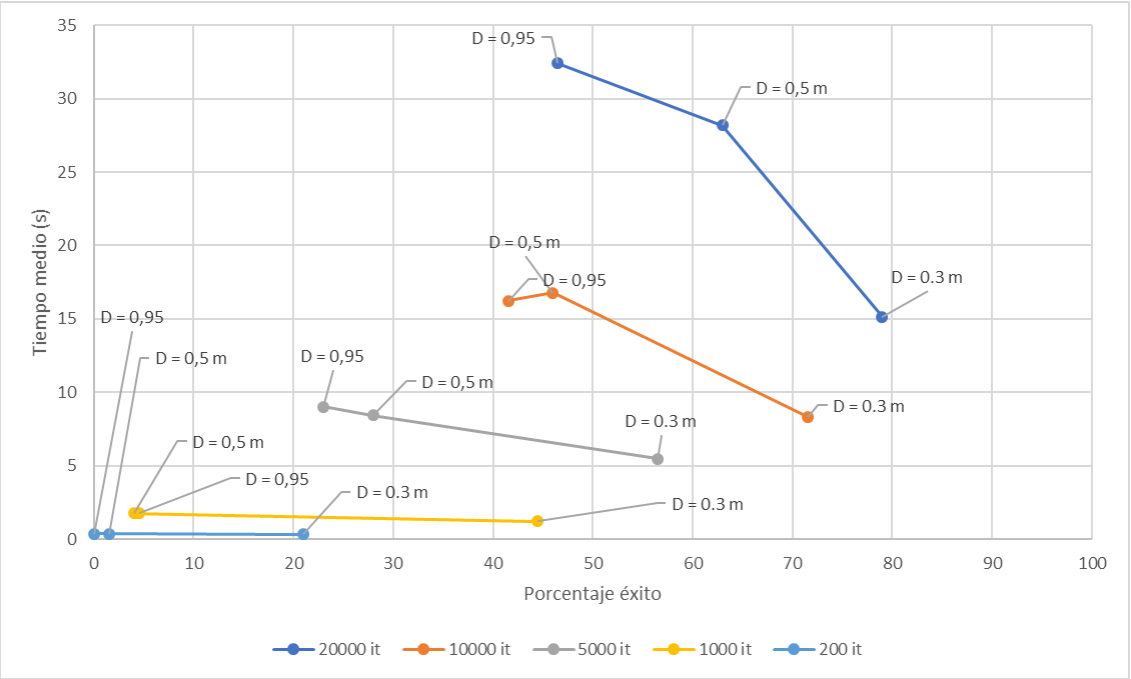

FIGURA 35 RELACIÓN TIEMPO MEDIO/ÉXITO, ITERACIONES (NO OBST) ND=0.1

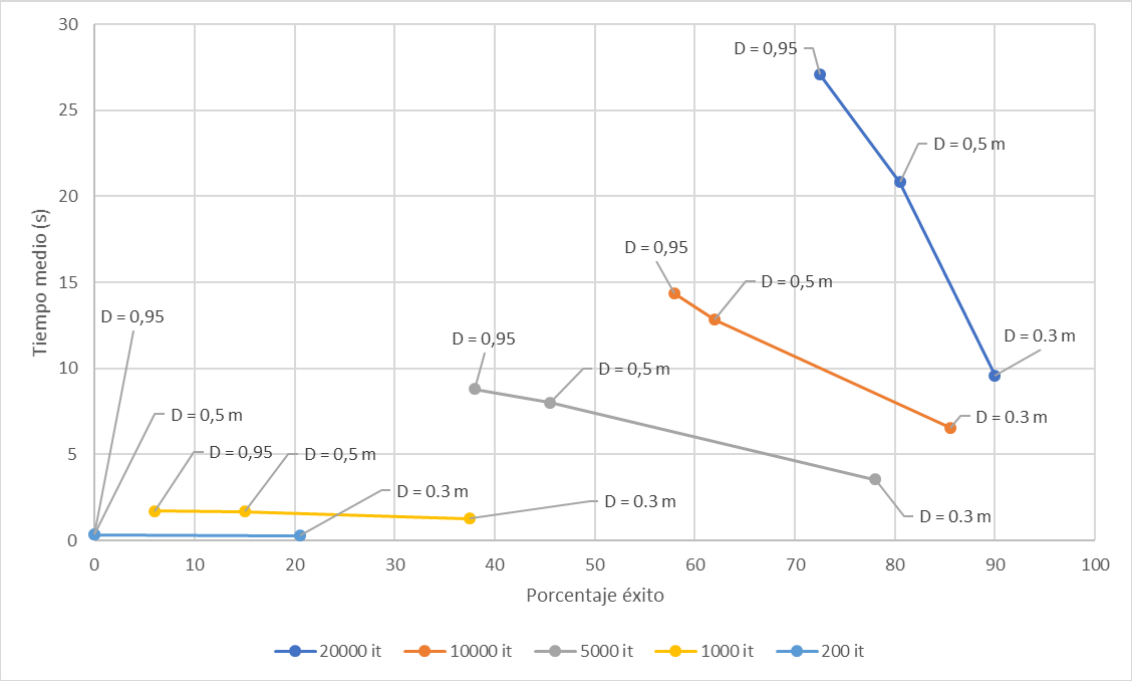

FIGURA 36 RELACIÓN TIEMPO MEDIO/ÉXITO, ITERACIONES (NO OBST) ND=0.15

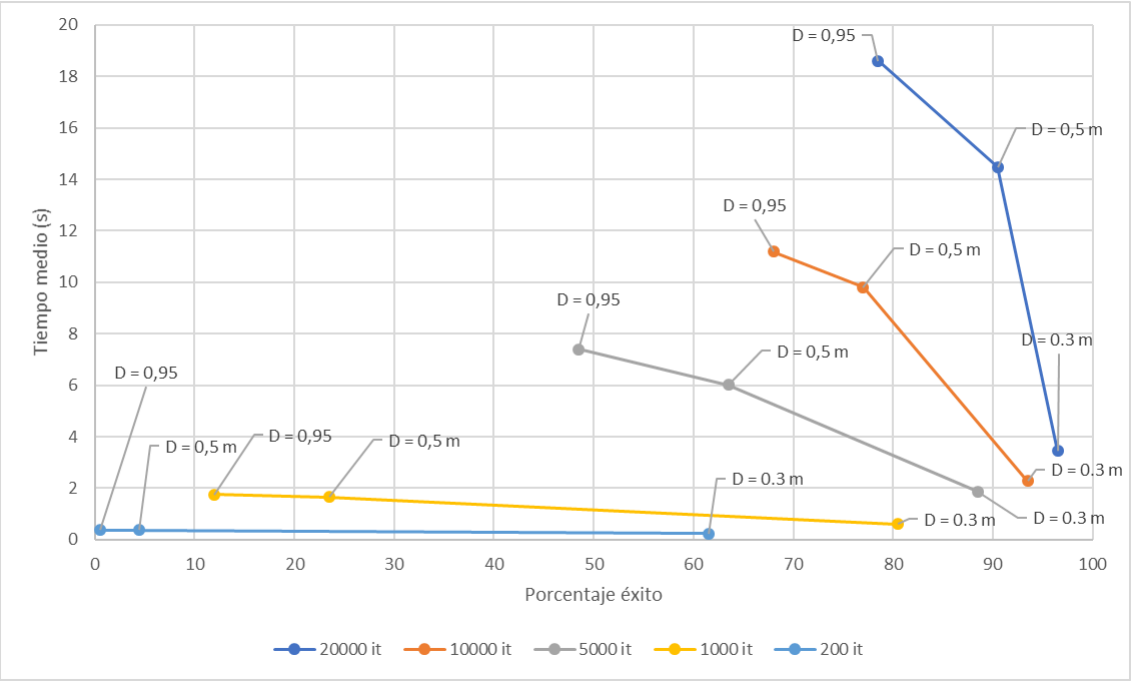

FIGURA 37 RELACIÓN TIEMPO MEDIO/ÉXITO, ITERACIONES (NO OBST) ND=0.2

Si **agrupamos por distancias**, podemos ver más claramente lo que se ha podido apreciar en apartados anteriores, es decir, que para distancias cortas los escalones tanto de tiempo como de porcentaje de éxito son menores que en los casos de media y larga distancia en los cuales estos son bastante más bruscos.

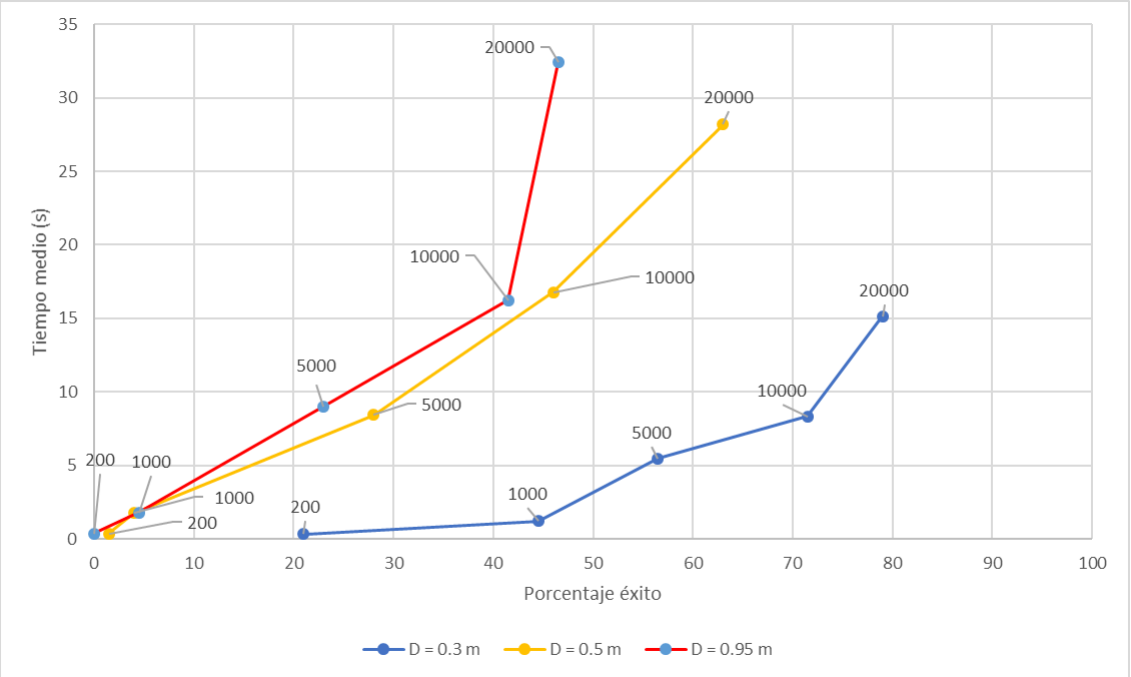

FIGURA 38 RELACIÓN TIEMPO MEDIO/ÉXITO, DISTANCIA (NO OBST) ND=0.1

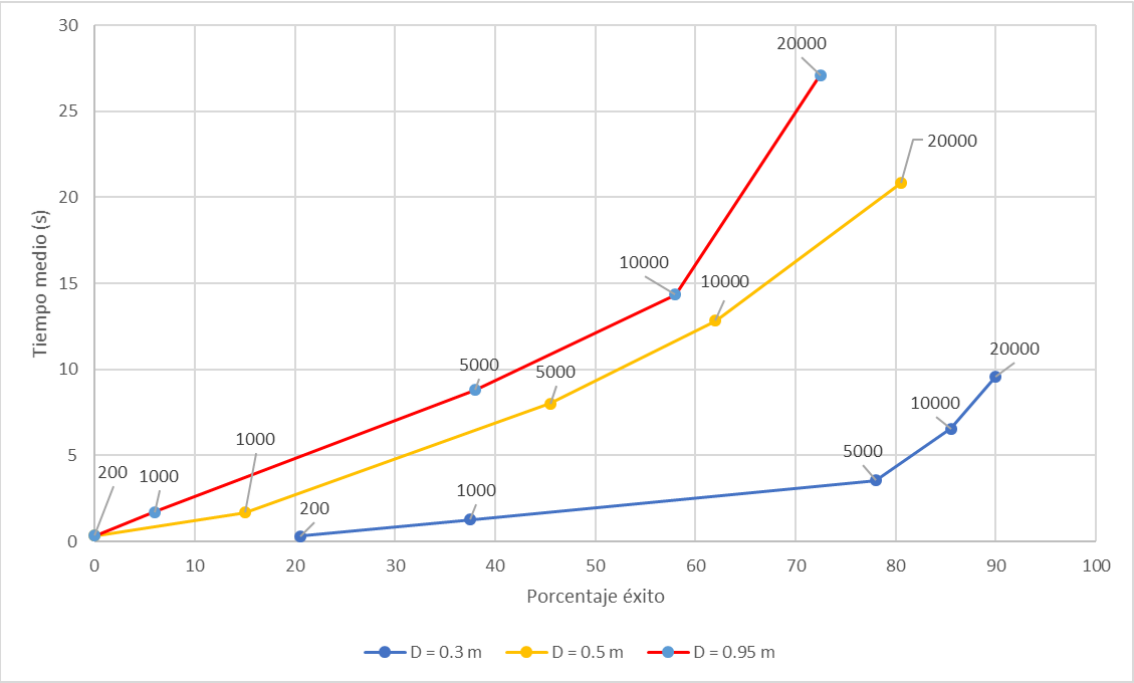

FIGURA 39 RELACIÓN TIEMPO MEDIO/ÉXITO, DISTANCIA (NO OBST) ND=0.15

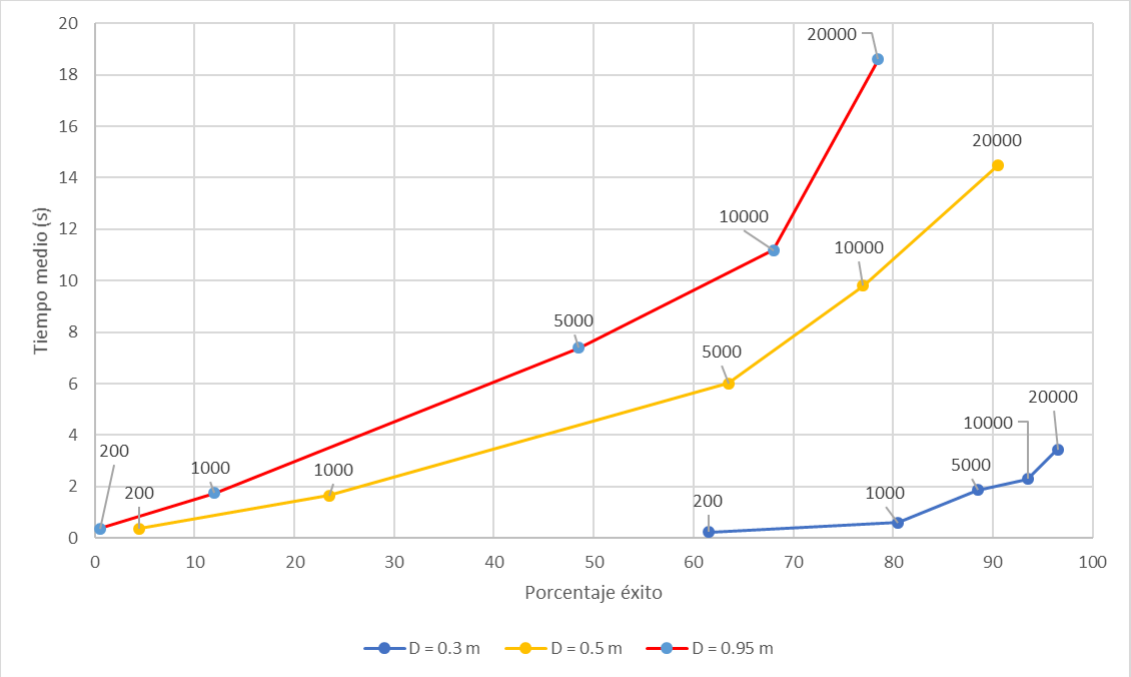

FIGURA 40 RELACIÓN TIEMPO MEDIO/ÉXITO, DISTANCIA (NO OBST) ND=0.2

4.1.3 Diferente distancia de conexión entre nodos para un mismo número de iteraciones y/o distancia entre inicio y objetivo.

Por último, va a analizarse cómo afecta el cambio en la distancia de conexión entre nodos a los resultados que nos arroja el algoritmo utilizado. Para ello nuevamente van a analizarse el porcentaje de éxito, el tiempo medio utilizado y la relación entre los pasos y el tiempo empleado.

Para ello van a utilizarse nuevamente los valores de los apartados anteriores, en concreto la distancia máxima de conexión entre nodos, *nd*, la cual tendrá unos valores de 0,1 m, 0,2 m y 0,3 m.

- Porcentaje de éxito

En las siguientes gráficas se han agrupado los valores correspondientes al porcentaje de éxito en función de *nd*, para cada una de las iteraciones, y además, esto para cada una de las 3 distancias entre punto inicial y objetivo utilizadas en apartados anteriores, es decir, 0,3, 0,5 y 0,95 metros.

Como se puede apreciar en las figuras 41 a 43, a menor distancia máxima de conexión entre nodos, el porcentaje de éxito es menor. Esto es debido, como se apreciará mejor posteriormente, a que a menor distancia de conexión, hace uso de más nodos y, por tanto, de más pasos para alcanzar el objetivo, quedándose antes sin el número de iteraciones disponibles.

También se pueden observar 2 casos concretos que llaman la atención. En primer lugar, en la primera gráfica correspondiente a la distancia corta (D = 0,3 m), tanto para 1000 como para 200 iteraciones, nd = 0,2 no disminuye de manera tan drástica de porcentaje como en los demás casos. Esto es debido a que, a diferencia del resto, y al ser la distancia tan corta, necesita de muy pocos nodos para conectar con el objetivo, manteniendo por tanto un alto porcentaje.

Como se puede ver, a priori, utilizar la distancia de conexión de 0,2m es más optima, sin embargo, en el momento que se añadan obstáculos puede repercutir en saltos grandes entre nodos, pudiendo existir entre medias obstáculos con los que colisionar. Por ello, en función del entorno, será mejor, en según qué casos, utilizar la conexión de nodos de 0,1 m que genera una trayectoria con mayor número de puntos más cercanos entre sí.

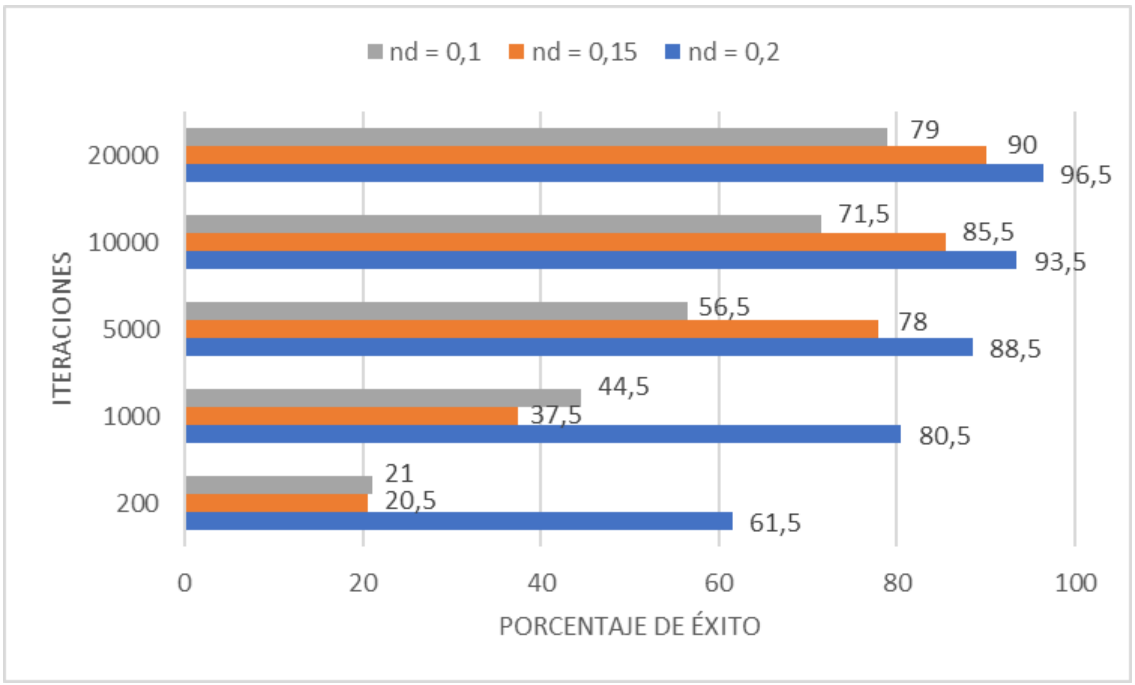

FIGURA 41 PORCENTAJE ÉXITO (NO OBST) D=0.3

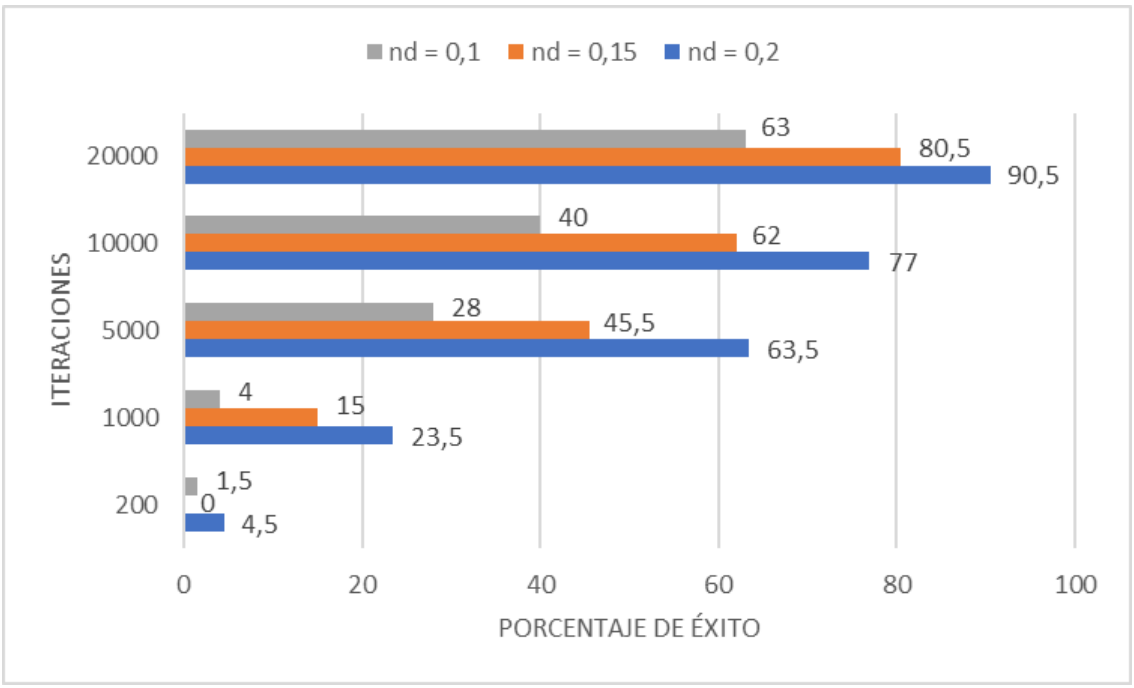

FIGURA 42 PORCENTAJE ÉXITO (NO OBST) D=0.5

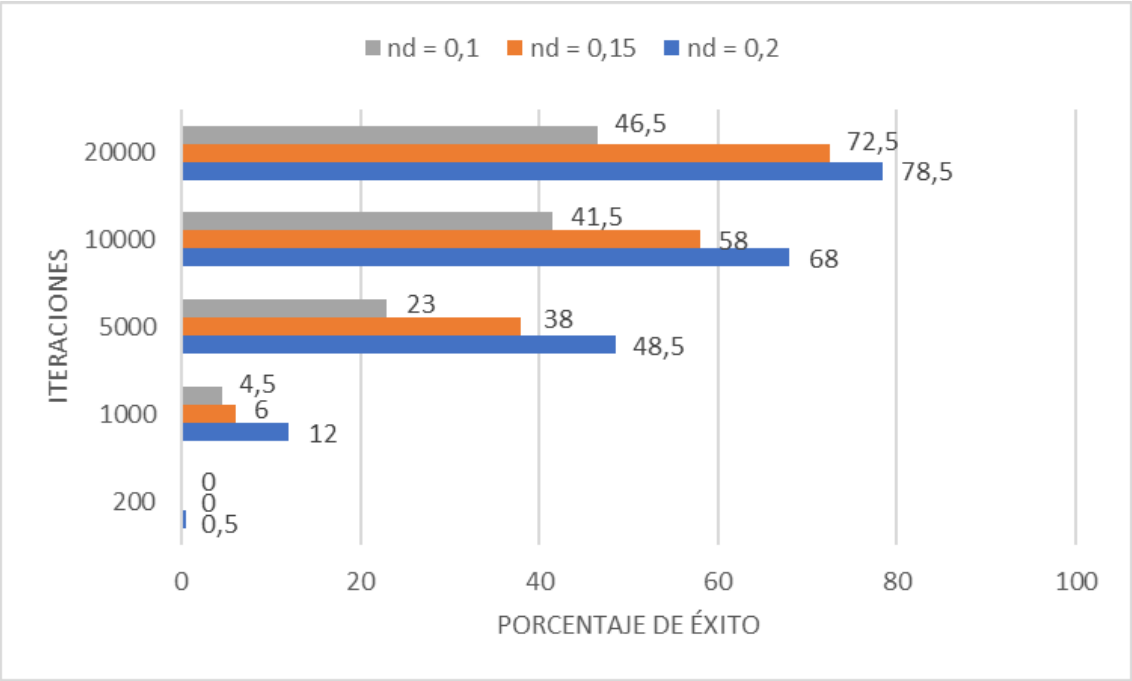

FIGURA 43 PORCENTAJE ÉXITO (NO OBST) D=0.95

- Tiempo medio por prueba

Siguiendo como en los apartados anteriores, con el tiempo medio, se puede apreciar que a mayor porcentaje de éxito dentro de un grupo para el mismo valor de unos parámetros concretos (en nuestro caso actual, mismo número de iteraciones y distancia D), obtenemos un menor tiempo de ejecución, manteniéndose por tanto la relación inversa que teníamos cuando variábamos la distancia entre punto inicial y objetivo, D.

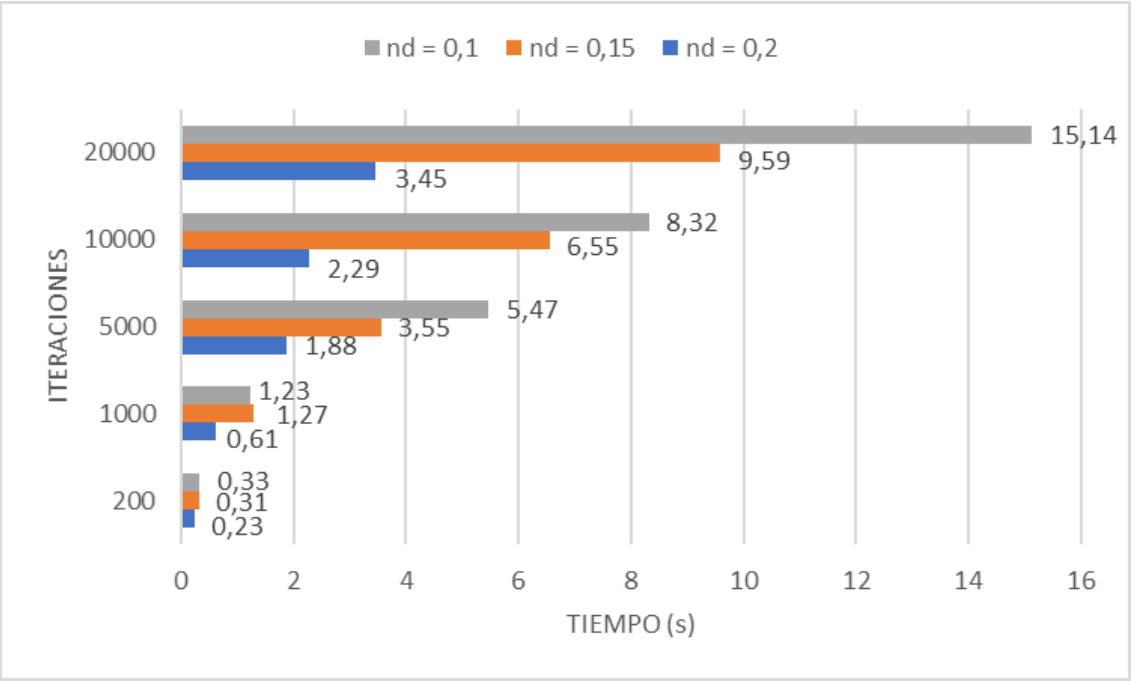

FIGURA 44 TIEMPO MEDIO (NO OBST) D=0.3

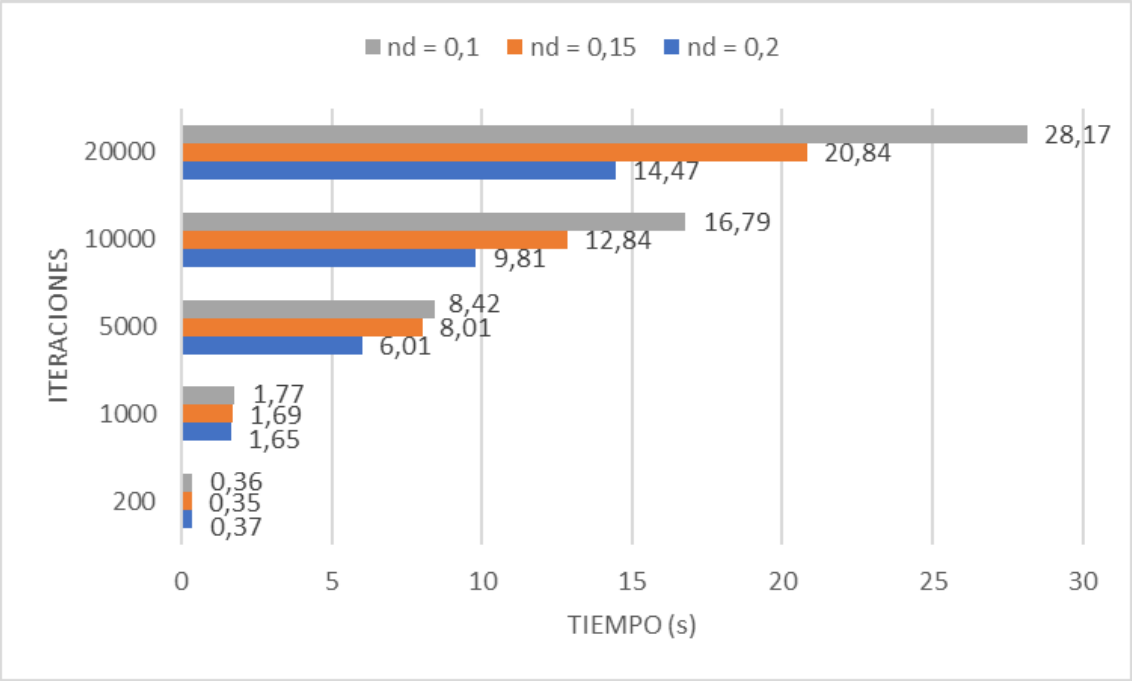

FIGURA 45 TIEMPO MEDIO (NO OBST) D=0.5

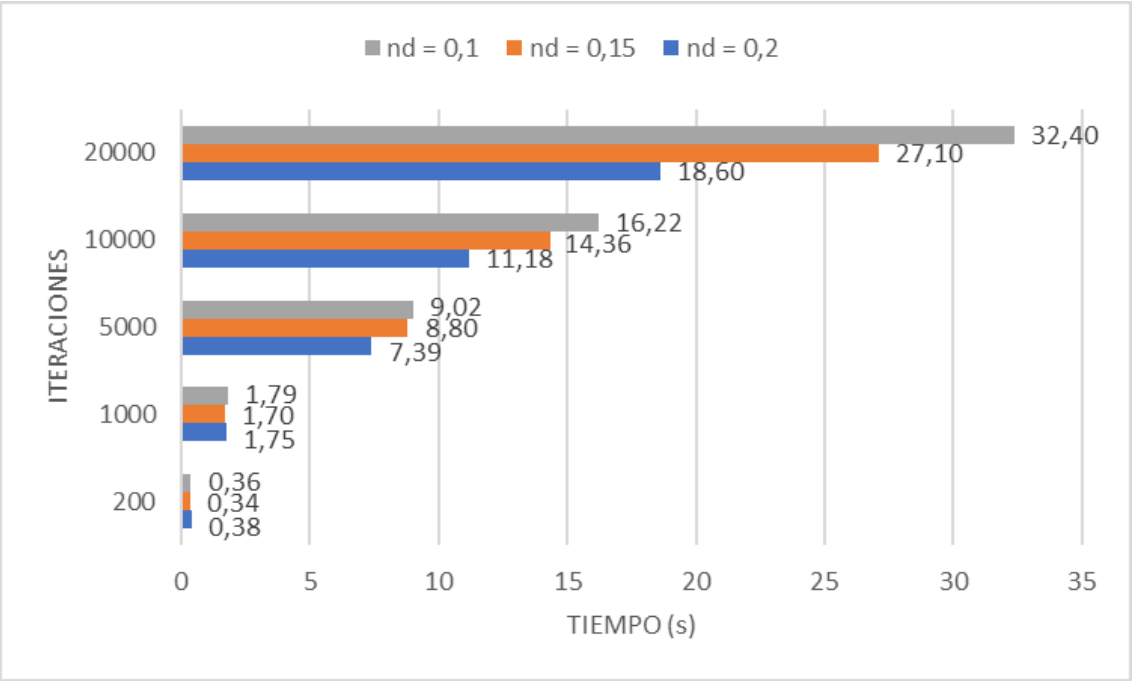

FIGURA 46 TIEMPO MEDIO (NO OBST) D=0.95

- Relación entre pasos y tiempo de ejecución

En este apartado se han incluido tres gráficas correspondientes al estudio de los pasos en el sistema. Para este caso se ha, concretado para 3 casos concretos, siendo para el resto de caso el mismo comportamiento.

En primer lugar, se puede observar una relación entre el número de pasos y el tiempo empleado para las tres distancias de conexión de nodos existentes. Como se puede ver, responden de manera exactamente igual, que se corresponde, además, con la que se pudo observar en el mismo apartado. Con ello se puede ver por tanto, que hablar de pasos y de tiempo utilizado durante la ejecución son términos equivalentes.

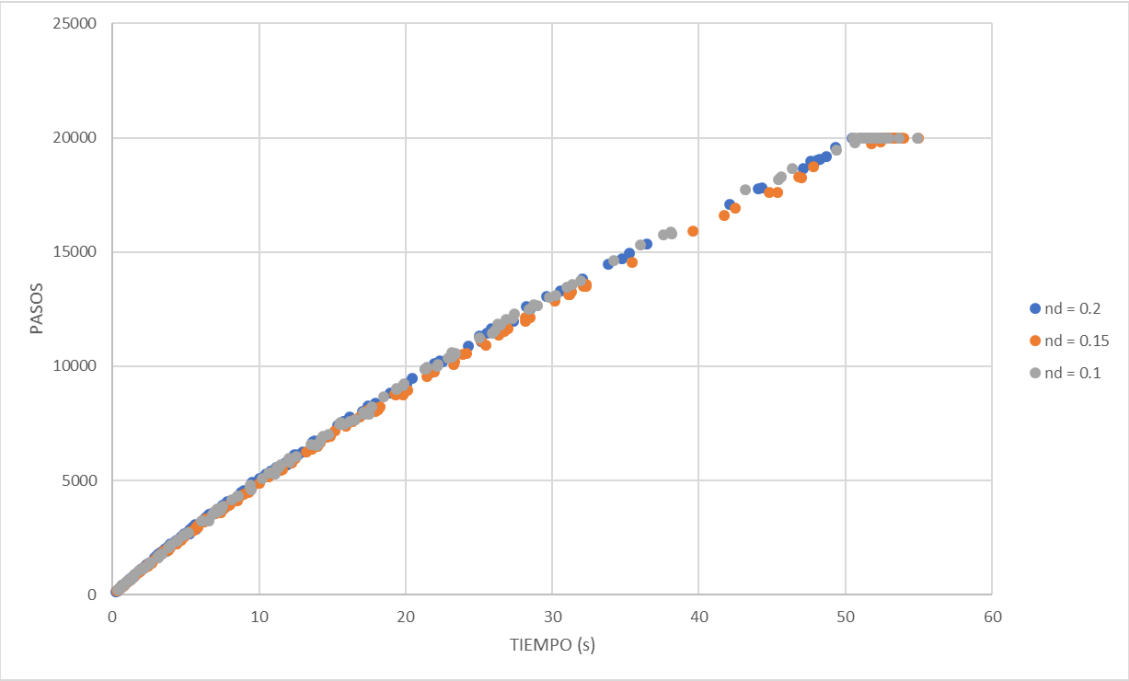

FIGURA 47 RELACIÓN PASOS/TIEMPO (NO OBST) D=0.5

En segundo lugar, se ha añadido un estudio sobre el número de pasos en función de la distancia de conexión *nd*. Como se puede apreciar, para una misma distancia, y un mismo número de iteraciones y distancia *D*, los pasos utilizados son mayores, confirmando lo dicho con anterioridad al hablar del porcentaje de éxito y el número de pasos.

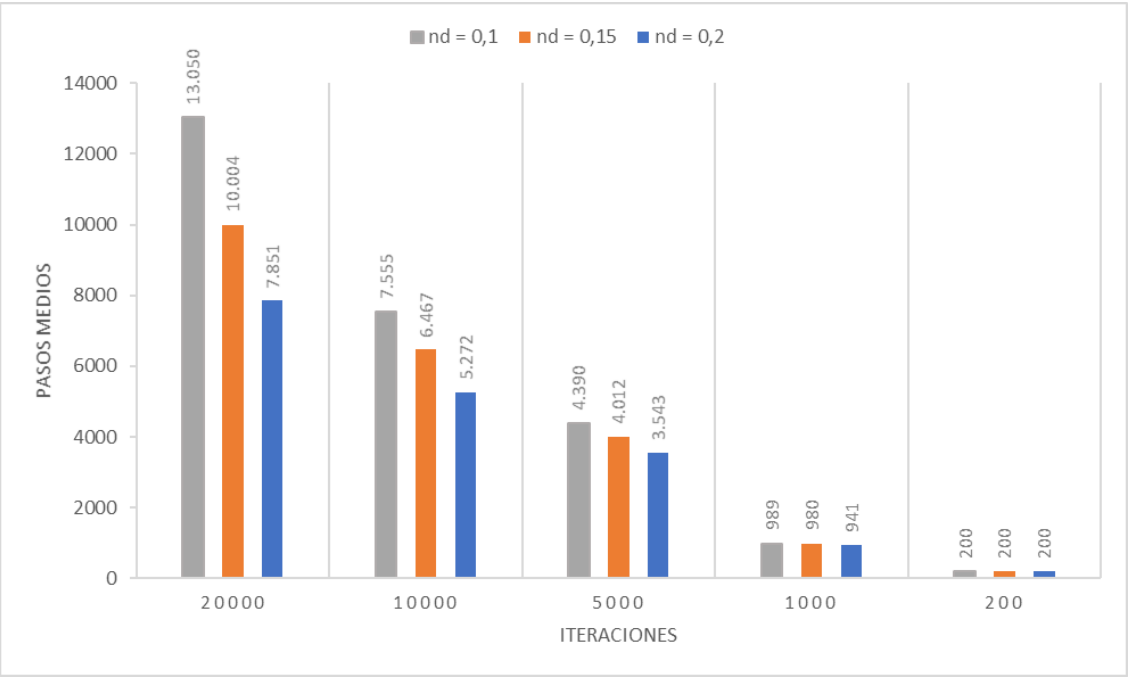

FIGURA 48 PASOS MEDIOS (NO OBST) D=0.95

Finalmente, se ha añadido una gráfica de cajas (figura 49), que nos permiten observar la distribución de los pasos utilizados en cada una de las ejecuciones para un caso concreto. En él, está representado con "X" y su valor correspondiente, la media de cada uno de los casos estudiados. Además, el límite inferior define el valor mínimo para cada grupo de iteraciones y las cajas son las zonas o rangos de pasos en los que existe mayor concentración de resultados para las distintas pruebas.

Como se puede observar, según crece el número de iteraciones los pasos se van distribuyendo en mayor rango de "posibles números de pasos utilizados durante la ejecución". En el caso de 200 iteraciones, se usan prácticamente el 100% de las iteraciones disponibles. Sin embargo, en el caso de 20000 iteraciones, se ve que la media es aproximadamente la mitad del total de iteraciones disponibles, pudiendo alcanzarse el objetivo tanto en 200 iteraciones como en 20000; estando el grueso de iteraciones utilizadas entre 4000-5000 y el máximo disponible.

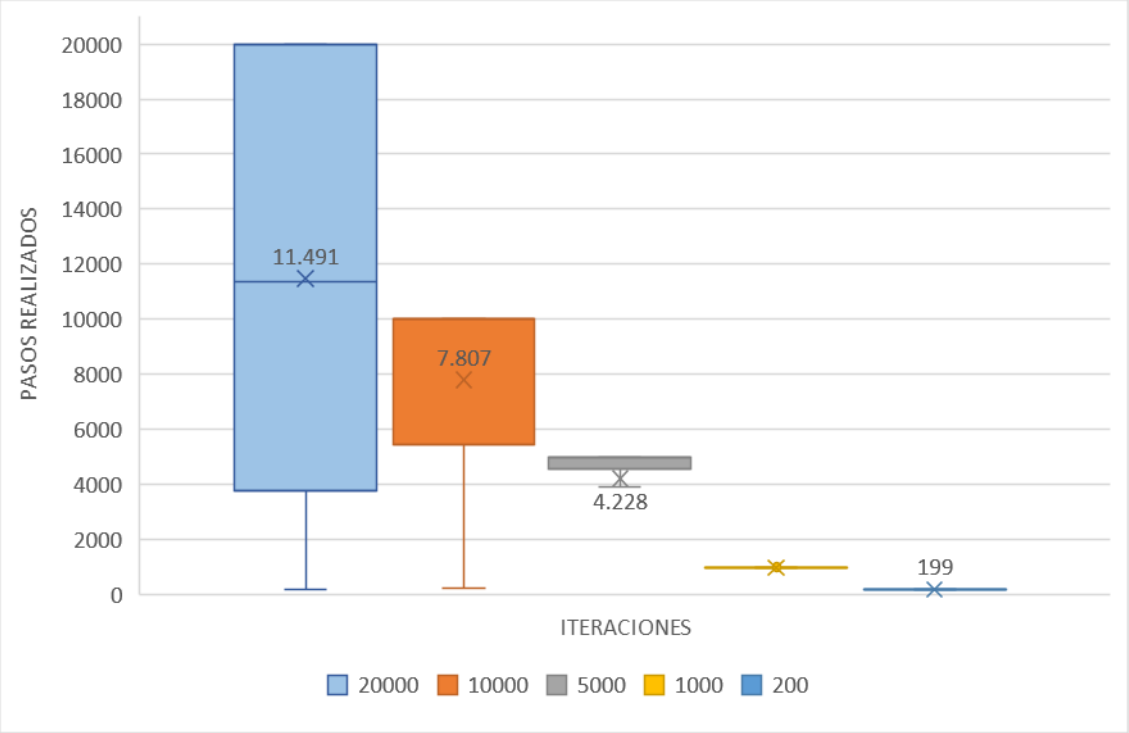

FIGURA 49 DISTRIBUCIÓN DE PASOS (NO OBST) D=0.5 ND=0.1

4.2 Con obstáculos

La diferencia principal entre las pruebas con y sin obstáculos es esa misma, que existen obstáculos y por tanto el algoritmo difiere respecto al anterior en que se le añade una nueva comprobación a que realizar (colisiones).

Como se pudo observar en la figura 18, los obstáculos añadidos a las pruebas serían la propia base del robot, una base con un trofeo encima y un muro lateral.

Por tanto, en dicho caso, tanto los parámetros distancia (entre punto inicial y objetivo final), iteraciones y distancia máxima de conexión de nodos, no suponen una diferencia respecto al caso sin obstáculos.

A continuación, se va a estudiar por tanto el porcentaje de éxito y el tiempo medio empleado para la realización de las pruebas, comparándolas con los resultados obtenidos para el escenario sin obstáculos.

- Porcentaje de éxito

En las tres gráficas siguientes (una para cada valor de *nd* utilizado, es decir, 0,2, 0,15 y 0,10m), se puede apreciar la comparación entre el porcentaje de éxito obtenido entre simulaciones sin y con obstáculos para cada una de las 3 distancias entre inicio y objetivo (D = 0,3 m, D = 0,5 m, D = 0,95 m) y para las distintas interacciones utilizadas (20000, 10000, 5000, 1000, 200).

Se han representado de azul los escenarios sin obstáculos y de naranja los escenarios con obstáculos.

Se observa, que no existen cambios en el porcentaje de éxito entre los dos escenarios, siguiendo un patrón prácticamente idéntico, calcando en algunos casos el porcentaje de éxito entre la misma prueba para obstáculos o sin ellos. Únicamente llaman la atención, los dos porcentajes en la distancia corta (D=0,3 m), para en el caso de 1000 y 200 iteraciones con *nd* = 0,2 m. En éstos se puede observar un mayor porcentaje de éxito respecto a su prueba compañera sin obstáculos. Esto únicamente es explicable debido a una anomalía estadística en el estudio, pues al pasar a *nd* = 0.1m, estos valores vuelven a igualarse al de sus compañeros sin obstáculos.

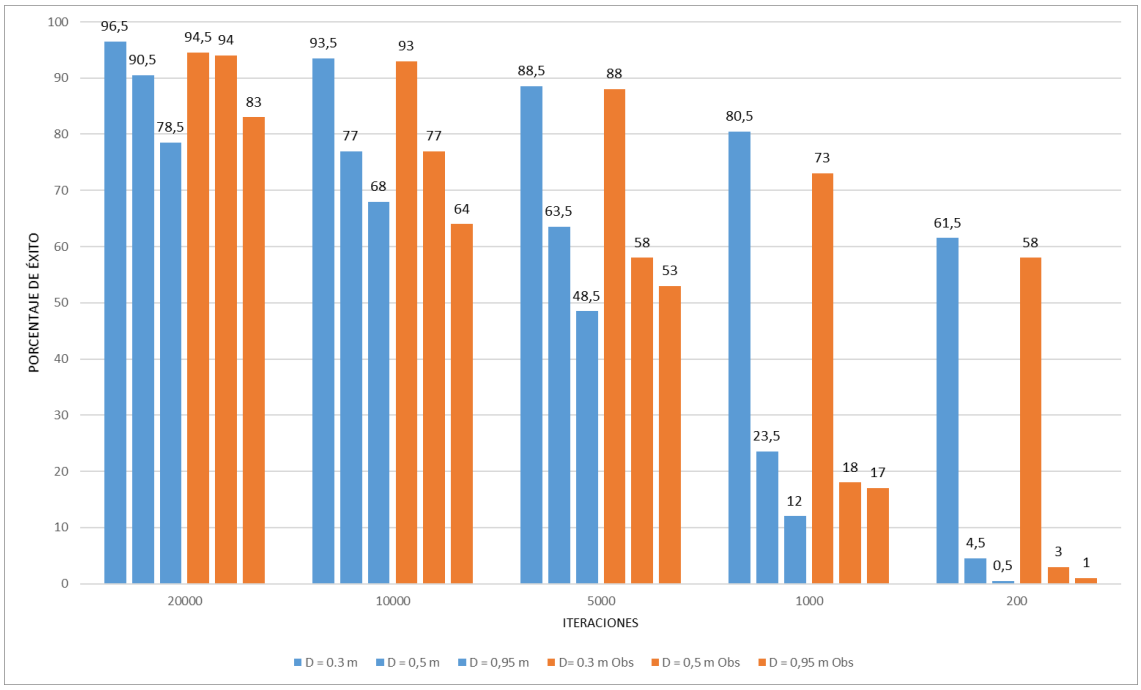

FIGURA 50 PORCENTAJE DE ÉXITO ND=0.2

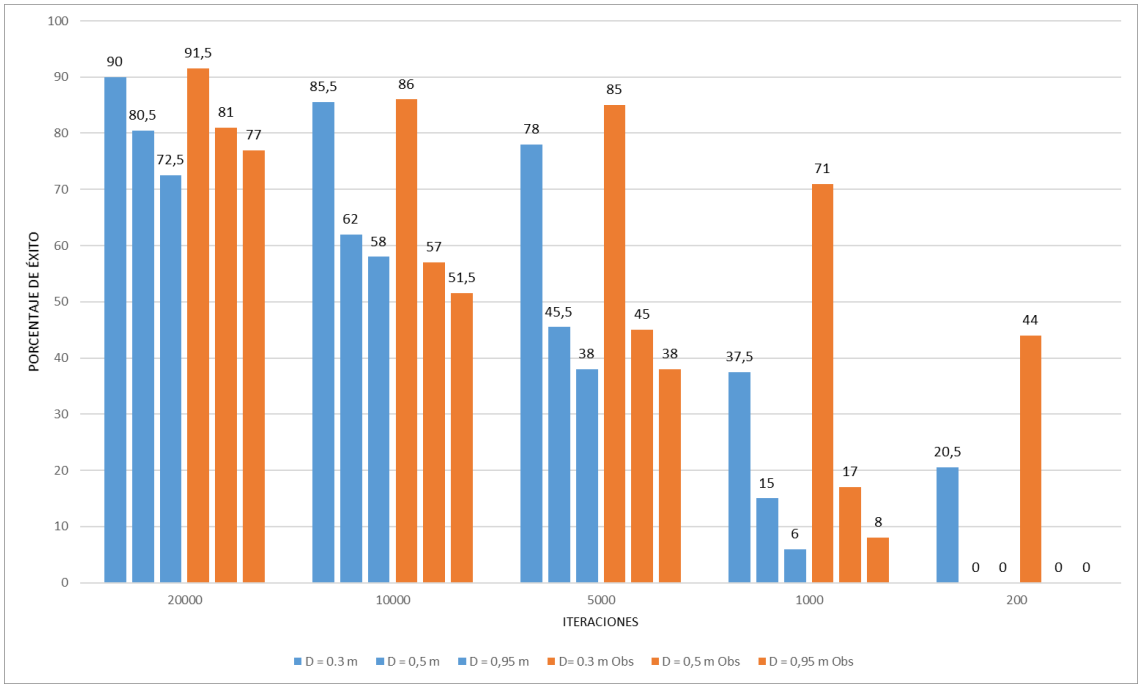

FIGURA 51 PORCENTAJE DE ÉXITO ND=0.15

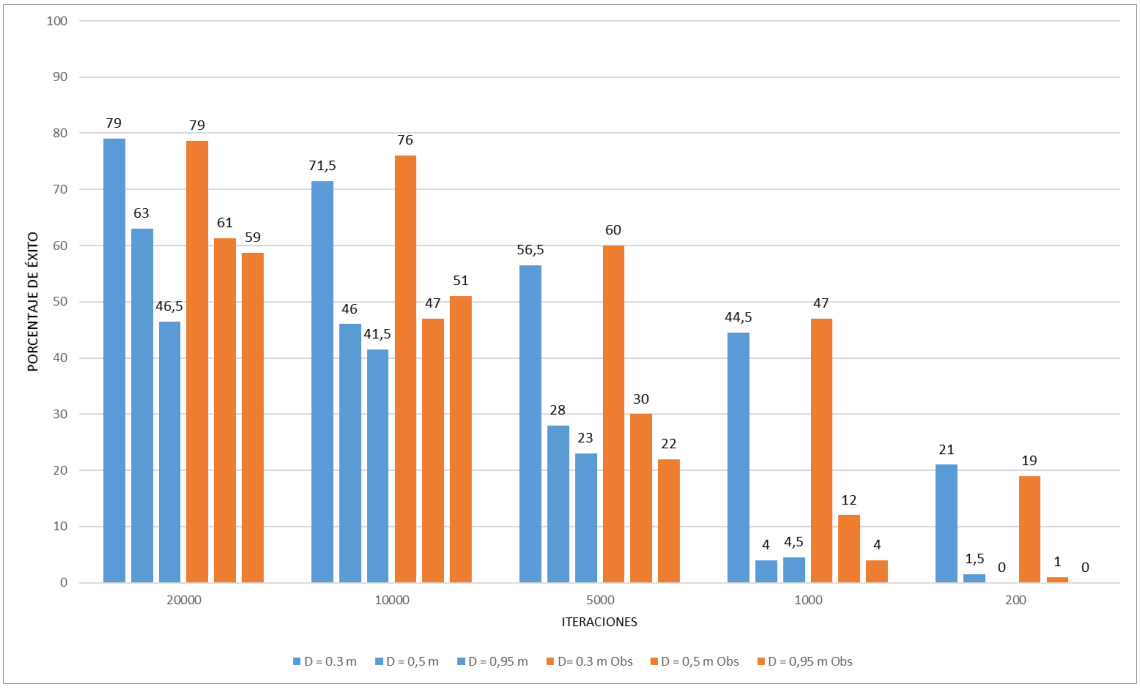

FIGURA 52 PORCENTAJE DE ÉXITO ND=0.10

- Tiempo medio por prueba

A continuación, se pueden observar tres imágenes que recogen los resultados pertenecientes al tiempo medio por prueba de ejecución, siguiendo las mismas condiciones que en el apartado del porcentaje de éxito.

En este caso, sin embargo, nos encontramos con grandes cambios respecto al apartado anterior. Como se puede comprobar, el tiempo de ejecución para el escenario con obstáculos, crece considerablemente respecto al escenario sin ellos. Este tiempo llega a crecer incluso hasta 3 o 4 veces por encima del valor de una misma prueba sin obstáculos. Esto es debido, principalmente a tres motivos.

- En primer lugar, la nueva comprobación que se realiza en el algoritmo, que se encarga de cerciorar que, para una nueva posible configuración, el brazo no está en colisión con el entorno.
- Otro de los motivos que ocasiona mayor retardo es la carga de todos los modelados correspondientes a al entorno y al propio brazo robótico en el inicio de la ejecución del algoritmo. Esta carga, lleva unos pocos segundos que terminan afectando finalmente en el tiempo final de la ejecución.
- Finalmente y relacionado con el primero de los motivos, al existir obstáculos, se restan el número de configuraciones disponibles, por lo tanto aumenta el número de vez que se debe descartar una posible configuración, aumentando por tanto el tiempo de ejecución.

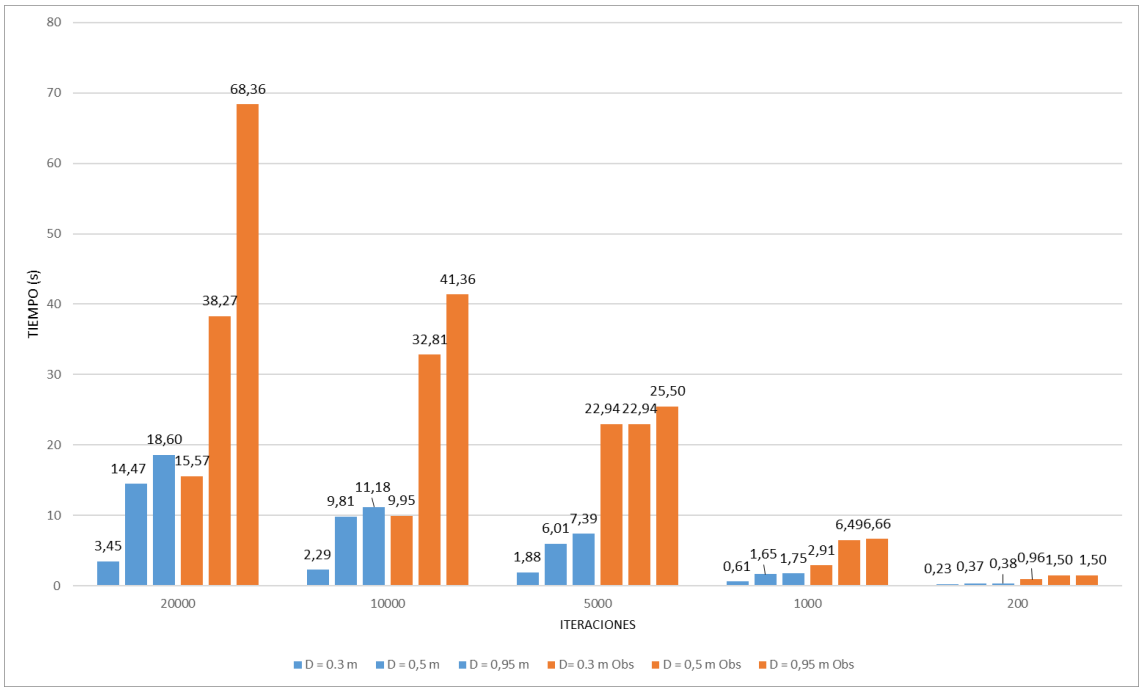

FIGURA 53 TIEMPO MEDIO ND=0.2

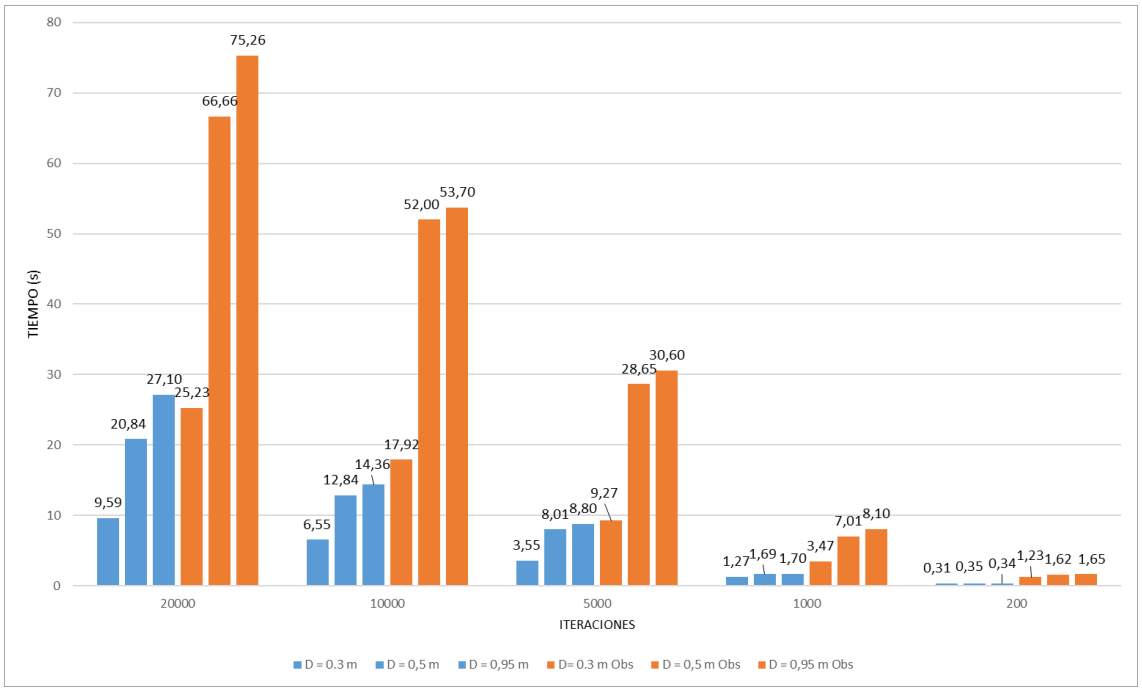

FIGURA 54 TIEMPO MEDIO ND=0.15

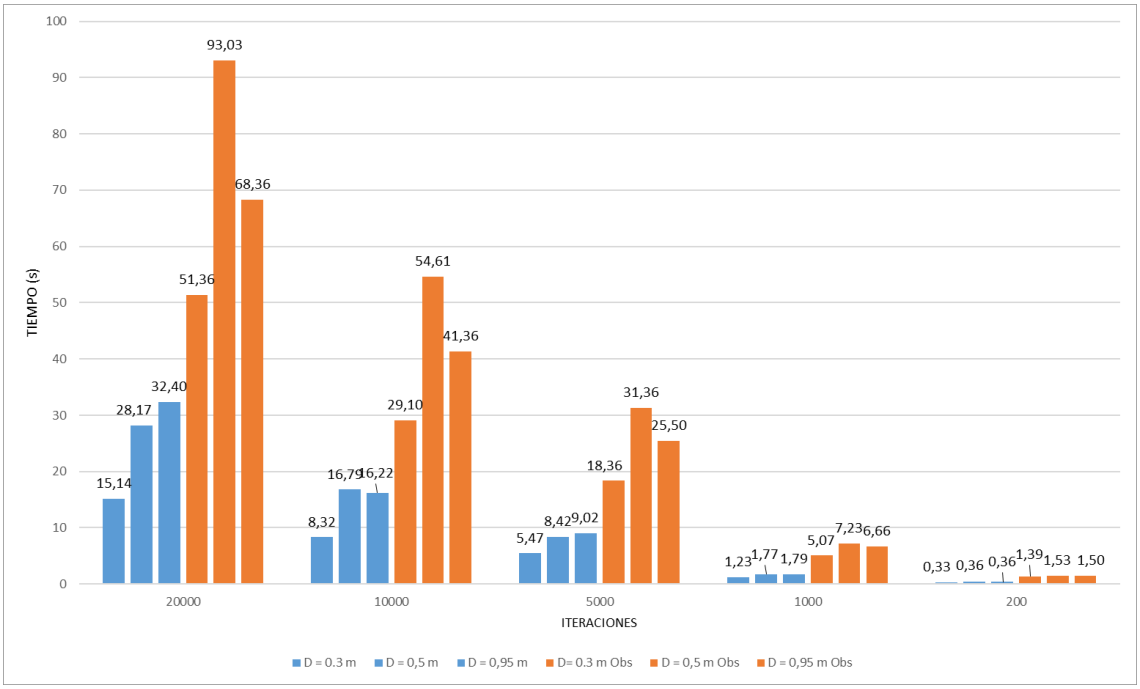

FIGURA 55 TIEMPO MEDIO ND=0.1

Finalmente, cabe comentar, que el único motivo que provocaría que aumentara aún más el tiempo de ejecución y además redujera finalmente el porcentaje de éxito, sería la adición de un obstáculo justo en medio del punto inicial y final que queremos conectar. Esto ocurriría, no por la incapacidad del sistema, si no por la necesidad de mayor número de nodos necesarios para la conexión del punto inicial y objetivo final, teniendo por tanto que subir el número de iteraciones, para obtener igualdad de porcentajes respecto a lo visto en este documento.

4.3 Otras pruebas

A continuación, se van a añadir dos escenarios de pruebas nuevos, que al no depender de la existencia o no de obstáculos, se han decidido poner en un apartado diferente. Estos escenarios son el número de árboles a utilizar durante la ejecución y el cambio de la posición inicial del robot al empezar la planificación.

4.3.1 Diferente número de árboles a utilizar

En el algoritmo RRT, uno de los parámetros a poder variar es el número de árboles usados durante la ejecución, pudiendo por tanto poder abarcar más espació de búsqueda.

En nuestro caso particular, existen ciertas ligaduras que impiden que esto se lleve a cabo. A diferencia de una búsqueda en un entorno totalmente abierto, los puntos se van buscando de manera aleatoria por todo el espacio, pudiendo ser conectados por diferentes

arboles al estar muy separados unos de otros. Para el caso que nos aplica, al ir buscando nuevos nodos mediante pequeños incrementos para asegurar que, si en un nodo no existe colisión, en el camino hacia otro nodo próximo tampoco exista, un solo árbol es capaz de conectar con facilidad estos puntos.

Se muestran los resultados obtenidos aumentando el diferente número de árboles para dos casos de los estudiados anteriormente.

- Distancia D = 0,5m, *nd* = 0,2m y 20000 iteraciones

Como se puede ver, aunque aumente el número de árboles drásticamente, tanto el porcentaje de éxito, como el tiempo medio de ejecución, como los pasos necesarios de media para la generación de la trayectoria se mantienen prácticamente invariables.

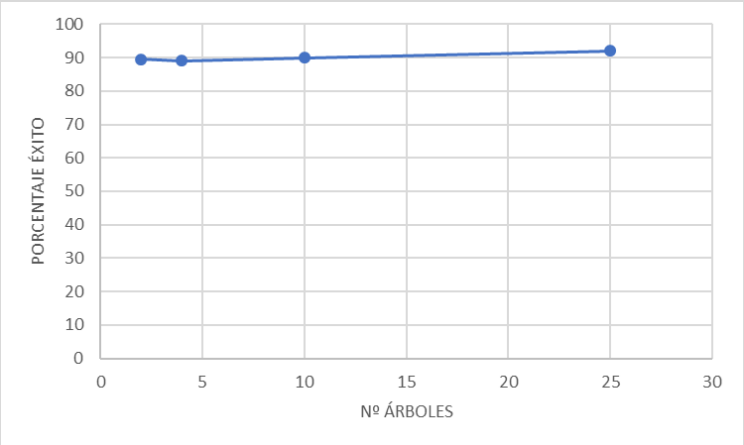

FIGURA 56 PORCENTAJE DE ÉXITO DISTINTOS ARBOLES I = 20000

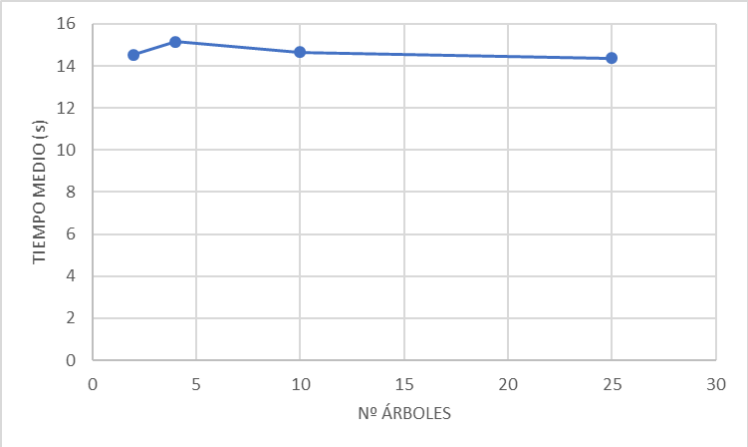

FIGURA 57 TIEMPO MEDIO DISTINTOS ARBOLES I = 20000

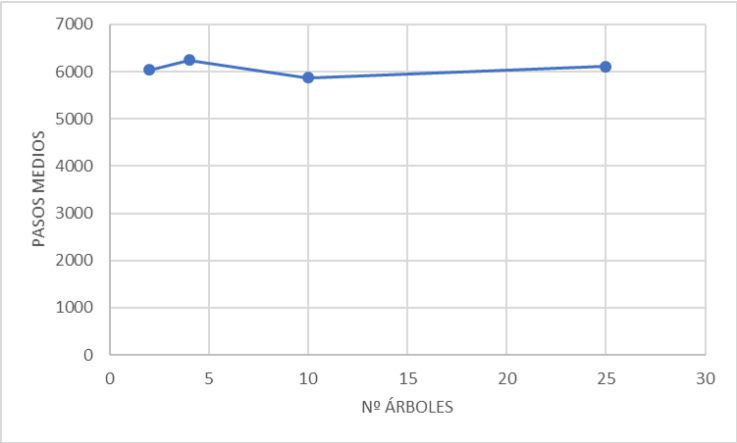

FIGURA 58 PASOS MEDIOS DISTINTOS ARBOLES I = 20000

- Distancia D = 0,5m, *nd* = 0,2m y 1000 iteraciones

Para este caso, se puede ver la misma reacción que encontrábamos para las condiciones del apartado anterior. Por tanto, se confirma que para nuestro caso concreto, aumentar o disminuir el número de árboles no afecta en absoluto.

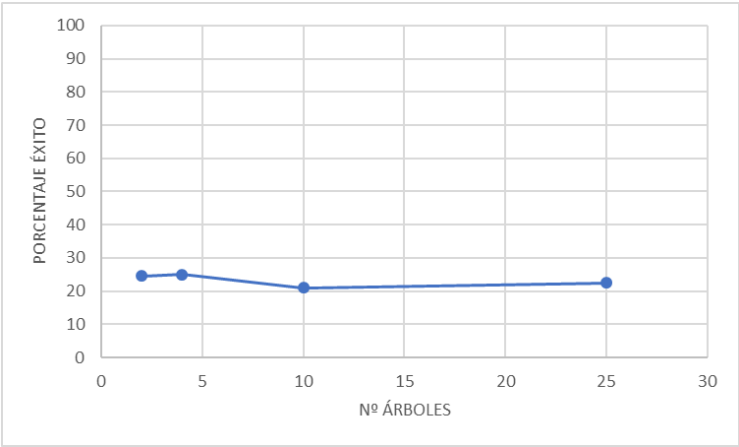

FIGURA 59 PORCENTAJE DE ÉXITO DISTINTOS ARBOLES I = 1000

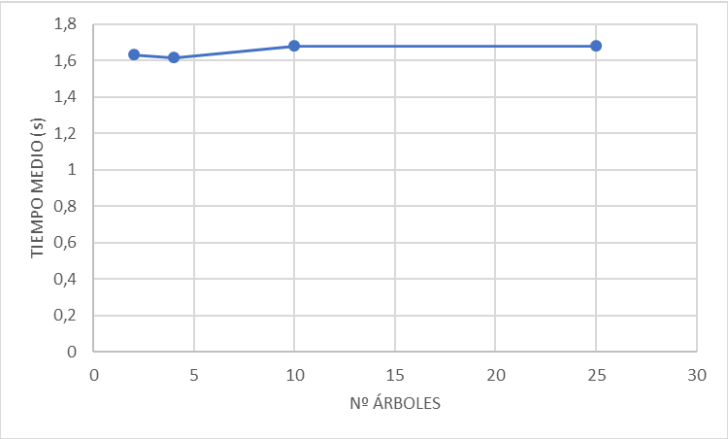

FIGURA 60 TIEMPO MEDIO DISTINTOS ARBOLES I = 1000

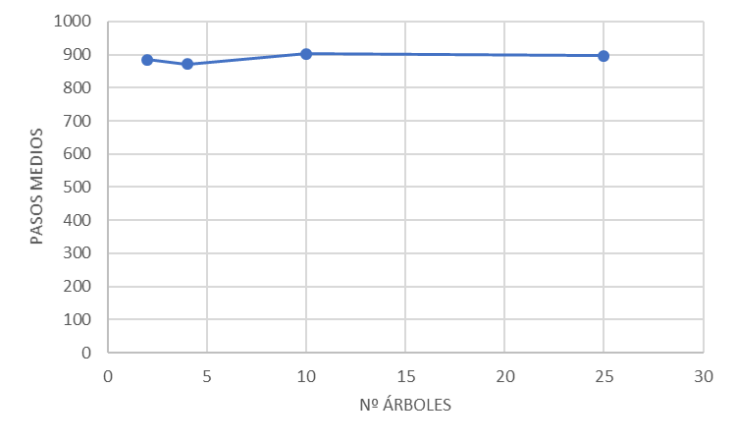

FIGURA 61 PASOS MEDIOS DISTINTOS ARBOLES I = 1000

4.3.2 Posición inicial del brazo robótico

Existe la posibilidad de que, para posiciones iniciales extremas del robot, es decir, posiciones en las cuales el brazo esté a rango máximo, el algoritmo encuentre mayores dificultades en encontrar nodos cercanos a partir de los cuales trazar una trayectoria hacia el punto objetivo. Para ello se va a realizar una prueba con el brazo totalmente estirado hacia abajo, con el fin del alcanzar los dos primeros puntos correspondientes con las distancias corta y media anteriores, D = 0,3 m y D = 0,5 m, pasando ahora a valer D = 0,87 m y D = 0,93 m respectivamente. Se puede apreciar que esta distancia es muy similar a la usada anteriormente para la larga distancia D = 0,95 m. Por tanto, los resultados obtenidos se compararán con esta.

El caso de estudio se realizará para 20000 iteraciones y distancia máxima de conexión entre nodos de 0,2 m, que es el caso que anteriormente demostraba tener mayores porcentajes de éxito.

Se representará en azul el caso estudiado anteriormente y en naranja y amarillo los casos correspondientes a la posición de inicio extrema.

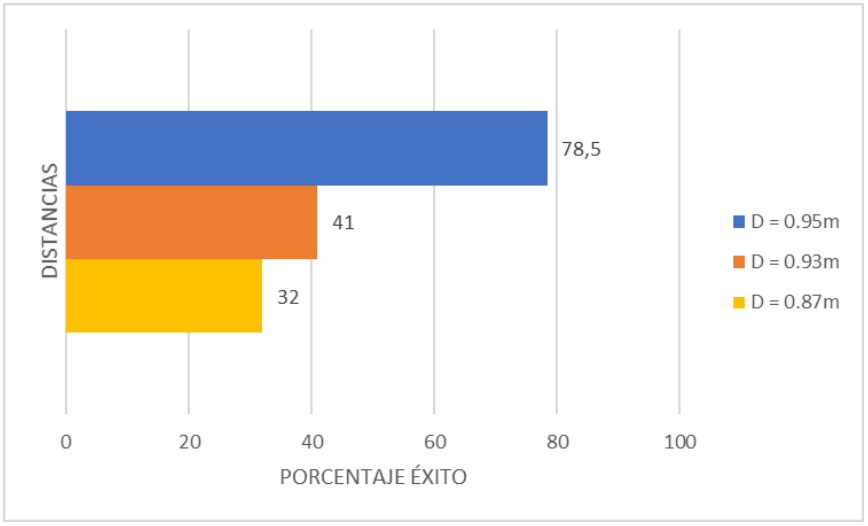

FIGURA 62 PORCENTAJE DE ÉXITO POSICIÓN INICIAL EXTREMA

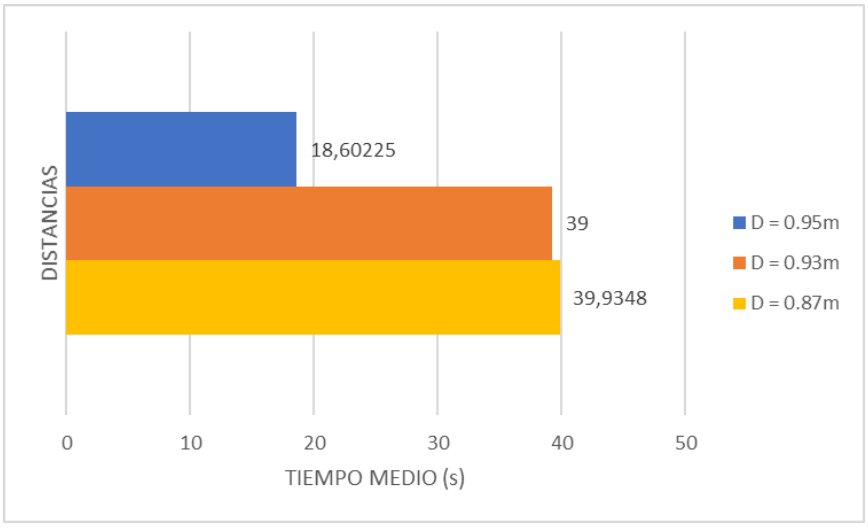

FIGURA 63 TIEMPO MEDIO POSICIÓN INICIAL EXTREMA

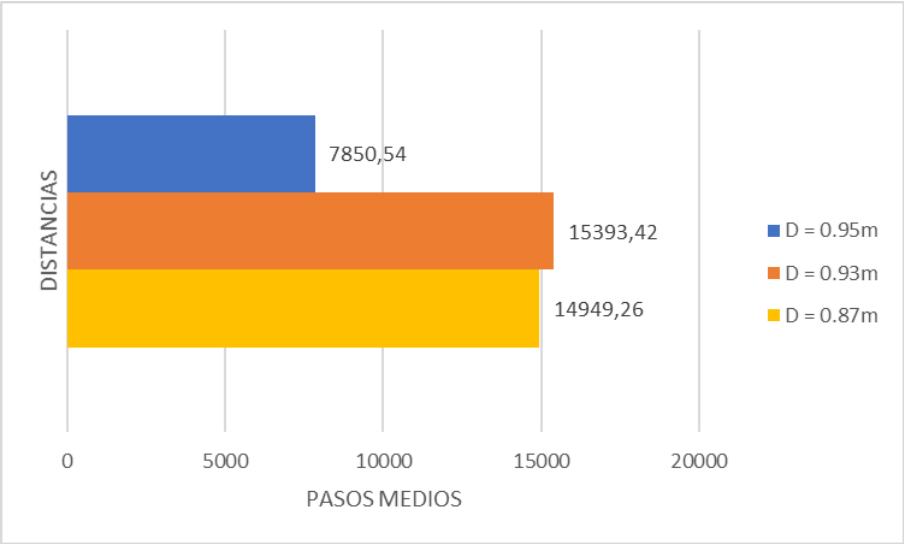

FIGURA 64 PASOS MEDIOS POSICIÓN INICIAL EXTREMA

Como se puede ver en las imágenes superiores a estas líneas, el partir de una posición extrema dificulta enormemente la ejecución del algoritmo, pues pierde muchos pasos en los momentos iniciales del algoritmo para "salir" de la posición inicial. Se puede observar que utiliza de media el doble de pasos aproximadamente durante una ejecución. Esto, repercute en el porcentaje de éxito y el tiempo, que se reducen a la mitad y se duplican respectivamente.

Por tanto, se puede concluir que cuan más extrema sea la posición inicial del brazo y, por tanto, tenga menos posibles configuraciones válidas en las proximidades para generar nodos de conexión, menor será el porcentaje de éxito, y más tiempo se empleará en la ejecución del algoritmo.

5. MARCO REGULADOR

En cuanto al marco regulador de este proyecto y documento en particular, no se considera que existan estándares técnicos o legislación aplicable sobre la implementación del proyecto descrita en este documento, a saber: riesgos profesionales, responsabilidades éticas, riesgos laborales, privacidad y seguridad, etc.

Al ser un trabajo teórico desarrollado a partir de los recursos comentados, si aplica en este caso la propiedad intelectual perteneciente a cada uno de los recursos principales utilizados para este proyecto, las cuales se desarrollan a continuación:

- *Robotics Toolbox*, se encuentra enmarcada bajo una licencia LGPL (Lesser General Public License), la cual permite la libre distribución y modificación del contenido incluido en ella, asegurando que el software es libre para todos los usuarios [13].
- pHRIWARE está licenciada bajo la LGPL, de igual modo que ocurría con *Robotics Toolbox*, asegurando que el software es libre para todos los usuarios tanto en la distribución como en la modificación del mismo[14].
- ARTE, al igual que con la toolboxes anteriores, se rige mediante la licencia LGPL (Lesser General Public License) [15].
- El planificador de trayectorias RRT, creado por Gavin Paul y Matthew Clifton [10], está protegido por Copyright (c) 2013, Gavin con todos los derechos reservados. Sin embargo, su redistribución y uso, con o sin modificación está permitido siempre y cuando se conserve el aviso de copyright anterior, tanto en el código como en la documentación adjunta al mismo.

Adicionalmente, al completar el proceso de entrega del presente Trabajo Fin de Grado, se publicará el código del proyecto con sus instrucciones de uso en un repositorio, bajo una licencia acorde, que cumpla con las licencias de todas las librerías utilizadas durante el desarrollo del proyecto, especialmente conservando la mención y el citado Copyright, perteneciente al código creado por Gavin Paul y Matthew Clifton.

6. ENTORNO SOCIO-ECONÓMICO

El trabajo de fin de grado desarrollado en este documento es un trabajo de investigación. El mismo, gira entorno la adaptación de un algoritmo para un brazo robótico de investigación.

Si hablamos del coste de realización del mismo, la licencia de Matlab para estudiantes es gratuita, así como las toolboxes utilizadas. Además, pese a simular el manipulador o brazo robótico MANFRED-2, éste no se ha utilizado y por tanto no es necesario incluirlo. Si el proyecto se hubiera trasladado al uso de MANFRED-2 habría que añadir el coste del mismo.

Además, si se hubiera realizado por una empresa, se debería incluir el coste de las licencias para estudiante como es la de Matlab.

Aun así, se podría realizar un presupuesto, si contamos las horas utilizadas por parte del autor del presente documento y del tutor del proyecto, así como el valor de los equipos utilizados para su realización.

COSTE PERSONAL	EUROS/HORA	Nº HORAS	EUROS
Fase de documentación/Investigación	10	40	400
Programación	15	50	750
Pruebas	15	190	2850
Redacción Memoria	10	110	1100
Reuniones con tutor y correccíón	50	70	3500
TOTAL		460	8600 €

TABLA 2COSTE PERSONAL

COSTE EQUIPOS	CANTIDAD	EUROS
Ordenador portátil HP Envy 13-AD102NS Intel Core i5-6200U 8 GB RAM, con Windows 10	1	900
Ordenador Sobremesa Intel Core i5 - 8600K 16 GB RAM, con Windows 10	1	1500
TOTAL		2400 €

TABLA 3 COSTE EQUIPOS

COSTE TOTAL	EUROS
Coste personal	8600
Coste equipos	2400
TOTAL	**11000 €**

TABLA 4 COSTE TOTAL

7. CONCLUSIONES Y MODIFICACIONES O MEJORAS FUTURAS

Se ha podido comprobar que mediante la elección del Algoritmo RRT se ha conseguido el objetivo de establecer una planificación de trayectorias para el MANFRED-2.

En función de diferentes parámetros como el número de trayectorias, distancia entre los puntos a establecer la trayectoria, distancia máxima de conexión, así como la posición inicial del brazo, se ha desarrollado un conjunto de pruebas tanto para entornos con obstáculos o sin obstáculos, que nos permiten entender cómo funciona este algoritmo, y que no existe una solución única para todos los casos.

En función del escenario que se presente, a pesar de existir una combinación de parámetros "ideal", es posible que sea necesario modificar alguno de ellos para adaptarse al requerimiento particular del problema al que nos enfrentemos.

Una posible mejora al sistema, podría ser la modificación de este propio algoritmo para evolucionarlo o bien a RRT* o RRT-Estrella [22], o bien a un sistema RRT-bidireccional, el cual en lugar de partir de inicio a fin como hace el RRT actual, utiliza dos árboles, uno desde el inicio y otro desde el final, generando la trayectoria una vez intersecan estos dos árboles [23].

Otra mejora sería la realización de un postprocesado. Una vez completada la trayectoria, existen métodos de postprocesado, que nos pueden permitir refinar una trayectoria, haciéndola, por ejemplo, más directa.

Uno estos métodos sería por ejemplo una función de "recableado", que como el nombre indica, se encarga de una vez alcanzado el objetivo mediante el algoritmo, vuelve a comprobar el camino por si existen conexiones de menor coste de las generadas en el trazado inicial de la trayectoria entre punto de origen y punto objetivo. Como punto negativo, tiene que obtendríamos una solución en mayor tiempo, al necesitar mayor carga computacional.

Algo parecido realiza el algoritmo que se ha adaptado para la planificación del RRT, es decir una vez finalizada la planificación, realiza un suavizado, que consiste en simplificar la trayectoria a pocos segmentos. Sin embargo, este suavizado es excesivamente drástico, provocando en la mayoría de ocasiones que una directamente punto inicial y objetivo con un solo segmento. Pese a que pueden proporcionar una trayectoria más corta, en el caso que nos aplica, esta trayectoria podría no ser tan optima, incluso ser irrealizable. Esto se debe a la posición del brazo robótico para un punto concreto que para la solución inicial era válida, pero al hacer el "recableado" o el "suavizado" puede presentar colisión o imposibilidad articular. Para ello habría que definir unas restricciones adecuada a estas circunstancias, añadiendo mayor complejidad y por tanto mayor tiempo de ejecución al proceso.

Una función similar, que contiene el mismo nombre, utiliza el algoritmo RRT* [22]. Éste, usando una configuración de costes como los algoritmos de cuadrícula, al insertar un nodo nuevo, usa la función *Rewire* o recableado para comprobar que el nuevo nodo añadido, es realmente el que posee menor coste, respecto a los nodos anteriores añadidos.

Con este modo de actuación, se consigue que la trayectoria generada sea más óptima que con el RRT original, usándose la función de "recableado" durante la ejecución y búsqueda de la trayectoria y no una vez definida.

8. BILBIOGRAFÍA

[1] A. Shkolnik and R. Tedrake, "Path planning in 1000+ dimensions using a task-space Voronoi bias", *2009 IEEE International Conference on Robotics and Automation*, pp. 2061-2067, 2009.

[2] L. Knispel, R. Matousek. *A Performance Comparison of Rapidly-exploring Random Tree and Dijkstra' s Algorithm for Holonomic Robot Path Planning*, Institute of Automation and Computer Science, Brno University of Technology, Faculty of Mechanical Engineering, 2013.

[3] J.V. Gómez *Fast Marching Methods in path and motion planning: improvements and high-level applications* (Tesis Doctoral), Universidad Carlos III de Madrid, 2015.

[4] T. Asano, T. Asano, L. J. Guibas, J. Hershberger, and H. Imai, "Visibility of Disjoint Polygons", *Algorithmica*, vol. 1, no. 1, pp. 49-63, 1986

[5] B. Delaunay, "Sur la Sphère Vide", *Bulletin of Academy of Sciences of the USSR*, vol. 7, pp. 793-800, 1934.

[6] P.Corke, Robotics, "Map-Based Planning", en *Vision and Control, Fundamental Algorithms In MATLAB*, 2011, pp. 97

[7] P.Corke, Robotics, "Map-Based Planning", en *Vision and Control, Fundamental Algorithms In MATLAB*, 2011, pp. 95-97

[8] J. Barranquand, B. Langlois, and J. C. Latombe, "Numerical Potential Field Techniques for Robot Path Planning*", IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 2, pp. 224-241, 1992.

[9] P.Corke, Robotics, "Map-Based Planning", en *Vision and Control, Fundamental Algorithms In MATLAB*, 2011, pp. 99

[10] G. Paul, M. Clifton Multiple, *Rapidly-exploring Random Tree (RRT)*, Oct 2013.[En linea]. Disponible en https://es.mathworks.com/matlabcentral/fileexchange/21443-multiple-rapidly-exploring-random-tree-rrt . Última visita en Septiembre 2018.

[11] J. Denavit, R.S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices". *Trans ASME J. Appl. Mech*, vol 23, pp. 215–221, 1955.

[12] A. Barrientos, L.F. Peñín, C. Balaguer, R. Aracil, "Cinemática del robot" en *Fundamentos de Robótica*, Ed 2, 1997, pp. 93-103.

[13] P. Corke, *Robotics Toolbox for MATLAB*, Release 10, 2017 [En linea]. Disponible en http://petercorke.com/wordpress/toolboxes/robotics-toolbox#Documentation. Última visita en Agosto 2018.

[14] “pHRIWARE” Toolbox, Jul 24, 2014 [En linea]. Disponible en https://code.google.com/archive/p/phriware/. Ultima visita en Agosto 2018.

[15] M. Hernández, “ARTE” Toolbox, 2012. [En linea]. Disponible en https://arvc.umh.es/arte/index_en.html. Ultima visita en Agosto 2018.

[16] D. López, F. Gómez-Bravo, F. Cuesta, A. Ollero, *“*Planificación de trayectorias con el algoritmo RRT, aplicación a robots no holónomos*”*, *Revista Iberoamericana de Automática e Informática Industrial*, Vol. 3, Núm. 3, Julio 2006, pp. 56-67.

[17] M. Klingensmith, *“*Overview of Motion Planning*”*. [En linea]. Disponible en *https://www.gamasutra.com/blogs/MattKlingensmith/20130907/199787/Overview_of_Motion_Planning.php*. Ultima visita en Septiembre 2018.

[18] L. DalleMole, V. Renata, A. Araújo, *The Self-Organizing Approach for Surface Reconstruction from Unstructured Point Clouds*, Federal Technologic University of Paraná, Federal Institute of Pernambuco, Federal University of Pernambuco, April 2010.

[19] D.Dimitrova, M.Brogle, T.Braun, G.Heijenk, N.Meratnia, “Joint ERCIM eMobility and MobiSense Workshop”, *IEEE Journal on Selected Areas in Communications - JSAC*, pp. 1651-1656, January 2012.

[20] Kavraki, L.E., P. Svestka, J.-C. Latombe, M.H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces*”*, *IEEE Transactions on Robotics and Automation*. Vol. 12, No. 4, pp. 566-580, Aug 1996

[21] P.Corke, Robotics, “Robot Arm Kinematics”, en *Vision and Control, Fundamental Algorithms In MATLAB*, 2011, pp. 137-143

[22] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, M.S. Muhammad, *“*RRT*-SMART: A Rapid Convergence Implementation of RRT*”, *International Journal of Advanced Robotic Systems*, June 2013.

[23] A. Hussain, Y. Ayaz, “Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments”, *Elsevier Journal of Robotics and Autonomous Systems*, Vol 68, pp. 1-11, March 2015.

[24] P.Corke, Robotics, “Map-Based Planning”, en *Vision and Control, Fundamental Algorithms In MATLAB*, 2011, pp. 102