Universidad Zaragoza
1542

Trabajo Fin de Máster

NAVEGACIÓN EN FORMACIÓN DE UN SISTEMA MULTI-ROBOT

MULTI-ROBOT NAVIGATION WITH SYSTEM FORMATION

Autor/es

Bruno Manuel Gallán Farina

Director/es

Carlos Sagüés Blázquiz

EINA
2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Bruno Gallán Farina,

con nº de DNI 25203585R en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Máster, (Título del Trabajo) Navegación en formación de un sistema multi-robot,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 19/11/2018

Fdo: B Gallán

Resumen

Es seguro decir que la robótica en general es una buena inversión de futuro. Existen muchos tipos de robot con los que solucionar problemas, especialmente los de carácter más repetitivo. Cada vez es más frecuente encontrar sistemas en los que varios robots del tipo que sea colaboran entre ellos. Esta nueva estrategia permite abordar problemas más grandes y complicados.

Entre todo el océano de posibilidades que ofrecen los sistemas multi-robot, este proyecto de fin de máster se centra en la navegación en formación de un conjunto de robots tipo "TurtleBot". El objetivo principal consiste en que un grupo de cualquier número de robots navegue en formación sin separarse. En el caso de que sea necesario, los robots se situarán en fila y recuperarán la formación cuando sea posible.

Se pretende que la implementación se realice en el marco del sistema operativo ROS, que está pensado para sistemas robóticos. Este sistema operativo es cada vez es más usado en la industria y en la investigación, podría decirse que es ya estado del arte. A priori parecen unos objetivos sencillos, pero conforme se va profundizando, van apareciendo una serie de problemas a resolver. A modo de síntesis, se procederá de la siguiente manera:

Primero se propondrán unas estructuras que permitan controlar donde se sitúan los robots dentro de la formación. Una vez construidas dichas estructuras, el siguiente paso es llevar de manera ordenada a cada robot a sus posiciones correspondientes. Con los robots ya colocados en sus posiciones, habrá que buscar alguna estrategia para que los distintos robots no se pierdan durante la navegación.

En cualquier momento puede ser necesario situar a los robots en fila, por consiguiente, las estructuras que controlarán la formación y la fila deberán de ser capaces de generarse y destruirse en cualquier lugar del mapa donde se encuentren los robots.

Una vez planteadas las estrategias comentadas anteriormente, se implementa una simulación con ROS y Stage. En principio se proponía usar Gazebo, otro simulador mucho más bonito visualmente, pero, debido a sus altas exigencias sobre los recursos del ordenador obligaron a sustituirlo por Stage, que es mucho más sencillo.

Finalmente, con los robots ya implementados en ROS y los algoritmos corriendo en Matlab, se realizan varias simulaciones sobre Stage para comprobar el buen funcionamiento de las estrategias modificando algunos parámetros.

Índice

Contenido

Índice de figuras

1 Introducción

1.1 Motivación y objetivos

A pesar de que parte de la sociedad ve a los robots con cierta inquietud, entre otras razones debido a motivos laborales y las películas de ciencia ficción, estos han sido, desde siempre, de interés para la investigación y la industria. Su utilidad es innegable, son capaces de automatizar y agilizar procesos sistemáticos, posibilitar la ejecución de tareas costosas, e incluso realizar actividades peligrosas en sustitución del ser humano.

Gracias en gran medida al abaratamiento y mejora de los procesos de producción, es cada vez más frecuente encontrar configuraciones donde un elevado número de robots colabora para realizar una determinada tarea. A este tipo de sistemas se les conoce comúnmente como multi-robot o swarms.

Este tipo de estructuras plantea nuevas ventajas e inconvenientes bastante interesantes desde el punto de vista de la investigación en disciplinas como la teoría de control. Cabe destacar la posibilidad de escalar el sistema al tamaño del problema con solo variar el número de robots utilizados y la posibilidad de especializar a cada robot para la realización de una tarea específica.

Actualmente, la investigación que se genera en torno a este tipo de sistemas consiste mayoritariamente en el desarrollo de nuevas aplicaciones. Cada grupo investigador trabaja con sus propios robots para desarrollar sus proyectos concretos, lo cual implica que existen en el mundo una gran cantidad de algoritmos distintos.

Entre las aplicaciones más interesantes que se desarrollan para este tipo de sistemas multi-robot, destacan especialmente el control de formaciones, seguimiento, evitación de obstáculos, cobertura de una zona determinada, apoyo a infraestructuras, tareas de búsqueda y rescate cooperativo, actividades relacionadas con el entorno militar y protección, e incluso exploración espacial.

Este trabajo se centra en situar a los robots en formación y mantenerla durante su recorrido. En el caso de que dicha formación no sea capaz de atravesar un determinado obstáculo o pasillo, los robots se situarán en fila para intentar continuar. Los robots que se utilizan para la simulación están controlados por el sistema operativo ROS (Robot Operating System), el cual está cobrando cada vez más importancia tanto para investigación como en la industria.

Debido a la manera en que dicho sistema calcula las trayectorias de los robots, se generan unos delicados problemas a la hora de situarlos en formación y su desplazamiento. En síntesis, cuando dos o más robots están relativamente cerca, es posible que ROS considere que han chocado o haya problemas a la hora de calcular las trayectorias.

Dicho lo cual, los objetivos del presente trabajo de fin de máster son:

- Desarrollar un método que permita situar y mantener a un conjunto de robots en formación.
- La formación debe de ser capaz de situarse en fila y reconstruirse en el caso de que sea necesario para atravesar un obstáculo o pasillo.
- Implementar una simulación con los robots controlados por ROS.

1.2 Estado del arte

Cubrir todo el estado del arte sobre la robótica y los sistemas multi-robot en una única sección de esta memoria es imposible. Así que, a continuación, se ofrece una visión muy general y se particulariza a los trabajos más similares a este proyecto.

Existen muchas aplicaciones distintas, sin embargo, a la hora de su diseño e implementación, destacan sobre todo dos métodos bien diferenciados. Uno de ellos consiste en la ejecución centralizada del algoritmo, donde un único robot u otro dispositivo realiza los cálculos y envía las órdenes correspondientes a cada robot, mientras que el resto envían los datos de sus sensores al líder.

El segundo método de implementación presenta bastante más dificultad, consiste en desarrollar los algoritmos de manera distribuida, es decir, que todos los agentes del sistema calculan, se comunican e intercambian ordenes entre sí. Entre otras dificultades de este método, destaca la complejidad a la hora de realizar las comunicaciones, ya que en este caso se manejan muchos mensajes de relativa complejidad simultáneamente.

Por lo general, la mayoría de los trabajos y artículos actuales desarrollan sus algoritmos según el método de implementación centralizada, por su mayor sencillez, pero muy posiblemente la vía distribuida cobrará más importancia en el futuro. A modo de ejemplo, en [1], se propone un método para gestionar una implementación distribuida. Más concretamente, se genera un líder o marco ficticio al que se referirán las posiciones de cada robot. Entonces, actualizando según se avance la posición de dicho líder o marco ficticio, utilizando información de todos los robots, se consigue mantener una formación, la cual posibilita la comunicación entre los agentes.

Al margen de estos dos métodos de implementación, no sería descabellado considerar al propio ROS como uno de los referentes del estado del arte actual. Cada vez está cobrando más importancia, ofrece una gran cantidad de funciones tanto como para simulación como para implementación en sistemas reales. El mejor lugar para encontrar información sobre ROS es su wiki [2].

Continuando por el camino del control distribuido, en [3] se propone un algoritmo para resolver la cobertura en un entorno cambiante utilizando datos locales. En este caso, para tener información precisa sobre el mapa, se construye una red de comunicación entre los robots, la cual se supone que no se rompe. En el caso de que se quisiera contemplar la posibilidad de la rotura de dicha red, se pueden incluir restricciones al movimiento de los robots como se indica en [4].

Particularizando al control de formaciones, como se recopila en [5], existen varias maneras de resolver el problema de la localización de los robots. Entre ellas destacan las divisiones de Voronoi, modelos probabilísticos o campos de potenciales.

En este proyecto se opta por las divisiones de Voronoi para repartir un área determinada como se hace en [6], donde se implementan en Matlab dos métodos para el control de formaciones. El primero consiste en construir una superficie que englobe a todos los robots. A partir de aquí, se desplaza dicha abstracción y se calculan leyes de control para que los robots continúen dentro de ella. El segundo método, que es el que en parte se sigue en este trabajo, consiste en repartir la superficie anterior mediante divisiones de Voronoi y hacer que los robots se sitúen específicamente en los centroides de cada división.

De esta manera se consigue situar a los robots minimizando la función de localización óptima propuesta en [7] para la mejor percepción de los robots. En esta referencia se discute dicha función, la cual depende de los sensores en sí y de cómo se divide la superficie. Tras lo cual, se continua con la implementación de un algoritmo de Lloyd o Voronoi iterativo para realizar la cobertura de una zona sobre la que se aplica una función de densidad o importancia. Gracias a lo que se consigue situar un conjunto de robots de manera uniforme alrededor de la parte más importante de la superficie.

1.3 Alcance

Como se ha comentado en el punto anterior, en este trabajo se pretende desarrollar un método para la navegación en formación basado en superficies divididas mediante particiones de Voronoi.

Los algoritmos se ejecutan en Matlab, los robots están controlados por el sistema operativo ROS y la simulación se realiza en Stage. En principio se planeaba usar Gazebo, centrado en entornos 3D, pero debido a problemas de rendimiento en el ordenador disponible, se decidió cambiar de simulador a uno más sencillo.

Para el cálculo de trayectorias con ROS, se han probado los métodos conocidos como A* y Dijkstra. Debido a la naturaleza del proyecto, no se han encontrado diferencias notables entre ambos.

Para la implementación de la simulación en Stage, ha sido necesario dibujar un mapa, controlando la relación pixel/metros con ayuda de Photoshop.

2 Descripción del algoritmo

En este capítulo se van a describir las funciones necesarias para realizar la navegación en formación teniendo en cuenta las particularidades que trae consigo ROS. Los problemas a abordar son los siguientes:

- Establecer un método para situar a todos los robots en formación a partir de unas posiciones aleatorias.
- Se debe poder romper y recuperar la formación cuando se tenga que superar un obstáculo que impida su avance.
- Durante la navegación, es preciso evitar que los robots tomen caminos diferentes.

2.1 Situar a los robots en formación

Resulta imprescindible generar una estructura con la que controlar la posición de los robots. Para ello se propone delimitar una superficie circular en la cual situar la formación. En los siguientes subapartados, se propone un criterio para repartir dicha superficie y cómo construirla.

2.1.1 Despliegue de la formación

La idea principal para solucionar este problema se ha extraído de [6] y [7]. Consiste en optimizar la percepción que el conjunto de los robots tiene de su entorno. La solución propuesta se basa en repartir el área que ocupa la formación mediante divisiones de Voronoi.

En líneas generales, un diagrama de Voronoi es el conjunto de particiones de un plano basadas en la distancia a ciertos puntos denominados semillas. Dicho conjunto de puntos debe ser especificado de antemano. Entonces, a cada semilla le corresponde una división que contenga a todos los puntos del plano que estén más próximos a ella que a ninguna otra.

Formalmente, siendo Q la superficie a repartir, una división de Voronoi V_i de Q se define como la subdivisión del plano cuyos puntos cumplen la condición de proximidad basada en el conjunto de semillas p:

$$V_i = \{q \in Q | \, \|q - p_i\| \leq \|q - p_i\|, \forall j \neq i\}$$

A partir de aquí, de una región de Voronoi se puede calcular su masa, su centroide y el momento polar de inercia como sigue:

$$M_v = \int_V \rho(q)\, dq$$

$$C_v = \frac{1}{M_V} \int_V q\, \rho(q)\, dq$$

$$J_{V,p} = \int_V \|q - p\|^2 \rho(q) dq$$

La función *ρ(q)* representa la densidad de la superficie. En otras palabras, indica que zonas de la abstracción son más importantes. En el proyecto que nos ocupa, el objetivo es situar a los robots en formación dentro de un área. En principio no hay ninguna zona de dicha superficie más importante que otra, por consiguiente, la función de densidad a utilizar es igual a uno.

En las figuras 1 a 3, se muestra la superficie sobre la que se sitúa la formación. Las regiones de distintos colores son las divisiones de Voronoi, mientras que los puntos en rojo son sus correspondientes centroides.

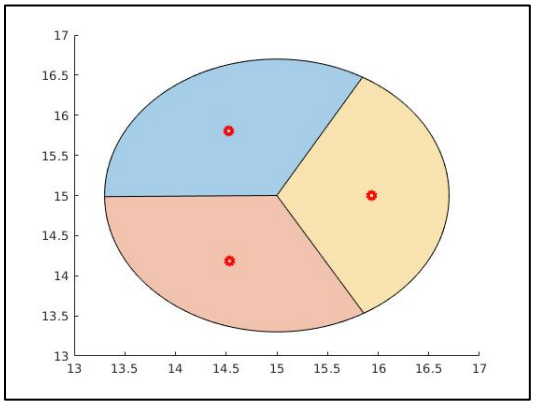

Figura 1. 3 Divisiones

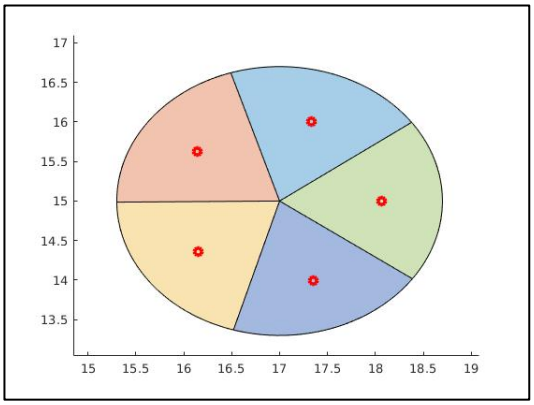

Figura 2. 5 Divisiones

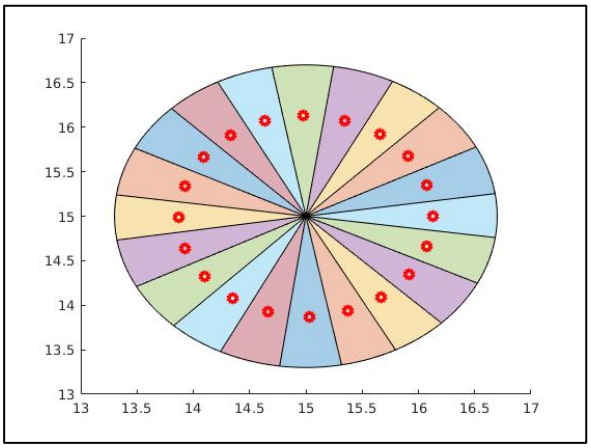

Figura 3. 20 Divisiones

A partir de aquí, se plantea la siguiente función de localización óptima:

$$H(P,W) = \sum_{i=1}^{n} \int_{W_i} f(\|q - p_i\|) \rho(q) dq$$

Donde la función *f* representa el rendimiento del sensor de los robots según la distancia. Siendo ρ(q) la función de densidad comentada anteriormente, P = {p1,…,pn} las posiciones de los robots y W se corresponde con la división de Voronoi.

Sustituyendo la función *f* por $\|q - p\|^2$ y aplicando el teorema de los ejes paralelos o de Steiner, con el que se puede recalcular el momento de inercia para otros puntos de la superficie:

$$J_{V,p} = J_{V,C_V} + M_v \|p - C_V\|^2$$

Se alcanza la siguiente expresión:

$$H_V(P) = \sum_{i=1}^{n} J_{V_i,C_{Vi}} + \sum_{i=1}^{n} M_{Vi}\|p_i - C_{Vi}\|^2$$

Derivando se obtiene:

$$\frac{\partial H_V}{\partial p_i}(P) = 2M_{Vi}(p_i - C_{Vi})$$

Para hacer dicha derivada igual a cero, las posiciones de los robots tienen que ser los centroides de las divisiones. Por consiguiente, los puntos óptimos para la localización de los robots en el área de trabajo son los centroides de las divisiones de Voronoi. En resumen, el objetivo es repartir una determinada superficie circular que delimite el movimiento de la formación y mover a los robots a los centroides de sus divisiones de Voronoi.

Respecto al cálculo de las divisiones en Matlab, una posibilidad es implementar el método explicado en [8], el cual consiste en el cálculo de la envolvente de la región de Voronoi E para poder determinar cuáles de las demás semillas son las vecinas. Se procede como sigue para calcular la región de Voronoi asociada a cada semilla:

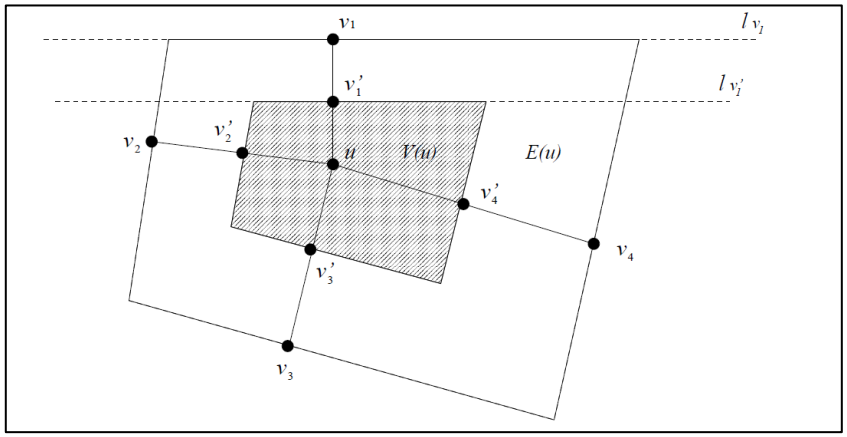

Figura 4. Cálculo divisiones de Voronoi

[1] Para la semilla u, se calcula su envolvente E_1 con la semilla que tenga más cercana, v_1, como el semiplano resultante de la intersección entre el plano de trabajo S (toda la superficie donde están las semillas) y la recta l_{v1}, que pasa por v_1 siendo perpendicular a la mediatriz del segmento que une u y v_1.

[2] Se comprueba si hay más vecinos, si los hay, se repite el mismo proceso con las siguientes semillas más cercanas.

[3] Una vez ya se han determinado todos los vecinos, se calcula la división de Voronoi como la intersección de las mediatrices mostradas en la Figura 4.

Para saber si ya se han encontrado a todos los vecinos, se define un radio como la máxima distancia entre la semilla en cuestión u y su envolvente calculada hasta ahora. A continuación, se traza una circunferencia como en la Figura 5, con centro en u y dicho radio. Si no quedan nuevas semillas dentro de esa circunferencia, ya se han encontrado a todos los vecinos.

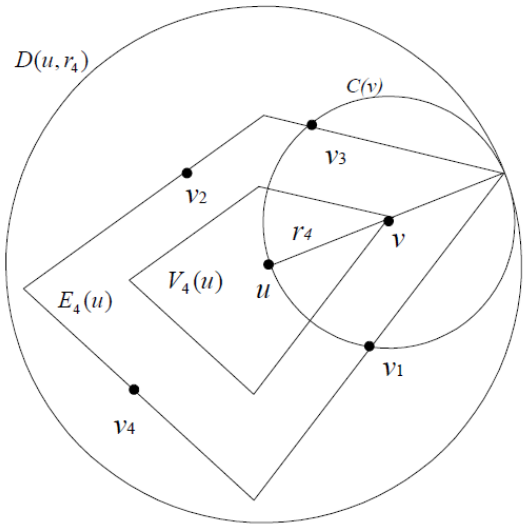

Figura 5. Criterio para encontrar a los vecinos

En este proyecto se ha empleado la función Voronoi que ya viene por defecto en Matlab [9].

2.1.2 Partición de la superficie

En el subapartado anterior se ha propuesto realizar el despliegue de los robots, moviéndolos a los centroides de las divisiones de Voronoi de una superficie circular. Sin embargo, no se ha mencionado con que criterio generar dichas divisiones.

Para ello, se propone situar a una distancia de valor la mitad del radio del área a las semillas que se usan para generar las divisiones de Voronoi. El ángulo que las separa es 360 grados dividido entre el número de robots.

A continuación, en las figuras 6, 7 y 8 se muestra con un punto rojo el centroide de cada división y con uno verde la semilla utilizada para generar cada partición de Voronoi. Se puede observar, como según aumenta el número de robots, más se separan los centroides de las semillas.

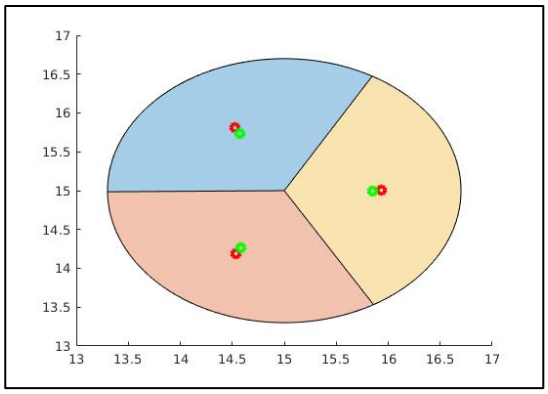

Figura 6. Semillas 3 divisiones

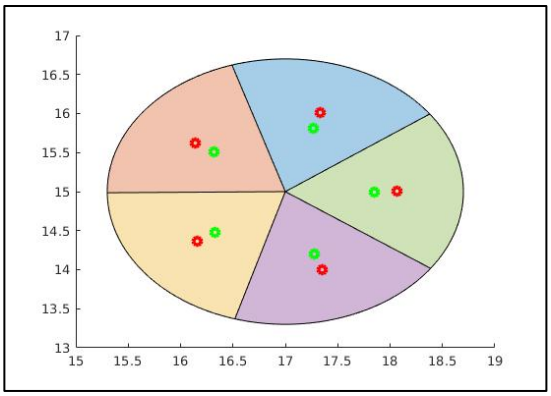

Figura 7. Semillas 5 divisiones

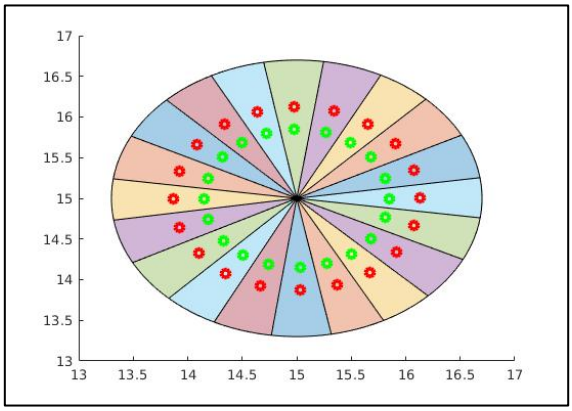

Figura 8. Semillas 20 Divisiones

2.1.3 Posicionamiento de los robots en las particiones

Una vez ya se han generado las divisiones, hay que buscar una manera de mover los robots de una manera ordenada a sus centroides. Este es uno de los momentos más delicados, ya que es muy fácil que dos o más robots se interfieran entre ellos.

Si esto sucede, ROS detecta a los demás robots como un obstáculo, entonces, debido a la proximidad de los centroides/puntos objetivo, no es capaz de encontrar ninguna trayectoria, por lo que los robots se atascan.

Para solucionar esto, se propone un método en tres fases. En la primera, todos los agentes rotan sobre si mismos hasta apuntar al objetivo que tengan asignado. En la segunda fase, los robots se desplazan por turnos hasta que todos alcancen sus centroides. En la última fase, todos los agentes se orientarán de la misma manera.

Para añadir mayor robustez al método, se lleva la cuenta del tiempo de ejecución. Sobrepasado cierto umbral, se reinicia la función. En ciertas situaciones enrevesadas, es posible que los robots se atasquen y sea necesario reiniciar. En este caso se volvería a comenzar con la fase uno desde las posiciones en las que se quedaron parados, donde se reajustaría nuevamente la orientación para buscar nuevas trayectorias.

A continuación, en la Figura 9 se muestra el proceso. Estas capturas han sido tomadas con la simulación con Stage y ROS ya en marcha. Los puntos rojos representan

los centroides de las divisiones. Los puntos azules y las puntas de flecha son las posiciones de los robots y su orientación.

Los robots no alcanzan exactamente las posiciones deseadas ya que ROS trabaja con ciertas tolerancias. Para agilizar el proceso, el centro sobre el cual se genera la superficie de la formación es la posición de uno de los robots.

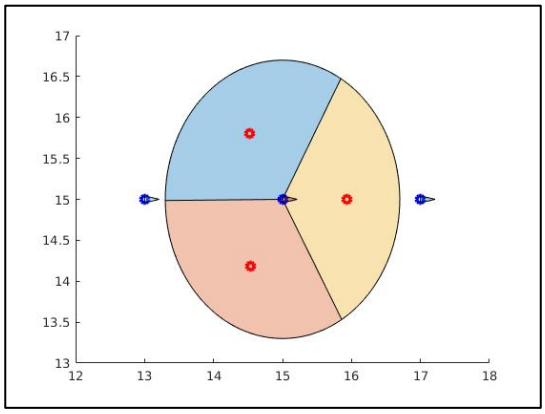

Figura 9.a. Apuntar

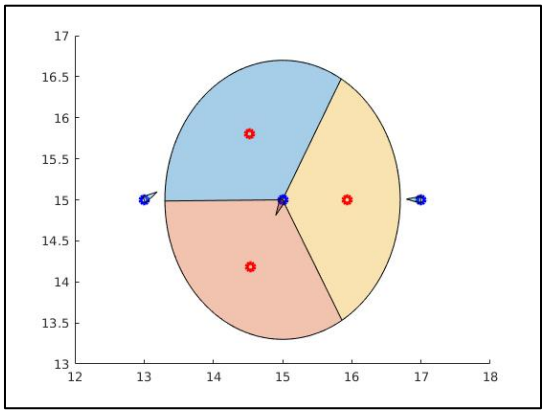

Figura 9.b. Apuntar

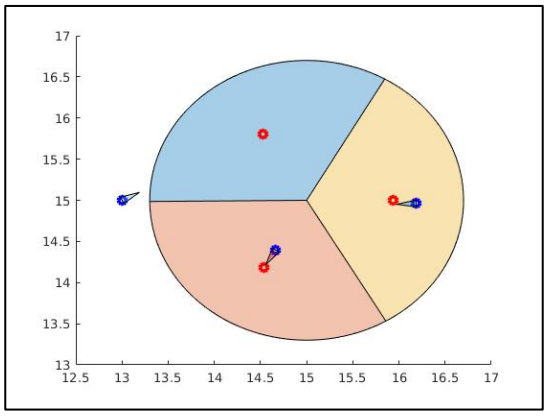

Figura 9.c. Movimiento

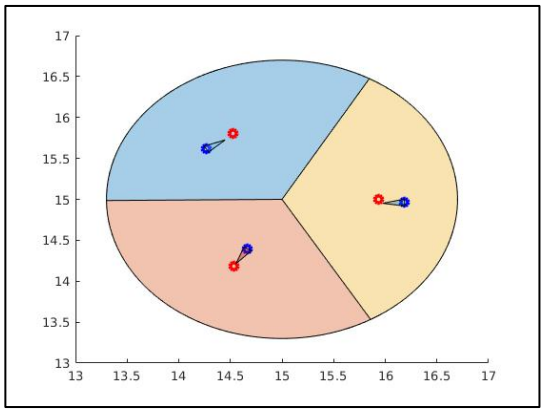

Figura 9.d. Movimiento

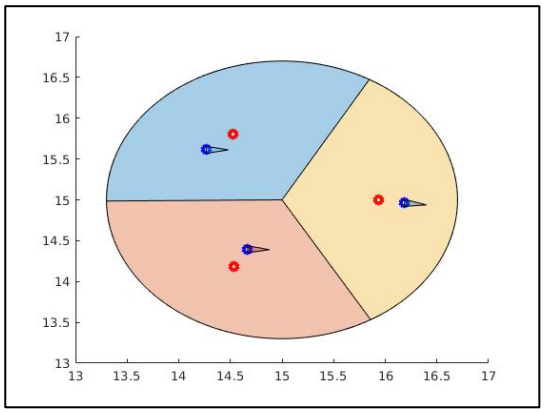

Figura 9.e. Orientar

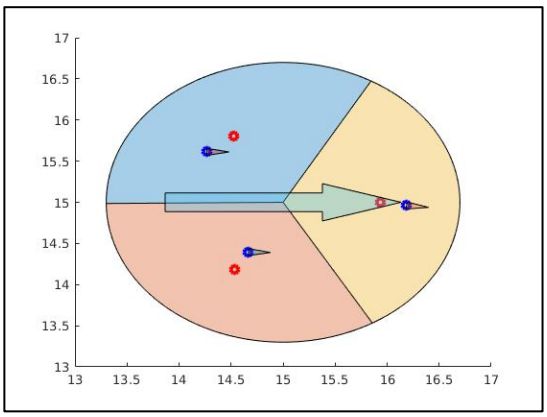

Figura 9.f. Orientar

A continuación, se propone una implementación en pseudocódigo similar a la que finalmente se utilizó.

[1] Orientar cada robot hacia su objetivo asignado.
[2] Desplazar los robots de uno en uno, comenzando por los más cercanos a sus objetivos.
[3] Orientar de la misma manera, al alcanzar el objetivo.

```
begin
switch Fase
    case 'Apuntar' [1]
        for i=1:1:Nrobots  //Número total de robots.
        diferencia[i].x=objetivos[i].x-posicionRobot[i].x;
        diferencia[i].y=objetivos[i].y-posicionRobot[i].y;
        AngulosApuntar=atan2(diferencia.y, diferencia.x); //Ángulo de robot a objetivo.
        sendGoals(posiciones actuales, AngulosApuntar);
        waitForRobot(); //Esperar a que el robot termine de moverse.
        Fase='Movimiento';
    case 'Movimiento' [2]
        indice=1;
        EnProgreso='true';
        while EnProgreso
        Robot=orden(indice); //Los robots con objetivo más cercano a ellos salen
                             //primero, orden calculado en el apartado 2.1.4.
        sendGoal(Robot,Goals(Robot),AngulosApuntar);
        waitForRobot(); //Esperar a que el robot termine de moverse.
        indice=indice+1;
        if(indice>Nrobots){EnProgreso=false; Fase='Orientar';}
        end
    case 'Orientar' [3]
        sendGoals(posiciones actuales, AnguloFormacion);
        waitForRobot(); //Esperar a que el robot termine de moverse.
end
end
```

Figura 10. Pseudocódigo propuesto para situar a los robots en sus posiciones

2.1.4 Asignación de objetivos

Para que el método explicado en el apartado anterior funcione mejor, hay que buscar que centroide se le asigna a cada robot. En este proyecto se propone la siguiente estrategia:

[1] A cada robot se le asigna su centroide más cercano.
[2] En el caso de que dos o más robots compartan un mismo centroide más cercano, se asignan según su orden en el "array" que almacena los datos de los robots y centroides. Por ejemplo, al robot número uno se le asigna el centroide número uno, al dos el dos, etc.
[3] Finalmente, se empareja al último robot con el centroide que queda.

A continuación, se propone una implementación en pseudocódigo similar a la que finalmente se realizó en Matlab, siguiendo los tres pasos anteriores.

```
begin
[1]
for i=1:1:Nrobots  //Calcular la distancia entra cada robot con cada posible objetivo
    for j=1:1:Nsemillas
        distancias [i,j] = distancia(robot[i],semilla[j]);
    end
end

asignación[1]=1:1:Nrobots;
asignación[2]=find(min(distancias)); //Asignar a cada robot su objetivo más cercano
[2]
for i=1:1:length(asignación)
     //Empezando por el principio de la lista, asignar los objetivos libres a los robots con
     //objetivos que se hayan asignado varias veces
    if (repetida(asignación[2])){ asignación[2]=SiguienteGoalNoAsignado();}
end
[3]
//Asignar el último objetivo
i=find(Objetivo no asignado);
j=find(Robot no asignado);
asignación[j,2]=i;
[extra]
//Generar una lista que ordene los robots según tengan su objetivo asignado más cerca, usado
//en el apartado 2.1.3 Posicionamiento de los robots en las particiones
for i=1:1:Nrobots
distancias=distancia(asignacion[1,i],asignacion[2,i]);
end
orden=sort(distancias);
end
```

Figura 12. Pseudocódigo propuesto para la asignación de objetivos

2.1.5 Implementación del despliegue de la formación

En este subapartado se propone una implementación en pseudocódigo para realizar el despliegue completo de la formación. De esta manera, se consigue situar a los robots en formación a partir de unas posiciones iniciales arbitrarias.

[1] Generar la superficie circular alrededor de uno de los robots.
[2] Esparcir las semillas.
[3] Calcular las divisiones de Voronoi y sus centroides.
[4] Asignar los objetivos de cada robot según las distancias.
[5] Desplazar los robots de manera ordenada partiendo de sus posiciones iniciales.
[6] Orientar a todos de la misma manera.

```
begin
[1]
centro = coordenadas del robot número round(Nrobots/2);
radio = el deseado para construir la superficie;
superficie.x = centro.x+radio*cos(0:1:360);
superficie.y = centro.y+radio*sin(0:1:360);
superficieCircular = figure(superficie); //función de Matlab que genera una figura a partir de esos puntos.
[2]
for i=1:1:Nrobots
    semillas[i].x=centro.x+(radio/2)*cos(i·separación);
    semillas[i].y=centro.y+(radio/2)*sin(i·separación);
end
[3]
vertices = voronoi(semillas);
particiones de la superficie = intersección(vertices, superficie);
objetivos = centroides(particiones de la superficie);
[4]
asignarObjetivos();   // Apartado 2.1.4 Asignación de objetivos.
[5]
situar a los robots en formación(); // Apartado 2.1.3 Posicionamiento de los robots en las particiones.
[6]
orientarRobots(mismaOrientacion); // Apartado 2.3.2 Rotar a la formación.

end
```

Figura 13. Pseudocódigo propuesto para el despliegue de la formación

2.2 Transición entre formaciones

Otro aspecto muy delicado del proceso es cuando hay que cambiar la formación para colocar a los robots en fila. Para abordar esta cuestión, se procede de manera similar que en el caso de la formación circular. Entonces, como se muestra en la Figura 14, se genera una nueva estructura para controlar la fila y que ningún robot se separe del resto.

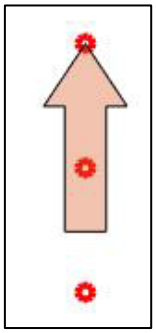

Figura 14. Estructura para gestionar la formación en fila

La nueva estructura consta únicamente de la flecha que indica la orientación general de la formación y las posiciones objetivo. Estas posiciones se sitúan una detrás de otra siguiendo la orientación de la formación, sin realizar ningún tipo de cálculo adicional. La distancia de separación entre los objetivos se puede variar, en el grueso de simulaciones realizadas en este proyecto se colocaron a 1,5 metros.

A partir de aquí, hay que resolver dos problemas. El primero es, como en el apartado anterior, evitar que los robots se choquen y bloqueen entre ellos. Esta cuestión se resuelve con el mismo método explicado en el subapartado “2.1.3 Posicionamiento de los robots en las particiones”. Los robots se sitúan en fila siguiendo tres pasos, primero apuntan a sus objetivos, después avanzan por turnos y finalmente se orientan todos de la misma manera.

En la Figura 15 se puede observar el proceso completo. Siendo los puntos rojos los centroides/objetivos, los azules, los robots y las puntas de flecha las orientaciones de cada uno.

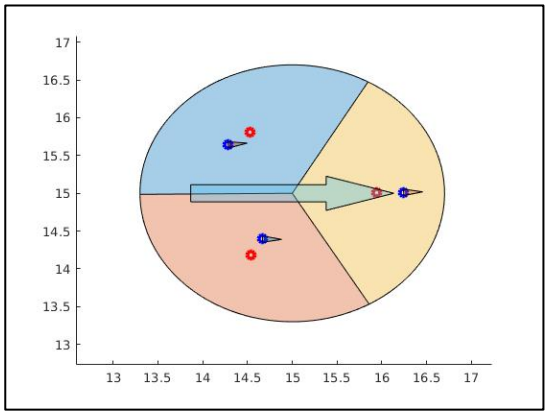

Figura 15.a. Formación principal

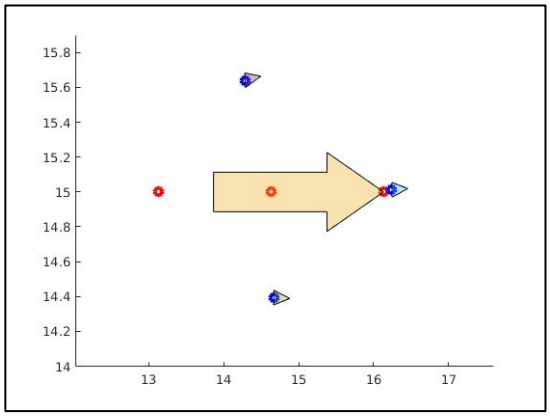

Figura 15.b Generación de la estructura "fila"

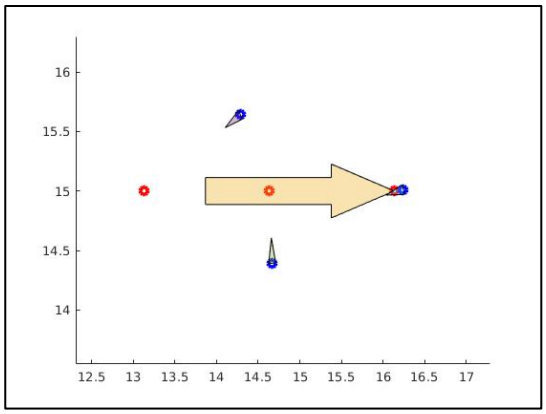

Figura 15.c. Apuntar (fila)

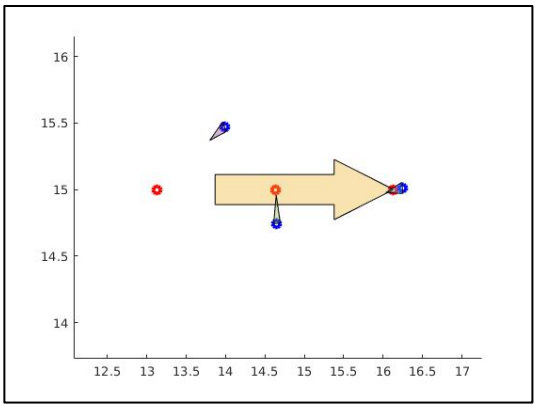

Figura 15.d. Movimiento (fila)

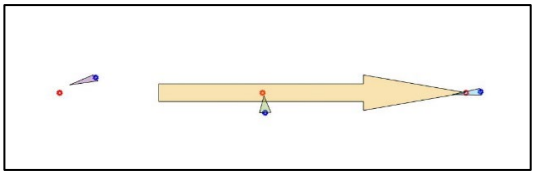

Figura 15.e. Movimiento (fila)

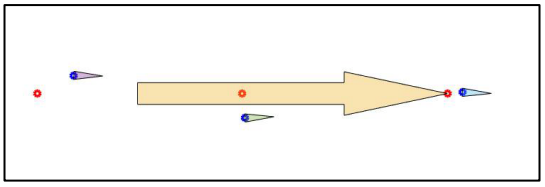

Figura 15.f. Orientación (fila)

El segundo problema consiste en que hay que tener cuidado acerca de dónde se generan las superficies que contienen a la formación. Teniendo en cuenta que, el obstáculo que fuerza la puesta de los robots en fila puede estar en cualquier sitio es preciso que la transición se realice lo más cerca posible de las posiciones reales de los robots.

Si, por ejemplo, la creación de la estructura que controla a la nueva formación en fila se implementa de tal manera que esta se genera en un punto fijo, los robots se desplazarían hacia sus nuevas posiciones generadas a partir de dicho punto fijo, independientemente de la posición en la que estuviesen los robots.

Esa situación puede resultar poco efectiva, por lo que en este proyecto se propone construir la fila empezando por un punto que se encuentre dentro del área circular antes de romperla. En las simulaciones se ha aprovechado la punta de la flecha que indica la dirección de la formación.

Para más detalle, la punta de la flecha de la Figura 15.a es la referencia que se ha utilizado para poner a los robots en fila. En la Figura 15.b, se puede observar como justamente coincide la punta de la flecha con el primero de los puntos objetivo.

Una vez ya no es necesario continuar en fila, se restaura la estructura para la formación circular. Para evitar el problema comentado anteriormente, dicha estructura se genera entorno al robot cuyo número en la lista sea el intermedio. Por ejemplo, si hay tres robots, la nueva formación se genera en torno al segundo. En la Figura 16 se puede ver como el centro de la nueva estructura es el segundo de los robots.

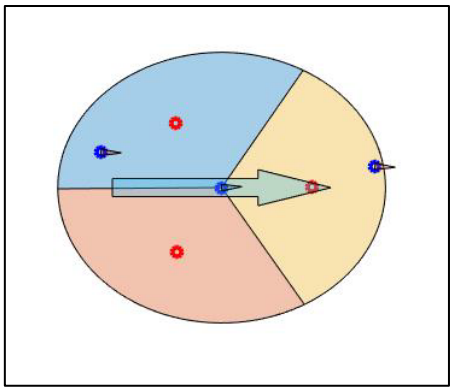

Figura 16. Restaurar formación circular

A partir de aquí hay que repetir el método para llevar a los robots a sus posiciones tal y como se ha explicado en el apartado "2.1.3 Posicionamiento de los robots en las particiones".

A continuación, se propone una implementación en pseudocódigo para realizar la transición entre formaciones.

Partiendo de la formación principal:

[1] Generar la nueva estructura para gestionar la fila tomando como referencia un punto cercano a los robots.
[2] Colocar los objetivos uno de tras de otro según la orientación de deseada.
[3] Asignar los objetivos de cada robot según las distancias.
[4] Desplazar los robots de manera ordenada hacia sus objetivos.
[5] Orientar a todos según la orientación deseada.

```
begin
[1]
puntoInicial.x=centroFormacion.x+(radio/2)·cos(orientacionFormacion);
puntoInicial.y=centroFormacion.y+(radio/2)·sin(orientacionFormacion);
objetivos[1]=puntoInicial;
[2]
for i=1:1:(Nrobots-1)
siguiente.x = puntoInicial.x+(i·distancia·cos(orientacionFormacion));
siguiente.y = puntoInicial.y+(i·distancia·sin(orientacionFormacion));
objetivos[i+1]=siguiente;
end
[3]
asignarObjetivos();  // Apartado 2.1.4 Asignación de objetivos.
[4]
situar a los robots en formación(); // Apartado 2.1.3 Posicionamiento de los robots en las particiones.
[5]
orientarRobots(mismaOrientacion); // Apartado 2.3.2 Rotar a la formación.
end
```

Figura 17. Pseudocódigo propuesto para alternar entre la formación principal y fila

Si se parte de la formación en fila, se da el mismo caso que en el apartado “2.1.5 Implementación del despliegue de la formación”, ya que se trata de la misma situación en la que se pretende situar a los robots en formación partiendo de unas posiciones cualquiera, solo que en este caso, son sus posiciones en la fila.

2.3 Navegación en formación

Esta parte es de gran trascendencia dentro del objetivo final planteado. El grueso del tiempo los robots van a navegar manteniendo la formación principal. Una vez ya está generada la estructura para la formación circular y los robots ya estén situados en sus posiciones, solo falta buscar una estrategia para que los robots no rompan la formación mientras se mueven hacia sus objetivos.

Para ello, en este proyecto se proponen dos métodos, uno para desplazar la formación en línea recta y otro para modificar su orientación. Se trata de un planteamiento sencillo, pero con mucho margen de mejora, lo cual no quita para que sea funcional y pueda superar algún que otro problema derivado de ROS.

2.3.1 Desplazar a la formación en línea recta

Se pretende crear un método que, dada una determinada distancia, desplace a la formación en línea recta dicha longitud. El método será válido para las dos formaciones con sus correspondientes ajustes.

A la hora de solucionar esta cuestión, aparece un problema de cierta magnitud. Dicho problema radica en cómo el propio ROS funciona. El cálculo de trayectorias con ROS se explica con mayor profundidad en el apartado “3.3.5 Cálculo de trayectorias”.

La cuestión es que siempre va a haber al menos un robot que tenga a otro más o menos delante. Teniendo en cuenta que los agentes van a estar muy juntos, el robot que va detrás detectará a los demás como un obstáculo. En ese caso, ROS tiene problemas para calcularle una trayectoria, lo que desemboca en el bloqueo de ese robot.

En la Figura 18 se muestra esta situación en el entorno de simulación Stage. En la parte de la derecha se representa el mapa de costes que ha generado el robot rojo. Se ve claramente como el robot que está detrás se piensa que tiene una pared delante.

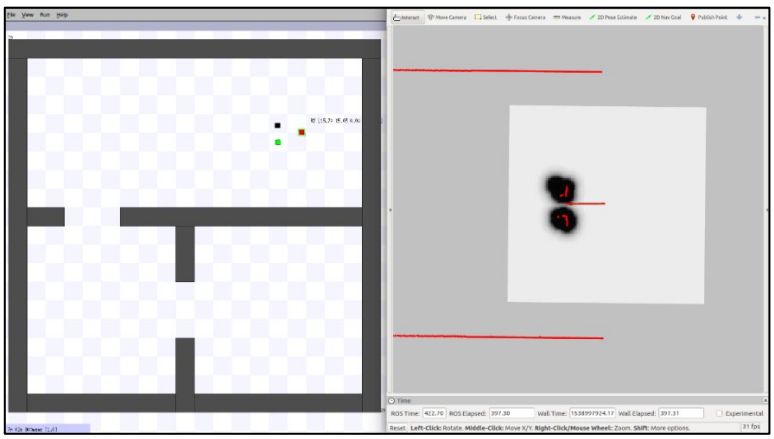

Figura 18. ¿Qué ve el robot de atrás?

Además, hay que recordar que uno de los requisitos para la navegación es que los robots no deben romper la formación por si solos. Es decir, no basta con calcular una nueva estructura y desplazarla hasta el punto deseado. En ese caso, cada robot intenta alcanzar su centroide correspondiente, pero cada uno va por su lado, volviéndose a juntar cuando todos alcancen sus objetivos.

Para solucionar estos dos problemas se propone, en lugar de trasladar a la formación de golpe, ir poco a poco. Se va desplazando la estructura que controla la formación una distancia reducida. Se propone que esa distancia sea una fracción del radio de la superficie de control.

De esta manera, se consigue que los robots alcancen los centroides desplazados sin interferirse unos con otros. Además, también se consigue que los robots no rompan la formación en ningún momento, ya que, al enviarles objetivos a tan poca distancia, no se les da margen para que marchen cada uno por su cuenta.

Finalmente, cuando al menos uno de los robots alcance su objetivo final, dejan de generarse objetivos auxiliares y se espera a que todos los demás lleguen. En la Figura 19 se puede observar a los robots, los objetivos finales, y la primera superficie auxiliar.

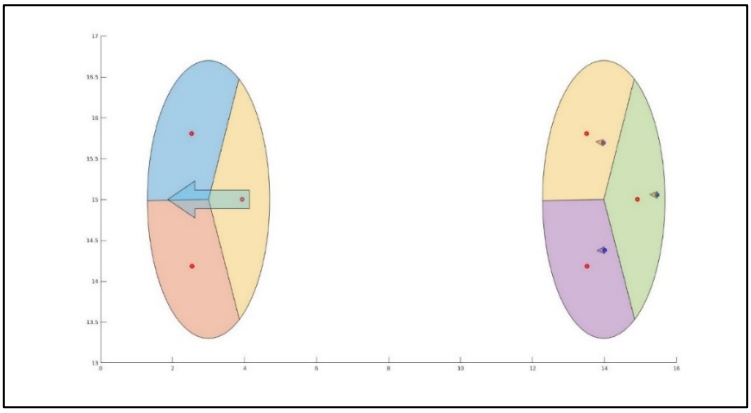

Figura 19. Objetivos auxiliares formación principal

Para desplazar la formación en fila se emplea la misma estrategia, se muestra en la Figura 20.

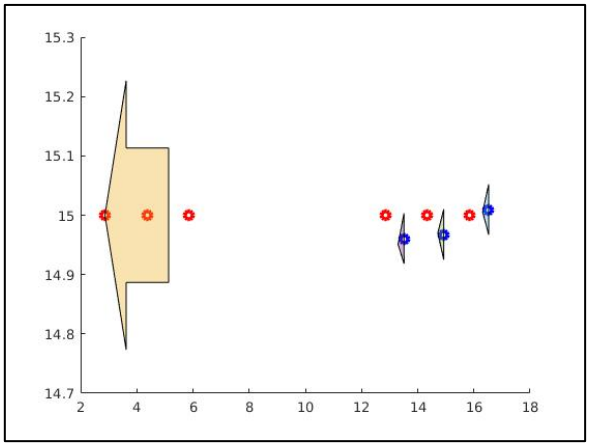

Figura 20. Objetivos auxiliares formación en fila

Resumiendo, para desplazar a la formación en línea recta se procede de la siguiente manera:

[1] Desplazar la estructura de la formación una distancia de valor una fracción del radio de la superficie de control.
[2] Esperar a que los robots alcancen los objetivos auxiliares.
[3] Repetir los pasos 1 y 2 hasta recorrer la distancia deseada.

```
begin
paso=radio/5 //Se propone una fracción del radio de la superficie como paso
while distancia deseada no recorrida

trasladar(particiones,paso);
recalcularObjetivos();
enviarObjetivos();
esperarHastaAlcanzarObjetivos();

end
```

Figura 21. Pseudocódigo propuesto para desplazar a la formación en línea recta

2.3.2 Rotar a la formación

Ahora, el objetivo consiste en hacer rotar a todos los robots sobre si mismos hasta alcanzar la misma orientación. Aquí no surge ningún problema digno de mención, ya que los robots no interfieren unos con otros y ni siquiera hace falta calcular trayectorias.

Lo único que hay que hacer es enviar a los robots como posiciones objetivo, la misma en la que están ya y como orientación, la misma para todos. En las figuras 22 y 23 se muestra el proceso.

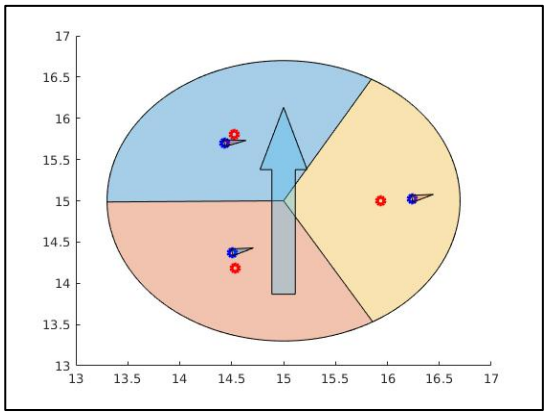

Figura 22. Rotar la formación 1

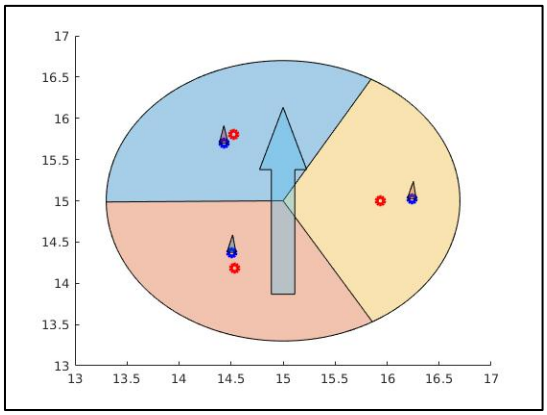

Figura 23. Rotar la formación 2

2.4 Realización de un recorrido completo

Una vez montada toda la infraestructura para controlar la formación de los robots y desplazarla, surge el interrogante de cómo hacer que naveguen por determinada zona. Para resolver esta cuestión, resulta ineludible el poder acceder a un mapa de la zona donde se ejecuta el algoritmo, en este proyecto, Matlab. Se proponen dos formas, de las cuales la primera tuvo que descartarse por los motivos que se exponen en el siguiente subapartado.

2.4.1 Utilizar el mapa de coste que van generando los robots

Como se menciona en el apartado "2.3.1 Desplazar a la formación en línea recta" y con mayor profundidad en "3.3.4 Navegación", los robots van construyendo un mapa de la zona con lo que van observando a través de sus cámaras durante su recorrido.

En el caso de poder disponer de dicho mapa de costes, la idea sería aplicar algunos de los algoritmos de navegación más conocidos en el estado del arte como A* o Dijkstra para calcular una trayectoria para toda la formación.

Esta solución presenta los siguientes inconvenientes que quedaron sin resolver:

[1] ¿Con qué criterio se establecen los puntos objetivos para A* y Dijkstra teniendo en cuenta que no se conoce el mapa?
[2] Habría que "masajear" la trayectoria calculada, ya que solo se han implementado funciones para que la formación avance en línea recta y rote sobre sí misma.
[3] ROS no permite acceder a los mapas de coste de manear habitual. Es posible obtenerlos, pero requiere demasiado tiempo de cómputo.

Es por estas tres razones, sobre todo la última que se descartó esta solución y se optó por la siguiente.

2.4.2 Navegación con mapa previo

La alternativa consiste en considerar que ya se conoce el mapa antes de empezar. A partir de aquí, se propone elaborar una "receta" siendo los "ingredientes" las funciones para rotar, avanzar en línea recta y alternar entre formaciones. Se elabora una lista con estas instrucciones que van guiando a la formación para completar el laberinto.

El llevar a los robots "sobre raíles" no implica que estén ciegos. ROS continúa generando sus mapas de coste y, en caso de que aparezca un obstáculo inesperado, los robots pueden reaccionar de dos maneras. En el caso de que su objetivo actual resulte coincidir con el obstáculo, el robot se para. Si el obstáculo aparece entre el robot y su objetivo actual, ROS calculará una nueva trayectoria de tal manera que circunvale al obstáculo.

Teniendo en cuenta como se ha implementado el control de la navegación en el apartado "2.3.1 Desplazar a la formación en línea recta", las trayectorias que se van generando son muy cortas y sería muy raro que se diera el segundo caso. En la Figura 24 se muestra el laberinto propuesto y el recorrido que se pretende seguir. Los robots se deberán poner en fila en las dos zonas que el mapa se estrecha.

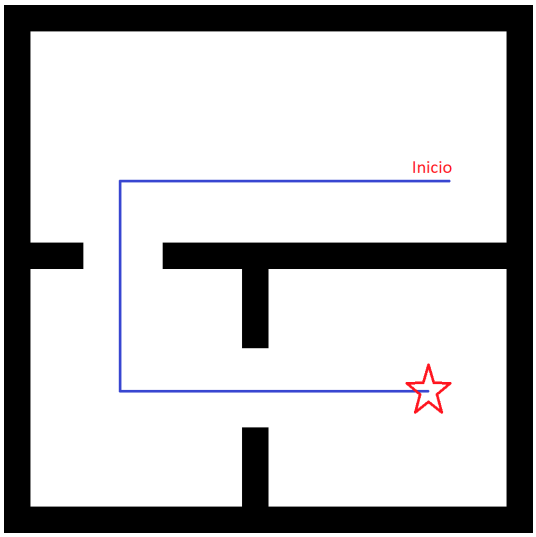

Figura 24. Mapa propuesto y recorrido

2.5 Discusión sobre el número de robots

Todo lo que se ha propuesto hasta ahora se ha hecho pensando en que sea aplicable a cualquier número de robots. Lo único que hace falta para variar el número de robots es modificar la cantidad de divisiones de la estructura.

Sin embargo, a medida que aumenta el número de robots, más difícil es colocarlos en formación, ya que es más probable que se molesten entre ellos. Para evitarlo hay que adaptar el radio de la formación para que sea más fácil situarlos.

3 Implementación de la simulación

Cada vez resulta más interesante comprobar el funcionamiento de los algoritmos en simulación antes de su implementación real. Entonces, para poder ver todas estas ideas en acción, se propone realizar una simulación sobre una plataforma realista, en este caso, ROS.

Se trata de un sistema operativo con un conjunto de herramientas bastante complejo. En este capítulo se realiza una presentación del software empleado y se indica qué hace falta para hacerlo funcionar.

3.1 Matlab

Se trata de un programa matemático muy conocido donde se implementan los algoritmos. Cuenta con un lenguaje de programación propio muy potente a la hora de representar y operar con matrices y vectores. Existen muchos complementos que le pueden añadir aún más funcionalidades, como por ejemplo herramientas para el análisis estadístico, cálculo de controladores y redes neuronales entre otros.

En principio, MathWorks, la empresa que lo mantiene no tiene que ver con ROS: Sin embargo, Matlab trae por defecto funciones que facilitan mucho la comunicación con ROS:

3.2 Plataforma de simulación

La plataforma de simulación soportará los algoritmos ejecutados en Matlab y es la encargada de representar visualmente el comportamiento de los robots. Se han utilizado dos programas distintos que se presentan a continuación.

3.2.1 Gazebo

Gazebo [10] es un simulador de entornos 3D especializado en robótica. Permite crear entornos complejos e importar modelos de otros programas CAD. Es muy agradable visualmente, pero por contra exige muchos recursos al ordenador que lo utilice. Está pensado para probar algoritmos y diseñar robots.

Incluye un motor de físicas e interfaz de programación. Además, es un programa gratuito mantenido por la comunidad. Existe soporte de ROS que posibilita la comunicación con Gazebo.

Incluye funcionalidades adicionales como la posibilidad de realizar las simulaciones en la nube con Amazon AWS, soporte para varios tipos de sensores y la posibilidad de desarrollar plugins personalizados.

En principio para este proyecto se iba a utilizar Gazebo, pero el ordenador disponible carecía de la potencia adecuada. Aproximadamente, un segundo de simulación equivalía a 50 en la realidad.

3.2.2 Stage

Stage [11] forma parte del mismo proyecto que desarrolla Gazebo. Es un programa gratuito el cual se puede redistribuir o modificar, como es el caso de la versión que viene junto con la instalación de ROS. En su descripción oficial, lo definen cono un simulador 2.5D especializado en robótica, pensado para simular sistemas multi-robot.

En esencia es lo mismo que Gazebo, pero mucho más simple visualmente. Puede utilizarse Stage como programa único o en forma de plugin para otros programas. Existen modelos de robots y documentación creada por la comunidad disponibles pasa usar y consultar libremente.

Entre los sensores que se pueden simular, destacan los sonar, sensores láser, cámaras y odometría. El entorno de simulación está basado en mapas de bits, siendo posible para el usuario dibujar su propio mapa en un programa externo.

3.3 ROS

La implementación de este sistema se hace de manera similar a [12]. ROS es un sistema operativo para el desarrollo de software para robots. Se trata de una colección de herramientas y librerías cuyo propósito es ayudar a crear aplicaciones complejas y robustas para una gran variedad de tipos de robots. Es completamente libre y abierto.

Su razón de ser consiste en que crear software de calidad para robots desde cero es muy complicado. Ningún laboratorio, institución o individuo que quiera ver resultados en un plazo razonable puede permitirse lidiar con todas esas complicaciones.

A raíz de esta problemática nació ROS como un entorno de desarrollo colaborativo. Por ejemplo, un grupo puede ser muy bueno mapeando entornos y puede colaborar incorporando sus métodos y librerías a ROS. Por esta razón, es cada vez más común verlo en entornos industriales además de los académicos. Para buscar o aportar información sobre ROS lo mejor es acudir a su wiki [2].

3.3.1 Estructura del funcionamiento de ROS

Debido a la gran complejidad que supone gestionar todos los aspectos de un robot, se genera la necesidad de que la estructura que gobierna el funcionamiento de ROS sea lo más robusta y sencilla posible.

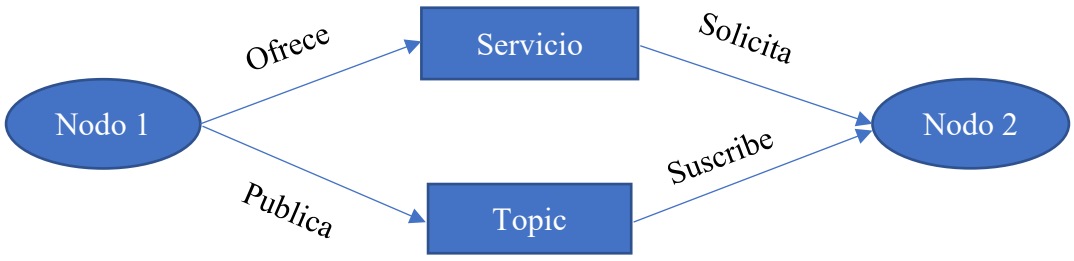

Figura 25. Esquema del funcionamiento de ROS

La estructura está basada en nodos, cada uno se dedica a realizar una tarea específica. Se comunican entre sí de dos maneras posibles, la primera de ellas y las más habitual consiste en que el nodo publica mensajes continuamente en un "topic" o asunto, se podría asemejar a un foro común de internet. La segunda manera consiste en que un nodo puede, en un momento puntual, solicitar un servicio a otro.

A modo de ejemplo, en este proyecto se ha usado un nodo que publica constantemente información sobre la odometría del robot y a otro que se le solicita de vez en cuando que limpie el mapa de costes de los robots para que no se hagan lio entre ellos.

De entre todos los paquetes de nodos que se van a usar en este trabajo, destacan los que se encargan de controlar y describir a los robots, los necesarios para la navegación y localización y los que gestionan comunicación entre Matlab, ROS y Stage.

3.3.2 Implementación de un robot

En principio, ni ROS, ni Matlab ni la plataforma de simulación tienen algún tipo de información acerca del robot con el que se quiere trabajar, lo que implica que el usuario es quien debe proporcionar dicha información. Esta parte también tiene cierta complejidad, pero, por otro lado, hay mucha libertad respecto a los tipos de robots que se pueden implementar. A continuación, se describen los pasos que hay que seguir para que ROS disponga de la información necesaria para gestionar un robot.

Lo primero que hay que hacer es escribir el fichero tipo URDF que describe a los robots. Entre otras cosas, hay que especificar las uniones del robot, la carcasa y los sensores. De esta manera se posibilita el representar al robot en simuladores. Si se utiliza un simulador 3D como Gazebo, también hace falta, con ayuda de algún programa CAD, crear el modelo 3D de las distintas partes del robot y los sensores.

El siguiente paso es escribir los nodos que sean capaces de leer los sensores y publicar su información. Estos nodos trabajarán directamente con el hardware que sea y transforman su información en mensajes reconocibles en un "topic".

Finalmente, hay que escribir los controladores para todos los actuadores que tenga el robot. Estos nodos proporcionan las consignas al hardware adecuadas para que el robot se mueva.

Afortunadamente, en este proyecto se trabaja con turtlebots, un tipo de robot ampliamente extendido del cual ya existen modelos y controladores en internet. En este proyecto se utilizó un modelo de turtlebot para Stage de 25 cm de radio desarrollado por el departamento de robótica de la universidad de Zaragoza. Como controlador se utilizó el "dwa_local_planner" disponible en la wiki de ROS. Se trata de un paquete que, dada una trayectoria a seguir y un mapa de costes, genera los comandos de velocidad adecuados.

3.3.3 Localización

El sistema de localización está basado en un nodo que ejecuta un algoritmo AMCL (Adaptative Monte Carlo Localization). El utilizado para este proyecto se puede encontrar en la wiki de ROS con más información. Este nodo compara las medidas tomadas por el láser de los robots con un mapa conocido de la zona mediante un filtro de partículas para ofrecer una estimación probabilística de la localización del robot. Es posible modificar este nodo para trabajar con otro tipo de sensores.

Esta estimación se compara con la posición calculada por la odometría para calcular una corrección que se usa para obtener una aproximación más exacta de la posición del robot. Este algoritmo parte de la hipótesis de que el robot se puede encontrar en cualquier parte. Conforme se obtienen más datos con el movimiento del robot y la percepción de objetos del entorno, el filtro va mejorando su estimación.

3.3.4 Navegación

ROS ofrece un sistema de navegación en dos dimensiones. Es bastante simple conceptualmente, pero se complica bastante a la hora de su implementación. Se basa en un nodo principal que recibe información sobre la localización, estructura del robot, odometría, sensores y una posición objetivo. A partir de todo esto, genera una trayectoria y comandos de velocidad para el controlador de las ruedas. En la Figura 26 se muestra un esquema, más información en la wiki de ROS.

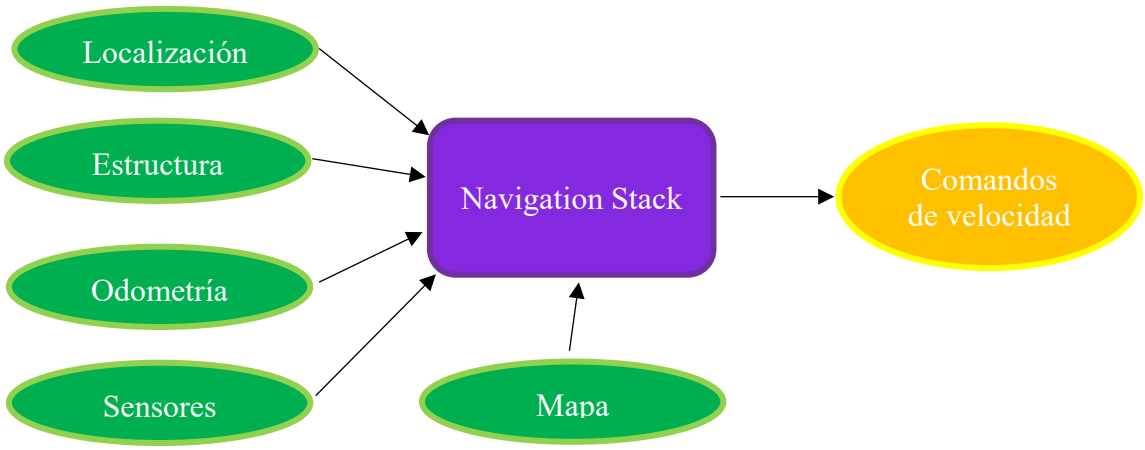

Figura 26. Esquema del sistema de navegación

Entrando un poco más en detalle, el sistema genera un mapa de costes global con la información del mapa (no es estrictamente necesaria) y los sensores del robot. Cada punto de dicho mapa indica la probabilidad de que haya algún obstáculo. Con la información que contiene se calcula una trayectoria hacia el punto objetivo.

En paralelo a todo esto, se genera un mapa de costes local que, junto con la trayectoria obtenida anteriormente, sirven para calcular las consignas de velocidad. Este segundo paso está pensado para la evitación de obstáculos, ya que en el caso de que algún objeto aparezca de repente sobre la trayectoria calculada, el mapa de costes local se actualiza por medio del sensor correspondiente y las consignas de velocidad generadas evitan el choque.

En caso de que se complique la circunvalación del obstáculo, existen una serie de protocolos para intentar desatascar el robot, como, por ejemplo, ponerse a dar vueltas sobre sí mismo para actualizar el mapa de costes más cercano al robot y calcular nuevas trayectorias.

3.3.5 Cálculo de trayectorias

Respecto al cálculo de la trayectoria, ROS ofrece dos métodos considerados estado del arte, que son el método Dijkstra y A*.

Método Dijkstra

El método Dijkstra fue desarrollado por Edsger Dijkstra, científico en computación de los Países Bajos en 1956 [13]. Este método busca el camino más corto entre todos los nodos de un grafo. Existen variaciones de este algoritmo como la que se usa aquí, donde se detiene la ejecución cuando ya se ha encontrado el camino más corto entre el nodo de origen y un destino dado.

Funciona de la siguiente manera, donde en un vector V se almacenan las distancias entre el nodo inicial y todos los demás y en X el camino. Dado un nodo inicial y un objetivo final:

[1] Establecer como hipótesis que todas las distancias de V son infinito y el primer nodo del camino X es el inicial.
[2] Calcular, para el último nodo almacenado en X, las distancias entre todos sus nodos adyacentes y el inicial, pasando por el camino calculado hasta ahora.
[3] Actualizar el vector V, almacenando en el las mínimas distancias entre cada nodo y el inicial. Es posible que una distancia mínima calculada en un paso intermedio no sea la mínima absoluta.
[4] Añadir a X el nodo cuya distancia calculada en [2] en esta iteración sea la mínima.
[5] Comprobar si se ha alcanzado el objetivo final. En caso afirmativo se para el algoritmo, en caso contrario, se vuelve al punto [2].

A*

Por otra parte, el algoritmo A* fue propuesto por Peter Hart, Nils Nilsson y Bertram Raphael en 1968 [14]. Se trata de una nueva versión del algoritmo de Dijkstra más rápida gracias al uso de técnicas heurísticas.

En este caso, la idea consiste en mantener dos listas, las conocidas como lista abierta y lista cerrada. La lista abierta comienza con el nodo inicial y almacena los nodos

potencialmente mejores aún no considerados. Por otro lado, en la lista cerrada se almacenan los nodos ya visitados.

En el bucle principal, se busca el nodo de la lista abierta cuyo coste sea el menor. Si dicho nodo no es el objetivo final, se añaden todos sus vecinos a la lista abierta. El nodo considerado en esta iteración se mueve a la lista cerrada. Una vez se alcanza el objetivo final, se construye la trayectoria, para ello es necesario que, a la hora de la implementación, se tenga constancia de que nodos son los padres de cada uno.

Cabe destacar que, si en algún momento no hay ningún elemento en la lista abierta y aún no se ha alcanzado el objetivo, quiere decir que no existe camino posible. Respecto al cálculo de los costes, este método utiliza métodos heurísticos, para más detalles consultar la referencia [14].

Tal como se han implementado estos métodos en ROS, se puede afirmar que A* es mucho más rápido que Dijkstra, pero en algunos casos, las trayectorias que calcula no son tan óptimas.

En la Figura 27 y Figura 28 se puede observar la diferencia en el comportamiento de los dos. Dijkstra explora mucha más superficie y encuentra un camino más directo, mientras que A*, es mucho más rápido pero su trayectoria no es tan buena.

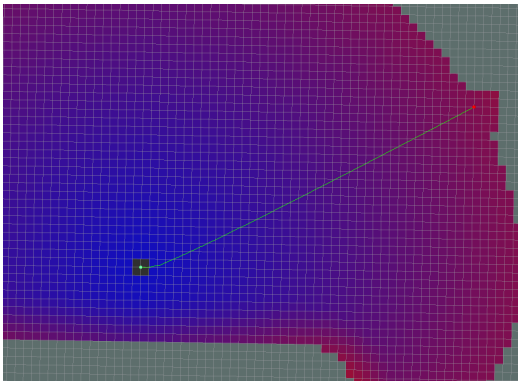

Figura 27. Dijkstra

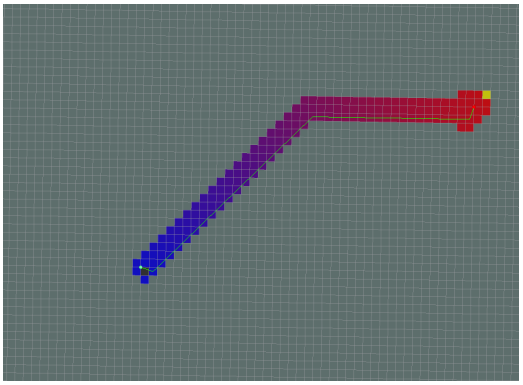

*Figura 28. A**

Para este proyecto se ha concluido que los dos métodos son igual de buenos, ya que las trayectorias que se calculan siempre son muy cortas teniendo en cuenta que los puntos de destino que se envían están muy cerca de los robots.

3.3.6 Uso de varios robots

Como se indica en [12], para poder usar más de un robot hay que recurrir a una utilidad que ofrece ROS conocida como “namespace” o espacio de nombres. De esta manera, todos los nodos, mensajes y servicios se pueden agrupar colocando delante de ellos un prefijo.

Gracias a esto, una vez se tiene montada las estructura para un robot, esta puede replicarse el número de veces que haga falta siempre y cuando el ordenador pueda con ello.

4 Simulación y pruebas

Este apartado tiene el objetivo de mostrar el funcionamiento del algoritmo en simulación. Primero se muestra uno de los casos más favorables, tras lo cual se realizarán pruebas modificando los parámetros más relevantes, que son el paso de avance de los robots, el radio de la formación y el número de robots. Todos los vídeos se muestran a x8 de velocidad. En la ventana de la izquierda se muestra la simulación en sí mientras que en la de la derecha lo que ve el robot verde.

Se considera como parámetro más crítico el radio de la formación, por lo que el paso de avance se referencia a fracciones de dicho radio. Esto es así por dos motivos, el primero consiste en que es indispensable acomodar a todos los robots dentro de la superficie de la formación. El segundo motivo es que, el paso de avance está limitado por el radio de la formación, habiendo concluido en las simulaciones que lo mejor es utilizar un paso de valor un quinto del radio de la formación.

Respecto al mapa, se ha dibujado un laberinto sobre un marco de 1000x1000 píxeles con una densidad de 50 píxeles por metro, resultando en una habitación de 20 metros de lado. Los pasillos por los que pasan los robots en fila son de 3 metros de ancho. Cada robot mide 25 centímetros de radio y el mínimo radio para la formación utilizado es de 1,7 metros.

4.1 Simulación básica

La configuración más usada para la verificación de los métodos durante su desarrollo consiste en tres robots con un radio de la formación de 1,7 metros y una quinta parte de dicho radio como paso de avance. A continuación, se muestra un vídeo para este caso, se puede observar como los robots llegan adecuadamente al final a pesar de que uno entra al final en lo que ROS denomina “rotate recovery behavior”. Esto es debido a que en algún momento uno de los robots interfiere con otro y no es capaz de calcular una trayectoria.

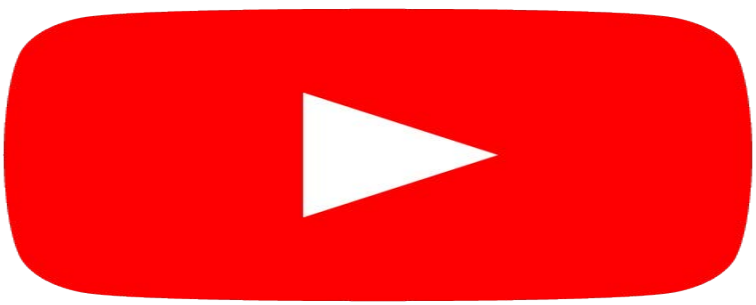

Figura 29. Simulación básica

4.2 Pruebas modificando el paso de avance

En este caso se modifica la distancia que avanzan los robots en cada paso del desplazamiento de la formación en línea recta. Se puede observar que, al hacer muy próximo dicho paso al radio de la formación, el sistema deja de funcionar. Esto es así porque de esta manera, los robots de atrás le envían al planificador de trayectorias un objetivo que coincide o está demasiado próximo a los robots que van delante, resultando en dificultades o directamente la imposibilidad de calcular trayectorias.

Al reducir el paso, se reduce notablemente la posibilidad de que los robots se molesten entre ellos a expensas de que tarden más en completar el recorrido. Por consiguiente, para encontrar un equilibrio entre velocidad y estabilidad, se propone un paso de avance de un quinto del radio de la formación. A continuación, se muestran tres simulaciones con distintos pasos de avance.

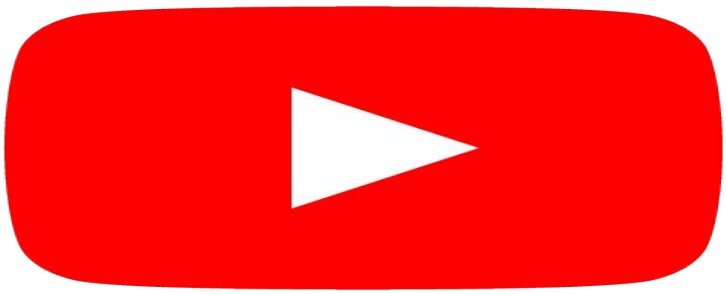

Figura 30. Paso igual al radio de la formación

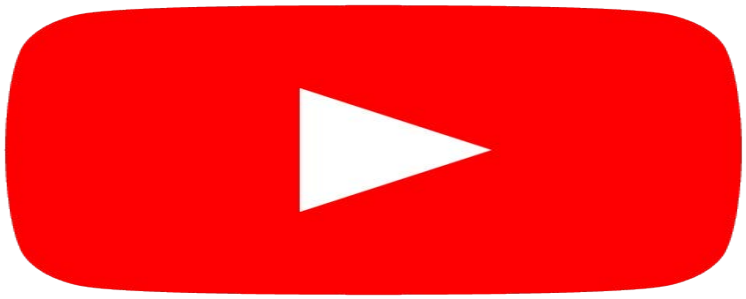

Figura 31. Paso igual a un décimo del radio de la formación.

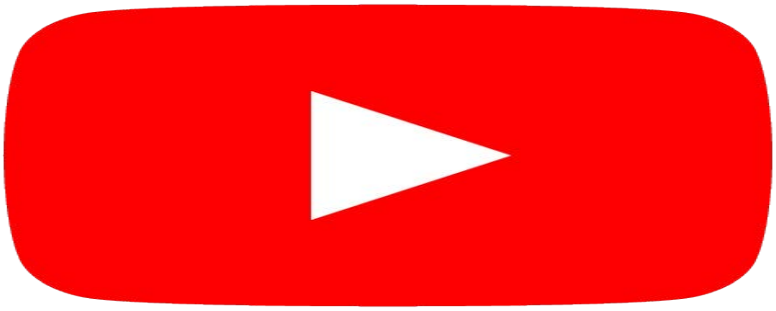

Figura 32. Paso igual a un cincuentavo del radio de la formación

4.3 Pruebas modificando el radio de la formación

El efecto que provoca modificar el radio de la formación principal repercute en lo mucho o poco que se interfieren los robots entre ellos. Al hacerlo muy pequeño, llega un momento en que los robots no son capaces de maniobrar al estar tan juntos. Por otro lado, es posible aumentar el tamaño del radio, ya que así será más fácil evitar que los robots se atasquen.

Hay que tener en cuenta que radios muy grandes pueden no ser prácticos para el proyecto en cuestión o directamente no disponer del espacio necesario, por lo que se concluye que lo mejor para este parámetro es dejarlo en un punto medio. Para encontrar este punto medio se propone buscar un radio que permitan a los robots tener el espacio suficiente para colocarse dentro de la formación y maniobrar con un paso de avance de un quinto de dicho radio. Sin olvidar que el radio elegido debe ser apto para el proyecto que se esté manejando.

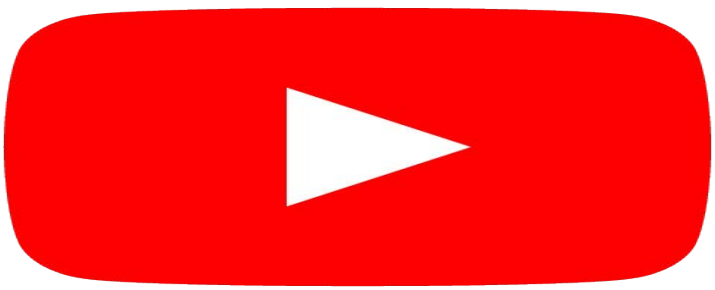

Figura 33. Radio igual a un metro

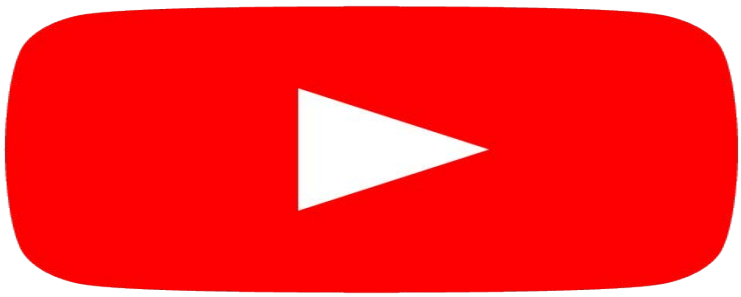

Figura 34. Radio igual a 3.5 metros

4.4 Pruebas modificando el número de robots

Modificar el número de robots también es un proceso delicado que puede requerir la acomodación de los dos parámetros anteriores. En este caso, las principales limitaciones son dos, por una parte, vuelve a aparecer el hecho de que los robots se bloqueen entre ellos. Cuantos más son, más difícil resulta situarlos en formación y colocarlos en fila, además de que se requiere mucho más espacio. Por otro lado, están los recursos del ordenador, como se puede apreciar en la simulación, con cinco robots las prestaciones del ordenador para la ejecución se empiezan a degradar.

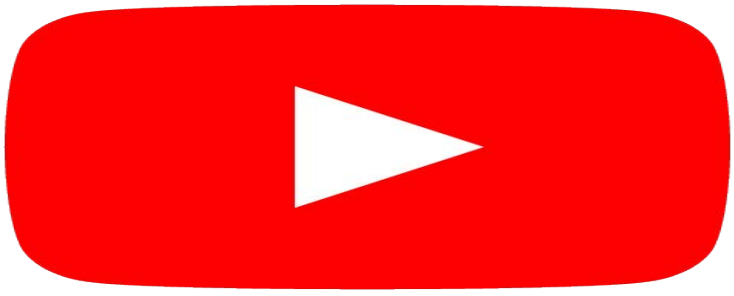

Figura 35. Con cuatro robots

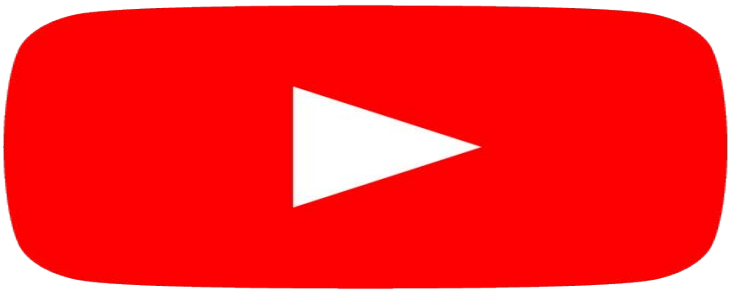

Figura 36. Con cinco robots

5 Conclusiones y posibles mejoras futuras

5.1 Conclusiones

En este proyecto se han desarrollado estrategias para que un conjunto de robots se sitúe y navegue en formación. Se han simulado con ayuda de ROS, Matlab y Stage. Durante la realización del proyecto, se han observado algunos detalles dignos de mención.

Lo primero que se observó fue que, por una parte, es sencillo crear estructuras para cualquier número de robots. Por otro lado, cuanto mayor es el número de robots a colocar en formación, más difícil resulta por la posibilidad de que aparezcan bloqueos.

Otro detalle que en principio no se le dio mucha importancia fue la asignación de objetivos a cada robot, pero al ir avanzando se convirtió en una cuestión clave. Asignar de manera inteligente los objetivos a cada robot contribuye en gran manera a que los robots se sitúen en formación más fácilmente.

En contraposición a lo difícil que resulta situar a los robots en formación, una vez los robots están colocados, desplazar a la formación en línea recta resulta más sencillo. Esto es así porque, salvando el que los robots de atrás puedan entorpecerse, se bloquean mucho menos que al modificar las formaciones.

Respecto al cálculo de trayectorias. Dijkstra calcula, en algunas situaciones, mejores trayectorias que A*, pero es mucho más lento computacionalmente. Por esta razón, se recomienda A* como método por defecto, quedando Dijkstra relegado a situaciones nicho donde sea estrictamente necesario.

Un detalle a destacar fue el rendimiento del PC con Gazebo. Debido a lo mal que funcionaba sobre el ordenador disponible, no quedó más remedio que cambiar de simulador. Para poder usar con garantías este tipo de simuladores en 3D se necesita un ordenador con muchos recursos, especialmente memoria RAM y de video.

Finalmente, se ha apreciado la importancia y posibilidades que ofrece ROS. Aunque tal vez no sea la mejor opción para todos los tipos de robots, es muy probable que acabe consolidándose como referente a nivel global en el tema de la robótica.

5.2 Posibles mejoras y trabajo futuro

Las posibilidades para seguir trabajando a partir de este proyecto son muchas, a continuación, se comentan algunos caminos que se podrían seguir.

Una de las opciones consiste en implementar las estrategias de manera distribuida, es decir, que cada robot realice su parte del cómputo sin necesidad de una unidad central que dirija a todos. Este es, uno de los aspectos más punteros en el ámbito de los sistemas multi-robot, el poder repartir el mando entre todos los agentes. Esto implicaría prescindir de Matlab e implementar todo únicamente en ROS. Habría que añadir todos los nodos, paquetes y algoritmos necesarios para poder efectuar una buena

comunicación entre los robots. Además de revisar la implementación de todas las estrategias.

Continuando por este camino se podría plantear una implementación sobre robots reales. Aquí, el principal problema es que haría falta trabajar mucho el sistema de localización.

Respecto a las estrategias, también tienen margen de mejora. Por ejemplo, se podría empezar por buscar mejores maneras para situar a los robots en formación, esta cuestión se podría considerar ya un arte y todo dependería de las horas invertidas.

Muy ligado a esta última cuestión, también se podrían mejorar los criterios para la asignación de objetivos a cada robot. De esta manera se facilitaría en gran medida el colocar a los robots en sus puestos.

Respecto a las estrategias para mover a la formación, se han propuesto dos ideas relativamente sencillas, que son rotarla sobre sí misma y desplazarla en línea recta. Partiendo de aquí se podrían desarrollar métodos más sofisticados como intentar que la formación sea capaz de seguir arcos de circunferencia.

6 Referencias

[1] W.Ren and N. Sorensen, “Distributed coordination architecture for multi-robot formation control,” 2007. [Online]. Available: "https://www.sciencedirect.com/science/article/pii/S0921889007001108" . [Accessed: 17-Oct-2018].

[2] “Documentación sobre ROS,” [Online]. Available: http://wiki.ros.org/. [Accessed: 17- Oct- 2018].

[3] J.M.P. Gascós, E.Montijano, C.Sagües and S.Llorente, “Distributed Coverage Estimation and Control for Multirobot Persistent Tasks,” *IEEE Transactions on Robotics,* vol.32, no. 6, pp. 1444-1460, December 2016.

[4] L.Sabattini, A.Gasparri, C.Secchi, and N.Chopra, “Enhanced connectivity maintenance for multi-robot systems,” in *Proc. 10th Int. IFAC Symp. Robot Control*, 2012, pp. 319-324.

[5] M. Schwager, D.Rus, and J.J.Slotine, “Unifying geometric, probabilistic, and potential field approaches to multi-robot deployment,” *The International Journal of Robotics Research,* vol. 30, no. 3, pp. 371-383, 2011.

[6] D.V.Soriano, “Planificación y control con restricciones de formaciones de robots”, Universidad de Zaragoza, 2017.

[7] J.Cortés, S.Martinez, T.Karatas and F.Bullo, “Coverage control for mobile sensing networks,” *IEEE Transactions on Robotics and Automation*, vol.20, pp. 243-255, April 2004.

[8] C. N. Hadjicostis and M. Cao, “Distributed Algorithms for Voronoi Diagrams and Applications in Ad-hoc Networks,” 2003. [Online]. Available: http://hdl.handle.net/2142/99588. [Accessed: 17- Oct- 2018].

[9] “Documentación sobre Matlab,” [Online]. Available: https://es.mathworks.com/products/matlab.html. [Accessed: 17- Oct- 2018].

[10] Documentación sobre Gazebo,” [Online]. Available: http://gazebosim.org/. [Accessed: 17- Oct- 2018].

[11] Documentación sobre Stage,” [Online]. Available: https://codedocs.xyz/CodeFinder2/Stage/modules.html. [Accessed: 17- Oct- 2018].

[12] J. Tardós, “Estrategias multi-robot de despliegue y cobertura con mantenimiento de la conectividad”, Universidad de Zaragoza, 2017.

[13] Dijkstra's algorithm," [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm. [Accessed: 17- Oct- 2018].

[14] "A* search algorithm," [Online]. Available: https://en.wikipedia.org/wiki/A*_search_algorithm. [Accessed: 17- Oct- 2018].