Departamento: Tecnología de Computadores y de las Comunicaciones

Memoria de la asignatura de cuarto curso: Robótica

Juan Antonio Silva Luján
Escuela Politécnica, Universidad de Extremadura, Avenida de la Universidad Cáceres, Extremadura, España
jsilvalu@alumnos.unex.es
http://www.linkedin.com/in/jsilvalu/

Rodrigo Puerto Pedrera
Escuela Politécnica, Universidad de Extremadura, Avenida de la Universidad Cáceres, Extremadura, España}
ropuertop@alumnos.unex.es
http://www.linkedin.com/in/ropuertop/

Para la asignatura de robótica se ha desarrollado un proyecto basado en la implementación de un componente para dotar de movimiento a un robot eficientemente. Este robot debe ser capaz de recorrer el mayor porcentaje de suelo posible en un rango de tiempo limitado, llegar a un punto del mapa virtualizado definido por el usuario, etc.

For the robotics subject, we develop a project based on the implementation of a component has been developed to provide a robot with movement efficiently. This robot must be able to travel the highest percentage of soil possible in a limited time range, reach a point on the user-defined virtualized map, etc.

Keywords: robot; obstáculos; interfaces; componentes; sincronización; laser.

1. Barrido feature

En esta primera práctica se propone resolver la ocupación del mayor porcentaje del mapa virtualizado propuesto por el profesor de la asignatura. Esta ocupación debe ser realizada de la forma más eficiente posible, recorriendo la totalidad del mapa en el menor tiempo posible. En los siguientes apartados se muestra la resolución propuesta por los alumnos que firman este documento.

1.1. *Solución propuesta*

La lógica del desarrollo del proyecto ha cambiado en varias ocasiones (teniendo que volver a implementar el proyecto desde cero); las causas serán abordadas en el apartado "3.4 Problemas encontrados" de la presente documentación. La idea llevada a cabo consiste en que el robot, en un estado inicial, busque una esquina y se posicione en ella (véase en Fig.1 Búsqueda de la esquina).

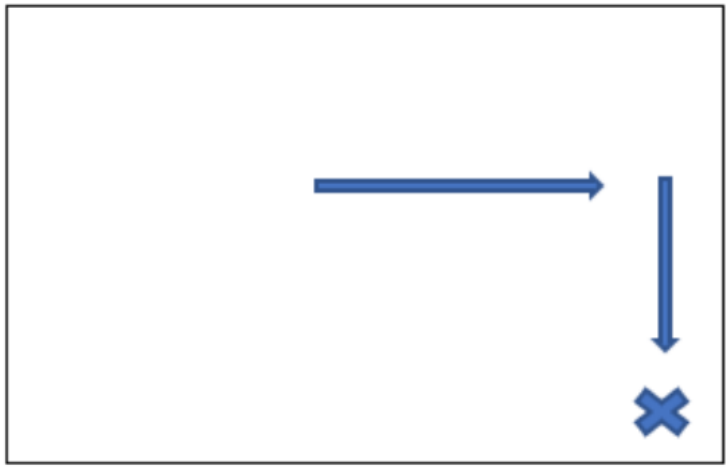

Fig. 1. Búsqueda de la esquina.

En este momento, el robot comenzará a realizar un recorrido horizontal en el que cada vez que encuentre una pared o un obstáculo girará, hará un pequeño avance y volverá a girar para realizar el nuevo recorrido (Fig. 2. Recorrido Horizontal).

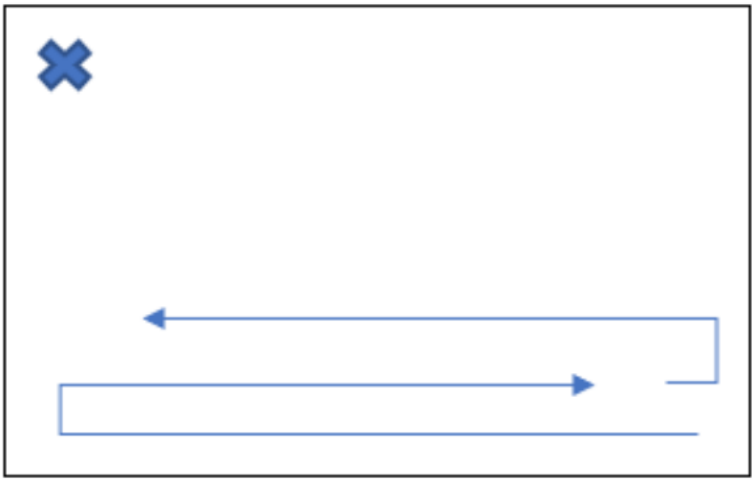

Fig. 2. Recorrido horizontal.

Esta práctica concluirá en el momento en el que el componente llegue a la última de las esquinas del mapa, en ese momento llevará a cabo un recorrido vertical. El

sentido de dicho tipo de recorrido consiste en que si existen espacios no recorridos entre obstáculos de forma horizontal, con este recorrido vertical, el robot podrá recorrer todo el suelo del mapa (véase en Fig. 3. Recorrido Vertical).

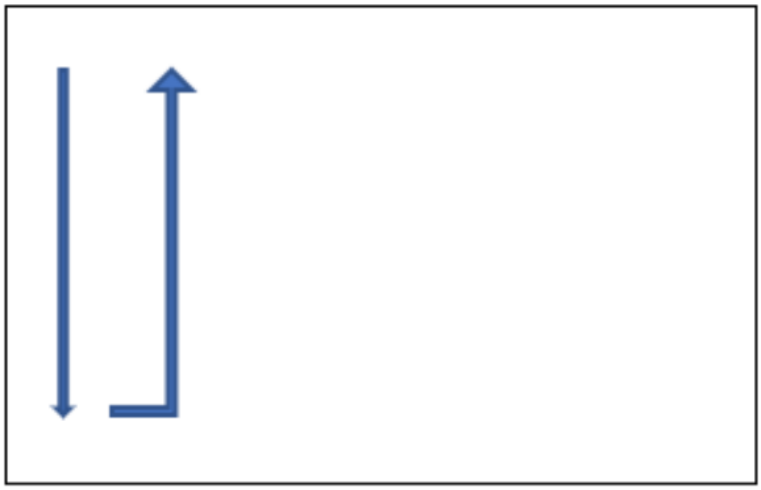

Fig. 3. Recorrido vertical.

Aunque en las ilustraciones no se muestran obstáculos por el camino, están considerados en la ejecución de la práctica. Hemos concluido que tan solo con el recorrido horizontal, el porcentaje total recorrido es muy prometedor pero, en caso de poder recurrir a mayor tiempo de ejecución, se puede llevar a cabo el recorrido vertical para obtener mejores resultados.

1.2. *Implementaciones críticas*

Durante el desarrollo del algoritmo principal basado en la lógica de la máquina de estados que se describirá en el siguiente apartado han surgido ciertas implementaciones destacables en las que se basa el funcionamiento del robot durante la ejecución: Un aspecto destacable es que el robot es capaz de distinguir en todo momento si se topa con una pared o un mero obstáculo.

Esta diferenciación es posible gracias al control de coordenadas. El componente toma los valores máximos y mínimos de las coordenadas de los ejes principales y opera en consecuencia. Para nuestro algoritmo, el robot debe ser capaz de diferenciar estos aspectos, para ello, cuando busca la primera esquina se realiza una invocación al método bool esquinaEncontrada() que toma como referencia las coordenadas mínimas y máximas de los ejes para conocer su estado actual, además,

hemos delimitado una "zona roja" gracias a un umbral de error aplicado a los márgenes del mapa para que el robot pueda llegar correctamente y saber hacia que dirección debe girar.

Una vez que la invocación al método retorna en valor booleano TRUE, el robot se dispone a realizar el recorrido horizontal delimitando esa "zona roja" y conociendo si se encuentre cerca de una pared. Llegado este momento, el robot realiza giros a la izquierda o la derecha dependiendo del valor de referencia de la esquina desde la que ha partido. La lógica que sigue el algoritmo se basa en que la realización de este barrido realiza giros opuestos en cada iteración. Para un correcto recorrido barriendo horizontal y verticalmente los giros izquierdos y derechos deben ir oscilando de uno a otro continuamente, de esta forma se logrará este recorrido. Dada la anterior explicación, el robot debe saber a dónde girar para recorrer el mayor rango posible de mapa.

1.3. *Máquina de Estados*

La lógica de la máquina de estados del nuevo componente se divide en MacroEstados, cada uno de ellos en varios MicroEstados, de esta forma, podemos modularizar el problema recurriendo a una forma jerárquica muy comprensible para otros programadores. Los MacroEstados implementados son:

- Esquina
- RecorridoH
- RecorridoV

Cada uno de ellos está formado por los MicroEstados:

- IDLE
- GO
- IZQ
- DER

Podemos representar gráficamente la máquina de estados anterior en la Figura 4. Grafo de estados. Así mismo, cada uno de los estados cuenta con estos subestados

Fig. 4. Grafo de estados.

o MicroEstados para poder controlar todos los aspectos de la ejecucion (véase en Figura 5. Jerarquía de MicroEstados).

Fig. 5. Jerarquía de MicroEstados.

Ejecución	Porcentaje
1	19.03%
2	20.33%
3	18.56%
4	24.59%

Table 1. Ejecuciones iniciales

1.4. *Resultados de la ejecución*

Los resultados obtenidos siguiendo la lógica detallada anteriormente en las ejecuciones iniciales fueron mejorados notablemente en los resultados de las ejecuciones finales. En algunas fases iniciales, las cuales, estaban a falta de implementar soluciones todavía no encontradas, los resultados de las ejecuciones pueden observarse en la Tabla 1.

Cuando empezamos a obtener resultados más significativos en fases posteriores los resultados mejoraron considerablemente puesto que el algoritmo completo ya funcionaba correctamente y nos estábamos preocupando de optimización. En estas circunstancias logramos los resultados de la Tabla 2.
En nuestra última ejecución, el recorrido completo a logrado alcanzar un 65% en el mapa simpleworld.Dependiendo de la cantidad de tiempo con la que se mida el porcentaje, el resultado puede variar pero esta última tabla recoge resultados muy significativos.

1.5. *Problemas encontrados*

Durante el inicio del cuatrimestre, la lógica inicial era añadir a una estructura de datos las coordenadas de visitados y realizar un recorrido en espiral, este algoritmo

Ejecución	Porcentaje
1	29.05%
2	34.45%
3	36.24%
4	41.55%
5	45.89%
6	52.27%
7	58.23%
8	65.13%

Table 2. Ejecuciones finales

genera una solución muy eficiente ya que no recorre coordenadas ya recorridas y asegura el 100% del recorrido. No obstante, después de un par de semanas de trabajo y desarrollo encontramos la siguiente problemática: El robot nunca tiene una exactitud tan concreta como para que las coordenadas almacenadas pudieran repetirse, es decir, nunca recorría exactamente el mismo terreno en la versión espiral, intentamos solventar este problema realizando redondeos sobre los valores en milímetros de las coordenadas pero resultó imposible de implementar. Semanas después, el docente de la asignatura nos orientó a otra posible solución:

Utilizar el grid.h incorporado en el repositorio del proyecto para poder almacenar las "baldosas" del mapa y poder consultar la lista de vecinos, así el robot podría ejecutar un algoritmo de mano derecha en el que siempre recorre "baldosas" que nunca ha llegado a pisar. Resultaba una solución muy eficiente y más fácil de implementar. El problema fue que tras tres semanas de trabajo y creyendo que debíamos hacer uso del grid.h, nos encontramos con que las "baldosas" son de tamaño milimétrico y no del tamaño del robot por lo que de nuevo y a pocos días de la entrega del nuevo componente tuvimos que volver a iniciar el desarrollo desde 0 hasta pensar la solución actual. En nuestra versión final hemos resuelto el problema teniendo en cuenta que para realizar los giros tenemos en cuenta que el mapa se divida en dos hemisferios:

- Hemisferio Norte
- Hemisferio Sur

El robot parte del hemisferio en el que ha encontrado la esquina, en este caso, el hemisferio sur, podemos ver la diferenciación entre los hemisferios en la Figura 4. Hemisferios. Por consiguiente, ejecutará el algoritmo en base a que se encuentra en dicho hemisferio de la siguiente forma: Cuando llega el momento de toparse con una pared y realizar el giro, el robot evalúa a que dirección debe girar tomando como referencia la mayor distancia, evidentemente, si gira a la dirección donde exista más distancia, recorreremos espacio todavía inexplorado mientras que la distancia

menor es el terreno que ya hemos recorrido (véase en Figura 5. Elección de giro). Una vez que sobrepasamos el "ecuador" es decir, sobrepasamos el punto 0 del

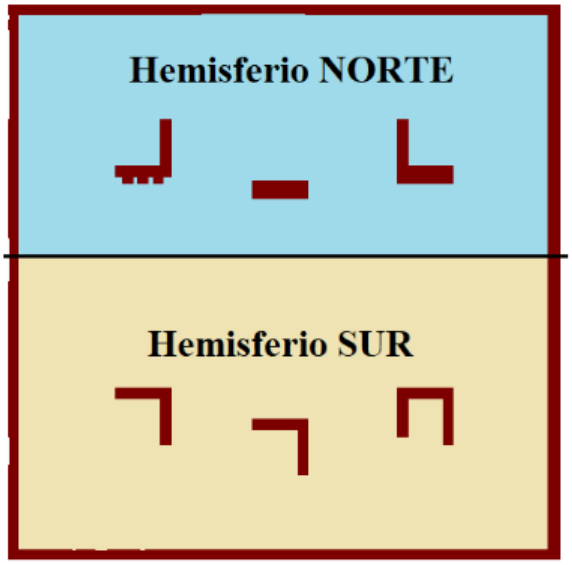

Fig. 6. Hemisferios.

mapa, se produce un cambio de hemisferio y el robot actúa justo con la lógica inversa a la que se ha detallado anteriormente. De esta forma el robot recorrerá horizontalmente todo el mapa y siempre realizará giros en direcciones eficientes. No obstante, la versión definitiva ha resultado ser bastante más sencilla que esta puesto que podemos solventar el problema oscilando los giros izquierda y derecha en cada iteración para lograr el barrido.

1.6. *Conclusión grupal*

Desde los inicios, el desarrollo del componente llevado a cabo en la asignatura ha sido complejo. Implementar una solución a un problema complejo del mundo real aplicado al comportamiento que debe tener el componente no ha sido tarea fácil. Aunque teníamos una breve idea, no sabíamos como abordar el problema desde un principio y se ha invertido una gran cantidad de tiempo y esfuerzo en desarrollos a los que hemos tenido que renunciar por falta de transparencia en como deberíamos actuar de cara a la asignatura.

La entrega del nuevo componente ha sido implementada en tres ocasiones distintas desde cero. En primer lugar, hemos usado un mapa de coordenadas que no nos proporcionaba la exactitud con la que deberíamos trabajar. La segunda opción fue

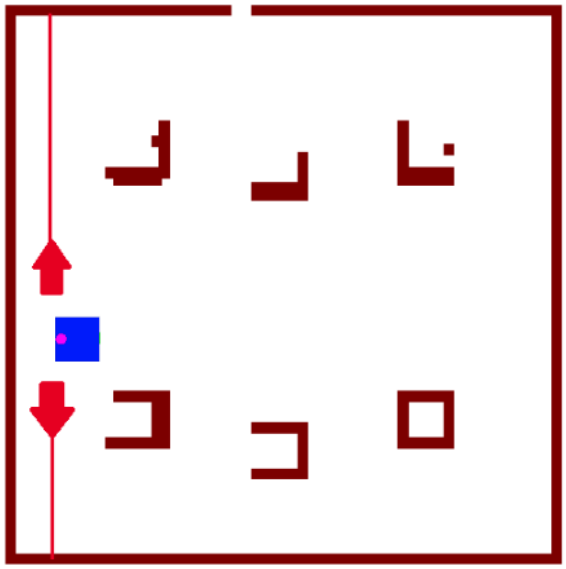

Fig. 7. Elección de giro.

aplicar el grid.h para usar los vecinos desconociendo el problema del tamaño de dichos vecinos, que sean tan pequeños no nos servía para abordar nuestra solución y agrandar el tamaño a un valor de 400 mm presentaba demasiados inconvenientes a la hora de representar el mapa por lo que hemos tenido que acudir a esta tercera solución. Todos estos aspectos nos han hecho invertir demasiado tiempo y esfuerzo que podrían haber sido utilizados para optimizar la solución para que fuera lo más eficiente posible.

Por otro lado, el abordaje de la práctica ha sido muy didáctico y útil, hemos trabajado con datos y variables fieles al mundo real de una forma muy inmediata, algo novedoso en los estudios que ocupamos puesto que nunca hemos realizado un proyecto de este tipo en este lenguaje de programación.

En conclusión, el desarrollo de este proyecto ha sido tan laborioso como didáctico pero nos hemos visto frustrados en muchas ocasiones por tener que renunciar al trabajo de semanas para comenzar la implementación desde cero en tres ocasiones distintas.

2. Imperium

El desarrollo del componente tiene el siguiente comportamiento: el usuario "clickeará" en un punto concreto del mapa (un par de coordenadas del mapa) y el robot deberá acudir a dicho punto inmediatamente. Evidentemente, el robot puede

toparse con uno o varios obstáculos de camino al punto de destino y deberá sortearlos correctamente. El nombre del nuevo componente "Imperium" viene por el mítico juego de estrategia Imperium donde los personajes del videojuego acuden al punto indicado por el usuario del mapa.

2.1. *Solución propuesta*

Cuando el usuario "pincha" en un punto concreto del mapa, el método RCISMousePickersetPick(Pick myPick) aplica la coordenada y la almacena para su uso, de esta forma ya tenemos constancia de cual es el punto objetivo.

Debido al tamaño del grid, hemos tenido en cuenta la necesidad de aplicar un umbral de error de 100 mm en las posiciones derecha, izquierda, superior e inferior. De esta manera, podemos asegurar que el robot llegará a la coordenada seleccionada por el usuario con un umbral de error mínimo. En ese momento, el robot se parará. Para llevar a cabo esta comprobación se usa el método regionobjetivoDIOS enviándole por parámetros las coordenadas x y z. Este método únicamente retorna el valor booleano TRUE si nos encontramos en la región objetivo definida por el usuario y FALSE en caso contrario.

Cuando el robot inicia la máquina de estados que se detallará en el apartado a continuación, es posible que entre en un estado denominado OBSTACLE, dicho estado se define durante la ejecución cuando el robot se encuentra con un obstáculo frente a él, en ese momento, el robot deberá ejecutar un algoritmo de mano derecha para rodear el obstáculo hasta volver a alinearse con la recta definida entre el punto de origen y el punto de destino.

Para poder comprobar si el robot se encuentra con un punto perteneciente a una recta se hace uso de la función pinline cuya expresión artimética sería la siguiente (1):

$$d = (Y2 - Y1) * X0 + (X1 - X2) * Y0 + (X2 * Y1 - Y2 * X1) \tag{1}$$

Dónde:

- P1 (X1,Y1) - Punto de origen
- P2 (X2,Y2) - Punto objetivo
- P0 (X0,Y0) - Punto actual

Todos los detalles mencionados en este apartado pueden observarse en la Figura 8. Esquema general de comportamiento.
Así mismo, el mismo algoritmo reiniciará el mismo comportamiento si encuentra en lugar de uno, n obstáculos. Se alineará, esquivará, comprobará si el punto pertenece a la recta y repetirá este proceso hasta que alcance la región objetivo.

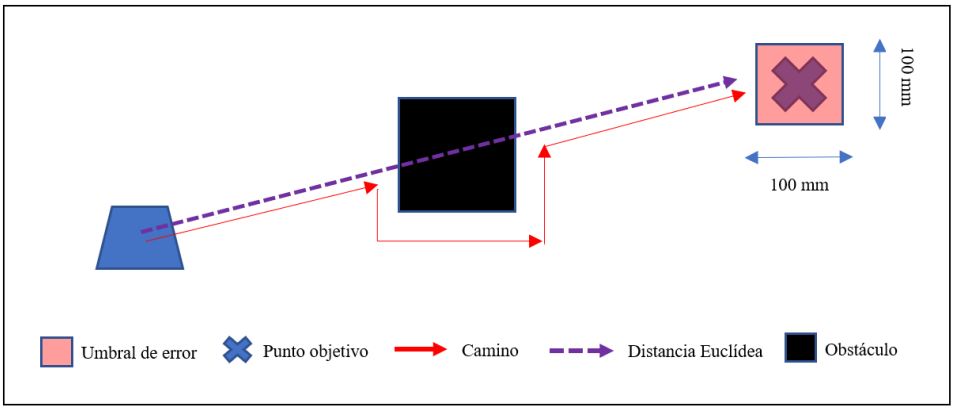

Fig. 8. Esquema general de comportamiento.

2.2. *Implementaciones críticas*

En este apartado se detalla el apartado de eludir un obstáculo. Para ello haremos uso del laser proporcionado por el robot y las coordenadas del robot con respecto al mapa virtualizado. A continuación se detalla el uso de cada idea:

(1) **Componente laser del robot**: Su utilización consistirá en saber si el robot tiene el lado derecho libre. En el caso de que el robot tenga el lado derecho libre podrá alinearse con respecto al eje que corresponda. Una representación de esta alineación la podemos ver en la Figura 9. Alineamiento del robot con respecto al objetivo.
(2) **Coordenadas del robot**: Puesto que los obstáculos ocupan cuadrados alineados perfectamente con respecto a los ejes del mapa, podemos establecer una lógica de alineamiento perfecto. El robot cuando se topa con un obstáculo deberá alinearse con respecto al eje x o el eje z (dependiendo en que dirección esté ubicado) y avanzar a una velocidad hasta encontrar su mano izquierda libre. Podemos ver la idea en la Figura 10. Representación del ángulo del robot con respecto al mundo.

En definitiva, la lógica establecida para la superación del obstáculo es el algoritmo de la mano izquierda.

2.3. *Máquina de estados*

En esta práctica tendremos varias posibles situaciones con las que el robot puede encontrarse: necesidad de alinearse con el objetivo, parar cuando se encuentre en la región objetivo, esquivar un obstáculo y necesidad de establecer una velocidad para avanzar.

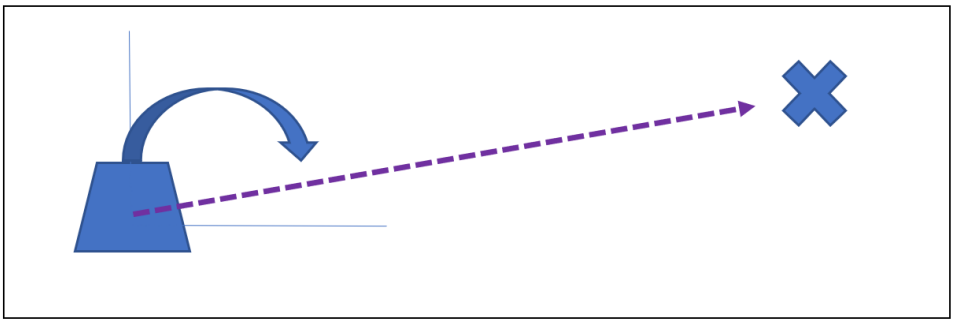

Fig. 9. Alineamiento del robot con respecto al objetivo.

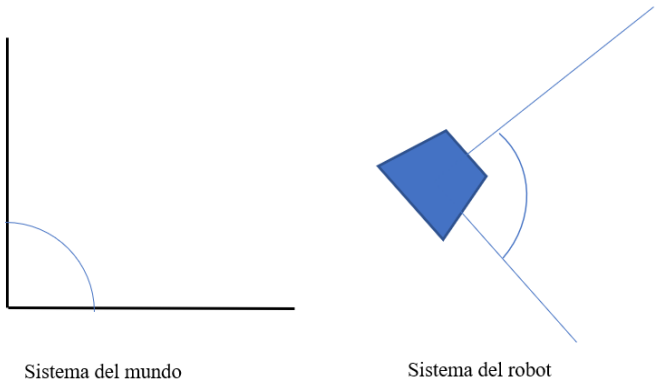

Fig. 10. Representación del ángulo del robot con respecto al mundo.

A diferencia de la primera práctica propuesta, en este caso no serán necesarios los sub-estados. Podemos diferenciar entre varios estados:

- IDLE
- CONDITIONS
- RUN
- OBSTACLE
- ALIGN

Es obvio que el robot no necesita hacer nada cuando se encuentra en el objetivo dado por el usuario, por ello existirá un estado llamado *idle*. Cuando el usuario clicka sobre un punto del mapa virtualizado, será necesaria la alineación con el objetivo; para ello se crea el estado ALLIGN. Una vez alineado, se establecerá una velocidad al robot: estado RUN; el cuál pasa a estado CONDITIONS (véase en Figura 11. Esquema de máquina de Estados).

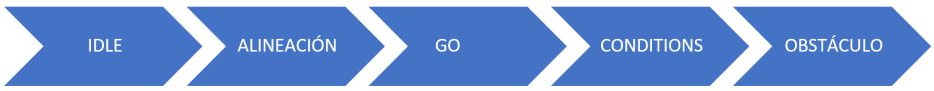

Fig. 11. Esquema de máquina de Estados.

Este último estado se encargará de comprobar si el robot se encuentra en la zona objetivo o si simplemente tiene delante un obstáculo. Estas dos condiciones derivan en dos estados diferentes: si nos encontramos en la región objetivo, el robot no necesitará realizar ninguna acción más; y si el robot se topa con un obstáculo en su camino, pasará al estado OBSTACLE (el cuál se encarga de establecer la rutina de la mano derecha para rodear el obstáculo y seguir su camino). Para poder realizar el rodeo del obstáculo se usan unos estados aplicados dentro del estado OBSTACLE:

- IDLE
- ALINEACION
- GO
- INICIAL

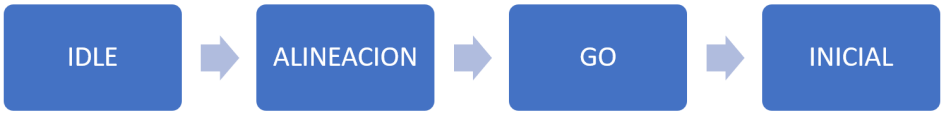

Fig. 12. Esquema de máquina de Estados para OBSTÁCULO.

2.4. *Resultados de ejecución*

Inicialmente, el desarrollo de la entrega se ha realizado teniendo en cuenta un mundo en el que no existían obstáculos, es decir, el robot debería llegar al objetivo sin necesidad de ejecutar algoritmos relacionados con un bug. De esta manera, podríamos ir desarrollando los métodos para asignar las coordenadas de destino, la alineación y la detección de la región objetivo alcanzada, todas ellas con su respectiva máquina de estados detallada en la sección "2.3. Máquina de estados".
Como podemos observar en "figura 13. Alineación y aproximación al objetivo", inicialmente, el robot espera a que el usuario seleccione el objetivo, a continuación produce la alineación con el mismo y, en última instancia, se aproxima hacía él hasta alcanzarlo.

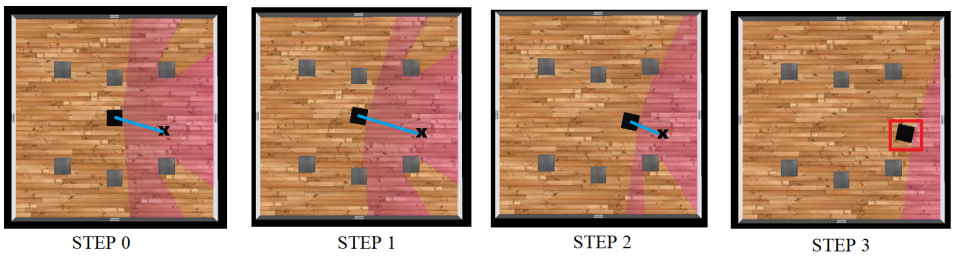

Fig. 13. Alineación y aproximación al objetivo

Este objetivo ha sido alcanzado la semana del 11 de Noviembre exitósamente. Posteriormente, hemos comenzado a actualizar la máquina de estados para adaptarla al caso de que existan obstáculos, implementando la ecuación de la recta y la determinación de si el punto pertenece para forzar la salida del estado obstáculo, es decir, si se ha rodeado completamente el obstáculo. Este objetivo ha sido alcanzado la semana del 25 de Noviembre con los siguientes resultados:

2.5. *Problemas encontrados*

El único problema con el que nos hemos encontrado ha sido la forma con la que el robot esquivará el obstáculo. Estableciendo distintas soluciones (como el control de la distancia del laser, la creación de un poligono, etc.), se ha llegado a la conclusión de que la forma más eficiente de hacerlo fuese la siguiente:

- El robot debe establecer en un inicio la linea que se establece cuando el usuario pincha en el mapa; esto lo hace viendo el punto inicial (coordenadas) donde se encuentra el robot y el punto final (que serán las coordenadas establecidas por el puntero del ratón).
- Teniendo en cuenta esta línea, el robot cuando se encuentre con un obstáculo mantendrá la mano derecha pegada a él hasta encontrarse en un punto (coordenadas) pertenecientes a la línea establecida.
- Si el robot se encuentra en esta situación, seguirá los siguientes cálculos:
  - Una vez el robot se ha encontrado con un obstáculo, necesitará guardar en una variable la distancia euclidea que le falta para llegar al punto objetivo.
  - Sabiendo esta distancia euclidea inicial, el robot comprobará su distancia actual con ella; comprobando si está más cerca o más lejos del punto objetivo. Con esto salvaremos el posible problema de encontrarnos en una habitación encerrados.

En definitiva, con esta solución los cálculos para establecer la condición de salida

del estado "OBSTACLE" se reducirían al cálculo de la distancia euclidea actual con respecto al punto objetivo y la comprobación de ésta con la distancia calculada inicialmente; siendo esta solución la más eficiente posible conocida.

2.6. *Conclusión grupal*

Ambos miembros del equipo hemos notado una considerable mejora en torno a la adaptación del trabajo para la nueva entrega práctica. Evidentemente, en nuestra primera entrega, existían numerosas métricas que no terminábamos de comprender completamente, no nos encontrábamos totalmente familiarizados con el modelo de trabajo para el desarrollo del choca choca. No obstante, a medida que se le ha dedicado más horas de trabajo a la asignatura hemos ido notando una mejora didáctica del empleo del tiempo, es decir, durante el desarrollo de esta práctica hemos logrado obtener más calidad de trabajo en menor cantidad de tiempo. Además, hemos llegado a conclusiones con una mayor convergencia de opiniones entre ambos miembros del grupo juro.robotics. Ha resultado ser una práctica interesante por la aplicacón matemática que suponen ciertos comportamientos del robot como por ejemplo:

- **Alineación**: Transformación del sistema de referencia del robot con el del mundo para alinearse con el objetivo concreto.
- **Salida del estado OBTÁCULO**: Determinación matemática para comprobar si un punto pertenece a una recta y así poder completar el proceso de rodeo del obstáculo.
- **Determinación de región objetivo**: Estimación del umbral de error para el tamaño del grid cuando el robot alcanza su objetivo.

En definitiva, consideramos que nos hemos desenvuelto con más práctica en el trabajo práctica durante el desarrollo del proyecto y en la elaboración de la documentación ya que en la primera versión hemos tenido fallos en el formato y en la forma de representar los aspectos de la memoria.