UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA INDUSTRIAL
ESPECIALIDAD ELECTRÓNICA INDUSTRIAL

PROYECTO FINAL DE CARRERA

ALGORITMOS DE PLANIFICACION DE ROBOTS USANDO FAST MARCHING

Autor: Miguel Ángel Castaño León

Tutor: Luis Santiago Garrido Bullón

AGRADECIMIENTOS

En primer lugar quiero darles las gracias a mis padres por su apoyo incondicional ante todos los retos a los que me he enfrentado a lo largo de mi vida. Sin la educación y los principios que me han transmitido yo nunca hubiera podido llegar tan lejos y hacer realidad este sueño tanto tiempo anhelado. Gracias por estar a mi lado en los buenos y malos momentos.

Gracias a la Dra. Ana Mª Castaño por su inestimable ayuda en los difíciles comienzos de la carrera, tengo que destacar en especial el apoyo prestado en los conocimientos de cálculo e inglés. Para mí ha sido un ejemplo de constancia y perseverancia de los que creo que una persona no puede carecer en especial para conseguir este tipo de metas.

Gracias a mi hermano Carlos Alberto por la ayuda prestada durante mis duros comienzos con la programación en "C".

Tengo que dar las gracias a Oscar García por su inestimable aportación de conocimientos de economía de empresa que tanto me han ayudado a superar las asignaturas donde éstos eran requeridos.

Gracias a Celia García por la ayuda prestada durante el desarrollo de este proyecto y a la paciencia que ha tenido durante estos duros y largos años de carrera.

Quiero agradecer especialmente la ayuda y paciencia que ha mostrado hacia mi Luis Santiago Garrido tutor de este proyecto, sin él este proyecto no hubiera sido posible.

Por último no puedo terminar este apartado sin nombrar a mis compañeros de carrera de los que destaco su apoyo moral y ayuda en los malos momentos.

Gracias por vuestra ayuda y apoyo: David Solís, Andrés Sacristán, Rafael Jurado, Roberto Covarrubias, Ana Paula Mateo, Alberto Peña y Alberto Palomo.

RESUMEN

El objeto de este proyecto es conseguir que el robot LEGO NXT Mindstorm siga lo más fielmente posible una ruta que le es planificada mediante un algoritmo planificador (ya desarrollado por el Departamento de Sistemas y Automática de la Universidad Carlos III) basado en la metodología Fast Marching (marcha rápida)

Para el seguimiento de la ruta por el robot vamos a necesitar por una parte un software desarrollado mediante la aplicación informática Matlab (2007b) y un sistema de relocalización mediante el cual podemos conocer con una gran precisión donde se encuentra el robot dentro del mapa que hemos cargado.

El sistema de relocalización se basa en el uso del algoritmo "Differential Evolution" ya implementado en un proyecto anterior. Se trata de una especie de algoritmo genético mejorado, se parte de una población inicial en la que se introduce una perturbación (diferencia de dos vectores) y dicha perturbación se suma a un tercer vector para obtener un vector de prueba. Cada uno de los vectores correspondientes a los puntos de la población inicial, competirá con el vector de prueba que tenga su mismo índice pasando a la siguiente generación los individuos más aptos. Por cada una de las sucesivas generaciones se repetirá el proceso de introducción de perturbación, crear vector de prueba y el proceso de competición o selección de los individuos que habrán de pasar a la siguiente generación.

Cuando el algoritmo ha convergido totalmente (tras el paso de sucesivas generaciones) obtenemos la solución con mejor valor de la función de Fitness o función de salud. La función del algoritmo "Differential Evolution" es obtener la mejor función de salud (Fitness) o minimizar la función de coste (Cost).

En lo referente a este proyecto, el algoritmo “Differential evolution” se encarga de minimizar la diferencia entre los vectores V0 y V1, siendo V0 el vector que contiene las mediciones realizadas por el sensor de ultrasonidos del entorno que rodea al robot y V1 el vector que contiene las medidas estimadas que se obtendrían si el robot se encontrara donde nosotros creemos que está. El cuadrado de la diferencia entre V0 y V1 se denomina error cuadrático.

El sistema de relocalización nos permite hacer las correcciones necesarias en la ruta para que solventando el inconveniente de la acumulación de errores en un seguimiento de ruta odométrico se llegue hasta el punto de destino convenido.

ABSTRACT

The purpose of this proyect Is to make the LEGO NXT Mindstorm robot follow as closely as possible a route that is planned by a scheduler algorithm (already developed by the Department of Systems and Automation, at Carlos III University in Madrid) based on the Fast Marching method (quick start).
In order for the robot to monitor the route, we will need the following. On the one hand, we will need software developed using the Matlab IT (information Technology) application (2007b). On the other hand, we will need a relocation system to achieve the maximum precision about it's location in the map.

The relocation system is based on the use of the algorithm "Differential Evolution". This has been already implemented in a previous draft. It is a kind of improved genetic algorithm. It starts from an initial population in which a disturbance (difference between two vectors) is introduced. This disturbance is added to a third vector in order to get a test vector. Each one of the vectors corresponding to the initial population points will compete with the test vector with the same index. The most skilled individuals will move into the next generation.

When the algorithm has completely converged (after the passage of consecutive generations) we can get the best value solution of the fitness function (or health function). The function of the algorithm "Differential Evolution" is to get the best health function (Fitness) or to minimize the cost function (cost).

With regard to this proyect, the algorithm "Differential Evolution" makes sure the minimization of the difference between vectors V0 and V1. Vector V0 contains the measurements of the robot environment, taken by the ultrasonic sensor. Vector V1 contains the estimated measurements if the robot were located where we think it is. The difference between V0 and V1 squared is called the quadratic error.

The relocation system allows us to make the necessary corrections to the route so that it is possible to get to the agreed point of destination by solving the problem of error accumulation in an odometric route tracking.

INDICE GENERAL

Agradecimientos.....2

Resumen.....3

Índice General.....6

Índice de figuras.....8

Índice de ecuaciones.....12

Motivación y Objetivos.....13

1. Introducción.....14
2. Estado del arte.....15

2.1 Robótica..... 15

2.2 Reseña histórica sobre la robótica.....16

2.3 Movilidad y navegación de robots.....18

2.3.1 Estudios realizados para la planificación y exploración de rutas en robots móviles.....20

2.3.2 Método de navegación de robots basado en metodología Fast Marching.....23

2.4 Localización y percepción.....30

2.4.1 Medios de percepción.....30

2.4.2 Localización o determinación.....34
de la posición en robótica

2.5 Odometría.....36

3. Desarrollo.....41

3.1 Construcción de prototipo de robot Móvil mediante Kit de Lego NXT Mindstorm.....41

3.2 Comunicación con PC.....51

3.3 Firmware y aplicación informática para el desarrollo del software de control.....53

3.4 Funciones implementadas mediante programa Matlab para el seguimiento de la ruta planificada.....67

3.5 Relocalización.....72

3.5.1 Algoritmo "Diferential Evolution".....72

3.5.2 Funciones utilizadas para la relocalización.....80

4. Resultados experimentales.....81

5. Aportaciones.....97

6. Conclusiones y trabajos futuros.....98

7. Presupuesto.....100

Referencias.....101

Apéndices.....104

Apéndice A: funciones implementadas.....105

Apéndice B: código de funciones implementadas.....114

INDICE DE FIGURAS

Figura 1: *Procedimiento clásico de navegación*

Figura 2: *Método clásico con supervisor de ruta global*

Figura 3: *Reflexión de una onda*

Figura 4: *Refracción de la luz en objeto introducido en agua*

Figuras 5 y 6: Cambio de dirección de la luz cuando atraviesa diferentes medios.

Figura 7*:* *Trayectoria calculada directamente con FastMarchig*

Figura 8: *Mapa del entorno realizado mediante el método de Voronoi (mediante skeletonization)*

Figura 9: *Representación del potencial de la Transformada de Voronoi extendida*

Figura 10: *Trayectoria calculada mediante FastMarching sobre la Transformada de Voronoi Extendida.*

Figura 11: *Hardware de sensor ultrasonidos*

Figura 12*:* *Reflexión de una onda*

Figura 13: *Sistema de referencia móvil asociado al robot*

Figura 14: *Sistemas de estimación de la posición de un robot*

Figura 15: *Dibujo esquemático de un vehículo triciclo*.

Figura 16: *Sensor de contacto*

Figura 17: *Sensor de sonido (sound Sensor)*

Figura 18: *Sensor de óptico (Light sensor)*

Figura 19: *Sensor de óptico (Light sensor)*

Figura 20: *Esquema interno de servomotor*

Figura 21: *Cable de conexión tipo RJ12*

Figura 22: *Conexión con PC mediante cable USB*

Figura 23: *Conexión con PC mediante dispositivo bluetooth*

Figura 24: *Alimentación mediante pilas de tipo AA/LR6 de 1,5V*

Figura 25: *Vista lateral 1 robot*

Figura 26: *Vista frontal robot*

Figura 27: *Vista lateral 2 robot*

Figura 28: *Vista trasera robot*

Figura 29: *Vista de planta robot*

Figura 30: *Montaje final robot móvil*

Figura 31: *Dispositivo Bluetooth USB 200M NANO*

Figura 32: *Emparejamiento entre dispositivos bluetooth*

Figura 33: *Icono para conexión del ladrillo NXT al puerto serie COM11*

Figura 34: *Proceso para descarga de firmware a ladrillo NXT inteligente*

Figura 35: *Página principal de matlab*

Figura 36: *Actualización del 'path' de matlab*

Figura 37: *Añadido de directorios (Add folder)*

Figura 38: *Incidencia por el uso de dos instalaciones de matlab*

Figura 39: *Contenido del archivo startup.m*

Figura 40: *Vista del laberinto previa*

Figura 41: *Vista del cursor para definir ruta*

Figura 42: *Punto inicial de la ruta*

Figura 43: *Puntos inicial, final y ruta trazada entre ambos*

Figura 44: *Creación de población inicial*

Figura 45: *Introducción de perturbación*

Figura 46: *Mutación*

Figura 47: *Selección*

Figura 48: *Un vector de la nueva población es mutado mediante una perturbación aleatoria*

Figura 49: *Selección*

Figura 50: *Inicio de rutas propuestas*

Figura 51: *Trazado del primer tramo de la ruta (ruta1)*

Figura 52: *Diagrama polar primera relocalización (ruta 1)*

Figura 53: *Nueva ruta tras la primera relocalización (ruta1)*

Figura 54: *Trazado de segundo tramo de ruta (ruta 1)*

Figura 55: *Diagrama polar previo a relocalización 2º tramo (ruta 1)*

Figura 56: *Datos mostrados sobre la mejor "pose" tras la optimización (ruta 1)*

Figura 57: *Nueva ruta tras la segunda relocalización (ruta 1)*

Figura 58: *Trazado inicial ruta 2*

Figura 59: *Diagrama polar previo a la relocalización (ruta 2)*

Figura 60: *Datos sobre la mejor "pose" tras la optimización (ruta2)*

Figura 61: *Nueva ruta tras la relocalización (ruta 2)*

Figura 62: *Trazado de ruta inicial (ruta 3)*

Figura 63. *Diagrama polar previo a la relocalización (ruta 3)*

Figura 64. *Datos sobre la mejor "pose" tras la optimización (ruta 3)*

Figura 65. *Nueva ruta tras la relocalización (ruta 3)*

Figura 66. *Triángulo formado entre los puntos (Xrob,yrob) y (Xrut,yrut).*

INDICE DE ECUACIONES

1) *Distancia recorrida por el punto "P" en un intervalo de tiempo Δt*

2) *Variación en la orientación del vehículo $\Delta\vartheta$*

3) *Distancia recorrida por el punto P*

4) *Orientación del vehículo*

5) *Coordenada X del punto P*

6) *Coordenada Y del punto P*

MOTIVACION Y OBJETIVOS

La navegación y la localización de robots son los mayores problemas que se presentan a la hora de conseguir el movimiento de robots dentro de un entorno definido como es por ejemplo un mapa.

Se necesita conocer aquella ruta que permite la navegación con seguridad de tal modo que evite los obstáculos que pueda encontrarse en el camino.

Dado que el movimiento del robot por sí solo no es del todo preciso, en especial durante los giros va acumulando errores (muy comunes en movimientos realizados mediante la Odometría) y por ello para hacer dicho movimiento lo más preciso posible necesitamos un sistema de corrección que nos permita conocer con la mayor exactitud posible donde se encuentra el robot.

Los objetivos de este proyecto son:

* Seguimiento de la ruta definida por el algoritmo planificador.

* Relocalizar al robot periódicamente para que ante la previsible desviación de trayectoria debida a los errores acumulativos propios de la odometría, se evite que el robot móvil continúe desviándose de la ruta con el consiguiente riesgo de choque contra los obstáculos del entorno.

1. INTRODUCCION

Los retos que plantea la navegación de robots móviles son:

- Choques con su medio ambiente o entorno que les rodea
- Choques con obstáculos no previstos mientras el robot intenta alcanzar la meta.
- Planificar una ruta que lleve con seguridad al robot desde el punto de origen al punto de destino.

Las soluciones que da la robótica a estos retos se dividen en:

- Ante los choques contra el entorno se aplican métodos de planificación de rutas seguras
- Ante los choques contra obstáculos imprevistos se aplican métodos de evasión de obstáculos

Los métodos clásicos de navegación, se basan en la separación entre el módulo encargado de trazar una ruta entre el punto de origen y destino (módulo "global") y el encargado de hacer frente a los obstáculos imprevistos (módulo "local"), este último se basa en procesar en tiempo real (elevado ratio) la información aportada por los sensores. El uso de estos módulos por separado supone un elevado esfuerzo computacional.

Aun habiendo proporcionando al robot una ruta óptima, el seguimiento de la misma puede no resultar lo suficientemente preciso como se desearía. Esto es debido a que durante el seguimiento de la ruta se puede producir una acumulación de errores, lo que lleva al robot a un progresivo alejamiento de la ruta inicialmente planificada. Para solucionar este inconveniente se precisa la implementación de un sistema de relocalización que permita conocer la mejor posición o "pose" ocupada por el robot, tras lo cual se replanifica la ruta conservando el punto de destino.

2. ESTADO DEL ARTE

En este capítulo se hará una breve introducción en la evolución histórica de la robótica, la localización, movilidad y navegación de la robótica.

2. 1 ROBOTICA

La robótica es un campo de la tecnología donde se recogen otras disciplinas como son la mecánica, informática, automática, electrónica y organización.

La mecánica es necesaria para llevar a cabo la construcción del prototipo del robot mediante el ensamblado de los diferentes elementos que lo componen como son: piezas, motores, ordenador y sensores para la percepción del entorno por parte del robot.

Informática necesaria para la implementación del programa/as que ayudan al robot a realizar la tarea que se espera de él.

Automática necesaria para poder realizar procesos de forma repetitiva

Electrónica necesaria para implementar funcionamientos no incluidos dentro del sistema básico del robot.

Organización necesaria para definir como debe ser el movimiento a realizar por el robot.

Entre las muchas definiciones que existen para definir el concepto de "robótica", podemos destacar las siguientes:

- "Robótica, Técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales". Real Academia Española

- "La robótica es una rama de la tecnología, que estudia el diseño y construcción de maquinas capaces de desempeñar tareas repetitivas, tareas en las que se necesita una alta precisión, tareas peligrosas para el ser humano o tareas irrealizables sin intervención de una maquina."

- La Robótica consiste en el diseño de sistemas. Actuadores de locomoción, manipuladores, sistemas de control, sensores, fuentes de energía, software de calidad... Todos estos subsistemas tienen que ser diseñados para trabajar conjuntamente en la consecución de la tarea del robot" [Joseph L. Jones and Anita M. Flynn. Mobile robots: Inspirations to implementation. A K Peters Ltd, 1993]

2.2 RESEÑA HISTORICA SOBRE LA ROBOTICA

En la antigüedad el ser humano ideó máquinas que imitaban partes del cuerpo humano. Los antiguos egipcios dotaban a las estatuas de sus dioses de brazos mecánicos y en la antigua Grecia se construyeron estatuas dotadas de movimiento mediante sistemas hidráulicos [1].

Podemos fijar el inicio de la robótica actual en el S. XVIII cuando Joseph Jacquard inventó una máquina textil que era programable mediante el uso de tarjetas perforadas [2], después durante la revolución industrial se fomento el desarrollo de estos sistemas mecánicos. Así mismo entre los siglos XVII y XVIII se construyeron muñecos mecánicos con características de robots.

La palabra robot aparece por primera vez en 1920 en una obra literaria escrita por el dramaturgo checo Karel Capek titulada "Los Robots universales de Rossum"[3].

El término “robótica” se atribuye a Isaac Asimov y este enuncia las denominadas tres leyes de la robótica [4].

Existen diferentes factores para que se desarrollen los primeros robots en la década de los 50’s. La investigación de Inteligencia artificial consigue emular mediante computadoras el procesamiento de información humana.

En 1954 Devol diseña el primer robot programable

En 1960 se introdujo el primer robot "Unimate", basada en la transferencia de artículos.
En 1961 Un robot Unimate se instaló en la Ford Motors Company para atender una máquina de fundición de troquel.
En 1966 Trallfa, una firma noruega, construyó e instaló un robot de pintura por pulverización.
En 1971 El "Standford Arm", un pequeño brazo de robot de accionamiento eléctrico, se desarrolló en la Standford University.
En 1978 Se introdujo el robot PUMA para tareas de montaje por Unimation, basándose en diseños obtenidos en un estudio de la General Motors.

En la actualidad el mundo de la robótica ha evolucionado hacia los robots móviles autónomos, estos tienen la capacidad de desenvolverse en entornos desconocidos y parcialmente cambiantes por si mismos.

En los años setenta la NASA colaboró con el Jet Propulsión Laboratory para desarrollar plataformas capaces de explorar terrenos hostiles como por ejemplo su envío a otros planetas como sucedió a posteriori con el envío del robot PathFinder a Marte (1997) [5].

Los últimos estudios en la actualidad son muy ambiciosos como por ejemplo el robot IT diseñado para expresar emociones, el COG también conocido como el robot de cuatro sentidos, el famoso SOUJOURNER [6] o el LUNAR ROVER [7], vehículo de turismo con control remoto, y otros mucho más específicos como el CYPHER [8], un helicóptero robot de uso

militar, el guardia de tráfico japonés ANZEN TARO o los robots mascotas de Sony.

La historia de la robótica en general se puede clasificar en cinco generaciones:

-Las dos primeras ya fueron alcanzadas en los años 80's, los robots estaban preparados para gestionar tareas repetitivas pero con una autonomía aún muy limitada.

-La tercera generación incluía la visión artificial que ha sido muy desarrollada en los 80's y 90's.

-La cuarta incluye el movimiento en interiores y exteriores.

-La quinta, en la que se está trabajando actualmente incluye la inteligencia artificial.

2.3 MOVILIDAD Y NAVEGACION DE ROBOTS

Reseña histórica

La introducción de la movilidad en la robótica ha supuesto un elemento muy importante para muchas tareas y aplicaciones humanas. Resulta de vital importancia si tenemos en cuenta que con la movilidad el robot puede sustituir a los humanos en muchos trabajos tediosos, fatigantes, peligrosos o irrealizables [9]. Podemos destacar como ejemplos de estos tipos de trabajos:

-Tareas repetitivas como por ejemplo el ensamblaje de piezas en la industria. En este caso el uso del robot evita que se produzca bajas laborales de los trabajadores por las enfermedades profesionales (sobreesfuerzos por trabajos repetitivos)

-Realización de soldaduras en la industria del automóvil

-Trabajos en atmósferas peligrosas

-Desactivado de explosivos y reconocimiento aéreo del terreno (en el ejército y las fuerzas de seguridad)

-En el mundo de la medicina y la cirugía

La navegación es una tarea muy importante dentro de la robótica móvil. Esta constituye un elemento esencial, para controlar directamente la función de movilidad. La navegación le permite a un robot, por ejemplo, pasar o no por un lugar preciso. Un robot deberá navegar de alguna manera dentro de un ambiente dado para ejecutar una tarea encomendada. Para el logro de esta tarea se necesita que los niveles de decisión del robot tengan a su disposición suficientes elementos del ambiente (primitivas) eficientes, provistos por subsistemas de percepción.
La complejidad de la navegación se deriva de un sin números de variantes relacionadas con las características del propio vehículo, los medios de percepción a emplear, el tipo de tarea a ejecutar y hasta el tipo de ambiente en el que va a evolucionar. Aunque todos estos factores tienen un compromiso para el desarrollo exitoso de un sistema de navegación, el ambiente es sin duda un elemento determinante. Muchos de los sistemas hasta la fecha desarrollados se basan en ambientes bien conocidos y poco variantes o que varían de una manera conocida. Una de las tareas claves de la navegación es la localización (es decir, conocimiento de su posición global y seguimiento de su posición local).
Un sistema de robótica móvil requiere para la navegación tener conocimiento de su localización y un conocimiento preciso de su ambiente.

2.3.1 ESTUDIOS REALIZADOS PARA LA PLANIFICACION Y EXPLORACION DE RUTAS EN ROBOTS MOVILES

La navegación de robots móviles requiere un compromiso entre la necesidad de obtener una ruta óptima y la necesidad de reaccionar ante obstáculos imprevistos.

Se considera una ruta óptima aquella que realiza un trazado suave y que evita los obstáculos desconocidos (dinámicos o no).

Según Latombe (1991), los métodos de planificación de rutas para robots móviles se dividen en cinco tipos:

Roadmap: Este método consiste en representar en forma de red el entorno y luego aplicar algoritmos de búsqueda gráfica para encontrar la ruta.

Exact cell Decomposition: consiste en dividir el entorno en celdas no superpuestas que son llevadas a un gráfico.

Approximate Cell Decomposition: a diferencia del método anterior las celdas tienen un tamaño predefinido y no cubren el espacio libre en su totalidad.

Potential Fields (Campos de potencial): este método [10] fue desarrollado por Khatib (1986). Se basa en la creación de un campo de potencial artificial en el cual la meta o destino es un polo de atracción y los obstáculos polos de repulsión.

El robot sigue el gradiente de este potencial hacia su mínimo.

La ventaja de este método es la simplicidad y la facilidad de tratamiento de obstáculos tanto fijos como móviles. La desventaja es la existencia de mínimos locales y de oscilaciones bajo ciertas configuraciones de obstáculos (zonas estrechas).

Navigation Functions: son consideradas un caso especial del método *Potential Fields.*

Además de los cinco tipos anteriores podemos incluir otros dos tipos más:

Vector Field Histogram (VFH): Este método [11] fue desarrollado por Borenstein & Koren (1991). Este método es una variante del método *Potential Fields* y se vale de un histograma construido alrededor del robot. En dicho histograma se representan las densidades polares de obstáculos, el área con menor densidad de obstáculos será elegida y se elige el área con menor densidad de obstáculos como dirección a la meta u objetivo.

Si bien la técnica es simple puede conducir a generar oscilaciones en el seguimiento de la ruta y puede quedar bloqueada bajo ciertas configuraciones. Posteriormente se desarrolló una versión mejorada que incluía distancia de seguridad (**VFH+)** [12] que lograba disminuir las oscilaciones y obtener una ruta suave

The Elastic Bands (Quinlan & Khatib, 1993): Este método [13] representa burbujas sujetas a fuerzas repulsivas provenientes de los obstáculos y a fuerzas atractivas provenientes de las burbujas colindantes. Este método crea rutas suaves y se adapta al movimiento o a obstáculos inesperados

La mayor parte de estos métodos generales [14] no son de aplicación si el entorno es dinámico o hay obstáculos no definidos. Por ello se corre el riesgo de trazar rutas que no se ajustan a la realidad que conllevan al inevitable choque contra obstáculos.

La evasión de obstáculos se ha basado en la clásica división entre módulo "global" que es el encargado de trazado de ruta entre un punto de origen y otro de destino pero que no tiene en cuenta los obstáculos desconocidos o imprevistos y el módulo "local" que mediante la información aportada por sensores se encarga de proceder a la evasión del obstáculo pero siempre a nivel local. La utilización de dos módulos para conseguir el seguimiento de una ruta realista como es lógico supone un mayor trabajo de computación y mayor tiempo de ejecución.

Para enfrentarse al problema se desarrollaron algoritmos de evasión de obstáculos. Estos métodos se basan en repetir el proceso de percepción-acción en un ratio muy elevado (frecuencia). Como puede verse en la figura 1, el proceso consta de dos pasos: primero se lee la información de los sensores y seguidamente se genera una orden de movimiento para evitar la colisión mientras el robot continúa su camino hacia su destino sin alterar la ruta global planificada.

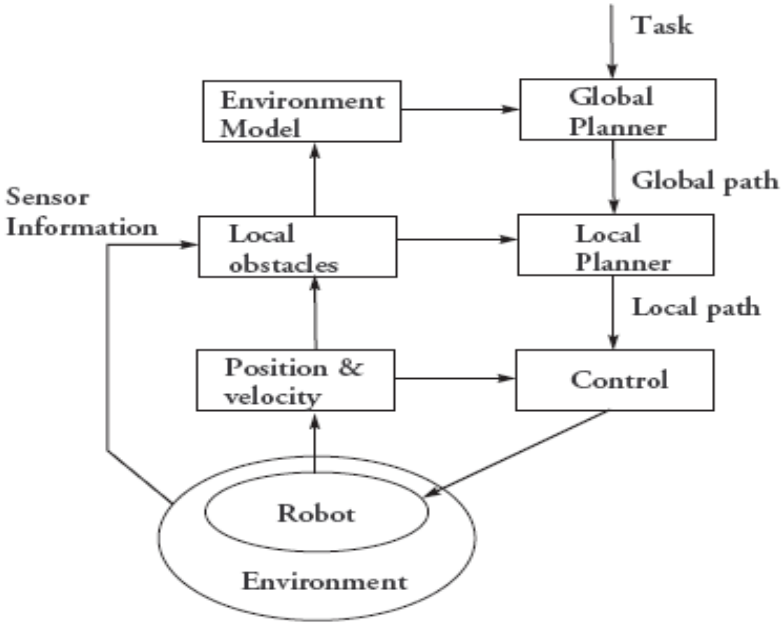


Figura 1 . Procedimiento clásico

Gracias al elevado ratio de funcionamiento de estos algoritmos de evasión de obstáculos pudieran ser una solución robusta para la evasión de obstáculos, sin embargo resulta difícil obtener buenos resultados ante situaciones trampa solamente utilizando la información proporcionada por los sensores.

La mayoría de estos métodos tienen dificultades para conmutar entre rutas locales y globales. Los algoritmos de evasión local disponen de modificación de trayectorias globales pero no ofrecen soluciones cuando la trayectoria global queda atrapada en un mínimo local.

La clásica solución a este problema es incluir un supervisor que analice la supervisión de la ruta global como puede verse en la figura 2. En el caso de que una trampa local o bloqueo de la trayectoria sea detectado, comienza la búsqueda de una nueva ruta global desde la posición global hasta el punto final o destino.

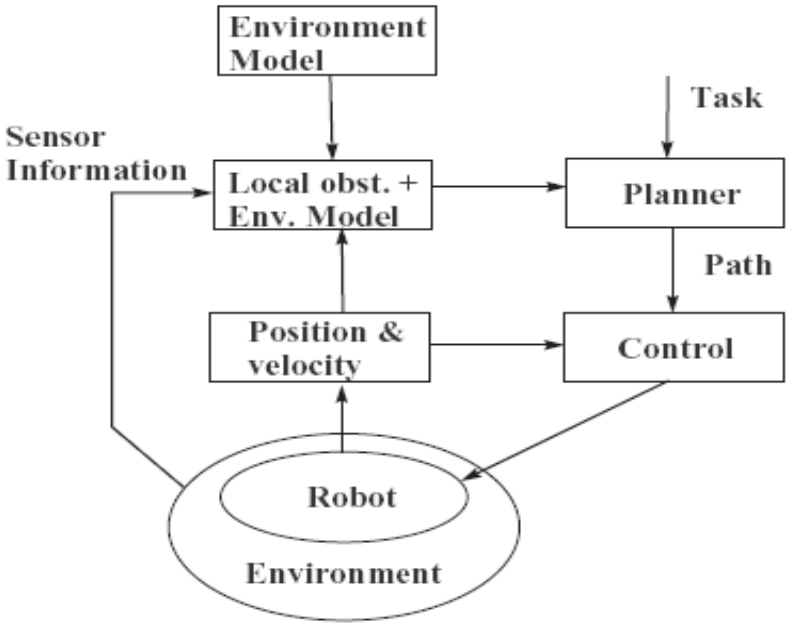


Figura 2. Procedimiento con supervisión de ruta global (fusiona en un solo módulo los módulos local y global)

2.3.2 METODO DE NAVEGACION DE ROBOTS BASADO EN METODOLOGIA FAST MARCHING

El objetivo de este método es unificar en un único planificador la planificación global y la planificación de evitación de obstáculos necesaria para la navegación de robots móviles.

Al fusionar estos dos métodos conseguimos reducir el tiempo de procesamiento que se produce en otros métodos basados en la clásica división entre “local” y “global”.

El método Fast Marching se basa en la propagación de las ondas electromagnéticas. La idea es que situando una antena en el destino que emite ondas electromagnéticas el robot pueda conducirse así mismo siguiendo las zonas de mayor potencial (gradiente) hasta llegar a la fuente de las ondas.

Las ondas electromagnéticas tienen dos cualidades muy importantes para la navegación de robot móviles: la suavidad en su propagación y la ausencia de mínimos locales.

La propagación de la onda de Fast Marching sigue el mismo principio que la refracción de la luz cuando está pasa de un medio a otro con distintas propiedades. Esto se puede explicar fácilmente con la siguiente figura:

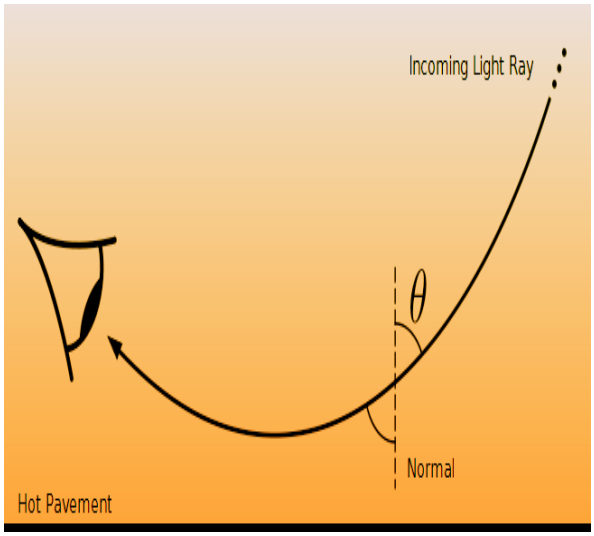


Figura 3. Inflexión de los rayos solares por el cambio de medio

Como se puede ver en la figura 3, los rayos solares no chocan directamente contra el pavimento sino que se produce una reflexión o cambio de dirección, ello es debido a que los rayos en su recorrido cruzan distintas capas de aire con distintas temperaturas. Cada capa de aire corresponde a un medio distinto.

Otro ejemplo de refracción muy conocido es la que se produce al introducir un objeto en agua:

Figura 4. Refracción de la luz en objeto introducido en agua

Como puede verse en la figura 4, cuando la luz llega a una superficie transparente, una parte de la luz se refleja y el resto se refracta al penetrar en el nuevo medio. En general la luz que penetra en el nuevo medio lo hace cambiando su dirección de propagación.

Por otra parte, si las ondas tienen que atravesar múltiples medios el efecto sería el observado en las figuras 5 y 6:

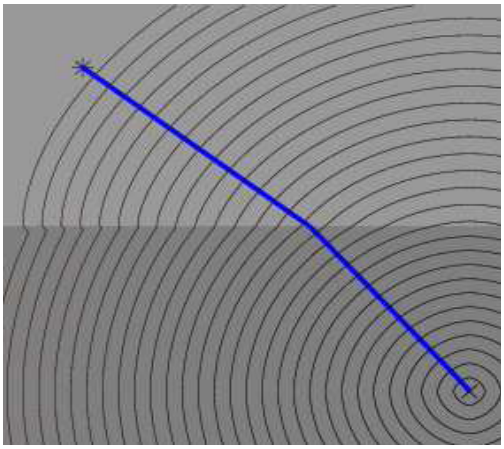


Figura 5

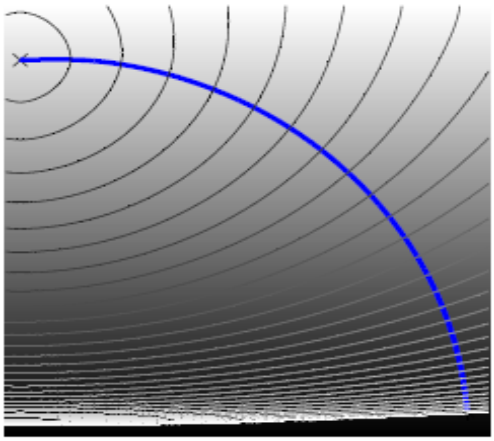


Figura 6

Figuras 5 y 6. Cambio de dirección de la luz cuando atraviesa diferentes medios.

La aplicación del método Fast Marching por si solo presenta problemas, dado que la óptima ruta de movimiento puede llevar al robot demasiado cerca de los obstáculos (esquinas, paredes, etc..) lo cual no es seguro como puede comprobarse en la figura 7:

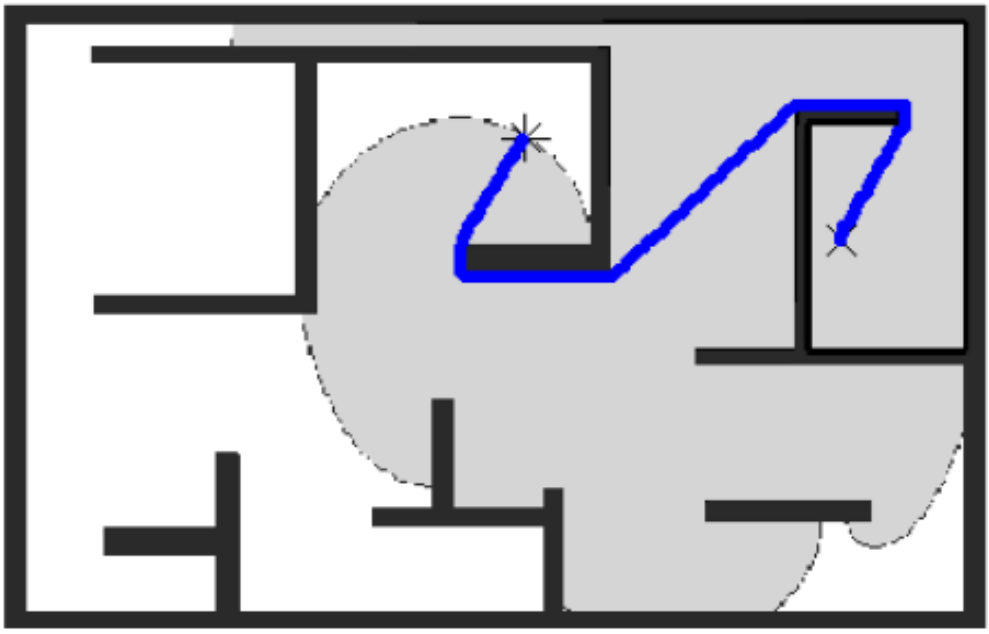


Figura 7. Trayectoria calculada directamente con Fast Marchig

En la figura anterior podemos observar en gris el frente de propagación de ondas y la ruta trazada que si bien es la mínima distancia geométrica no es segura por la proximidad a los obstáculos.

Podemos intentar resolver el problema ampliando los obstáculos pero aún así la trayectoria tiende a cerrarse contra los obstáculos y la trayectoria no resultará suficientemente suave y segura.

DIAGRAMA DE VORONOI

El diagrama de Voronoi es una construcción geométrica que permite construir una partición del plano Euclídeo. Se crea uniendo puntos entre sí y trazando las mediatrices de los segmentos de unión, las intersecciones de estas mediatrices determinan una serie de polígonos en el plano bidimensional.

El perímetro de estos polígonos es equidistante a los puntos vecinos (los situados a ambos lados del perímetro). Si realizamos el diagrama de Voronoi del entorno de movimiento del robot el resultado será:

Como puede observarse en la figura 8, los puntos pertenecientes a las líneas paralelas a los obstáculos tienen la propiedad de encontrarse a igual distancia de los obstáculos situados a ambos lados de éstas.

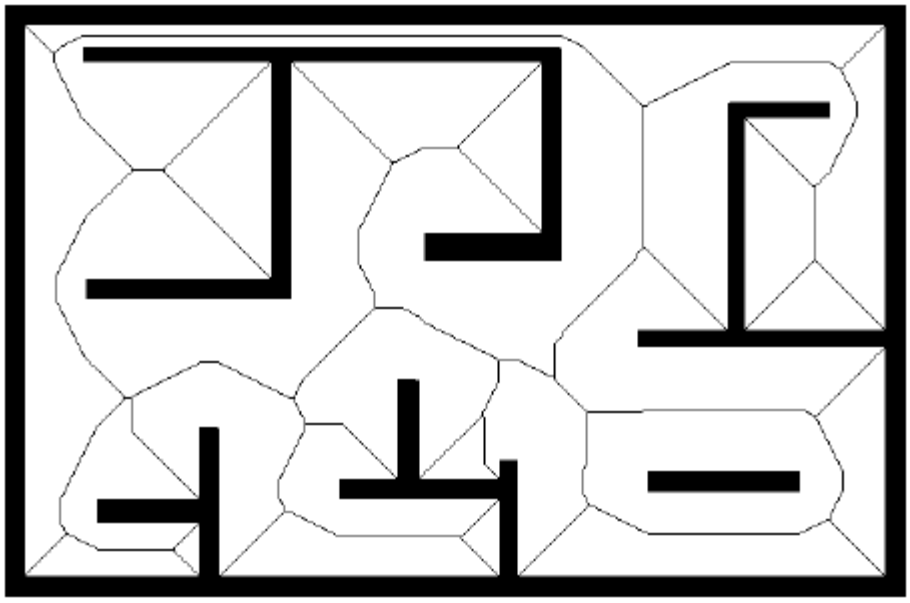


Figura 8. Mapa del entorno realizado mediante el método de Voronoi (mediante skeletonization)

TRANSFORMADA EXTENDIDA DE VORONOI

Con el fin de evitar que durante el seguimiento de las trayectorias el robot móvil se acerque demasiado a las esquinas u obstáculos, se necesita obtener un potencial similar a un potencial eléctrico repulsivo de tal forma que los obstáculos repelan al robot y permitan a éste evitarlos.

La Transformada Extendida de Voronoi se obtiene aplicando sobre el mapa la figura 7 diferentes tonos de grises. Los tonos de grises con una mayor intensidad corresponden a las zonas más próximas a los obstáculos mientras que las zonas más alejadas (las zonas equidistantes a los obstáculos) son de un gris mucho menos intenso.

A cada nivel de intensidad de gris se le asigna un potencial que varía en función de la zona del mapa.

El potencial cero que corresponde al color negro, indica que la zona o celdas que conforman la misma forman parte de un obstáculo y el máximo potencial corresponde a la zona o celdas que son equidistantes a los obstáculos:

Figura 9. Representación del potencial de la Transformada extendida de Voronoi

A esta representación del potencial también se le denomina como potencial de lentitudes, dificultad o viscosidad.

Si sobre el mapa de potenciales que puede verse en la figura 9 lanzamos la onda de Fast Marching, podremos comprobar cómo se produce una considerable mejora del cálculo de la trayectoria.

Como puede observarse en la figura 10, la trayectoria por una parte tiende a seguir la representación de Voronoi, ya que como se ha indicado en la página.24 el robot móvil se guía hasta el objetivo siguiendo las zonas de mayor potencial (gradiente) que son las más alejadas de los obstáculos (las que se encuentran equidistantes a los mismos) y por otra parte la trayectoria es considerablemente suave (esta suavidad se debe como se ha indicado anteriormente al comportamiento o propagación de las ondas electromagnéticas) según se puede ver en la representación esqueletizada o diagrama de Voronoi de la figura 8.

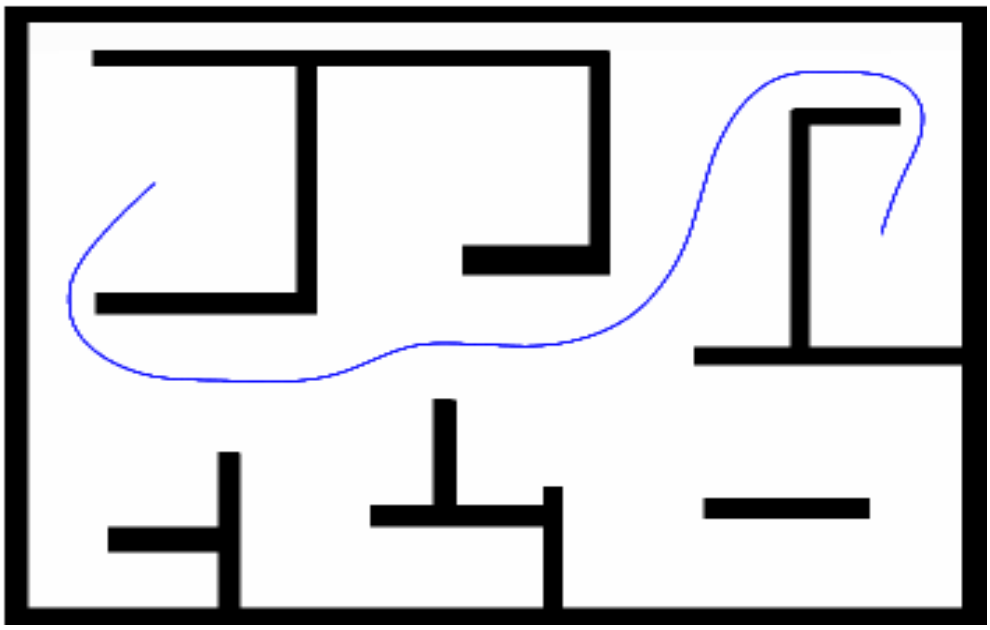


Figura 10. Trayectoria calculada mediante Fast Marching sobre la Transformada extendida de Voronoi.

2.4 LOCALIZACION Y PERCEPCION

La percepción y localización son dos tareas claves en la navegación de robots móviles. Su estrecha relación queda de manifiesto al hacer un análisis de estos dos conceptos. Se llama percepción a la disciplina que se encarga de dotar al robot con capacidades sensoriales que le permita extraer la información necesaria del mundo que lo rodea (mundo exterior). Esta información es requerida para modelar el ambiente (hacer una reconstrucción del espacio que rodea al robot) y para localizar el robot dentro de éste. La localización por su parte, permite determinar el lugar que ocupa el robot en el medio ambiente donde se desenvuelve.

De lo anterior se entiende que el robot como sistema inteligente y autónomo que pretendemos que sea, debe tener una noción del espacio que lo rodea y de la posición que ocupa dentro de dicho espacio con respecto a elementos que coexisten dentro de este espacio como por ejemplo paredes, objetos, puertas, etc..

2.4.1 MEDIOS DE PERCEPCION

El proceso de percepción se basa en el uso de sensores. El término sensor se refiere al dispositivo físico que permite extraer información del mundo real (en nuestro caso del espacio físico que rodea al robot).

Los sensores se pueden clasificar en internos y externos. Los sensores internos nos dan información propia del robot como la medición de desplazamientos, de velocidades y de fuerzas aplicadas o desarrolladas. Dentro de los más utilizados en la robótica cabe destacar los odómetros y los sensores inerciales.

Los sensores externos son los que nos dan información sobre el mundo real que rodea al robot, pueden ser sensores 2D o 3D. Los sensores 2D realizan una proyección tridimensional sobre un plano o 3D que guarda las tres dimensiones de un punto en el espacio.

También podemos clasificar los sensores según la energía utilizada. Se distinguen dos tipos: sensores activos y sensores pasivos.

Los sensores activos emiten energía al medio que rodea al robot como por ejemplo el sensor de ultrasonidos o los telémetros láser.

Por otra parte los sensores pasivos como las cámaras de vídeo que se valen de la luz natural.

A continuación se hace una breve descripción de los sensores más utilizados en el mundo de la robótica:

Las cámaras de vídeo: pertenecen al grupo de sensores externos 3D, existen dos tipos: las tradicionales basadas en los tubos de vacío y las modernas de estado sólido muy compactas y ligeras que son las habitualmente utilizadas en el campo de la robótica.

Las cámaras de vídeo son atractivas desde el punto de vista económico pero caras desde el punto de vista del tratamiento de la información que conlleva el trabajo con imágenes, pues esto puede ser incompatible para algunos sistemas en lo referente al tiempo de respuesta deseado.

Un tiempo de respuesta no acorde con lo esperado puede resultar muy perjudicial para el movimiento de un robot pues el sistema de procesamiento de la imagen debe adaptarse a la velocidad de movimiento del robot móvil.

Las cámaras de vídeo también son muy dependientes del ambiente donde se desenvuelva el robot, por ejemplo ante situaciones ambientales adversas como por ejemplo niebla, exceso de luz solar, etc.. ,será difícil la recopilación de información y en ocasiones imposible de tratar.

Los telémetros: pertenecen al grupo de sensores externos tipo 2D que son muy populares en el mundo de la robótica. El fin de un telémetro es medir la distancia existente entre el emisor y el punto de impacto en el espacio.

Los telémetros pueden usar diferentes tipos de ondas para la propagación de éstas en el espacio: los radares utilizan ondas electromagnéticas, los ultrasonidos utilizan ondas sonoras y los telémetros láser basados en la emisión de luz.

Ultrasonidos: son de gran utilidad para la navegación de robots en interiores y en exteriores. Son muy populares debido a su bajo coste y en su capacidad para abarcar grandes áreas.

Están construidos mediante una pastilla que realiza a la vez las funciones de emisor y receptor como puede apreciarse en la figura 11:

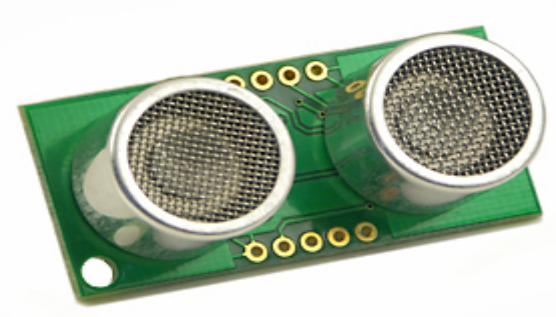


Figura 11. Hardware de sensor ultrasonidos

La medición se considera en la dirección al eje central del cono de la propagación de la onda sonora, dicho cono resulta perpendicular a la superficie del sensor. La precisión de la medida del sensor va en función de la amplitud del cono o del ángulo formado por este: cuando más pequeño sea el ángulo de propagación de la onda, mejor será la resolución angular y más exacta será la medida. Por otra parte si el ángulo del cono es grande, el sensor cubre más espacio y la probabilidad de detectar un obstáculo es más grande, pero en este caso, la resolución angular será débil y la precisión sobre la posición del objeto detectado será inexacta.

Otro problema inherente a los ultrasonidos es la reflexión. Cuando una onda sonora incide en la superficie de un objeto, esta será reflejada según la textura de la superficie de impacto y según la longitud de la onda emitida. Cuando una onda incidente encuentra una superficie pulida o lisa como un espejo, esta se refleja de una forma bien definida, decimos entonces que la reflexión es especular (figura 12 a).

Por lo contrario, si la superficie es irregular cada pequeña porción saliente proyectará la onda en direcciones diferentes provocando una reflexión difusa o de Lambert (figura 12 b).
Debido a la baja resolución angular (divergencia) y a su limitado alcance, los ultrasonidos son considerados como sensores de proximidad dentro de la robótica móvil. La navegación más explotada con este tipo de sensores es el seguimiento de pared.

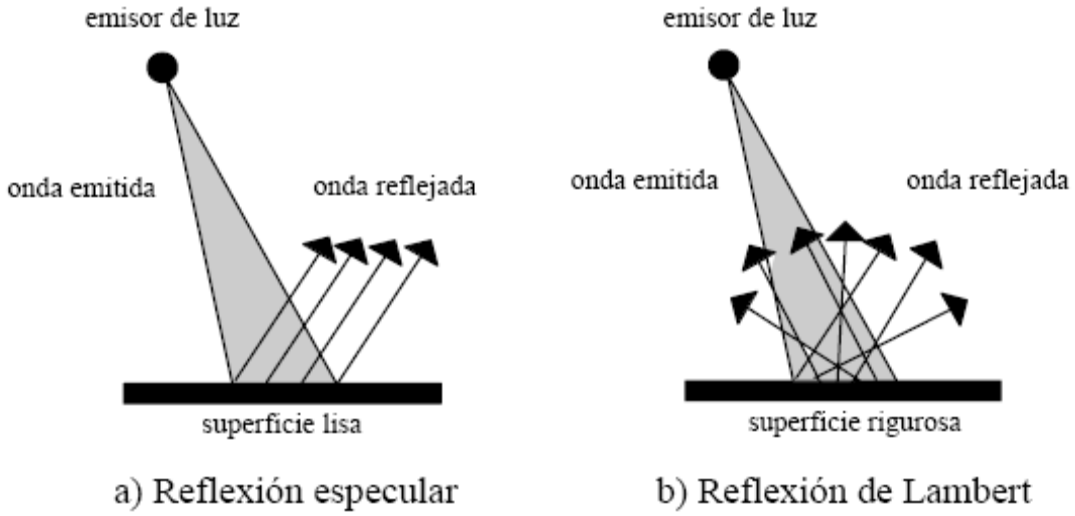


Figura 12. Reflexión de una onda

Telémetros láser: Los sistemas basados en una fuente láser, son sensores físicos activos, que miden las distancias del sensor a los objetos que integran la escena de manera directa. Estos sensores pueden ser considerados semi-activos si se apoyan en otro tipo de sensores, generalmente pasivos como son las cámaras, para completar su mecanismo de adquisición de datos.
Después de los ultrasonidos y las cámaras, estos sistemas láser constituyen el medio de percepción más usado en la navegación, sobre todo para tareas en ambientes exteriores.

El principio de funcionamiento de un sensor láser o telémetro láser, se basa en el análisis de la trayectoria de una señal luminosa emitida por una fuente láser, la cual es reflejada por algún objeto de la escena observado. Una vez que la señal es captada por el receptor, es tratada para obtener la distancia hacia el punto de rebote.

Las ventajas de estos sensores se derivan de las propiedades naturales de las fuentes láser. Rayos regularmente finos pueden ser obtenidos con pequeñas lentes de emisión, lo que es posible gracias a las pequeñas longitudes de onda alcanzadas por un rayo láser, pudiendo lograr una buena resolución angular. Además, las longitudes de onda corta nos aseguran un comportamiento reflexivo difuso lo que nos permite obtener con mayor probabilidad, una proyección del rayo a partir de cualquier ángulo de incidencia.

2.4.2 LOCALIZACION O DETERMINACION DE LA POSICION EN ROBOTICA

La localización surge ante la necesidad de conocer su posición y orientación con respecto a un sistema de referencia absoluto (en nuestro caso el mapa por donde va a moverse el robot) con el fin de poder desplazarse por el entorno. Esta información resulta de vital importancia para realizar trayectorias, evitar obstáculos, etc..

Para determinar la localización del robot es necesario encontrar las componentes de translación (tx, ty, tz) y de rotación (θx, θy, θz) con respecto a un sistema absoluto.

Como podemos ver en la figura 13, en un sistema de dos dimensiones como es por ejemplo un plano 2D el problema se reduce a encontrar la terna (tx, ty, θ), (tx, ty) corresponden a la posición y θ a la orientación del robot con respecto al eje X.

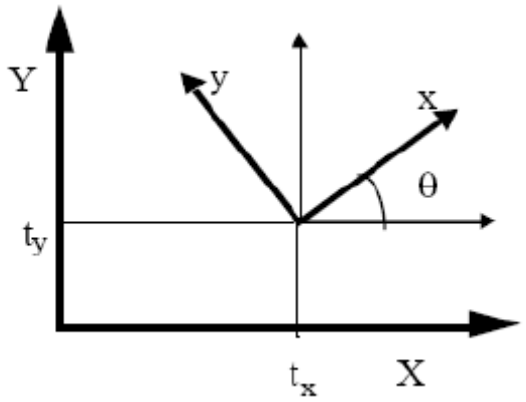


Figura 13. Sistema de referencia móvil asociado al robot

Los métodos para determinar la posición del robot pueden clasificarse en dos grupos principales como muestra la figura 14:

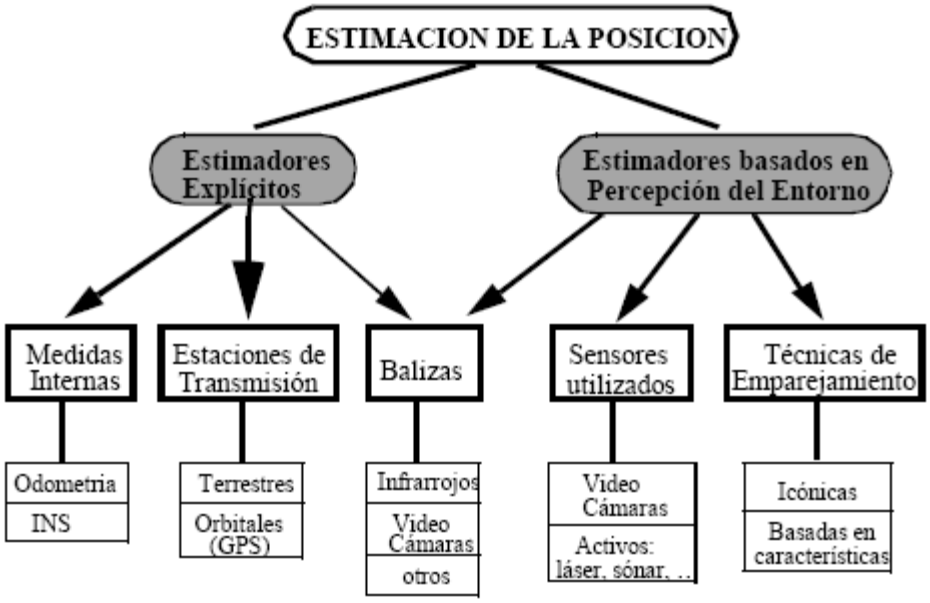


Figura 14. Sistemas de estimación de la posición de un robot

-Sistemas de estimación explícitos: Dentro de esta categoría se engloban todos aquellos que son capaces de estimar la posición del robot sin necesidad de conocer el entorno que lo rodea (interpretación del entorno).

Dentro de estos sistemas pueden distinguirse claramente dos grupos:

- Estimación basada en medidas internas
- Estimación basada en estaciones de transmisión

Los primeros utilizan exclusivamente sensores integrados en el propio robot móvil como codificadores, giróscopos, buscadores de norte, acelerómeros, tacómetros, etc., y por supuesto sin ningún tipo de información del exterior.

Los segundos se basan en dos unidades bien diferenciadas, la unidad montada en el propio vehículo y la unidad/es que son instaladas en puntos concretos del entorno. Normalmente la unidad montada en el vehículo actúa como receptor y las situadas en el entorno como emisores

Estimación basada en medidas internas

Resulta ser la manera más simple de determinar la posición y orientación de un robot móvil. Se basa en integrar la trayectoria recorrida por éste a partir de una serie de medidas internas como son: giro de las ruedas, velocidades , aceleraciones, cambios de orientación, etc..

Dependiendo de la información que se utilice se distinguen dos grupos fundamentales: **sistemas odométricos** y **sistemas de navegación inercial**.

El primero integra la velocidad del robot respecto el tiempo (e=v.t), (el espacio es la variación de la velocidad en el tiempo) para obtener el espacio recorrido mediante el uso de Encoders [15] y el segundo utiliza las aceleraciones del robot en su trayectoria mediante Acelerómetros [16] y Giroscopios [17].

2.5 ODOMETRIA

La Odometría es un método muy antiguo que fue utilizado inicialmente en la navegación marítima, donde era el método principal para estimar la posición sobre los mapas. En la actualidad es el método más simple de posicionamiento de un robot móvil. Los robots móviles utilizan la Odometría para estimar (no determinar) su posición relativa con respecto a su posición inicial. Un sistema odométrico se basa en la información aportada por los sensores internos del vehículo sin apoyarse para nada en sensores externos que pudieran estar localizados en el entorno.

Para llevar la cuenta del número de vueltas dadas por las ruedas (o bien fracciones de éstas) del robot móvil se utilizan codificadores ópticos o también más comúnmente llamados Encoders.
Para la estimación de la posición se requiere el registro odométrico de las ruedas motrices del robot.

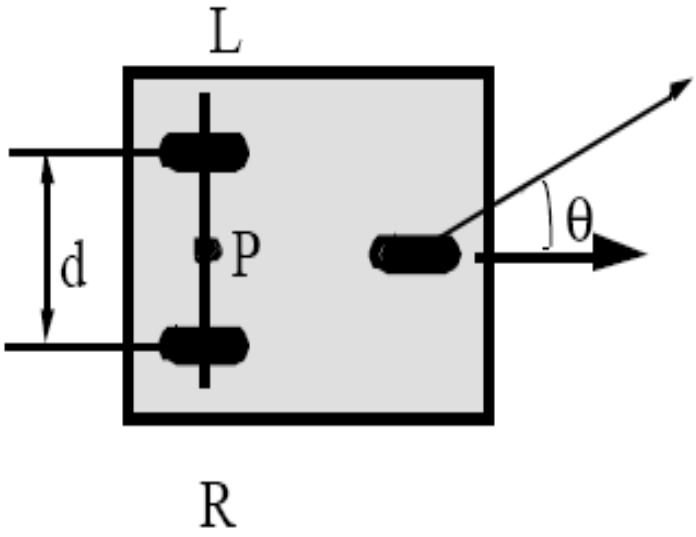


Figura 15. Dibujo esquemático de un vehículo triciclo.

Para ilustrar el funcionamiento de esta técnica consideramos el vehículo de la figura anterior (figura 15).

Como punto de referencia, se ha escogido "P" situado entre ambas ruedas. La distancia recorrida por el punto "P" en un intervalo de tiempo Δt viene dada por la expresión:

$$\Delta s = \frac{1}{2}(\Delta s_R + \Delta s_L) \qquad (1)$$

Donde sR y sL son las distancias recorridas por la rueda derecha e izquierda, respectivamente. La variación en la orientación del vehículo Δθ, viene dada por:

$$\Delta\theta = \frac{1}{d}(\Delta s_R - \Delta s_L) \qquad (2)$$

Donde "d" es la distancia entre ambas ruedas.

Mediante la integración de las ecuaciones (1) y (2) se obtiene, respectivamente:

$$s(t) = \frac{1}{2}(s_R(t) + s_L(t)) \qquad (3)$$

$$\theta(t) = \theta(0) + \frac{1}{d}(s_R(t) - s_L(t)) \qquad (4)$$

Donde s(t) es la distancia recorrida por el punto P y θ(t) es la orientación del vehículo, ambas para un tiempo dado t.

Las coordenadas x e y del punto P pueden obtenerse mediante las ecuaciones:

$$x(t) = x(0) + \int_0^t \frac{1}{2}(\dot{s}_R(t) + \dot{s}_L(t))\cos(\theta(t))dt \qquad (5)$$

$$y(t) = y(0) + \int_0^t \frac{1}{2}(\dot{s}_R(t) + \dot{s}_L(t))\,\mathrm{sen}(\theta(t))dt \qquad (6)$$

Si bien la gran ventaja de la Odometría reside en su simplicidad y bajo coste, los sistemas basados en la Odometría tienen tendencia a la acumulación de errores debidos en gran medida a los giros o cambios de orientación, causando grandes errores en la estimación de la posición. Así mismo estos errores se van acumulando proporcionalmente a la distancia recorrida por el robot.

A pesar de las limitaciones que comporta el uso de la Odometría, muchos investigadores están de acuerdo en que la Odometría forma parte muy importante del sistema de navegación de un robot pero que debe de ser complementada mediante un sistema de medidas de posicionamiento absolutas para proporcionar una estimación de la posición más precisa.

La Odometría se basa en ecuaciones simples que se pueden implementar fácilmente y que utiliza los datos de los Encoders situados en las ruedas del robot. También está basada en la suposición de que las revoluciones (giros de la rueda) pueden ser traducidas a desplazamiento lineal de las ruedas con respecto al suelo (el trayecto recorrido tras un giro completo de una rueda o 360º será función de la expresión L=(2*Π*r) ó longitud de la circunferencia, siendo "r" el radio de la rueda del robot).

En un caso extremo donde se implicará un error en la estimación de la posición será por ejemplo cuando una rueda patina debido a una mancha de aceite y la otra no, entonces dicha rueda registrará revoluciones si bien no se corresponderán a desplazamiento lineal.
Además de este ejemplo hay muchas más razones por la que se pueden introducir imprecisiones en la traducción de las lecturas del Encoder a desplazamiento lineal.

Todos estos errores se pueden agrupar en dos categorías, errores sistemáticos y no sistemáticos:

Errores sistemáticos

-diámetros de ruedas no iguales
-La media de los diámetros de las ruedas difieren del diámetro de fábrica de las ruedas
-Alineado incorrecto de las ruedas

Errores no sistemáticos

-Desplazamiento por suelos no nivelados
-Desplazamiento sobre objetos inesperados sobre el suelo
-Deslizamiento de ruedas debido a:

* Suelos resbaladizos
* Sobre-aceleración
* Derrapes debidos a la pérdida de adherencia de ruedas al iniciar el movimiento.
* Fuerzas externas (interacción con cuerpos externos)

A la hora de reducir errores en la Odometría es importante la distinción entre errores sistemáticos y no sistemáticos.
Los errores sistemáticos son especialmente graves por que se van acumulando continuamente. En interiores donde las superficies no son rugosas los errores sistemáticos son mucho más importantes que los no sistemáticos, en cambio en exteriores sobre superficies rugosas o con mayor adherencia los que predominarán serán los no sistemáticos.
El mayor problema de los errores no sistemáticos es que aparecen inesperadamente como por ejemplo cuando el robot pasa por encima de un objeto situado en el suelo y ello provoca errores muy importantes a la hora de estimar la posición.

-Sistemas basados en el entorno: estos sistemas son una alternativa a los sistemas explícitos, se basan en dotar al robot móvil de un sistema de sensores que le proporcionen a éste suficiente información del entorno para poder determinar su localización. El sistema sensorial puede estar formado por diferentes tipos de sensores (cámaras, láser, ultrasonidos, Escaner,etc..).

En todo caso la localización se realiza mediante el emparejamiento de los datos extraídos del entorno mediante el sistema sensorial del robot y los datos previamente conocidos del entorno (por ejemplo de un mapa) por donde se moverá el robot.

3. DESARROLLO

3.1 CONSTRUCCION DE PROTOTIPO DE ROBOT MOVIL MEDIANTE KIT DE LEGO NXT MINDSTORM

Para la construcción del robot móvil utilizado en el desarrollo del proyecto se ha utilizado el kit para construcción de robots NXT Mindstorm de Lego.

Dicho kit se compone de los siguientes elementos:

- Diferentes tipos de piezas (varios tamaños y modelos) para poder construir el robot móvil
- Diferentes tipos de sensores para poder extraer información del mundo exterior para su procesamiento
- Cables tipo Ethernet RJ12 para la conexión de sensores y servomotores al ladrillo inteligente NXT
- Cable tipo USB para la descarga de programas desde el ordenador(también se puede realizar mediante dispositivo Bluetooth)
- Servomotores que harán posible el desplazamiento del robot móvil
- Ladrillo Inteligente NXT con un microprocesador de 32-bit.

Tipos de sensores

El kit de construcción viene con cuatro tipos de sensores:

-Sensor de contacto o presión (touch sensor): este sensor da al robot el sentido del tacto. Detecta cuando ha sido presionado y cuando deja de ser presionado volviendo a su posición inicial

Figura 16. Sensor de contacto

-Sensor de sonido: dota al robot del sentido del oído. Este sensor se puede ajustar a la sensibilidad del oído humano y hace que el robot reaccione ante sonidos

Figura 17. Sensor de sonido (sound Sensor)

-Sensor óptico: junto con el sensor de ultrasonidos, da al robot la capacidad de poder ver. Permite distinguir entre la claridad y la oscuridad, puede medir la intensidad de luz de una habitación y medir la intensidad de luz de superficies coloreadas.

Figura 18. Sensor de óptico (Light sensor)

Si bien el kit de Lego NXT Mindstorm viene con cuatro tipos distintos de sensores para la construcción del robot móvil únicamente se ha empleado el sensor de ultrasonidos que podemos ver en la figura 19 para la medición de las distancias a los obstáculos.

-Sensor de ultrasonidos: se ha optado por este sensor para la medición de las distancia a objetos que conviven con el robot dentro del entorno por donde va a desplazarse el robot (obstáculos) debido a su reducido coste en comparación con otros sensores más precisos como los de tipo láser y debido a que las distancias a medir no van a ser muy grandes (para medir grandes distancias habrían de utilizarse los tipo láser pues de lo contrario el margen de error sería grande).

El sensor de ultrasonidos de Lego habilita al robot para ver y detectar objetos, también puede ser utilizado para evitar obstáculos.

Este sensor está preparado para medir distancias en cm y en pulgadas (inches). Viene preparado para medir distancias de 0 a 255 cm con una precisión de +/-3cm.

Las mejores mediciones se obtienen sobre superficies pulidas en lugar de superficies rugosas por el ya mencionado efecto especular.

Figura 19. Sensor de ultrasonidos (ultrasonic sensor)

-Servomotores: El kit de Lego Mindstorm viene con tres servomotores como el mostrado en la figura 20. Se han empleado dos de ellos para generar la tracción del robot mediante la conexión a cada uno de ellos de una rueda (dentro del kit vienen un total de cuatro ruedas) y el tercer servomotor ha sido usado para conectarle el sensor de ultrasonidos mostrado en la figura 19 y construir con ello el sistema de percepción del mundo externo para que el robot tenga una idea lo más precisa posible del entorno que le rodea.

Los servomotores que se encargan de la tracción del robot móvil han sido gobernados básicamente de dos formas distintas:

a) Modo de funcionamiento asíncrono

b) Modo de funcionamiento síncrono

El modo de funcionamiento asíncrono: se ha utilizado para realizar los giros de orientación del robot. Debido a que el robot tiene la fuerza motriz en las ruedas delanteras (no directrices), se ha optado por generar los giros mediante un sistema muy simple consistente en hacer que la rueda de un servomotor gire en sentido opuesto a la del otro servomotor. También se podría haber generado el giro imitando el movimiento de un carro de combate consistente en dejar un eje girando normalmente (en ese caso sería una cadena en lugar de rueda) y bloquear el otro eje.

En este proyecto hemos elegido el primer método por la rapidez con la que se consigue efectuar el giro.

Modo de funcionamiento síncrono: se utiliza para una vez es efectuado el giro continuar en línea recta, este aspecto es de suma importancia pues de lo contrario en una trayectoria amplia se corre el riesgo de acabar chocando contra los obstáculos existentes.

En el movimiento síncrono se activan los dos servomotores encargados de la tracción simultáneamente y por ello permite describir movimientos rectilíneos.

De haber utilizado el movimiento de motores asíncrono tendríamos problemas a la hora de describir un movimiento rectilíneo (en línea recta) pues siempre hay una pequeña diferencia a la hora de comenzar el movimiento entre ambas ruedas, y esta pequeña diferencia puede suponer un error grande cuando se pretende describir una línea recta de una determinada longitud.

Esto es debido a que en el movimiento asíncrono primeramente se manda la orden para que comience a girar una rueda y a continuación para que comience la otra, pues bien ese pequeño retraso (puede ser del orden de milisegundos) entre la ejecución de estas dos órdenes de accionamiento puede provocar el efecto indicado en el párrafo anterior.

Como puede apreciarse en la figura 20, cada servomotor va dotado de un sensor de rotación que permite que el robot móvil realice movimientos precisos. Este sensor mide rotaciones parciales en grados o bien rotaciones completas (vueltas) y tiene una precisión de +/- 1º .

Una rotación es equivalente a un giro de 360º.

A continuación se muestra gráfico donde puede verse el esquema interno de un servomotor:

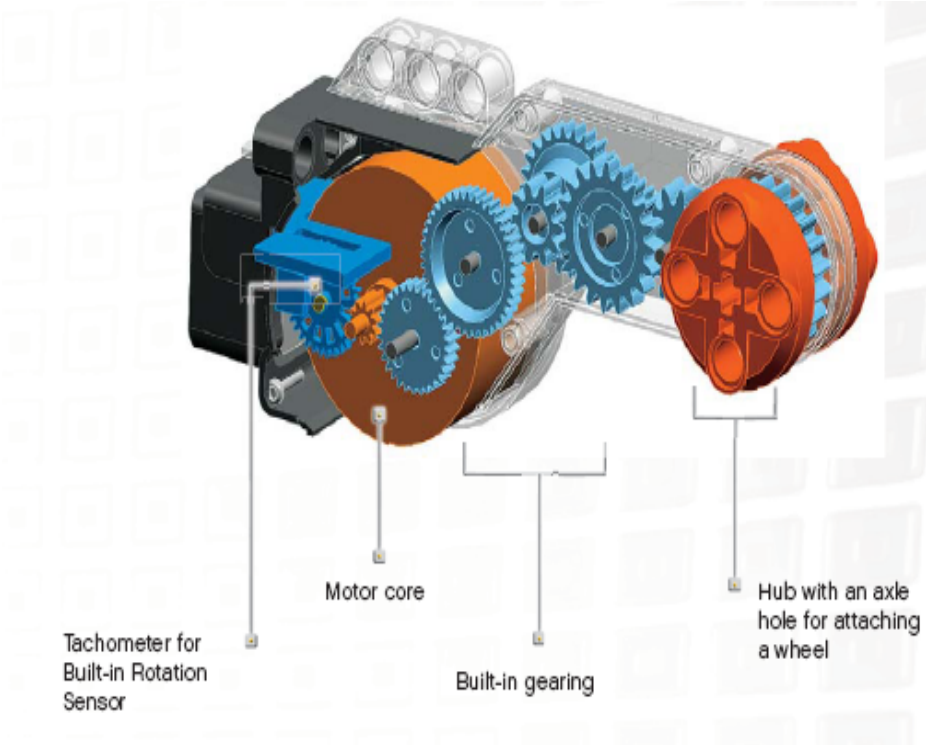


Figura 20. Esquema interno de servomotor

-Cables tipo Ethernet RJ12

En la figura 21 se muestra este tipo de cable, éstos vienen incluidos dentro del Kit Lego NXT Mindstorm y permiten la conexión de los servomotores y el sensor de ultrasonidos al ladrillo del NXT

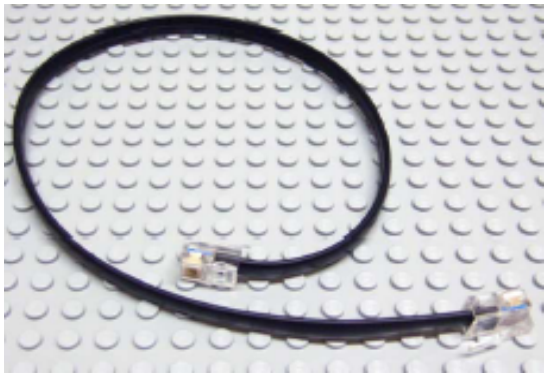


Figura 21. Cable de conexión tipo RJ12

- Ladrillo Inteligente NXT con las siguientes características:

* Microprocesador de 32 bit-ARM7

* 256 Kbytes de memoria Flash, 64 Kbytes RAM

* Microprocesador de 8 bits

* Posibilidad de comunicación con PC mediante dispositivo Bluetooth o cable USB.

* Cuatro puertos de entrada y tres de salida

* Display de 60x100 pixels

* Altavoz de 8Khz

Dispone de cuatro puertos de entrada numerados como 1, 2, 3, y 4. Estos puertos de entrada sirven para la conexión de sensores, en nuestro caso como se ha indicado anteriormente únicamente se ha utilizado el sensor de ultrasonidos.

Tres puertos de salida (A, B y C) que son los encargados de gobernar los servomotores.

Dispone así mismo de una conexión para cable de tipo USB para la descarga de programas desde el ordenador como se muestra en la figura 22 o bien según vemos en la figura 23 podemos realizar la descarga mediante un dispositivo Bluetooth.

Como podemos ver en la figura 24, para la alimentación del ladrillo se utilizan 6 pilas de 1,5V del tipo AA/LR6

Figura 22. Conexión con PC mediante cable USB

Figura 23. Conexión con PC mediante dispositivo Bluetooth

Figura 24. Alimentación mediante pilas de tipo AA/LR6 de 1,5V

-Seguidamente se muestra desde distintos ángulos de visión el montaje final del robot móvil:

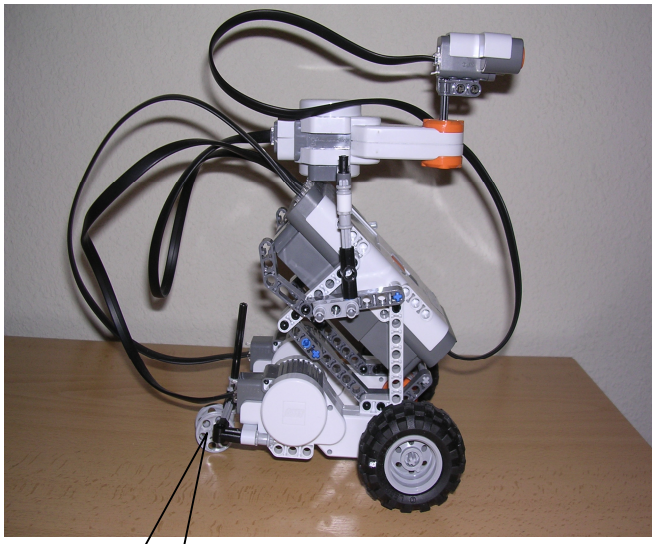


Figura 25. Vista lateral 1 robot

Eje trasero o "rueda loca"

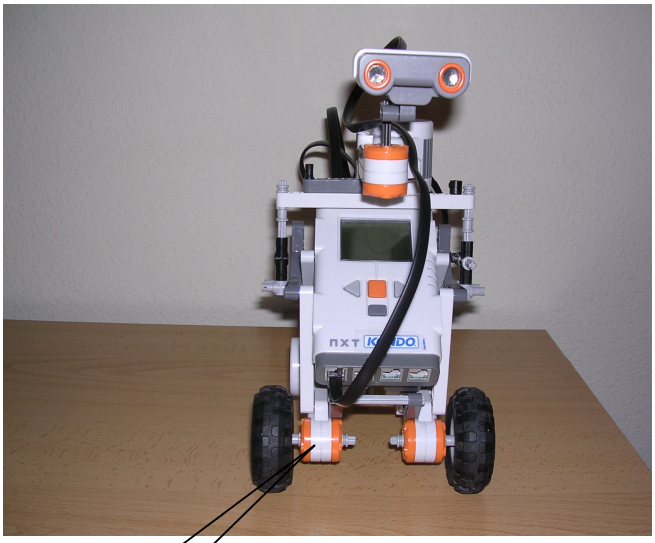


Figura 26. Vista frontal robot

Eje delantero (motriz)

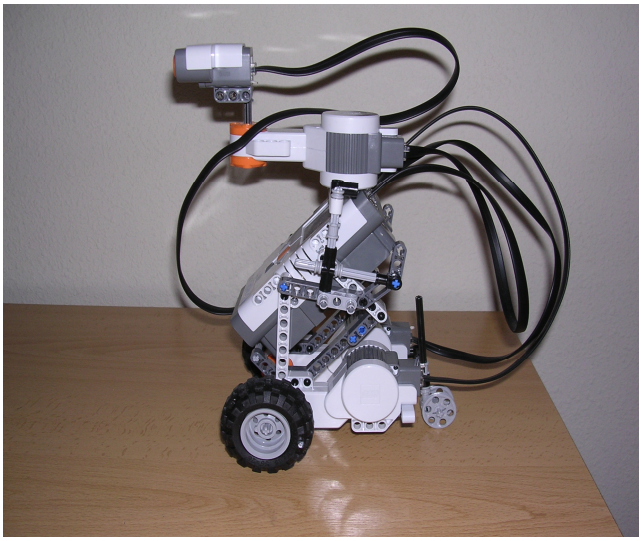


Figura 27. Vista lateral 2 robot

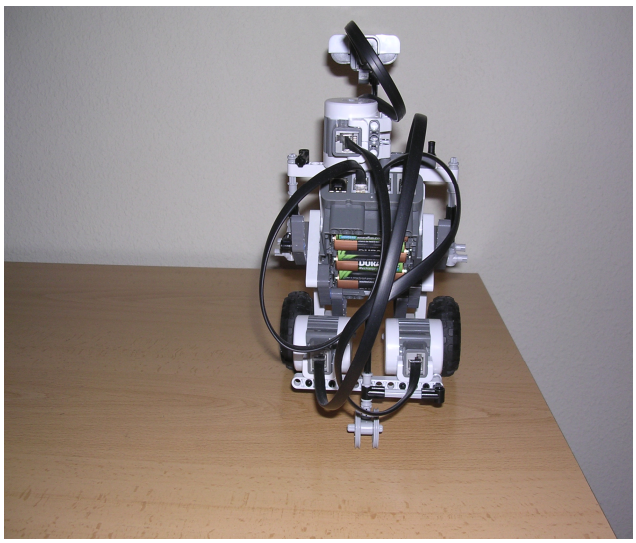


Figura 28. Vista trasera robot

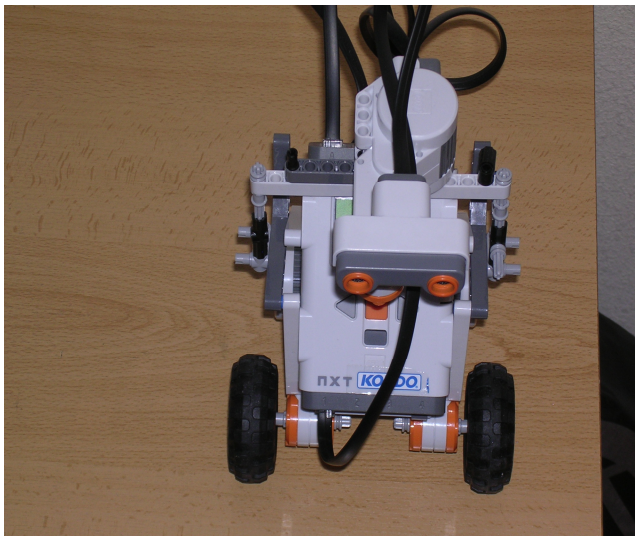


Figura 29. Vista de planta robot

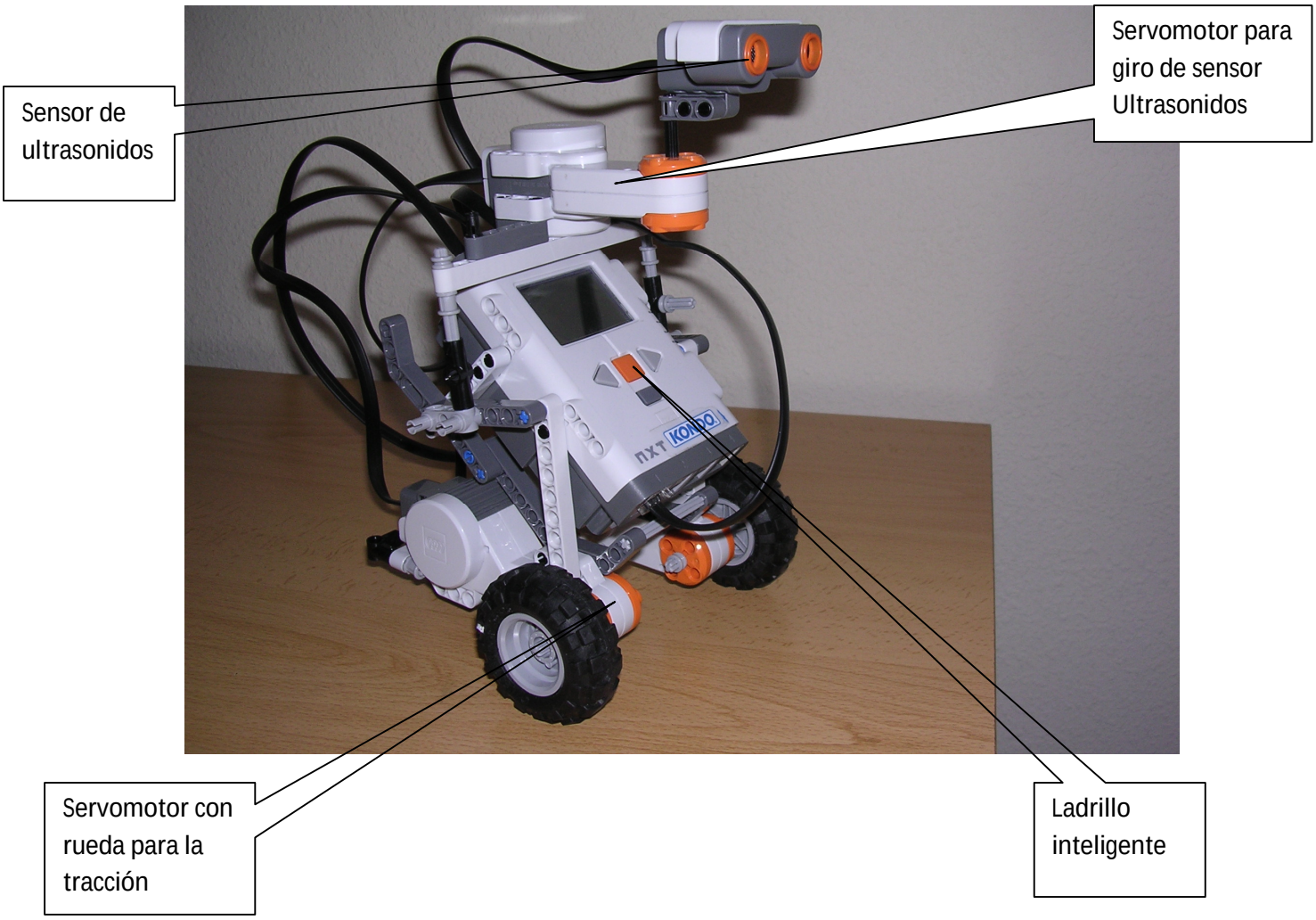


Figura 30. Montaje final robot móvil

3. 2 COMUNICACIÓN CON PC

Como se ha indicado anteriormente se ha optado por la comunicación vía Bluetooth para la transmisión de información entre el PC y el robot móvil.

Se ha elegido este dispositivo debido a la facilidad de utilización pues no hay que utilizar ningún cable físico para la transmisión.

Se ha utilizado el dispositivo Bluetooth USB 200M NANO CONCEPTRONIC , que se conecta al PC mediante conexión USB y tiene las siguientes características:

- Conexión USB 2.0
- Tipo Bluetooth 2.0 alta velocidad
- Velocidad de transferencia: 3 Mbps
- Alcance máximo: 200 metros
- Diseño ultracompacto

Figura 31. Dispositivo Bluetooth USB 200M NANO

Se ha optado por este dispositivo por su reducido coste, su diseño de muy reducidas dimensiones como puede apreciarse en la figura 31 y sobre todo por estar contrastado su correcto funcionamiento con otros proyectos anteriores donde se gobernaba el Lego NXT Mindstorm mediante este dispositivo.

Para la instalación de este dispositivo utilizamos el software con el cual venía acompañado.

Una vez instalado el software para el dispositivo, lo primero que hubo de hacerse es iniciar la búsqueda de dispositivos con características Bluetooth. Una vez visualizado el robot Lego NXT el siguiente paso consistió en emparejarlo con el dispositivo Bluetooth del PC.

Una vez realizado el emparejamiento entre ambos dispositivos nos aparecerá el siguiente icono:

Figura 32. Emparejamiento entre dispositivos Bluetooth

El siguiente paso es la conexión del NXT a un puerto virtual para que a través de éste pueda ser transmitida la información al robot (en nuestro caso el puerto virtual utilizado ha sido el COM11). Para ello nos situamos sobre el icono mostrado en la figura 32 y con el botón derecho del ratón seleccionamos “conectar Puerto Serie Bluetooth (COM11)”.

Una vez ejecutada esta orden y si el ladrillo inteligente del NXT se encuentra encendido nos aparece el icono mostrado en la figura 33 de conexión Bluetooth pero en color verde indicando que es posible la transmisión de información entre ambos dispositivos.

Figura 33. Icono para conexión del ladrillo NXT al puerto serie COM11

Tan solo restaría mediante la aplicación informática utilizada proceder mediante comandos a abrir una sesión de conexión por Bluetooth y de este modo ya está todo listo para comenzar a enviar programas al robot móvil desde el PC.

3.3 FIRMWARE Y APLICACIÓN INFORMATICA PARA EL DESARROLLO DEL SOFTWARE DE CONTROL

Acerca del firmware

El firmware es un programa que está altamente integrado en un dispositivo hardware y forma parte de la lógica de más bajo nivel que controla la electrónica. Normalmente suele grabarse en memorias de tipo ROM, FLASH , EEPROM o similar. El hecho de estar tan integrado en la electrónica de un dispositivo es lo que le hace diferente del software convencional y no tenga uso fuera del hardware para el que ha sido creado.

Básicamente el firmware es el intermediario entre las órdenes externas que recibe el dispositivo y su electrónica (hardware).

En nuestro caso el firmware será el interlocutor entre las instrucciones que mandamos al robot y la electrónica como pueden ser los motores, sin el firmware esto no sería posible y las órdenes no se llevarían a cabo.

Por encima del firmware, tendríamos el Kernel ,sistema operativo y aplicaciones. Normalmente al ser de bajo nivel esta realizado en sistema binario o bien en hexadecimal.

Firmware utilizado para el proyecto

En nuestro caso se ha utilizado la versión de firmware LEGO Mindstorm NXT firmware v1.28. Para su instalación en el ladrillo inteligente se han seguido los siguientes pasos:

-Instalación en PC de aplicación informática LEGO NXT Mindstorm Software v1.1 que viene en el CD que acompaña al Kit de montaje. Tras la instalación de este software nos aparecerá el siguiente acceso directo en la pantalla del escritorio:

A continuación conectamos el cable USB suministrado en el kit de montaje como se muestra en la figura 22.

Seguidamente entramos en la aplicación y en la pantalla principal que nos aparece seleccionamos tools->update NXT firmware con lo que se despliega la siguiente pantalla:

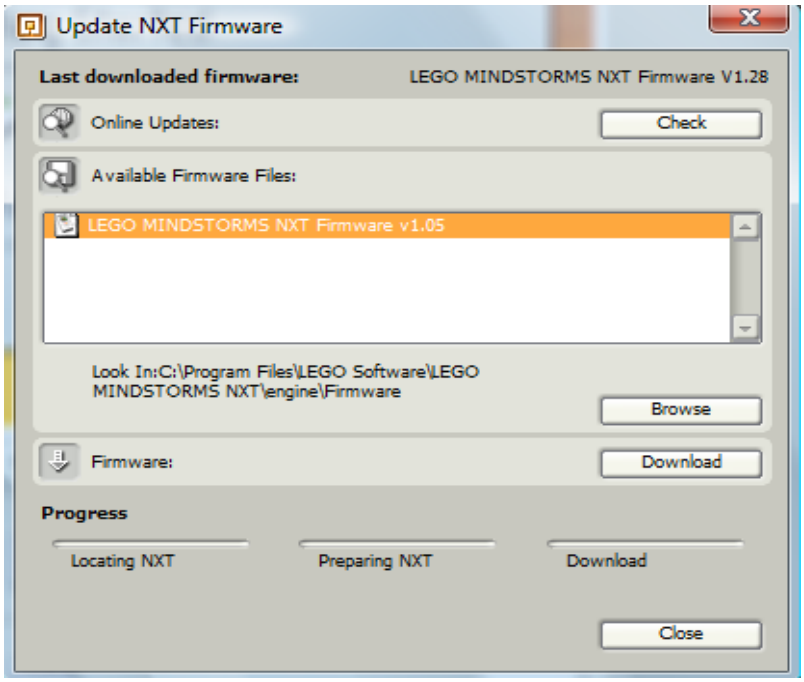


Figura 34. Proceso para descarga de firmware a ladrillo NXT inteligente

Según se muestra en la figura 34, nos aparece la versión de firmware instalada en el robot móvil y disponemos de la opción de elegir una versión superior como ha sido nuestro caso. Para ello simplemente con "Browse" le indicamos la ruta donde se encuentra la nueva versión que previamente hemos descargado desde internet.

Una vez seleccionada la versión que deseamos utilizar en el robot damos a "Download" para volcarla en el ladrillo inteligente. Ahora el robot ya está preparado para recibir programas y ejecutarlos.

APLICACIÓN INFORMATICA PARA EL DESARROLLO DEL SOFTWARE DE CONTROL (MATLAB)

Para el proyecto se ha elegido la aplicación informática "Matlab" en su versión R2007b. A continuación se hace una introducción histórica acerca del programa Matlab:

-Matlab en su estricto significado es "Matrix laboratory", para mayor claridad podemos decir que se trata de un software de alto nivel para realizar cálculos numéricos y obtener resultados gráficos.

Este programa fue creado en los 70's por Cleve Moler (matemático e informático especializado en el análisis numérico) y en la actualidad está distribuido por Math Works Inc desde 1984.

Algunas de las características de Matlab son que es relativamente lento comparado con otras aplicaciones informáticas como el "Fortran" o el "C" ,ya que Matlab es un lenguaje de interpretación. Al programar con Matlab es más corta la estructura, existe una gran variedad de "Toolboxes", es simple de entender y de usar.

USOS DEL MATLAB

El programa Matlab cuenta con una gran variedad de usos, siendo los más importantes: simular, modelar, crear prototipos, analizar datos y encontrar soluciones a sistemas complejos. Actualmente Matlab cuenta con cerca de 20 Toolboxes para usos tan dispares como por ejemplo: acústica, aeronáutica, astronomía, biología, biotecnología, cálculo, control, estadística, robótica, etc..

CONJUNTO DE FUNCIONES O TOOLBOXES

Las ToolBoxes de Matlab son conjuntos de funciones para aplicaciones específicas como las que se han citado en el apartado anterior, en nuestro proyecto se han utilizado las siguientes ToolBoxes:

- RWTHMindstormsNXTv4.03
- Toolbox_fast_marching

La toolBox **RWTHMindstormsNXTv4.03** contiene todas las funciones necesarias para el control del hardware del robot móvil de Lego (servomotores, sensores, etc..), mientras que la **toolbox_fast_marching** contiene todas las funciones necesarias para la implementación del algoritmo de marcha rápida o Fast_Marching con el que conseguimos trazar la ruta más corta entre dos puntos cualesquiera dentro del laberinto.

EMPEZAR A TRABAJAR CON MATLAB

Una vez accedemos al programa nos aparecerá la siguiente pantalla:

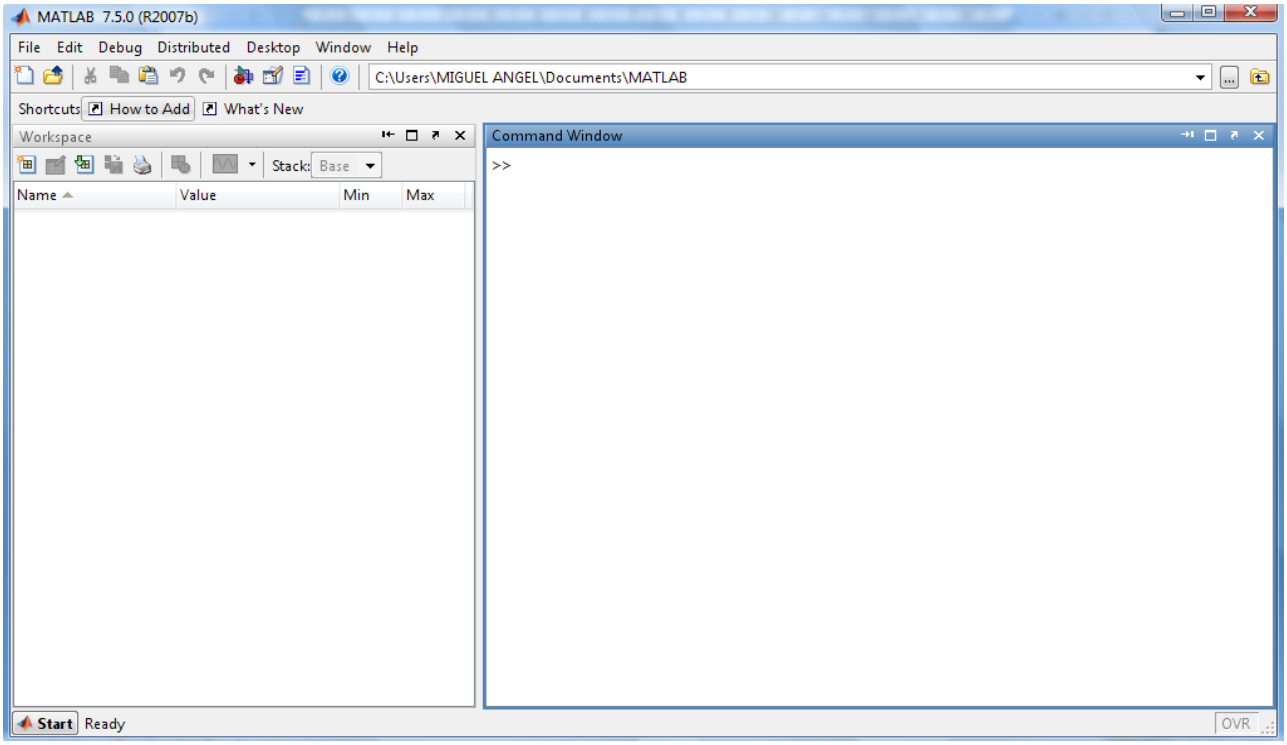


Figura 35. Página principal de matlab

Como podemos ver en la parte derecha de la pantalla mostrada en la figura 35 tenemos “comand Windows”, esta parte de la pantalla se utiliza para escribir los comandos que se ejecutarán con Matlab.

En la parte izquierda tenemos “work space”, en esta ventana vamos viendo los sucesivos valores que van tomando las variables que forman parte de los comandos o programas ejecutados desde “comand Windows”. Esta pantalla resulta de especial utilidad cuando trabajamos con el Debugger o depurador que se utiliza para la depuración de los programas de modo que en ella podemos visualizar los cambios de datos que se van produciendo en las variables a medida que vamos ejecutando las instrucciones del programa o función.

Lo primero que hay que hacer es proceder a incluir en el “path” de Matlab la ubicación exacta donde se encuentra la/as toolboxes que vamos a utilizar en la programación con Matlab.

En el desarrollo del proyecto hubo de instalarse una nueva versión del programa Matlab concretamente la 2009b, ello fue debido a que la versión 2007b no contaba con un icono para el modo Debugger o depuración denominado “continue”  . Este icono resulta muy útil para la depuración de programas en especial cuando llegamos a un bucle de tipo while por ej que ha de ejecutarse muchas veces.

Sin la opción que nos da este icono resultaría muy tedioso tener que estar ejecutando el bucle hasta poder salir de él tras numeras pulsaciones de ratón, para evitar esto pulsamos este icono para la ejecución del bucle y poder continuar en la línea siguiente del programa.

A continuación en las figuras 36 y 37 se muestra como ha de realizarse esta inclusión en el “path” de matlab:

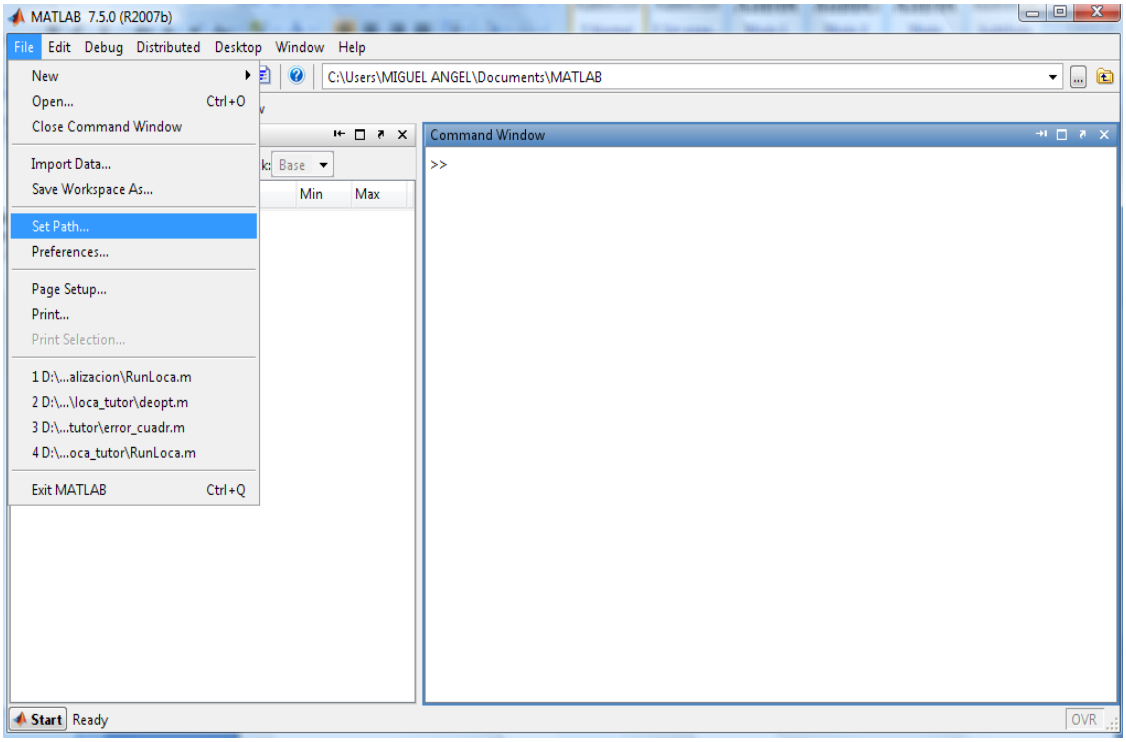


Figura 36. Actualización del ‘path’ de Matlab

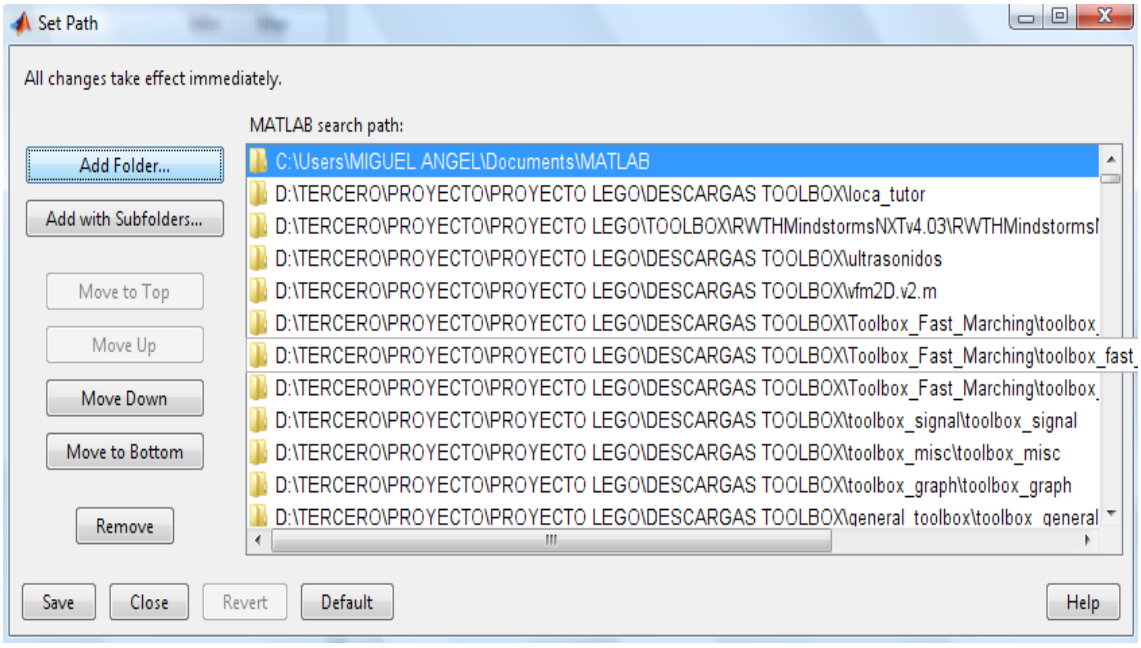


Figura 37. Añadido de directorios (Add folder)

Según la figura anterior, para añadir los directorios donde se encuentran las funciones que vamos a utilizar, tenemos que seleccionar Add Folder e indicar la ruta donde se encuentra la toolbox que queremos incluir en el "path" de Matlab. Una vez incluida procedemos a salvar los cambios seleccionando "Save".

En nuestro caso al tener instaladas dos versiones de matlab (2007b y 2009b) nos daba el problema indicado en figura 38:

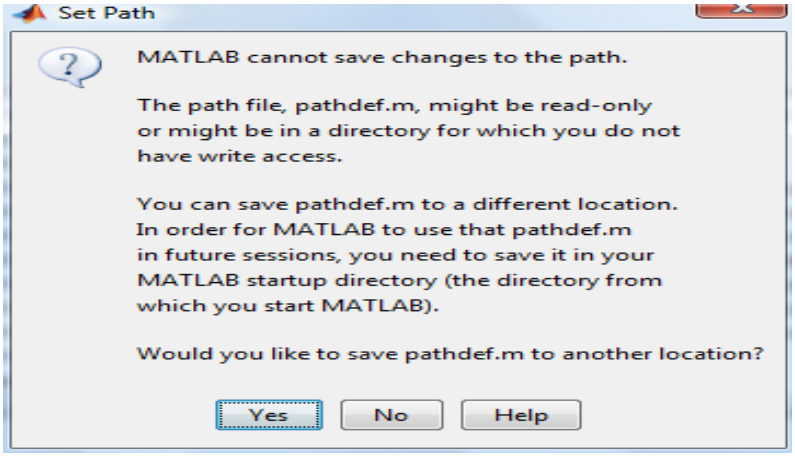


Figura 38. Incidencia por el uso de dos instalaciones de Matlab

Ello es debido a que diferentes versiones de Matlab no pueden utilizar el mismo archivo pathdef.m , este archivo es donde se guardan las rutas de los directorios que queremos incluir dentro del “path”.

Para solucionar este problema hemos empleado el archivo startup.m que es un archivo que se ejecuta nada más arrancar cualquier versión de Matlab, es decir es un archivo que comparten todas las versiones instaladas.

Como puede verse en la figura 39, hemos escrito dentro de startup.m todas las rutas de acceso que queríamos haber introducido en el “path” (ver figura 37) y no pudimos hacer directamente. Al arrancar cualquier versión de Matlab se añadirá al “path” existente el contenido de startup.m. En el caso de la nueva versión instalada (2009b) como dentro del “path” no hemos podido añadir nada, todo lo necesario se cargará con el contenido de startup.m

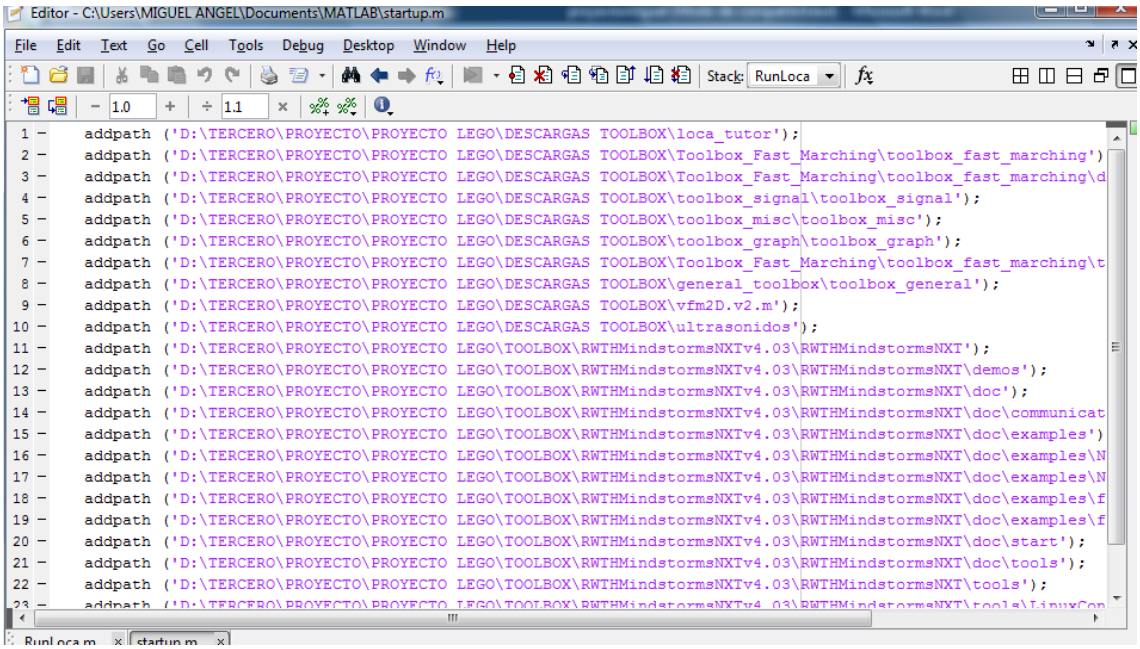


Figura 39. Contenido del archivo startup.m

La programación en Matlab [18] utiliza sentencias similares a la programación en "C" como son:

If,switch,for, while, break,etc..

La programación desarrollada para este proyecto se divide en dos partes principales: la correspondiente al **software para el guiado del robot móvil** por la ruta trazada por el algoritmo planificador y la correspondiente a determinar la **localización del robot móvil** con la mayor exactitud posible dentro del laberinto propuesto.

Para la primera parte de la programación (guiado del robot móvil) se han utilizado las funciones para el control de robots que vienen incluidas dentro de la toolbox RWTHMindstormsNXTv4.03.

FUNCIONES PARA EL CONTROL DEL ROBOT MOVIL (TOOLBOX RWTHMindstormsNXTv4.03)

-Todas las siguientes funciones que se usan para controlar los movimientos del robot van incluidas dentro la TOOLBOX RWTHMindstormsNXTv4.03:

NXTMotor

Esta función permite crear un objeto para el control de un puerto de salida (el ladrillo dispone de tres puertos numerados como A,B y C) con diferentes características o atribuciones.

La sintaxis de esta función es la siguiente:

```
M = NXTMotor(PORT, 'PropName1', PropValue1, 'PropName2', PropValue2,
...)
```

M: corresponde al nombre del objeto creado. Para gobernar un motor siempre hay que asociarle el nombre de un objeto.

PORT: aquí se indica el puerto utilizado para el control de los servomotores del robot móvil, normalmente utilizamos la sintaxis más simple como por ejemplo: 'A', si bien también podemos utilizar estas otras sintaxis: 0 o bien MOTOR_A.

'PropName1': aquí se indica la propiedad del motor a modificar

PropValue1: valor indicativo de la propiedad

Las propiedades de control del motor son:

Power: afecta al nivel de potencia que podemos aplicar al motor, los valores de dicho nivel están comprendidos en el rango -100 a +100

SpeedRegulation: esta propiedad puede tener dos valores, 'true' o 'false'. Si ponemos el valor 'true' el servomotor intentará mantener una velocidad constante ajustando la potencia de salida a la carga. Si ponemos el valor 'false' no se realiza ningún tipo de regulación.

Tacholimit: es para especificar el ángulo de giro que completará el servomotor en grados. Puede tener un valor comprendido entre 0 y 99999 no se admiten valores negativos ni con decimales.

ActionAtTachoLimit: aquí se indica la acción que sucederá al finalizar el giro en grados indicado en Tacholimit. Los valores utilizados son habitualmente 'Coast' o 'Brake'. El primero para frenar el motor por inercia (desconectando el flujo de corriente eléctrica al motor) y el segundo a demás de dicha desconexión se aplica un frenado activo. Hay que recordar si elegimos 'Brake' el motor quedará bloqueado por lo que no hay que forzarlo manualmente, para que vuelva a quedar con giro libre tenemos que ejecutar el comando StopMotor(MOTOR_A, 'off');

Limitaciones de la función NXTMotor: si enviamos un nuevo comando a un servomotor sin esperar a que haya terminado de ejecutar la orden anterior el ladrillo emite un pitido en señal de error, para evitar esto tenemos que utilizar la función WaitFor().

StopMotor

Esta función sirve para detener el funcionamiento de un servomotor , la parada puede ser por inercia ('Coast') o bien por frenado activo ('Brake'). El inconveniente del frenado activo es su elevado consumo de energía por lo que no es conveniente abusar de él.

La sintaxis de la función es: StopMotor(port, mode), donde port corresponde al puerto de salida utilizado y mode el modo de frenado elegido.

OpenUltrasonic

Esta función abre al mundo exterior el sensor de ultrasonidos y lo deja listo para empezar a tomar medidas de distancias. Sin esta orden el sensor de ultrasonidos no podrá realizar ninguna lectura.

La sintaxis es: OpenUltrasonic(port)

En 'port' indicamos el puerto de entrada del ladrillo al cual vamos a conectar el sensor de ultrasonidos, por ejemplo SENSOR_4.

GetUltrasonic

Esta función da la orden de almacenar en una variable el valor actual de la distancia existente entre el sensor de ultrasonidos y el objeto hacia donde se encuentra dirigido este sensor. Como se indicó en el apartado 3.1 la medida tiene una precisión de +/- 3 cm.

La sintaxis es: distance = GetUltraSonic(port), donde 'port' es el puerto al cual se encuentra conectado el sensor de ultrasonidos y 'distance' puede ser cualquier variable donde quedará almacenada la medida realizada.

CloseSensor(port)

Esta función viene habitualmente a finalizar el proceso de las mediciones realizadas por el sensor de ultrasonidos y cierra al mundo exterior dicho sensor hasta que reciba una nueva orden de OpenUltrasonic.

La sintaxis es: CloseSensor(port), donde 'port' es el puerto de entrada donde se encuentra conectado el sensor de ultrasonidos.

SendToNXT

Esta función sirve para una vez que ya tenemos todos los ajustes que deseamos para el servomotor (Tacholimit, power, ActionAtTachoLimit , etc..), validarlos mediante el envío de los mismos a dicho servomotor mediante el dispositivo "Bluetooth".

La sintaxis de esta función es: OBJ.SendToNXT(); donde OBJ es el objeto que hace referencia al servomotor.

ReadFromNXT

A parte de las funciones anteriores, existe una función llamada 'ReadFromNXT' que nos devuelve información sobre el estado actual de un servomotor. Con esto conseguimos que el robot nos informe de lo que está sucediendo realmente que no tiene porque coincidir exactamente con lo que le hemos ordenado. Este proceso de lectura de un servomotor forma parte de la técnica de Odometría.

Al realizar una llamada mediante 'ReadFromNXT' se nos devuelve una estructura de datos con diferentes campos que nos dan información en tiempo real del servomotor.

Los campos que forman parte de esta estructura son:

Port: hace referencia al servomotor de donde proviene la información (valores: 0, 1, y 2).

Power: hace referencia a la potencia suministrada al servomotor (valores en el siguiente rango -100 a +100)

IsRunning: es un indicativo de tipo 'boolean' que puede tomar únicamente dos valores (0 o 1)

Mientras IsRunning =1, se nos indica que el servomotor está en funcionamiento. Cuando valga '0' implica que el servomotor se ha detenido.

SpeedRegulation: Al igual que el anterior es un indicativo booleano que hace referencia a que el servomotor funciona con regulación de la velocidad (cuando su valor es '1') .

TachoLimit: al igual que lo indicado para la función NXTMotor, hace referencia al límite e giro del motor.

Position: Nos da el valor en tiempo real de la posición del motor en grados

La sintaxis de esta función es: DATA = OBJ.ReadFromNXT(); donde objeto hace referencia al servomotor de donde vamos a realizar la lectura de datos.

A continuación y para mayor detalle se pone un sencillo ejemplo para ilustrar el funcionamiento de esta función:

```
motorC = NXTMotor('C', 'Power',8,'TachoLimit', 90);
 motorC.ResetPosition();
 motorC.SendToNXT();
 OpenUltrasonic(SENSOR_4);
 h= GetUltrasonic(SENSOR_4);
 [DATA1] = motorC.ReadFromNXT()
 while(DATA1.IsRunning)
     h= GetUltrasonic(SENSOR_4);
     disp(sprintf('las distancias escaneadas son: %d % complete/n', h));
     DATA1 = motorC.ReadFromNXT(); % refresh
  end%while
```

Se trata de un sencillo programa para hacer que un sensor de ultrasonidos colocado en un servomotor gire por un ángulo de 90º y realice diversas medidas durante el tiempo que el servomotor tarda en completar dicho giro. Como se puede observar antes de introducirse en el bucle se realiza una primera toma de datos, después se pasa al bluce 'while' que es ejecutado mientras el servomotor se encuentre en movimiento (IsRunning=1) y al final del bucle se vuelve a refrescar la información con una nueva toma de datos.

ResetPosition

Esta función se utiliza cada vez que comenzamos a leer datos mediante ReadFromNXT y su función es poner a cero el campo 'Position' que es el contador de los grados que lleva girados el servomotor.

La sintaxis de esta función es: OBJ.ResetPosition( ); , donde objeto hace referencia al servomotor de donde vamos a realizar la lectura de datos.

3.4 FUNCIONES IMPLEMENTADAS MEDIANTE PROGRAMA MATLAB PARA EL SEGUIMIENTO DE LA RUTA PLANIFICADA

Myapp:

Como se indicó en los apartados anteriores para trazar la ruta que deseamos que siga el robot móvil, se ha utilizado una función realizada mediante la aplicación Matlab denominada 'myapp.m'. Esta función ya desarrollada por el Departamento de Sistemas y Automática de la Universidad Carlos III, permite una visualización gráfica del laberinto por dónde se desea que se mueva nuestro robot móvil. Esta función ha sido desarrollada basándose en la metodología "Fast_Marching" o avance rápido [19].

El primer paso para utilizar esta función es escribir su nombre (myapp) en la ventana de comandos de Matlab. Tras ejecutar el comando correspondiente al nombre de la función según podemos ver en la figura 40, nos muestra por pantalla la imagen perteneciente al laberinto que hemos elegido para el movimiento del robot móvil:

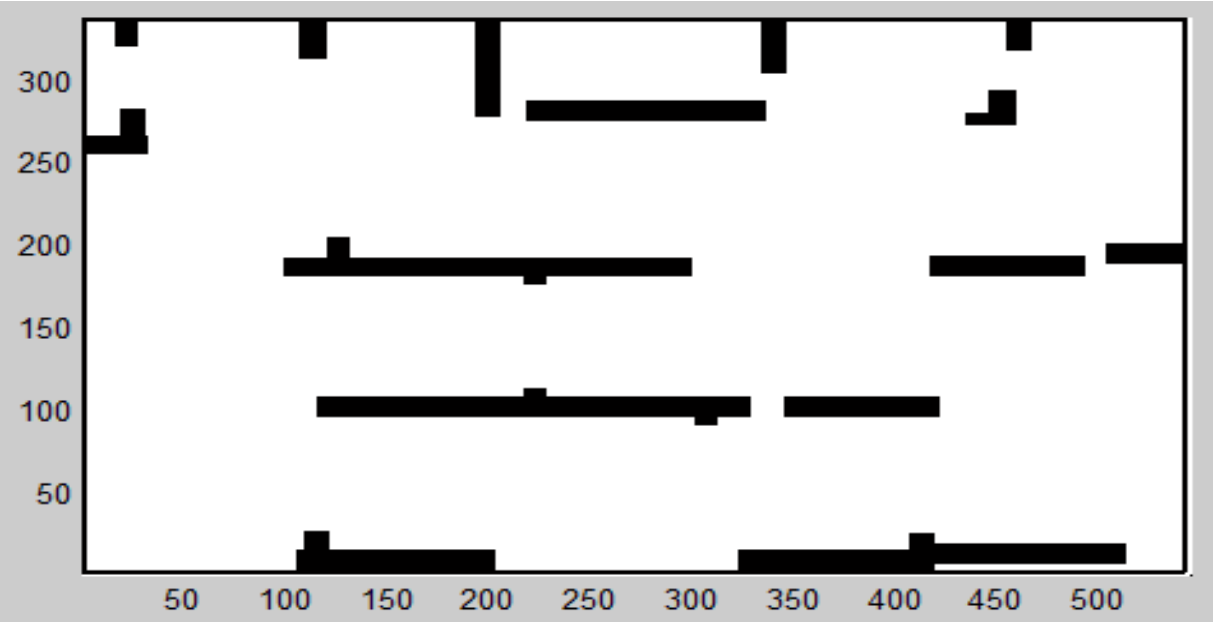


Figura 40. Vista del laberinto previa

Como se aprecia en las figuras 41 y 42, el puntero de nuestro ratón se ha transformado en dos ejes móviles XY para el marcado de los puntos inicial y final de la ruta a seguir por el robot. A continuación procederemos a marcar dichos puntos empezando por el punto inicial de la ruta:

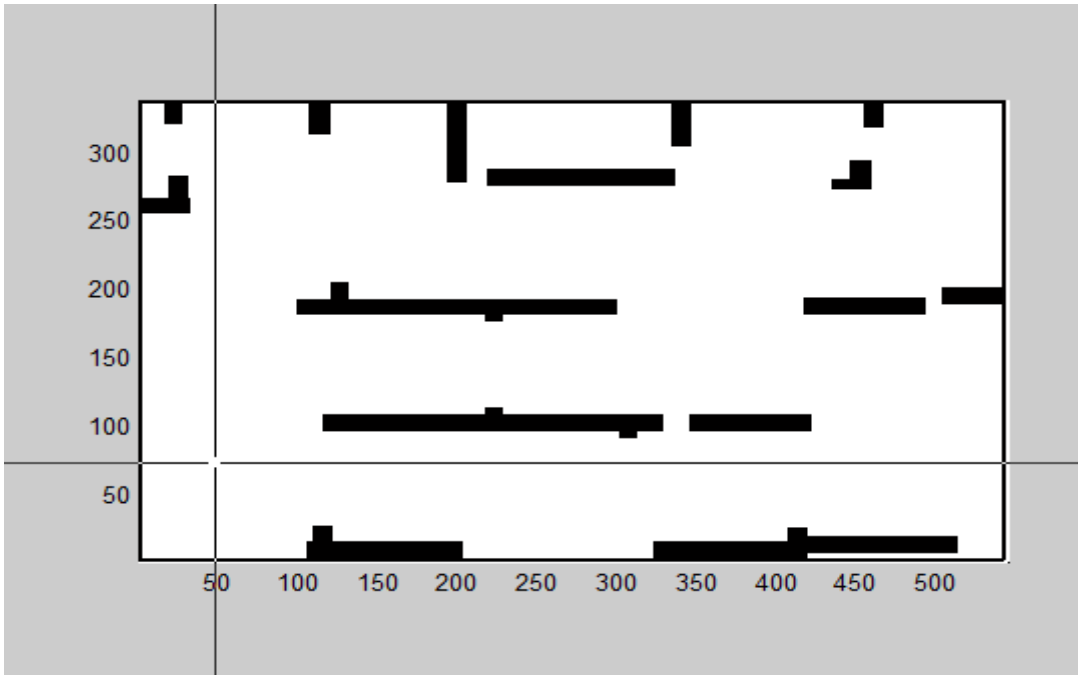


Figura 41. Vista del cursor para definir ruta

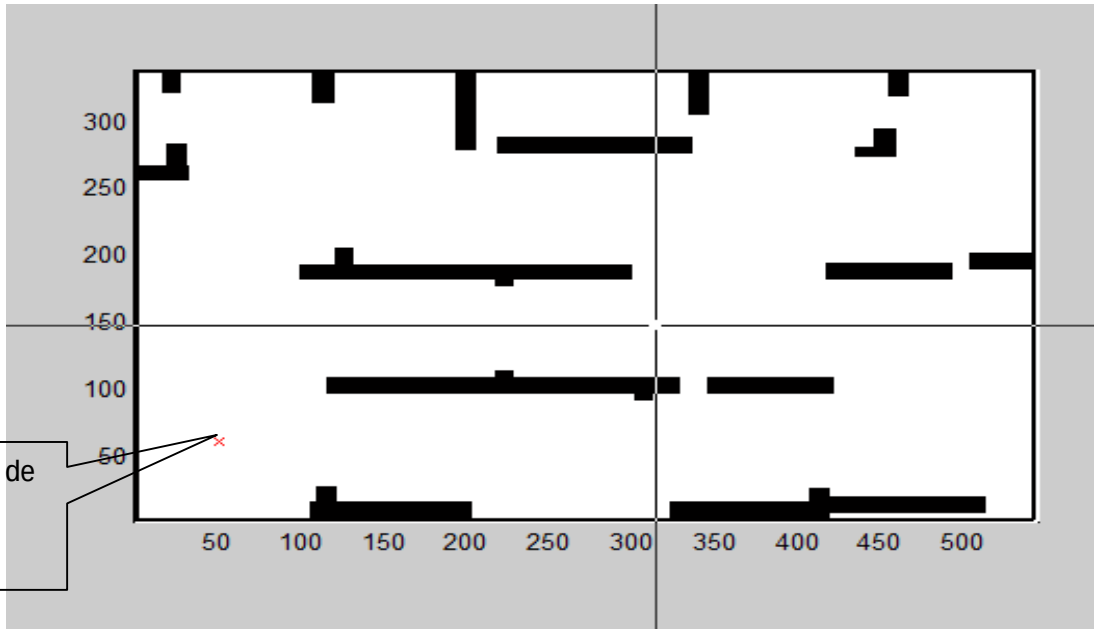


Figura 42. Punto inicial de la ruta

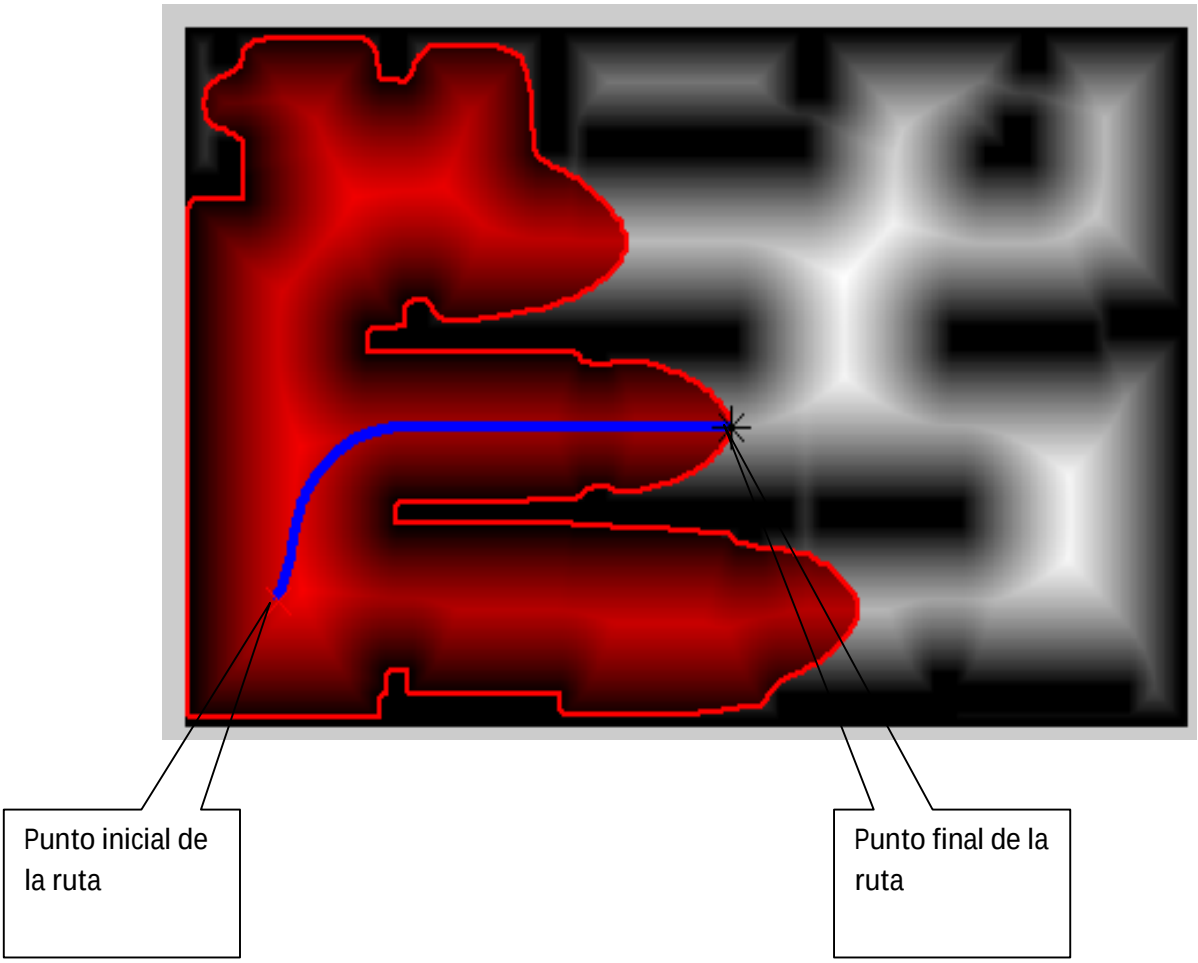


Figura 43. Puntos inicial, final y ruta trazada entre ambos

Como puede observarse en la figura 43, una vez hemos señalado los puntos inicial y final de la ruta la función 'myapp' nos representa gráficamente la ruta comprendida entre ambos puntos. Dado que la función utiliza la metodología "Fast_Marching" entendemos que la ruta trazada entre esos dos puntos es la ruta más corta (distancia mínima geométrica).

Como se puede apreciar la ruta trazada ha tenido en cuenta los obstáculos que la rodean (paredes del laberinto) por lo cual mantiene una separación con respecto a dichos obstáculos y en los giros se abre para dar tiempo al robot móvil a realizarlos con una separación suficiente de las aristas o esquinas que de lo contrario provocarían que el robot acabara chocando con ellas.

El siguiente paso que hay llevar a cabo es lógicamente el seguimiento de la ruta por parte del robot móvil. Para ello se han desarrollado dos funciones y el programa principal que se encarga de llamar a éstas cuando se considera necesario.

Para hacer que el robot móvil parta de un punto donde se encuentra situado y llegue hasta un punto determinado de la ruta, se estimado necesario completar dos movimientos básicos: giro de orientación hacia el punto de la ruta desde el ángulo que presenta el robot inicialmente con respecto al eje 'X' y posterior traslación en línea recta hasta el punto determinado de la ruta. Estos movimientos se van a repetir un número determinado de veces (intervalos) hasta poder completar totalmente la ruta, este tipo de movimiento hace que el seguimiento de la ruta tenga forma poligonal.

Tanto las funciones como el programa principal han sido realizados mediante la aplicación Matlab, en el **Apéndice A** se encuentra explicado el funcionamiento del programa principal y de las funciones y en el **Apéndice B** se encuentra el código de programa correspondiente.

Funciones implementadas

Calculocoord.m

Girocuadrante.m

A continuación se realiza una breve descripción del funcionamiento de las funciones y el programa principal:

Función Calculocoord.m

La sintaxis de esta función es:

[angulo,distpuntos]=calculocoord(xrut,yrut,xrob,yrob)

Esta función sirve para conocer mediante los dos argumentos que nos devuelve, el ángulo que forma la línea imaginaria que une el punto donde se encuentra situado el robot y el punto de la ruta hacia donde queremos dirigirlo y la distancia en cm que deberá recorrer nuestro robot móvil para llegar al destino.

Función girocuadrante.m

La sintaxis de esta función es:

Girocuadrante (angulo,theta,distpuntos,xrut,yrut,xrob,yrob,j)

Esta función sirve para según lo indicado en la pagina. 70, el robot primeramente realice un giro de orientación hacia el punto de la ruta y seguidamente efectúe una traslación en línea recta para llegar hasta dicho punto.

Programa principal

Este programa no forma parte de una función definida sino que se carga en su totalidad en la ventana de comandos de Matlab (Comand Windows)

Este programa es el encargado de gestionar todo lo necesario para que nuestro robot móvil sea capaz de seguir la ruta planificada, relocalizarse y corregir errores de desvío de ruta, cosa por otra parte de esperar cuando el robot se mueve basándose en un funcionamiento odometrico.

3.5 RELOCALIZACION

3.5.1 ALGORITMO EVOLUTIVO "DIFFERENTIAL EVOLUTION" (DE)

"Differential Evolution" [20] es una especie de algoritmo evolutivo pero mejorado y es utilizado para la resolución de problemas de optimización.

(DE) fue desarrollado por Rainer Storn y Kenneth Price para la optimización de espacios continuos.

(DE) se compone de los siguientes pasos:

- Inicialización de la población
- Generación de perturbación
- Mutación
- Selección (del individuo más apto para pasar a la siguiente generación)

En (DE), las variables se representan mediante vectores de números reales. Cada vector tiene una dirección única porque cada uno de ellos tiene que entrar en la competición.

A cada vector le corresponde un índice que va desde el 0 hasta NP-1, siendo NP el número total de individuos que forman la población inicial.

La población inicial se genera de forma aleatoria y se seleccionan tres individuos (vectores) para llevar a cabo la generación de una nueva solución.

Uno de estos vectores se denomina "vector base" y se perturba con el vector diferencia de los otros dos vectores.

Lo primero que se hace es generar una población inicial de manera aleatoria según se muestra en la figura 44:

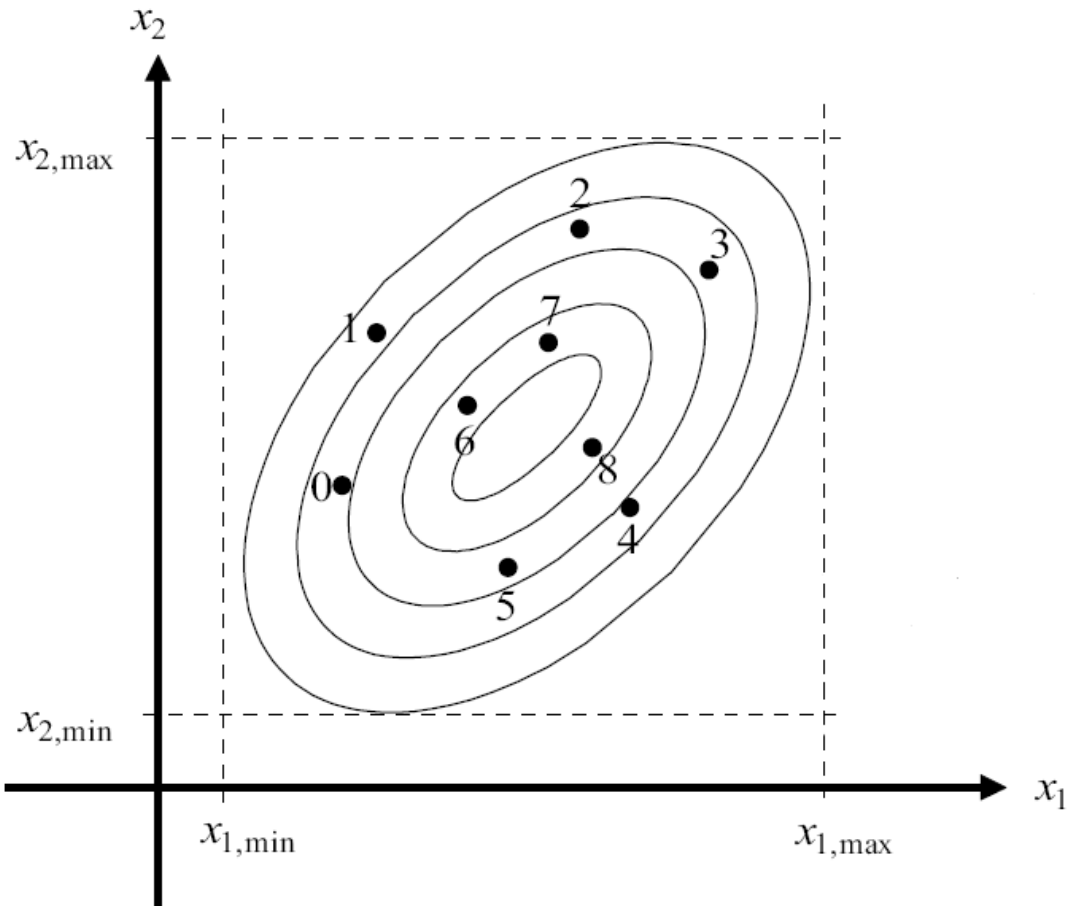


Figura 44. Creación de población inicial

Seguidamente se genera una perturbación mediante la diferencia de dos vectores elegidos aleatoriamente (Xr1 e Xr2) como puede verse en la figura 45:

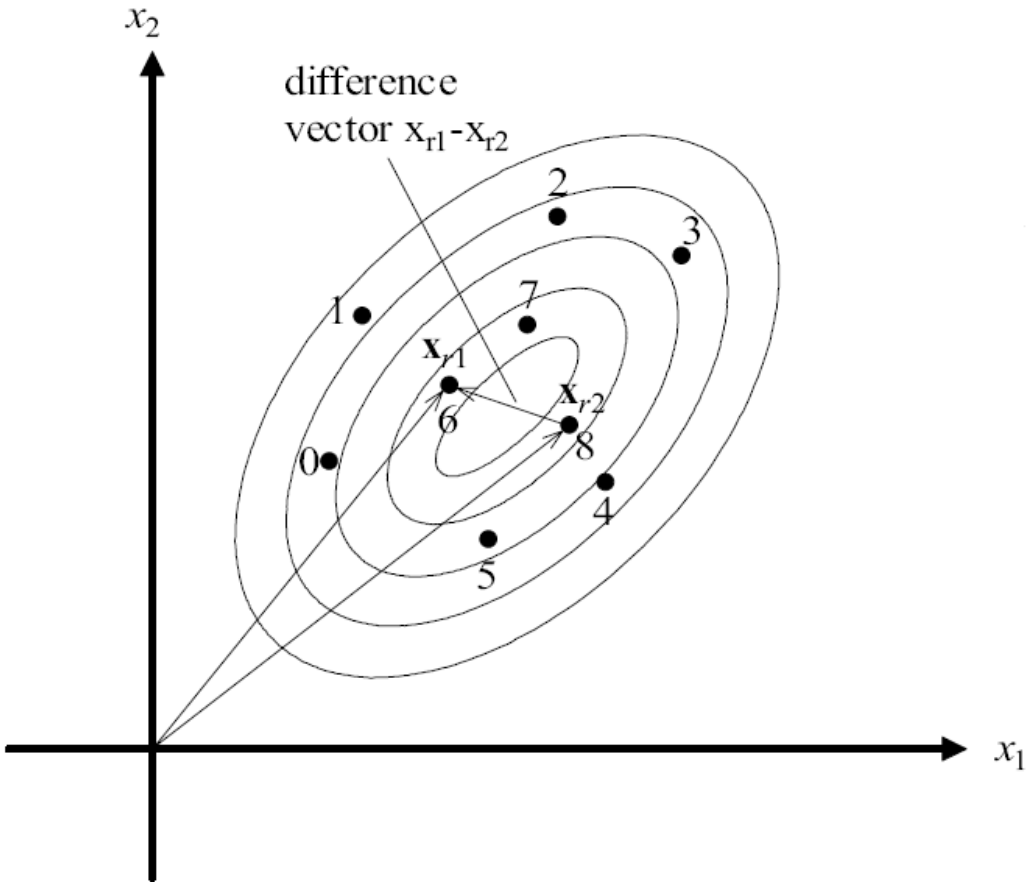


Figura 45. Introducción de la perturbación

El siguiente paso es la mutación. Mediante la suma a otro vector aleatorio (Xr3) de la perturbación o diferencia (Xr1-Xr2).

Se obtiene el vector de prueba U0 mostrado en la figura 46:

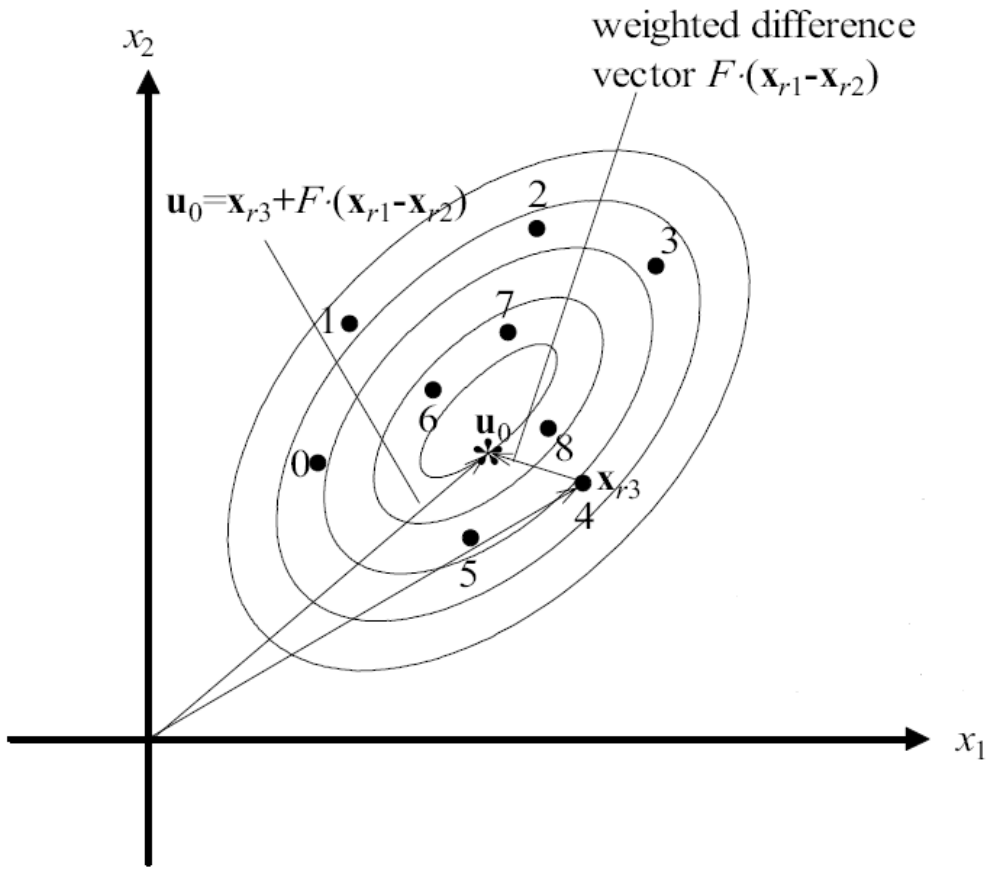


Figura 46 . Mutación

En la figura 47 se muestra el proceso de selección, para ello U0 compite contra el vector nº 0 de la población. El vector con el menor valor de la función objetivo será seleccionado como vector nº 0 de la siguiente población:

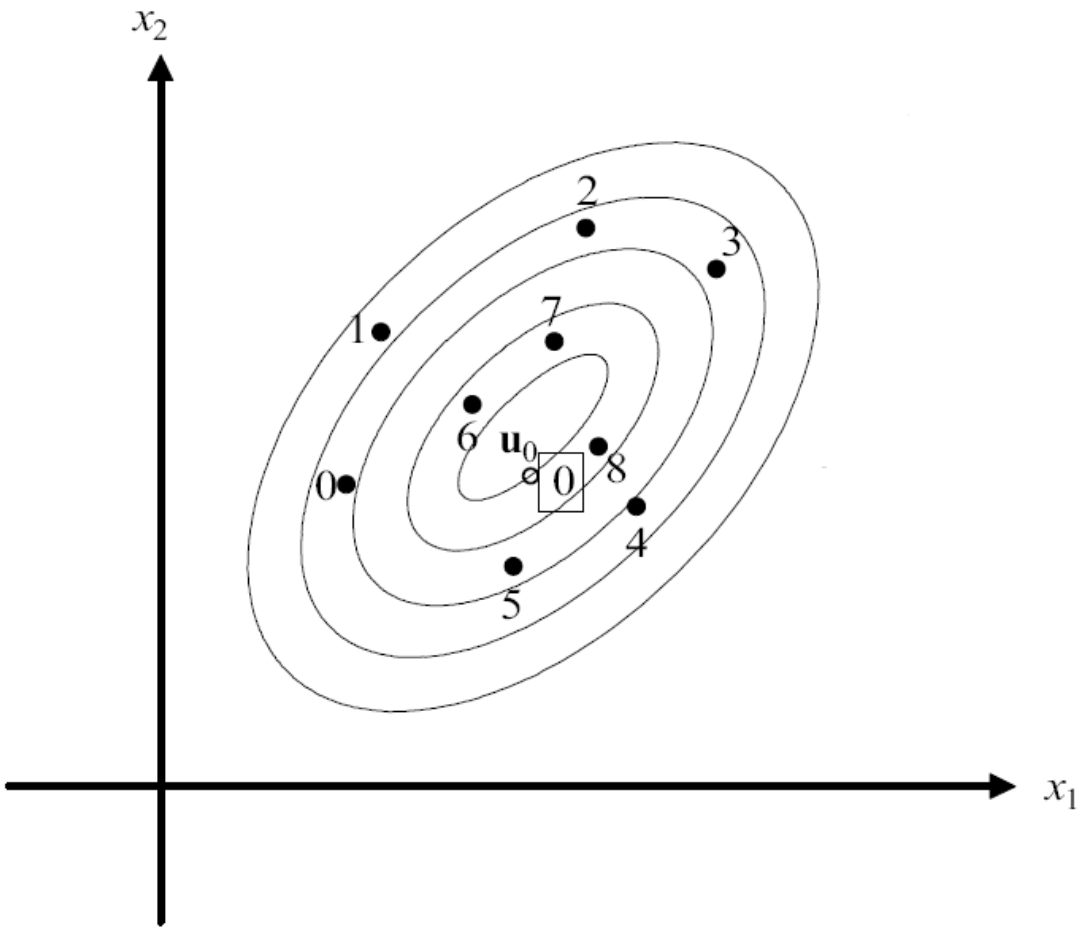


Figura 47. Selección

Mediante Xr1,Xr2 y Xr3 se genera un nuevo vector de prueba U1:

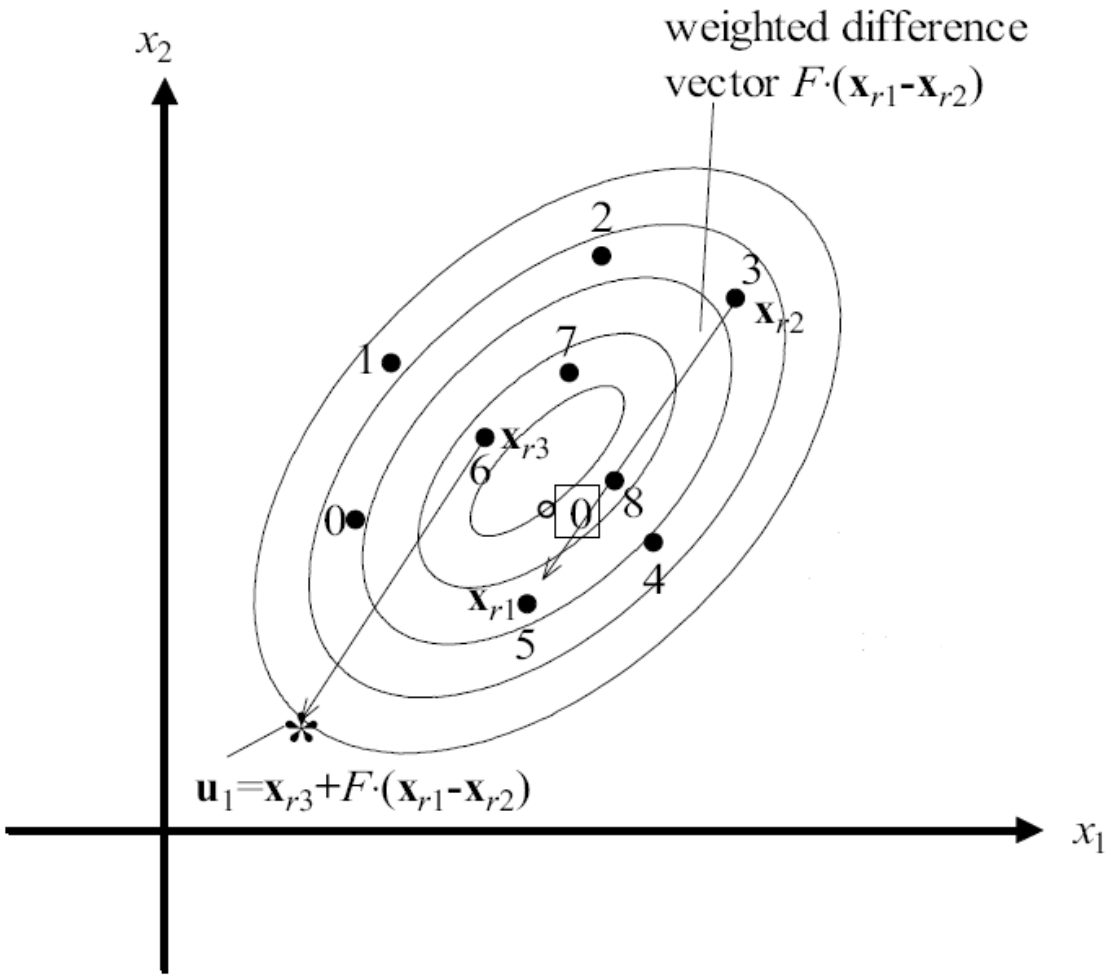


Figura 48. Un vector de la nueva población es mutado mediante una perturbación aleatoria

Ahora U1 compite contra el vector con el mismo índice de la población y pierde. Esta vez el vector de prueba pierde y el vector con el índice 1 de la antigua población será seleccionado para sobrevivir a la siguiente población:

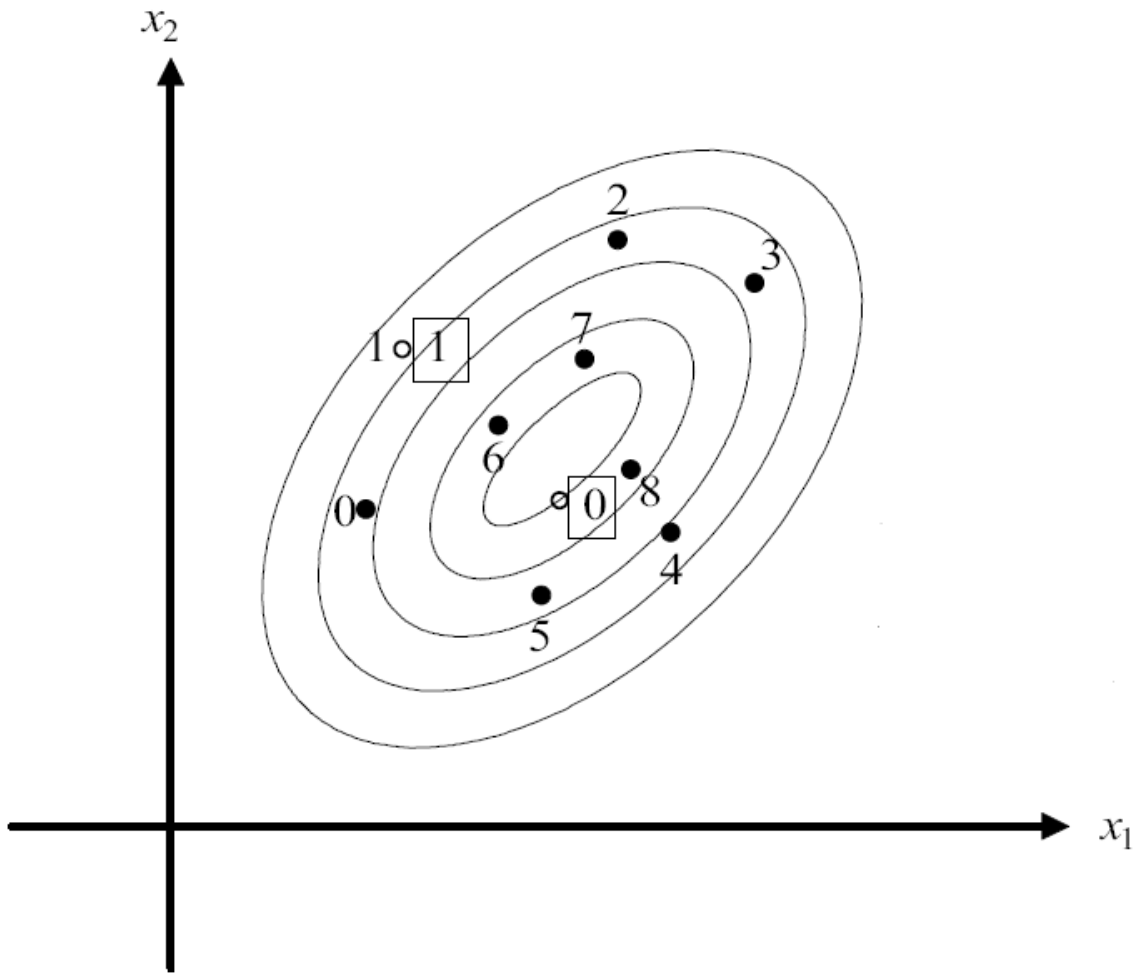


Figura 49. Selección.

El proceso de introducir una perturbación aleatoria para generar una mutación y la posterior selección del individuo más apto mediante la competición por parejas de vectores (el vector de prueba con el vector de población con el mismo índice) se repite hasta que todos los vectores de la población (NP) hayan competido con el vector de prueba generado aleatoriamente según se muestra en las figuras 48 y 49.

Una vez el último vector de prueba ha sido contrastado, los supervivientes de la competición por parejas de la población inicial (NP) se convierten en padres para la siguiente generación en el ciclo de evolución.

En este proyecto, el algoritmo “Differential Evolution” se encarga básicamente de reducir el error entre lo que ve el robot mediante el sensor de ultrasonidos y lo que realmente debería de ver si se encontrara justamente en un punto determinado.

3.5.2 FUNCIONES UTILIZADAS PARA LA RELOCALIZACION

Las funciones utilizadas para la relocalización nuestro robot han sido:

- Ultra4mcl
- RunLoca

FUNCION Ultra4mcl

Mediante esta función y a través del sensor de ultrasonidos unido al eje del servomotor situado en la parte alta del robot (ver figura. 19) se van realizando medidas de distancia a intervalos de 15º con el fin de obtener un diagrama polar (ver figura 52). Se trata pues de que el robot nos informe sobre lo que ve a su alrededor.

El resultado de estas medidas será uno de los argumentos de entrada a la función RunLoca.

Para llamar a esta función basta con ejecutarla directamente desde la ventana de comandos de Matlab.

FUNCION RunLoca

Esta función ha sido desarrollada por el Departamento de Sistemas y Automática de la Universidad Carlos III y está basada en el algoritmo evolutivo (DE) o "Differential Evolution". Mediante esta función nos proponemos poder dar con la mayor exactitud posible los valores de la terna (X,Y,θ) también denominada en el mundo de la robótica "pose" o lugar donde se encuentra el robot móvil.

La sintaxis de esta función es:

FVr_bestmem= RunLoca(v0,theta,xrob,yrob)

4. RESULTADOS EXPERIMENTALES

Una vez construido nuestro prototipo de robot móvil con el kit de Lego y el entorno en 3D por donde éste deberá de moverse, procederemos a realizar pruebas para comprobar el correcto seguimiento de las rutas propuestas.

Para empezar se marcaron físicamente en el entorno los puntos correspondientes a las coordenadas que hemos elegido como comienzo de las rutas que muestran los resultados experimentales.

En la figura 50 se muestra un gráfico del entorno con diferentes coordenadas y en color rojo las que se han elegido como inicio de las rutas propuestas:

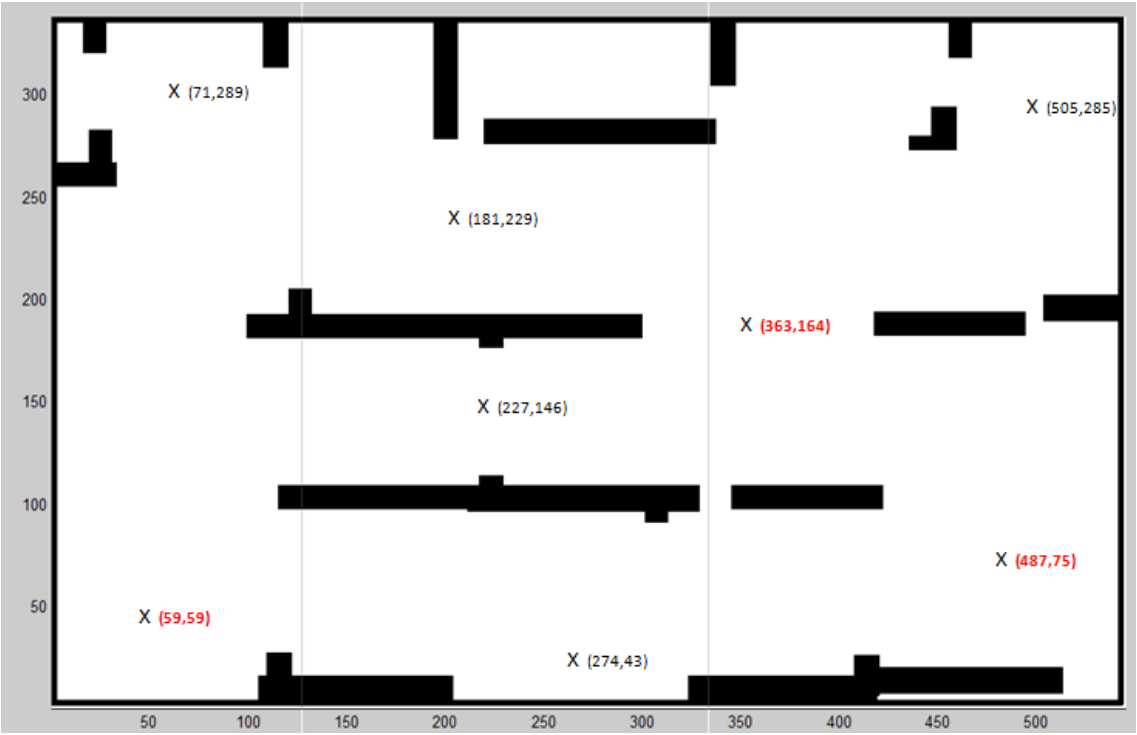


Figura 50. Inicios de rutas propuestas

Para la realización de cada una de las rutas, ejecutamos el programa principal en la ventana "Command Windows" de Matlab. Este programa principal incluye las llamadas necesarias a todas las funciones implementadas para nuestro objetivo.

En la línea de programa donde viene la función que genera la ruta siguiendo la metodología Fast Marching "myapp", procedemos a indicar las coordenadas de origen y las coordenadas del punto de destino.

Seguidamente hemos de cargar las coordenadas iníciales del robot que hemos hecho corresponder con las del punto elegido como origen de ruta, si bien físicamente sabemos que no tienen porque coincidir ya que nosotros a simple vista no podemos asegurar que colocamos el robot justamente sobre el punto de origen.

Además de las coordenadas del robot, para completar su "pose inicial" tenemos que incluir el ángulo de orientación que decidimos que presenta inicialmente con respecto al eje X.

Por último hemos de incluir nuevamente las coordenadas del destino en el nuevo llamamiento a la función "myapp" que se produce tras el proceso de relocalización con el fin de mantener el punto de destino elegido inicialmente.

Nota: tanto el punto de origen como el de destino siguen el esquema (x,y,θ)

RUTA 1.

Origen: (363,164)

Destino: (59,59)

Orientación inicial: 180º

En esta prueba han sido efectuadas dos relocalizaciones y la ruta se ha dividido en dos tramos.

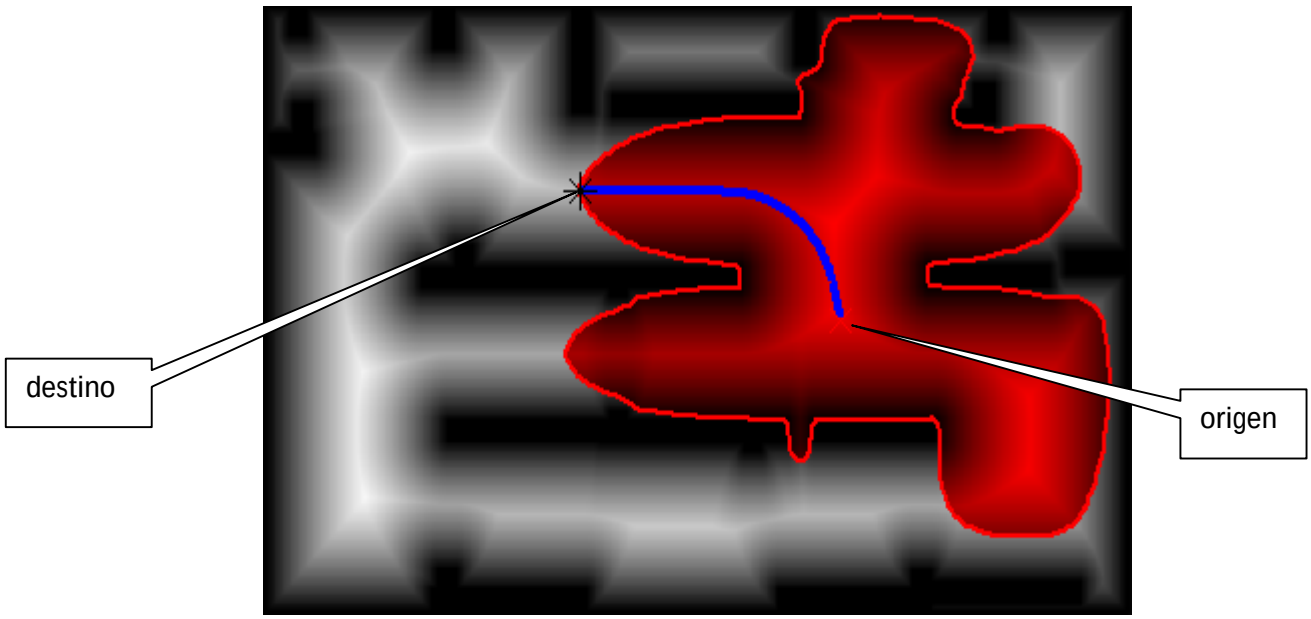


Figura 51. Trazado del primer tramo de la ruta (ruta 1)

Hasta llegar el momento de efectuar la relocalización, el seguimiento de la ruta se hace únicamente por Odometría (giros necesarios y avances) y el recorrido se muestra en la figura 51.

Llegado el momento elegido para efectuar la relocalización, lo primero que se hace es que el sensor de ultrasonidos tome las medidas del contorno (lo que rodea al robot) para pasar dichas medidas a la función que efectúa la relocalización (RunLoca). En la figura 52 se muestra el diagrama polar visualizado que representa gráficamente las medidas tomadas a intervalos de 15º hasta completar un giro de 360.

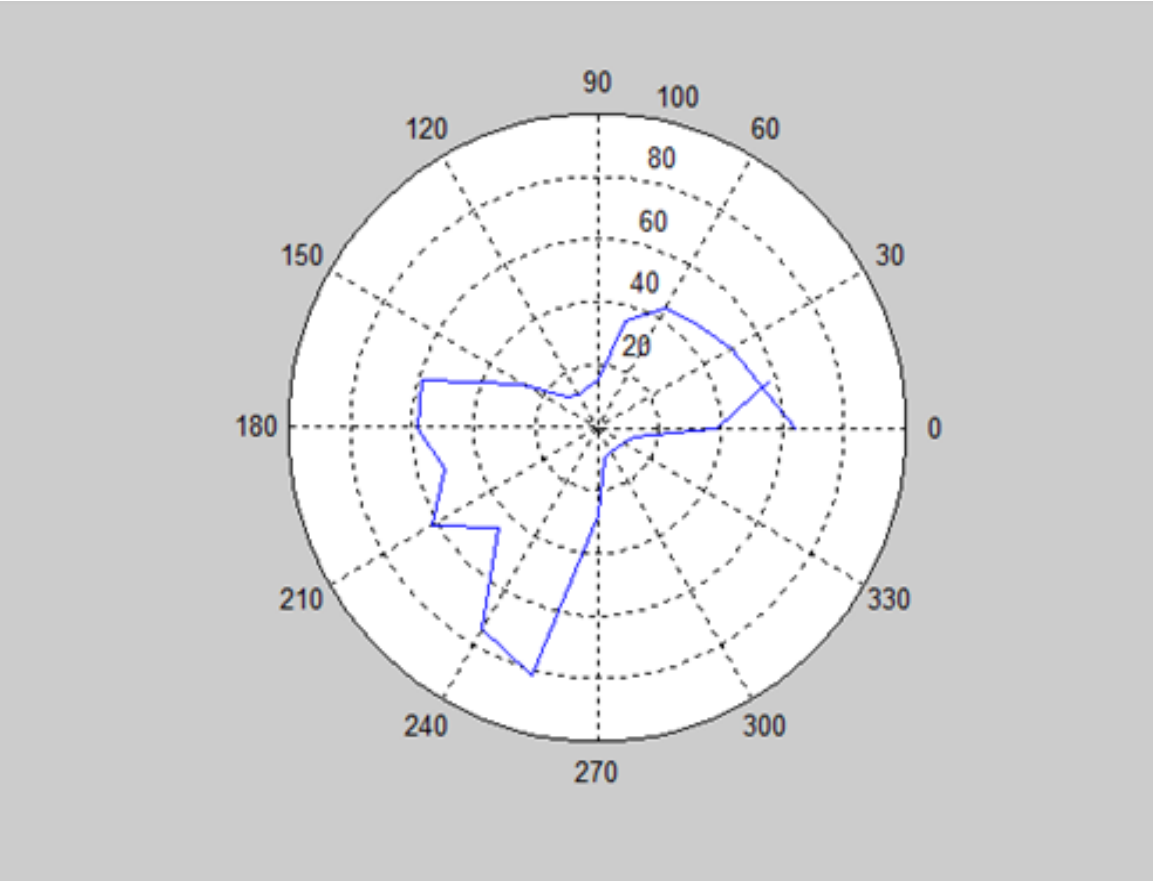


Figura 52 . Diagrama polar primera relocalización (ruta 1)

Como puede verse en la figura 56, una vez ha convergido el algoritmo "Differential Evolution" nos devuelve la mejor solución o "pose" donde se supone que se encontrará realmente el robot, seguidamente el programa procede a la generación de una nueva ruta cuyo punto inicial coincide con las coordenadas de la "pose" obtenida mediante optimización. Por supuesto el punto de destino no ha cambiado sigue siendo el mismo que antes de relocalizarse con el fin de llegar hasta el destino elegido.

En la figura 53 se muestra el trazado de la nueva ruta tras la primera relocalización:

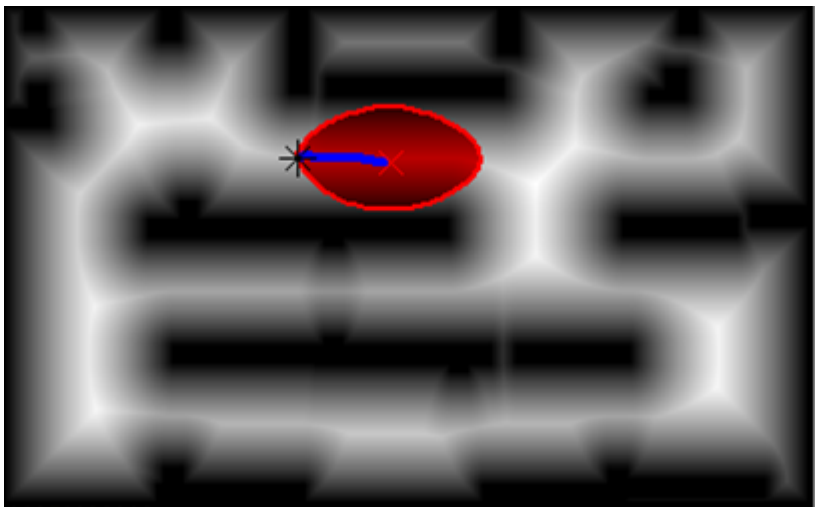


Figura 53. Nueva ruta tras la primera relocalización (ruta1)

En el segundo tramo de la ruta según muestran las figuras 54,55 y 56 se vuelven a repetir todos los procedimientos explicados para el primero. Seguidamente se muestran las imágenes obtenidas:

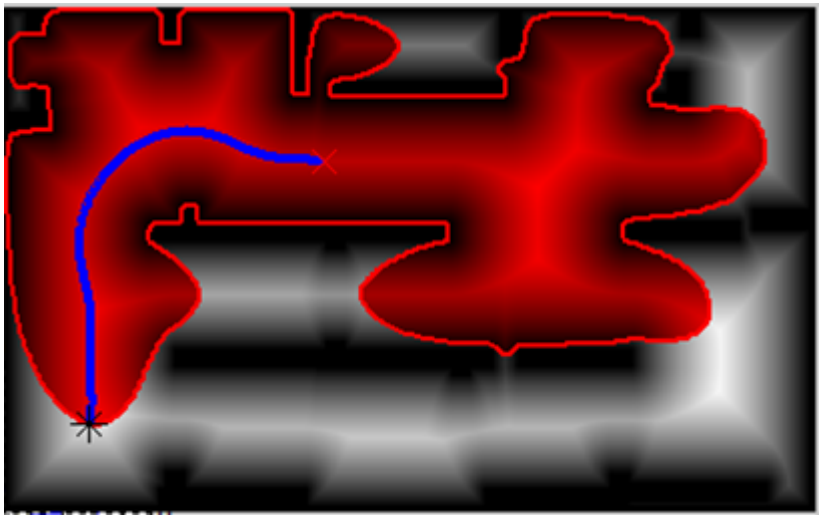


Figura 54. Trazado de segundo tramo de ruta (ruta 1)

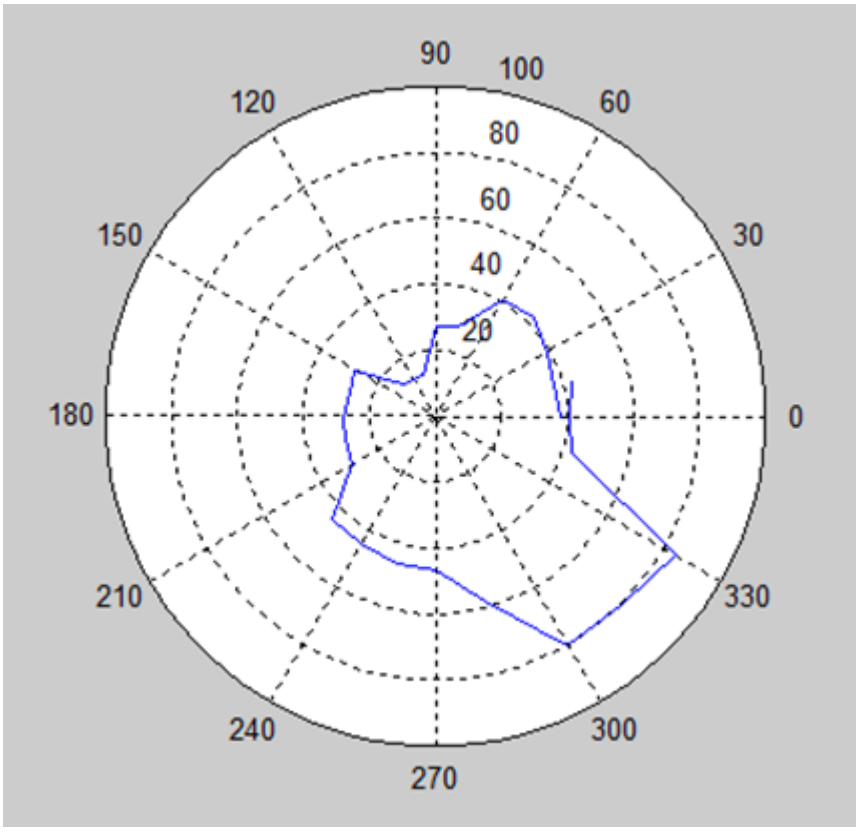


Figura 55. Diagrama polar previo a relocalización 2º tramo (ruta 1)

```
F_cost =

    8.271293731938191e+002

Iteration: 14,   Best: 827.129373,   F_weight: 0.850000,   F_CR: 1.000000,   I_NP: 100
best(1) = 169.765
best(2) = 62.688
best(3) = 4.93754

FVr_bestmem =

    1.697649324417514e+002    6.268796897888579e+001    4.937542337940090e+000


S_bestval =

      I_nc: 0
    FVr_ca: 0
      I_no: 1
    FVr_oa: 8.271293731938191e+002


I_nfeval =

        1500

X teoria es: 5.445150e+001
Y teoria es: 1.611000e+002
theta teoria es: 278

FVr_bestmem =

  169.7649   62.6880    4.9375
```

Figura 56. Datos mostrados sobre la mejor “pose” tras la optimización (ruta 1)

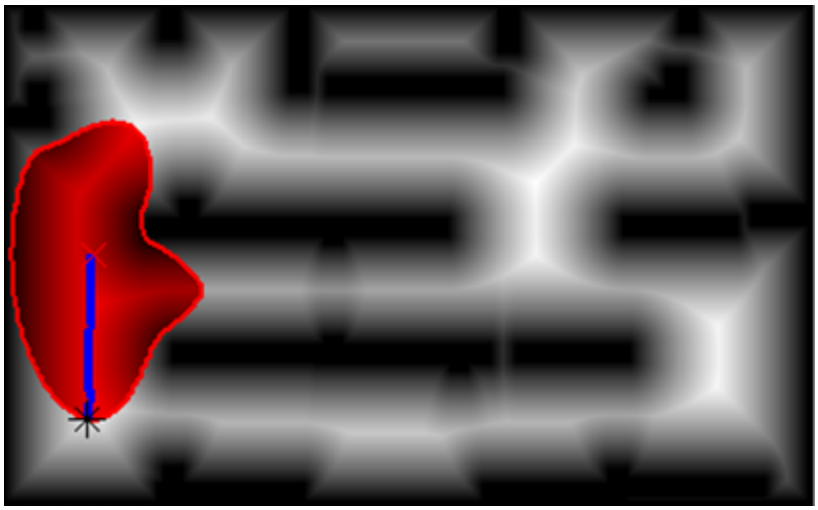
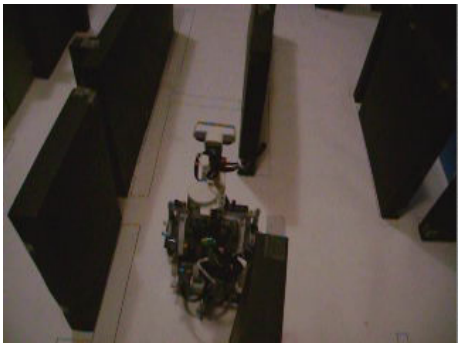
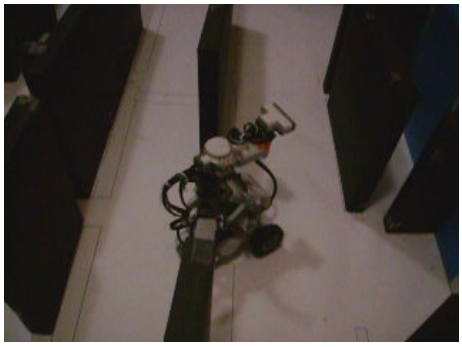


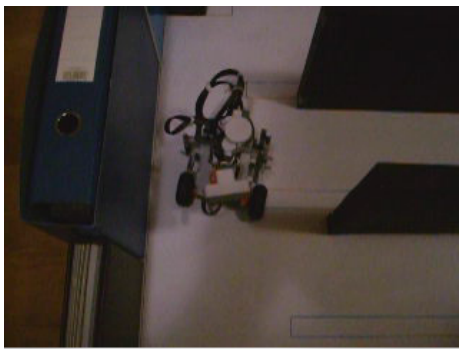
Figura 57. Nueva ruta tras la segunda relocalización (ruta 1)

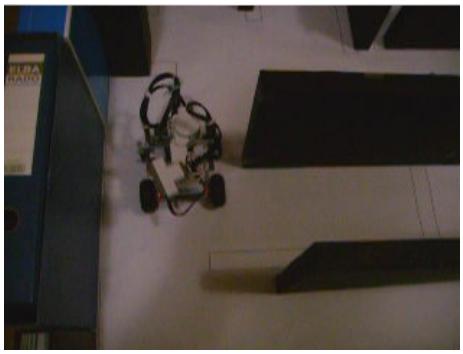
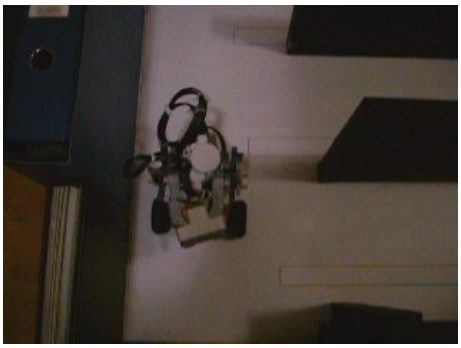
A continuación se muestra una secuencia de fotogramas que muestran el seguimiento de la ruta por el robot móvil:







En los fotogramas podemos ver como el robot móvil parte del reposo con una orientación aproximada de 180º e inicia el seguimiento realizando una serie de giros progresivos a derechas para evitar el primer obstáculo

Tras superar el primer obstáculo continúa su camino y cuando se encuentra aproximadamente paralelo al pasillo realiza la primera relocalización. Tras efectuarla modifica la trayectoria manteniendo el punto de destino pero tomando como punto inicial el determinado por el algoritmo "Differential Evolution".

Continúa con el segundo trazado de ruta evitando la esquina izquierda al final del pasillo mediante Odometría. A la salida del giro realiza una nueva relocalización ante la previsible acumulación de errores tras los giros efectuados para solventar la esquina y modifica nuevamente la ruta para acabar llegando al punto de destino convenido.

RUTA 2.

Origen: (487,75)

Destino: (134,239)

Orientación inicial: 90º

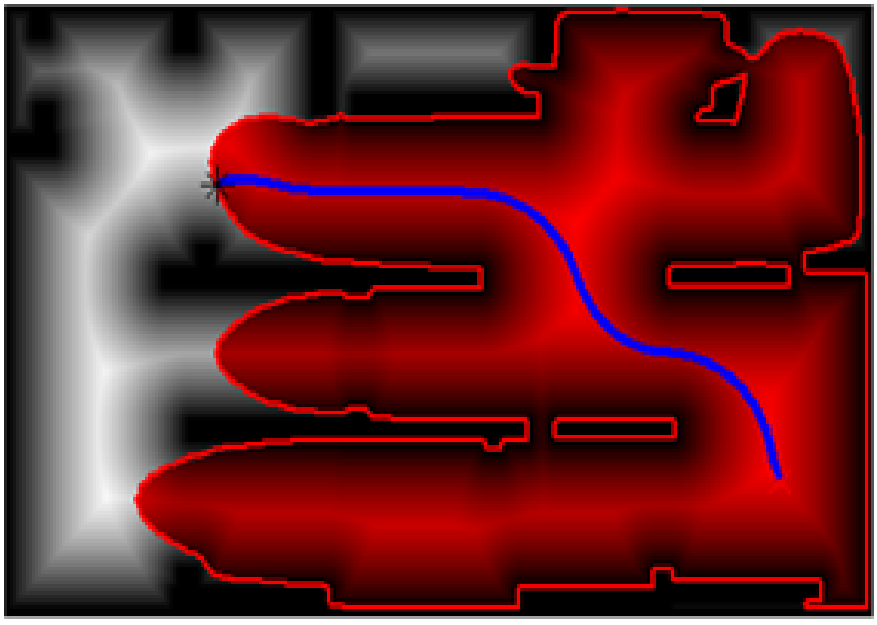


Figura 58. Trazado de ruta inicial (ruta 2)

Al igual que en la Ruta 1, el robot móvil comienza a seguir la ruta empleando únicamente Odometría hasta llegar al lugar donde se relocaliza para un nuevo trazado de ruta cuyo punto inicial es el obtenido por optimización y el punto final mantiene el establecido inicialmente.

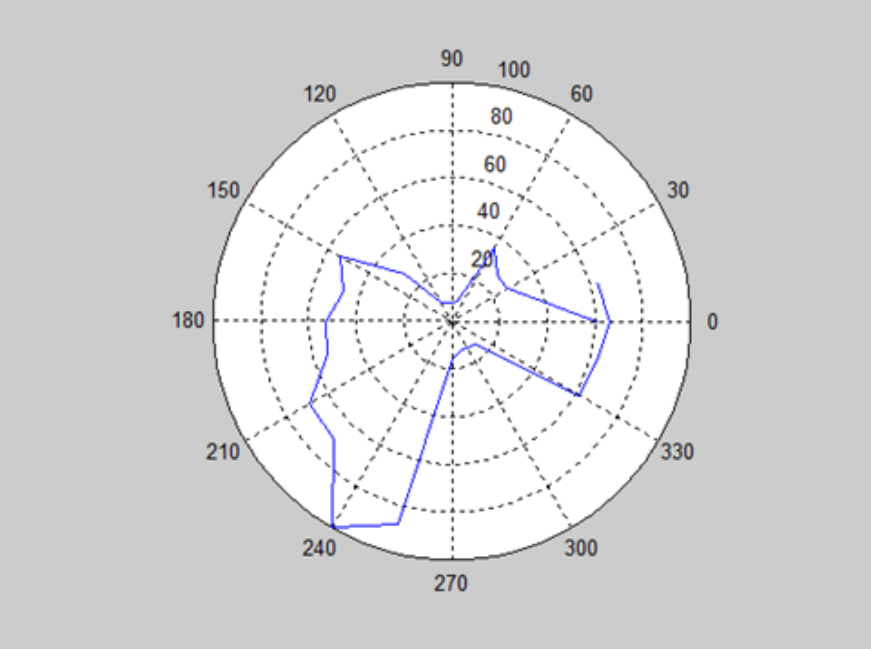


Figura 59. Diagrama polar previo a la relocalización (ruta 2)

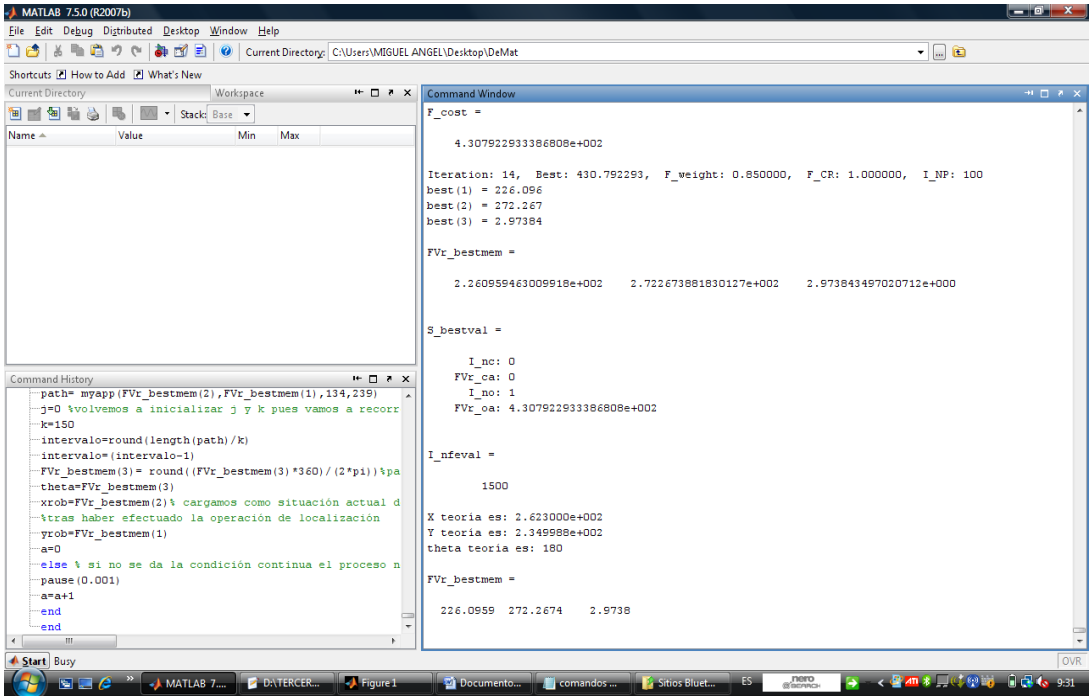


Figura 60. Datos sobre la mejor "pose" tras la optimización (ruta2)

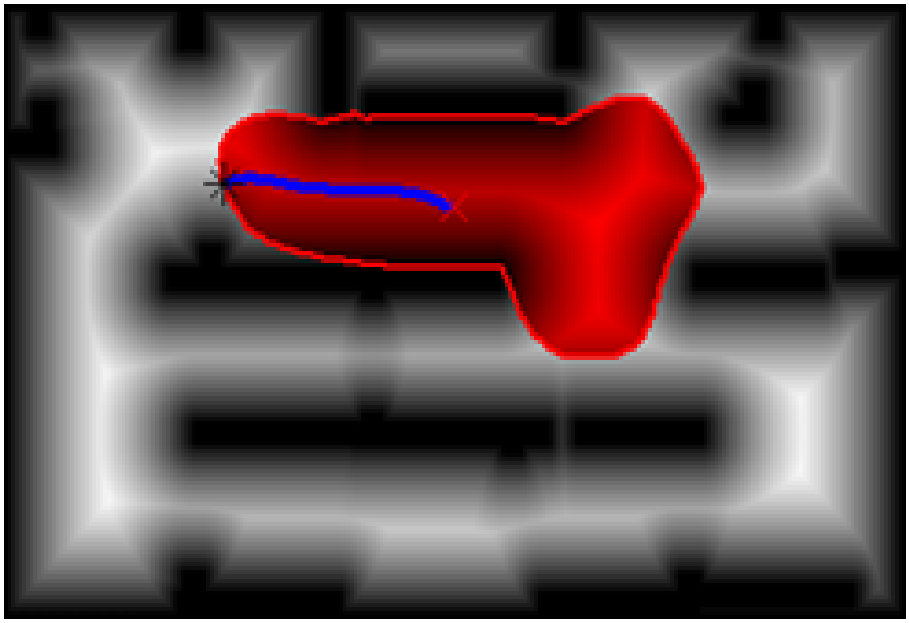
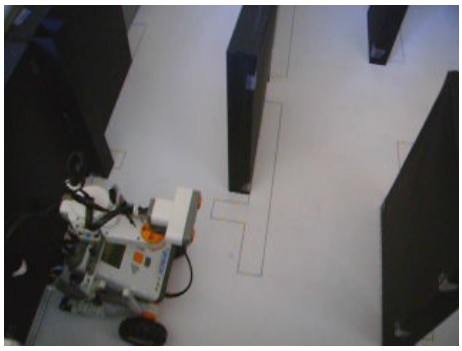
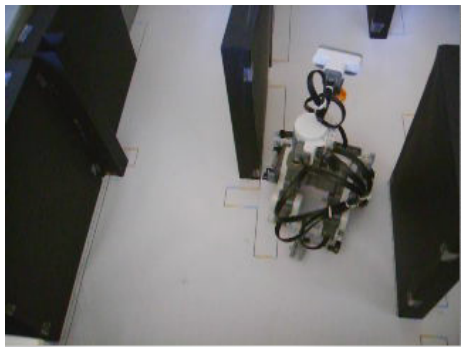
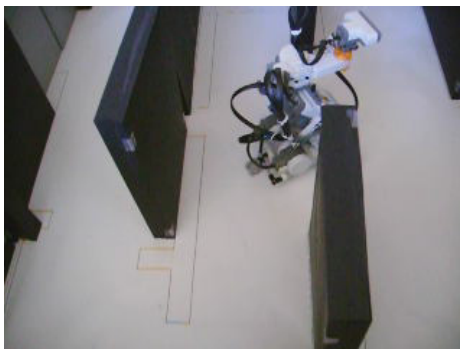


Figura 61. Nueva ruta tras la relocalización (ruta 2)

A continuación se muestra una secuencia de fotogramas que muestran el seguimiento de la ruta por el robot móvil:






RUTA 3.

Origen: (59,59)

Destino: (392,293)

Orientación inicial: 60º

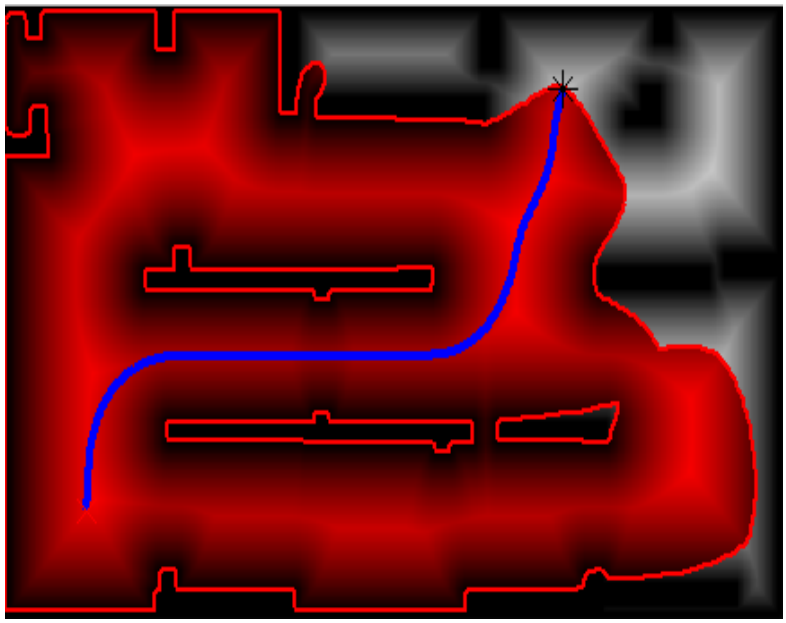


Figura 62: Trazado de ruta inicial (ruta 3)

-Al igual que en las anteriores pruebas, el robot móvil comienza a seguir la ruta empleando únicamente Odometría hasta llegar al lugar donde se relocaliza para un nuevo trazado de ruta cuyo punto inicial es el obtenido por optimización y el punto final mantiene el establecido inicialmente.

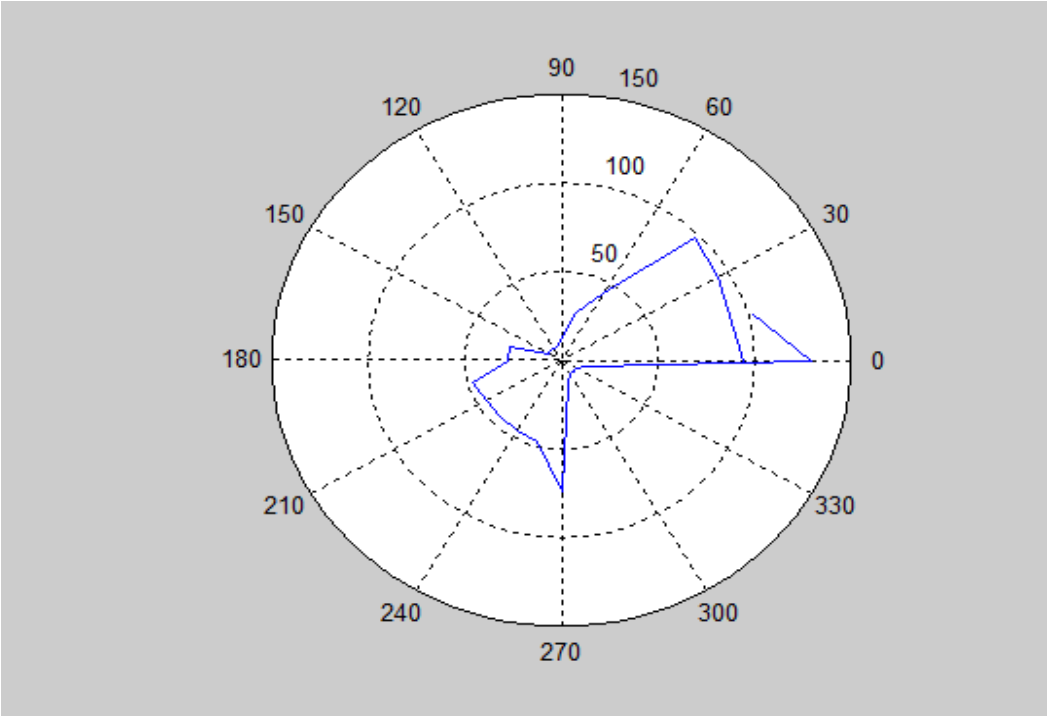


Figura 63. Diagrama polar previo a la relocalización (ruta 3)

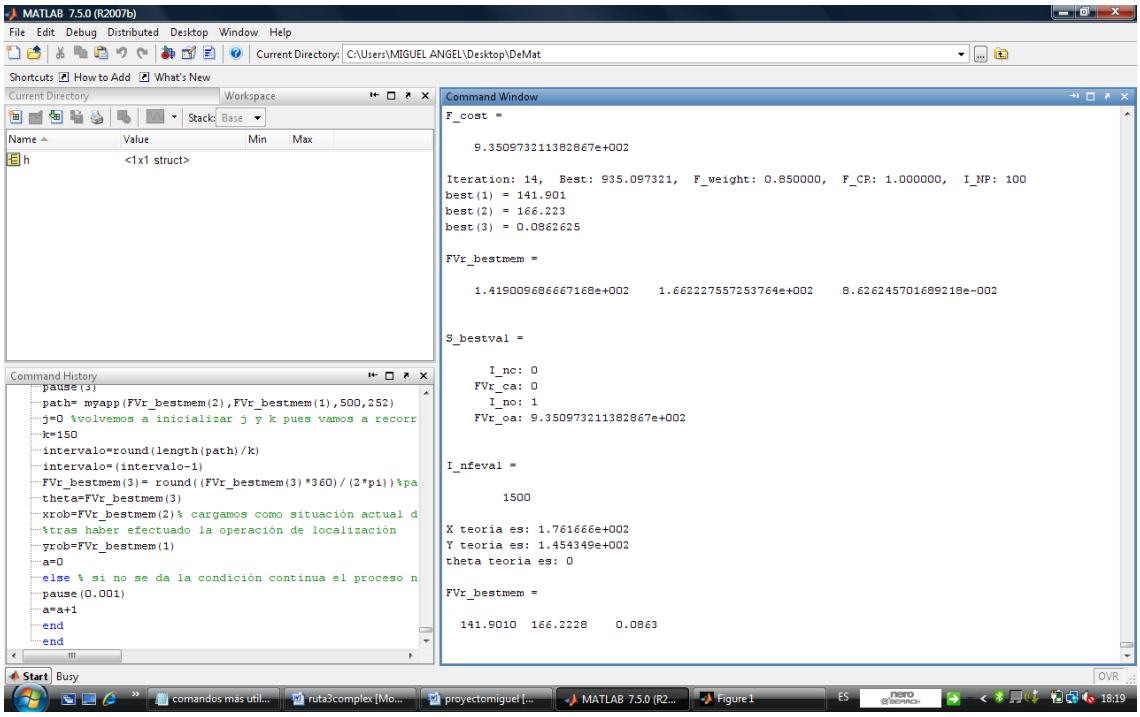


Figura 64. Datos sobre la mejor "pose" tras la optimización (ruta 3)

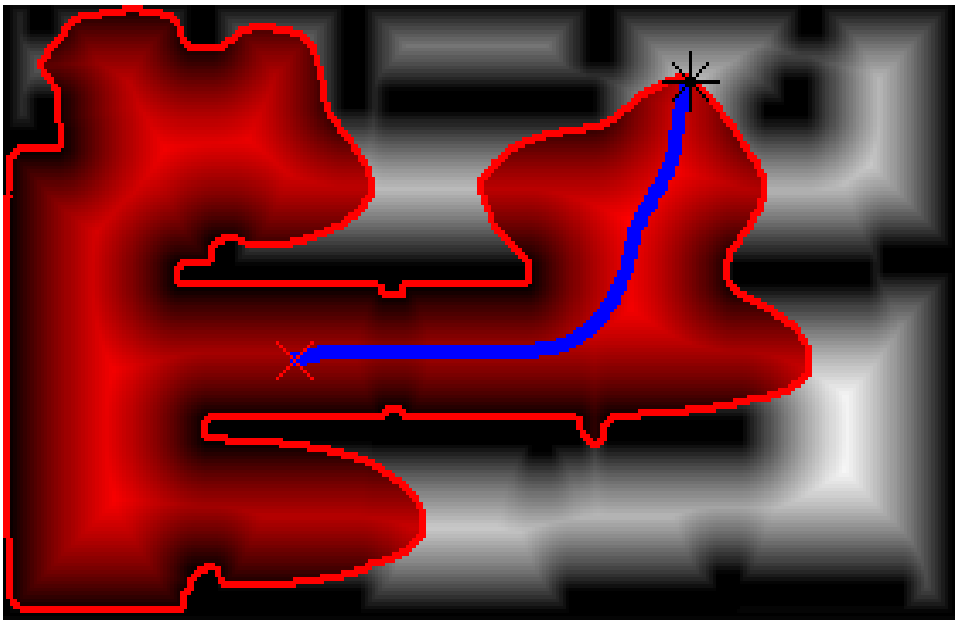
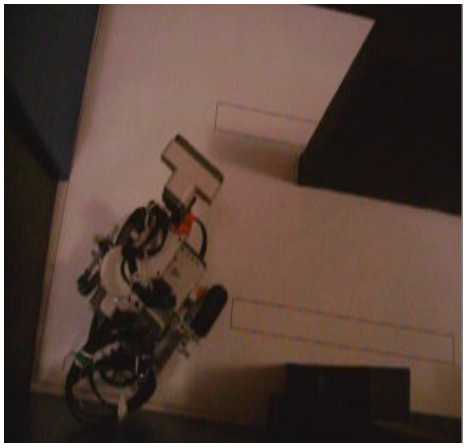
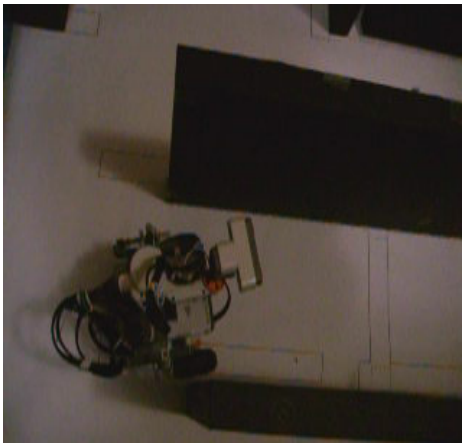


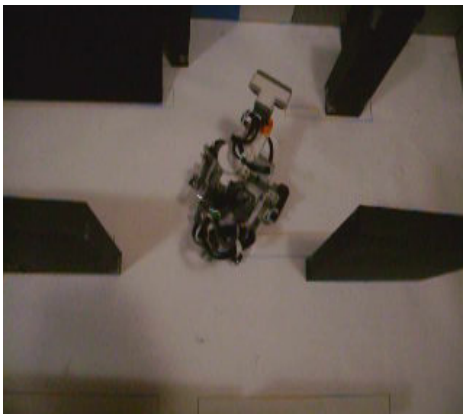
Figura 65. Nueva ruta tras la relocalización (ruta 3)

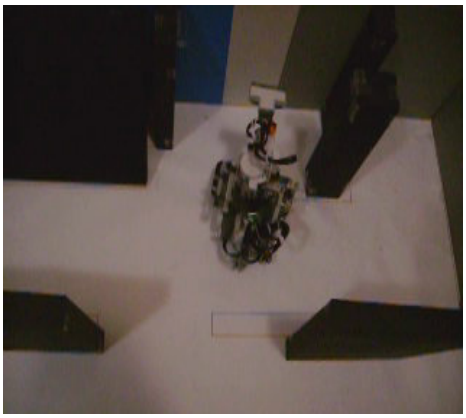
A continuación se muestra una secuencia de fotogramas que muestran el seguimiento de la ruta por el robot móvil:







5. APORTACIONES

Tras las pruebas y ensayos realizados podemos asumir como aportaciones realizadas:

Aplicación del algoritmo Voronoi Fast Marching (VFM) para el trazado de rutas óptimas en robots móviles. En nuestro caso ha sido utilizado sobre un robot Lego NXT Mindstorm.

Se consigue la generación de una ruta suave y segura que guía al robot desde el origen hasta el destino teniendo en cuenta los obstáculos presentes en el entorno.

Mediante el seguimiento de la ruta trazada, el robot no requiere del uso de sensores instalados en el propio robot o situados en el propio entorno para la evasión de obstáculos y la determinación de su posición.

Implementación del algoritmo “Differential Evolution” para la corrección de las desviaciones de ruta que conlleva el uso de un sistema basado en la Odometría (utilizando únicamente sensores integrados en el propio robot móvil).

Se observa una mejora en la precisión de los datos obtenidos en comparación con los de Odometría con respecto a la posición ocupada por el robot o “pose”. Ello junto con el sistema de planificación permite modificar el trazado original de la ruta para evitar que la desviación saque al robot totalmente de la ruta planificada inicialmente.

6. CONCLUSIONES Y TRABAJOS FUTUROS

Se consigue una mejora de la localización del robot, el sistema es fácil de implementar en cualquier tipo de robots móviles incluyendo robots tan sencillos como el Lego NXT Mindstorm y el esfuerzo computacional es menor que en otros sistemas basados en la división entre módulo global y local.

Por otra parte, se ha observado que las mediciones de distancia perimetral realizadas por el sensor de ultrasonidos que forman parte de uno de los argumentos que requiere la función de optimización no resultan del todo exactas. Esta falta de exactitud se debe a dos factores principales:

* Precisión del sensor de ultrasonidos que es de +/- 3 cm

* Fenómeno de reflexiones múltiples

En cuanto a la precisión del sensor de ultrasonidos poco más podemos hacer pues este tipo de sensores no tienen una gran precisión si los comparamos con otros tipos de sensores para la medición de distancias.

El fenómeno de reflexiones múltiples sucede cuando la onda emitida por el sensor se refleja en diversas superficies antes de retornar por lo que nos puede dar una medida errónea.

La observación del fenómeno de reflexiones múltiples se ha podido comprobar durante los ensayos, dado que cuando el ángulo que forma el eje longitudinal del sensor de ultrasonidos con respecto a la superficie reflectora es inferior a 45º (ángulos <45) las medidas de distancia aportadas difieren enormemente con respecto a la realidad, e incluso pueden llegar a indicar que la medida es la de máximo rango del sensor (255cm). Esto no es muy apropiado a la hora de relocalizarse pues estamos pasando un dato a la función de Relocalización que no es real y puede llegar a influir en el resultado.

Para solucionar los inconvenientes de la falta de precisión del sensor de ultrasonidos y el fenómeno de reflexiones múltiples se podría incorporar un sensor de tipo láser con una precisión infinitamente mejor (del orden de mm).

El uso de un sensor láser para la medición de distancias implica además de una considerable mayor precisión de las medidas, el no verse afectado por el fenómeno de las reflexiones múltiples.

Se propone así mismo como trabajo futuro la implementación de la metodología desarrollada en robots más sofisticados.

7. PRESUPUESTO

CONCEPTO	UNIDADES	PRECIO UNITARIO	TOTAL
Kit Lego Mindstorm NXT v 1.0	1	189€	189€
Papel continuo	1	--	2€
Cartulina	20	0,3€	6€
Pilas recargables	8	3,5€	28€
Total presupuesto	--	--	**225€**

REFERENCIAS

[1] (Los robots ya existían en la antigüedad)
http://www.taringa.net/posts/ciencia-educacion/5711486/Los-robots_-Ya-existian-en-la-Antiguedad_.html

[2](El telar automatizado. Joseph Marie Jacquard)
http://es.wikilingue.com/pt/Joseph-Marie_Jacquard

[3] (Definición moderna de robot. Karel Capek)
http://es.wikipedia.org/wiki/Karel_%C4%8Capek

[4] (Término robótica y las tres leyes de la robótica. Isaac Asimov)

http://www.laflecha.net/perfiles/ciencia/isaac_asimov

[5] (El robot PathFinder. Aldo Mateos Martínez)
http://www.roboticspot.com/spot/artic.shtml?todo=&block=12&newspage=robots

[6] (El robot microrover Sojourner, primer robot enviado a Marte. Guillermo Bataller López)

http://www.upv.es/satelite/trabajos/pracGrupo15/Marte/PF/microro.html

[7] (vehículo de exploración lunar “Rover Lunar”)

http://translate.google.es/translate?hl=es&sl=en&u=http://en.wikipedia.org/wiki/Lunar_rover_(Apollo)&ei=7ITVTKCsCsiOjAemqPjVCQ&sa=X&oi=translate&ct=result&resnum=6&ved=0CDEQ7gEwBQ&prev=/search%3Fq%3DLUNAR%2BROVER%26hl%3Des%26prmd%3Div

[8] (La Cypher, Vehículo no tripulado para vigilancia militar)

http://translate.google.es/translate?hl=es&sl=en&u=http://i-heart-robots.blogspot.com/2006/04/unmanned-aerial-vehicle-cypher.html&ei=iIXVTOfTKNW7jAf3i43ACQ&sa=X&oi=translate&ct=result&resnum=1&ved=0CBkQ7gEwAA&prev=/search%3Fq%3DROBOT%2BCYPHER%26hl%3Des%26prmd%3Dv

[9] (Ventajas de los sistemas robotizados)

http://www.cypsela.es/especiales/pdf209/robotica_automat.pdf

[10] Victor J. González Villea & Robert M. Parkin. Evadiendo obstáculos con robots móviles. *Revista Digital Universitaria UNAM*, 6 (1): 4-5, 2005

[11] Santiago Garrido Bullón. *Robot planning and exploration using Fast Marching*. Tesis de Máster, Universidad Carlos III de Madrid, 2008.

[12] Iwan Ulrich & Johann Borenstein. VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation (ICRA),* 1998.

[13] Alberto Poncela González. *Exploración completa basada en comportamientos cooperativos para un agente autónomo móvil*. Tesis doctoral, E.T.S de Ingeniería de Telecomunicación de la Universidad de Málaga, 2008.

[14] Steven M. La Valle. *Planning Algorithms*. Cambridge University Press, 2006

[15] (Información técnica sobre encoders)

http://www.forosdeelectronica.com/f16/encoders-informacion-tecnica-25/

[16] (El acelerómetro)

http://es.wikipedia.org/wiki/Aceler%C3%B3metro

[17] (El giroscopio)
http://www.cienciapopular.com/n/Experimentos/Giroscopios/Giroscopios.php

[18] (Manual para el uso de MatLab. Javier García de Jalón)

http://mat21.etsii.upm.es/ayudainf/aprendainf/Matlab70/matlab70primero.pdf

[19] S. Garrido, L. Moreno, D. Blanco & M.L. Munoz. Sensor-based global planning for mobile robot navigation. *Robotica*, 25 (2): 189-199, feb 2007

[20] Kenneth V. Price , Rainer M. Storn & Jouni A. Lampinen. *Differential Evolution – A Practical Approach to Global Optimization*. Springer-Verlag Berlin Heidelberg, 2005

APENDICES

APENDICE A

FUNCION CALCULOCOORD

Como puede apreciarse esta función tiene cuatro argumentos de entrada y dos argumentos de salida:

(xrut,yrut): estas coordenadas pertenecen a un punto de la ruta que se obtiene mediante el comando path(length(path)-150,:) incluido dentro de la función 'myapp', encargada de generar el trazado de ruta según la metodología Fast Marching. En este ejemplo después de haber lanzado dicha función nos dará las coordenadas X,Y del punto nº 150 de la ruta.

(xrob,yrob): Estas coordenadas corresponden la posición del robot móvil dentro del laberinto. Si no se dispone de sistema de relocalización no podemos saber con exactitud si el robot se encuentra en la posición en la que creemos que está.

(angulo): en este argumento de salida, se nos devuelve el ángulo que con respecto al eje 'X' presenta la línea imaginaria que une el punto donde creemos que se encuentra el robot con un punto determinado de la ruta a donde queremos que el robot llegue tras un movimiento de traslación.

(distpuntos): Aquí se nos devuelve la distancia existente entre los dos puntos anteriores. Esta es la distancia que deberá recorrer (en línea recta) el robot móvil durante el movimiento de traslación después de efectuado el giro de orientación.

Para la obtención de los dos parámetros de salida indicados la función realiza los siguientes pasos:

Primeramente se determina las distancias X e Y con respecto al origen de coordenadas de cada uno de los puntos (Xrut,yrut) y (Xrob,yrob). Lo que se hace es, conociendo las coordenadas en pixeles de cada uno de los puntos y la equivalencia de pixeles a cm de los límites del mapa o entorno (alto y ancho) obtener la distancias X e Y expresadas en cm.

Según se muestra en la figura 66, una vez obtenidas las cuatro distancias las restamos por parejas ((dxruta-dxrobot) y (dyruta-dyrobot)) para obtener los lados o catetos del triángulo formado por la unión del punto donde se encuentra el robot y el punto de la ruta a donde queremos que vaya.

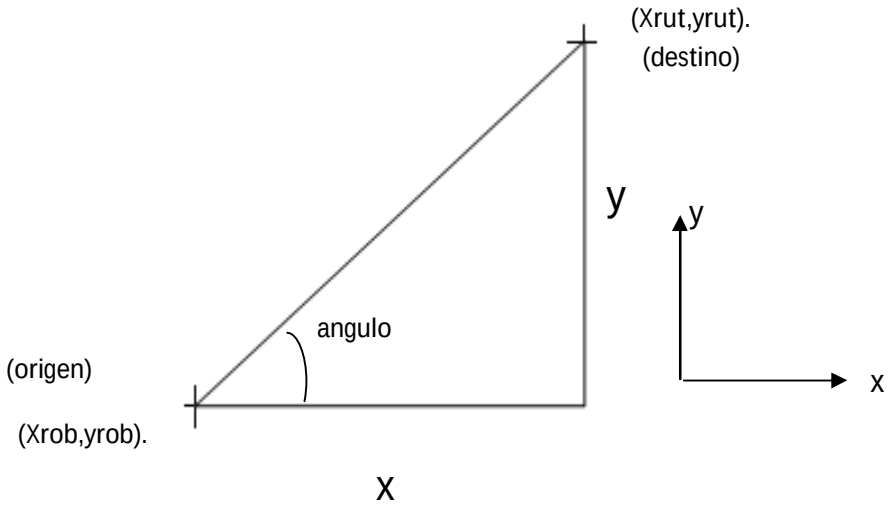


Figura 66. Triángulo formado entre los puntos (Xrob,yrob) y (Xrut,yrut).

A partir de la obtención de los catetos ya podemos determinar 'distpuntos' y ' angulo'. Para 'distpuntos' aplicamos trigonometría, sabiendo que 'distpuntos' es equivalente a la hipotenusa de dicho triángulo hacemos la raíz cuadrada de la suma al cuadro de los dos catetos.

Para 'angulo' aplicamos la función trigonométrica de Matlab "atan2" que nos da el valor de 'angulo' expresado en radianes en lugar de grados.

La sintaxis de la función atan2 es: atan2(Y,X).

Para pasar a grados se hace una sencilla conversión.

Ahora ya finalmente está todo preparado para retornar los argumentos 'angulo' y 'distpuntos'. Estos dos argumentos van a ser utilizados como argumento de entrada en la siguiente función "girocuadrante".

FUNCION GIROCUADRANTE

Como puede observarse esta función tiene 8 argumentos de entrada y no genera ningún argumento de salida.

-Los argumentos de entrada 'angulo' y 'distpuntos' corresponden a los dos argumentos de salida generados por la función "calculocoord".

Theta: corresponde al ángulo que presenta el robot con respecto al eje X (xrut,yrut) e (xrob,yrob): corresponden al igual que en la función anterior a la posición ocupada por el robot y a un punto determinado de la ruta.

j: este argumento corresponde a una variable que hace las veces de contador. Va incrementándose a medida que se van ejecutando los distintos avances o tramos en los que se ha dividido la ruta planificada.

Como se ha comentado anteriormente esta función realiza dos operaciones bien definidas: el **giro** para la orientación hacia el punto de la ruta a donde queremos llevar el robot móvil y el otro el **avance** en línea recta hacia dicho punto. El avance en línea recta lo conseguimos mediante el funcionamiento síncrono de los dos motores del eje delantero.

Antes de explicar cómo se realizan los giros, hay que tener en cuenta que los ángulos medidos se hacen con respecto al eje X y en sentido antihorario.

Al realizar un giro podemos encontrarnos con dos situaciones básicas:

a) Que el ángulo 'Theta' que presenta la orientación del robot con respecto al eje X sea mayor que el argumento 'angulo' (theta>angulo), en cuyo caso nos interesa realizar un giro a derechas para llevar a cabo la orientación hacia el punto de la ruta.

b) Que el ángulo ' Theta' sea menor que el argumento 'angulo' (theta<angulo), en este caso nos interesa hacer un giro a izquierdas para conseguir la orientación hacia el punto de la ruta.

-Si no se da ninguno de estos dos casos el robot circulará en línea recta.

El siguiente paso a realizar por esta función sería conseguir el avance hasta el punto de la ruta indicado, para ello aplicamos un pequeño tiempo de pausa para garantizar que ha terminado de realizarse el giro de orientación, después tenemos que convertir la distancia almacenada en la variable 'distpuntos' a valores reales del plano o entorno original, para ello multiplicamos dos veces por 10. La primera para pasar la distancia que se encuentra almacenada en cm a mm y la segunda para pasar a valores de escala real (se determinó una escala en el plano de 1:10).

Una vez tenemos la distancia pasada a valores reales en mm, hemos de traducir esta distancia lineal a número de grados que deberán girar las ruedas (estos grados serán el valor que tomará el atributo 'Tacholimit' durante el movimiento síncrono de los servomotores).

PROGRAMA PRINCIPAL

Lo primero que tenemos que hacer es introducir los datos de la posición inicial del robot móvil en el plano del laberinto y las coordenadas (x,y) del destino, estos valores son la terna (theta,xrob,yrob) e (xrut,yrut). Se trata de indicar desde donde parte el robot para comenzar a seguir la ruta trazada.

El otro paso previo que tenemos que seguir es inicializar el contador de los avances que va haciendo el robot que en nuestro caso hemos denominado 'j'.

Seguidamente inicializamos la variable 'intervalos', en esta variable almacenaremos la cantidad de intervalos o avances en los que vamos a dividir la ruta a seguir y debido a esta división tendrá forma poligonal.

Para obtener el número de intervalos o avances dividimos el número total de puntos de que consta la ruta entre una variable denominada 'k' que contiene un número que nosotros damos en función de la cantidad de intervalos o avances en que deseamos se descomponga nuestra ruta.

Después de estos pasos nos introducimos en un bucle "While" que se ejecutará tantas veces como número de intervalos se tenga dividida la ruta.

Dentro de este bucle mediante el siguiente comando de Matlab path(length(path)-k,:) obtenemos las coordenadas (xrut,yrut) de un punto de la ruta situado 'k' distancia desde el origen de ésta. Cada vez que volvamos a entrar en este bucle 'k' se incrementará en 150 unidades con lo que conseguiremos las coordenadas de un nuevo punto de la ruta situado a 'k' puntos del último valor dado y así hasta completar totalmente la ruta.

Seguidamente guardamos las coordenadas obtenidas en las variables (xrut,yrut) .

Una vez tenemos las coordenadas del punto de la ruta hacia donde queremos ir y las coordenadas iniciales donde se encuentra el robot, lanzamos la función "calculocoord" para obtener mediante los argumentos de retorno 'angulo' y 'distpuntos', el ángulo que con respecto al eje X forma la recta que une el punto donde se encuentra el robot y el punto de la ruta hacia donde queremos ir y la distancia que deberá recorrer el robot hasta llegar a dicho punto.

Seguidamente se ejecuta la función "girocuadrante" que efectuará el giro de orientación y el avance necesarios para llevar a cabo uno de los intervalos en los que se encuentra dividida la ruta.

Si no contamos con un sistema de Relocalización, es decir funcionando únicamente mediante Odometría, una vez terminado el proceso de traslación del robot móvil hasta el punto de la ruta que hemos determinado no podemos conocer con exactitud si el robot móvil ha llegado exactamente hasta el lugar deseado.

Por ello utilizando únicamente Odometría, antes de volver al principio del bucle "while" lo que hacemos es la siguiente operación de asignación:

- theta=angulo ; xrob=xrut; yrob=yrut

Estamos suponiendo (si bien en la realidad no tiene porque ser así bajo un funcionamiento odométrico) que el robot móvil ha llegado hasta el punto de la ruta convenido con el mismo ángulo que el presentado por la unión de los puntos de partida del robot y el punto de llegada dentro de la ruta, así mismo estamos también suponiendo que el robot ha llegado justo hasta el punto de la ruta por lo que las nuevas coordenadas de partida para el siguiente intervalo van a ser justamente las del punto de la ruta donde teóricamente debería de encontrarse.

Antes de volver a ejecutar el bucle se ha procedido a incrementar las variables 'k' y 'j'.

Este bucle continuará ejecutándose continuadamente hasta que deje de cumplirse la condición de ejecución.

FUNCION Ultra4mcl

La función comienza abriendo el mundo exterior al sensor de ultrasonidos mediante la instrucción `'OpenUltrasonic',` seguidamente preparamos el servomotor que va hacer posible que el sensor de ultrasonidos realice medidas a lo largo de un giro completo o 360º.

Se determinó la realización de un total de 24 medidas a razón de 15º cada una de ellas, siendo realizada la primera medida con el servomotor en reposo.

Dado que se deberán de realizar incrementos de giro de 15º , la propiedad `'TachoLimit'` ha sido puesta a valor 15.

Seguidamente se crea una matriz de ceros de 24 fil x 2 Col , en la primera columna se recogerá el dato referente al valor en grados donde se realizan cada una de las medidas y en la segunda columna quedarán los valores propios de las medidas expresados estos en cm y con el ya mencionado margen de precisión de +/- 3º que conlleva el uso de nuestro sensor de ultrasonidos.

Mediante el bucle 'while' de esta función hemos podido visualizar en la ventana de Matlab "comand Windows" en tiempo real como el sensor de ultrasonidos va completando el giro de 360º a intervalos de 15º.

Una vez se ha completado el giro, se muestra por pantalla un diagrama polar que nos muestra gráficamente las medidas realizadas durante todo el proceso. Seguidamente el servomotor realiza un giro completo de -360º con el propósito de retornar el sensor de ultrasonidos a la posición inicial de partida.

Finalizamos el proceso de medición cerrando al sensor de ultrasonidos el acceso al mundo exterior mediante el comando `'CloseSensor'`

Para terminar, creamos un vector denominado 'v0' que formará parte de los argumentos de entrada para la función que nos dará la localización de nuestro robot con la mayor precisión posible (optimización).

v0 es un vector de 1 fil x 24 col por lo que hemos de extraer la información que nos interesa de la matriz que es concretamente la almacenada en la 2ª columna.

Dado que la función 'RunLoca' no admite las medidas en cm sino en pixel se ha tenido que realizar una sencilla transformación

FUNCION RUNLOCA

-Su argumento de salida es:

FVr_bestmem= RunLoca(v0,theta,xrob,yrob)

Esta función dispone de cuatro argumentos de entrada:

v0: es el vector de 1 fil x 24 col que contiene las mediciones realizadas por el sensor de ultrasonidos a intervalos de 15º. Las medidas se pasan en pixeles en lugar de cm.

Theta, xrob e yrob: son los datos de la última situación o "pose" obtenidos por el sistema de Odometría justo antes de proceder a la relocalización

La función dispone de un argumento de salida:

FVr_bestmem: se trata de un vector (1fil x 3 col) que contiene la mejor "pose" que se obtiene una vez ha convergido el algoritmo "Differential Evolution" contenido dentro de RunLoca.

Una vez tenemos el vector correspondiente a las mediciones realizadas mediante el sensor de ultrasonidos v0 (correspondiente a lo que "ve el robot"), necesitamos otro vector de igualmente otras 24 posiciones, para obtener este otro vector se dispone de otra función denominada "dist_est que nos da los 24 valores que se medirían con el sensor de ultrasonidos si el robot estuviera sobre un punto determinado.

El error cuadrático sería la raíz cuadrada de la suma al cuadrado de cada una de las 24 diferencias entre la medición estimada y la realizada por el sensor de ultrasonidos. El objetivo del algoritmo “Differential Evolution” es minimizar este error.

APENDICE B

FUNCION CALCULOCOORD

```
function [angulo,distpuntos]=calculocoord(xrut,yrut,xrob,yrob)

%en esta función utilizamos  cuatro argumentos de entrada y dos de
salida
%introducimos las coordenadas de un punto dentro de la ruta indicada
por el planificador (xrut,yrut) y las coordenadas del punto donde
pensamos que se encuentra el robot (xrob,yrob)
%la información que nos devuelve, es el ángulo que forma el punto de
la ruta con respecto al eje X y la distancia que tendrá que recorrer
el robot para llegar hasta el punto de la ruta partiendo desde donde
se encuentra.
clc;

dxruta=((xrut*14.32)/(544));
dyruta=((yrut*8.85)/(337));
dxrobot=((xrob*14.32)/(544));
dyrobot=((yrob*8.85)/(337));
%aquí determinamos las distancias (x,y) de cada uno de los puntos con
%respecto al origen de coordenadas del plano, estas distancias van en
cm.

distentrpuntx=(dxruta-dxrobot);
distentrpunty=(dyruta-dyrobot);

%aquí hemos determinado los dos lados (los catetos) del triángulo que
se forma entre los puntos de origen y destino

distpuntos=sqrt((dxruta-dxrobot)^2+(dyruta-dyrobot)^2);
%aquí se determina la distancia en línea recta entre los dos puntos
que será la distancia que tendrá que avanzar el robot para llegar
hasta el punto incluido dentro de la ruta planificada.

disp(sprintf('distpuntos es: %d % complete/n',distpuntos));
x=distentrpuntx;
y=distentrpunty;

d=atan2(y,x);

angulo=(d*360)/(2*pi);%aquí se determina el ángulo del triangulo
formado por los dos catetos del triángulo.
disp(sprintf('el angulo en grados del punto de la ruta con respecto al
eje X es'));
angulo=round(angulo);
```

```
 if (angulo<0)%modificación para transformar ángulos negativos en
positivos (sentido antihorario)
     angulo=angulo+360;
 else
 pause(0.001)
 end
disp(sprintf('el angulo es: d % complete/n',angulo));



end  )%aquí finaliza la función
```

FUNCION GIROCUADRANTE

```
function girocuadrante(angulo,theta,distpuntos,xrut,yrut,xrob,yrob,j)

%esta función mediante la introducción de los datos generados por la
%función calculocoord y del angulo "theta" que presenta el robot con
%respecto al eje X, nos permite por una parte orientar al robot en la
%dirección del punto de la ruta planificada al que quiere dirigirse y
por otra el avance del robot hasta llegar a dicho punto.
%los argumentos de entrada a esta función son:
% angulo: ángulo con respecto al eje X que presenta el punto de la
ruta al que quiere dirigirse el robot
% distpuntos: distancia entre el punto de la ruta y el punto donde se
% encuentra situado el robot antes de realizar la maniobra de
aproximación
%theta: angulo que presenta el robot con
%respecto al eje X

clc;

    if ((theta>angulo & yrob<yrut & j~=0)|(theta>angulo & xrob<xrut &
j~=0))
       disp(sprintf('ahora giramos a la derecha/n'));
       mayortheta=((360-angulo)-(360-theta));
       mayortheta=(mayortheta*2);%para girar un número determinado de
grados, hemos de poner en 'TachoLimit' el doble de los grados que
deseamos girar.
       disp(sprintf('el valor de mayortheta es: %d %
complete/n',mayortheta));
      motorA = NXTMotor('A', 'Power',
15,'TachoLimit',mayortheta);%aqui generamos el giro para orientarnos
hacia el punto de la ruta
      motorB = NXTMotor('B', 'Power', -15,'TachoLimit',mayortheta);
      motorA.SendToNXT();
      motorB.SendToNXT();
      %para generar un giro del robot hacemos que las ruedas giren en
sentido contrario cambiando el signo de “TachoLimit”.

    elseif (theta<angulo & j~=0)%estas instrucciones se ejecutan si el
punto de la ruta presenta un angulo superior al del robot (giro a
izquierdas)
```

```
        disp(sprintf('ahora giramos a la izquierda complete/n'));
        mayoralpha=((360-theta)-(360-angulo));
        mayoralpha=(mayoralpha*2);
        disp(sprintf('el valor de mayorapha6 es: %d %
complete/n',mayoralpha));
        motorA = NXTMotor('A', 'Power', -
15,'TachoLimit',mayoralpha);%aqui generamos el giro para orientarnos
hacia el punto de la ruta
        motorB = NXTMotor('B', 'Power', 15,'TachoLimit',mayoralpha);
        motorA.SendToNXT();
        motorB.SendToNXT();


     elseif (j==0 & yrob>yrut & theta>angulo)

         mayoral= (theta-angulo);
         mayoral=(mayoral*2);
         disp(sprintf('el valor de mayoral3 es: %d %
complete/n',mayoral));
         motorA = NXTMotor('A', 'Power', 15,'TachoLimit',mayoral);%aqui
generamos el giro para orientarnos hacia el punto de la ruta
         motorB = NXTMotor('B', 'Power', -15,'TachoLimit',mayoral);
         motorA.SendToNXT();
         motorB.SendToNXT();

     elseif (j==0 & yrob>yrut & theta<angulo)

         mayoral= (angulo-theta);
         mayoral=(mayoral*2);
         disp(sprintf('el valor de mayoral es: %d %
complete/n',mayoral));
         motorA = NXTMotor('A', 'Power', -
15,'TachoLimit',mayoral);%aqui generamos el giro para orientarnos
hacia el punto de la ruta
         motorB = NXTMotor('B', 'Power', 15,'TachoLimit',mayoral);
         motorA.SendToNXT();
         motorB.SendToNXT();

     elseif (j==0 & yrob<yrut & theta<angulo)
         mayorall= (angulo-theta);
         mayorall=(mayorall*2);
         disp(sprintf('el valor de mayorall es: %d %
complete/n',mayorall));
         motorA = NXTMotor('A', 'Power', -
15,'TachoLimit',mayorall);%aqui generamos el giro para orientarnos
hacia el punto de la ruta
         motorB = NXTMotor('B', 'Power', 15,'TachoLimit',mayorall);
         motorA.SendToNXT();
         motorB.SendToNXT();

     elseif (j==0 & yrob<yrut & theta>angulo)
         mayorall= (theta-angulo);
         mayorall=(mayorall*2);
         disp(sprintf('el valor de mayorall7 es: %d %
complete/n',mayorall));
         motorA = NXTMotor('A', 'Power',
15,'TachoLimit',mayorall);%aqui generamos el giro para orientarnos
hacia el punto de la ruta
```

```
        motorB = NXTMotor('B', 'Power', -15,'TachoLimit',mayorall);
        motorA.SendToNXT();
        motorB.SendToNXT();

     elseif (j==0 & yrob==yrut)
        distpunt=round(distpunt*100);
%En esta circunstancia el robot no realizará giro sino que avanzará en
línea recta
        avanc=round((distpunt*360)/(2*pi*28));%pasamos la distancia
que tiene que avanzar a grados

        motorAB = NXTMotor('AB', 'Power', 15,'TachoLimit',avanc);%aquí
avanzamos con funcionamiento síncrono hacia el punto de la ruta
        motorAB.SendToNXT();



   end

   pause(2);

  distpuntos=round(distpuntos*100);%distancia que deberá avanzar el
robot en mm
    %x10 para pasar la distancia en cm y nuevamente x10 para pasar la
  %distancia a la realidad del plano
  avance=round((distpuntos*360)/(2*pi*28));%pasamos la distancia que
tiene que avanzar a grados

  motorAB = NXTMotor('AB', 'Power', 15,'TachoLimit',avance);%aquí
avanzamos con funcionamiento síncrono hacia el punto de la ruta
  motorAB.SendToNXT();
     end
```

PROGRAMA PRINCIPAL PARA EL SEGUIMIENTO DE RUTA PLANIFICADA MEDIANTE FAST_MARCHING

```
path= myapp(363,164,200,235)%lanzamos inicialmente el algoritmo
planificador mediante la ejecución de la función “myapp”
%seguidamente elegimos un punto del plano donde decidimos que se
encuentra situado el robot
theta= 180 %orientación inicial del robot con respecto al eje X.
xrob=363
yrob=164
j=0 %inicialización de variable
a=0 %inicialización de variable

%inicializamos el contador de intervalos recorridos por la ruta (j),
esto provocará que el seguimiento de la
%ruta tenga forma poligonal.

intervalo=0 %inicializamos esta variable que se corresponde con el
número de intervalos en los que se ha
%dividido la ruta teniendo en cuenta el valor de k
k=150 %se inicializa esta variable para incrementar el número de los
puntos para el avance por la ruta.
intervalo=round(length(path)/k)
%operación para obtener el número de intervalos a realizar en la ruta
para ello dividimos “k” entre el número total de puntos de la ruta
intervalo=(intervalo-1)
%esto lo hacemos para indicar el límite de la cantidad de avances
%de estar mal calculado el número de avances nos pasaremos del punto
final provocando un mensaje de error
%por eso hacemos la resta anterior

%seguidamente encontramos el bucle principal que permite el
seguimiento de la ruta planificada

while(j<intervalo) %condición de funcionamiento "contador menor que el
límite de intervalos recorridos

 path(length(path)-k,:) %con este commando obtenemos la coordenadas de
un punto de la ruta situado a “k” distancia del origen

 xrut=ans(1,1) %para almacenar la coordenada X de un punto de la ruta
trazada por el planificador
 yrut=ans(1,2) %idem para la coordenada Y

 [angulo,distpuntos]=calculocoord(xrut,yrut,xrob,yrob)
 %con el lanzamiento de esta función conseguimos obtener los
argumentos necesarios para orientarnos hacia el punto de la ruta a
donde queremos ir y avanzar la distancia necesaria.
 disp(sprintf('angulo : %d % complete/n',angulo));

 girocuadrante(angulo,theta,distpuntos,xrut,yrut,xrob,yrob,j)
 k=k+150 % aquí incrementamos la variable que indica el número de
puntos de la ruta que tenemos que avanzar para realizar el siguiente
movimiento de giro-traslación.
 j=j+1 % aquí incrementamos el contador de los invervalos de avance
por ruta
```

```
 %las siguientes 3 instrucciones tienen la función de que cada vez que
se realice un nuevo avance, éste se realice de tal forma que las
coordenadas del robot se correspondan con el punto final de la ruta al
que suponemos que habrá llegado el robot (sin sistema corrector no
podemos hacer otra cosa). Por otra parte el angulo 'theta' hacemos que
se corresponda con el angulo al que tuvo que orientarse al principio
del avance realizado.

 theta=angulo
 xrob=xrut
 yrob=yrut

 pause(4);

  if((theta>179) & (a>6))
    Ultra4mcl
    pause(14);%lanzamos movimiento de torreta y esperamos a que
termine las mediciones de la periferia que rodea al robot
    FVr_bestmem= RunLoca(v0,theta,yrob,xrob)
% se nos devuelve la "pose" del mejor punto posible (la terna X,Y,θ)
    % en la función de relocalización RunLoca(v0,theta,yrob,xrob)
 %se hizo un cambio de yrob,xrob por xrob,yrob (pues teníamos el
problema que la función quedaba bloqueada porque algún punto de la
población de las sucesivas generaciones quedaba fuera de los límites
del mapa o entorno)
    pause(3)

%seguidamente volvemos a lanzar el planificador de ruta pero esta vez
utilizando como punto de origen el obtenido tras el proceso de
optimización

    path= myapp(FVr_bestmem(2),FVr_bestmem(1),200,235)
    j=0 %volvemos a inicializar j y k pues vamos a recorrer una nueva
ruta trazada tras la relocalización pero con el mismo punto final que
el elegido al principio del lanzamiento del programa principal.
    k=150
    intervalo=round(length(path)/k)
    intervalo=(intervalo-1)
    FVr_bestmem(3)= round((FVr_bestmem(3)*360)/(2*pi))%pasamos de
radianes a grados
    theta=FVr_bestmem(3) %asignamos a theta el angulo optimizado
    xrob=FVr_bestmem(2)% seguidamente cargamos como situación actual
del robot los datos obtenidos tras la optimización
    yrob=FVr_bestmem(1)
    a=0
  else % si no se da la condición continua el proceso normal
   pause(0.001)
   a=a+1
  end

end


% las siguientes líneas de comando se han utilizado para realizar una
ruta larga en dos tramos en lugar de uno. Todos los comandos se han
utilizado igualmente que en las líneas anteriores.
```

```
pause(2.5)

round(xrob);
round(yrob);
path= myapp(xrob,yrob,59,59) %volvemos a lanzar la función de
planificación de ruta para el segundo tramo pero conservando como
punto final o destino el elegido al principio del programa principal.

% en este caso elegimos como coordenadas del robot las últimas dadas
por odometría al final del primer tramo de la ruta
theta= theta
xrob=xrob
yrob=yrob
j=0 %inicialización de variables
a=0

%inicializamos el contador de intervalos recorridos por la ruta, esto
provocará que el seguimiento de la
%ruta tenga forma poligonal.

intervalo=0 %inicializamos esta variable que se corresponde con el
número de intervalos en los que se ha
%dividido la ruta teniendo encuenta el valor de k

k=150 %se inicializa esta variable para incrementar el número de los
puntos para el avance por la ruta

intervalo=round(length(path)/k)
%operación para obtener el número de intervalos a realizar en la ruta

intervalo=(intervalo-1)

%esto lo hacemos para indicar el límite de la cantidad de avances
%de estar mal calculado el número de avances nos pasaremos del punto
final provocando un mensaje de error
%por eso hacemos la resta anterior


while(j<intervalo) %condición de funcionamiento "contador menor que el
límite de intervalos recorridos

 path(length(path)-k,:)

 xrut=ans(1,1) %para almacenar la coordenada X de un punto de la ruta
trazada por el planificador
 yrut=ans(1,2) %idem para la coordenada Y

 [angulo,distpuntos]=calculocoord(xrut,yrut,xrob,yrob)
 %end
 disp(sprintf('angulo : %d % complete/n',angulo));
 girocuadrante(angulo,theta,distpuntos,xrut,yrut,xrob,yrob,j)
 k=k+150 % aquí incrementamos la variable que indica el número de
puntos
 j=j+1 % aquí incrementamos el contador de los invervalos de avance
por ruta
```

```
%las siguientes 3 instrucciones tienen la función de que cada vez que
se realice un nuevo avance, éste se realice de tal forma que las
coordenadas del robot se correspondan con el punto final de la ruta al
que suponemos que habrá llegado el robot (sin sistema corrector no
podemos hacer otra cosa). Por otra parte el angulo 'theta' hacemos que
se corresponda con el angulo al que tuvo que orientarse al principio
del avance realizado.

theta=angulo
 xrob=xrut
 yrob=yrut

 pause(4);


  if((theta>269) & (a>8))
    Ultra4mcl
    pause(14);%lanzamos movimiento de torreta y esperamos a que
termine las mediciones perimétricas
    FVr_bestmem= RunLoca(v0,theta,yrob,xrob)% se nos devuelve el mejor
punto posible (la terna)

    pause(3)

    path= myapp(FVr_bestmem(2),FVr_bestmem(1),59,59)
    j=0
    k=150
    intervalo=round(length(path)/k)
    intervalo=(intervalo-1)
    FVr_bestmem(3)= round((FVr_bestmem(3)*360)/(2*pi))%pasamos de
radianes a grados
    theta=FVr_bestmem(3)
    xrob=FVr_bestmem(2)
    yrob=FVr_bestmem(1)
    a=0
  else % si no se da la condición continua el proceso normal
   pause(0.001)
   a=a+1
  end

end
```

FUNCION Ultra4Mcl

```
OpenUltrasonic(SENSOR_1); %apertura del sensor de ultrasonidos al
mundo real
motorC = NXTMotor('C', 'Power', 15,'TachoLimit',15); %configuración
(setting) de características de motor torreta
motorC.ResetPosition(); %puesta a cero del contador del servomotor de
la torreta
incr=15;% variable par un barrido de 24 medidas

m=zeros(360/incr,2); %inicializa matriz datos
%la matriz contiene 24 fil x 2 col en la primera columna va el angulo
donde se encuentra el motor y en la segunda columna el resultado de la
medición con el sensor de ultrasonidos en cm
i=1;
incr=15;
m(i,1)=0;
a=GetUltrasonic(SENSOR_1); %orden para guardar en 'a' la lectura del
sensor ultrasonidos cuando la torreta está en posición inicial (cero
grados)
m(i,2)=a; %guardamos la medición inicial en la fil 1 columna segunda
(2)

motorC.SendToNXT(); %envío de las configuraciones del motor (setting)
al motor
pause(1);
StopMotor(MOTOR_C, 'brake');
[DATA2] = motorC.ReadFromNXT() %orden de lectura de datos del
servomotor de la torreta
%los datos presentan forma de una estructura de datos(incluyen varios
campos)

while (DATA2.Position<=347) %entrada en bucle para el barrido de 360º
(una vuelta completa)
   i=i+1;
   m(i,1)=i*incr;
   a=GetUltrasonic(SENSOR_1);
   m(i,2)=a;
   motorC.SendToNXT();
   motorC.WaitFor();
   StopMotor(MOTOR_C, 'brake');
   [DATA2] = motorC.ReadFromNXT()
end

m2=m;
m2(:,2)=[m(3:25,2); m(1:2,2)];
pause(0.1);
polar(m2(:,1)*pi/180,m2(:,2)); %obtención del mapa polar
disp('return start'); %se avisa de que la torreta retorna a su
posición inicial
motorC = NXTMotor('C', 'Power', -15,'TachoLimit',360); %invertimos
sentido de giro de motor para retornar a posición inicial
motorC.SendToNXT();
motorC.WaitFor();
StopMotor(MOTOR_C, 'brake');
```

```
pause(1);

StopMotor(MOTOR_C, 'off'); %dejamos el motor de la torreta sin freno
activo 'brake' porque supone un consumo de energía muy elevado
CloseSensor(SENSOR_1); %finalizamos el proceso volviendo a cerrar el
sensor de ultrasonidos al mundo exterior
v0=zeros(1,24);
% después de finalizar el proceso extraemos las mediciones situadas en
la 2ª columna de la matriz
%y las almacenamos en el vector v0 (1 fil x 24 col)que se utiliza como
%parámetro de entrada en la función RunLoca para la Relocalización.
x=1;
fil=1;
while (fil<25)%con este bucle transformamos los valores de v0 que
vienen en cm a pixeles
v0(x)=round(m(fil,2)*544/143.2)
x=x+1;
fil=fil+1;
end
```

FUNCION RunLoca

```
function FVr_bestmem= RunLoca(v0,theta,xrob,yrob)
map_name = 'labmiguelII.png';
[Wo, cm] = imread(map_name); %devuelve el mapa convertido en 1 y 0 (1=
espacio libre; 0= obstaculos)
%disp(sprintf('X teoría es: %d % complete/n',xrob));
%disp(sprintf('Y teoría es: %d % complete/n',yrob));
%disp(sprintf('theta teoría es: %d % complete/n',theta));
thet=(theta*2*pi)/(360);%transformamos theta a radianes
inc_x=10; %incremento de 10 pixel
inc_y=10;  %incremento de 10 pixel
inc_th=(pi/36); %incremento de 5 grados expresado en radianes, antes
la prueba se hizo con increm de pi/18 o 10 grados
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Script file for the initialization and run of the differential
% evolution optimizer.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% F_VTR     "Value To Reach" (stop when ofunc < F_VTR)
        F_VTR = -10;

% I_D       number of parameters of the objective function
        I_D = 3;

% FVr_minbound,FVr_maxbound   vector of lower and bounds of initial
population
%           the algorithm seems to work especially well if
[FVr_minbound,FVr_maxbound]
%           covers the region where the global minimum is expected
%               *** note: these are no bound constraints!! ***
      FVr_minbound = [(xrob-inc_x),(yrob-inc_y),(thet-inc_th)];
      FVr_maxbound = [(xrob+inc_x),(yrob+inc_y),(thet+inc_th)];
      I_bnd_constr = 1;  %1: use bounds as bound constraints, 0: no
bound constraints
```

```
% I_NP             number of population members
        I_NP = 100;  %pretty high number - needed for demo purposes
only

% I_itermax       maximum number of iterations (generations)
        I_itermax = 15;

% F_weight        DE-stepsize F_weight ex [0, 2]
        F_weight = 0.85;

% F_CR            crossover probabililty constant ex [0, 1]
        F_CR = 1;

% I_strategy     1 --> DE/rand/1:
%                      the classical version of DE.
%                2 --> DE/local-to-best/1:
%                      a version which has been used by quite a number
%                      of scientists. Attempts a balance between
robustness
%                      and fast convergence.
%                3 --> DE/best/1 with jitter:
%                      taylored for small population sizes and fast
convergence.
%                      Dimensionality should not be too high.
%                4 --> DE/rand/1 with per-vector-dither:
%                      Classical DE with dither to become even more
robust.
%                5 --> DE/rand/1 with per-generation-dither:
%                      Classical DE with dither to become even more
robust.
%                      Choosing F_weight = 0.3 is a good start here.
%                6 --> DE/rand/1 either-or-algorithm:
%                      Alternates between differential mutation and
three-point-
%                      recombination.

        I_strategy = 3

% I_refresh     intermediate output will be produced after "I_refresh"
%               iterations. No intermediate output will be produced
%               if I_refrsh is < 1
      I_refresh = 1;
      I_plotting =0;

S_struct.I_NP         = I_NP;
S_struct.F_weight     = F_weight;
S_struct.F_CR         = F_CR;
S_struct.I_D          = I_D;
S_struct.FVr_minbound = FVr_minbound;
S_struct.FVr_maxbound = FVr_maxbound;
S_struct.I_bnd_constr = I_bnd_constr;
S_struct.I_itermax    = I_itermax;
S_struct.F_VTR        = F_VTR;
S_struct.I_strategy   = I_strategy;
S_struct.I_refresh    = I_refresh;
S_struct.I_plotting   = I_plotting;
S_struct.medidas=(v0); %medidas  actuales del robot segun odometria
```

```
S_struct.Wo=Wo;
%S_struct.medidas=v0; %medidas  actuales del robot segun odometria

%*****************************************************************
% Start of optimization
%*****************************************************************

[FVr_bestmem,S_bestval,I_nfeval]= deopt('objfun',S_struct)
disp(sprintf('X teoría es: %d % complete/n',yrob));
disp(sprintf('Y teoría es: %d % complete/n',xrob));
disp(sprintf('theta teoría es: %d % complete/n',theta));
format short
save medidas.mat
%[FVr_x,S_y,I_nf] = deopt('objfun',S_struct) %pasamos a la funci?n
deopt el resultado devuelto por la func 'objfun' y la estructura
```