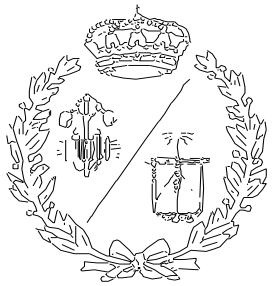
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Trabajo Fin de Grado

Técnicas de guiado, SLAM y visión artificial para robots móviles

(Guide, SLAM and computer vision techniques for mobile robots)

Para acceder al Título de

GRADUADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Autor: Fernando Pastor Durántez

Julio – 2014

TÍTULO	**Técnicas de guiado, SLAM y visión artificial para robots móviles. (Guide, SLAM and computer vision techniques for mobile robots).**		
AUTOR	**Fernando Pastor Durántez**		
DIRECTOR / PONENTE	**José Ramón Llata García**		
TITULACIÓN	*Grado en Ingeniería en Tecnologías Industriales*	FECHA	11 de Julio de 2014

PLABRAS CLAVE

SLAM (Simultaneous Localization And Mapping).
MRDS (Microsoft Robotics Develop Studio).
CCR (Concurrency and Coordination Runtime), DSS (Decentralized Software Services).
RobuLAB, Robosoft, robótica móvil, visión artificial (Computer vision), Navegación autónoma.

PLANTEAMIENTO DEL PROBLEMA

La necesidad del hombre por obtener la máxima cantidad posible de información del entorno motiva el problema del SLAM.
En las últimas décadas, el fuerte avance de la tecnología, ligado con la utilización de herramientas cada vez más complejas y la tendencia por automatizar en la medida de lo posible diferentes procesos, permite la realización de tareas cada vez más complejas, que precisan de un mayor grado de precisión, y que pueden incluso tener un cierto factor de peligrosidad añadido, en el que la integridad física de un hipotético operario se viera afectada.
La acción combinada de las técnicas de navegación autónoma en entornos no estructurados, que ni si quiera son conocidos de antemano, con una localización aleatoria de obstáculos a lo largo del terreno, con las técnicas de exploración del mismo y visión artificial para el reconocimiento de objetos, lugares o incluso personas, abre un enorme abanico de aplicaciones en las que la robótica mejorará nuestra inquietud de conocimiento del entorno de trabajo, proporcionándonos precisos mapas de los mismos con objetos, lugares o personas concretas especificadas por el usuario localizadas en el mismo, y la posibilidad de una futura asistencia dentro el mismo en situaciones que abarcan desde las meramente rutinarias, hasta las de emergencia en las que el objetivo sea la seguridad.

DESCRIPCIÓN DEL PROYECTO

El proyecto abarca desde las primeras fases de estudio del estado del arte, tanto de la robótica móvil y sus aplicaciones más actuales, como del SLAM y los diferentes tipos de este que en la actualidad existen, hasta la posterior elaboración de los algoritmos de control y guiado de navegación autónoma de una plataforma móvil de tipo deferencial, concretamente la llamada RobuLAB10, desarrollada por Robosoft, y la propia creación de los programas que controlan al mismo en sus labores de exploración en entornos no estructurados ni

conocidos de antemano, con obstáculos localizados aleatoriamente, la posterior reconstrucción del mapa de los mismos junto con la extracción de coordenadas de los lugares, objetos o incluso personas que considere el usuario de su interés, localizadas en el mismo haciendo uso de cámaras externas y técnicas de visión artificial, y finalmente la posibilidad de la navegación autónoma a estas coordenadas incluso cuando la distribución de obstáculos intermedios sea dinámica.
Para ello, se realizará previamente el estudio y desarrollo del modelo cinemático del robot móvil en cuestión para comprender su movimiento y tener la capacidad de generar las trayectorias deseadas, así como del software necesario para la creación de todos estos programas (MRDS y su paquete especifico RobuBOX y MATLAB, haciendo uso de una de sus Toolboxes: Image Acquisition).
Una vez realizados los programas y algoritmos necesarios, se desarrollará una batería de pruebas con el robot móvil y el posterior procesado de los datos recogidos, contrastando la precisión y exactitud de los resultados, concluyendo con la proposición de una alternativa de para la consecución de las diversas aplicaciones ya mencionadas de entre todas las propuestas.

CONCLUSIONES / PRESUPUESTO

En las soluciones que se aportan en este proyecto, se puede apreciar que los resultados de las técnicas utilizadas de SLAM no son bastos y aproximados, si no que recrean con bastante exactitud y precisión los entornos explorados, permitiéndonos incluso la localización de objetos concretos en el mismo. Estas tareas, arduas en muchos de los casos para el hombre, son un mero trámite de programación y puesta a punto para una serie de robots que pueden desempeñar de forma muy robusta este tipo de misiones.
Se proponen como futuras líneas de estudio y trabajos a realizar, así como otro tipo de proyectos, la modificación de los programas de visión artificial para el reconocimiento de otro tipo de objetos de amplio interés en distintos ámbitos de aplicaciones, como puede ser reconocimiento de extintores, reconocimiento de salidas de emergencia, planes de evacuación, reconocimiento de minas, reconocimiento de minerales o incluso reconocimiento de personas concretas.
La adición a la plataforma móvil de un manipulador robótico, para que esta pueda realizar cierta clase de tareas más elaboradas, como no solo el desplazamiento hasta el lugar del objeto concreto deseado, si no la manipulación del mismo.
Se proponen también la adición al robot de accesorios para una mayor interacción externa con el mismo, como pueden ser altavoces, micrófonos o incluso iluminación propia para la exploración de entornos oscuros.

BIBLIOGRAFÍA

- Robótica: manipuladores y robots móviles. Aníbal Ollero, Macomb, 2001

- Smith, Randall; Self, Matthew; Cheeseman, Peter (1987). «Estimating uncertain spatial relationships in robotics». 1987 IEEE International Conference on In Robotics and Automation. Proceedings.

- Montemerlo, Michael (2003). «FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data

Association». doctoral dissertation, tech. report CMU-RI-TR-03-28, Robotics Institute, Carnegie Mellon University.

- Proceedings of IEEE Robotics and Automation, 1988
- Mathworks
- Robosoft. Technical support.
- Craig J, John (2006). Robótica. México: pearson education.

Resumen

El objetivo del presente proyecto es desarrollar algoritmos de control y guiado automático de un robot móvil, para realizar labores de inspección (SLAM, Simultáneos Localization And Mapping) y navegación autónoma en entornos interiores no estructurados ni conocidos de antemano. Se utilizará un robot móvil de tipo diferencial llamado RobuLAB10, desarrollado por Robosoft, cuyo modelo cinemático se calculara en las etapas iniciales del proyecto. Se realizara una navegación odometrica, basada en la información proporcionada por los encoders del robot, así como un escáner laser para la detección del entorno explorado y cuerpos externos próximos. En base a esta información, se desarrollaran estrategias de guiado del robot para el seguimiento de trayectorias a través de obstáculos localizados aleatoriamente, y la creación de un mapa del entorno explorado por el robot. Paralelamente, se incorporaran cámaras de visión artificial que permitan reconocer el entorno y objetos concretos para así obtener una mayor información del mismo.

Abstract

The main goals of this project are to develop control and automatic guide algorithms of a mobile robot, in order to accomplish inspection task (SLAM, Simultaneous Localization and Mapping) and autonomous navigation in indoor unstructured and unknown in advance environments. It will be used a differential type mobile robot called RobuLAB10, developed by Robosoft, whose kinematic model will be calculated in the initial stages of the project. It will be used an odometric navigation, based on the information provided by the robot encoders, as well as a laser scanner used for the detection of the explored environment and the near external bodies. Taking into account this information, it will be developed guided strategies for the robot for the trajectory following through obstacles randomly located, and the creation of the explored environment map by the robot. At the same time, computer vision cameras will be incorporated to allow the robot to recognize in a better way the environment and also the concrete wised objects so we could obtain better information.

Índice General

1 INTRODUCCIÓN

1.1 Justificación

A lo largo de la historia de la humanidad, el hombre siempre ha intentado utilizar todos los recursos disponibles de su alrededor, tanto herramientas como el propio entorno, para ayudarse a realizar tareas que con el tiempo se han convertido en cada vez más complejas e incluso peligrosas.

Paralelamente a esta creciente complejidad en las tareas a realizar, las herramientas en las que se ha apoyado el hombre también se han vuelto más sofisticadas y eficientes con el avance, cada vez más acelerado, de la tecnología.

El resultado son herramientas complejas capaces de realizar trabajos de forma más precisa, rápida y eficiente que una persona.

Por lo tanto, con el fin de eliminar la necesidad de un operario en trabajos no solo complejos o repetitivos, sino peligrosos, que añadan un riesgo para la integridad física de la persona, surge el deseo de automatizar estos procesos.

Este tipo de trabajos engloba una gran cantidad de tareas que se hacen necesarias a día de hoy, como puede ser la de explorar un entorno no conocido, o simplemente navegar a través de el una vez se ha realizado este reconocimiento previo.

De esta necesidad surgen las técnicas de SLAM (simultaneous localization and mapping) que nos permiten explorar este entorno desconocido, proporcionándonos un mapa del mismo mientras el autómata que se ha destinado para tal tarea es capaz de localizarse en dicho lugar.

Así no solo obtenemos una información básica de los entornos deseados, si no que tenemos la posibilidad de desarrollar técnicas de navegación para posicionar a un robot en un punto concreto en ese entorno, para que posteriormente realice una tarea asignada, sin que exista si quiera un riesgo de colisión entre el robot propiamente dicho y cualquier cuerpo externo a él.

Para aumentar el campo de aplicación de estas técnicas, se tiende a mejorar la información que podemos recibir del entorno que exploramos, ya sea recreando el mismo de forma realista, utilizando imágenes reales o virtuales, o reconociendo objetos, lugares, personas... etc, que se encuentren en dicho lugar.

Con esta finalidad se utilizan técnicas de visión artificial (computer vision) que se superponen a las utilizadas para la exploración y navegación, mediante el uso de una o varias cámaras.

A partir de este punto surgen una gran cantidad de aplicaciones tanto comerciales, como industriales, militares, médicas o incluso espaciales.

1.2 Alcance y descripción del entorno

El alcance del presente proyecto será la exploración y construcción del mapeado de un entorno no estructurado ni conocido de antemano, en el cual podrá haber obstáculos intermedios, el reconocimiento de objetos situados a lo largo del lugar a explorar mediante la obtención de imágenes a través de una cámara para su posterior análisis (offline) y su posterior ubicación en el mapa generado, y finalmente la navegación a unas coordenadas indicadas por el usuario, las cuales podrán ser las de los objetos reconocidos.

El robot utilizado para el proyecto es la plataforma móvil RobuLAB10, fabricada por Robosoft, que dispone de un sistema de locomoción diferencial y dos ruedas de castor, una superficie lisa en la zona superior, 9 sensores de ultra sonidos, un láser de rango y 2 parachoques de infrarrojos.

El entorno en el cual se realizarán las pruebas para la exploración, reconocimiento y navegación será el laboratorio de robótica y visión artificial de la universidad de Cantabria. La zona destinada a este fin está definida por un terreno plano sin desniveles y paredes lisas. En dicha zona se han añadido obstáculos en la etapa de la exploración (manipulador robótico), y posteriormente para la etapa de navegación, se añadieron cajas como obstáculos adicionales para dificultar el movimiento del robot y comprobar que la solución adoptada funciona correctamente.

1.3 Estructura del proyecto

El proyecto se estructura intentando seguir la línea de progresos que se fueron alcanzando en su realización.

Se comienza primera con la descripción del problema que surge y cómo abordarlo.

Posteriormente se da una referencia a como se encuentra el estado del arte en el campo en el que nos estamos moviendo para poder comenzar con la descripción de la memoria.

Esta empezara como no puede ser de otra manera, con los datos que se tienen de partida, tanto del software como del hardware.

Una vez esto esté claro se puede comenzar con la definición de los servicios desarrollados para el funcionamiento del robot; el navegador,

que lleva al robot a unas determinadas coordenadas, y el explorador, que realiza el reconocimiento y recogida de datos del entorno.

Con estos datos nos metemos con la interpretación de estos datos recogidos para la construcción del mapa del lugar explorado y el análisis de las imágenes captadas para buscar en ellas los objetos que queremos encontrar.

Finalizamos, como no puede ser de otra manera, con las conclusiones, y añadiremos 3 anejos que hacen referencia al modelo cinemático del robot, y a servicios que necesita paralelamente para que el navegador y el explorador funcionen debidamente.

1.4 Objetivos

Podemos marcar como objetivos del proyecto los siguientes:

1. Estudio del modelo cinemático de la plataforma móvil RobuLAB10
2. Estudio de Microsoft Robotics Studio
3. Estudio del paquete RobuBOX
4. Desarrollo de los algoritmos de exploración y navegación
5. Desarrollo del software del robot
    a. Desarrollo del servicio navegador en C#
    b. Desarrollo del servicio explorador en C#
    c. Ajuste de las trayectorias tomadas en ambos servicios mediante pruebas en el laboratorio

6. Desarrollo de los algoritmos para la interpretación de datos y mapeado
7. Desarrollo de los algoritmos de visión artificial para el reconocimiento del objeto deseado.
8. Desarrollo del software de mapeado y reconocimiento de objetos
    a. Desarrollo del programa "SLAM.m" en MATLAB
9. Estudio de los resultados obtenidos.

1.5 Descripción y justificación de la solución adoptada

La solución adoptada presenta un sistema robusto que es capaz de recorrer siempre la totalidad el entorno en el que el robot se encuentra, sin perderse y sin encontrarse en situaciones sin salida.

El sistema de navegación, también de gran robustez, llega en todas las situaciones a la meta siempre que esta sea físicamente alcanzable, en las versiones 1.9.2 y en la final.

Ambos servicios contienen las características requeridas y siempre protegerán la integridad física de los cuerpos externos que puedan estar situados en el lugar a recorrer, por lo que también protegerán al robot de cualquier golpe.

Además, en la solución adoptada, se incluye en la programación de los servicios un sistema de seguridad que detiene al robot y lo hace girar sobre sí mismo para evitar colisiones cuando estas ya son inevitables.

Esto asegura que ninguna persona resulte jamás herida mientras el robot realiza su trabajo (tal y como dicta la primera ley de la robótica), e incluso el propio robot jamás resultara dañado.

Los algoritmos y el programa de mapeado son válidos en tanto que reflejan con bastante exactitud el entorno que el robot explora cuando comparamos el mapa generado con el plano del lugar explorado.

Los objetos encontrados por el programa de visión artificial se corresponden con los deseados y comprobamos que los sitúa en el lugar en el que se depositaron para la realización de las pruebas.

Por su puesto el objeto a reconocer puede variar y queda en la mano del usuario el añadir los algoritmos deseados para el reconocimiento del objeto que se desee encontrar en cada momento. En este proyecto, los objetos serán cuadrados de color verde, por lo que el algoritmo reconocerá cuadrados preferentemente con tonalidades verdes. No hay más que cambiar el código del algoritmo para estos objetos por el que reconoce los deseados para que el robot localice aquello que precise el usuario.

2 Estado del arte de la robótica móvil.

2.1 Robots móviles.

El desarrollo de los robots móviles responde a una necesidad de extender el propio campo de aplicación de la robótica, el cual estaba restringido en sus inicios al alcance de la estructura mecánica anclada en uno de sus extremos. De esta forma también se consigue abordar otra de las cuestiones de la robótica, y que es incrementar la autonomía del robot, limitando todo lo posible la intervención humana.

Desde este punto de vista, el de la autonomía, los robots móviles tienen como precedentes los dispositivos electromecánicos, creados desde los años 30's con el objeto de desarrollar funciones inteligentes tales como por ejemplo solucionar laberintos. Destaca la tortuga de Walter (1948) (figura 0) que reaccionaba ante la presencia de obstáculos, subía pendientes y se autoredireccionaba a la estación de carga cuando el computo de la energía restante advertía que era preciso.

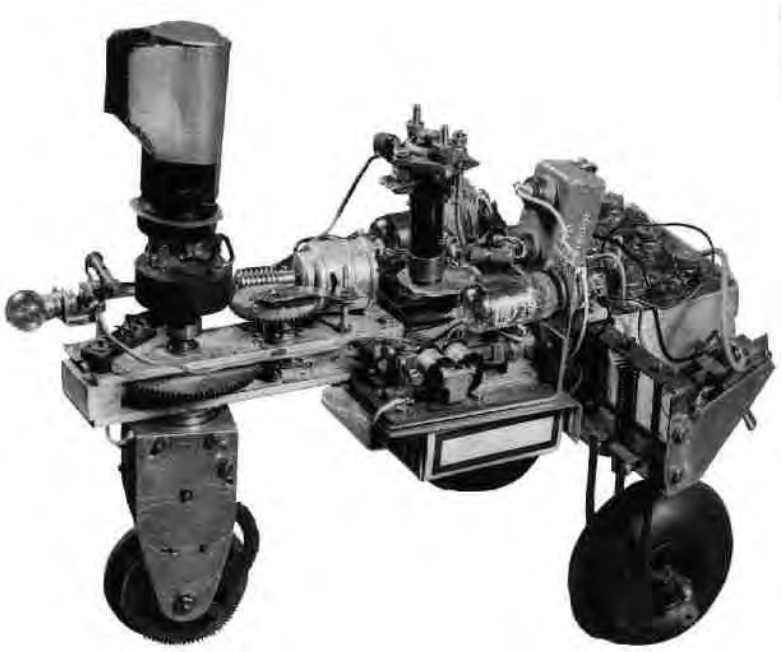


Figura 0. Tortuga de Walter. Fuente: http://vonneumannmachine.wordpress.com/

No obstante, estos trabajos de investigación no guardan relación con los vehículos autónomos que comenzarse a desarrollarse desde los años ó0's en la industria, siendo estos guiados por cables bajo el suelo o mediante sensores ópticos para seguir líneas trazadas en las plantas.

Posteriormente, en los años 70's, se vuelve a trabajar en robots móviles de mayor autonomía, aunque el desarrollo tecnológico aun no era suficiente para lograr la navegación autónoma de una forma eficiente.

En los años 80's, el incremento sobrecogedor de la capacidad computacional, acompañado del desarrollo de nuevos sensores, mecanismos y sistemas de control, permitieron aumentar esta deseada autonomía.

La autonomía de un robot móvil se basa en el sistema de navegación automática. En estos sistemas se incluyen tareas de planificación, percepción y control.

En los robots móviles, el problema de planificación se descompone en:

1. Planificación global de la misión.
2. Planificación de la ruta.
3. Planificación de la trayectoria
4. Evitar obstáculos no esperados.

En cualquier caso, el problema del control automático preciso de un vehículo con ruedas resulta más complejo que el de los manipuladores, debido a la presencia de restricciones no holónomas. Los bucles de control se han de plantear no solo en el espacio de las variables articulares, sino que también en coordenadas world, y las ecuaciones del movimiento son complejas si tenemos en cuenta la interacción con el terreno.

También, el control del vehículo requiere disponer de medidas de su posición y orientación a intervalos suficientemente cortos. La técnica más simple, que además es la utilizada en este proyecto, es el uso de la Odometría a partir de las medidas suministradas por los sensores situados en los ejes de movimiento, en nuestro caso, encoders o codificadores ópticos.

Esta técnica así como su aplicación en el proyecto será explicada de forma más extensa a lo largo de los sucesivos apartados

2.1.1 Vehículos con ruedas

Los vehículos con ruedas son la solución más simple para conseguir la movilidad en terrenos suficientemente duros y libres de obstáculos, permitiendo conseguir velocidades relativamente altas.

Como limitación más significativa, cabe mencionar el deslizamiento en la impulsión.
Dependiendo de las características del terreno pueden presentarse también deslizamientos y vibraciones. La locomoción mediante ruedas es poco eficiente en terrenos blandos.

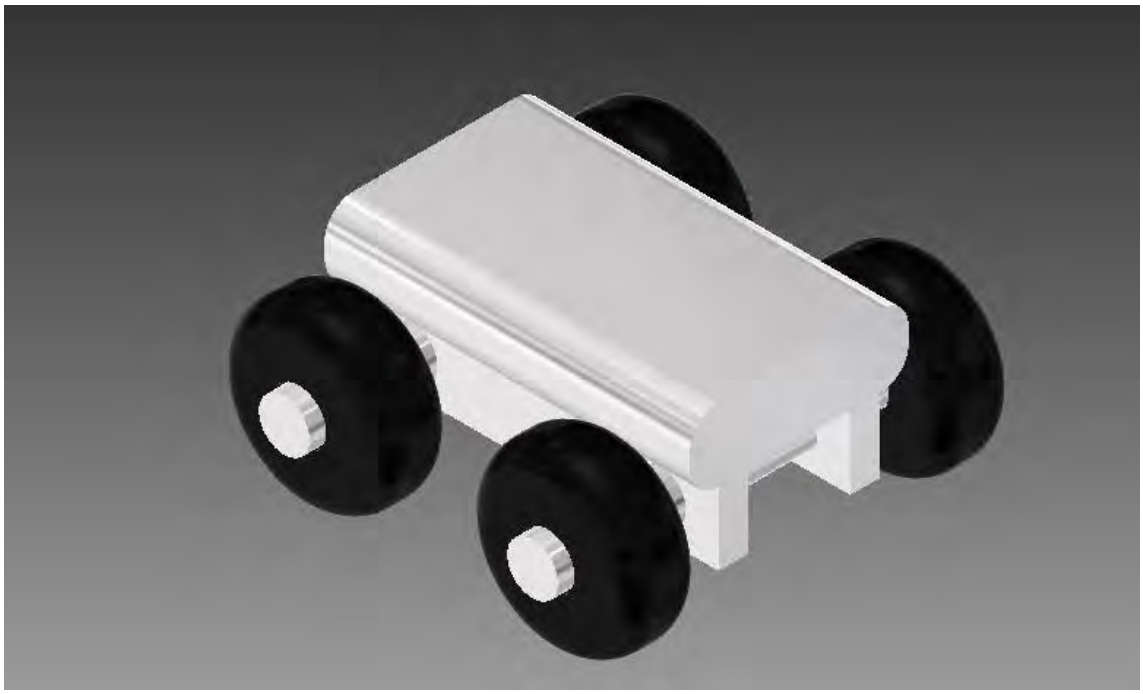
Por otra parte, excepto en configuraciones muy especiales, no es posible alterar internamente el margen de estabilidad para adaptarse a la configuración del terreno, lo que limita de forma importante los caminos aceptables del soporte.

Los robots móviles emplean diferentes tipos de locomoción mediante ruedas que les confieren características y propiedades diferentes respecto a la eficiencia energética, dimensiones, cargas y maniobrabilidad. La mayor maniobrabilidad se consigue en vehículos omnidireccionales. Un vehículo omnidireccional en el plano es capaz de trasladarse simultánea e independientemente en cada eje del sistema de coordenadas, y rotar sobre el eje perpendicular.

A continuación se comentan brevemente las características más significativas de los sistemas de locomoción más comunes en robots móviles

2.1.2 Ackerman o Tipo Coche

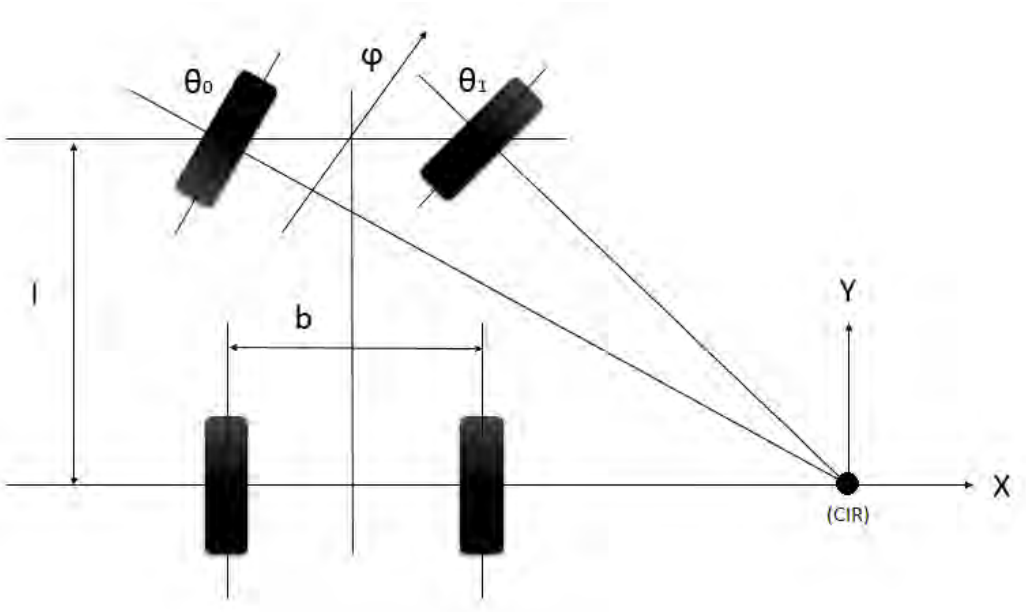
Es el utilizado en vehículos de cuatro ruedas convencionales. De hecho, los vehículos robóticos para exteriores resultan normalmente de la modificación de vehículos convencionales tales como automóviles o incluso vehículos más pesados (figura 1).



(Figura 1)

El sistema se basa en dos ruedas traseras tractoras que se montan de forma paralela en el chasis principal del vehículo, mientras que las ruedas delanteras son del tipo direccionamiento, y se utilizan para seguir la trayectoria del robot (ver figura 2).
Debido a la similitud con los vehículos convencionales, este sistema

también recibe el nombre de tipo coche o modelo Ackerman. La rueda delantera interior gira un ángulo ligeramente superior a la rueda exterior, de forma tal que los ejes de prolongación de las ruedas delanteras (directrices) se cortan en el CIR o centro instantáneo de rotación, que se sitúa en un punto del eje generado de la prolongación del eje de las ruedas traseras (motrices). Esto elimina el deslizamiento que provoca los sobre virajes de la plataforma. El lugar de los puntos trazados sobre el suelo por los centros de los neumáticos, son circunferencias concéntricas con centro el eje de rotación CIR. Si no se tienen en cuenta las fuerzas centrifugas, los vectores de velocidad instantánea son tangentes a estas curvas. Por lo que las velocidades de movimiento del móvil, deber evitar que las ruedas no resbalen.

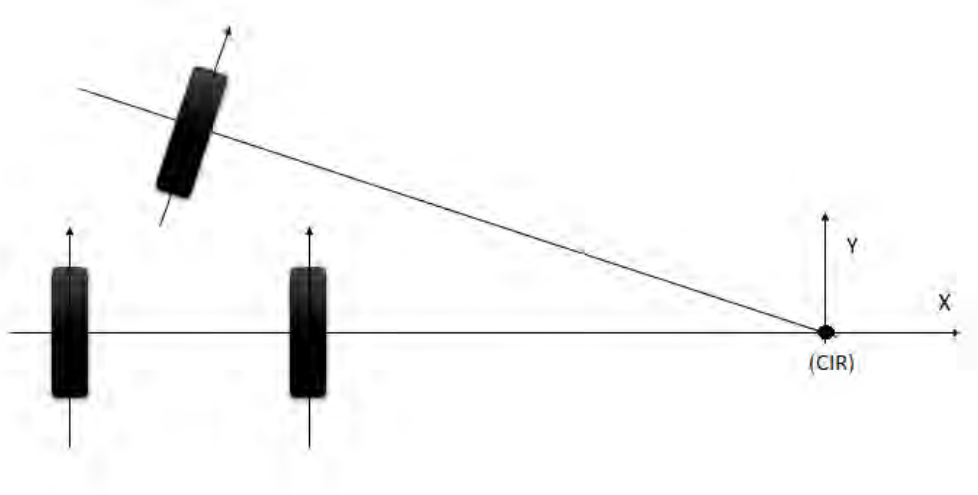


(Figura 2)

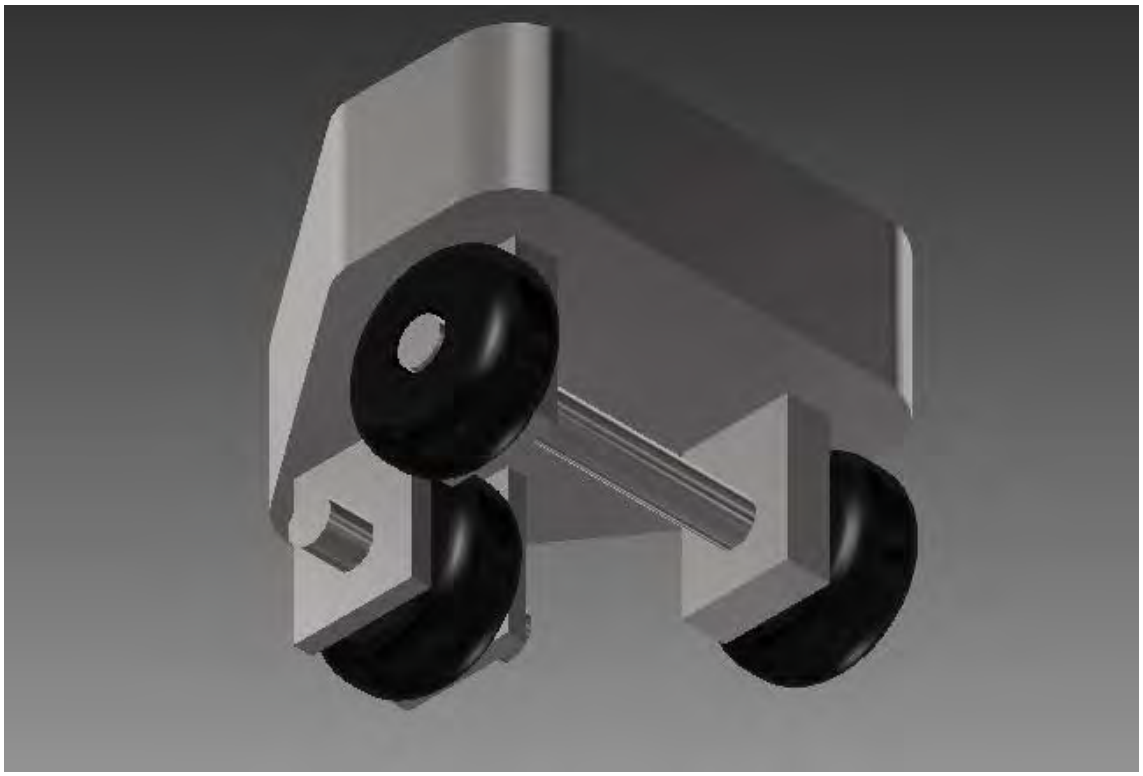
En los robots móviles con configuración Ackerman, se presentan dos ángulos de giro, uno en cada rueda. Esto genera mayores problemas a la hora de realizar el control, por lo que en muchas ocasiones lo que se hace es unificar los ángulos de direccionamiento en uno solo, por lo que los radios de giro para los cuales el robot no muestra deslizamiento lateral son mayores que en otras configuraciones.

2.1.3 Triciclo clásico.

Este sistema de locomoción se basa en una rueda delantera que sirve tanto para la tracción como para el direccionamiento. El eje trasero, con dos ruedas laterales, es pasivo y sus ruedas se mueven libremente. La maniobrabilidad es mayor que en la configuración Ackerman motivado por la existencia de una sola rueda de direccionamiento (ver figura 3 y 4), pero a su vez esto causa que se puedan presentar problemas de estabilidad en terrenos difíciles. El centro de gravedad tiende a desplazarse cuando el vehículo se desplaza por una pendiente, provocando una pérdida de tracción, o incluso el volcado, por lo que es preferible situar el Centro de gravedad cerca del suelo.



(Figura 3)

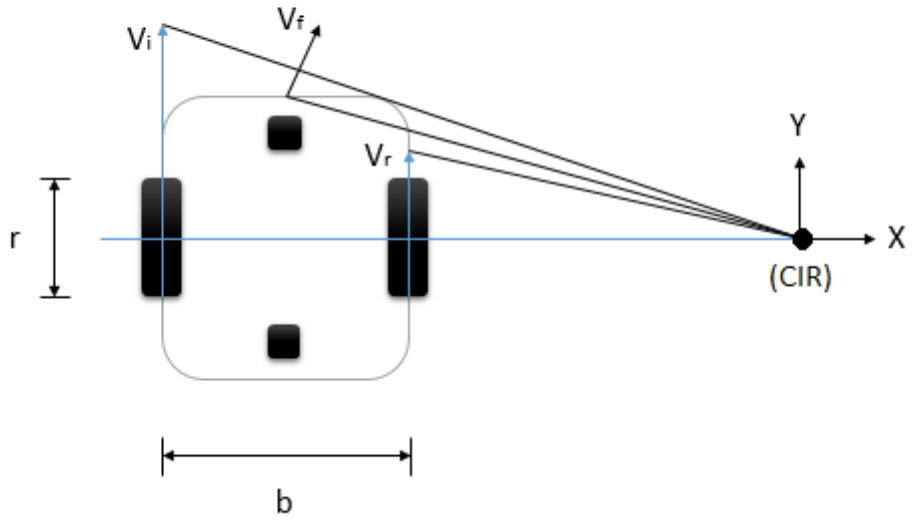


(Figura 4)

2.1.4 Robot diferencial

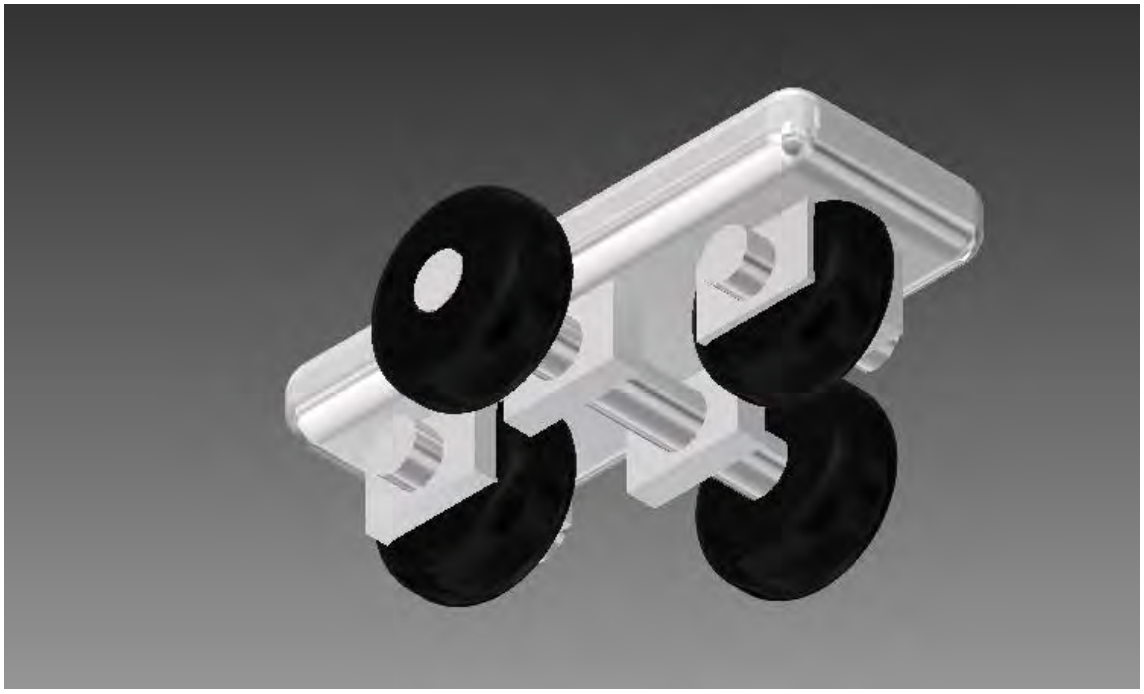
Este tipo de direccionamiento viene dado por la diferencia de velocidades de las ruedas laterales. La tracción se consigue también con estas mismas ruedas. Dos ruedas montadas en un único eje son independientemente propulsadas y controladas, proporcionando ambas tracción y direccionamiento. La combinación del movimiento de las dos ruedas provoca el movimiento alrededor del CIR. Este sistema es muy útil si consideramos la habilidad del movimiento del móvil, presentando la posibilidad de cambial su orientación sin movimientos de traslación, lo que podríamos llamar cambio de spin. Las variables de control de este sistema son las velocidades angulares de las ruedas izquierda y derecha. Los diferentes modelos cinemáticos existentes proporcionan trayectorias perfectamente definidas, y con ello obtenemos el posicionamiento deseado.

En la figura 5 se presenta el diagrama de velocidades que se estudiará más a fondo en los apartados sucesivos.



(Figura 5)

Adicionalmente, existen una o dos ruedas para soporte. Esta configuración es la más frecuente en robots para interiores de pequeño tamaño.



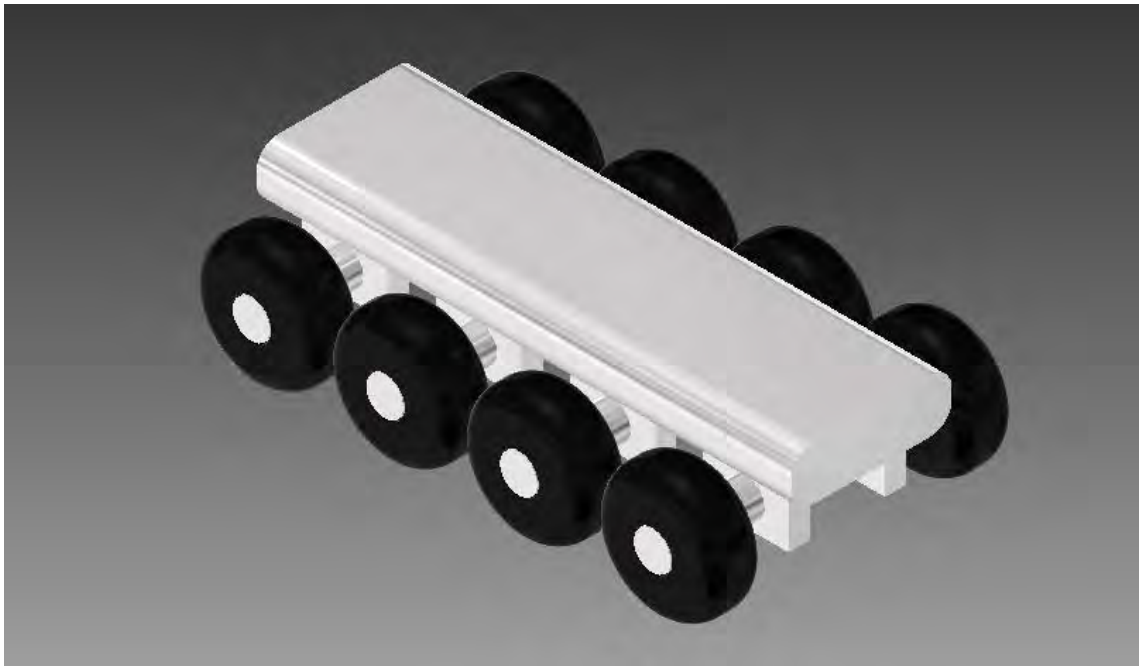
(Figura 6)

El robot utilizado para la realización de este proyecto, robuLAB10 (figura 7), es de tipo diferencial.

(Figura 7) RobuLAB10. Fuente: http://www.robosoft.com/

2.1.5 Skid Steer.

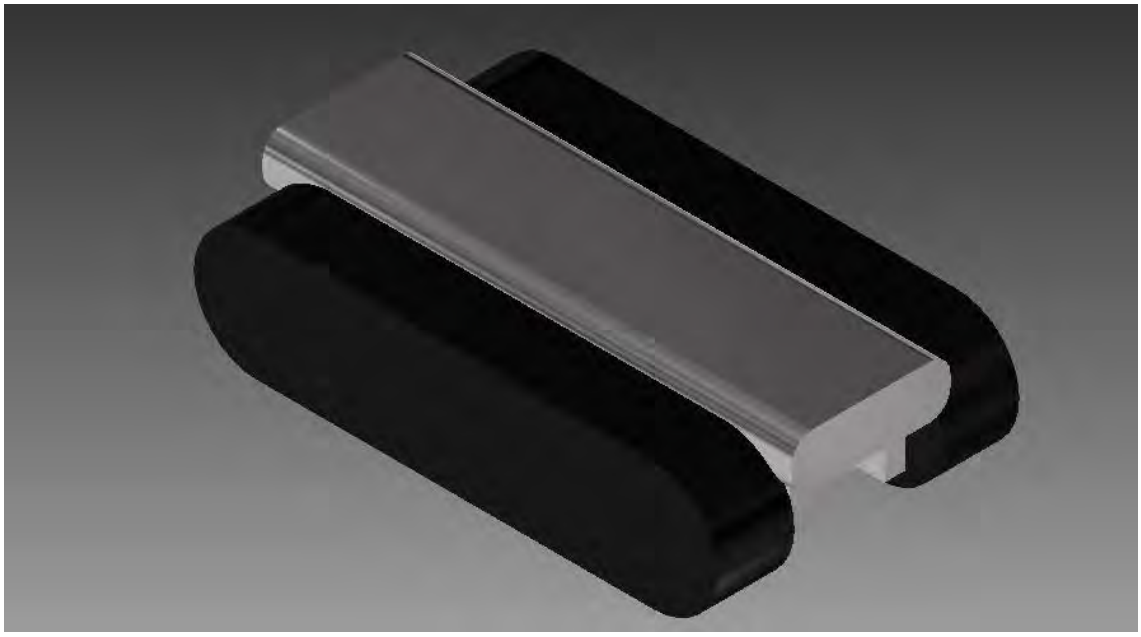
Se disponen varias ruedas en cada lado del vehículo que actúan de forma simultánea. El movimiento es el resultado de combinar las velocidades de las ruedas de la izquierda con las de la derecha (figura 8). Estos robots se han usado para la inspección y obtención de mapas de tuberías enterradas empleando para ello sistemas de radar ("Groud Penetraining Radar"). Se han utilizado en aplicaciones mineras y en misiones de exploración espaciales no tripuladas.



(Figura 8)

2.1.6 Pistas de Deslizamiento.

Son vehículos tipo oruga en los que tanto la impulsión como el direccionamiento se consiguen mediante pistas de deslizamiento. Pueden considerarse funcionalmente análogas al skid steer. De forma más precisa, las pistas actúan de forma análoga a ruedas de gran diámetro. La locomoción mediante pistas de deslizamiento es útil en navegación "campo a través" o en terrenos irregulares, en los cuales presenta un buen rendimiento. En este caso, la impulsión está menos limitada por el deslizamiento y la resistencia al desgaste es mayor.



(Figura 9)

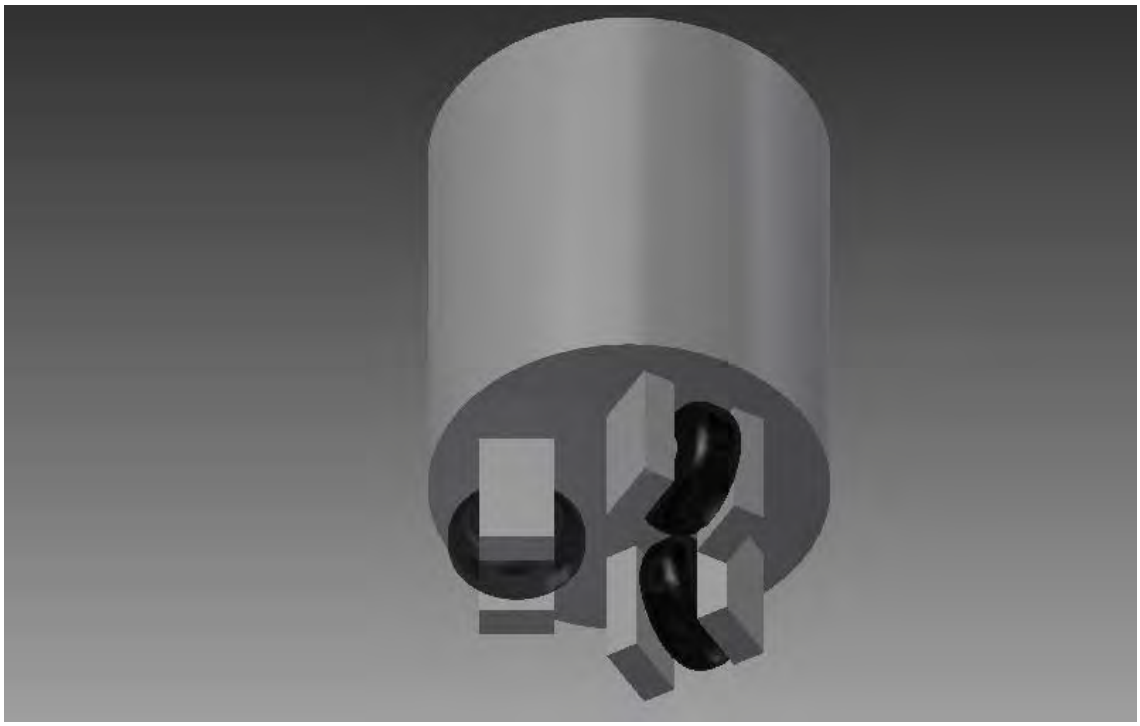
2.1.7 Síncronas.

Consiste en la actuación simultánea de todas las ruedas, que giran de forma síncrona. La transmisión se consigue mediante coronas de engranajes ("syncro drive") o con correas concéntricas.

En una conducción sincrónica del robot, cada rueda es capaz de ser conducida y dirigida.
Las configuraciones típicas son:

- Tres ruedas directrices se montan acopladas en los vértices de un triángulo equilátero muchas veces debajo en una plataforma cilíndrica (figura 10).

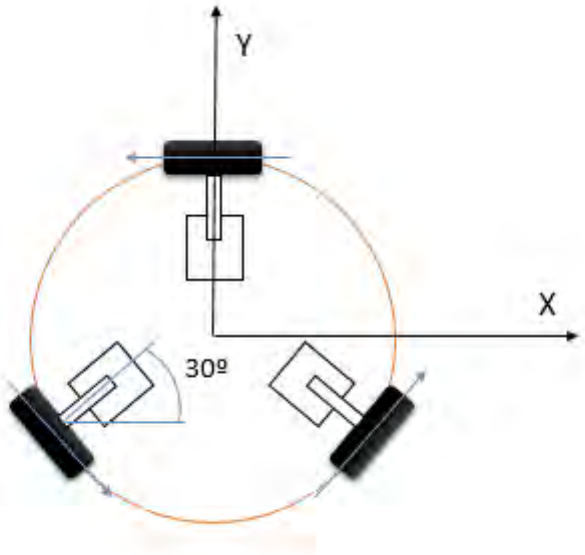
- Todas las ruedas propulsan y giran al unísono.



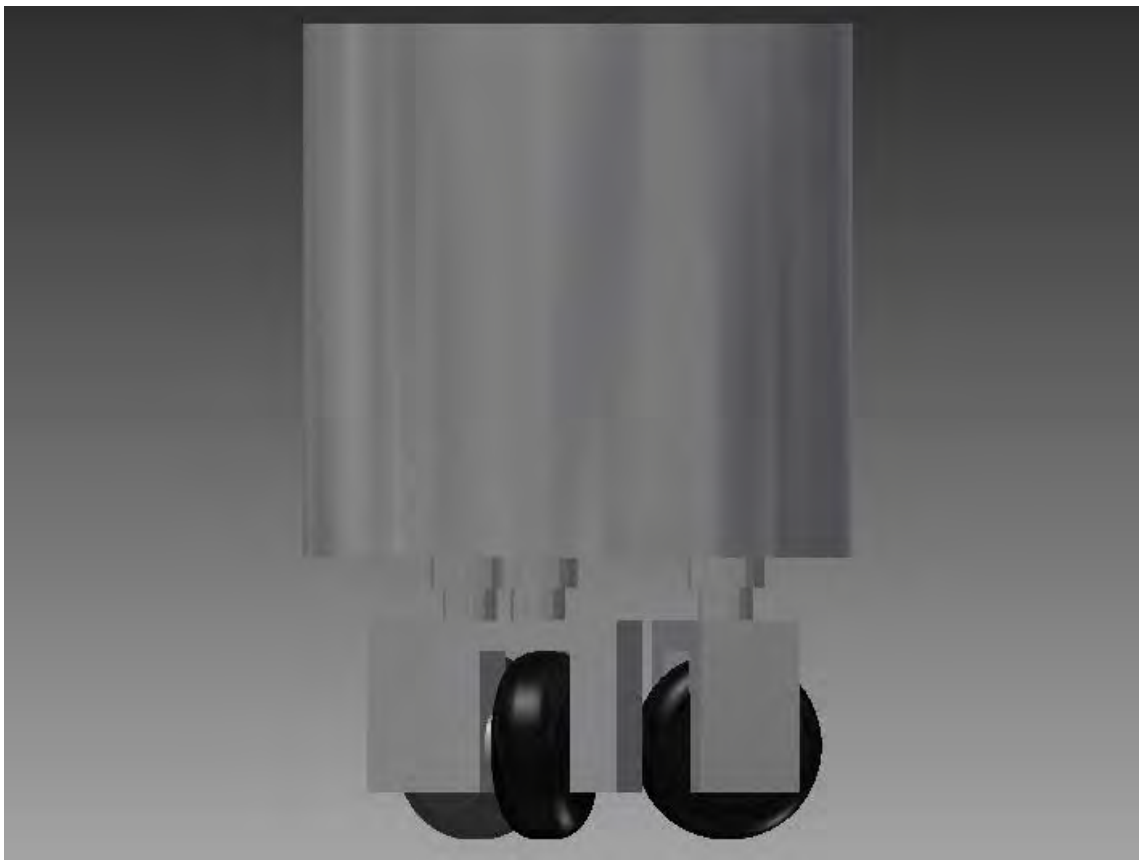
(Figura 10)

2.1.8 Tracción Omnidireccional.

Este sistema de tracción de basa en la utilización de tres ruedas directrices y motrices (figura 11). Esta configuración tiene tres grados de libertad, por lo que puede realizar cualquier movimiento, y posicionarse en cualquier posición en cualquier orientación. No presenta limitaciones cinemáticas.



(Figura 11)



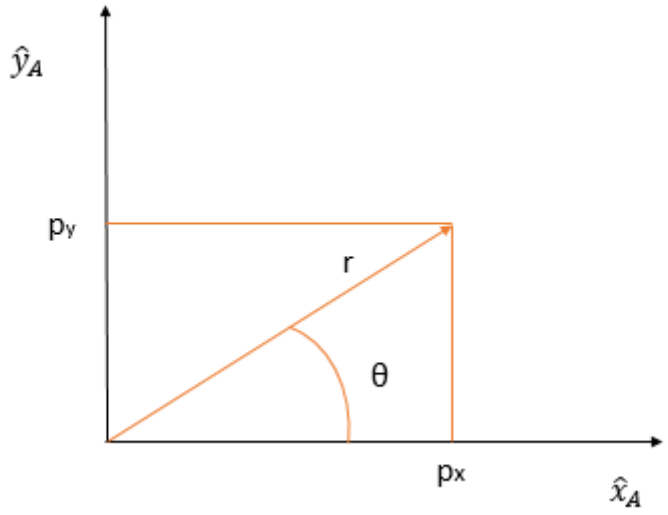
(Figura 12)

2.1.9 Representación de la Posición y Orientación de un Robot Móvil en el Plano.

Para poder describir de forma conveniente las posiciones y orientaciones de los robots móviles en el espacio, utilizaremos los movimientos de rotación y traslación de forma tal que podremos conocer la posición y la orientación del robot en el plano realizando operaciones elementales. Para conocer el movimiento del móvil es necesario conocer la posición y la orientación inicial y final, este camino nos introduce en el estudio de la cinemática del móvil.

El conocimiento de la localización de un robot en el plano, es esencial pasa el estudio de los robots móviles, ya que estos se mueven en la mayoría de aplicaciones en un plano conocido y sin irregularidades. En este caso es necesario conocer las dos coordenadas (x,y) y el ángulo de orientación (theta) para tener totalmente definido el móvil.

La notación que se utilizará será la notación de Craig (1989).

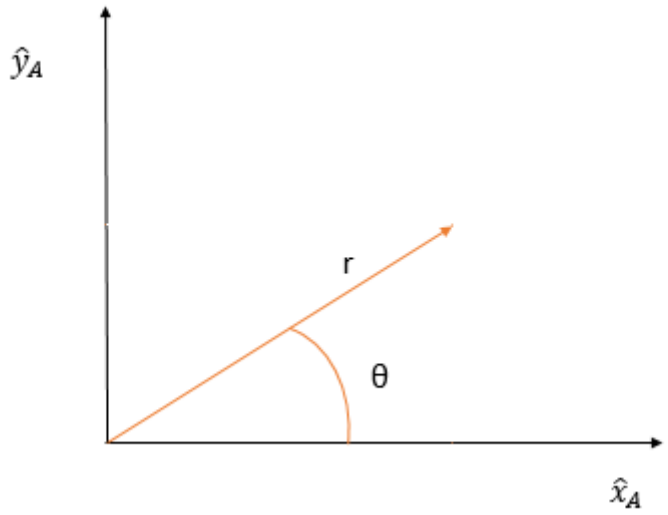


(Figura 13)

Partiremos de un sistema de coordenadas fijo que llamamos {A}. Este representa las coordenadas world de nuestro sistema. La expresión de las coordenadas de un punto situado en P respecto al sistema de coordenadas {A} (Figura 13) se puede expresar mediante un vector de posición de la siguiente manera:

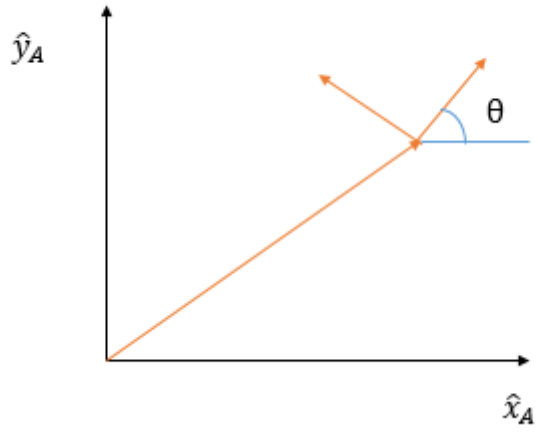
$$P = \begin{bmatrix} P_x \\ P_y \end{bmatrix}$$

A parte de sistemas de coordenadas cartesianas, podemos utilizar coordenadas polares, de forma que el punto P se puede expresar como la distancia al origen de coordenadas r y el ángulo θ que forma el vector r con el eje $\hat{x}_A$ (figura 14).



(Figura 14)

Ahora colocamos un sistema de coordenadas {B} en el centro de masas del móvil de forma que los vectores unitarios de este segundo sistema serán $\hat{y}_A$ y $\hat{y}_B$, tal y como se muestra en la figura 15. La dirección del vector $\hat{y}_B$ forma un ángulo θ con el vector $\hat{x}_A$. Obsérvese que al tratarse de localizar a un robot móvil que se desplaza en el piano, este segundo sistema será solidario al robot con $\hat{y}_B$ en la orientación del robot.

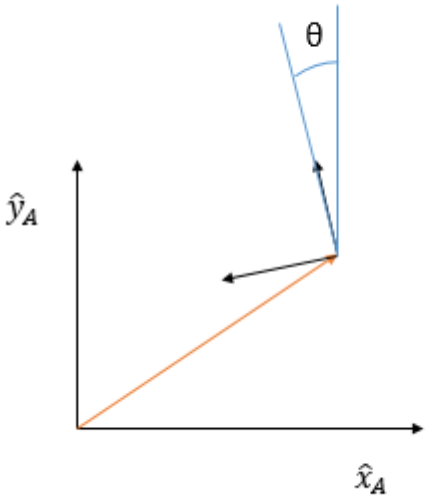


(Figura 15)

Si Se expresan los vectores unitarios del sistema {B} en el {A}, se escribe $\hat{X}_B^A, \hat{Y}_B^A$

$$R_B^A = (\hat{X}_B^A \quad \hat{Y}_B^A) = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}$$

Esta matriz se conoce como matriz de rotación, y desempeña un papel muy importante en los modelos utilizados en robótica.

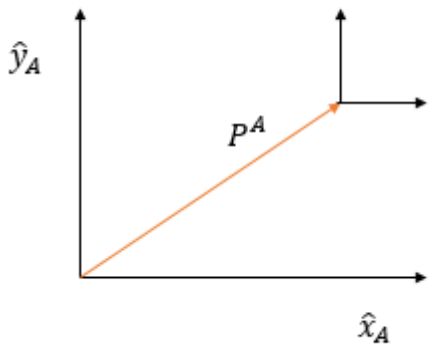


(Figura 16)

Si utilizamos el siguiente ángulo θ para orientar al robot (figura 16) podemos expresar R_B^A como:

$$R_B^A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Para conocer la posición del origen del sistema de coordenadas de {B} en coordenadas de los ejes universales que llamamos {A}, es necesario además, realizar una traslación de los ejes:



(Figura 17)

Supongamos que el sistema {B} tiene sus vectores de dirección coincidentes con los del {A}, como se ilustra en la Ilustración 17. Esto sucede cuando el robot se desplaza sin variar su orientación. El origen del sistema {B} se localizar con respecto al {A} mediante el vector:

$$P^A = \left[P^A_{xorg} P^A_{yorg}\right]^T$$

Por consiguiente, las coordenadas de un punto cualquiera del plano en los dos sistemas están relacionadas mediante:

$$P^A = \begin{cases} P^A_x = \hat{X}^B_A \cdot P^B \\ P^A_y = \hat{Y}^B_A \cdot P^B \end{cases}$$

Expresiones en las cuales las componentes de los vectores pueden sumarse por estar estos vectores en la misma dirección. Estas dos ecuaciones definen la traslación del sistema de coordenadas {B} sobre el sistema de coordenadas {A}.

De esta forma podemos definir la posición del móvil como el producto de una rotación y una traslación. La composición de los términos rotación y traslación para hacer coincidir los ejes sobre los de referencia con los ejes universales expresa las coordenadas de la cinemática directa del móvil.

Esto significa que, para expresar un punto objetivo, al que llamamos F, será necesario realizar varias operaciones para situarlo en coordenadas universales.

2.2 Estado del arte SLAM

2.2.1 Motivación del problema del SLAM

Existen ocasiones en las que se conoce de antemano el entorno en el cual se moverá el robot y es posible proporcionarle un mapa del mismo. Puede tomarse como ejemplo un agente que siempre limpia una misma habitación de un apartamento.

Por otro lado existen situaciones en las que un agente robótico debe moverse en un entorno desconocido, pero cuenta con información precisa sobre su ubicación. Para conocer su ubicación el agente puede utilizar, por ejemplo, visión global, GPS u odometría, técnica utilizada en este proyecto.

En este caso, el robot deberá armar un mapa del entorno apoyándose constantemente en la información de su ubicación para tener mayores probabilidades de éxito. Sin embargo, existes casos, aún más complejos que los mencionados, en los que no se conoce el entorno y tampoco se tiene información sobre la ubicación exacta del robot. En este caso el robot deberá generar un mapa y mantener su ubicación en el mismo de forma concurrente. Esta tarea resulta compleja por el hecho que para poder localizarse de forma precisa se necesita un mapa, y por otro lado, para poder crear un mapa es menester estar localizado de forma precisa. Esta es la tarea que estudia el SLAM, localización y creación de mapas simultaneo. Algunos ejemplos en los cuales es importante realizar SLAM son:

- Exploración espacial
- Rescate en zonas modificadas por catástrofes
- Robots que realizan tareas domésticas
- Vehículos dirigidos de forma autónoma

2.2.2 El problema del SLAM

El problema del SLAM (en español: Localización y creación de mapas simultaneo) se aplica cuando el robot no tiene acceso a un mapa del entorno ni conoce tampoco su posición en el mismo. Por ejemplo, se puede dar la situación en la que tengamos que utilizar un robot para tareas de rescate en una zona donde el humano no puede acceder debido a los elevados niveles de radioactividad, en este caso el robot no posee conocimiento de su ubicación ni del ambiente.

Luego, el robot solo dispone de la información proporcionada por las medidas obtenidas de los sensores y la noción de movimiento propio. El robot intentará obtener un mapa del entorno y simultáneamente localizarse en dicho mapa

En el contexto del SLAM existen diversas fuentes de incertidumbre, es decir factores que incrementan la dificultad de estimar la ubicación y mapa del entorno correctas. A continuación se incluyen algunos de estos factores:

- **Ruido de los sensores**: Los sensores utilizados en un robot presentan usualmente ruido en los datos que proporcionan.
- **Desplazamiento impreciso del robot**: El resultado de un movimiento del robot es de naturaleza no determinista. Esto se debe a que, por ejemplo, las ruedas de un robot pueden resbalar sobre el suelo. Como resultado, no es posible conocer a ciencia cierta cómo fue el movimiento real del robot como resultado de una orden de movimiento (p. e. avanzar, retroceder, girar) o en base a información de odometría.
- **Simetrías en el entorno**: El entorno sobre el que opera el robot puede presentar simetrías que dificultan la determinación de la posición actual del robot en el mapa confeccionado.
- **Observabilidad parcial**: La ausencia de un mecanismo de visión global del entorno dificulta la estimación de la posición y la confección del mapa.
- **Entorno dinámico**: Si se trabaja en un entorno dinámico, los cambios en el entorno dificultarán el proceso de estimación de la

posición debido a que el mapa confeccionado puede estar desactualizado.

En pos de adaptarse a esta incertidumbre, la gran parte de las soluciones de SLAM plantea la solución de la estimación de la posición del robot y el mapa del entorno como distribuciones de probabilidades.

En la actualidad existen métodos de SLAM robustos para mapear ambientes estáticos, estructurados y de tamaño limitado. Realizar SLAM en entornos dinámicos, espacialmente grandes y desestructurados sigue siendo un problema abierto a la investigación.

Los principales desafíos a los que se enfrenta el SLAM en la actualidad se presentan en los siguientes apartados.

2.2.3 Manejo de incertidumbre

Como se mencionó anteriormente, determinar la posición exacta del robot requiere conocimiento sobre la posición exacta del mapa. Por otro lado, la determinación del mapa del entorno requiere conocimiento de la posición exacta del agente.

Dado que el agente no posee inicialmente ninguno de estos datos, y dado que la incertidumbre de su posición aumenta con su movimiento, el algoritmo de SLAM debe ser capaz de manejar cierto error en los datos que son computados.

En la figura 18 se puede observar un robot que realiza una tarea doméstica para la cual no tiene un mapa ni tampoco su ubicación. Mientras realice la tarea de armar el mapa deberá manejar la incertidumbre de la posición. Esta incertidumbre debe ser manejada de forma tal que el error en las estimaciones no crezca constantemente (de

manera de evitar una divergencia en la estimación de la posición del robot y del mapa).

(Figura 18)

Imagen extraída de www.irobot.com.

2.2.4 Sensores

Los sensores son limitados en lo que pueden percibir y poco precisos en sus medidas.

Estas limitaciones vienen dadas por muchos factores. El rango y la resolución de un sensor están sujeto a limitaciones físicas. Un claro ejemplo son las cámaras cuya calidad de imagen es limitada.

Los sensores además están sujetos a ruido estocástico, lo que perturba las medidas de formas impredecibles y hace que la información extraída sea poco fiable.

Otro problema que incrementa el ruido en los sensores es el hecho de realizar medidas con el robot en movimiento, hecho que genera mayor error en las medidas de los sensores.

2.2.5 Controles

En la fase de exploración del SLAM se envían órdenes de movimiento al robot, conocidos como controles. Una vez que se le envía una orden de movimiento al robot, los sensores en los motores (odometría) permiten conocer cuál ha sido el movimiento que hizo el robot a partir de la orden que fue emitida.

Esta información luego se procesará para actualizar la posición del robot. El problema ocurre cuando una de las ruedas del robot queda en el aire o simplemente resbala sobre una superficie con baja adherencia, lo que implica que el robot no cambió su posición, o lo hizo pero en menor medida de la esperada, sin embargo los datos devueltos por los sensores de odometría indican que el movimiento fue completo.

Esto genera un error entre la pose real del robot y su creencia de la posición. En la figura 19 se puede ver un robot con una rueda en el aire, producto de las irregularidades del entorno en el que se encuentra. Esta es una clásica situación en la que la información de odometría no se corresponde con el movimiento real del robot.

(Figura 19)

Imagen extraída de Wordpress.com.

2.2.6 Ciclos

Cerrar ciclos refiere a la situación en que el robot debe poder recordar un lugar cuando ya ha sido visitado. Realizar esta tarea con éxito se vuelve primordial en los mapas que poseen algún cruce.

2.2.7 Asociación incorrecta

El algoritmo de SLAM debe ser lo suficientemente robusto como para no confundir dos lugares diferentes de forma que lo lleve a creer que son el mismo.

En caso de que el robot piense que dos lugares diferentes con características similares son efectivamente el mismo, errará un ciclo

donde no lo hay y generará un mapa incorrecto del entorno lo cual puede llevar a errores catastróficos.

2.2.8 Capacidad de cómputo

Una gran limitación del SLAM es el procesamiento. Los algoritmos de SLAM suelen realizar tareas de procesamiento pesadas debido a la densidad de la representación del mapa y a la complejidad de los cálculos involucrados en la estimación de la posición.

Esto muchas veces hace difícil el procesamiento dentro del robot y obliga a extraer los datos recogidos para ser procesados externamente.

2.2.9 Clasificaciones de SLAM

A continuación se incluyen algunas posibles clasificaciones del problema del SLAM.

Offline vs. Online

En el caso del SLAM online se procesa la información en el mismo robot mientras este navega en el entorno.

Por otro lado, en el offline se realiza SLAM sobre un conjunto de datos que previamente fueron recuperados con algún robot, tanto de la medida de sus sensores como las medidas de odometría (movimiento).

Topológico vs. Métrico

Algunas técnicas de creación de mapas solamente mantienen la descripción de algunas características del entorno, que caracteriza la relación entre lugares.

Estos métodos son conocidos como topológicos.

Un mapa topológico se define por un conjunto de lugares diferentes y otro conjunto que caracterizan las relaciones entre estos.

Por otro lado, los métodos métricos proveen información métrica entre los lugares.

En los últimos años los métodos topológicos han pasado de moda a pesar de la amplia evidencia de que los humanos utilizan a menudo información topológica.

Activo vs. Pasivo

En los algoritmos de SLAM pasivos, es otra entidad quien se encarga de controlar el robot, mientras que el algoritmo de SLAM es puramente observador. La gran mayoría de los algoritmos de SLAM son de este tipo.

En el caso de los algoritmos de SLAM activo, el robot explora de forma activa el entorno en busca de conseguir un mapa más preciso en el menor tiempo posible. Existen técnicas híbridas donde el algoritmo de SLAM solo controla la dirección de los sensores y otra entidad se encarga de la dirección del movimiento del robot.

Estático vs. Dinámico

En el caso del SLAM estático se asume que el entorno no cambia con el tiempo, a diferencia de los métodos dinámicos que sí lo hacen. La gran mayoría de la literatura asume entornos estáticos

Volumétrico vs. Basado en marcas

En SLAM volumétrico, el mapa es muestreado a una resolución que permite una reconstrucción fotográfica del entorno. En este caso el costo computacional es alto. Por otro lado, en el SLAM basado en marcas se extraen características de las medidas de los sensores de forma de armar el mapa en base a marcas dispersas.

Las técnicas que utilizan SLAM basado en marcas suelen ser más eficientes ya que se descarta información de los sensores.

SLAM con un solo robot vs. Multirobot

La gran parte de los problemas están definidos para un solo robot, sin embargo, recientemente el problema de trabajar con más de un robot ha ganado mucha popularidad.

Los problemas para multirobots vienen de muchas maneras. En algunos casos los robots son capaces de observarse entre ellos.

También se distinguen por el tipo de comunicación que utilizan. Los más realistas permiten que solamente los robots que están más cerca se puedan comunicar.

Manejo de mucha incertidumbre vs. Poca incertidumbre

Estos algoritmos se distinguen por la cantidad de incertidumbre que pueden manejar. Los más simples solo pueden manejar poca incertidumbre en la localización. Son útiles para los casos en los cuales un camino no se intersecta a sí mismo.

En cambio para los mapas los cuales tienen lugares que se pueden alcanzar de muchas maneras, es necesario que el robot pueda manejar mucha incertidumbre en su posición.

La incertidumbre puede ser disminuida si el robot puede recibir información sobre su posición de forma absoluta. Un ejemplo puede ser mediante el uso sistema de posicionamiento global (GPS).

2.2.10 Fundamentos Teóricos

El problema del SLAM puede dividirse en varios módulos o subproblemas, siendo cada uno de estos un campo de investigación en sí.

Existen dos grandes enfoques de la solución del SLAM, Bioinspirados y Probabilísticos, que determinan la naturaleza de la solución de cada subproblema del SLAM

Estos enfoques difieren principalmente en la forma en que procesan la información de entrada de forma de encontrar una buena estimación de la ubicación del robot y mapa del entorno.

Actualmente, el enfoque probabilístico domina el campo y ha logrado implementaciones que escalan a ambientes grandes y complejos.

Sin embargo, la gran capacidad de navegar que poseen mamíferos como las ratas, simios y seres humanos han motivado la investigación y

desarrollo de sistemas que imitan los mecanismos biológicos de navegación de estos animales. A continuación se incluye una descripción del enfoque bioinspirado seguida del probabilístico.

2.2.11 Bioinspirados

Desde la década de 1970 el entendimiento del funcionamiento del cerebro mamífero vinculado a las actividades de navegación y confección de mapas del entorno ha ido aumentando. Sin embargo, descubrimientos a partir del año 2005 en adelante han cambiado la forma en que se conciben los mecanismos mentales utilizados por los mamíferos para estas actividades.

Estos descubrimientos corresponden al hallazgo de células especializadas en las actividades de navegación y reconocimiento del entorno, principalmente ubicadas en el hipotálamo. Estas células evidencian la existencia de una representación interna del entorno en el que el mamífero se desplaza.

2.2.12 Células de lugar

El primer gran hallazgo corresponde al descubrimiento de las células de lugar o "place cells". Las células de lugar son neuronas en el hipocampo que presentan una razón alta de disparo cuando el animal se encuentra en una posición específica del entorno que se corresponde con el campo de la célula de lugar.

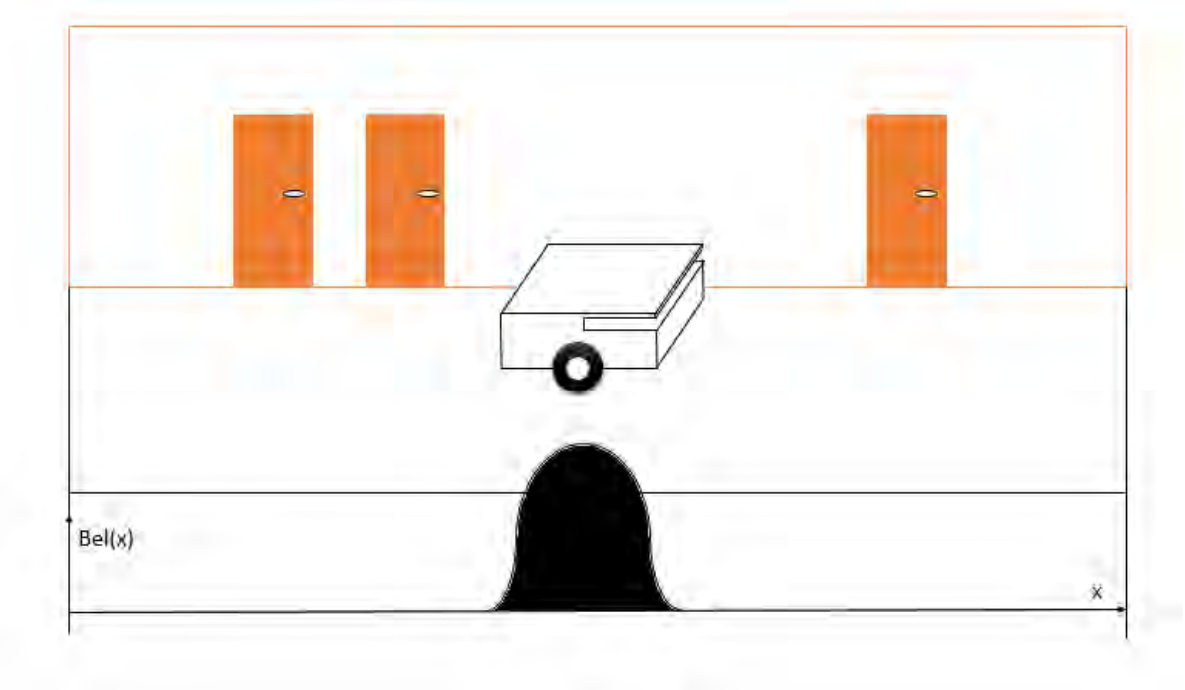
Es decir, que cada una de estas células estará asociada a una determinada zona del espacio, y se disparará cuando el animal se

encuentre en dicha zona. La existencia de estas células plantea la hipótesis de que el hipocampo mantiene un mapa o representación interna del entorno.

2.2.13 SLAM Probabilísticos

Los métodos probabilísticos se concentran en encontrar la distribución de probabilidad de la posición del robot y del mapa del entorno a través del tiempo. Para ilustrar este concepto de estimación como una distribución de probabilidad, resulta conveniente analizar la Figura 20 propuesta por Thrun.

En esta figura se puede observar un robot navega en un entorno unidimensional con tres puertas como únicos objetos distinguibles. En la parte inferior se despliega la estimación de la posición del robot como una distribución de probabilidad, en este caso una distribución gaussiana, donde se le asigna a cada intervalo del eje x una probabilidad de que el robot se encuentre en dicho intervalo.



(Figura 20)

Luego, se definen tres variables estocásticas involucradas en el modelo de SLAM probabilista:

- **xi** modela el valor a estimar en el instante de tiempo i. Para el caso del SLAM, esta variable puede representar la posición del robot, el mapa, o ambos. En ocasiones, esta variable se separa en la posición del robot xi y el mapa mi , en otras xi involucra a ambos conceptos. Esta variable suele recibir el nombre de "estado oculto" debido a que su valor no es directamente observable y, por eso, debe estimarse en función de las restantes variables observables.
- **zi** modela la observación realizada en el momento i. Esta observación puede constar de una medida, un conjunto de medidas

o una observación de alto nivel, como puede ser "el objeto distinguible 1 está a 3 metros de distancia y a 30 grados".

- **ui** modela los controles de movimiento emitidos en el robot al momento i. Algunos ejemplos son la velocidad del robot en un instante dado, la velocidad enviada a cada motor, o la distancia a recorrer. Otra fuente de información que es modelada utilizando esta variable es la información de sensores introspectivos como los encoders de las ruedas o sensores de inercia, que brindan nociones del movimiento propio al robot. Esta variable recibe varios nombres en la literatura, como por ejemplo información de controles o información de odometría.

El tiempo t suele tomarse como un conjunto discreto de instantes, que suelen coincidir con los momentos en los que el robot recibe información. Los subindices en las variables estocásticas xi , zi y ui corresponden al instante de tiempo al que corresponden estas variables. Tomando todo esto en cuenta, la probabilidad a estimar para solucionar el problema de SLAM puede formularse como:

$$p(x_{1:t}|z_{1:t},u_{1:t})$$

Es decir, la distribución de probabilidad de estado x1:t que se adapta mejor a las observaciones z1:t y controles u1:t . Alternativamente se puede encontrar en la literatura esta probabilidad representada como:

$$Bel_{(x_{1:t}|z_{1:t},u_{1:t})}$$

Se utiliza para hacer énfasis en el hecho de que es la creencia (belief) del robot sobre el estado del sistema.

En conclusión, resolver el problema de SLAM implicará estimar la distribución de probabilidad para la variable que representa el estado del sistema xi en cada instante de tiempo discreto del 1 al t, es decir la variable x1:t.

2.2.14 Representación del mapa

Un algoritmo de SLAM puede llevar una representación o mapa de su ambiente de diversas maneras. Cada uno de estos tipos de mapas tienen sus ventajas y desventajas asociadas. Al momento de elegir un tipo de mapa a implementar, se deben tener en cuenta las siguientes preguntas:

- ¿Es necesario mantener en el mapa nociones métricas?
- ¿Cuál es el uso que se le dará al mapa?
- ¿El entorno es dinámico?
- ¿De qué sensores se dispone?
- ¿De cuánto poder de cómputo se dispone?

3 Memoria descriptiva

3.1 Datos de partida

3.1.1 Explicación de los datos de partida

Como datos de partida se utilizaran los proporcionados por el fabricante del robot RobuLAB10, Robosoft, en sus hojas de características y planos.

Se utilizaran también referencias e información proporcionadas por Microsoft en MSDN, su centro de ayuda, para todo lo referente al software, esto es, Microsoft Robotics Develop Studio.

Ahora bien, para tener una estructura de los datos de partida a utilizar, los organizaremos de la siguiente manera

- Hardware
    - RobuLAB10
        - Hoja de características
        - Dimensiones del robot
            - Planos con las cotas correspondientes
        - Movimiento del robot
        - Sensores
            - Laser
            - Ultra sonidos
            - "bumpers"
            - Encoders

- Cámara

- Software
  - MRDS
    - C#
    - Paquete RobuBOX
  - CCR y DSS
  - MATLAB
    - Image Adquisition Toolbox

Cabe incluir que aquí se comentarán todos los datos que se han utilizado como punto de partida para el comienzo del proyecto, y que algunos de ellos estarán desarrollados en profundidad en los diversos anejos que se han incluido.

Apoyándose en estos datos y procediendo al estudio del algunos de ellos se podría comenzar con el desarrollo de las aplicaciones elaboradas en este proyecto para la exploración y navegación en entornos no estructurados ni conocidos de antemano, y el reconocimiento de objetos situados aleatoriamente en ellos mediante visión artificial.

3.1.2 Datos del Hardware utilizado

3.1.2.1 Datos del Robot y hoja de características

Como ya se ha venido comentando, el robot utilizado es la plataforma móvil RobuLAB10, desarrollada y fabricada por Robosoft principalmente para trabajos de investigación (figura 21).

(figura 21)

RobuLAB10 es una plataforma móvil para entornos cerrados principalmente, con dos ruedas propulsoras (tipo de movimiento diferencial). La plataforma está equipada con un PC (Windows XP ED) y se programa usando RobuBOX y MRDS.

Tabla de datos técnicos:

Dimensiones	L x l x h = 450 x 400 x 252.5 mm
Distancia hasta el suelo	76.5 mm
Peso	23 kg
Número de ruedas	2 ruedas propulsoras, 2 ruedas de castor
Dirección	Tipo diferencial
Radio de giro	Giro sobre sí mismo (punto medio de

	las ruedas de propulsión). Área necesaria definida por un diámetro de 552 mm
Velocidad Mínima	0.01 m/s
Velocidad Máxima	3 m/s
Desnivel	Máximo desnivel soportado: 11% sin carga
Carga máxima	30 kg
Temperatura de uso	0ºC – 50ºC
Temperatura de almacenamiento	0ºC – 60º C
Humedad	0- 90 % sin condensación
Baterías	4 baterías 24 VDC − 18 Ah
Autonomía	Aproximadamente 4h (dependiendo del uso)
Controlador empotrado	robuBOX
Sistema Operativo	Windows XPe
Modo de navegación	Xbox 360 Wirless Gamepad, interface gráfico, autónomo mediante programación de servicios

Panel de control (figura 22):

(figura 22)

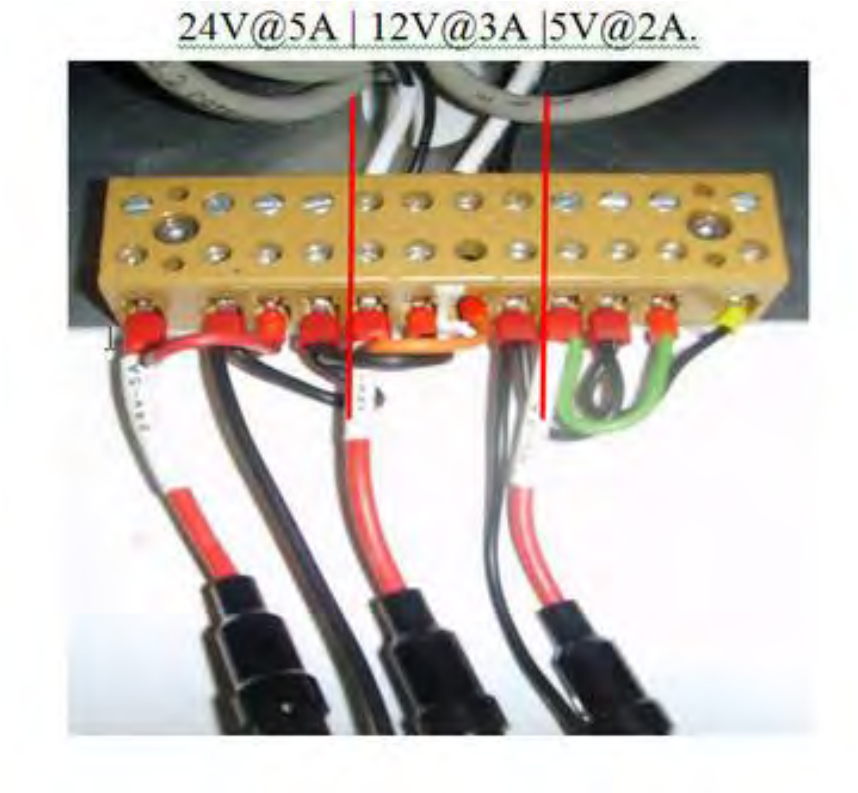
Panel de control del robot robuLAB10. Fuente: http://www.doc-center.robosoft.com/

Nº	Descripción	Función
1	On / Off cámara	Enciende o apaga la cámara CMUcam 3
2	MIC	Permite enchufar un micrófono
3	Aux IN	Permite enchufar un aparato auxiliar
4	L S	Permite enchufar un altavoz
5	PS2	Permite enchufar un aparato con conexión PS2, como un teclado o un ratón
6	Reset de la Cámara	Resetea la cámara CMUcam 3
7	USB	Permite conectar dispositivos con conexión

		USB
8	VGA	Permite conectar dispositivos con conexión VGA, tales como monitores
9	RJ45	Provee acceso a la conexión Ethernet del robot
10	Sensor de US trasero	Sensor de ultra sonidos trasero del robot
11	Interruptor On / Off del robot	Interruptor utilizado para encender o apagar el robot, cuando esta encendido, el led del interruptor esta encendido
12	Conector de 5V	Conector utilizado pare tener acceso a 5V@2A
13	Conector de 24V	Conector utilizado pare tener acceso a 24V@5ª
14	Conector para carga de las baterías	Conector utilizado para recargar las baterías del robot.
15	Fusible de 5V	Fusible de 2A para 5V
16	Fusible de 24V	Fusible de 5A para 24V
17	Led indicador del nivel de carga de las baterías	El led de 3 estados del nivel de la batería da las siguientes indicaciones: Verde: las baterías están totalmente cargadas Naranja: las baterías tienen un nivel de varga bajo y deberían ser recargadas

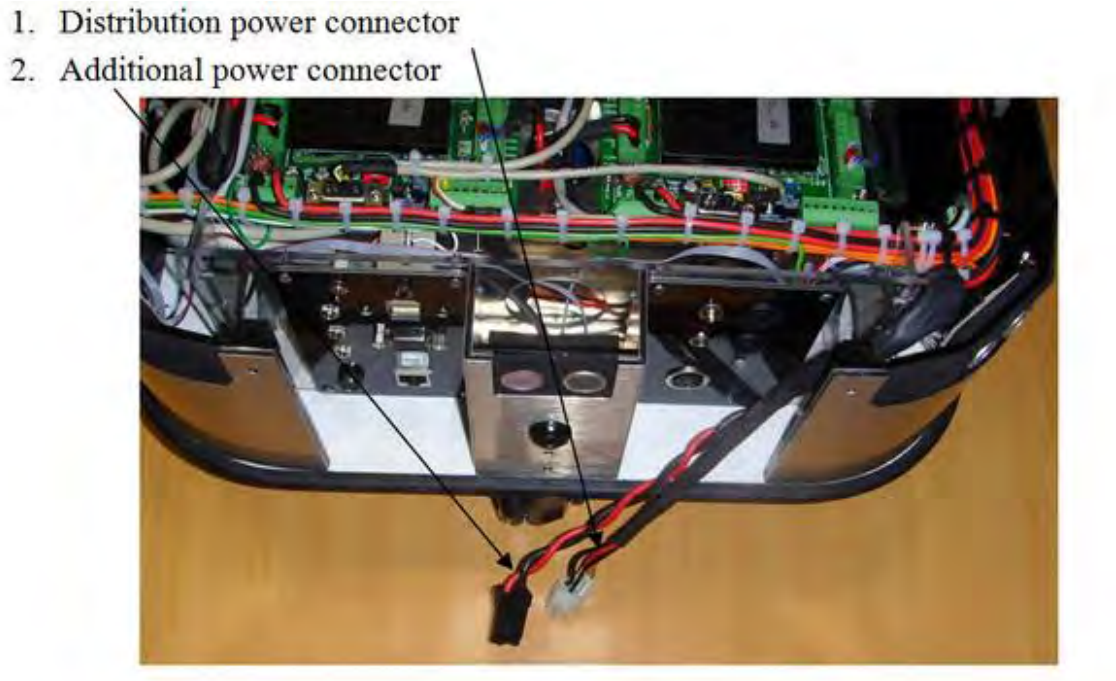
		Rojo: las baterías tienen un nivel de carga muy bajo y deben ser recargadas

Compartimento de extensión (figura 23 y 24):



(figura 23)

Compartimento de extensión del robot. Fuente: http://www.doc-center.robosoft.com/



(figura 24)

Vista de los conectores. Fuente: http://www.doc-center.robosoft.com/

El conector negro da potencia por medio de las baterías del compartimento de extensión. El blanco permite tener 5V (cable verde), 12V (cable naranja) y 24V (cable rojo) al compartimento de extensión.

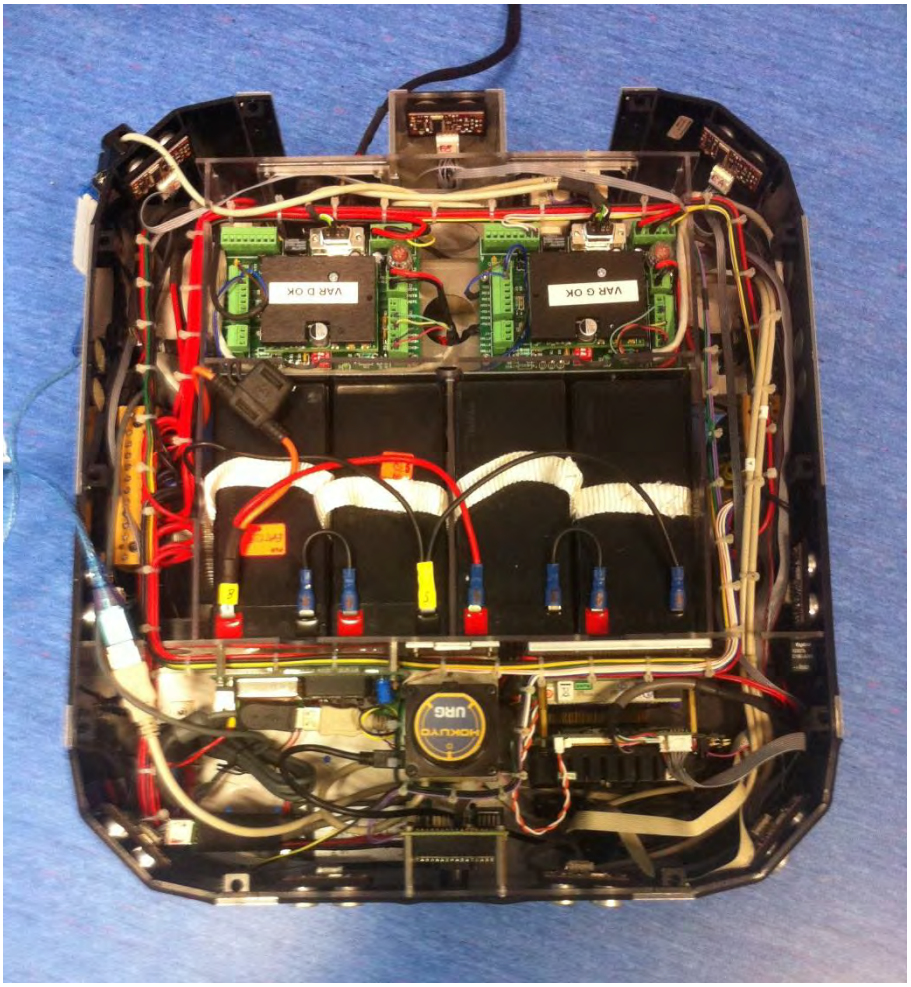
Guiado mediante el controlador Xbox 360 (figura 25):

(figura 25)

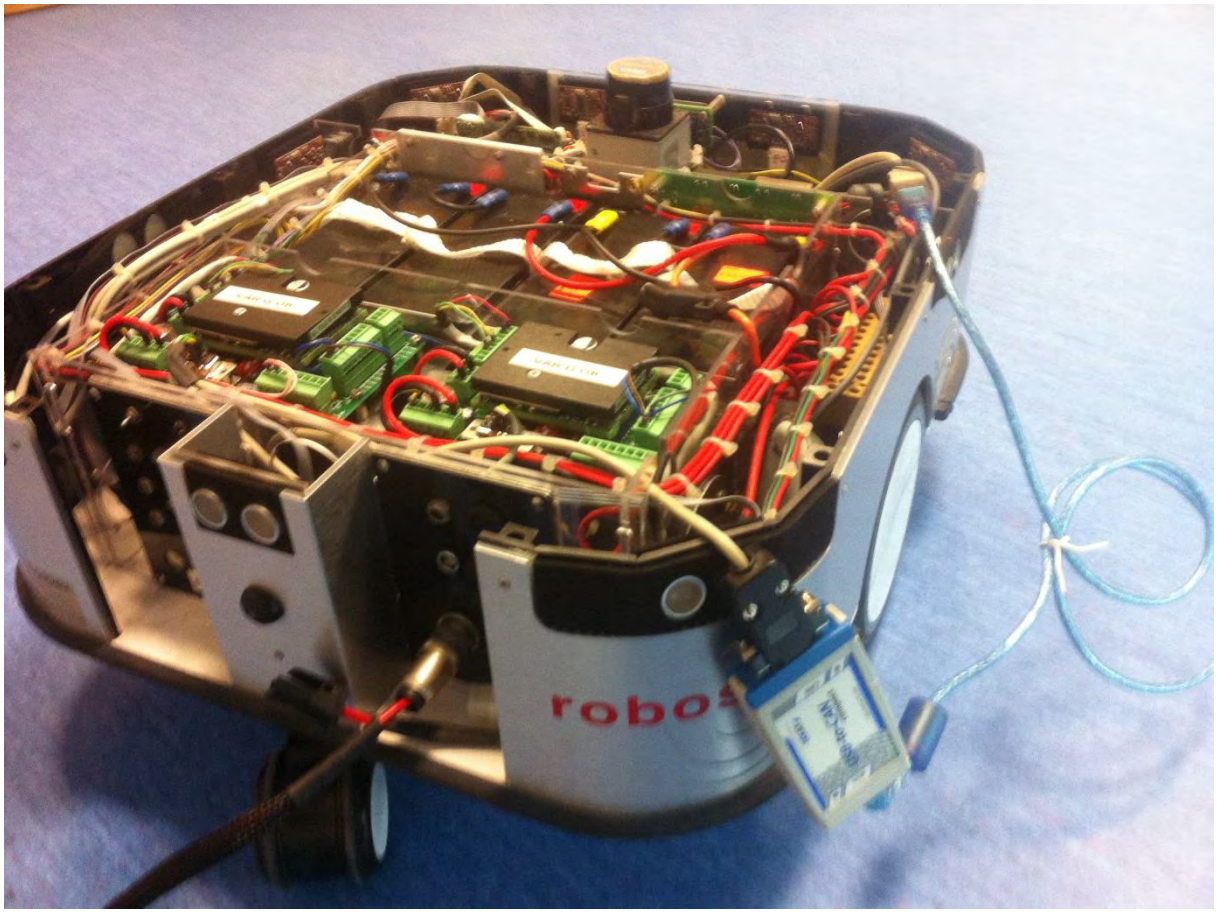
Controlador de la consola Xbox 360 y receptor para el manejo manual del robot. Fuente: http://www.doc-center.robosoft.com/

Acción	Botón
Ir hacia delante	RT
Ir hacia atrás	LT
Girar a la derecha	Derecha en el stick "pulgar" izquierdo
Girar a la izquierda	Izquierda en el stick "pulgar" izquierdo

Vista interna del robot (figura 26 y 27):



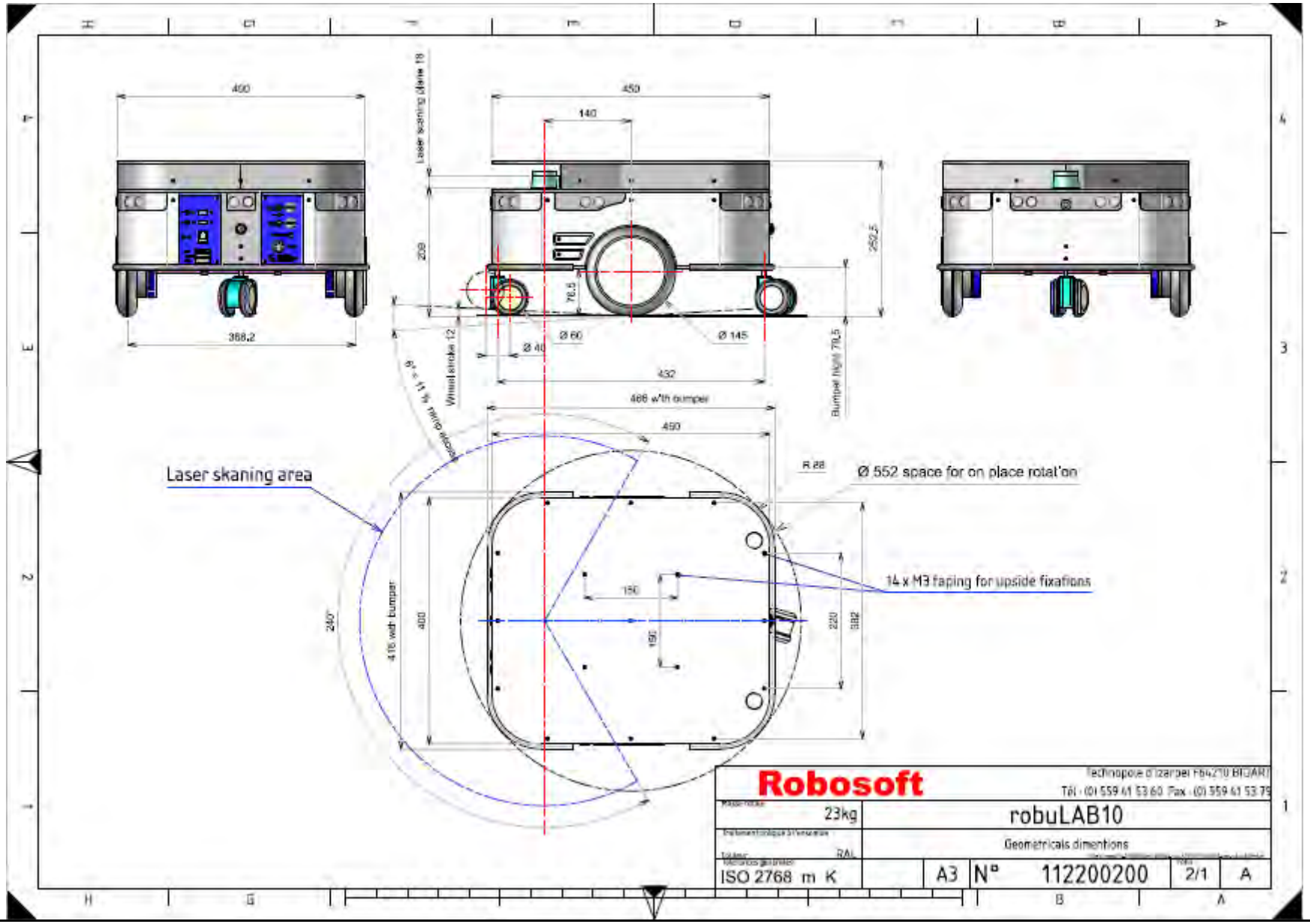
(figura 26)



(figura 27)

Dimensiones del robot:

Para poder apreciar realmente cuales son las dimensiones del robot, como se estructura su geometría, así como visualizar el área necesaria para el giro sobre sí mismo y el área que abarca el láser se añaden los planos técnicos del robot (figura 28):


490
368.2
Laser scaning plane 18
459
140
239
76.5
252.5
Ø 40
Ø 60
Ø 145
Wheel stroke 12
Bumper high 70.5
432
466 with bumper
490
R 28
Ø 552 space for on place rotat'on
Laser skaning area
14 x M3 taping for upside fixations
240°
416 with bumper
400
150
150
220
382
Robosoft
Technopole d'Izarbel 64210 BIDART
Tél : (0) 559 61 53 60 Fax : (0) 559 61 53 79
23kg
robuLAB10
Geometricals dimentions
RAL
ISO 2768 m K
A3
N° 112200200
2/1
A

(figura 28) planos técnicos del robot. Fuente: documentación RobuLAB10. "Technical drawins". Robosoft

El movimiento del robot es de tipo diferencial, por lo que el modelo cinemático se estructurará en torno a esta premisa.

El modelo completo con los detalles se desarrolla en uno de los anejos, de tal forma que así se pueda comprender el movimiento de la plataforma para así poder programar las trayectorias deseadas que queremos que el robot tome.

3.1.2.2 Sensores

Los sensores del robot y externos son:

- 9 sensores de ultra sonidos
- 1 laser de rango
- 2 "bumpers" (frontal y trasero)
- 2 encoders en las ruedas propulsoras
- 1 cámara interna (CMUcam 3)
- Una cámara externa

Sensores de ultra sonidos:

Los sensores de ultra sonidos contenidos en el robot son "Module OEM ultrason <<MSU08>>" (figura 29).

(figura 29) sensor de ultra sonidos Module OEM ultrason <<MSU08>>.
Fuente: Robooft, documentación técnica de robuLAB10

Sus características técnicas son los siguientes:

Rango de distancias	3cm – 6m
Células de ultrasonidos	2 por sensor
Comunicación con el micro controlador	I2C
Alimentación del modulo	Regulada y filtrada de + 5Vcc / 500 mA
Longitud hilos de conexión de la alimentación	No superior nunca a 10 cm

Laser de rango:

El láser de rango utilizado es el modelo "HOKUYO URG-04LX". (figura 30).

(figura 30) laser de rango HOKUYO URG-04LX. Fuente: Robosoft, documentación técnica RobuLAB10.

La hoja de características es la siguiente:

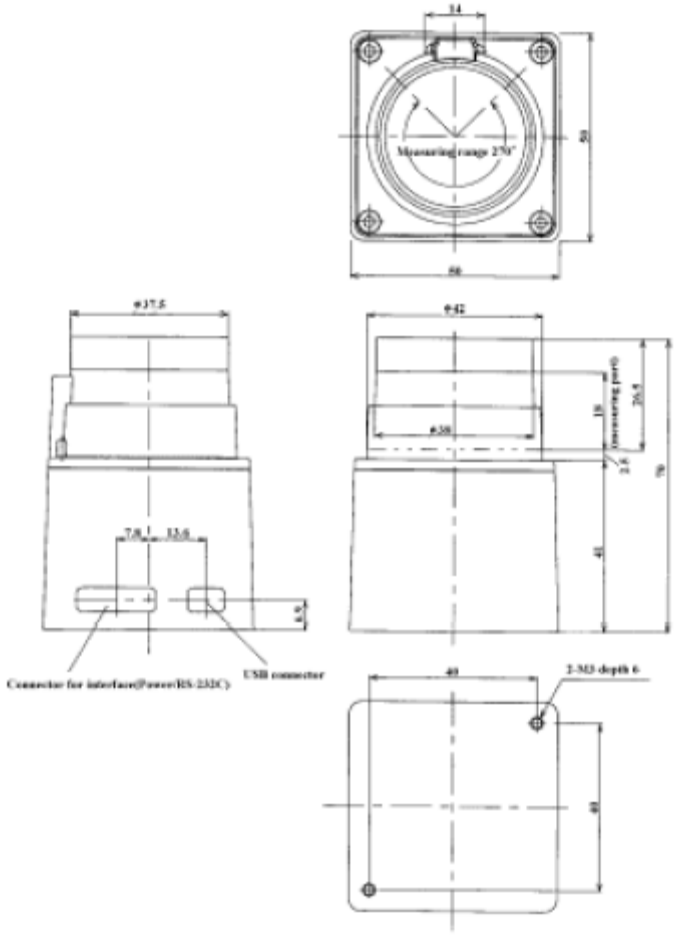
Dimensiones	L x W x H = 50 x 50 x 70 mm
Peso	160 g
Consumo de potencia	2.5 W
Precisión	+- 10 mm
Resolución	0.36º (360º/1024 pasos)

Ancho del área escaneada	240º

Especificaciones técnicas:

Fuente de luz	Semiconductor laser λ = 785 mm, seguridad laser clase 1 (IEC60825-1)
Fuente de alimentación	5 VDC, +-5%
Consumo máximo	500 mA
Distancia a objetos detectados	20 mm - 4000 mm
Precisión	20 – 1000 mm: +-10mm, 1000 – 4000mm: +-1% de la medida
Resolución	1 mm
Resolución del ángulo	0.36º (360º/1024 pasos)
Tiempo de escaneo	100 msec/scan
Interface	RS-232C(19.2k, 57.6k, 115.2k, 500k, 750kbps), USB: Ver 2.0 FS mode (12Mbps)
Temperatura ambiente/humedad	-10 - +50ºC, 85%RH o inferior (sin condensación ni congelación)
Estructura protectora	Optics: IP64, Case: IP40
Material	Policarbonato

Los dibujos técnicos del sensor se presentan a continuación (figura 31):



(figura 31) dibujos técnicos del sensor laser HOKUYO URG-04LX. Fuente: Robosoft, documentación técnica de ROBULAB10

Parachoques:

Los parachoques o bumpers del robot se activan una vez son pulsados y mandan un mensaje de alerta. Solo tienen 2 valores, pulsado o no pulsado (1 o 0), es decir, no tienen sensibilidad con distintas presiones.

No obstante, aunque solo posean dos posibles estados, los parachoques son independientes entre sí, pudiendo entonces diferenciar el robot entre si el parachoques pulsado es el anterior o el posterior.

Encoders:

Los encoders que el robot posee son el modelo "S4 Miniature Optical Shaft Encoder" (figura 32):

(Figura 32). Encoder S4 Miniature Optical Shaft Encoder. Fuente: Robosoft, documentación técnica de ROBULAB10

Su hoja de características es la siguiente:

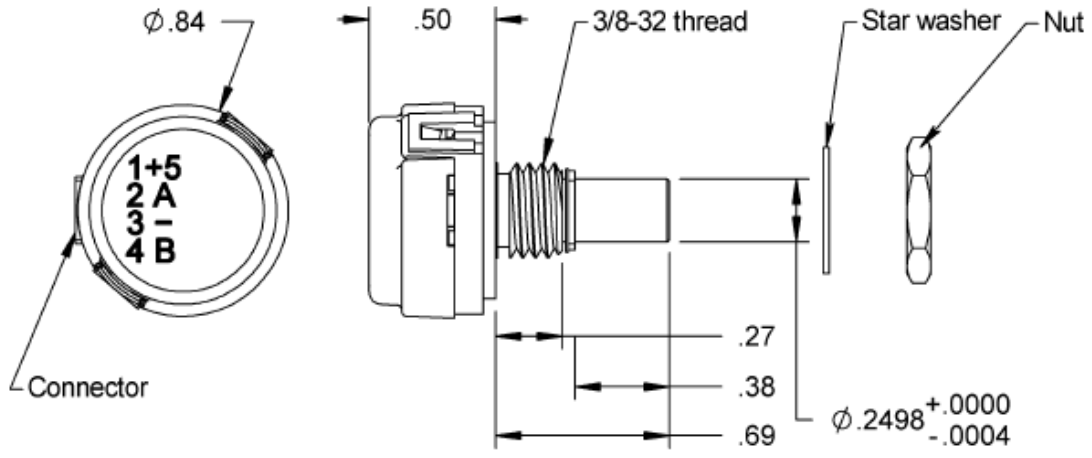
Ciclos captados por segundo	0 – 30000 ciclos/seg
Temperatura de uso	-10 - +85ºC
Ciclos por revolución	100 – 300 CPR
Pulsaciones por revolución	400 – 1200 PPR
Canales	2 canales TTL con outputs de forma de onda de forma cuadrada
Material del eje	Latón
Material del casquillo	Laton
Conector	Oro plateado

Diámetro del agujero	0.375 in. +0.005 − 0
Holgura del panel	0.125in. max
Momento máximo del panel	20 in. −lbs

Especificaciones mecánicas

Aceleración	10000 rad/seg^2
Vibración	20 g. 5 − 2 KHz
Velocidad del eje	100 RPM máx. Continuo
Momento del eje	0.5 +-0.2 in. oz 0.3 in. oz. Máx
Carga del eje	2 lbs. max. dinamico 20 lbs. max estatico
Peso	0.46 oz.

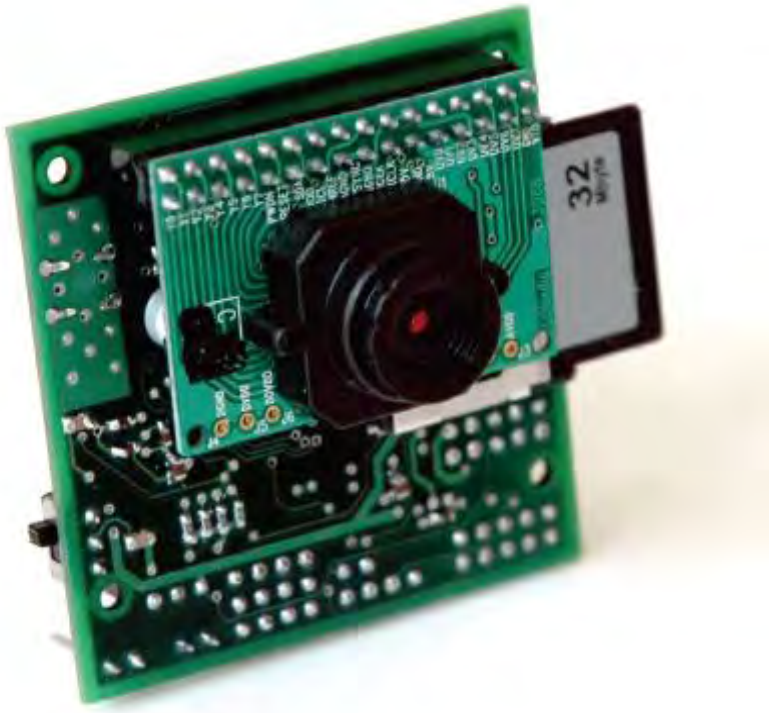
Dibujos técnicos del modelo mecánico (figura 33):



(figura 33) dibujos técnicos del modelo mecánico del encoder. Fuente: Robosoft, documentación técnica de RobuLAB10

Cámara interna:

La cámara interna que posee el robot es el modelo "CMUcam 3" (figura 34)



(figura 34) Cámara CMUcam 3. Fuente: Robosoft, documentación técnica de RobuLAB10

Hoja de características

Resolución	CIF resolution (352x288)
Sensor de color	Sensor de color RGB

Servo controlador	Servo controlador de 4 puertos
Frames	26 frames por segundo
Compresión	Software JPEG compression
Lenguajes admitidos	Interpreta LUA
Librerías	Arbitrari image Clipping Image Downsampling Mutable camera image properties Threshold and convolution functions
Video output	B/W analog video output (Pal o NTSC)
Procesado de imagen	FIFO image buffer
Wireless	Teleos 802.15.4

Cámara externa:

La cámara utilizada e incorporada al robot para la recogida de imagines desde un lateral del mismo es el modelo "Logitech HD Webcam C270" (figura 35).

(Figura 35) cámara Logitech HD Webcam C270. Fuente: http://img.misco.eu/

Sus características fundamentales son las siguientes:

- Previsualización HD (1280 x 720 píxeles)
- Captura de vídeo: Hasta 1280 x 720 píxeles
- Fotos: Hasta 3.0 megapíxeles

3.1.3 Datos del software utilizado

3.1.3.1 MRDS y robuBOX

Navegador es el nombre que se le ha puesto al servicio que controla el comportamiento de robuLAB10 en el desempeño de la navegación autónoma en entornos no estructurados, que no son conocidos de antemano, para que el mismo alcance los objetivos que se le han asignados, los cuales serán llegar a unas coordenadas "x" e "y" en el sistema de coordenadas que representa los puntos del espacio plano del entorno en el que se encuentra la plataforma móvil.

Este sistema de coordenadas viene dado por un origen que coincide con el centro del robot y el eje de ordenadas que tendrá la dirección a la que está orientado el mismo.

Las coordenadas se introducirán de forma manual, mediante interface humano-máquina (HMI), o mediante la localización de un objetivo concreto utilizando técnicas de visión artificial, las cuales se explicarán posteriormente.

El algoritmo de navegación del que se ha dotado al robot robuLAB10 ha sido desarrollado utilizando robuBOX.

RobuBOX kit de desarrollo de software abierto (SDK) desarrollado por Robosoft, y basado en Microsoft Robotics Developer Studio (MRDS). Esta desarrollado en el lenguaje de programación C#.

MSRS permite crear aplicaciones con arquitectura orientada a servicios, (SOA).

Un "servicio" básicamente tiene lo siguiente:

- Un estado estructurado
- Puertos de comunicación, los cuales se usan para recibir mensajes e información de otros servicios tanto dentro de un mismo programa, como de programas externos.
- Comportamientos que son ejecutados cuando se recibe información de los servicios seleccionados utilizando la función "interleave"
- Un conjunto de asociaciones (Partners). Son asociaciones con las que el servicio se relaciona.
- Un "contract" que resume las especificaciones del servicio y las publica de forma pública.

Estas partes las podremos ver de forma más concreta en el servicio navegador.

Los mensajes que llegan de los puertos de comunicación disparan la ejecución de los actuadores (handlers) asociados, generando las acciones que, por otra parte, están modificando el estado del sistema, produciendo a su vez comportamientos específicos que son los que se busca que tenga el robot en cada momento. Esta interacción entre cada servicio se realiza mediante los "contracts" de cada servicio ya mencionados. Estas interacciones incluyen las del tipo Get (tomar información), Replace (reemplazar información de un estado), Update (actualizar información de un estado), subscribe (conectarse a un servicio "ej: conectarse al servicio del láser de rango del robot da acceso a la información captada por el mismo"), Drop (desconectarse de un servicio) como los más importantes.

Todas las arquitecturas de control de robuBOX tienen una serie de características en común:

- **Servicios estructurados en capas:** Hay dos capas en las arquitecturas de control de robuBOX: una "capa" que corresponde al Hardware del sistema (motores, laser, US...) y la "capa" de aplicaciones. Esta separación se debe al hecho de que en los componentes robóticos del Hardware deben ser inicializados y manejados cuidadosamente.

- "**DevicesManager":** Este servicio "transversal" es de suma importancia y es el primero que ha de ser lanzado. Su función es la perfecta coordinación de las aplicaciones que se están manejando.

- **Interfaces entre servicios:** Muchos de los "contracts" de MRDS están definidos con robuBOX. El objetivo es permitir la definición de nuevas aplicaciones de forma rápida.

Como se acaba de explicar, la capa del Hardware se debe lanzar primero y ha de ser iniciada de forma apropiada y cuidadosamente.

El hecho por el que el servicio DevicesManager es tan importante es porque:

- Lanza la capa del Hardware y se asegura de que se inicie adecuadamente.
- Lanza los servicios de aplicaciones "encima" del Hardware.
- Monitoriza las operaciones de estos servicios de Hardware

3.1.3.2 Interacción con la plataforma móvil robuLAB10

La versión de robuBOX utilizada (robuBOX 6.1) funciona en Windows XPembedded. Para permitir el control de bajo nivel en tiempo real de las

funcionalidades principales, las aplicaciones de Robosoft usan controladores de motor inteligente a través de CanOpen bus, o micro controladores dependiendo de la complejidad de las funciones requeridas.

3.1.3.3 CCR Y DSS

MRDS funciona con CCR:

Concurrency and Coordination Runtime (CCR) es una "Dynamically Linked Library" (DLL) que está basada en .NET Framework de Microsoft, distribuida con Microsoft Robotics Developer Studio (MRDS).

CCR sacia la necesidad de las aplicaciones orientadas a servicios de ser manejadas con operaciones asíncronas, tratar con la concurrencia, la cual se explicará de forma clara con el servicio Navegador, aprovechar el paralelismo que tiene el Hardware del sistema, y tratar con fallos parciales. Permite diseñar aplicaciones en las que los módulos de software (o también componentes) puedan estar poco acopladas. Esto significa que puedan ser desarrolladas de forma independiente.

En esta línea, Decentralized Software Services (DSS) es un entorno de "ejecución" (runtime) ligero, obviamente también basado en .NET, que se ubica en la cumbre por decirlo de alguna manera de CCR.

DSS pone a disposición un modelo de servicio ligero orientado al estado que combina la noción de la transferencia del estado representacional "notion of representational state transfer (REST)" Con un sistema de alta eficiencia. En DSS, los servicios son expuestos como recursos accesibles a través de la programación y por manipulación "UI", la cual no es de nuestro interés en este proyecto.

Gracias a las herramientas de DSS que MRDS pone en nuestro alcance, podemos realizar el proceso completo desde crear el nuevo servicio, hasta hacerlo funcionar en nuestro robot objetivo. Dss ofrece una ventana de

comandos con una serie de funcionalidades, de entre las cuales las que se han utilizado para el desarrollo del proyecto son las siguientes:

- dssnewservice: Con este comando creamos un nuevo servicio en la carpeta seleccionada. El servicio lo podremos desarrollar con un entorno de programación. En este proyecto se ha optado por utilizar Microsoft Visual Studio 2008. La sintaxis del comando es la siguiente:
    - dssnewservice /service:nombre_del_servicio
- dssdeploy: con este empaquetamos en un ejecutable los archivos que contiene la carpeta de nuestro proyecto. Este ejecutable es el que debemos transferir a nuestro robot mediante usb o vía wi-fi. Cuando se ejecuta el .exe (executable) obtenido se abrirá una ventana en la que se pide donde desempaquetar los archivos del servicio, dando lugar a 2 carpetas. Una "bin" en la que se encuentran los archivos .dll (Dynamically Linked Library) y otra "nombre de la carpeta" en la que se encuentra el archivo .xml, el llamado "manifest". La sintaxis es la siguiente:
    - dssdeploy/p/m:"ruta_del_proyecto\nombre_del_manifest.xml" nombre_del_servicio.exe
    - /p hace referencia a que la función de dssdeploy que se va a utilizar es el packing (empaquetado)
- Dsshost: es el comando con el que ejecutamos un archivo .dll, .xml o el contract (de forma más habitual se utiliza el contract). Lo que se hace es ejecutar un DSS Node en un puerto y un puerto tcp especificados. Realmente este comando no es necesario ya que como se explicará a continuación, una vez se ha lanzado la capa transversal DevicesManager, se puede acceder a través del

navegador a una página local de intranet (en la que no es necesaria conexión a internet) en la que se pueden acceder a diferentes funcionalidades. Una de ellas es el Control Panel (panel de control) en el cual se pueden lanzar servicios simplemente haciendo click en ellos. La sintaxix es:

- bin\dsshost -p:50000 -t:50001 -contract:http://www.robosoft.fr/2006/11/devicesmanager.html
- este commando ejecutaría el servicio devices manager
- −p:50000 indica que se va a utilizar el puerto 50000
- −t:50001 indica que se va a utilizar el puerto tcp 50001
- −c: o −contract: indica que se va a indicar el contract
- Para especificar archivos dll se escribe −d:
- Para archivos xml se escribe −m:

3.1.3.4 Matlab

MATLAB (abreviatura de MATrix LABoratory, "laboratorio de matrices") es una herramienta de software matemático que ofrece un entorno (IDE) con un lenguaje de programación propio (lenguaje M) y servicio de especie.

El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multi-dominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes).

En este proyecto, la versión de MATLAB utilizada es la 2013b, y se hace imprescindible para la realización de las aplicaciones aquí mostradas la toolbox Image Acquisition.

Esta toolbox captura video e imágenes desde hardware industrial standard.

Image Acquisition permite capturar estas imágenes y video directamente en MATLAB y Simulink. Puede detectar el hardware automáticamente y configurar propiedades del mismo.

Trabajos más avanzados permiten realizar esta captura durante el procesado, realizar capturas del fondo y sincronizar muestreo a través de varios dispositivos multimodales. Con soporte para múltiples proveedores de hardware y estándares de la industria, puede utilizar dispositivos de imágenes, que van desde cámaras Web baratas para "high-end" dispositivos científicos e industriales que cumplan con poca luz, de alta velocidad y otros requisitos difíciles.

3.2 El servicio navegador

3.2.1 Introducción al servicio navegador

El servicio navegador cuenta con numerosas versiones, desde la 1.0 hasta la 3.0

El cambio más significativo fue el experimentado del salto de la versión 1.9.2 a la 2.0, en el que la estructura y la forma de operar del servicio cambia casi por completo. Cabe destacar la versión 1.9.2, que se presenta como una alternativa de funcionamiento del programa Navegador, y la versión final, la 3.0.

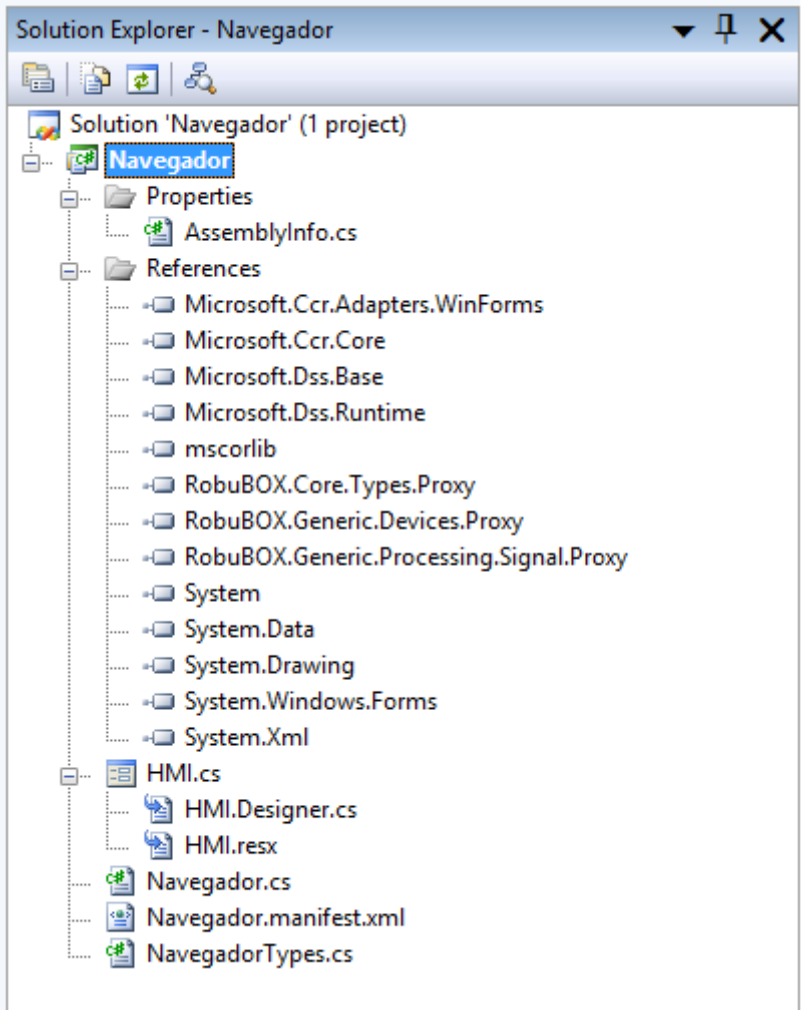
A continuación se explicarán dichas versiones, empezando por la versión final, y repasando finalmente la alternativa propuesta por la versión 1.9.2.

3.2.2 VERSION 3.0

3.2.2.1 EXPLORADOR DE SOLUCIONES

En esencia, el servicio navegador es un programa que nos permite navegar por un entorno no estructurado ni conocido de antemano para llegar a unas coordenadas requeridas de forma externa, evitando la colisión con obstáculos intermedios que impidan la llegada al destino del robot.

En el explorador de soluciones de Visual Studio (figura 36), podemos ver las diferentes partes que componen el servicio navegador. Estas son las siguientes:



(figura 36)

Solution 'Navegador' contiene el proyecto navegador, que a su vez contiene los ficheros de propiedades y de referencias, así como el HMI (human machine interface) con el que nos comunicamos con el programa, y los archivos navegador.cs, navegadorTypes.cs y navegador.manifest.xml.

3.2.2.2 PROPERTIES

En el fichero properties podemos encontrar el archivo AssemblyInfo.cs, el cual se genera automáticamente con la herramienta anteriormente mencionada "dssnewservice"

Este archivo contiene el siguiente código:

```
using System.Reflection;
using dss = Microsoft.Dss.Core.Attributes;
using interop = System.Runtime.InteropServices;

[assembly:
dss.ServiceDeclaration(dss.DssServiceDeclaration.ServiceBehavi
or)]
[assembly: interop.ComVisible(false)]
[assembly: AssemblyTitle("Navegador")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyProduct("Navegador")]
[assembly: AssemblyCopyright("")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("1.0.0.0")]
```

3.2.2.3 REFERENCES

En references, y como su propio nombre indica, debemos añadir todas las referencias que nuestro programa va a usar. En la primera parte declarativa de nuestro código, como posteriormente veremos, todas las declaraciones "using" deben tener una referencia en el programa que puedan usar.

Las referencias que se han incluido en el programa navegador son:

- Referencias por defecto:
  - Microsoft.Dss.Base
  - Microsoft.Dss.Runtime

- Mscorlib
- System
- System.Data
- System.Xml

Estas referencias son necesarias para el funcionamiento del programa y también son incluidas con la herramienta "dssnewservice" por lo que no se hará más incapié en ellas al no ser de interés en el este proyecto en concreto.

- Referencias de Windows Forms:
  - Microsoft.Ccr.Adapters.WinForms
  - Microsoft.Ccr.Core
  - System.Drawing
  - System.Windows.Forms

Todas estas referencias han sido incluidas para poder visualizar y utilizar el HMI de manera que se permita un fácil manejo del software pudiendo incluir las coordenadas destino del robot a mano por el usuario con una ventana emergente interactiva que se mostrará más adelante, en la presentación del HMI

- Referencias de RobuBOX
  - RobuBOX.core.Types.Proxy
  - RobuBOX.Generic.Devices.Proxy
  - RobuBOX.Services.Processing.Signal.Proxy

Estas referencias se utilizan para utilizar funcionalidades del paquete de RobuBOX.

Con "RobuBOX.core.Types.Proxy" podemos utilizar la clase "types.Point2D" con la que creamos una variable que corresponde a un punto en un sistema de coordenadas cartesiano.

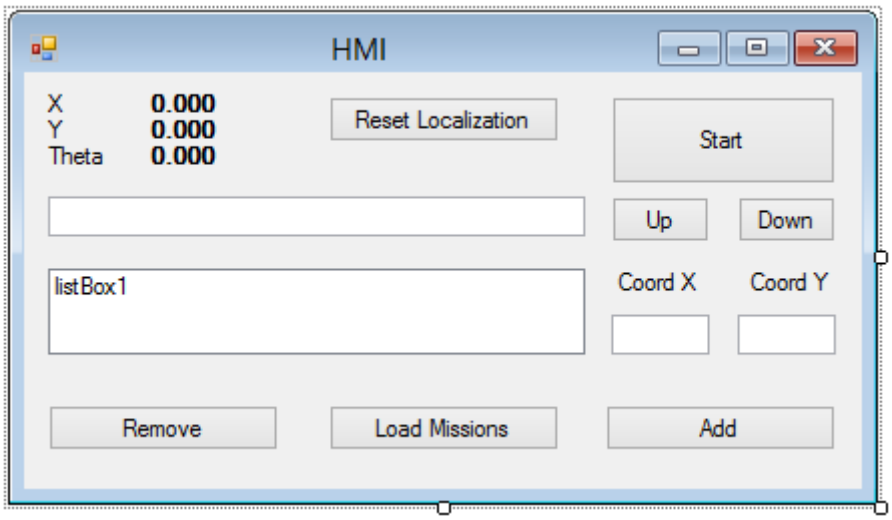
Con "RobuBOX.Generic.Devices.Proxy" obtenemos acceso tanto al laser de rango que vamos a utilizar para evitar obstáculos, como al servicio del control diferencial del robot, el cual es el encargado de asignar una velocidad lineal y angular al mismo y proveerle de movimiento.

Por último, con "RobuBOX.Services.Processing.Signal.Proxy" obtenemos acceso al servicio de localización del robot, el cual nos determinará la posición del mismo en un sistema de coordenadas cartesiano en el que el origen es la posición inicial del robot, y los ejes x e y serán la dirección perpendicular del robot y la dirección paralela al mismo respectivamente, de forma que el eje z quedaría situado apuntando hacia el suelo. Este servicio también nos da la posibilidad de obtener la orientación del robot, dándonos un ángulo medido en la dirección contraria a las agujas del reloj siendo 0 el eje x. La localización del robot se lleva a cabo mediante odometría, integrando la velocidad de las ruedas del robot gracias a los encoders que lleva instalados.

Más adelante, cuando se explique cada una de las partes del código del servicio, se explicará cómo funcionan cada uno de estos servicios

3.2.2.4 HMI

El aspecto de la ventana del HMI (figura 37) es el siguiente:



(figura 37)

El código completo del diseño del HMI.Designer.cs es el siguiente:

```
namespace Navegador
{
    partial class HMI
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components =
null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
  /// <param name="disposing">true if managed resources
should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
```

```
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not
modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.textBox2 = new
System.Windows.Forms.TextBox();
            this.textBox3 = new
System.Windows.Forms.TextBox();
            this.label1 = new System.Windows.Forms.Label();
            this.listBox1 = new
System.Windows.Forms.ListBox();
            this.button3 = new System.Windows.Forms.Button();
            this.button4 = new System.Windows.Forms.Button();
            this.button5 = new System.Windows.Forms.Button();
            this.button6 = new System.Windows.Forms.Button();
            this.label3 = new System.Windows.Forms.Label();
            this.button7 = new System.Windows.Forms.Button();
            this.textBox4 = new
System.Windows.Forms.TextBox();
            this.buttonResetOdo = new
System.Windows.Forms.Button();
            this.valx = new System.Windows.Forms.Label();
            this.valy = new System.Windows.Forms.Label();
            this.theta = new System.Windows.Forms.Label();
            this.xval = new System.Windows.Forms.Label();
            this.yval = new System.Windows.Forms.Label();
            this.thetaval = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Location = new
System.Drawing.Point(293, 12);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(112,
44);
            this.button1.TabIndex = 0;
```

```
            this.button1.Text = "Start";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new
System.EventHandler(this.button1_Click);
            //
            // textBox2
            //
            this.textBox2.Location = new
System.Drawing.Point(12, 62);
            this.textBox2.Name = "textBox2";
            this.textBox2.Size = new System.Drawing.Size(268,
20);
            this.textBox2.TabIndex = 3;
            //
            // textBox3
            //
            this.textBox3.Location = new
System.Drawing.Point(293, 121);
            this.textBox3.Name = "textBox3";
            this.textBox3.Size = new System.Drawing.Size(49,
20);
            this.textBox3.TabIndex = 6;
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new
System.Drawing.Point(294, 98);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(45,
13);
            this.label1.TabIndex = 7;
            this.label1.Text = "Coord X";
            //
            // listBox1
            //
            this.listBox1.FormattingEnabled = true;
            this.listBox1.Location = new
System.Drawing.Point(12, 98);
            this.listBox1.Name = "listBox1";
            this.listBox1.Size = new System.Drawing.Size(268,
43);
            this.listBox1.TabIndex = 8;
            //
            // button3
```

```
            //
            this.button3.Location = new
System.Drawing.Point(290, 166);
            this.button3.Name = "button3";
            this.button3.Size = new System.Drawing.Size(115,
23);
            this.button3.TabIndex = 9;
            this.button3.Text = "Add";
            this.button3.UseVisualStyleBackColor = true;
            this.button3.Click += new
System.EventHandler(this.button3_Click);
            //
            // button4
            //
            this.button4.Location = new
System.Drawing.Point(12, 166);
            this.button4.Name = "button4";
            this.button4.Size = new System.Drawing.Size(115,
23);
            this.button4.TabIndex = 10;
            this.button4.Text = "Remove";
            this.button4.UseVisualStyleBackColor = true;
            this.button4.Click += new
System.EventHandler(this.button4_Click);
            //
            // button5
            //
            this.button5.Location = new
System.Drawing.Point(293, 62);
            this.button5.Name = "button5";
            this.button5.Size = new System.Drawing.Size(49,
23);
            this.button5.TabIndex = 11;
            this.button5.Text = "Up";
            this.button5.UseVisualStyleBackColor = true;
            this.button5.Click += new
System.EventHandler(this.button5_Click);
            //
            // button6
            //
            this.button6.Location = new
System.Drawing.Point(356, 62);
            this.button6.Name = "button6";
            this.button6.Size = new System.Drawing.Size(49,
23);
```

```
            this.button6.TabIndex = 12;
            this.button6.Text = "Down";
            this.button6.UseVisualStyleBackColor = true;
            this.button6.Click += new
System.EventHandler(this.button6_Click);
            //
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.Location = new
System.Drawing.Point(360, 98);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(45,
13);
            this.label3.TabIndex = 14;
            this.label3.Text = "Coord Y";
            //
            // button7
            //
            this.button7.Location = new
System.Drawing.Point(152, 166);
            this.button7.Name = "button7";
            this.button7.Size = new System.Drawing.Size(115,
23);
            this.button7.TabIndex = 15;
            this.button7.Text = "Load Missions";
            this.button7.UseVisualStyleBackColor = true;
            this.button7.Click += new
System.EventHandler(this.button7_Click);
            //
            // textBox4
            //
            this.textBox4.Location = new
System.Drawing.Point(356, 121);
            this.textBox4.Name = "textBox4";
            this.textBox4.Size = new System.Drawing.Size(49,
20);
            this.textBox4.TabIndex = 19;
            //
            // buttonResetOdo
            //
            this.buttonResetOdo.Location = new
System.Drawing.Point(152, 12);
            this.buttonResetOdo.Name = "buttonResetOdo";
```

```
            this.buttonResetOdo.Size = new
System.Drawing.Size(115, 23);
            this.buttonResetOdo.TabIndex = 20;
            this.buttonResetOdo.Text = "Reset Localization";
            this.buttonResetOdo.UseVisualStyleBackColor =
true;
            //
            // valx
            //
            this.valx.AutoSize = true;
            this.valx.Location = new System.Drawing.Point(9,
9);
            this.valx.Name = "valx";
            this.valx.Size = new System.Drawing.Size(14, 13);
            this.valx.TabIndex = 21;
            this.valx.Text = "X";
            //
            // valy
            //
            this.valy.AutoSize = true;
            this.valy.Location = new System.Drawing.Point(9,
22);
            this.valy.Name = "valy";
            this.valy.Size = new System.Drawing.Size(14, 13);
            this.valy.TabIndex = 22;
            this.valy.Text = "Y";
            //
            // theta
            //
            this.theta.AutoSize = true;
            this.theta.Location = new System.Drawing.Point(9,
35);
            this.theta.Name = "theta";
            this.theta.Size = new System.Drawing.Size(35, 13);
            this.theta.TabIndex = 23;
            this.theta.Text = "Theta";
            //
            // xval
            //
            this.xval.AutoSize = true;
            this.xval.Font = new
System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
```

```
            this.xval.Location = new System.Drawing.Point(60,
9);
            this.xval.Name = "xval";
            this.xval.Size = new System.Drawing.Size(39, 13);
            this.xval.TabIndex = 24;
            this.xval.Text = "0.000";
            //
            // yval
            //
            this.yval.AutoSize = true;
            this.yval.Font = new
System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.yval.Location = new System.Drawing.Point(60,
22);
            this.yval.Name = "yval";
            this.yval.Size = new System.Drawing.Size(39, 13);
            this.yval.TabIndex = 25;
            this.yval.Text = "0.000";
            //
            // thetaval
            //
            this.thetaval.AutoSize = true;
            this.thetaval.Font = new
System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.thetaval.Location = new
System.Drawing.Point(60, 35);
            this.thetaval.Name = "thetaval";
            this.thetaval.Size = new System.Drawing.Size(39,
13);
            this.thetaval.TabIndex = 26;
            this.thetaval.Text = "0.000";
            //
            // HMI
            //
            this.AutoScaleDimensions = new
System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(418,
208);
            this.Controls.Add(this.thetaval);
```

```
            this.Controls.Add(this.yval);
            this.Controls.Add(this.xval);
            this.Controls.Add(this.theta);
            this.Controls.Add(this.valy);
            this.Controls.Add(this.valx);
            this.Controls.Add(this.buttonResetOdo);
            this.Controls.Add(this.textBox4);
            this.Controls.Add(this.button7);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.button6);
            this.Controls.Add(this.button5);
            this.Controls.Add(this.button4);
            this.Controls.Add(this.button3);
            this.Controls.Add(this.listBox1);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.textBox3);
            this.Controls.Add(this.textBox2);
            this.Controls.Add(this.button1);
            this.Name = "HMI";
            this.Text = "HMI";
            this.FormClosed += new
System.Windows.Forms.FormClosedEventHandler(this.HMI_FormClose
d);
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.TextBox textBox2;
        private System.Windows.Forms.TextBox textBox3;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.ListBox listBox1;
        private System.Windows.Forms.Button button3;
        private System.Windows.Forms.Button button4;
        private System.Windows.Forms.Button button5;
        private System.Windows.Forms.Button button6;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Button button7;
        private System.Windows.Forms.TextBox textBox4;
        public System.Windows.Forms.Button buttonResetOdo;
        private System.Windows.Forms.Label valx;
        private System.Windows.Forms.Label valy;
```

```
        private System.Windows.Forms.Label theta;
        private System.Windows.Forms.Label xval;
        private System.Windows.Forms.Label yval;
        private System.Windows.Forms.Label thetaval;
    }
}
```

Este código se puede generar de forma automática diseñando de forma visual el HMI, o escribiendo el código.

En este proyecto la ventana fue diseñada de forma visual, pero modificando algunas partes del código:

Los labels que indican la localización del robot fueron cambiados de nombre para facilitar su ubicación posteriormente en el Handler de la odometría y poder visualizar en pantalla las coordenadas en las que se encuentra el robot en cada momento:

```
//
            // valx
            //
            this.valx.AutoSize = true;
            this.valx.Location = new System.Drawing.Point(9,
9);
            this.valx.Name = "valx";
            this.valx.Size = new System.Drawing.Size(14, 13);
            this.valx.TabIndex = 21;
            this.valx.Text = "X";
            //
            // valy
            //
            this.valy.AutoSize = true;
            this.valy.Location = new System.Drawing.Point(9,
22);
            this.valy.Name = "valy";
            this.valy.Size = new System.Drawing.Size(14, 13);
            this.valy.TabIndex = 22;
            this.valy.Text = "Y";
            //
            // theta
```

```
            //
            this.theta.AutoSize = true;
            this.theta.Location = new System.Drawing.Point(9,
35);
            this.theta.Name = "theta";
            this.theta.Size = new System.Drawing.Size(35, 13);
            this.theta.TabIndex = 23;
            this.theta.Text = "Theta";
            //
            // xval
            //
            this.xval.AutoSize = true;
            this.xval.Font = new
System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.xval.Location = new System.Drawing.Point(60,
9);
            this.xval.Name = "xval";
            this.xval.Size = new System.Drawing.Size(39, 13);
            this.xval.TabIndex = 24;
            this.xval.Text = "0.000";
            //
            // yval
            //
            this.yval.AutoSize = true;
            this.yval.Font = new
System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.yval.Location = new System.Drawing.Point(60,
22);
            this.yval.Name = "yval";
            this.yval.Size = new System.Drawing.Size(39, 13);
            this.yval.TabIndex = 25;
            this.yval.Text = "0.000";
            //
            // thetaval
            //
            this.thetaval.AutoSize = true;
            this.thetaval.Font = new
System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
```

```
            this.thetaval.Location = new
System.Drawing.Point(60, 35);
            this.thetaval.Name = "thetaval";
            this.thetaval.Size = new System.Drawing.Size(39,
13);
            this.thetaval.TabIndex = 26;
            this.thetaval.Text = "0.000";
```

Por lo tanto, los labels que indican la coordenada x, y y theta pasan a llamarse xval, yval y theta, y los que indican el valor de las mismas valx, yval y thetaval. Ademas con la siguiente instrucción haremos que este en negrita:

```
this.thetaval.Font = new System.Drawing.Font("Microsoft Sans
Serif", 8.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
```

Además, se modificó también por el mismo motivo el nombre del botón que realiza el reset en las coordenadas, es decir, reinicia la posición del robot cambiando el sistema de coordenadas a uno nuevo que corresponde con la posición que tiene en ese mismo instante el robot:

```
            //
            // buttonResetOdo
            //
            this.buttonResetOdo.Location = new
System.Drawing.Point(152, 12);
            this.buttonResetOdo.Name = "buttonResetOdo";
            this.buttonResetOdo.Size = new
System.Drawing.Size(115, 23);
            this.buttonResetOdo.TabIndex = 20;
            this.buttonResetOdo.Text = "Reset Localization";
            this.buttonResetOdo.UseVisualStyleBackColor =
true;
```

Ahora pasamos a ver el código completo para la posterior explicación parte a parte del HMI.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Navegador
{
    public partial class HMI : Form
    {

        //Member
        HMIPort _hmiPort;

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="port"></param>
        public HMI(HMIPort port)
        {
            this._hmiPort = port;

            InitializeComponent();
        }

        /// <summary>
        /// Send new mission when the button1 is clicked
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button1_Click(object sender, EventArgs e)
        {
            this._hmiPort.Post(new StartMission());
        }

        /// <summary>
```

```
        /// Display a text box
        /// </summary>
        /// <param name="s">String to display in
textBox</param>
        public void LogInfo(string s)
        {
            MessageBox.Show(s);
        }

        /// <summary>
        /// Display on a textBox the mission list
        /// </summary>
        /// <param name="MissionList"></param>
        public void DisplayCoordenadasList(List<Coordenadas>
CoordenadasList)
        {
            listBox1.Items.Clear();
            foreach (Coordenadas coord in CoordenadasList)
            {
                listBox1.Items.Add("Coordenadas: X = " +
coord.x.ToString() + ", Y = " + coord.y.ToString());
            }
        }

        /// <summary>
        /// Clear textBox displaying current mission
        /// </summary>
        public void clearTextBox2()
        {
            this.textBox2.Text = "";
        }

        /// <summary>
        /// Display on textbox the current mission
        /// </summary>
        /// <param name="M"></param>
        public void DisplayCurrentCoordenadas(Coordenadas C)
        {
            textBox2.Text = "Dirigiendose a coordenadas X = "
+ C.x.ToString() + ", Y = " + C.y.ToString();
        }

        /// <summary>
        /// Add new mission to list
        /// </summary>
```

```
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button3_Click(object sender, EventArgs e)
        {
            try
            {
                Coordenadas _newcoordenada = new
Coordenadas();

                _newcoordenada.x =
Convert.ToDouble(textBox3.Text);
                _newcoordenada.y =
Convert.ToDouble(textBox4.Text);

                AddCoordenadas _addcoordenadas = new
AddCoordenadas();
                _addcoordenadas.AddedCoordenadas =
_newcoordenada;
                this._hmiPort.Post(_addcoordenadas);
            }
            catch
            {
                MessageBox.Show("Bad value entered");
            }
        }

        /// <summary>
        /// Remove Mission from the list
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button4_Click(object sender, EventArgs e)
        {
            int _index = -1;
            _index = listBox1.SelectedIndex;
            if (_index != -1)
            {
                RemoveCoordenadas _removeCoordenadas = new
RemoveCoordenadas();
                _removeCoordenadas.Index = _index;
                this._hmiPort.Post(_removeCoordenadas);
            }
        }

        /// <summary>
```

```
        /// Up the mission in the list
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button5_Click(object sender, EventArgs e)
        {
            int _index = -1;
            _index = listBox1.SelectedIndex;
            if (_index != -1)
            {
                UpCoordenadas _upCoordenadas = new
UpCoordenadas();
                _upCoordenadas.Index = _index;
                this._hmiPort.Post(_upCoordenadas);
            }
        }

        /// <summary>
        /// Decrease the position of the mission selected in
the mission list
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button6_Click(object sender, EventArgs e)
        {
            int _index = -1;
            _index = listBox1.SelectedIndex;
            if (_index != -1)
            {
                DownCoordenadas _downCoordenadas = new
DownCoordenadas();
                _downCoordenadas.Index = _index;
                this._hmiPort.Post(_downCoordenadas);
            }
        }

        /// <summary>
        /// Click on this button to save missions in the state
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button7_Click(object sender, EventArgs e)
        {
            _hmiPort.Post(new LoadMission());
        }
```

```
        /// <summary>
        /// Form closed, drop the service
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void HMI_FormClosed(object sender,
FormClosedEventArgs e)
        {
            _hmiPort.Post(new DropFromGui());
        }

        private void buttonResetLocalization_Click(object
sender, EventArgs e)
        {
            this._hmiPort.Post(new ResetLocalization());
        }

        public void UpdateOdo(double X, double Y, double
Theta)
        {
            xval.Text = X.ToString("0.000");
            yval.Text = Y.ToString("0.000");
            thetaval.Text = Theta.ToString("0.000");
        }

    }
}
```

En primer lugar declaramos que vamos a usar. En este caso, queremos utilizar concretamente las herramientas de Windows para formas (ventanas interactivas…, etc).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
```

Ahora es momento de declarar el puerto HMI en la clase publica parcial HMI, en la que vamos a heredar para utilizar las funcionalidades de las formas de windows la clase Form.

Para inicializar el HMI necesitamos el constructor, el cual lo definimos con la propia clase HMI, en la cual debemos introducir como argumento de entrada el puerto.

Una vez hecho, llamamos a la función que inicializa la "Windows Form".

```
public partial class HMI : Form
    {

        //Member
        HMIPort _hmiPort;

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="port"></param>
        public HMI(HMIPort port)
        {
            this._hmiPort = port;

            InitializeComponent();
        }
```

A continuación vamos a definir la función de los primero botones del HMI:

El botón que habíamos diseñado y llamado "Start", en la parte superior derecha de la ventana, es el que se encarga de dar comienzo a la misión de llegar a las coordenadas objetivo. Al clicar este botón estamos ordenando a la función StartMission() que comience (la estamos llamando). Como vemos, es una función que no tiene argumentos de

entrada ni de salida, ya que como veremos posteriormente cuando la definamos en el navegador.cs, es una clase de tipo void, es decir, que no devuelve ningún valor.

La sintaxis que debemos escribir para definir que estamos pulsando un botón y queremos que se active su funcionalidad es: buttonN_Click, y como argumentos de entrada: object sender, EventArgs s.

```
/// <summary>
        /// Send new mission when the button1 is clicked
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button1_Click(object sender, EventArgs e)
        {
            this._hmiPort.Post(new StartMission());
        }
```

A continuación definimos una función en el HMI que vamos a utilizar con bastante frecuencia sobre todo a la hora de hacer las conexiones a los servicios para ver si estas se han realizado de forma correcta o ha habido errores durante el proceso. Esta funcion es LogInfo.

LogInfo recive como argumento de entrada un string (o cadena de caractéres) que será mostrado por pantalla en una nueva ventana emergente para informar de un suceso, por ejemplo, que el servicio del laser ha sido conectado de forma correcta. Esta ventana tiene un botor predeterminado de aceptar y otro de "X", ambos para cerrar la ventana. La instrucción para mostrar la ventana es MessageBox.Show(s)

```
        /// <summary>
        /// Display a text box
        /// </summary>
```

```
        /// <param name="s">String to display in
textBox</param>
        public void LogInfo(string s)
        {
            MessageBox.Show(s);
        }
```

Vamos ahora a definir que es lo que aparecerá en el campo de texto más grande en cuanto a tamaño del HMI.

Para ello creamos la funcion DisplayCoordenadasList, en la que como se puede intuir, debemos añadir como argumento de entrada la lista de coordenadas que queremos mostrar.

Esta lista de coordenadas son las coordenadas que se introducen de forma externa y que serán las coordenadas objetivo del robot. La lista de coordenadas ha sido creada a traves de una clase que se definirá posteriormente en el NavegadorTypes.cs.

Lo que hace la función es limpiar el campo de texto para despues añadir todas las coordenadas (x e y) con el bucle foreach, el cual recorre todos los elementos de un vector o una lista, como es nuestro caso, y los define con una variable que cambia continuamente en cada bucle de valor y que hemos llamado coord.

Como veremos despues, coord es una variable de la clase Coordenadas, que son cada uno de los miembros de la lista de coordenadas, y que también se definirá en el NavegadorTypes.cs.

Pues bien, podemos adelantar que las variables de la clase Coordenadas tendrán la componente x y la componente y:

```
        /// <summary>
        /// Display on a textBox the mission list
        /// </summary>
        /// <param name="MissionList"></param>
        public void DisplayCoordenadasList(List<Coordenadas>
CoordenadasList)
```

```
        {
            listBox1.Items.Clear();
            foreach (Coordenadas coord in CoordenadasList)
            {
                listBox1.Items.Add("Coordenadas: X = " +
coord.x.ToString() + ", Y = " + coord.y.ToString());
            }
        }
```

Definimos ahora la funcion que limpia el segundo campo de texto mas grande, y la función que escribe en el mismo la misión que se va a levar a cabo, es decir, las coordenadas objetivo de la misión actual.

En DisplayCurrentCoordenadas el argumento de entrada requerido serán las coordenadas, es decir, podemos escribir (Coordenadas C), y la función de la función será escribir en el campo de texto a donde se está dirigiendo el robot.

```
        /// <summary>
        /// Clear textBox displaying current mission
        /// </summary>
        public void clearTextBox2()
        {
            this.textBox2.Text = "";
        }

        /// <summary>
        /// Display on textbox the current mission
        /// </summary>
        /// <param name="M"></param>
        public void DisplayCurrentCoordenadas(Coordenadas C)
        {
            textBox2.Text = "Dirigiendose a coordenadas X = "
+ C.x.ToString() + ", Y = " + C.y.ToString();
        }
```

Definimos el comportamiento del botón “Add”, el cual simplemente lee y convierte en double el texto en los campos de texto 3 y 4, que corresponden a los campos de texto donde se escriben la coordenada x e

y que se encuentran debajo de los labels x e y respectivamente, y da el valor a una variable definida con la clase AddCoordenadas.

AddCoordenadas se definirá posteriormente en el NavegadorTypes.cs.

Lo que se hace con esta variable es situarla en el puerto HMI

Como detalle, se ha añadido la instrucción try y catch, cuya función es observar si se ha introducido correctamente el valor de las coordenadas, esto es, que ambos campos de texto contengan texto, aunque sea un 0, y ambos sean de tipo numérico para convertirlos a doble. Si el valor introducido no es valido, es decir, no cumple estos requisitos, una ventana de información será lanzada con el mensaje ("Bad value entered") (valor mal introducido).

```
        /// <summary>
        /// Add new mission to list
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button3_Click(object sender, EventArgs e)
        {
            try
            {
                Coordenadas _newcoordenada = new
Coordenadas();

                _newcoordenada.x =
Convert.ToDouble(textBox3.Text);
                _newcoordenada.y =
Convert.ToDouble(textBox4.Text);

                AddCoordenadas _addcoordenadas = new
AddCoordenadas();
                _addcoordenadas.AddedCoordenadas =
_newcoordenada;
                this._hmiPort.Post(_addcoordenadas);
            }
            catch
            {
```

```
                MessageBox.Show("Bad value entered");
            }
        }
```

Se han creado también 3 funcionalidades más que corresponden a los botones Up, Down y Remove, es decir, subir una lista de coordenadas en la lista que componen todas ellas en el campo de texto de coordenadas disponibles, bajar un puesto a las mismas, y por ultimo, suprimirla.

Cabe destacar que para seleccionar con cual se quiere realizar la operación, lo único que hay que hacer es resaltarla pinhandola con el ratón. De este modo, podemos recibir cual está seleccionada con la herramienta "index".

Además, ha sido creada para estas funciones una clase en NavegadorTypes.cs.

Como con add, lo unico que tenemos que hacer, es pasar el valor de la lista de coordenadas con la que queremos operar a una variable definida con la clase correspondiente y publicarla en el puerto del HMI. De este modo una vez definido el Navegador.cs se realizará la función por medio de la llamada de la función correspondiente que también habrá que definir como ya veremos:

```
        /// <summary>
        /// Remove Mission from the list
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button4_Click(object sender, EventArgs e)
        {
            int _index = -1;
            _index = listBox1.SelectedIndex;
            if (_index != -1)
            {
                RemoveCoordenadas _removeCoordenadas = new
RemoveCoordenadas();
```

```
                _removeCoordenadas.Index = _index;
                this._hmiPort.Post(_removeCoordenadas);
            }
        }

        /// <summary>
        /// Up the mission in the list
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button5_Click(object sender, EventArgs e)
        {
            int _index = -1;
            _index = listBox1.SelectedIndex;
            if (_index != -1)
            {
                UpCoordenadas _upCoordenadas = new
UpCoordenadas();
                _upCoordenadas.Index = _index;
                this._hmiPort.Post(_upCoordenadas);
            }
        }


        /// <summary>
        /// Decrease the position of the mission selected in
the mission list
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button6_Click(object sender, EventArgs e)
        {
            int _index = -1;
            _index = listBox1.SelectedIndex;
            if (_index != -1)
            {
                DownCoordenadas _downCoordenadas = new
DownCoordenadas();
                _downCoordenadas.Index = _index;
                this._hmiPort.Post(_downCoordenadas);
            }
        }
```

Llegamos al punto en el que definimos el comportamiento del botón load missions.

Este botón realizará una serie de sencillos cálculos e inicialización de algunas variables en Navegador.cs como ya verémos, necesarias para la definición de los objetivos del navegador.

Basicamente lo que hace es coger la información de las distancias de las coordenadas introducidas y meterlas en variables de estado del servicio como ya veremos.

```
        /// <summary>
        /// Click on this button to save missions in the state
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button7_Click(object sender, EventArgs e)
        {
            _hmiPort.Post(new LoadMission());
        }
```

A continuación, una sencilla funcionalidad que indica que si cerramos el HMI, nos desconectamos del servicio.

Esto es util para no dejar nada conectado o incluso como parada de emergencia.

La función DropFromGui() se definirá posteriormente en el Navegador.cs

```
        /// <summary>
        /// Form closed, drop the service
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void HMI_FormClosed(object sender,
FormClosedEventArgs e)
        {
            _hmiPort.Post(new DropFromGui());
        }
```

Por último la localización del robot.

A parte de ser una parte primordial en el desarrollo del algoritmo de navegación, se ha introducido una serie de labels para poder visualizar en tiempo real la posición del robot así como la orientación en nuestro sistema de coordenadas.

A parte de poner estos labels se ha añadido un botón que resetea las coordenadas y la orientación llamado Reset Localization. Este boton cuando es presionado llama a la función ResetLocalization() que se define en Navegador.cs

Para visualizar en tiempo real la odometría, es decir, la posición, definimos en el HMI.cs una función de tipo void llamada UpdateOdo que coge como parámetros de entrada la x la y y la theta correspondientes a las actualizaciones de la posición y orientación del robot (las cuales ya veremos como se obtienen en el Navegador.cs) y las escribe previa conversión a string en los labels en negrita correspondientes.

```
        private void buttonResetLocalization_Click(object
sender, EventArgs e)
        {
            this._hmiPort.Post(new ResetLocalization());
        }

        public void UpdateOdo(double X, double Y, double
Theta)
        {
            xval.Text = X.ToString("0.000");
            yval.Text = Y.ToString("0.000");
            thetaval.Text = Theta.ToString("0.000");
        }

    }
}
```

3.2.2.5 NavegadorTypes.cs

La siguiente parte corresponde al NavegadorTypes.cs

El código completo es el siguiente:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using W3C.Soap;
using laser =
Robosoft.RobuBOX.Generic.Devices.LaserSensor.Proxy;
using Navegador = Navegador;

namespace Navegador
{
      public sealed class Contract
      {
            [DataMember]
            public const string Identifier =
"http://schemas.tempuri.org/2014/05/navegador.html";
      }

      [DataContract()]
      public class NavegadorState
      {
        /// <summary>
        /// List of coordenadas to send
        /// </summary>
        [DataMember]
        public List<Coordenadas> CoordenadasList = new
List<Coordenadas>();

        public double vel;

        public double ang;

        public double x;

        public double y;
```

```
        public double theta;

        public double metax;

        public double metay;

        public bool prev;

        public int estado;

      }

      [ServicePort]
    public class NavegadorOperations :
PortSet<DsspDefaultLookup, DsspDefaultDrop, Get, Replace,
Update, Subscribe>
    {
    }

    /// <summary>
    /// Port to communicate between the service and its HMI
    /// </summary>
    public class HMIPort : PortSet<DsspDefaultLookup,
DsspDefaultDrop, StartMission, AddCoordenadas,
RemoveCoordenadas,
        UpCoordenadas, DownCoordenadas, LoadMission,
ResetLocalization, DropFromGui>
    {
    }

    //generic class operation
    public class Operation
    {
    }

    /// <summary>
    /// Provides Read-Access to the  state.
    /// </summary>
    public class Get : Get<GetRequestType,
PortSet<NavegadorState, Fault>>
    {
    }
```

```
    public class Subscribe : Subscribe<SubscribeRequestType,
PortSet<SubscribeResponseType, Fault>>
    {
    }

    /// <summary>
    /// Used to replace state
    /// </summary>
    public class Replace : Replace<NavegadorState,
PortSet<DefaultReplaceResponseType, Fault>>
    {
    }

    /// <summary>
    /// Used to update state
    /// </summary>
    public class Update : Update<NavegadorState,
PortSet<DefaultReplaceResponseType, Fault>>
    {
    }

    /// <summary>
    /// Internal message from HMI to send mission when
clicking on start button
    /// </summary>
    public class StartMission
    {
    }

    /// <summary>
    /// Coordinates definition
    /// </summary>
    [DataContract]
    public class Coordenadas
    {
        /// <summary>
        /// Coordenadas
        /// </summary>
        [DataMember]
        public double x;

        [DataMember]
        public double y;
    }
```

```
    public class AddCoordenadas
    {
        public Coordenadas AddedCoordenadas;
    }

    public class RemoveCoordenadas
    {
        public int Index;
    }

    public class UpCoordenadas
    {
        public int Index;
    }

    public class DownCoordenadas
    {
        public int Index;
    }

    public class LoadMission
    {
    }

      public class DropFromGui
    {
    }

     public class ResetLocalization : Operation
     {
     }

}
```

Como se viene presentando en apartados anteriores, la primera parte declarativa corresponde con los usings

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
```

```
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using W3C.Soap;
using laser =
Robosoft.RobuBOX.Generic.Devices.LaserSensor.Proxy;
using Navegador = Navegador;
```

Una vez cargados los usings podemos acceder a sus funcionalidades.

Ahora bien, el siguiente paso a realizar será definir el namespace de nuestro servicio (Navegador) y definir el contract que se explicó anteriormente:

```
namespace Navegador
{
	public sealed class Contract
	{
		[DataMember]
		public const string Identifier =
"http://schemas.tempuri.org/2014/05/navegador.html";
	}
```

Lo que hacemos así es informar al servicio cual es el contract, es decir, por decirlo de alguna manera, lo estamos identificando.

Ahora declaramos el estado del servicio en general.

El estado del servicio es de gran importancia, ya que al ser algo tan general que engloba al servicio por completo, podemos acceder a lo que definamos en el desde cualquier parte del código de la programación. Así pues, definimos el estado del navegador y declaramos una serie de variables.

```
[DataContract()]
	public class NavegadorState
	{
	  /// <summary>
	  /// List of coordenadas to send
	  /// </summary>
	  [DataMember]
```

```
        public List<Coordenadas> CoordenadasList = new
List<Coordenadas>();

        public double vel;

        public double ang;

        public double x;

        public double y;

        public double theta;

        public double metax;

        public double metay;

        public bool prev;

        public int estado;

      }
```

Como habíamos dicho, dentro de la clase publica NavegadorState (que es dato del contract definido antes) definimos como miembro la lista de coordenadas donde se guardan las coordenadas a utilizar.

La manera de definir una lista es la siguiente:

```
public List<ObjetoUtilizado> NombreDeLaLista = new
List<UbjetoUtilizado>();
```

No debemos confundir una lista con un vector.

Se definen también una serie de variables que se van a utilizar en algunas de las partes del código. Estas son:

- Vel: es la velocidad lineal que deseamos que tenga el robot en cada momento. Es de tipo double.

- Ang: es la velocidad angular que deseamos que tenga el robot en cada momento. Es de tipo double.
- X: es la coordenada x local del sistema de localización odométrico que se corresponde con la coordenada y del sistema de coordenadas mundo. Es de tipo double
- Y: es la coordenada y local del sistema de localización odométrico que se corresponde con la coordenada -x del sistema de coordenadas mundo. Es de tipo double
- Theta: es la orientación local del sistema de localización odométrico que se corresponde con el ángulo π/2 + theta del sistema de coordenadas mundo tomando el eje x como ángulo nulo. Es de tipo double
- Metax: es la coordenada x objetivo de la misión actual. Es de tipo double
- Metay: es la coordenada y objetivo de la misión actual. Es de tipo double
- Prev: es una variable de control booleana que se activa cuando se muestra la ventana emergente que informa de que ya se ha llegado al objetivo y cambia su valor al opuesto para así evitar que la ventana se muestre en un bucle infinito, ya que la parte declarativa en la que se encuentra, como se verá, es un iterador enumerador situado en un puesto exclusivo. Es de tipo boolean.
- Estado: es un valor entero que indica en qué estado de navegación se encuentra el sistema. La variable controla la velocidad linear y angular que se va a actualizar en la lógica en cada momento. Es de tipo integer.

Ahora se fijarán las operaciones del navegador y del HMI:

```
[ServicePort]
    public class NavegadorOperations :
PortSet<DsspDefaultLookup, DsspDefaultDrop, Get, Replace,
Update, Subscribe>
    {
    }

    /// <summary>
    /// Port to communicate between the service and its HMI
    /// </summary>
    public class HMIPort : PortSet<DsspDefaultLookup,
DsspDefaultDrop, StartMission, AddCoordenadas,
RemoveCoordenadas,
        UpCoordenadas, DownCoordenadas, LoadMission,
ResetLocalization, DropFromGui>
    {
    }
```

Como se puede apreciar, ambas clases heredan la clase PortSet, perteneciente al paquete de MRDS y RobuBOX con algunas de sus funciones.

En NavegadorOperations tenemos como siempre el DsspDefaultLookup y el DsspDefaultDrop, que se activan al iniciar y terminar o cerrar el servicio respectivamente, y luego vemos que incluye los DssHandlers Get, Replace, Update y Subscribe:

Get nos da información del estado o servicio que seleccionemos. La información puede ser de numerosos tipos y siempre depende del servicio que utilizamos y la opción dentro del servicio o del estado que seleccionamos.

Replace cambia el valor de una variable, estado de un servicio, etc… a un valor que nosotros seleccionamos. Se utiliza muy comúnmente para realizar resets.

Update es una herramienta muy útil y comúnmente utilizada. Cuando utilizamos Update, cada vez que un servicio, funcionalidad, estado, etc… recibe nueva información (es decir, cambian sus valores) llama a la función o clase a la que está afectando y por tanto se activa el código de esa función o clase en el que están incluidas variables que toman valores de la información que proporciona el servicio. Esto es especialmente interesante ya que como se ha dicho, el valor ahora es nuevo y es interesante saber con qué se trata en cada momento. Un ejemplo directo (que además se ha usado en este servicio) es el caso del láser. En el servicio que se ha conectado del láser se ha definido como ya se verá posteriormente un Update handler. Este Update handler se activa cada vez que el láser recibe nueva información y esto es de vital importancia porque es este quien nos avisa de los obstáculos que tenemos en nuestro entorno, y es precisamente porque se actualiza cuando cambia de valor por lo que percibimos estos objetos y podemos esquivarlos.

Por ultimo subscribe nos permite subscribirnos como su propio nombre indica a los diferentes servicios para poder conectarnos o hacer unsubscribes para desconectarnos.

El caso del puerto del HMI es algo mas complejo ya que en este caso debemos definir todas y cada una de las clases que hemos utilizado y que ya se han definido.

De nuevo estas son: DsspDefaultLookup, DsspDefaultDrop, StartMission, AddCoordenadas, RemoveCoordenadas, UpCoordenadas, DownCoordenadas, LoadMission, ResetLocalization, DropFromGui.

Ahora que todo está predefinido, pasamos a definir cada una de estas clases, además de otras que se utilizarán en el servicio posteriormente:

```
//generic class operation
    public class Operation
    {
    }
```

Definimos una clase generica llamada operacion que heredaremos en el reset de la localización.

```
    /// <summary>
    /// Provides Read-Access to the  state.
    /// </summary>
    public class Get : Get<GetRequestType,
PortSet<NavegadorState, Fault>>
    {
    }
```

El get, como ya hemos dicho, es utilizado para tener acceso a la lectura del estado del navegador.

```
    public class Subscribe : Subscribe<SubscribeRequestType,
PortSet<SubscribeResponseType, Fault>>
    {
    }
```

Con subscribe tenemos acceso a las subscripciones.

```
    /// <summary>
    /// Used to replace state
    /// </summary>
    public class Replace : Replace<NavegadorState,
PortSet<DefaultReplaceResponseType, Fault>>
    {
    }
```

Usamos replace para cambiar el valor de la parte que nos interese del estado

```
    /// <summary>
    /// Used to update state
    /// </summary>
    public class Update : Update<NavegadorState,
PortSet<DefaultReplaceResponseType, Fault>>
    {
    }
```

Declaramos el update para tener acceso a su funcionalidad en los handlers que definamos.

```
    /// <summary>
    /// Internal message from HMI to send mission when
clicking on start button
    /// </summary>
    public class StartMission
    {
    }
```

Definimos la clase StartMission, que como se explicará mas adelante cuando se defina, dará comienzo a la realizacion de la mision consistente en alcanzar las coordenadas objetivo

```
    /// <summary>
    /// Coordinates definition
    /// </summary>
    [DataContract]
    public class Coordenadas
```

```
    {
        /// <summary>
        /// Coordenadas
        /// </summary>
        [DataMember]
        public double x;

        [DataMember]
        public double y;
    }
```

Definimos la clase coordenadas, que tendrá como funcion crear variables con un valor double x y otro y que definan un punto en el espacio plano de dos dimensiones definido por el sistema de coordenadas cartesianas mundo.

```
    public class AddCoordenadas
    {
        public Coordenadas AddedCoordenadas;
    }

    public class RemoveCoordenadas
    {
        public int Index;
    }

    public class UpCoordenadas
    {
        public int Index;
    }

    public class DownCoordenadas
    {
        public int Index;
    }

    public class LoadMission
    {
    }

      public class DropFromGui
    {
    }
```

Todas estas clases son las que corresponden a las funcionalidades descritas del HMI. Como se puede apreciar, algunas de ellas tienen como valores internos enteros (index) que corresponden con la posicion en el cuadro de texto del elemento resaltado para poder operar con el.

En el caso de addcoordenadas vemos que tiene dentro como valor una variable de tipo coordenadas, definida en esta misma parte del servicio.

```
    public class ResetLocalization : Operation
    {
    }

}
```

Por ultimo vemos la clase publica resetLocalization que hereda la clase generica que habiamos creado Operation.

3.2.2.6 Navegador.cs

Pasamos por fin al Navegador.cs, el cual contiene la parte principal del programa y es donde se define el comportamiento que tendrá el robot.

Como se ha venido haciendo, el código completo es el siguiente:

```
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Xml;
using Navegador = Navegador;
using System.Security.Permissions;
```

```
using Microsoft.Ccr.Adapters.WinForms;
using System.Windows.Forms;
using types = Robosoft.RobuBOX.Core.Types.Proxy;

using ds = Microsoft.Dss.Services.Directory;
using submgr = Microsoft.Dss.Services.SubscriptionManager;

using laser =
Robosoft.RobuBOX.Generic.Devices.LaserSensor.Proxy;

using drive =
Robosoft.RobuBOX.Generic.Devices.DifferentialDrive.Proxy;

using loc =
Robosoft.RobuBOX.Generic.Processing.Signal.Localization.Proxy;

namespace Navegador
{
      [Contract(Contract.Identifier)]
      [DisplayName("Navegador")]
      [Description("Navegador service (no description
provided)")]
      class NavegadorService : DsspServiceBase
      {

            private NavegadorState _state = new
NavegadorState();

            [ServicePort("/Navegador", AllowMultipleInstances
= false)]
            private NavegadorOperations _mainPort = new
NavegadorOperations();

        [Partner("differentialdrive", Contract =
drive.Contract.Identifier, CreationPolicy =
PartnerCreationPolicy.UseExisting)]
        drive.DifferentialDriveOperations _drivePort = new
drive.DifferentialDriveOperations();

        drive.DifferentialDriveState _driveState;

        private laser.LaserSensorOperations _laserPort;
        private laser.LaserSensorOperations _laserNotifyPort =
new laser.LaserSensorOperations();
```

```
        private loc.LocalizationOperations _locPort;
        private loc.LocalizationOperations _locNotifyPort =
new loc.LocalizationOperations();

        //internal port for HMI
        private HMIPort _hmiPort = new HMIPort();

        //Variables
        HMI _mainHMI;

        public List<Coordenadas> _CoordenadasList = new
List<Coordenadas>();

        Port<bool> _periodicHandlerPort = new Port<bool>();

        string _laserSubMgr = string.Empty;
        string _laserSubscriber = string.Empty;

        string _localizationSubMgr = string.Empty;
        string _localizationSubscriber = string.Empty;

            public NavegadorService(DsspServiceCreationPort
creationPort)
                  : base(creationPort)
            {
            }

            protected override void Start()
            {
            if (_state == null)
            {
                Coordenadas inicial = new Coordenadas();
                inicial.x = 0;
                inicial.y = 0;
                _state.CoordenadasList.Add(inicial);//add
coordenada to coordenadas list

                SaveState(_state);
            }

            foreach (Coordenadas inic in
this._state.CoordenadasList)
            {
                _CoordenadasList.Add(inic);
```

```
            }

            base.Start();

            if (_state == null)
            {
                _state.CoordenadasList = new
List<Coordenadas>();
                Coordenadas inicial = new Coordenadas();
                inicial.x = 0;
                inicial.y = 0;
                _state.prev = true;
                _state.CoordenadasList.Add(inicial);//add
coordenada to coordenadas list
                SaveState(_state);
            }

            foreach (Coordenadas inic in
this._state.CoordenadasList)
            {
                _CoordenadasList.Add(inic);
            }

            MainPortInterleave.CombineWith(
                Arbiter.Interleave(
                    new TeardownReceiverGroup(),
                    new ExclusiveReceiverGroup(

Arbiter.ReceiveWithIterator<bool>(true, _periodicHandlerPort,
PeriodicHandler),
                        Arbiter.Receive<laser.Update>(true,
_laserNotifyPort, UpdateLaserHandler),
                        //Arbiter.Receive<loc.Update>(true,
_locNotifyPort, LocalizationUpdateHandler),
                        Arbiter.Receive<LoadMission>(true,
_hmiPort, LoadHandler),
                        Arbiter.Receive<DropFromGui>(true,
_hmiPort, DropFromGuiHandler)
                        ),
                    new ConcurrentReceiverGroup(
                        Arbiter.Receive<StartMission>(true,
_hmiPort, SendCoordenadasHandler),
                        Arbiter.Receive<AddCoordenadas>(true,
_hmiPort, AddCoordenadasHandler),
```

```
Arbiter.Receive<ResetLocalization>(true, _hmiPort,
ResetLocalizationHandler),

Arbiter.Receive<RemoveCoordenadas>(true, _hmiPort,
RemoveCoordenadasHandler),
                        Arbiter.Receive<UpCoordenadas>(true,
_hmiPort, UpCoordenadasHandler),
                        Arbiter.Receive<DownCoordenadas>(true,
_hmiPort, DownCoordenadasHandler)
                )));

            //launch Winform
            WinFormsServicePort.Post(new
RunForm(RunMainWindow));

            //Display the mission list in the HMI
            WinFormsServicePort.Post(new FormInvoke(delegate()
            {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
            }));

            string texto = "X Y alpha";
            System.IO.StreamWriter sw = new
System.IO.StreamWriter(@"C:\datos\datos.txt", false);
string texto2 = "Vlinear Vangular";
            System.IO.StreamWriter sw2 = new
System.IO.StreamWriter(@"C:\datosv\datosv.txt", false);
            sw2.WriteLine(texto2);
            sw2.Close();

            sw.WriteLine(texto);
            sw.Close();

            SpawnIterator(ConnectTodriveHandler);
            SpawnIterator(ConnectToLaserHandler);
            SpawnIterator(ConnectToLocalizationHandler);

            }
        IEnumerator<ITask> ConnectTodriveHandler()
        {
            bool error = false;
            //Informe to which sercice to connect
```

```
            ServiceInfoType info = new
ServiceInfoType(drive.Contract.Identifier);
            //Declare a port to the node/machine where the
service to connect is running, giving URI
            ds.DirectoryPort remotePort =
ServiceForwarder<ds.DirectoryPort>(new Uri("http://" +
"robuLAB10" + ":50000/directory"));
            //Declare the request
            ds.Query query = new ds.Query(new
ds.QueryRequestType(info));
            //post
            remotePort.Post(query);

            //listen for a response Display in a textbox the
sentence in red
            yield return Arbiter.Choice<ds.QueryResponseType,
W3C.Soap.Fault>(query.ResponsePort,
                delegate(ds.QueryResponseType response)
                {
                    //instanciate the service port
                    _drivePort =
ServiceForwarder<drive.DifferentialDriveOperations>(response.R
ecordList[0].Service);

                },
                delegate(W3C.Soap.Fault fault)
                {
                    error = true;
                });

            if (!error)
            {
                yield return
Arbiter.Choice<drive.DifferentialDriveState, W3C.Soap.Fault>(
                _drivePort.Get(),
                delegate(drive.DifferentialDriveState
response)
                {
                    _driveState = response;
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                    {
                        _mainHMI.LogInfo("connected to
DifferentilDrive service");
                    }));
```

```
                },
                delegate(W3C.Soap.Fault fault)
                {
                    LogError("Failed to get DifferentialDrive
State");
                });
            }
            yield break;
        }

        IEnumerator<ITask> ConnectToLaserHandler()
        {
            bool error2 = false;
            ServiceInfoType info = new
ServiceInfoType(laser.Contract.Identifier);
            //Change "laser" by the alias of the service to
connect
            info = new
ServiceInfoType(laser.Contract.Identifier);
            //Declare a port to the node/machine where the
service to connect is running, giving URI
            ds.DirectoryPort remotePort2 =
ServiceForwarder<ds.DirectoryPort>(new Uri("http://" +
"robuLAB10" + ":50000/directory"));
            //Declare request
            ds.Query query2 = new ds.Query(new
ds.QueryRequestType(info));
            //post
            remotePort2.Post(query2);

            //listen for a response
            yield return Arbiter.Choice<ds.QueryResponseType,
W3C.Soap.Fault>(query2.ResponsePort,
                delegate(ds.QueryResponseType response)
                {
                    error2 = false;
                    //instanciate services port: receive the
two intance of laser
                    _laserPort =
ServiceForwarder<laser.LaserSensorOperations>(response.RecordL
ist[0].Service);
                },
                delegate(W3C.Soap.Fault fault)
                {
                    error2 = true;
```

```
                    _laserPort = null;
                    LogInfo(LogGroups.Console, "ERROR
connecting to Laser service");
                });
            if (error2 == false)
            {
                //Subscribe with the odometry
                yield return
Arbiter.Choice<SubscribeResponseType, W3C.Soap.Fault>(
                    _laserPort.Subscribe(_laserNotifyPort),
                    delegate(SubscribeResponseType response)
                    {
                        //Make he link between the message
receive and the handler
                        LogInfo("subscribed to Laser
service...");
                        _laserSubMgr =
response.SubscriptionManager;
                        _laserSubscriber =
response.Subscriber;


Activate(Arbiter.Receive<laser.Update>(false,
_laserNotifyPort, UpdateLaserHandler));

                        WinFormsServicePort.Post(new
FormInvoke(delegate()
                        {
                            _mainHMI.LogInfo("connected to
Laser service");
                        }));
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogInfo("Failed to subscribe to
Laser...");

                    });
            }
        }

        IEnumerator<ITask> ConnectToLocalizationHandler()
        {
            bool error = false;
```

```
            //Change "loc" by the alias of the service to
connect
            ServiceInfoType info = new
ServiceInfoType(loc.Contract.Identifier);
            //Declare a port to the node/machine where the
service to connect is running, giving URI
            ds.DirectoryPort remotePort =
ServiceForwarder<ds.DirectoryPort>(new Uri("http://" +
"robuLAB10" + ":50000/directory"));
            //Declare the request
            ds.Query query = new ds.Query(new
ds.QueryRequestType(info));
            //post
            remotePort.Post(query);
            //listen for a response
            yield return Arbiter.Choice<ds.QueryResponseType,
W3C.Soap.Fault>(query.ResponsePort,
                delegate(ds.QueryResponseType response)
                {
                    //to be alert when odometry is updated
                    _locPort =
ServiceForwarder<loc.LocalizationOperations>(
                        response.RecordList[0].Service);
                },
                delegate(W3C.Soap.Fault fault)
                {
                    error = true;
                    _locPort = null;
                });

            if (!error)
            {
                SubscribeRequestType request = new
SubscribeRequestType();
                request.Subscriber = "http://" + "robuLAB10" +
":50000/directory";

                //Subscribe with the localization
                yield return
Arbiter.Choice<SubscribeResponseType, W3C.Soap.Fault>(
                    _locPort.Subscribe(_locNotifyPort),
                    delegate(SubscribeResponseType response)
                    {
                        _localizationSubMgr =
response.SubscriptionManager;
```

```
                        _localizationSubscriber =
response.Subscriber;


Activate(Arbiter.Receive<loc.Update>(true, _locNotifyPort,
LocalizationUpdateHandler));
                        WinFormsServicePort.Post(new
FormInvoke(delegate()
                        {
                            _mainHMI.buttonResetOdo.Enabled =
true;
                        }));
                        WinFormsServicePort.Post(new
FormInvoke(delegate()
                        {
                            _mainHMI.LogInfo("connected to
Localization service");
                        }));
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogInfo("Failed to subscribe to
Localization...");
                    });
            }
        }

        #region drop & save

        /// <summary>
        /// Drop message sent from gui
        /// </summary>
        /// <param name="d"></param>
        void DropFromGuiHandler(DropFromGui d)
        {
            _mainPort.Post(new DsspDefaultDrop());
        }

        /// <summary>
        /// Save current missions in state xml file
        /// </summary>
        /// <param name="s"></param>
        void LoadHandler(LoadMission s)
        {
```

```
            _state.metax = _CoordenadasList[0].x;
            _state.metay = _CoordenadasList[0].y;
            _state.prev = true;
            _state.contador = 1;

            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
                _mainHMI.LogInfo("Coordenadas lanzadas");
            }));

        }

        #endregion

        public void UpdateLaserHandler(laser.Update update)
        {
            double xrobot = -_state.y;
            double yrobot = _state.x;
            double xob = _state.metax - xrobot;
            double yob = _state.metay - yrobot;
            int estado = 1;

            foreach (laser.Echo E in update.Body.Echos)
            {
                types.Point2D pointCartesian = new
types.Point2D();
                pointCartesian.X = E.Distance *
Math.Cos(E.Angle);
                pointCartesian.Y = E.Distance *
Math.Sin(E.Angle);

                if (pointCartesian.Y < 0.25 &&
pointCartesian.Y > -0.25 && pointCartesian.X > 0 &&
pointCartesian.X <= 0.5)

                    estado = 3;

                else if (_state.contador == 1)
                {
                    if (pointCartesian.Y < 0.25 &&
pointCartesian.Y > -0.25 && pointCartesian.X > 0.5 &&
pointCartesian.X <= 1.1)
                    {
                        estado = 2;
                        _state.contador = 2;
```

```
                    }

                }

                else if (_state.contador == 2)
                {
                    if (pointCartesian.Y < 0 &&
pointCartesian.Y > -1 && pointCartesian.X > 0.0 &&
pointCartesian.X <= 1.1)
                    {
                        estado = 2;
                        _state.contador = 2;
                    }

                }

            }

            if (_state.prev == false)
            {
                _state.vel = 0.0;
                _state.ang = 0.0;

            }
            else if (_state.prev == true)
            {
                if (estado == 2)
                {
                    _state.vel = 0.5;
                    _state.ang = 0.5;
                    _state.contador = 2;
                }
                else if (estado == 3)
                {
                    _state.vel = 0.0;
                    _state.ang = 1.0;
                    _state.contador = 1;
                }
                else if (xrobot > _state.metax + 0.020)
                {
                    double angle = (Math.PI / 2) -
Math.Atan(yob / xob) - _state.theta;

                    if (angle < -0.2)
                    {
```

```
                        _state.vel = 0.2;
                        _state.ang = -0.6;
                        _state.contador = 1;
                    }

                    else if (angle > 0.2)
                    {
                        _state.vel = 0.2;
                        _state.ang = 0.6;
                        _state.contador = 1;
                    }
                    else
                    {
                        _state.vel = 0.5;
                        _state.ang = 0;
                        _state.contador = 1;
                    }
                }
                else if (xrobot < _state.metax - 0.020)
                {
                    double angle = (Math.PI / 2) -
Math.Atan(yob / xob) + _state.theta;

                    if (angle < -0.2)
                    {
                        _state.vel = 0.2;
                        _state.ang = 0.6;
                        _state.contador = 1;
                    }

                    else if (angle > 0.2)
                    {
                        _state.vel = 0.2;
                        _state.ang = -0.6;
                        _state.contador = 1;
                    }
                    else
                    {
                        _state.vel = 0.5;
                        _state.ang = 0;
                        _state.contador = 1;
                    }
                }
                else
                {
```

```
                    _state.vel = 0.5;
                    _state.ang = 0;
                    _state.contador = 1;
                }
            }


                /* pruebas
                if (_state.estado == 1)
                {
                    _state.vel = 0.2;
                    _state.ang = 0;
                }
                else
                {
                    _state.vel = 0;
                    _state.ang = 0;
                }
                */

        }

        //Function to instatiate the HMI window
        HMI RunMainWindow()
        {
            _mainHMI = new HMI(_hmiPort);
            return _mainHMI;
        }

        IEnumerator<ITask> PeriodicHandler(bool b)
        {
            drive.DriveRequest request = new
drive.DriveRequest();

            if (-_state.y < _state.metax + 0.2 && -_state.y >
_state.metax - 0.2 && _state.x < _state.metay + 0.2 &&
_state.x > _state.metay - 0.2)
            {
                request.LinearSpeed = 0.0;
                request.AngularSpeed = 0.0;

                if (_state.prev == true)
                {
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
```

```
                    {
                        _mainHMI.LogInfo("Coordenadas
alcanzadas");
                    }));
                    _state.prev = false;
                }
            }
            else
            {
                request.LinearSpeed = _state.vel;
                request.AngularSpeed = _state.ang;
            }

            System.IO.StreamWriter sw = new
System.IO.StreamWriter(@"C:\datosv\datosv.txt", true);
            sw.WriteLine(request.LinearSpeed + " " +
request.AngularSpeed);
            sw.Close();

               yield return
Arbiter.Choice<DefaultSubmitResponseType, W3C.Soap.Fault>(
                    _drivePort.Drive(request),
                    delegate(DefaultSubmitResponseType
response)
                    {
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                    });


        yield return Arbiter.Receive(false, TimeoutPort(80),
delegate(DateTime d) { });

            _periodicHandlerPort.Post(true);

            yield break;
        }

        #region Dss handlers

        /// <summary>
        /// Get Handler
        /// </summary>
        /// <param name="get"></param>
```

```
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Concurrent)]
        public virtual IEnumerator<ITask> GetHandler(Get get)
        {
            get.ResponsePort.Post(_state);
            yield break;
        }

        /// <summary>
        /// Replace Handler
        /// </summary>
        /// <param name="replace"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Exclusive)]
        public virtual IEnumerator<ITask>
ReplaceHandler(Replace replace)
        {
            _state = replace.Body;

replace.ResponsePort.Post(DefaultReplaceResponseType.Instance)
;
            yield break;
        }
        /// <summary>
        /// Drop Handler
        /// </summary>
        /// <param name="replace"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Teardown)]
        public virtual IEnumerator<ITask>
DropHandler(DsspDefaultDrop drop)
        {
            DirectoryDelete();
            Shutdown();

drop.ResponsePort.Post(DefaultDropResponseType.Instance);

            if (_laserSubMgr != string.Empty)
            {
                //unsubscribe to laser
                submgr.DeleteSubscription delete = new
submgr.DeleteSubscription();
                delete.Body = new
submgr.DeleteSubscriptionMessage(_laserSubscriber);
```

```
ServiceForwarder<submgr.SubscriptionManagerPort>(_laserSubMgr)
.Post(delete);
                yield return
Arbiter.Choice<DefaultDeleteResponseType, W3C.Soap.Fault>(
                    delete.ResponsePort,
                    delegate(DefaultDeleteResponseType
response)
                    {
                        _laserSubMgr = string.Empty;
                        _laserSubscriber = string.Empty;
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogError("Error deleting subscription
to laser");
                    });
            }
            if (_localizationSubMgr != null)
            {
                //unsubscribe to localization
                submgr.DeleteSubscription delete = new
submgr.DeleteSubscription();
                delete.Body = new
submgr.DeleteSubscriptionMessage(_localizationSubscriber);

ServiceForwarder<submgr.SubscriptionManagerPort>(_localization
SubMgr).Post(delete);
                yield return
Arbiter.Choice<DefaultDeleteResponseType, W3C.Soap.Fault>(
                    delete.ResponsePort,
                    delegate(DefaultDeleteResponseType
response)
                    {
                        _localizationSubMgr = string.Empty;
                        _localizationSubscriber =
string.Empty;
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogError("Error deleting subscription
to localization");
                    });
            }
```

```
            yield break;
        }

        #endregion

        #region StartMission, AddMission, RemoveMission, Up
and Down, reset and update loc

        /// <summary>
        /// Reset odometry
        /// </summary>
        /// <param name="r"></param>
        void ResetLocalizationHandler(ResetLocalization r)
        {
            loc.Replace reset = new loc.Replace();
            reset.Body.X = 0;
            reset.Body.Y = 0;
            reset.Body.Theta = 0;
            this._locPort.Post(reset);

            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
                _mainHMI.UpdateOdo(0,0,90);
            }));

        }

        /// <summary>
        /// Occure when odometry is updated
        /// </summary>
        /// <param name="up">Odometry</param>
        void LocalizationUpdateHandler(loc.Update up)
        {
            _state.x = up.Body.X;
            _state.y = up.Body.Y;
            _state.theta = up.Body.Theta;

            double xpos = -_state.y;
            double ypos = _state.x;
            double alpha = _state.theta * (180 / Math.PI) +
90;

            // De aqui
            System.IO.StreamWriter sw = new
System.IO.StreamWriter(@"C:\datos\datos.txt", true);
```

```
            sw.WriteLine(xpos + " " + ypos + " " + alpha);
            sw.Close();
            // a aqui

            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
                _mainHMI.UpdateOdo(xpos, ypos, alpha);
            }));
        }

        /// <summary>
        /// User clicks on HMI: the next mission is sent send
next mission
        /// </summary>
        /// <param name="s"></param>
        void SendCoordenadasHandler(StartMission s)
        {
            _periodicHandlerPort.Post(true);

_mainHMI.DisplayCurrentCoordenadas(_CoordenadasList[0]);
        }


        /// <summary>
        /// Add mission to missionlist
        /// </summary>
        /// <param name="a"></param>
        void AddCoordenadasHandler(AddCoordenadas a)
        {
            _CoordenadasList.Add(a.AddedCoordenadas);
            WinFormsServicePort.Post(new FormInvoke(delegate()
            {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
            }));
        }

        /// <summary>
        /// Remove a mission
        /// </summary>
        /// <param name="r"></param>
        void RemoveCoordenadasHandler(RemoveCoordenadas r)
        {
            _CoordenadasList.RemoveAt(r.Index);
            WinFormsServicePort.Post(new FormInvoke(delegate()
```

```
            {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
            }));
        }

        /// <summary>
        /// increate the position of a mission in list
        /// </summary>
        /// <param name="r"></param>
        void UpCoordenadasHandler(UpCoordenadas r)
        {
            if (r.Index - 1 >= 0)
            {
                Coordenadas _m = new Coordenadas();
                _m = _CoordenadasList[r.Index];
                _CoordenadasList.RemoveAt(r.Index);
                _CoordenadasList.Insert(r.Index - 1, _m);
                WinFormsServicePort.Post(new
FormInvoke(delegate()
                {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                }));
            }
        }

        /// <summary>
        /// Decrease position of a mission in a list
        /// </summary>
        /// <param name="r"></param>
        void DownCoordenadasHandler(DownCoordenadas r)
        {
            if (r.Index + 1 <= _CoordenadasList.Count - 1)
            {
                Coordenadas _m = new Coordenadas();
                _m = _CoordenadasList[r.Index];
                _CoordenadasList.RemoveAt(r.Index);
                _CoordenadasList.Insert(r.Index + 1, _m);
                WinFormsServicePort.Post(new
FormInvoke(delegate()
                {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                }));
```

```
            }
        }

        #endregion
    }
}
```

Vemos que aquí es donde tenemos las partes declarativas más extensas así como la parte más extensa del código en si

Comenzamos con los usings. En esta ocasión tenemos una serie de novedades. Estas son:

```
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Xml;
using Navegador = Navegador;
using System.Security.Permissions;

using Microsoft.Ccr.Adapters.WinForms;
using System.Windows.Forms;
using types = Robosoft.RobuBOX.Core.Types.Proxy;

using ds = Microsoft.Dss.Services.Directory;
using submgr = Microsoft.Dss.Services.SubscriptionManager;

using laser =
Robosoft.RobuBOX.Generic.Devices.LaserSensor.Proxy;

using drive =
Robosoft.RobuBOX.Generic.Devices.DifferentialDrive.Proxy;

using loc =
Robosoft.RobuBOX.Generic.Processing.Signal.Localization.Proxy;
```

Como se ha comentado, las novedades en la parte declarativa del using son:

```
using types = Robosoft.RobuBOX.Core.Types.Proxy;

using ds = Microsoft.Dss.Services.Directory;
using submgr = Microsoft.Dss.Services.SubscriptionManager;

using laser =
Robosoft.RobuBOX.Generic.Devices.LaserSensor.Proxy;

using drive =
Robosoft.RobuBOX.Generic.Devices.DifferentialDrive.Proxy;

using loc =
Robosoft.RobuBOX.Generic.Processing.Signal.Localization.Proxy;
```

- Types hace referencia a una clase que nos da acceso a poder definer puntos en 2D en un Sistema de coordenadas cartesianas que nosotros elegimos.
- Ds y submgr como ahora veremos se utilizan para hacer la conexión y suscripción a los servicios que se van a utilizar, que son el servicio de motores diferencial, el láser y la localización odométrica
- Laser es el servicio del láser de rango, que como hemos visto nos devuelve la longitud de los ecos y el ángulo que forman con el eje x del robot
- Drive podemos intuir sin equivocarnos que hace referencia al servicio de control diferencial del robot que podemos configurar estableciendo una velocidad linear y una velocidad angular.
- Loc es la forma que tenemos de utilizar el sistema de localización odometrica.

Para empezar definimos como antes el nombre del servicio en el namespace y establecemos donde está el contract y definimos la clase principal que es la de NavegadorService, que tiene que heredar (como cualquier otra) DsspServiceBase por completo.

```
namespace Navegador
{
        [Contract(Contract.Identifier)]
        [DisplayName("Navegador")]
        [Description("Navegador service (no description provided)")]
        class NavegadorService : DsspServiceBase
        {
```

Ahora establecemos una variable _state de la clase del estado del navegador. Con esta variable tendremos acceso a todas y cada una de las variables que asignamos a esta clase NavegadorState en NavegadorTypes.cs desde cualquier parte del código. Esto cobra especial importancia ya que podemos comunicar las funciones internas del servicio mediante estas variables, por ejemplo y como se va a ver, asignamos valores en la parte del update del láser que son las velocidades requeridas del robot, y que habra que pasar a la parte del comportamiento periodico (periodicHandler) mediante dos variables que son vel y ang, de tipo double.

con _state definida, definimos ahora el puerto principal del servicio desde la clase Navegador operation, y que como vemos, llamaremos _mainPort.

```
private NavegadorState _state = new NavegadorState();

            [ServicePort("/Navegador", AllowMultipleInstances
= false)]
```

```
            private NavegadorOperations _mainPort = new
NavegadorOperations();
```

Establecemos los partners, que no son otra cosa que todo lo que se tendrá en cuenta o intervendra principalmente en el servicio, esto es, el differential drive, el laser y el localization.

Para el differential drive nos basta con definir su puerto (_drivePort), pero para el laser y el localizador tenemos que definir un puerto para las notificaciones. La razón es que en estos servicios o partners en este caso, vamos a utilizar updates, que son funciones que son llamadas cada vez que hay disponible nueva información, la cual ira al puerto que definimos de notificación.

La sintaxis es la siguiente:

```
 [Partner("differentialdrive", Contract =
drive.Contract.Identifier, CreationPolicy =
PartnerCreationPolicy.UseExisting)]
        drive.DifferentialDriveOperations _drivePort = new
drive.DifferentialDriveOperations();

        drive.DifferentialDriveState _driveState;

        private laser.LaserSensorOperations _laserPort;
        private laser.LaserSensorOperations _laserNotifyPort =
new laser.LaserSensorOperations();

        private loc.LocalizationOperations _locPort;
        private loc.LocalizationOperations _locNotifyPort =
new loc.LocalizationOperations();
```

No podemos olvidar definir un puerto para el HMI, ya que es totalmente necesario para que funcione y para poder acceder a el.

```
//internal port for HMI
        private HMIPort _hmiPort = new HMIPort();
```

definimos ahora una serie de variables.

El HMI en si, como _mainHMI, la lista de coordenadas _CoordenadasList, un puerto para el PeriodicHandler, que es un iterador fundamental en el programa cuya funcion es transmitir la información de la velocidad angular y lineal requerida continuamente a los motores, y por ultimo, una serie de strings que se utilizan para hacer la subscripcion y conexión al laser y la localización:

```
        //Variables
        HMI _mainHMI;

        public List<Coordenadas> _CoordenadasList = new
List<Coordenadas>();

        Port<bool> _periodicHandlerPort = new Port<bool>();

        string _laserSubMgr = string.Empty;
        string _laserSubscriber = string.Empty;

        string _localizationSubMgr = string.Empty;
        string _localizationSubscriber = string.Empty;
```

Definimos a continuación el constructor.

El constructor es la parte en la que definimos como comienza el programa, es decir, que es lo que va a hacer cuando comience. Ademas tambien tenemos que definir si es necesario el interleave.

Paso por paso, primero definimos el constructor: NavegadorService, al que pasamos como argumento de entrada el puerto de creacion

(creationPort) de la clase DsspServiceCreationPort, y la hacemos heredar base(creationPort).

Este procedimiento es totalmente repetitivo en cualquier servicio que se cree y se hace de la misma manera siempre.

Creamos ahora el comienzo del programa.

Escribimos protected override void Start() y comenzamos a definir el comportamiento de nuestro servicio.

```
            public NavegadorService(DsspServiceCreationPort
creationPort)
                  : base(creationPort)
            {
            }

            protected override void Start()
            {
```

Definimos ahora el base.Start()

Base.Start(); es la instrucion que se da que indica que el servicio se inicia. En otras palabras, la instrucion base.Start() da comienzo al programa y lo que le sigue será lo que hace el programa justo en el momento en el que empieza.

Primero indicamos que si el estado esta "vacio", es decir, no tiene valores asignados en sus variables, las inicializamos.

Primero definimos unas Coordenadas que tendrán como nombre "inicial" y estarán situadas en el origen de coordenadas (0,0). Despues las metemos en una lista y las añadimos a la lista de las listas de coordenadas del HMI, aunque estas no se mostrarán.

Lo que se acaba de hacer no es otra cosa de dar un valor por defecto a la lista de coordenadas a alcanzar. Es decir se ha añadido una mision a

realizar por defecto cuyo objetivo es llegar al punto (0,0), es decir, donde se encuentra el robot en su posición inicial.

Tambien indivamos que si la variable prev no tiene ningun valor asignado sera inicialmente true. Prev es una variable que nos ayuda a que la ventana de alerta de coordenadas alcanzada solo se lance una vez, ya que esta incluida en un enumerador y sin esta variable booleana de control la ventana aparecería infinitas veces sin cesar de aparecer jamas. Obviamente, el programa no respondería y el computador se quedaría sin memoria de cualquier tipo antes de que la ventana apareciera infinitas veces, llevando al servicio al fallo y mal funcionamiento por lo tanto, luego la variable prev aunque no lo parezva es de vital importancia.

```
            base.Start();

            if (_state == null)
            {
                _state.CoordenadasList = new
List<Coordenadas>();
                Coordenadas inicial = new Coordenadas();
                inicial.x = 0;
                inicial.y = 0;
                _state.prev = true;
                _state.CoordenadasList.Add(inicial);//add
coordenada to coordenadas list
                SaveState(_state);
            }

            foreach (Coordenadas inic in
this._state.CoordenadasList)
            {
                _CoordenadasList.Add(inic);
            }
```

Definimos el puerto principal interleave. Un interleave es un objeto en C# muy importante con una función principal en cualquier servicio de este tipo.

A veces, un servicio necesita escuchar a un número muy grande de puertos, como en este caso, y que todos los servicios que componen el servicio que se está programando puedan comunicarse entre ellos.

Cuando se transportan los mensajes a estos puertos la concurrencua es un grán problema y la estrategia que se utiliza para lidiar con esto es el interleave. Digamos que el interleave es una especie de semáforo que indica cuando puede pasar cada mensaje, además de definir la prioridad o el comportamiento que tiene cada uno de ellos.

La idea consiste en separar los Handlers (comportamientos) del servicio. Los handlers son las acciones que lleva a cabo el servicio por decirlo de una forma cercana.

Hay 3 tipos de Handlers con sus correspondientes mensajes:

- Un primer grupo que puede funcionar de forma simultanea (Concurrent)
- Otro que solo puede funcionar solo de uno en uno (Exclusive)
- El tercero engloba a toda operación que deba funcionar cuando el servicio se cierra (teardown)

Hay dos formas de crear el interleave; como un nuevo statement (new statement) o como se ha hecho en este proyecto, con la opción Arbiter.

Si se quisiera hacer como un nuevo Statement la sintaxis sería muy similar y tendria la forma:

```
New Interleave (
        New Teardown Receiver Group(),
    New Teardown Receiver Group(),
    New Teardown Receiver Group());
```

Gracias al interleave las funciones se llaman solas automáticamente en el momento en el que definimos que queremos que se llamen en la sintaxis de su descripción.

Es decir, para resumir, el Interleave recive los mensajes de los Handlers y no los transmite hasta el momento exacto en el que hay que transmitirlos.

En el caso del proyecto, como hay que utilizar un interleave con varios puertos en cada seccion (es el caso mas complejo de los interleaves) utilizamos la opcion de crear un puerto de interleave principal y le combinamos con todos los statements de Arbiter.Interleave (y no la opción new).

Definimos los 3 grupos.

Dentro del grupo de exclusividad añadimos los siguientes Handlers y de la siguiente forma:

- Un receptor para el PeriodicHandler. Como va a ser un enumerador, esto es, un iterador que realiza continuamente bucles, lo definimos indicandolo como Arbiter.ReceiveWithIterator. si no fuera un enumerador simplemente pondriamos Arbiter.Receive. despues de indicar que es un iterador, tenemos que indicar la condición o el comportamiento del Handler, esto significa indicar cuando el Interleave va a transmitir los mensajes que le llegan del Handler. En este caso del periodic handler, queremos que el interleave transmita los mensajes siempre que el boton start halla sido pulsado. Desde una vista mas general, el periodic handler es el handler que se dedica a transmitir la información a los motores de la velocidad linear y angular que deben tener. Sabiendo esto, definimos que el comportamiento o la condicion del periodic

handler sea una condicion booleana, es decir, el puerto se activará y el enumerador comenzará a realizar bucles cuando se postee que el puerto tiene la condición true. Despues, al indicar que cuando esta condicion es true, indicamos el puerto y el nombre del enumerador.

- Un receptor para los updates del laser. El laser no es un enumerador, y solo se va a activar cuando reciva información nueva. Esto quiere decir que solo va a dejar de llamarse a la funcion si el robot esta parado o si el laser está cegado. Incluimos la condición del laser, que es laser.Update, que indica lo dicho, esto es, que el handler se activa cuando recive nueva información. Despues se indica true para informar que la activacion es con el update e indicamos el puerto en el que se recive la informacion, que sera el que definimos para tal función, el notify port, y damos el nombre del handler.
- Se ha incluido como un comentario una variación que se podria realizar, y que seria incluir el handler de la localizacion en el interleave con la condicion de update, pero esto no es realmente necesario.
- Los dos handlers que quedan son comportamientos del HMI, y son el que carga las misiones, que veremos en mas profundidad a continuacion, y el drop from gui, que indica que cuando cerremos el GUI (guided user interface), que no es mas que una forma comun de llamar al interface creado, igual que HMI (human machine interface). Ambos se activan cuando se realizan esas acciones, ya bien sea pulsar el boton de load misions, o cerrar el gui. Ambos estan definidos en el puerto del HMI, que es donde se encuentran y tienen definido el nombre de su handler.

El grupo de teardown o cierre lo dejamos vacio: ()

El grupo de concurrencia tiene los siguientes Handlers (todos pertenecientes al HMI):

- Start mission que da comienzo al recorrido hasta el objetivo.
- Addcoordenada que añade a la lista una nueva lista con la coordenada.
- Resetlocalization, handler importante que resetea la posición actual del robot a la inicial con x, y y orientación a (0,0,0)
- Remove coordenadas que borra las coordenadas selecionadas en la caja de texto.
- Upcoordenadas que sube un puesto las coordenadas seleccionadas en la lista de listas
- Down coordenadas que desciende un puesto las coordenadas seleccionadas.

```
            MainPortInterleave.CombineWith(
                Arbiter.Interleave(
                    new TeardownReceiverGroup(),
                    new ExclusiveReceiverGroup(

Arbiter.ReceiveWithIterator<bool>(true, _periodicHandlerPort,
PeriodicHandler),
                        Arbiter.Receive<laser.Update>(true,
_laserNotifyPort, UpdateLaserHandler),
                        //Arbiter.Receive<loc.Update>(true,
_locNotifyPort, LocalizationUpdateHandler),
                        Arbiter.Receive<LoadMission>(true,
_hmiPort, LoadHandler),
                        Arbiter.Receive<DropFromGui>(true,
_hmiPort, DropFromGuiHandler)
                        ),
                    new ConcurrentReceiverGroup(
                        Arbiter.Receive<StartMission>(true,
_hmiPort, SendCoordenadasHandler),
                        Arbiter.Receive<AddCoordenadas>(true,
_hmiPort, AddCoordenadasHandler),
```

```
Arbiter.Receive<ResetLocalization>(true, _hmiPort,
ResetLocalizationHandler),

Arbiter.Receive<RemoveCoordenadas>(true, _hmiPort,
RemoveCoordenadasHandler),
                        Arbiter.Receive<UpCoordenadas>(true,
_hmiPort, UpCoordenadasHandler),
                        Arbiter.Receive<DownCoordenadas>(true,
_hmiPort, DownCoordenadasHandler)
                    )));
```

Lo siguiente que queremos que se active en nuestro programa es el propio HMI.

Para ello posteamos que queremos que empiece a funcionar la ventana principal del HMI con la siguiente instrucción:

```
            //launch Winform
            WinFormsServicePort.Post(new
RunForm(RunMainWindow));

            //Display the mission list in the HMI
            WinFormsServicePort.Post(new FormInvoke(delegate()
            {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
            }));
```

Tambien es interesante quedarnos en un fichero de texto con los valores que tomen la x la y y la alpha para poder graficarlos posteriormente y sacar conclusiones con el recorrido seguido.

Para ello creamos un fichero de texto y escribimos x y y alpha para posteriormente escribir debajo los valores que vaya dandonos el update de la odometria.

Las instrucciones las tenemos a continuación:

```
            string texto = "X Y alpha";
            System.IO.StreamWriter sw = new
System.IO.StreamWriter(@"F:\datos\datos.txt");
            sw.WriteLine(texto);
            sw.Close();
```

De igual forma, creamos otro fichero de texto que nos recoja los valores que toman en cada instante la velocidad lineal y la velocidad angular para aportar más información de el comportamiento que tiene el robot en las misiones que realiza.

```
string texto2 = "Vlinear Vangular";
            System.IO.StreamWriter sw2 = new
System.IO.StreamWriter(@"C:\datosv\datosv.txt", false);
            sw2.WriteLine(texto2);
            sw2.Close();
```

La siguiente parte tambien tiene grán importancia, ya que es la que nos va a indicar que la conexión a los 3 servicios tiene que comenzar: el differential drive, el laser y la localizacion.

Para ello utilizamos la funcion SpawnIterator y como argumento de entrada indicamos que handler queremos que empiece a funcionar:

```
            SpawnIterator(ConnectTodriveHandler);
            SpawnIterator(ConnectToLaserHandler);
            SpawnIterator(ConnectToLocalizationHandler);

      }
```

Se definiran ahora los tres handlers de conexión.

Comenzamos con el connect to drive handler, que realiza la conexión con el servicio del control del motor diferencial.

Lo primero que hay que comentar, es que se trata de un IEnumerator. IEnumerator es el interface base para todos los enumeradores (enumeradores no genéricos). De forma genérica se escribe IEnumerator<T>.

En este servicio se utilizarán con ITask en la parte declatariva T, por lo tanto tiene mas sentido comentar lo que son en si los IEnumerator<ITask>.

ITask es una tarea que constituye una unidad de codigo ejecutable usado para desarrollar operaciones. Por lo tanto, IEnumerator<ITask> es un iterador de tareas que usa como statement de devolucion las instrucciones yield return y/o yield break en lugar de las más conocidas return.

Lo que distingue a este método de otros, es el valor devuelto, es decir, indica que es un iterador de C# sobre instancias de CCR ITask al compilador, que puede que contenga los statements mencionados de yield.

Con esto en mente, cabe definir lo que son las instrucciones de devolucion yield return <expresion> y yield break.

Cuando se utiliza yield, indicas que el metodo, operador o get accesor (Get) donde aparece es un iterador.

Usando yield para definir al iterador, quitamos la necesidad de añadir una clase explicita extra.

Ahora bien, yield return se utiliza para devolver cada elemento uno cada vez, mientras que yield break finaliza el iterador, lo rompe, y sale de el sin devolver ningun valor.

Pasamos entonces a definir la conexión al control diferencial con un enumerador IEnumerator<ITask>, cuya tarea será esta misma de establecer la conexión con el servicio predefinido de control dferencial.

Lo que vamos a iterar es el handler de conexión, por lo que lo indicamos seguidamente del IEnumerator como IEnumerator<ITask> ConnectTodriveHandler().

La forma de definir las instrucciones para la conexión es muy similar para cualquier servicio desde cualquier servicio, teniendo una serie de código identico para todos los casos.

En este caso empezamos definiendo una variable booleana llamada error que utilizaremos despues para indicar que queremos hacer si la conexión se ha establecido sin problemas.

Definimos ahora 3 variables fundamentales en la conexión que vamos a llamar info, remoteport y query, de las clases ServiceInfoType, ds.DirectoryPort y query respectivamente.

Aunque se ha añadido al codigo una serie de comentarios para indicar que significa cada parte en ingles, se aclara que con info definimos a que servicio deseamos conectarnos, en este caso, al control diferencial, definiendo el contract (drive.Contract.Identifier)

Con remote port se declara un puerto donde el servicio va a funcionar y se le da el nombre del robot en un string. Para un fin más didactico, en el string del URI (el "nombre" que tiene el robot) se ha separado la cadena de caracteres donde diferenciamos el modelo del robot que estamos utilizando (robuLAB10) del resto del string, donde tambien es interesante ver que hay que añadir como siempre http://, y el puerto (50000).

Query simplemente constituye una "peticion" de la conexión.

```
IEnumerator<ITask> ConnectTodriveHandler()
        {
            bool error = false;
            //Informe to which sercice to connect
            ServiceInfoType info = new
ServiceInfoType(drive.Contract.Identifier);
            //Declare a port to the node/machine where the
service to connect is running, giving URI
            ds.DirectoryPort remotePort =
ServiceForwarder<ds.DirectoryPort>(new Uri("http://" +
"robuLAB10" + ":50000/directory"));
            //Declare the request
            ds.Query query = new ds.Query(new
ds.QueryRequestType(info));
```

Posteamos el quey en el remote port que habiamos definimos y nos quedamos a la espera de una respuesta que nos indica si la conexión se establece sin problemas o se produce algun error a lo largo de la misma.

La respuesta, por supuesto, la definimos con el statement yield return.

Para definir el yield return definimos el tipo de respuesta, que sera del tipo Arbiter.choice.

Ahora indicamos las opciones de los valores que puede tomar la respuesta, que son la conexión o el fallo de conexión: ds.QueryResponseType y W3C.Soap.Fault respectivamente.

Y dentro de las posibles devoluciones del yield return, definimos que pasaría según se de una opcion u otra.

Con la conexión sin errores, ds.QueryResponseType, queremos que se realice la conexión en el puerto de la respuesta (query.ResponsePort). Para definir lo que sucederia en este caso utilizamos el statement delegate. Escribimos delegate(ds.QueryResponseType response) e

indicamos que queremos que este presente en el puerto drive, que corresponde al control diferencial.

Ahora, para definir que es lo que sucedería en caso de producirse un error durante la conexión, utilizamos de nuevo el statement delegate para W3C.Soap.Fault fault e indicamos que la variable que definimos y que nos advertia de que se habian producido errores durante la conexión y esta no se habia producido correctamente (o ni siquiera llegado a producir) cambia a tener como valor true.

Una vez hemos escuchado la respuesta que nos da el servicio al que nos queremos conectar (sin errores o con errores), indicamos con un segundo yield return, que solo se activara en caso de no haberse producido ningun error, las dos segundas posibles opciones que podemos tener.

Por un lado, si vuelven a no producirse errores, queremos que la conexión se complete dando el valor de la respuesta al estado del control diferencial. Además, para tener la noción de que efectivamente se ha producido la conexión con éxito y sin ningún tipo de error, indicamos que se nos notifique mediante una ventana emergente con la función que definimos en el HMI loginfo, que nos de el mensaje: ("connected to DifferentilDrive service").

En caso de producirse error, la segunda opcion del yield return, W3C.Soap.Fault fault, nos mostrará tambien una ventana emergente, pero que en este caso indicará que la conexión ha fallado mediante el siguiente mensaje: ("Failed to get DifferentialDrive State").

Para terminar el IEnumerator<ITask> ya que no queremos que nos esté continua e indefinidamente conectando al servicio de control diferencial, introducimos el statement yield break para romper la enumeración y salir del bucle.

```
 //post
            remotePort.Post(query);

            //listen for a response Display in a textbox the
sentence in red
            yield return Arbiter.Choice<ds.QueryResponseType,
W3C.Soap.Fault>(query.ResponsePort,
                delegate(ds.QueryResponseType response)
                {
                    //instanciate the service port
                    _drivePort =
ServiceForwarder<drive.DifferentialDriveOperations>(response.R
ecordList[0].Service);

                },
                delegate(W3C.Soap.Fault fault)
                {
                    error = true;
                });

            if (!error)
            {
                yield return
Arbiter.Choice<drive.DifferentialDriveState, W3C.Soap.Fault>(
                _drivePort.Get(),
                delegate(drive.DifferentialDriveState
response)
                {
                    _driveState = response;
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                    {
                        _mainHMI.LogInfo("connected to
DifferentilDrive service");
                    }));
                },
                delegate(W3C.Soap.Fault fault)
                {
                    LogError("Failed to get DifferentialDrive
State");
                });
            }
            yield break;
        }
```

Igual que hemos hecho con el control diferencial, realizamos la conexión con el servicio de localización y del laser de la misma forma:

```
IEnumerator<ITask> ConnectToLaserHandler()
        {
            bool error2 = false;
            ServiceInfoType info = new
ServiceInfoType(laser.Contract.Identifier);
            //Change "laser" by the alias of the service to
connect
            info = new
ServiceInfoType(laser.Contract.Identifier);
            //Declare a port to the node/machine where the
service to connect is running, giving URI
            ds.DirectoryPort remotePort2 =
ServiceForwarder<ds.DirectoryPort>(new Uri("http://" +
"robuLAB10" + ":50000/directory"));
            //Declare request
            ds.Query query2 = new ds.Query(new
ds.QueryRequestType(info));
            //post
            remotePort2.Post(query2);

            //listen for a response
            yield return Arbiter.Choice<ds.QueryResponseType,
W3C.Soap.Fault>(query2.ResponsePort,
                delegate(ds.QueryResponseType response)
                {
                    error2 = false;
                    //instanciate services port: receive the
two intance of laser
                    _laserPort =
ServiceForwarder<laser.LaserSensorOperations>(response.RecordL
ist[0].Service);
                },
                delegate(W3C.Soap.Fault fault)
                {
                    error2 = true;
                    _laserPort = null;
                    LogInfo(LogGroups.Console, "ERROR
connecting to Laser service");
                });
            if (error2 == false)
            {
```

```
                //Subscribe with the odometry
                yield return
Arbiter.Choice<SubscribeResponseType, W3C.Soap.Fault>(
                    _laserPort.Subscribe(_laserNotifyPort),
                    delegate(SubscribeResponseType response)
                    {
                        //Make he link between the message
receive and the handler
                        LogInfo("subscribed to Laser
service...");
                        _laserSubMgr =
response.SubscriptionManager;
                        _laserSubscriber =
response.Subscriber;


Activate(Arbiter.Receive<laser.Update>(false,
_laserNotifyPort, UpdateLaserHandler));

                        WinFormsServicePort.Post(new
FormInvoke(delegate()
                        {
                            _mainHMI.LogInfo("connected to
Laser service");
                        }));
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogInfo("Failed to subscribe to
Laser...");

                    });
            }
        }

        IEnumerator<ITask> ConnectToLocalizationHandler()
        {
            bool error = false;
            //Change "loc" by the alias of the service to
connect
            ServiceInfoType info = new
ServiceInfoType(loc.Contract.Identifier);
            //Declare a port to the node/machine where the
service to connect is running, giving URI
```

```
            ds.DirectoryPort remotePort =
ServiceForwarder<ds.DirectoryPort>(new Uri("http://" +
"robuLAB10" + ":50000/directory"));
            //Declare the request
            ds.Query query = new ds.Query(new
ds.QueryRequestType(info));
            //post
            remotePort.Post(query);
            //listen for a response
            yield return Arbiter.Choice<ds.QueryResponseType,
W3C.Soap.Fault>(query.ResponsePort,
                delegate(ds.QueryResponseType response)
                {
                    //to be alert when odometry is updated
                    _locPort =
ServiceForwarder<loc.LocalizationOperations>(
                        response.RecordList[0].Service);
                },
                delegate(W3C.Soap.Fault fault)
                {
                    error = true;
                    _locPort = null;
                });

            if (!error)
            {
                SubscribeRequestType request = new
SubscribeRequestType();
                request.Subscriber = "http://" + "robuLAB10" +
":50000/directory";

                //Subscribe with the localization
                yield return
Arbiter.Choice<SubscribeResponseType, W3C.Soap.Fault>(
                    _locPort.Subscribe(_locNotifyPort),
                    delegate(SubscribeResponseType response)
                    {
                        _localizationSubMgr =
response.SubscriptionManager;
                        _localizationSubscriber =
response.Subscriber;


Activate(Arbiter.Receive<loc.Update>(true, _locNotifyPort,
LocalizationUpdateHandler));
```

```
                        WinFormsServicePort.Post(new
FormInvoke(delegate()
                        {
                            _mainHMI.buttonResetOdo.Enabled =
true;
                        }));
                        WinFormsServicePort.Post(new
FormInvoke(delegate()
                        {
                            _mainHMI.LogInfo("connected to
Localization service");
                        }));
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogInfo("Failed to subscribe to
Localization...");
                    });
            }
        }
```

Nos quedan aun handlers del HMI sin definir, así que ahora se describirá su comportamiento.

Los handlers drop y save tienen como funcion cerrar y desconectar el servicio y lanzar las coordenadas respectivamente.

Ambas son de tipo void ya que no queremos que nos devuelvan nada. En el caso de drop simplemente introducimos como argumento de entrada DropFromGui d, que ya definimos. Esto cerrara y desconectara el servicio si el HMI es cerrado (_mainPort.Post(new DsspDefaultDrop());)

```
#region drop & save

        /// <summary>
        /// Drop message sent from gui
        /// </summary>
        /// <param name="d"></param>
        void DropFromGuiHandler(DropFromGui d)
        {
```

```
            _mainPort.Post(new DsspDefaultDrop());
        }
```

Lanzar las coordenadas significa que el servicio ya save cual es su objetivo, donde se tiene que dirigir. cuando nosotros introducimos las coordenadas en el HMI y las añadimos lo unico que hemos hecho es añadir una lista a la lista de las listas, valga la redundancia, pero no damos información al servicio de donde queremos que vaya.

Sin embargo, cuando pulsamos el botón load mision, según se define a continuación, lanzamos las coordenadas de la lista de coordenadas que se encuentra en la posición [0], es decir, la primera de la lista.

Ahora si que hemos dado información al robot de donde queremos que vaya. De hecho, el comportamiento de este handler es basicamente guardar en las variables del estado del navegador x e y las coordenadas objetivo como vemos:

```
        /// <summary>
        /// Save current missions in state xml file
        /// </summary>
        /// <param name="s"></param>
        void LoadHandler(LoadMission s)
        {

            _state.metax = _CoordenadasList[0].x;
            _state.metay = _CoordenadasList[0].y;
            _state.prev = true;
            _state.contador = 1;

            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
                _mainHMI.LogInfo("Coordenadas lanzadas");
            }));

        }

        #endregion
```

Como apunte añadiré que las zonas de código region no son mas que separadores que organizan el mismo. Cuando queremos empezar una region escribimos #region nombre_de_la_región, y para finalizarla escribimos solo #endregion.

De este modo la información queda totalmente ordenada en regiones de nuestro interes que nosotros mismos hemos definido en modo de desplegables y no solo contamos con los desplegables que nos ofrecen las clases, funciones, usings, namespaces… etc.

Sin más preambulos, pasamos a definir una de las partes más importantes del programa, que además, contiene la lógica principal del comportamiento de la navegación, y que es la función definida en el interleave que se activa mediante updates, UpdateLaserHandler.

El update laser handler es el handler que define el comportamiento del laser de rango, o más bien, el tratamiento de la información que recibimos del laser de rango.

Primero definimos el handler, que será por supuesto de tipo void. Añadiremos además que es público para evitar posibles errores de acceso al handler, ya que si no incluimos nada, C# considera que por defecto es de tipo private.

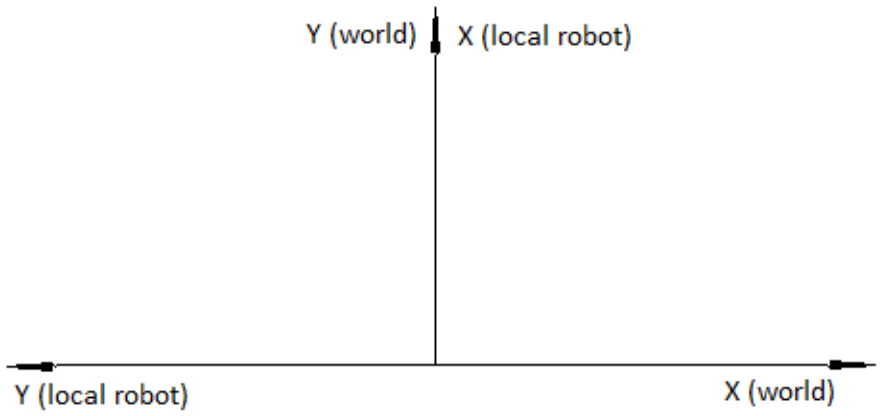
Una vez introducido el nombre del handler, añadimos como argumento de entrada el update de la forma normalizada (laser.Update update).

Y ahora, comenzamos a modelar su comportamiento.

```
public void UpdateLaserHandler(laser.Update update)
        {
```

Vamos a definir primero unas variables que nos ayudarán a ver de forma más clara los cálculos realizados para procesar la información, y que esta se pueda realizar en pasos y no de golpe, cosa que podria llevar a posibles errores.

Xrobot será la coordenada x en la que se encuentra el robot en el sistema de coordenadas global y no el local del robot, el cual viene dado por el update del handler de localización, el cual veremos posteriormente. Como se ha visto previamente, el servicio de localización nos da unas coordenadas locales giradas pi/2 radianes en la direccion positiva del eje z tomando como referencia el sistema de coordenadas local, por lo que xrobot será igual a menos la coordenada y local del robot (figura 38):



(figura 38)

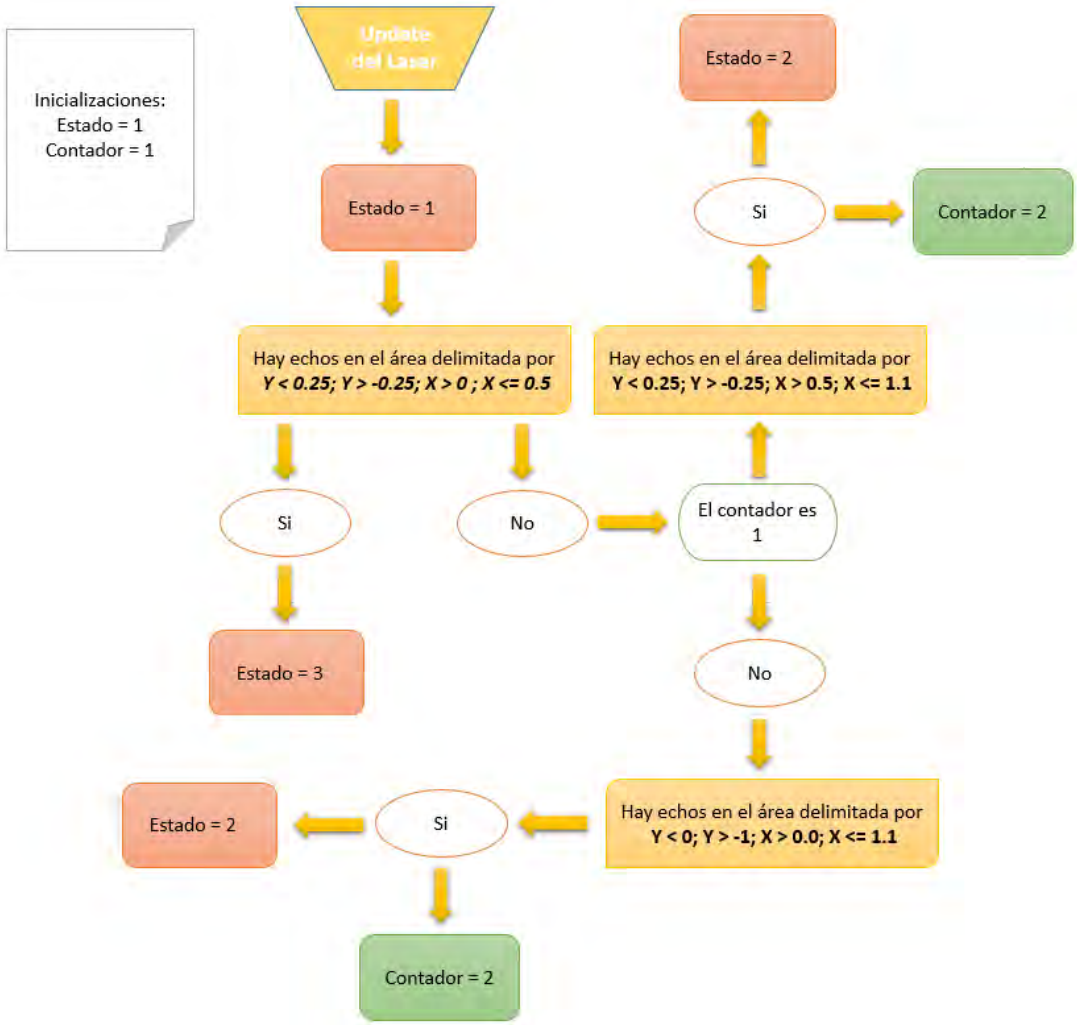
De igual forma definimos yrobot como la coordenada local x del robot.

Ahora añadimos dos variables que nos ayudarán a calcular el ángulo de error entre la orientación actual del robot y el objetivo, y que definen el vector que une las coordenadas actuales del robot y las del objetivo; xob e yob.

Por utltimo definimos otra variable de tipo entero llamada estado y la inicializamos a 1 cada vez que halla un update. Esta variable es de control y modela el comportamiento de la navegación como ahora se explicará.

Primero se introducirá un esquema con el algoritmo desarrollado para la navegación, y luego se explicará el código creado para llevarlo a cabo y la lógica empleada.

El esquema es el siguiente (figura 39):



(figura 39)

```
double xrobot = -_state.y;
            double yrobot = _state.x;
            double xob = _state.metax - xrobot;
            double yob = _state.metay - yrobot;
            int estado = 1;
```

Para conseguir saber si alguno de los ecos del laser se encuentra en el area definida en el esquema del algoritmo, desarrollamos el siguiente código.

Primero vamos observar todos y cada uno de los ecos que nos da el laser de rango del robot, y para ello utilizamos la instrucción foreach, que nos recorre todos los valores que contiene una lista o un vector, según sea el caso.

En nuestro caso la sintaxis será : foreach (laser.Echo E in update.Body.Echos)

Con Laser.Echo indicamos que estamos observando los ecos del servicio del laser que hemos llamado también laser. E es la variable que creamos y que va a cambiar de valor constantemente en cada bucle del foreach ademas de en cada bucle del update, y tenemos que indicar que esta variable tiene el valor cada vez de los distintos ecos que contiene el cuerpo del laser en cada update.

Ahora dentro de cada una de las opciones que va a recorrer el foreach, que son todos los valores de los ecos del laser, introducimos una variable de la clase que agregamos al principio llamada types.

En types, escogemos la variable del tipo point2d y la llamamos pointCartesian. Esta variable tiene un valor asignado para la componente x y otro para la y.

Para point cartesian x simplemente multiplicamos la distancia de cada eco por el coseno del angulo que forma, pero cuidado, no se debe pensar que esta coordenada va a estar referida al sistema de coordenadas global, si no al local del robot, ya que el laser tiene asignadas el mismo sistema de referencia local que el sistema de localización odométrica.

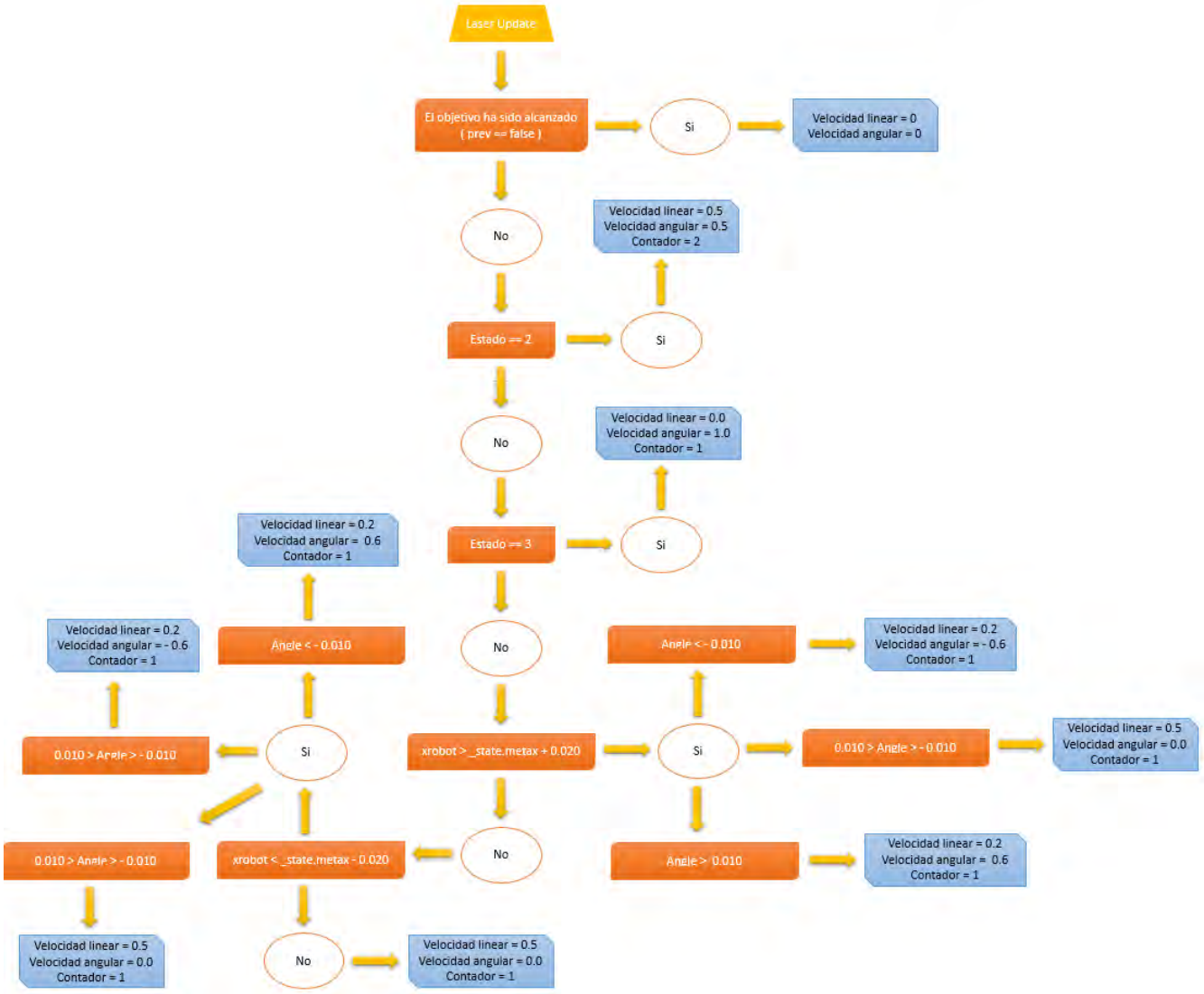
Esto hay que tenerlo en cuenta en la lógica if, que es donde preguntamos a los ecos si detectan algun cuerpo no esperado en el área definida. De igual forma que hemos hecho anteriormente, el cambio para expresar las coordenadas en el sistema de referencia world consiste simplemente en girar el sistema de coordenadas local pi/2 radianes en la direccion negativa del eje z. Así, pointCartesian.x será la coordenada y, y pointCartesian.y será la coordenada −x.

```
          foreach (laser.Echo E in update.Body.Echos)
            {
                types.Point2D pointCartesian = new
types.Point2D();
                pointCartesian.X = E.Distance *
Math.Cos(E.Angle);
                pointCartesian.Y = E.Distance *
Math.Sin(E.Angle);

                if (pointCartesian.Y < 0.25 &&
pointCartesian.Y > -0.25 && pointCartesian.X > 0 &&
pointCartesian.X <= 0.5)

                    estado = 3;

                else if (_state.contador == 1)
                {
                    if (pointCartesian.Y < 0.25 &&
pointCartesian.Y > -0.25 && pointCartesian.X > 0.5 &&
pointCartesian.X <= 1.1)
```

```
                    {
                        estado = 2;
                        _state.contador = 2;
                    }

                }

                else if (_state.contador == 2)
                {
                 if (pointCartesian.Y < 0 && pointCartesian.Y
> -1 && pointCartesian.X > 0.0 && pointCartesian.X <= 1.1)
                    {
                        estado = 2;
                        _state.contador = 2;
                    }

                }

            }
```

Ahora que la lógica del algoritmo que indica el estado en la que se encuentra el robot está definida, podemos pasar al algoritmo que define la velocidad lineal y angular que va a tener el robot:

El esquema es el siguiente (figura 40):



(figura 40)

Con estos 2 algoritmos trabajando de forma simultanea nos aseguramos que el robot siempre llega al objetivo superando sin chocarse con ningún obstáculo intermedio.

Las velocidades angulares y lineales requeridas las metemos en las variables del estado vel y ang del navegador para poder pasarselas al periodic handler.

Por otra parte, prev es la variable de control que comentamos que indicaba que el robot habia llegado al objetivo.

Angle es una variable de tipo double y cuyo valor es el ángulo que forma el robot con su orientación en cada momento y el objetivo, es decir, será 0 cuando el robot esté literalmente "mirando" a su objetivo.

La forma de calcularlo depende de si el robot está situado a la derecha o a la izquierda del objetivo, es decir, que tenemos que comparar las coordenadas x de ambos.

Una vez hecho calculamos el angulo que forma el vector que une el robot y el objetivo y obtenemos su angulo complementario restandoselo a pi/2.

De esta forma, tenemos un ángulo que podemos llamar alpha, por ejemplo.

Dependiendo ahora de si la coordenada x del robot es superior o inferior a la del objetivo, debemos restarle o sumarle respectivamente la orientación propia del robot obtenida por odometría para obtener el ángulo "angle". El cálculo de angle se verá mas detalladamente en los anejos.

Así, si el robot no está "mirando" al objetivo, corregimos su orientación mientras además sigue avanzando hasta el objetivo.

```
            if (_state.prev == false)
            {
                _state.vel = 0.0;
                _state.ang = 0.0;

            }
            else if (_state.prev == true)
            {
                if (estado == 2)
                {
                    _state.vel = 0.5;
                    _state.ang = 0.5;
```

```
                    _state.contador = 2;
                }
                else if (estado == 3)
                {
                    _state.vel = 0.0;
                    _state.ang = 1.0;
                    _state.contador = 1;
                }
                else if (xrobot > _state.metax + 0.020)
                {
                    double angle = (Math.PI / 2) -
Math.Atan(yob / xob) - _state.theta;

                    if (angle < -0.2)
                    {
                        _state.vel = 0.2;
                        _state.ang = -0.6;
                        _state.contador = 1;
                    }

                    else if (angle > 0.2)
                    {
                        _state.vel = 0.2;
                        _state.ang = 0.6;
                        _state.contador = 1;
                    }
                    else
                    {
                        _state.vel = 0.5;
                        _state.ang = 0;
                        _state.contador = 1;
                    }
                }
                else if (xrobot < _state.metax - 0.020)
                {
                    double angle = (Math.PI / 2) -
Math.Atan(yob / xob) + _state.theta;

                    if (angle < -0.2)
                    {
                        _state.vel = 0.2;
                        _state.ang = 0.6;
                        _state.contador = 1;
                    }
```

```
                    else if (angle > 0.2)
                    {
                        _state.vel = 0.2;
                        _state.ang = -0.6;
                        _state.contador = 1;
                    }
                    else
                    {
                        _state.vel = 0.5;
                        _state.ang = 0;
                        _state.contador = 1;
                    }
                }
                else
                {
                    _state.vel = 0.5;
                    _state.ang = 0;
                    _state.contador = 1;
                }
            }

                /* pruebas
                if (_state.estado == 1)
                {
                    _state.vel = 0.2;
                    _state.ang = 0;
                }
                else
                {
                    _state.vel = 0;
                    _state.ang = 0;
                }
                */
}
```

Vamos a hablar ahora de como funciona el algoritmo;

Como podemos ver, el estado 2 se corresponde con el momento en el que el robot avista un objetivo a una distancia lo suficientemente prudencial como para poder intentar evitarlo sin tener que dejar de avanzar en la coordenada y.

No obstante, el robot no solo intenta evitar el objetivo cuando lo localiza, si no que además entra entra en un tercer estado que podriamos considerar una extensión del 2:

Cuando el objetivo es localizado, y se activa el estado 2, ademas, el contador toma como valor 2, por lo que en el siguiente update del laser, el área que activa el estado 2 se agranda para que el objeto a evitar siga siendo visto una vez iniciado el giro para evitarlo. La razón para que este estado no sea el unico que active el estado 2 es que no queremos que el robot localice objetos que no tiene que evitar debido a este area agrandada que podria detectar objetos que no se encuentran en frente del robot, pero si objetos que supongan un riesgo de choque con el. De forma esquemática, esto es lo que pasa (figura 41):

(figura 41)

Aun así, ambas areas activan el mismo estado, por lo que el movimiento que demandan es el mismo.

El estado 2 demanda que la velocidad lineal y angular tengan como valor 0.5, es decir, 0.5 m/s y 0.5 rad/s respectivamente.

Esto determina que el robot va a describir como trayectoria una circunferencia de radio unidad por la fórmula $\vec{v} = \vec{w} \cdot R$

Como el robót comienza a girar cuando localiza el objetivo, vemos que esta trayectoria comienza a una distancia de 1.1 metros del objetivo, que es más que suficiente para que no halla choque, aunque esta situación jamas se daría gracias al estado de seguridad 3.

No obstante, se dedicarán anexos específicos para el cálculo de la trayectoria y la graficación de las mismas en casos reales mediante los datos recogidos.

Como se apuntaba hace un momento, el estado de seguridad 3 impide el choque en cualquiera de las situaciones, incluso cuando la reaccion del robot sufre un retardo. El estado 3 solo consta de velocidad angular, siendo la lineal 0. Lo que esto significa, esque el radio de la trayectoria que el robot sigue es 0, es decir, gira sobre el punto medio entre sus ruedas de traccion diferencial.

No esta de mas decir que todos los estados que puede adoptar el robot, excepto el 2 claro esta, reinician el contador a 1, para que así el robot vuelva a la situación de estado 2 pero con el contador a 1, es decir, solo localiza obstáculos forntales, y para los laterales está literalmente "ciego".

Cuando no hay ningún objeto delante del robot pueden darse 3 casos mas:

Que el robot no este orientado al objetivo, ya esté a la derecha o izquierda del mismo, o que si lo esté.

En este último caso, el "else" final, el robot solo avanza con velocidad lineal, sin velocidad angular. Como vemos, esto da lugar a una trayectoria con radio infinito, es decir, una linea recta.

En este estado el robot avanza sin problemas hasta que alcance el objetivo, momento en el que se detiene.

Pero puede darse que el robot efectivamente no esté orientado al objetivo. De hecho, esta situación se dará siempre que el robot lo hubiera estado y halla tenido que evitar un obstáculo.

En estos casos, puede darse que el robot esté situado con una coordenada x mayor o menor que la del objetivo, y que a su vez, como habiamos visto, el angulo que forma su orientación con el objetivo sea positivo o negativo, pero no 0.

En estos casos, el robot siempre realizará la misma trayectoria, cambiando únicamente si el giro lo realiza hacia la izquierda o la derecha, siempre siendo el que favorezca a que el robot se reoriente al objetivo.

Pues bien, este giro tendrá como velocidad angular 0.6, y velocidad lineal 0.2.

Esto produce un giro bastante cerrado, con radio 1/3 metros. Además, es un giro lento, debido a los bajos valores de las velocidades.

Esto se hace para que por muy cerca que esté el robot del objetivo en cualquier instante, sea capaz de reorientarse siempre a el, cosa que seria imposible si el radio fuera igual o mayor que 1. De hecho, es obvio que la reorientación se producirá más rápido cuanto más cerrado sea el giro, esto es, menor sea el radio. El hecho de que se haga tan lentamente se debe a que si el giro se produciese de forma rápida sería bastante dificil

conseguir llevar al robot a la posición de equilibrio consistente en que el angulo que buscamos tenga el robot con el objetivo sea 0, ya que cuando lo consiguiesemos, si en el siguiente instante no se produciera el update en el laser, las velocidades no cambiarían, y el ángulo cambiaría de signo pero jamás se haría 0. De hecho, cuanto más cerrada sea la curva, hemos visto que más rápido cambia el ángulo, luego sería incluso más dificil llegar así a la posición de equilibrio.

Para concluir esta parte, se debe añadir que todos los valores tienen tolerancias para evitar problemas de sincronía debidos a más que posibles retrasos en los updates de los handlers.

Asi, el angulo "angle", cuya condición de equilibrio debería ser simplemente cuando se iguala a 0, cambia a tener una tolerancia en la que la condición de equilibrio es cuando se encuentra entre el intervalo +- 0.2 radianes.

De la misma manera, el objetivo se alcanza cuando el robot llega a cualquier punto del área definida por las coordenadas objetivo con un intervalo de +- 0.200 metros.

También se ha aplicado una tolerancia de +- 0.020 metros a la condición que indica al robot que debe reorientarse cuando se encuentra a derecha o izquierda del robot, para evitar que lo haga si no esta perfectamente alineado con el (situación totalmente improbable en la práctica) cuando comienza el movimiento en los casos en los que la coordenada x del objetivo es 0, coincidiendo con la del robot inicialmente.

Tenemos que definir también la inicialización del HMI cuando comienza el programa, ya que la función aún no la habiamos definido:

```
//Function to instatiate the HMI window
        HMI RunMainWindow()
        {
```

```
            _mainHMI = new HMI(_hmiPort);
            return _mainHMI;
        }
```

Pasamos ahora al handler que pasa la información a los motores de como deben actuar. Este handler es el que hemos llamado PeriodicHandler.

Se trata de un IEnumerator, ya que va a estar continuamente llamandose a si mismo, así que lo definimos como tal. Escribimos IEnumerator<ITask> y damos su nombre, PeriodicHandler. Los argumentos de entrada que hay que darle, que indican cuando puede empezar a funcionar, será la variable booleana que definimos y que queríamos que cambiase a true cuando pulsabamos el botón start para que el robot comenzase el movimiento.

```
        IEnumerator<ITask> PeriodicHandler(bool b)
        {
```

Definimos una variable request usando drive.DriveRequest. request va a contener las velocidades lineales y angulares requeridas.

Incluimos también la condicion de llegada con la tolerancia ya mencionada, e indicando que request tiene que tomar como valores 0.0 para tanto la velocidad lineal como para la angular.

Además, vamos a utilizar la variable prev para indicar no solo que hemos alcanzado el objetivo al handler del update del laser también, si no para mostrar una ventana emergente que nos indica que hemos alcanzado las coordenadas, para que esta se muestre solo una vez y no infinitas, dando como resultado un fallo irreparable durante el programa, el cual haría que la memoria disponible se acabase, provocando que el servicio no responda.

```
 drive.DriveRequest request = new drive.DriveRequest();

            if (-_state.y < _state.metax + 0.2 && -_state.y >
_state.metax - 0.2 && _state.x < _state.metay + 0.2 &&
_state.x > _state.metay - 0.2)
            {
                request.LinearSpeed = 0.0;
                request.AngularSpeed = 0.0;

                if (_state.prev == true)
                {
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                    {
                        _mainHMI.LogInfo("Coordenadas
alcanzadas");
                    }));
                    _state.prev = false;
                }
            }
```

Si no, es decir, las condiciones no se han alcanzado aun, indicamos que request.LinearSpeed sea _state.vel, definida como vimos en el laserUpdateHandler, y que request.AngularSpeed sea _state.ang de la misma manera.

```
else
            {
                request.LinearSpeed = _state.vel;
                request.AngularSpeed = _state.ang;
            }
```

Además, es muy interesante que en este handler cada vez que halla una iteración, escribamos en el archivo de texto datosv.txt la velocidad linear y angular que tenemos en cada momento durante el recorrido del robot en su misión de alcanzar las coordenadas objetivo, para así poder analizar con una rica variedad de detalles el comportamiento del mismo.

Cuando escribimos System.IO.StreamWriter(@"C:\datosv\datosv.txt", true); true se utiliza para indicar que queremos anotar un nuevo dato, y no sobreescribir los existentes, situación en la que escribiriamos false.

```
            System.IO.StreamWriter sw = new
System.IO.StreamWriter(@"C:\datosv\datosv.txt", true);
            sw.WriteLine(request.LinearSpeed + " " +
request.AngularSpeed);
            sw.Close();
```

Ahora pasamos a definir el yield return.

De nuevo utilizamos Arbiter.Choice para modular las opciones que puede tomar la respuesta, así que escribimos <DefaultSubmitResponseType (para definir la opción en la que se envia el mensaje sin problemas), W3C.Soap.Fault (para definir la opción en la que se produzca algún error durante la emisión del mensaje>.

Con esto ya podemos definir el yield return, en el que simplemente nos basta con postear la variable request en el puerto del control diferencial drive.

Despues definimos las dos opciones (envio del mensaje con o sin errores) en los que ni si quiera es necesario escribir nada mas.

Lo unico que si es importante añadir es un yield break al final del enumerador.

```
                yield return
Arbiter.Choice<DefaultSubmitResponseType, W3C.Soap.Fault>(
                    _drivePort.Drive(request),
```

```
                    delegate(DefaultSubmitResponseType
response)
                    {
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                    });


        yield return Arbiter.Receive(false, TimeoutPort(80),
delegate(DateTime d) { });

            _periodicHandlerPort.Post(true);

            yield break;
        }
```

Pasamos pues a definir los Dss handlers. Separaremos esta region del resto con la instrucción que se vió #region.

```
#region Dss handlers
```

Comenzamos con el Get. La sintaxis es muy simple y practicamente repetitiva para cualquier servicio.

No obstante no se debe pensar que por ello estos handlers son menos importantes o incluso prescindibles, ya que de hecho son totalmentes necesarios para que el servicio funcione.

El servicio por si mismo puede arrancar, pero no podrá tener funcionalidad alguna sin estos handlers, ya que son básicos para las operaciones que queramos realizar.

Estos handlers son enumeradores, pero podemos definir la respuesta con un yield break.

Get es necesario si queremos obtener información de cualquier servicio.

```
        /// <summary>
        /// Get Handler
        /// </summary>
        /// <param name="get"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Concurrent)]
        public virtual IEnumerator<ITask> GetHandler(Get get)
        {
            get.ResponsePort.Post(_state);
            yield break;
        }
```

Replace es necesario para poder reemplazar valores en los servicios

```
        /// <summary>
        /// Replace Handler
        /// </summary>
        /// <param name="replace"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Exclusive)]
        public virtual IEnumerator<ITask>
ReplaceHandler(Replace replace)
        {
            _state = replace.Body;

replace.ResponsePort.Post(DefaultReplaceResponseType.Instance)
;
            yield break;
        }
```

Por ultimo tenemos el drop handler, que es necesario para poder cerrar servicios y desconectarnos de ellos.

Este ultimo tiene una sintaxis mas compleja y variopinta para diferentes servicios.

Es el único con comportamiento teardown (uno de los 3 grupos explicados en el interleave).

Para este servicio, tendremos que borrar la subscripción de los 3 servicios, el control diferencial, el laser y la localización.

```
  /// <summary>
        /// Drop Handler
        /// </summary>
        /// <param name="replace"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Teardown)]
        public virtual IEnumerator<ITask>
DropHandler(DsspDefaultDrop drop)
        {
            DirectoryDelete();
            Shutdown();

drop.ResponsePort.Post(DefaultDropResponseType.Instance);

            if (_laserSubMgr != string.Empty)
            {
                //unsubscribe to laser
                submgr.DeleteSubscription delete = new
submgr.DeleteSubscription();
                delete.Body = new
submgr.DeleteSubscriptionMessage(_laserSubscriber);

ServiceForwarder<submgr.SubscriptionManagerPort>(_laserSubMgr)
.Post(delete);
                yield return
Arbiter.Choice<DefaultDeleteResponseType, W3C.Soap.Fault>(
                    delete.ResponsePort,
                    delegate(DefaultDeleteResponseType
response)
                    {
                        _laserSubMgr = string.Empty;
                        _laserSubscriber = string.Empty;
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogError("Error deleting subscription
to laser");
                    });
            }
            if (_localizationSubMgr != null)
```

```
            {
                //unsubscribe to localization
                submgr.DeleteSubscription delete = new
submgr.DeleteSubscription();
                delete.Body = new
submgr.DeleteSubscriptionMessage(_localizationSubscriber);

ServiceForwarder<submgr.SubscriptionManagerPort>(_localization
SubMgr).Post(delete);
                yield return
Arbiter.Choice<DefaultDeleteResponseType, W3C.Soap.Fault>(
                    delete.ResponsePort,
                    delegate(DefaultDeleteResponseType
response)
                    {
                        _localizationSubMgr = string.Empty;
                        _localizationSubscriber =
string.Empty;
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogError("Error deleting subscription
to localization");
                    });
            }

            yield break;
        }
#endregion
```

Por último nos queda definir los handlers restantes del HMI y el update de la localización:

Comenzamos con el boton que definimos para reiniciar la odometria. Esta fución es extremadamente útil en la práctica, y nos permite devolver la localización del robot a sus valores iniciales (0,0,90)

Utilizamos el dsshandler replace para realizar esta tarea. Definimos la variable reset de replace y la aplicamos a todo el cuerpo de la localizacion (x y theta), lo posteamos en el puerto de la localización, y llamamos a la

función que introduce los valores en el campo de texto del HMI para visualizar el reseteo., que es UpdateOdo.

```
#region StartMission, AddMission, RemoveMission, Up and Down, reset and update loc

        /// <summary>
        /// Reset odometry
        /// </summary>
        /// <param name="r"></param>
        void ResetLocalizationHandler(ResetLocalization r)
        {
            loc.Replace reset = new loc.Replace();
            reset.Body.X = 0;
            reset.Body.Y = 0;
            reset.Body.Theta = 0;
            this._locPort.Post(reset);

            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
                _mainHMI.UpdateOdo(0,0,90);
            }));

        }
```

Por supuesto entonces, tenemos que definir tambien la función que actualiza el UpdateOdo.

LocalizationUpdateHandler es el handler de update que se llama cada vez que la localizacion tiene un update, por lo que lo definimos con el argumento de entrada (loc.Update up)

En la definicion de este handler tenemos que meter en las variables del estado del navegador x y y theta los updates del cuerpo de la localización (x y y theta también) para acceder desde los otros handlers que ya hemos definido.

Hacemos el cambio de coordenadas a las globales como siempre con la variación que pasamos la orientación de radianes a grados sexagesimales multiplicando por 180 y dividiendo por pi.

Además aquí es donde vamos a escribir cada vez que halla un update en el fichero de texto datos.txt los valores de la x y y alpha que va tomando el robot durante su recorrido hacia las coordenadas objetivo, para poder despues graficar el recorrido y orientación, incluso en función del tiempo en 3 dimensiones.

```
        /// <summary>
        /// Occure when odometry is updated
        /// </summary>
        /// <param name="up">Odometry</param>
        void LocalizationUpdateHandler(loc.Update up)
        {
            _state.x = up.Body.X;
            _state.y = up.Body.Y;
            _state.theta = up.Body.Theta;

            double xpos = -_state.y;
            double ypos = _state.x;
            double alpha = _state.theta * (180 / Math.PI) +
90;

            System.IO.StreamWriter sw = new
System.IO.StreamWriter(@"C:\datos\datos.txt", true);
            sw.WriteLine(xpos + " " + ypos + " " + alpha);
            sw.Close();

            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
                _mainHMI.UpdateOdo(xpos, ypos, alpha);
            }));
        }
```

Como vemos en el summary que se ha escrito para definir el comportamiento de este handler, lo que se está definiendo aquí es lo que ocurre al pulsar el botón start.

Cuando pulsamos el botón start, queremos que la información a los motores empiece a fluir, y para eso es necesario que el periodic handler tenga su condicion en true.

Asi que lo único que hay que hacer es postear el valor true en el puerto de este handler.

Además, se ha descrito que al pulsarse el botón, en el campo de texto se muestren las coordenadas objetivo actuales que el robot está tratando de alcanzar.

```
        /// <summary>
        /// User clicks on HMI: the next mission is sent send
next mission
        /// </summary>
        /// <param name="s"></param>
        void SendCoordenadasHandler(StartMission s)
        {
            _periodicHandlerPort.Post(true);

_mainHMI.DisplayCurrentCoordenadas(_CoordenadasList[0]);
        }
```

Aqui tenemos los handlers add, remove, up y down, que añaden quitan suben y bajan respectivamente la lista de coordenadas seleccionada en el HMI.

Su sintaxis es relativamente sencilla y no se ha tenido que utilizar demasiada contidad de código para definirlos.

Despues de realizar cualquiera de estas acciones, debemos volver a publicar en el campo de texto la lista de Coordenadas que nos queda resultante para hacer "visible" el cambio. La forma de hacerlo es:

```
WinFormsServicePort.Post(new FormInvoke(delegate()
            {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
            }));
```

Cabe añadir, que los argumentos de entrada de estos handlers excepto en el add, nos dan imformación mediante index de la Coordenada resaltada con el ratón.

En el caso de add, la información que pasamos en el argumento de entrada es lo escrito en los campos de textos disponibles para meter las coordenadas x e y.

Utilizaremos las instrucciones remove at para borrar una coordenada e insert para introducirla.

```
        /// <summary>
        /// Add mission to missionlist
        /// </summary>
        /// <param name="a"></param>
        void AddCoordenadasHandler(AddCoordenadas a)
        {
            _CoordenadasList.Add(a.AddedCoordenadas);
            WinFormsServicePort.Post(new FormInvoke(delegate()
            {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
            }));
        }

        /// <summary>
        /// Remove a mission
        /// </summary>
        /// <param name="r"></param>
        void RemoveCoordenadasHandler(RemoveCoordenadas r)
        {
            _CoordenadasList.RemoveAt(r.Index);
            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
```

```
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
            }));
        }


         /// <summary>
        /// increate the position of a mission in list
        /// </summary>
        /// <param name="r"></param>
        void UpCoordenadasHandler(UpCoordenadas r)
        {
            if (r.Index - 1 >= 0)
            {
                Coordenadas _m = new Coordenadas();
                _m = _CoordenadasList[r.Index];
                _CoordenadasList.RemoveAt(r.Index);
                _CoordenadasList.Insert(r.Index - 1, _m);
                WinFormsServicePort.Post(new
FormInvoke(delegate()
                {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                }));
            }
        }

        /// <summary>
        /// Decrease position of a mission in a list
        /// </summary>
        /// <param name="r"></param>
        void DownCoordenadasHandler(DownCoordenadas r)
        {
            if (r.Index + 1 <= _CoordenadasList.Count - 1)
            {
                Coordenadas _m = new Coordenadas();
                _m = _CoordenadasList[r.Index];
                _CoordenadasList.RemoveAt(r.Index);
                _CoordenadasList.Insert(r.Index + 1, _m);
                WinFormsServicePort.Post(new
FormInvoke(delegate()
                {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                }));
```

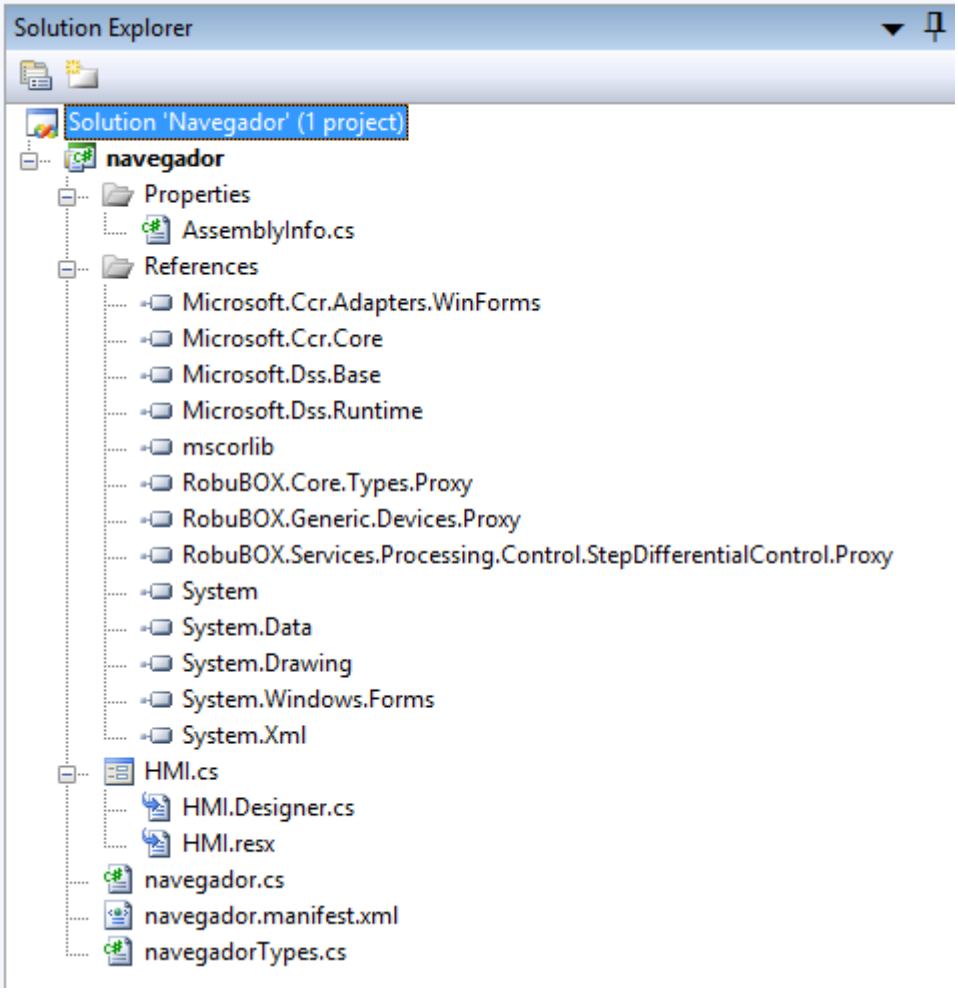
```
            }
        }

        #endregion
    }
}
```

3.2.3 VERSION 1.9.2

La versión 1.9.2 es una alternativa a la versión final posteriormente desarrollada que presenta un modo de control alternativo en el que el robot realiza también la misión de forma completa, pero de una forma muy diferente aunque también válida.

En la versión 1.9.2 tenemos como alternativa el uso del control diferencial por pasos, en el que el movimiento se desarrolla mediante "misiones", esto es, en vez de proponer una velocidad lineal o angular al robot, se le indica que recorrido debe realizar, es decir, una distancia hacia delante o hacia atrás, o un giro hacia la izquierda o derecha. Para comprender mejor cómo funciona el servicio, veremos la estructura del mismo como hicimos con la versión final.

En el explorador de soluciones de Visual Studio, podemos ver las diferentes partes que componen el servicio navegador1.9.2. Estas son las siguientes (figura 42):



(figura 42)

Solution 'Navegador' contiene el proyecto navegador, que a su vez contiene los ficheros de propiedades y de referencias, así como el HMI (human machine interface) con el que nos comunicamos con el programa, y los archivos navegador.cs, navegadorTypes.cs y navegador.manifest.xml.

3.2.3.1 PROPERTIES

Igual que en la versión final, podemos encontrar en properties el archivo AssemblyInfo.cs, el cual se genera automáticamente con la herramienta anteriormente mencionada "dssnewservice"

Este archivo contiene el siguiente código:

```
using System.Reflection;
using dss = Microsoft.Dss.Core.Attributes;
using interop = System.Runtime.InteropServices;

[assembly:
dss.ServiceDeclaration(dss.DssServiceDeclaration.ServiceBehavi
or)]
[assembly: interop.ComVisible(false)]
[assembly: AssemblyTitle("navegador")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyProduct("navegador")]
[assembly: AssemblyCopyright("")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("1.0.0.0")]
```

3.2.3.2 REFERENCES

Las referencias que se han incluido en el programa navegador1.9.2 son:

- Referencias por defecto:
  - Microsoft.Dss.Base
  - Microsoft.Dss.Runtime
  - Mscorlib
  - System
  - System.Data
  - System.Xml
- Referencias de Windows Forms:

- Microsoft.Ccr.Adapters.WinForms
- Microsoft.Ccr.Core
- System.Drawing
- System.Windows.Forms

- Referencias de RobuBOX
  - RobuBOX.core.Types.Proxy
  - RobuBOX.Generic.Devices.Proxy
  - RobuBOX.Services.Processing.Control.StepDifferentialControl.Proxy

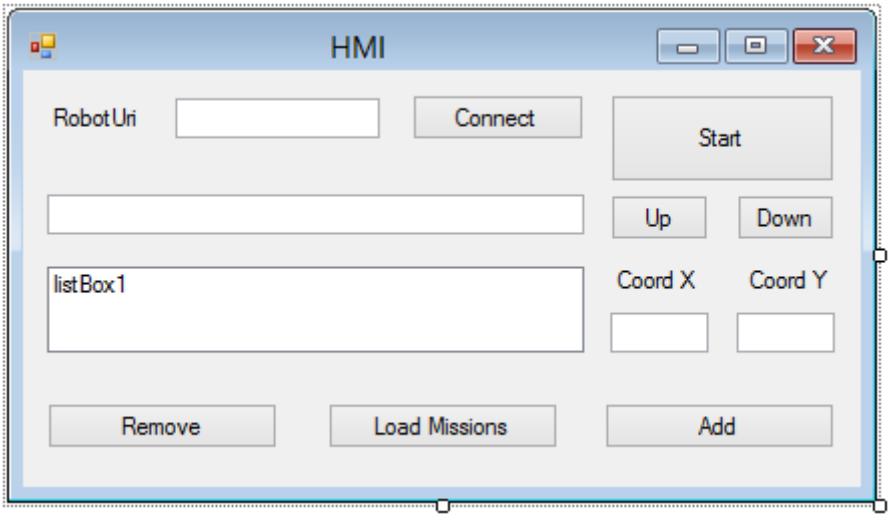
Como novedad encontramos la referencia RobuBOX.Services.Processing.Control.StepDifferentialControl.Proxy, la del control diferencial por pasos:

Con "RobuBOX.Services.Processing.Control.StepDifferentialControl.Proxy" obtenemos acceso al control diferencial de pasos del robot, en el que la potencia que se transmite al motor se calcula mediante "misiones" o "pasos", es decir, en vez de comunicar al robot que la potencia requerida en los motores es "x", le indicamos que queremos que avance un paso en dirección frontal una distancia "x", o que gire hacia la derecha un ángulo "α".

Es notable la ausencia a su vez del servicio de localización, el cual no está presente en esta versión. Para saber cuándo el robot alcanzaba la meta se utilizaban contadores de tipo double que acumulaban pasos para estimar donde se encontraba el robot en cada momento, una técnica mucho más rudimentaria que la utilizada en el navegador actual.

3.2.3.3 HMI

El aspecto de la ventana del HMI es el siguiente (figura 43):



(figura 43)

Este diseño se corresponde directamente con el siguiente código:

```
namespace navegador
{
    partial class HMI
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components =
null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources
should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code
```

```
        /// <summary>
        /// Required method for Designer support - do not
modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.textBox2 = new
System.Windows.Forms.TextBox();
            this.textBox3 = new
System.Windows.Forms.TextBox();
            this.label1 = new System.Windows.Forms.Label();
            this.listBox1 = new
System.Windows.Forms.ListBox();
            this.button3 = new System.Windows.Forms.Button();
            this.button4 = new System.Windows.Forms.Button();
            this.button5 = new System.Windows.Forms.Button();
            this.button6 = new System.Windows.Forms.Button();
            this.label3 = new System.Windows.Forms.Label();
            this.button7 = new System.Windows.Forms.Button();
            this.button8 = new System.Windows.Forms.Button();
            this.textBox1 = new
System.Windows.Forms.TextBox();
            this.label4 = new System.Windows.Forms.Label();
            this.textBox4 = new
System.Windows.Forms.TextBox();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Location = new
System.Drawing.Point(293, 12);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(112,
44);
            this.button1.TabIndex = 0;
            this.button1.Text = "Start";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new
System.EventHandler(this.button1_Click);
            //
            // textBox2
            //
```

```
            this.textBox2.Location = new
System.Drawing.Point(12, 62);
            this.textBox2.Name = "textBox2";
            this.textBox2.Size = new System.Drawing.Size(268,
20);
            this.textBox2.TabIndex = 3;
            //
            // textBox3
            //
            this.textBox3.Location = new
System.Drawing.Point(293, 121);
            this.textBox3.Name = "textBox3";
            this.textBox3.Size = new System.Drawing.Size(49,
20);
            this.textBox3.TabIndex = 6;
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new
System.Drawing.Point(294, 98);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(45,
13);
            this.label1.TabIndex = 7;
            this.label1.Text = "Coord X";
            //
            // listBox1
            //
            this.listBox1.FormattingEnabled = true;
            this.listBox1.Location = new
System.Drawing.Point(12, 98);
            this.listBox1.Name = "listBox1";
            this.listBox1.Size = new System.Drawing.Size(268,
43);
            this.listBox1.TabIndex = 8;
            //
            // button3
            //
            this.button3.Location = new
System.Drawing.Point(290, 166);
            this.button3.Name = "button3";
            this.button3.Size = new System.Drawing.Size(115,
23);
            this.button3.TabIndex = 9;
```

```
            this.button3.Text = "Add";
            this.button3.UseVisualStyleBackColor = true;
            this.button3.Click += new
System.EventHandler(this.button3_Click);
            //
            // button4
            //
            this.button4.Location = new
System.Drawing.Point(12, 166);
            this.button4.Name = "button4";
            this.button4.Size = new System.Drawing.Size(115,
23);
            this.button4.TabIndex = 10;
            this.button4.Text = "Remove";
            this.button4.UseVisualStyleBackColor = true;
            this.button4.Click += new
System.EventHandler(this.button4_Click);
            //
            // button5
            //
            this.button5.Location = new
System.Drawing.Point(293, 62);
            this.button5.Name = "button5";
            this.button5.Size = new System.Drawing.Size(49,
23);
            this.button5.TabIndex = 11;
            this.button5.Text = "Up";
            this.button5.UseVisualStyleBackColor = true;
            this.button5.Click += new
System.EventHandler(this.button5_Click);
            //
            // button6
            //
            this.button6.Location = new
System.Drawing.Point(356, 62);
            this.button6.Name = "button6";
            this.button6.Size = new System.Drawing.Size(49,
23);
            this.button6.TabIndex = 12;
            this.button6.Text = "Down";
            this.button6.UseVisualStyleBackColor = true;
            this.button6.Click += new
System.EventHandler(this.button6_Click);
            //
            // label3
```

```
            //
            this.label3.AutoSize = true;
            this.label3.Location = new
System.Drawing.Point(360, 98);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(45,
13);
            this.label3.TabIndex = 14;
            this.label3.Text = "Coord Y";
            //
            // button7
            //
            this.button7.Location = new
System.Drawing.Point(152, 166);
            this.button7.Name = "button7";
            this.button7.Size = new System.Drawing.Size(115,
23);
            this.button7.TabIndex = 15;
            this.button7.Text = "Load Missions";
            this.button7.UseVisualStyleBackColor = true;
            this.button7.Click += new
System.EventHandler(this.button7_Click);
            //
            // button8
            //
            this.button8.Location = new
System.Drawing.Point(194, 12);
            this.button8.Name = "button8";
            this.button8.Size = new System.Drawing.Size(86,
23);
            this.button8.TabIndex = 16;
            this.button8.Text = "Connect";
            this.button8.UseVisualStyleBackColor = true;
            this.button8.Click += new
System.EventHandler(this.button8_Click);
            //
            // textBox1
            //
            this.textBox1.Location = new
System.Drawing.Point(76, 14);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(102,
20);
            this.textBox1.TabIndex = 17;
            //
```

```
            // label4
            //
            this.label4.AutoSize = true;
            this.label4.Location = new
System.Drawing.Point(12, 17);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(49,
13);
            this.label4.TabIndex = 18;
            this.label4.Text = "RobotUri";
            //
            // textBox4
            //
            this.textBox4.Location = new
System.Drawing.Point(356, 121);
            this.textBox4.Name = "textBox4";
            this.textBox4.Size = new System.Drawing.Size(49,
20);
            this.textBox4.TabIndex = 19;
            //
            // HMI
            //
            this.AutoScaleDimensions = new
System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(418,
208);
            this.Controls.Add(this.textBox4);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.button8);
            this.Controls.Add(this.button7);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.button6);
            this.Controls.Add(this.button5);
            this.Controls.Add(this.button4);
            this.Controls.Add(this.button3);
            this.Controls.Add(this.listBox1);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.textBox3);
            this.Controls.Add(this.textBox2);
            this.Controls.Add(this.button1);
            this.Name = "HMI";
            this.Text = "HMI";
```

```
            this.FormClosed += new
System.Windows.Forms.FormClosedEventHandler(this.HMI_FormClose
d);
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.TextBox textBox2;
        private System.Windows.Forms.TextBox textBox3;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.ListBox listBox1;
        private System.Windows.Forms.Button button3;
        private System.Windows.Forms.Button button4;
        private System.Windows.Forms.Button button5;
        private System.Windows.Forms.Button button6;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Button button7;
        private System.Windows.Forms.Button button8;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.TextBox textBox4;
    }
}
```

Vemos que es muy parecida a la de la versión final, ya que de hecho, este HMI representa en si una versión anterior desde la que se ha evolucionado al HMI presentado antes.

Como principal diferencia vemos que este HMI precisa de un campo de texto para introducir el Robot Uri y un botón connect:

En "RobotUri" es donde debemos escribir el nombre del robot para que se conecte a los diversos servicios a los que tiene que acceder el programa (laser service y step service). El nombre que debemos añadir para operar con el robot con el que se ha desarrollado este proyecto es "RobuLAB10".

Una vez escrito esto, presionamos el botón Connect y seguidamente nos aparecerán ventanas emergentes que nos indican a que servicios nos acabamos de conectar: "connected to StepDifferentilDrive service" y "connected to Laser service". Sin embargo, si ocurre algún error durante la conexión también nos lo indicaran con la misma ventana emergente pero indicándonos que ha habido un error conectándose al servicio: "error connecting to Laser service".

Como vemos, esto no representa una pega, ya que incluso podríamos conectar cualquier otro tipo de robot escribiendo su numbre en el campo de texto para este mismo servicio, pero al ser un servicio íntegramente desarrollado para este proyecto en el que se utiliza robuLAB10, esta conexión en la versión final se realiza de forma automática sin tener que rellenar campos de texto ni presionar ningún botón, y se realiza al arrancar el servicio.

Una vez hemos conectado al robot, se activa el botón start, que es el encargado dar comienzo a las misiones que dan como resultado que el robot llegue al objetivo.

Las misiones se configuran en función a las coordenadas que tiene que alcanzar el robot, las cuales se introducen en los campos de texto debajo de los labels x e y respectivamente.

Las coordenadas por defecto son 0,0 es decir, la ubicación actual del robot. Si queremos añadir unas nuevas, escribimos en los campos de texto citados las coordenadas y presionamos el botón Add.

Podemos añadir más de unas coordenadas si queremos guardarlas, pero las que van a indicar las coordenadas a alcanzar son las que estén en lo alto de la lista que hagamos.

Si queremos mover una serie de coordenadas por la lista utilizamos los botones Up y Down. También podemos quitar una serie de coordenadas de la lista simplemente resaltándolas y presionando el botón remove.

Una vez hallamos agregado las coordenadas objetivo que queremos que el robot alcance, presionamos en Load Missions. Este botón cargará las distancias a la meta, reiniciará unos contadores que el programa navegador utiliza para alcanzar la misma y hace una serie de cálculos para dar comienzo al movimiento. Cuando este botón se presiona y los cálculos ya están hechos, una ventana emergente aparece con el mensaje "coordenadas lanzadas".

Por último solo tenemos que presionar el botón start para que el movimiento de comienzo y el robot se dirija y alcance el objetivo propuesto.

Por último aquí tenemos el código de funcionamiento del HMI:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using step =
Robosoft.RobuBOX.Services.Processing.Control.StepDifferentialC
ontrol.Proxy;

namespace navegador
{
    public partial class HMI : Form
    {
        //Member
        HMIPort _hmiPort;

        /// <summary>
        /// Constructor
        /// </summary>
```

```
        /// <param name="port"></param>
        public HMI(HMIPort port)
        {
            this._hmiPort = port;

            InitializeComponent();
            button1.Enabled = false;
        }

        /// <summary>
        /// Send new mission when the button1 is clicked
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button1_Click(object sender, EventArgs e)
        {
            this._hmiPort.Post(new StartMission());
        }

        /// <summary>
        /// Display a text box
        /// </summary>
        /// <param name="s">String to display in
textBox</param>
        public void LogInfo(string s)
        {
            MessageBox.Show(s);
        }

        /// <summary>
        /// Display on a textBox the mission list
        /// </summary>
        /// <param name="MissionList"></param>
        public void DisplayCoordenadasList(List<Coordenadas>
CoordenadasList)
        {
            listBox1.Items.Clear();
            foreach (Coordenadas coord in CoordenadasList)
            {
                listBox1.Items.Add("Coordenadas: X = " +
coord.x.ToString() + ", Y = " + coord.y.ToString());
            }
        }

        /// <summary>
```

```
        /// Clear textBox displaying current mission
        /// </summary>
        public void clearTextBox2()
        {
            this.textBox2.Text = "";
        }

        /// <summary>
        /// Display on textbox the current mission
        /// </summary>
        /// <param name="M"></param>
        public void DisplayCurrentMission(Mission M)
        {
            textBox2.Text = "";
            switch (M.direction)
            {
                case step.DirectionEnumeration.Front:
                    textBox2.Text = "Front " +
M.distance.ToString();
                    break;
                case step.DirectionEnumeration.Right:
                    textBox2.Text = "Right " +
M.distance.ToString();
                    break;
                case step.DirectionEnumeration.Left:
                    textBox2.Text = "Left " +
M.angle.ToString();
                    break;
                case step.DirectionEnumeration.Rear:
                    textBox2.Text = "Rear " +
M.angle.ToString();
                    break;
                default:
                    textBox2.Text = "No current mission";
                    break;
            }
        }

        /// <summary>
        /// Add new Coordinate to list
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button3_Click(object sender, EventArgs e)
        {
```

```
            try
            {
                Coordenadas _newcoordenada = new
Coordenadas();

                        _newcoordenada.x =
Convert.ToDouble(textBox3.Text);
                        _newcoordenada.y =
Convert.ToDouble(textBox4.Text);

                AddCoordenadas _addcoordenadas = new
AddCoordenadas();
                _addcoordenadas.AddedCoordenadas =
_newcoordenada;
                this._hmiPort.Post(_addcoordenadas);
            }
            catch
            {
                MessageBox.Show("Bad value entered");
            }
        }

        /// <summary>
        /// Remove Mission from the list
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button4_Click(object sender, EventArgs e)
        {
            int _index = -1;
            _index = listBox1.SelectedIndex;
            if (_index != -1)
            {
                RemoveCoordenadas _removeCoordenadas = new
RemoveCoordenadas();
                _removeCoordenadas.Index = _index;
                this._hmiPort.Post(_removeCoordenadas);
            }
        }

        /// <summary>
        /// Up the mission in the list
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
```

```
        private void button5_Click(object sender, EventArgs e)
        {
            int _index = -1;
            _index = listBox1.SelectedIndex;
            if (_index != -1)
            {
                UpCoordenadas _upCoordenadas = new
UpCoordenadas();
                _upCoordenadas.Index = _index;
                this._hmiPort.Post(_upCoordenadas);
            }
        }

        /// <summary>
        /// Decrease the position of the mission selected in
the mission list
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button6_Click(object sender, EventArgs e)
        {
            int _index = -1;
            _index = listBox1.SelectedIndex;
            if (_index != -1)
            {
                DownCoordenadas _downCoordenadas = new
DownCoordenadas();
                _downCoordenadas.Index = _index;
                this._hmiPort.Post(_downCoordenadas);
            }
        }
        /// <summary>
        /// Click on this button to connect to robot
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button8_Click(object sender, EventArgs e)
        {
            Connect _co = new Connect();
            _co.Uri = textBox1.Text;
            _hmiPort.Post(_co);
        }

        /// <summary>
        /// Click on this button to save missions in the state
```

```
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button7_Click(object sender, EventArgs e)
        {
            _hmiPort.Post(new SaveCoordenadas());
        }

        public void ActivateStartButton(bool b)
        {
            button1.Enabled = true;
        }

        /// <summary>
        /// Form closed, drop the service
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void HMI_FormClosed(object sender,
FormClosedEventArgs e)
        {
            _hmiPort.Post(new DropFromGui());
        }

    }
}
```

Como vemos es básicamente el mismo que en la versión final, salvando las ausencias de todo lo relacionado con la localización, y toda la parte añadida de la conexión del robot y la definición de la misión actual.

Esto se corresponde con esta parte del código:

Con esta funcionalidad mostramos en el campo de texto la misión actual que el robot está llevando a cabo. Estas son “Front, Right, Left, Rear”.

```
  /// <summary>
        /// Display on textbox the current mission
        /// </summary>
        /// <param name="M"></param>
        public void DisplayCurrentMission(Mission M)
```

```
        {
            textBox2.Text = "";
```

despues de haber despejado el campo de texto, mostramos cual es la misión que lleva a cabo mediante un switch, cuyas opciones serán que tipo de direccion tiene el step actual mediante la instrucción step.DirectionEnumeration.(Direccion).

```
            switch (M.direction)
            {
                case step.DirectionEnumeration.Front:
                    textBox2.Text = "Front " +
M.distance.ToString();
                    break;
                case step.DirectionEnumeration.Right:
                    textBox2.Text = "Right " +
M.distance.ToString();
                    break;
                case step.DirectionEnumeration.Left:
                    textBox2.Text = "Left " +
M.angle.ToString();
                    break;
                case step.DirectionEnumeration.Rear:
                    textBox2.Text = "Rear " +
M.angle.ToString();
                    break;
                default:
                    textBox2.Text = "No current mission";
                    break;
            }
```

Ahora definimos el botón de conexión como sigue.

```
  /// <summary>
        /// Click on this button to connect to robot
        /// </summary>
        /// <param name="sender"></param>
```

```
        /// <param name="e"></param>
        private void button8_Click(object sender, EventArgs e)
        {
```

Simplemente tenemos que postear en el puerto del HMI el string _co que coge la cadena de texto escrita en el textBox1.text, que no es otra cosa que el campo de texto adjuntado junto al botón de conexión.

```
Connect _co = new Connect();
            _co.Uri = textBox1.Text;
            _hmiPort.Post(_co);
        }
```

3.2.3.4 Navegador1.9.2Types.cs

Por supuesto, aun cuando el programa es tan similar, la definición del estado es muy diferente en esta versión. Esto se debe a que las variables de estado son completamente diferentes, ya que son las que definen el comportamiento del algoritmo de navegacion.

Por otro lado, también tenemos que definir variables del HMI y de los handlers, así que tendremos también partes muy similares.

El codigo completo para poder observar su estructura es el siguiente:

```
using Microsoft.Ccr.Core;
using Microsoft.Dss.ServiceModel.Dssp;
using System;
using System.Collections.Generic;
using Microsoft.Dss.Core.Attributes;
using W3C.Soap;
using navegador = navegador;
using types = Robosoft.RobuBOX.Core.Types.Proxy;

using step =
Robosoft.RobuBOX.Services.Processing.Control.StepDifferentialC
ontrol.Proxy;
```

```
using laser =
Robosoft.RobuBOX.Generic.Devices.LaserSensor.Proxy;


namespace navegador
{
    #region Contract

    /// <summary>
    /// MissionsManager Contract
    /// </summary>
    public sealed class Contract
    {
        /// The Unique Contract Identifier for the
MissionsManager service
        public const String Identifier =
"http://schemas.tempuri.org/2014/04/navegador.html";
    }

    #endregion

    #region State Types

    /// <summary>
    /// Service state class
    /// </summary>
    [DataContract()]
    public class navegadorState
    {
        /// <summary>
        /// List of coordenadas to send
        /// </summary>
        [DataMember]
        public List<Coordenadas> CoordenadasList = new
List<Coordenadas>();

        /// <summary>
        /// Maximum linear speed allowed
        /// </summary>
        [DataMember]
        public double LinearSpeedMax;

        /// <summary>
        /// Maximum Angular Speed allowed
        /// </summary>
```

```
        [DataMember]
        public double AngularSpeedMax;

        /// <summary>
        /// Uri of the service stepDifferentialControl to
connect
        /// </summary>
        [DataMember]
        public string RobotUri;

        public bool alarma;

        public double alpha;

        public double meta;

        public double contadorX;

        public double contadorY;

        public bool giro;

        public bool izquierda;

        public bool comprobacion;

        public bool girado;

        public bool inversion;

        public bool inversion2;

        public bool derecha;

    }

    #endregion

    #region Opeations and Messages

    /// <summary>
    /// MissionManager main port class
    /// </summary>
```

```
    public class navegadorOperations :
PortSet<DsspDefaultLookup, DsspDefaultDrop, Get, Replace,
Update, Subscribe>
    {
    }
    /// <summary>
    /// Port to communicate between the service and its HMI
    /// </summary>
    public class HMIPort : PortSet<DsspDefaultLookup,
DsspDefaultDrop, StartMission, AddCoordenadas,
RemoveCoordenadas,
        UpCoordenadas, DownCoordenadas, Connect,
SaveCoordenadas, DropFromGui>
    {
    }

    /// <summary>
    /// Provides Read-Access to the  state.
    /// </summary>
    public class Get : Get<GetRequestType,
PortSet<navegadorState, Fault>>
    {
    }

    /// <summary>
    /// Used to replace state
    /// </summary>
    public class Replace : Replace<navegadorState,
PortSet<DefaultReplaceResponseType, Fault>>
    {
    }

    /// <summary>
    /// Used to update state
    /// </summary>
    public class Update : Update<navegadorState,
PortSet<DefaultReplaceResponseType, Fault>>
    {
    }

    public class Subscribe : Subscribe<SubscribeRequestType,
PortSet<SubscribeResponseType, Fault>>
    {
    }
```

```
    /// <summary>
    /// Internal message from HMI to send mission when
clicking on start button
    /// </summary>
    public class StartMission
    {
    }

    /// <summary>
    /// Coordinates definition
    /// </summary>
    [DataContract]
    public class Coordenadas
    {
        /// <summary>
        /// Coordenadas
        /// </summary>
        [DataMember]
        public double x;

        [DataMember]
        public double y;
    }

    /// <summary>
    /// Mission definition
    /// </summary>
    [DataContract]
    public class Mission
    {
        /// <summary>
        /// Direction of the motion
        /// </summary>
        [DataMember]
        public step.DirectionEnumeration direction;

        /// <summary>
        /// Distance to travel
        /// </summary>
        [DataMember]
        public double distance;
        /// <summary>
        /// Angle to rotate
        /// </summary>
```

```
        [DataMember]
        public double angle;
    }

    public class AddCoordenadas
    {
        public Coordenadas AddedCoordenadas;
    }

    public class RemoveCoordenadas
    {
        public int Index;
    }

    public class UpCoordenadas
```

Se destacan las nuevas variables definidas, la definición de la clase Mission, la necesidad de añadir el using step y las inclusiones en el HMI:

Las variables del estado del navegador que se han definido en esta versión son las contenidas en la región State types

```
    #region State Types

    /// <summary>
    /// Service state class
    /// </summary>
    [DataContract()]
    public class navegadorState
    {
        /// <summary>
        /// List of coordenadas to send
        /// </summary>
        [DataMember]
        public List<Coordenadas> CoordenadasList = new
List<Coordenadas>();

        /// <summary>
        /// Maximum linear speed allowed
        /// </summary>
        [DataMember]
        public double LinearSpeedMax;
```

```
        /// <summary>
        /// Maximum Angular Speed allowed
        /// </summary>
        [DataMember]
        public double AngularSpeedMax;

        /// <summary>
        /// Uri of the service stepDifferentialControl to
connect
        /// </summary>
        [DataMember]
        public string RobotUri;

        public bool alarma;

        public double alpha;

        public double meta;

        public double contadorX;

        public double contadorY;

        public bool giro;

        public bool izquierda;

        public bool comprobacion;

        public bool girado;

        public bool inversion;

        public bool inversion2;

        public bool derecha;

    }

    #endregion
```

Vemos que en este caso tenemos dos variables que limitan las velocidades angular y lineal (algo innecesario como vimos en la versión 3.0 ya que en esta se definía directamente la velocidad requerida).

Además, tenemos la lista de coordenadas, necesaria en este servicio también, y las variables del estado novedosas.

El string robot Uri es en el que estará contenido el identificador del robot mediante el nombre de su modelo, que como ya hemos visto, en este caso será siempre robuLAB10 (razón por la cual esta parte se omitió y se optó por una conexión automática en la versión final).

Alarma es una variable booleana que se vuelve true cuando el láser detecta un objeto en las regiones definidas en el handler del Update del mismo sensor. Esta variable es muy importante y actúa de una forma similar a los estados que utilizábamos en navegador3.0.

Alpha es un valor double que calculamos al lanzar las coordenadas y que nos informa del ángulo que tiene que girar inicialmente el robot para encarar al objetivo cuando comienza la persecución de las coordenadas objetivo. Esto es realmente interesante para definir el algoritmo que posteriormente explicaremos.

Meta es el valor de la hipotenusa que une la posición inicial del robot en el sistema cartesiano definido en el plano que contiene al robot y el objetivo, con la posición de este mismo.

Contador x y contador y son dos variables de tipo double que van contando la distancia de los pasos que se van dando a lo largo de las misiones que lleva a cabo el robot, y que marcarán la condición de llegada cuando el contador y sea igual a la meta y el contador x sea igual a 0.

Comprobación es una variable de control booleana del algoritmo de navegación que informa al robot básicamente de cuando tiene que

comprobar si ha superado un obstáculo que detectó anteriormente, y que en caso afirmativo permitirá al mismo continuar la marcha.

Izquierda derecha y girado son variables booleanas también que informan al algoritmo de la orientación del robot con respecto a la posición inicial girada el ángulo Alpha.

Por otro lado, giro es la variable que indica al robot si ya ha realizado o no este giro inicial Alpha.

Inversión e inversión2 son dos variables booleanas que se activan en diferentes momentos del algoritmo y que definen el tipo de decisiones que el robot toma para llegar hasta el objetivo.

La clase Mission es una clase creada para facilitar la definición del paso que el robot tiene que dar.

Como vemos, tiene una dirección que viene definida como tipo step.DirectionEnumeration, es decir, la dirección del step, una distancia, que es la distancia del step, y un ángulo para el caso de que la misión sea un giro (right y left).

```
    /// <summary>
    /// Mission definition
    /// </summary>
    [DataContract]
    public class Mission
    {
        /// <summary>
        /// Direction of the motion
        /// </summary>
        [DataMember]
        public step.DirectionEnumeration direction;

        /// <summary>
        /// Distance to travel
        /// </summary>
        [DataMember]
```

```
        public double distance;
        /// <summary>
        /// Angle to rotate
        /// </summary>
        [DataMember]
        public double angle;
    }
```

Por último, tenemos que definir en el puerto del HMI la funcionalidad que definimos para conectar el servicio, es decir, el connect.

```
    public class HMIPort : PortSet<DsspDefaultLookup,
DsspDefaultDrop, StartMission, AddCoordenadas,
RemoveCoordenadas,
        UpCoordenadas, DownCoordenadas, Connect,
SaveCoordenadas, DropFromGui>
    {
    }
```

3.2.3.5 Navegador1.9.2.cs

Por último se presentará a continuación el código escrito para la versión navegador1.9.2.cs:

```
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Security.Permissions;
using navegador = navegador;
using xml = System.Xml;

using Microsoft.Ccr.Adapters.WinForms;
using System.Windows.Forms;
```

```
using types = Robosoft.RobuBOX.Core.Types.Proxy;

using ds = Microsoft.Dss.Services.Directory;
using submgr = Microsoft.Dss.Services.SubscriptionManager;

using step =
Robosoft.RobuBOX.Services.Processing.Control.StepDifferentialC
ontrol.Proxy;
using laser =
Robosoft.RobuBOX.Generic.Devices.LaserSensor.Proxy;



namespace navegador
{
    [DisplayName("navegador")]
    [Description("The navegador Service")]
    [Contract(Contract.Identifier)]
    //
    public class navegadorService : DsspServiceBase
    {

        #region State and Partners
        //State definition
        //If the parameter file exist in "Microsoft Robotics
Studio (October 2006)\store"
        //load the state by the parameter file
        [InitialStatePartner(Optional = true, ServiceUri =
"store/navegador.xml")]
        private navegadorState _state;

        private step.StepDifferentialControlOperations
_stepPort;
        private step.StepDifferentialControlOperations
_stepNotifyPort = new
step.StepDifferentialControlOperations();

        private laser.LaserSensorOperations _laserPort;
        private laser.LaserSensorOperations _laserNotifyPort =
new laser.LaserSensorOperations();
        #endregion

        #region Service port
        //Port definition
```

```
        [ServicePort("/navegador", AllowMultipleInstances =
false)]
        private navegadorOperations _mainPort = new
navegadorOperations();

        #endregion

        #region Members

        //internal port for HMI
        private HMIPort _hmiPort = new HMIPort();

        //Variables
        HMI _mainHMI;
        Mission Current = new Mission();
        public List<Coordenadas> _CoordenadasList = new
List<Coordenadas>();
        bool _robotStop = true;

        string _stepSubscriptionManager;
        string _stepSubscriber;

        string _laserSubMgr = string.Empty;
        string _laserSubscriber = string.Empty;


        #endregion

        #region Constructor and Start methode
        /// <summary>
        /// Default Service Constructor
        /// </summary>
        public navegadorService(DsspServiceCreationPort
creationPort)
            :
                base(creationPort)
        {

        }
        /// <summary>
        /// Service Start
        /// </summary>
        protected override void Start()
        {
            //Fill the state if the XML file doesn't exist
```

```
            if (_state == null)
            {
                _state = new navegadorState();
                _state.RobotUri =
"http://localhost:50000/directory";
                _state.CoordenadasList = new
List<Coordenadas>();
                _state.LinearSpeedMax = 0.3;
                _state.AngularSpeedMax = 0.3;

                Coordenadas inicial = new Coordenadas();
                inicial.x = 0;
                inicial.y = 0;
                _state.CoordenadasList.Add(inicial);//add
coordenada to coordenadas list

                _state.alarma = false;

                SaveState(_state);
            }


            foreach (Coordenadas inic in
this._state.CoordenadasList)
            {
                _CoordenadasList.Add(inic);
            }

            base.Start();

            //Interleave (the link between a message and the
response)
            MainPortInterleave.CombineWith(new Interleave(
                new TeardownReceiverGroup(),
                new ExclusiveReceiverGroup(
                        Arbiter.Receive<laser.Update>(true,
_laserNotifyPort, UpdateLaserHandler),

Arbiter.ReceiveWithIterator<Connect>(true, _hmiPort,
ConnectHandler),

Arbiter.Receive<SaveCoordenadas>(true, _hmiPort, SaveHandler),
                        Arbiter.Receive<DropFromGui>(true,
_hmiPort, DropFromGuiHandler)
                    ),
```

```
                new ConcurrentReceiverGroup(
                        Arbiter.Receive<StartMission>(true,
_hmiPort, SendCoordenadasHandler),
                        Arbiter.Receive<AddCoordenadas>(true,
_hmiPort, AddCoordenadasHandler),

Arbiter.Receive<RemoveCoordenadas>(true, _hmiPort,
RemoveCoordenadasHandler),
                        Arbiter.Receive<UpCoordenadas>(true,
_hmiPort, UpCoordenadasHandler),
                        Arbiter.Receive<DownCoordenadas>(true,
_hmiPort, DownCoordenadasHandler)


                ));

            //launch Winform
            WinFormsServicePort.Post(new
RunForm(RunMainWindow));

            //Display the mission list in the HMI
            WinFormsServicePort.Post(new FormInvoke(delegate()
            {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
            }));
        }
        #endregion

        #region Get Handler

        /// <summary>
        /// Get Handler
        /// </summary>
        /// <param name="get"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Concurrent)]
        public virtual IEnumerator<ITask> GetHandler(Get get)
        {
            get.ResponsePort.Post(_state);
            yield break;
        }

        #endregion
```

```
        #region replace Handler

        /// <summary>
        /// Replace Handler
        /// </summary>
        /// <param name="replace"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Exclusive)]
        public virtual IEnumerator<ITask>
ReplaceHandler(Replace replace)
        {
            _state = replace.Body;

replace.ResponsePort.Post(DefaultReplaceResponseType.Instance)
;
            yield break;
        }

        #endregion

        #region drop Handler

        /// <summary>
        /// Drop Handler
        /// </summary>
        /// <param name="drop"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Teardown)]
        public IEnumerator<ITask> DropHandler(DsspDefaultDrop
drop)
        {
            bool error = false;

            WinFormsServicePort.FormInvoke(delegate()
            {
                _mainHMI.Dispose();
            });
            if (_laserSubMgr != string.Empty)
            {
                //unsubscribe to laser
                submgr.DeleteSubscription delete = new
submgr.DeleteSubscription();
                delete.Body = new
submgr.DeleteSubscriptionMessage(_laserSubscriber);
```

```
ServiceForwarder<submgr.SubscriptionManagerPort>(_laserSubMgr)
.Post(delete);
                yield return
Arbiter.Choice<DefaultDeleteResponseType, W3C.Soap.Fault>(
                    delete.ResponsePort,
                    delegate(DefaultDeleteResponseType
response)
                    {
                        _laserSubMgr = string.Empty;
                        _laserSubscriber = string.Empty;
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogError("Error deleting subscription
to laser");
                    });
            }
            if (!_robotStop)
            {
                step.Update stopStep = new step.Update();
                stopStep.Body.MotionDirection =
step.DirectionEnumeration.Stop;
                stopStep.Body.AngleRequest = 0;
                stopStep.Body.DistanceRequest = 0;
                stopStep.Body.LinearSpeedMax = 0;
                stopStep.Body.AngularSpeedMax = 0;

                _stepPort.Post(stopStep);
                yield return
Arbiter.Choice<DefaultUpdateResponseType, W3C.Soap.Fault>(
                    stopStep.ResponsePort,
                    delegate(DefaultUpdateResponseType
response)
                    {

                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        error = true;
                        LogError("Error stopping robot");
                    });

            }
            if (_stepSubscriber != null)
```

```
            {
                submgr.DeleteSubscriptionMessage _body = new
submgr.DeleteSubscriptionMessage();
                _body.Subscriber = _stepSubscriber;
                _body.Expiration = DateTime.Now;
                submgr.DeleteSubscription _delete = new
submgr.DeleteSubscription();
                _delete.Body = _body;

ServiceForwarder<submgr.SubscriptionManagerPort>(_stepSubscrip
tionManager).Post(_delete);
                yield return
Arbiter.Choice<DefaultDeleteResponseType, W3C.Soap.Fault>(
                    _delete.ResponsePort,
                    delegate(DefaultDeleteResponseType
response)
                    {

                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        error = true;
                        LogError("Error deleting subscription
to StepDifferentilControl");
                    });
            }
            if (!error)
            {
                DirectoryDelete();
                Shutdown();

drop.ResponsePort.Post(DefaultDropResponseType.Instance);
            }
            else
            {
                drop.ResponsePort.Post(new W3C.Soap.Fault());
            }
            yield break;
        }

        #endregion

        #region Update Laser Handler

        public void UpdateLaserHandler(laser.Update update)
```

```
        {
            _state.alarma = false;

            foreach (laser.Echo E in update.Body.Echos)
            {
                types.Point2D pointCartesian = new
types.Point2D();
                pointCartesian.X = E.Distance *
Math.Cos(E.Angle);
                pointCartesian.Y = E.Distance *
Math.Sin(E.Angle);

                if (pointCartesian.Y < 0.4 && pointCartesian.Y
> -0.4 && pointCartesian.X > 0 && pointCartesian.X <= 0.8)
                {
                    _state.alarma = true;
                }
            }

       }

        #endregion

        #region Drop, save and stop motion

        /// <summary>
        /// Drop message sent from gui
        /// </summary>
        /// <param name="d"></param>
        void DropFromGuiHandler(DropFromGui d)
        {
            _mainPort.Post(new DsspDefaultDrop());
        }

        /// <summary>
        /// The robot is stoped notify it to subscribers
        /// </summary>
        /// <param name="s"></param>
        void PaddleStopMotionHandler(step.StopMotion s)
        {
            _robotStop = true;
            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
                _mainHMI.clearTextBox2();
            }));
```

```
            SendCoordenadas();
        }

        /// <summary>
        /// Save current missions in state xml file
        /// </summary>
        /// <param name="s"></param>
        void SaveHandler(SaveCoordenadas s)
        {
            this._state.CoordenadasList.Clear();
            foreach (Coordenadas coord in _CoordenadasList)
            {
                this._state.CoordenadasList.Add(coord); ;
            }
            SaveState(_state);

            double distX = _CoordenadasList[0].x;
            double distY = _CoordenadasList[0].y;
            double beta = Math.Atan(distY / distX);
            double alpha_rad = Math.PI / 2 - beta;
            _state.alpha = alpha_rad * 180 / Math.PI;

            if (distX == 0)
            {
                _state.meta = distY;
                if (distY > 0)
                    _state.alpha = 0;
                else if (distY < 0)
                    _state.alpha = 180;
            }
            else if (distY == 0)
            {
                _state.meta = distX;
                if (distX > 0)
                    _state.alpha = 90;
                else if (distX < 0)
                    _state.alpha = -90;
            }
            else
            {
                double hx = distX / Math.Cos(beta);
                double hy = distY / Math.Sin(beta);
                _state.meta = (hx + hy) / 2;
            }
```

```
            //contadores
            _state.contadorX = 0;
            _state.contadorY = 0;

            //variables de control iniciales
            _state.giro = false;
            _state.izquierda = false;
            _state.comprobacion = false;
            _state.girado = false;
            _state.inversion = false;
            _state.inversion2 = false;

            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
                _mainHMI.LogInfo("Coordenadas lanzadas");
            }));

        }

        #endregion

        #region connect to stepDifferentialControl and Laser
services

        /// <summary>
        /// Connect to StepDifferentialControl service
        /// See this part of code to understand how to connect
        /// to a service.
        /// </summary>
        IEnumerator<ITask> ConnectHandler(Connect co)
        {
            //Change "paddle" of this line by the alias of the
service to connect
            ServiceInfoType info = new
ServiceInfoType(step.Contract.Identifier);

            //Give the URI of the robot or where the service
is running
            ds.DirectoryPort remotePort =
ServiceForwarder<ds.DirectoryPort>(new Uri("http://" + co.Uri
+ ":50000/directory"));
            ds.Query query = new ds.Query(new
ds.QueryRequestType(info));
            remotePort.Post(query);
```

```
            //listen for a response
            yield return Arbiter.Choice<ds.QueryResponseType,
W3C.Soap.Fault>(query.ResponsePort,
                //Occure when connection is success
                delegate(ds.QueryResponseType response)
                {
                    //Change "_paddlePort" by the port of the
service to connect
                    //change paddle.PaddleServiceOperations"
by the type of the service main port to connect
                    _stepPort =
ServiceForwarder<step.StepDifferentialControlOperations>(respo
nse.RecordList[0].Service);
                    LogInfo(LogGroups.Console, "Panel Service
connected to Step differential Drive");
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                    {
                        _mainHMI.LogInfo("connected to
StepDifferentilDrive service");
                    }));

                },
                //Occure when connection failed
                delegate(W3C.Soap.Fault error)
                {
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                    {
                        _mainHMI.LogInfo("Error connecting to
StepDifferentilDrive service");
                    }));

                    LogInfo(LogGroups.Console, "ERROR
connecting to StepDifferentilDrive service");
                });
            if (_stepPort != null)
            {
                yield return
Arbiter.Choice<SubscribeResponseType, W3C.Soap.Fault>(

_stepPort.Subscribe(_stepNotifyPort),
                            delegate(SubscribeResponseType
response)
                            {
```

```
                                    LogInfo("subscribed to
stepDifferentialDrive service...");
                                    _stepSubscriptionManager =
response.SubscriptionManager;
                                    _stepSubscriber =
response.Subscriber;

Activate(Arbiter.Receive<step.StopMotion>(true,
_stepNotifyPort,
                                        PaddleStopMotionHandler));
                                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                                    {

_mainHMI.ActivateStartButton(true);
                                    }));

                                },
                                delegate(W3C.Soap.Fault fault)
                                {
                                    LogInfo("Failed to subscribe
to stepDifferentialDrive...");
                                    _stepNotifyPort = null;
                                });
            }

            bool error2 = false;
            //Change "laser" by the alias of the service to
connect
            info = new
ServiceInfoType(laser.Contract.Identifier);
            //Declare a port to the node/machine where the
service to connect is running, giving URI
            ds.DirectoryPort remotePort2 =
ServiceForwarder<ds.DirectoryPort>(new Uri("http://" + co.Uri
+ ":50000/directory"));
            //Declare request
            ds.Query query2 = new ds.Query(new
ds.QueryRequestType(info));
            //post
            remotePort2.Post(query2);

            //listen for a response
            yield return Arbiter.Choice<ds.QueryResponseType,
W3C.Soap.Fault>(query2.ResponsePort,
```

```
                delegate(ds.QueryResponseType response)
                {
                    error2 = false;
                    //instanciate services port: receive the
two intance of laser
                    _laserPort =
ServiceForwarder<laser.LaserSensorOperations>(response.RecordL
ist[0].Service);
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                    {
                        _mainHMI.LogInfo("connected to Laser
service");
                    }));
                },
                delegate(W3C.Soap.Fault fault)
                {
                    error2 = true;
                    _laserPort = null;
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                    {
                        _mainHMI.LogInfo("Error connecting to
Laser service");
                    }));

                    LogInfo(LogGroups.Console, "ERROR
connecting to Laser service");
                });
            if (error2 == false)
            {
                //Subscribe with the odometry
                yield return
Arbiter.Choice<SubscribeResponseType, W3C.Soap.Fault>(
                    _laserPort.Subscribe(_laserNotifyPort),
                    delegate(SubscribeResponseType response)
                    {
                        //Make he link between the message
receive and the handler
                        LogInfo("subscribed to Laser
service...");
                        _laserSubMgr =
response.SubscriptionManager;
                        _laserSubscriber =
response.Subscriber;
```

```
Activate(Arbiter.Receive<laser.Update>(false,
_laserNotifyPort, UpdateLaserHandler));

                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogInfo("Failed to subscribe to
Laser...");

                    });
            }
            yield break;

        }

        #endregion

        #region RunWindow,

        //Function to instatiate the HMI window
        HMI RunMainWindow()
        {
            _mainHMI = new HMI(_hmiPort);
            return _mainHMI;
        }

        #endregion

        #region SendCoordenadas Logica

        #region variables iniciales

        #endregion

        void SendCoordenadas()
        {

            #region misiones

            // conjunto de misiones standard
            Mission Mfront = new Mission();
            Mfront.direction =
step.DirectionEnumeration.Front;
```

```
            Mfront.distance = 0.5;
            Mfront.angle = 0;

            Mission Mright = new Mission();
            Mright.direction =
step.DirectionEnumeration.Right;
            Mright.distance = 0;
            Mright.angle = 90;

            Mission Mleft = new Mission();
            Mleft.direction = step.DirectionEnumeration.Left;
            Mleft.distance = 0;
            Mleft.angle = 90;

            Mission Mrear = new Mission();
            Mrear.direction = step.DirectionEnumeration.Rear;
            Mrear.distance = _state.contadorY - _state.meta;
            Mrear.angle = 0;

            #endregion

            #region inicio del movimiento
            // algoritmo de navegación
            if (_robotStop)
            {

                #region if (!giro)

                if (_state.giro == false)
                {
                    Mission Mgiro = new Mission();
                    Mgiro.direction =
step.DirectionEnumeration.Right;
                    Mgiro.distance = 0;
                    Mgiro.angle = _state.alpha;

                    //Send the mission at the top of the list
                    Current = Mgiro;

                    //Display missions list on HMI
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                    {

_mainHMI.DisplayCurrentMission(Current);
```

```
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                    }));

                    //Mark the robot as moving
                    _robotStop = false;

                    //Marcar giro como realizado

                    _state.giro = true;

                    //Send order
                    step.Update updateStep = new
step.Update();
                    updateStep.Body.MotionDirection =
Current.direction;
                    updateStep.Body.AngleRequest =
Current.angle;
                    updateStep.Body.DistanceRequest =
Current.distance;
                    updateStep.Body.LinearSpeedMax =
this._state.LinearSpeedMax;
                    updateStep.Body.AngularSpeedMax =
this._state.AngularSpeedMax;
                    _stepPort.Post(updateStep);
                }

                #endregion

                else if (_state.contadorX != 0 ||
_state.contadorY != _state.meta)
                {

                    #region if (_state.contadorY >
_state.meta)

                    if (_state.contadorY > _state.meta)
                    {
                        //Send the mission at the top of the
list
                        Current = Mrear;

                        //Display missions list on HMI
                        WinFormsServicePort.Post(new
FormInvoke(delegate()
```

```
                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                        }));

                        //Mark the robot as moving
                        _robotStop = false;

                        //legada
                        _state.contadorY = _state.meta;

                        //Send order
                        step.Update updateStep = new
step.Update();
                        updateStep.Body.MotionDirection =
Current.direction;
                        updateStep.Body.AngleRequest =
Current.angle;
                        updateStep.Body.DistanceRequest =
Current.distance;
                        updateStep.Body.LinearSpeedMax =
this._state.LinearSpeedMax;
                        updateStep.Body.AngularSpeedMax =
this._state.AngularSpeedMax;
                        _stepPort.Post(updateStep);
                    }

                    #endregion


                    else
                    {

                        if (_state.inversion == false &&
_state.inversion2 == false)
                        {

                            #region if (_state.alarma == true
&& _state.izquierda == false)

                            if (_state.alarma == true &&
_state.izquierda == false)
                            {
```

```
                                    Current = Mleft;

                                    //Display missions list on HMI
                                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //robot mirando a la izquierda

                                    _state.izquierda = true;

                                    //Marcar obstáculo como
evitado
                                    _state.alarma = false;


                                    //Send order
                                    step.Update updateStep = new
step.Update();

updateStep.Body.MotionDirection = Current.direction;
                                    updateStep.Body.AngleRequest =
Current.angle;

updateStep.Body.DistanceRequest = Current.distance;
                                    updateStep.Body.LinearSpeedMax
= this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

                                    _stepPort.Post(updateStep);
                                }

                                #endregion

                                else
                                {
```

```
                                #region  if (izquierda &&
!comprobacion && !_state.alarma)

                                if (_state.izquierda == true
&& _state.comprobacion == false && _state.alarma == false)
                                {
                                    //Send the mission at the
top of the list
                                    Current = Mfront;

                                    //Display missions list on
HMI
WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {
_mainHMI.DisplayCurrentMission(Current);
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //Incrementar el contador
X en - recorrido

                                    _state.contadorX =
_state.contadorX - 0.5;
                                    _state.comprobacion =
true;


                                    //Send order
                                    step.Update updateStep =
new step.Update();
updateStep.Body.MotionDirection = Current.direction;
updateStep.Body.AngleRequest = Current.angle;
updateStep.Body.DistanceRequest = Current.distance;
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
```

```
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                #region  else if (izquierda &&
_state.alarma)

                                else if (_state.izquierda ==
true && _state.alarma)
                                {
                                    //Send the mission at the
top of the list
                                    Current = Mright;

                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //inversion "espejo" del
problema
                                    _state.inversion = true;
                                    _state.inversion2 = false;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;
```

```
updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                    }

                                    #endregion

                                    #region (izquierda &&
comprobacion && !_state.alarma)

                                    else if (_state.izquierda ==
true && _state.comprobacion == true && _state.alarma == false)
                                    {
                                        Current = Mright;

                                        //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as moving
                                        _robotStop = false;

                                        //marcar comprobacion como
hecha
                                        _state.comprobacion =
false;

                                        //marcar izquierda falsa
                                        _state.izquierda = false;

                                        //Send order
                                        step.Update updateStep =
new step.Update();
```

```
updateStep.Body.MotionDirection = Current.direction;
updateStep.Body.AngleRequest = Current.angle;
updateStep.Body.DistanceRequest = Current.distance;
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;
_stepPort.Post(updateStep);
                                    }
                                    #endregion
                                    else
                                    {
                                        #region if
(_state.contadorY < _state.meta)
                                        if (_state.contadorY <
_state.meta)
                                        {
                                            //Send the mission at
the top of the list
                                            Current = Mfront;
                                            //Display missions
list on HMI
WinFormsServicePort.Post(new FormInvoke(delegate()
                                            {
_mainHMI.DisplayCurrentMission(Current);
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                            }));
                                            //Mark the robot as
moving
                                            _robotStop = false;
```

```
                                        //Incrementar el
contador y en 1

                                        _state.contadorY =
_state.contadorY + 0.5;

                                        //Send order
                                        step.Update updateStep
= new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                    }

                                    #endregion

                                    #region else if
(_state.contadorY == _state.meta && _state.contadorX != 0)

                                    else if (_state.contadorY
== _state.meta && _state.contadorX != 0)
                                    {

                                        //Send the mission at
the top of the list
                                        Current = Mright;

                                        //Display missions
list on HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
```

```
                                        }));

                                        //Mark the robot as
moving
                                        _robotStop = false;

                                        //giro de pi/2
radianes negativos en las coordenadas locales del robot
                                        //nuevos objeivos
                                        _state.meta =
Math.Abs(_state.contadorX);
                                        _state.contadorX = 0;
                                        _state.contadorY = 0;

                                        //Send order
                                        step.Update updateStep
= new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                    }

                                    #endregion
                                }

                            }
                        }
                        else if (_state.inversion == true &&
_state.inversion2 == false)
                        {

                            #region if (_state.alarma == true
&& !izquierda)

                            if (_state.alarma == true &&
_state.izquierda == false)
```

```
                                {
                                    Current = Mright;

                                    //Display missions list on HMI
                                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //robot mirando a la izquierda

                                    _state.izquierda = true;

                                    //Marcar obstáculo como
evitado
                                    _state.alarma = false;


                                    //Send order
                                    step.Update updateStep = new
step.Update();

updateStep.Body.MotionDirection = Current.direction;
                                    updateStep.Body.AngleRequest =
Current.angle;

updateStep.Body.DistanceRequest = Current.distance;
                                    updateStep.Body.LinearSpeedMax
= this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

                                    _stepPort.Post(updateStep);
                                }

                                #endregion

                                else
```

```
                                {

                                    #region if (izquierda &&
!comprobacion && !_state.alarma)

                                    if (_state.izquierda == true
&& _state.comprobacion == false && _state.alarma == false)
                                    {
                                        //Send the mission at the
top of the list
                                        Current = Mfront;

                                        //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as moving
                                        _robotStop = false;

                                        //Incrementar el contador
X en - recorrido

                                        _state.contadorX =
_state.contadorX + 0.5;
                                        _state.comprobacion =
true;


                                        //Send order
                                        step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;
```

```
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;
_stepPort.Post(updateStep);
                                    }

                                    #endregion

                                    #region else if (izquierda &&
_state.alarma)

                                    else if (_state.izquierda ==
true && _state.alarma)
                                    {
                                        //Send the mission at the
top of the list
                                        Current = Mleft;

                                        //Display missions list on
HMI
WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {
_mainHMI.DisplayCurrentMission(Current);
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as moving
                                        _robotStop = false;

                                        //inversion "espejo" del
problema
                                        _state.inversion = true;
                                        _state.inversion2 = true;

                                        //Send order
                                        step.Update updateStep =
new step.Update();
updateStep.Body.MotionDirection = Current.direction;
```

```
updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                    }

                                    #endregion

                                    #region else if (izquierda &&
comprobacion && !_state.alarma)

                                    else if (_state.izquierda ==
true && _state.comprobacion == true && !_state.alarma)
                                    {
                                        Current = Mleft;

                                        //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as moving
                                        _robotStop = false;

                                        //marcar comprobacion como
hecha
                                        _state.comprobacion =
false;

                                        //marcar izquierda falsa
                                        _state.izquierda = false;

                                        //Send order
```

```
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                else
                                {

                                    #region if (contadorY <
_state.meta)

                                    if (_state.contadorY <
_state.meta)
                                    {
                                        //Send the mission at
the top of the list
                                        Current = Mfront;

                                        //Display missions
list on HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as
moving
                                        _robotStop = false;
```

```
                                        //Incrementar el
contador y en 1
                                        _state.contadorY =
_state.contadorY + 0.5;
                                        //Send order
                                        step.Update updateStep
= new step.Update();
updateStep.Body.MotionDirection = Current.direction;
updateStep.Body.AngleRequest = Current.angle;
updateStep.Body.DistanceRequest = Current.distance;
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;
_stepPort.Post(updateStep);
                                    }
                                    #endregion
                                    #region else if (contadorY
== _state.meta && contadorX != 0)
                                    else if (_state.contadorY
== _state.meta && _state.contadorX != 0)
                                    {
                                        //Send the mission at
the top of the list
                                        Current = Mleft;
                                        //Display missions
list on HMI
WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {
_mainHMI.DisplayCurrentMission(Current);
```

```
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                                }));

                                                //Mark the robot as
moving
                                                _robotStop = false;

                                                //giro de pi/2
radianes positivos en las coordenadas locales del robot
                                                //nuevos objeivos
                                                _state.meta =
Math.Abs(_state.contadorX);
                                                _state.contadorX = 0;
                                                _state.contadorY = 0;

                                                //Send order
                                                step.Update updateStep
= new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                            }

                                            #endregion

                                        }

                                    }

                                }
                                else if (_state.inversion == true &&
_state.inversion2 == true)
                                {
                                    #region if (_state.alarma == true
&& !izquierda && !girado && !comprobacion)
```

```
                            if (_state.alarma == true &&
_state.izquierda == false)
                            {
                                Current = Mleft;

                                //Display missions list on HMI
                                WinFormsServicePort.Post(new
FormInvoke(delegate()
                                {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                }));

                                //Mark the robot as moving
                                _robotStop = false;

                                //robot mirando a la izquierda

                                _state.izquierda = true;

                                //Marcar obstáculo como
evitado
                                _state.alarma = false;


                                //Send order
                                step.Update updateStep = new
step.Update();

updateStep.Body.MotionDirection = Current.direction;
                                updateStep.Body.AngleRequest =
Current.angle;

updateStep.Body.DistanceRequest = Current.distance;
                                updateStep.Body.LinearSpeedMax
= this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

                                _stepPort.Post(updateStep);
                            }
```

```
                            #endregion

                            else
                            {

                                #region  if (izquierda &&
!comprobacion && !_state.alarma && !girado)

                                if (_state.izquierda == true
&& _state.comprobacion == false && _state.alarma == false &&
_state.girado == false && _state.derecha == false)
                                {
                                    //Send the mission at the
top of the list
                                    Current = Mfront;

                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //Incrementar el contador
X en - recorrido

                                    _state.contadorX =
_state.contadorX - 0.5;
                                    _state.comprobacion =
true;
                                    _state.girado = false;
                                    _state.izquierda = true;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;
```

```
updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                #region else if (!izquierda &&
!comprobacion && !_state.alarma && girado)

                                else if (_state.izquierda ==
false && _state.comprobacion == false && !_state.alarma &&
_state.girado == true && _state.derecha == false)
                                {
                                    //Send the mission at the
top of the list
                                    Current = Mfront;

                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //Incrementar el contador
X en - recorrido

                                    _state.contadorY =
_state.contadorY - 0.5;
```

```
                                        _state.comprobacion =
true;
                                        _state.girado = true;
                                        _state.izquierda = false;


                                        //Send order
                                        step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                    }

                                    #endregion

                                    #region  else if (izquierda &&
_state.alarma)

                                    else if (_state.izquierda ==
true && _state.alarma == true)
                                    {
                                        //Send the mission at the
top of the list
                                        Current = Mleft;


                                        //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));
```

```
                                    //Mark the robot as moving
                                    _robotStop = false;

                                    _state.girado = true;
                                    _state.izquierda = false;
                                    _state.comprobacion =
false;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                #region else if (izquierda &&
comprobacion && !_state.alarma && !girado)

                                else if (_state.izquierda ==
true && _state.comprobacion == true && _state.alarma == false
&& _state.girado == false && _state.derecha == false)
                                {
                                    Current = Mright;

                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);
```

```
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //marcar comprobacion como
hecha
                                    _state.comprobacion =
false;

                                    //marcar izquierda falsa
                                    _state.izquierda = false;
                                    _state.girado = false;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                #region else if (!izquierda &&
comprobacion && !_state.alarma && girado)

                                else if (_state.izquierda ==
false && _state.comprobacion == true && _state.alarma == false
&& _state.girado == true && _state.derecha == false)
                                {
                                    Current = Mright;
```

```
                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //marcar comprobacion como
hecha
                                    _state.comprobacion =
false;

                                    //marcar izquierda falsa
                                    _state.izquierda = true;
                                    _state.girado = false;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                #region else if (girado &&
_state.alarma && !izquierda && !comprobacion)
```

```
                                    else if (_state.izquierda ==
false && _state.comprobacion == false && _state.alarma == true
&& _state.girado == true && _state.derecha == false)
                                    {
                                        Current = Mleft;

                                        //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as moving
                                        _robotStop = false;

                                        //marcar comprobacion como
hecha
                                        _state.comprobacion =
false;


                                        //marcar izquierda falsa
                                        _state.izquierda = false;
                                        _state.girado = false;
                                        _state.derecha = true;

                                        //Send order
                                        step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
```

```
                                    }
                                    #endregion

                                    #region else if(derecha &&
_state.alarma)
                                    else if (_state.izquierda ==
false && _state.comprobacion == false && _state.alarma == true
&& _state.girado == false && _state.derecha == true)
                                         {
                                        Current = Mleft;

                                        //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as moving
                                        _robotStop = false;

                                        //marcar comprobacion como
hecha
                                        _state.comprobacion =
false;

                                        //marcar izquierda falsa
                                        _state.izquierda = false;
                                        _state.girado = false;
                                        _state.derecha = false;
                                        _state.inversion = false;
                                        _state.inversion2 = false;

                                        //Send order
                                        step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;
```

```
updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                    }
                                #endregion

                                #region else if(derecha &&
!_state.alarma)
                                else if (_state.izquierda ==
false && _state.comprobacion == false && _state.alarma ==
false && _state.girado == false && _state.derecha == true)
                                {
                                    Current = Mfront;

                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //marcar comprobacion como
hecha
                                    _state.comprobacion =
true;


                                    //marcar izquierda falsa
                                    _state.izquierda = false;
                                    _state.girado = false;
                                    _state.derecha = true;

                                    //Send order
```

```
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }
                                #endregion

                                #region else if(derecha &&
!_state.alarma && comprobacion)
                                else if (_state.izquierda ==
false && _state.comprobacion == true && _state.alarma == false
&& _state.girado == false && _state.derecha == true)
                                {
                                    Current = Mright;

                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //marcar comprobacion como
hecha
                                    _state.comprobacion =
false;

                                    //marcar izquierda falsa
```

```
                                    _state.izquierda = false;
                                    _state.girado = true;
                                    _state.derecha = false;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }
                                #endregion

                                else
                                {

                                    #region if (contadorY <
_state.meta)

                                    if (_state.contadorY <
_state.meta)
                                    {
                                        //Send the mission at
the top of the list
                                        Current = Mfront;

                                        //Display missions
list on HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));
```

```
                                        //Mark the robot as
moving
                                        _robotStop = false;

                                        //Incrementar el
contador y en 1

                                        _state.contadorY =
_state.contadorY + 0.5;

                                        //Send order
                                        step.Update updateStep
= new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                    }

                                    #endregion

                                    #region else if (contadorY
== _state.meta && contadorX != 0)

                                    else if (_state.contadorY
== _state.meta && _state.contadorX != 0)
                                    {

                                        //Send the mission at
the top of the list
                                        Current = Mright;

                                        //Display missions
list on HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
```

```
                                        {
_mainHMI.DisplayCurrentMission(Current);
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));
                                        //Mark the robot as
moving
                                        _robotStop = false;
                                        //giro de pi/2
radianes negativos en las coordenadas locales del robot
                                        //nuevos objeivos
                                        _state.meta =
Math.Abs(_state.contadorX);
                                        _state.contadorX = 0;
                                        _state.contadorY = 0;
                                        //Send order
                                        step.Update updateStep
= new step.Update();
updateStep.Body.MotionDirection = Current.direction;
updateStep.Body.AngleRequest = Current.angle;
updateStep.Body.DistanceRequest = Current.distance;
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;
_stepPort.Post(updateStep);
                                    }
                                    #endregion
                                }
                            }
                        }
                    }
                #region meta alcanzada
```

```
                else
                //If there is no missions to send
                {
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                    {
                        _mainHMI.LogInfo("Objetivo
alcanzado");
                    }));
                }

                #endregion

            }
            #endregion

            #region robot en movimiento aun

            else
            {
                WinFormsServicePort.Post(new
FormInvoke(delegate()
                {
                    _mainHMI.LogInfo("El objetivo no se ha
alcanzado todavia");
                }));
            }

            #endregion
        }

        #endregion

        #region StartMission, AddMission, RemoveMission, Up
and Down
        /// <summary>
        /// User clicks on HMI: the next mission is sent send
next mission
        /// </summary>
        /// <param name="s"></param>
        void SendCoordenadasHandler(StartMission s)
        {
            SendCoordenadas();
        }
```

```
        /// <summary>
        /// Add mission to missionlist
        /// </summary>
        /// <param name="a"></param>
        void AddCoordenadasHandler(AddCoordenadas a)
        {
            _CoordenadasList.Add(a.AddedCoordenadas);
            WinFormsServicePort.Post(new FormInvoke(delegate()
            {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
            }));
        }

        /// <summary>
        /// Remove a mission
        /// </summary>
        /// <param name="r"></param>
        void RemoveCoordenadasHandler(RemoveCoordenadas r)
        {
            _CoordenadasList.RemoveAt(r.Index);
            WinFormsServicePort.Post(new FormInvoke(delegate()
            {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
            }));
        }

        /// <summary>
        /// increate the position of a mission in list
        /// </summary>
        /// <param name="r"></param>
        void UpCoordenadasHandler(UpCoordenadas r)
        {
            if (r.Index - 1 >= 0)
            {
                Coordenadas _m = new Coordenadas();
                _m = _CoordenadasList[r.Index];
                _CoordenadasList.RemoveAt(r.Index);
                _CoordenadasList.Insert(r.Index - 1, _m);
                WinFormsServicePort.Post(new
FormInvoke(delegate()
                {
```

```
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                }));
            }
        }

        /// <summary>
        /// Decrease position of a mission in a list
        /// </summary>
        /// <param name="r"></param>
        void DownCoordenadasHandler(DownCoordenadas r)
        {
            if (r.Index + 1 <= _CoordenadasList.Count - 1)
            {
                Coordenadas _m = new Coordenadas();
                _m = _CoordenadasList[r.Index];
                _CoordenadasList.RemoveAt(r.Index);
                _CoordenadasList.Insert(r.Index + 1, _m);
                WinFormsServicePort.Post(new
FormInvoke(delegate()
                {

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                }));
            }
        }

        #endregion
    }
}
```

Vemos en la primera parte en la cual se declaran los usings que destaca por su ausencia como era de esperar el using loc utilizado en el navegador 3.0, pero que obviamente tenemos como novedad el using step:

```
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Security.Permissions;
using navegador = navegador;
```

```
using xml = System.Xml;

using Microsoft.Ccr.Adapters.WinForms;
using System.Windows.Forms;
using types = Robosoft.RobuBOX.Core.Types.Proxy;

using ds = Microsoft.Dss.Services.Directory;
using submgr = Microsoft.Dss.Services.SubscriptionManager;

using step =
Robosoft.RobuBOX.Services.Processing.Control.StepDifferentialC
ontrol.Proxy;
using laser =
Robosoft.RobuBOX.Generic.Devices.LaserSensor.Proxy;

Al tener que hacer uso del control diferencial de pasos,
tendremos que definir los puertos principal y de
notificaciones.
       #region State and Partners
        //State definition
        //If the parameter file exist in "Microsoft Robotics
Studio (October 2006)\store"
        //load the state by the parameter file
        [InitialStatePartner(Optional = true, ServiceUri =
"store/navegador.xml")]
        private navegadorState _state;

        private step.StepDifferentialControlOperations
_stepPort;
        private step.StepDifferentialControlOperations
_stepNotifyPort = new
step.StepDifferentialControlOperations();
```

Además de los strings que utilizaremos para conectarlo.

```
        #region Members

        //internal port for HMI
        private HMIPort _hmiPort = new HMIPort();

        //Variables
        HMI _mainHMI;
        Mission Current = new Mission();
```

```
        public List<Coordenadas> _CoordenadasList = new
List<Coordenadas>();
        bool _robotStop = true;

        string _stepSubscriptionManager;
        string _stepSubscriber;

        string _laserSubMgr = string.Empty;
        string _laserSubscriber = string.Empty;


        #endregion
```

No encontraremos más novedades hasta la definición del estado inicial (por defecto) del navegador en el que tendremos que dar un valor a nuestros nuevos límites para las velocidades:

```
protected override void Start()
        {
            //Fill the state if the XML file doesn't exist
            if (_state == null)
            {
                _state = new navegadorState();
                _state.RobotUri =
"http://localhost:50000/directory";
                _state.CoordenadasList = new
List<Coordenadas>();
                _state.LinearSpeedMax = 0.3;
                _state.AngularSpeedMax = 0.3;

                Coordenadas inicial = new Coordenadas();
                inicial.x = 0;
                inicial.y = 0;
                _state.CoordenadasList.Add(inicial);//add
coordenada to coordenadas list

                _state.alarma = false;

                SaveState(_state);
            }
```

Además, ahora ya no vamos a hacer uso del periodic handler para mandar mensajes constantemente al motor, si no que al utilizar el control por pasos, vamos a definir un handler para enviar las coordenadas e iniciar el movimiento, y uno del tipo enumerador para conectarnos a los servicios.

Para ello primero definimos estos handlers en el interleave:

```
//Interleave (the link between a message and the response)
            MainPortInterleave.CombineWith(new Interleave(
                 new TeardownReceiverGroup(),
                 new ExclusiveReceiverGroup(
                         Arbiter.Receive<laser.Update>(true,
_laserNotifyPort, UpdateLaserHandler),

Arbiter.ReceiveWithIterator<Connect>(true, _hmiPort,
ConnectHandler),

Arbiter.Receive<SaveCoordenadas>(true, _hmiPort, SaveHandler),
                          Arbiter.Receive<DropFromGui>(true,
_hmiPort, DropFromGuiHandler)
                         ),
                 new ConcurrentReceiverGroup(
                        Arbiter.Receive<StartMission>(true,
_hmiPort, SendCoordenadasHandler),
                        Arbiter.Receive<AddCoordenadas>(true,
_hmiPort, AddCoordenadasHandler),

Arbiter.Receive<RemoveCoordenadas>(true, _hmiPort,
RemoveCoordenadasHandler),
                        Arbiter.Receive<UpCoordenadas>(true,
_hmiPort, UpCoordenadasHandler),
                        Arbiter.Receive<DownCoordenadas>(true,
_hmiPort, DownCoordenadasHandler)


                )));
```

La siguiente parte del código que debemos cambiar es el dss handler correspondiente al drop, usado para desconectarnos de los servicios, ya

que en este caso, tenemos que desconectarnos del control diferencial de pasos, y no del servicio de localización odometrica.

```
if (!_robotStop)
            {
                step.Update stopStep = new step.Update();
                stopStep.Body.MotionDirection =
step.DirectionEnumeration.Stop;
                stopStep.Body.AngleRequest = 0;
                stopStep.Body.DistanceRequest = 0;
                stopStep.Body.LinearSpeedMax = 0;
                stopStep.Body.AngularSpeedMax = 0;

                _stepPort.Post(stopStep);
                yield return
Arbiter.Choice<DefaultUpdateResponseType, W3C.Soap.Fault>(
                    stopStep.ResponsePort,
                    delegate(DefaultUpdateResponseType
response)
                    {

                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        error = true;
                        LogError("Error stopping robot");
                    });

            }
            if (_stepSubscriber != null)
            {
                submgr.DeleteSubscriptionMessage _body = new
submgr.DeleteSubscriptionMessage();
                _body.Subscriber = _stepSubscriber;
                _body.Expiration = DateTime.Now;
                submgr.DeleteSubscription _delete = new
submgr.DeleteSubscription();
                _delete.Body = _body;

ServiceForwarder<submgr.SubscriptionManagerPort>(_stepSubscrip
tionManager).Post(_delete);
                yield return
Arbiter.Choice<DefaultDeleteResponseType, W3C.Soap.Fault>(
```

```
                    _delete.ResponsePort,
                    delegate(DefaultDeleteResponseType
response)
                    {

                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        error = true;
                        LogError("Error deleting subscription
to StepDifferentilControl");
                    });
            }
            if (!error)
            {
                DirectoryDelete();
                Shutdown();

drop.ResponsePort.Post(DefaultDropResponseType.Instance);
            }
            else
            {
                drop.ResponsePort.Post(new W3C.Soap.Fault());
            }
            yield break;
        }
```

Vemos que aquí primero paramos el robot en caso de que este no estuviera parado anteriormente para desconectarlo del servicio, ya que si no no acabaría el movimiento hasta que cumpliera su misión aun cuando estuviese ya desconectado. Este problema no hay que solventarlo con la versión final, ya que simplemente no existe al ser la comunicación de la velocidad a los motores constante en cada iteración, y no cada vez que se cumple un "paso".

Tenemos que definir un nuevo comportamiento en el Update del láser para esta versión, ya que el comportamiento que tiene para alcanzar al objetivo es completamente diferente.

Nada más lejos de la realidad, el Update del láser contiene mucha menos lógica y una parte muy pequeña del algoritmo de navegación. De hecho, el servicio laser solo nos hace saltar una alarma que indica la presencia de un objeto delante del robot a una distancia lo suficientemente prudencial como para que este no se choque con él en el caso más extremo posible.

```
        #region Update Laser Handler

         public void UpdateLaserHandler(laser.Update update)
         {
             _state.alarma = false;

             foreach (laser.Echo E in update.Body.Echos)
             {
                 types.Point2D pointCartesian = new
types.Point2D();
                 pointCartesian.X = E.Distance *
Math.Cos(E.Angle);
                 pointCartesian.Y = E.Distance *
Math.Sin(E.Angle);

                 if (pointCartesian.Y < 0.4 && pointCartesian.Y
> -0.4 && pointCartesian.X > 0 && pointCartesian.X <= 0.8)
                 {
                     _state.alarma = true;
                 }
             }

        }

         #endregion
```

Como se puede ver, no es necesario un diagrama de flujo para entender el comportamiento esta vez de este handler.

Su única función es cambiar el valor de la alarma a true si un objeto es detectado a una distancia inferior a 0.8 metros del láser de rango en la dirección del eje x del robot, y dentro del área delimitada por los limites laterales distantes 0.4 metros del mismo, distancia más que suficiente para evitar incluso choques cuando el robot rota sobre su propio eje debido a la comparación de esta medida con las dimensiones de robuLAB10, las cuales se verán en la sección de planos.

Vamos a definir ahora él sabe handler, el encargado de guardar esas coordenadas que hemos definido y lanzaras al algoritmo de control de navegación.

```
/// <summary>
        /// Save current missions in state xml file
        /// </summary>
        /// <param name="s"></param>
        void SaveHandler(SaveCoordenadas s)
        {
            this._state.CoordenadasList.Clear();
            foreach (Coordenadas coord in _CoordenadasList)
            {
                this._state.CoordenadasList.Add(coord); ;
            }
            SaveState(_state);
```

Despues de guardar estas coordenadas en la lista de coordenadas del estado del robot, la cual nos permite guardar mas de una coordenada, debemos definir las variables que definen el objetivo de las misiones:

Dijamos los valores de distx y disty, dos variables temporales de tipo double, a los valores que tenemos de x e y en las coordenadas que ocupan el primer lugar de la lista.

Ahora calculamos el ángulo que forma la orientacion del robot con las coordenadas objetivo mediante unos sencillos calculos de trigonometria, que consisten simplemente en calcular el arco tangente de los catetos del triangulo formado por la posicion actual del robot y las coordenadas x y del objetivo.

Posteriormente se calcula la meta con el coseno del angulo formado e incluso se hace la media entre las dos valores obtenidos.

```
double distX = _CoordenadasList[0].x;
double distY = _CoordenadasList[0].y;
double beta = Math.Atan(distY / distX);
double alpha_rad = Math.PI / 2 - beta;
_state.alpha = alpha_rad * 180 / Math.PI;

if (distX == 0)
{
    _state.meta = distY;
    if (distY > 0)
        _state.alpha = 0;
    else if (distY < 0)
        _state.alpha = 180;
}
else if (distY == 0)
{
    _state.meta = distX;
    if (distX > 0)
        _state.alpha = 90;
    else if (distX < 0)
        _state.alpha = -90;
}
else
{
    double hx = distX / Math.Cos(beta);
    double hy = distY / Math.Sin(beta);
    _state.meta = (hx + hy) / 2;
}
```

Ahora, cada vez que lancemos las coordenadas, reseteamos todos los contadores y las variables de control del algoritmo.

```
            //contadores
            _state.contadorX = 0;
            _state.contadorY = 0;

            //variables de control iniciales
            _state.giro = false;
            _state.izquierda = false;
            _state.comprobacion = false;
            _state.girado = false;
            _state.inversion = false;
            _state.inversion2 = false;

            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
                _mainHMI.LogInfo("Coordenadas lanzadas");
            }));

        }

        #endregion
```

La conexión a los servicios se realiza de igual manera.

La única excepción es que el robot uri lo pasamos con un string que leemos del campo de texto como vimos en el HMI, en los argumentos de entrada (co)

```
IEnumerator<ITask> ConnectHandler(Connect co)
```

Ahora veremos la parte más interesante del código, que es el algoritmo de navegación incluido en el handler SendCoordenadas.

Básicamente se sigue una estructura de lógica de toma de decisiones

```
#region SendCoordenadas Logica

        void SendCoordenadas()
        {

            #region misiones
```

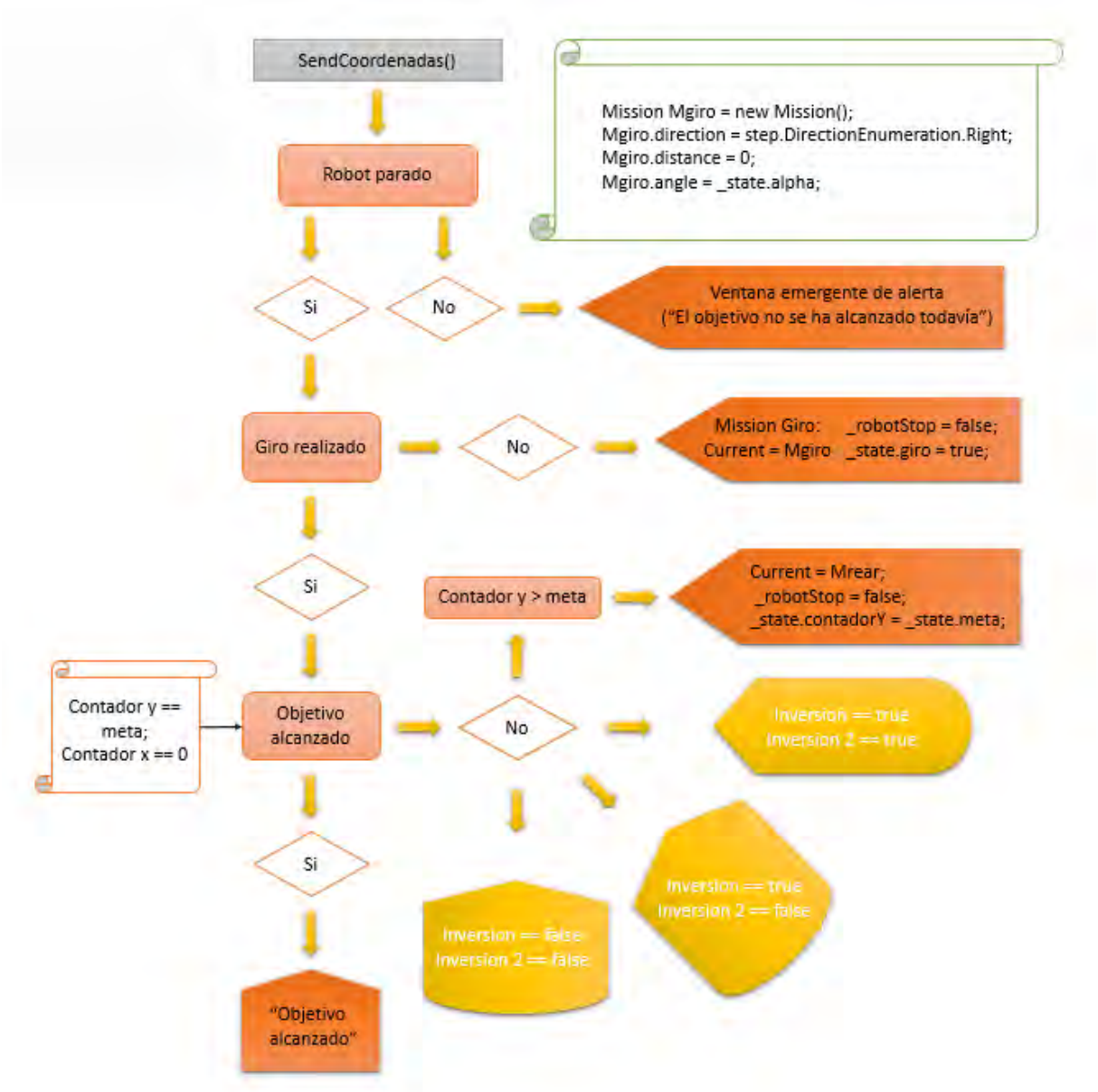
Inicialmente, inicializamos un conjunto de misiones standard que son las que utilizaremos:

```
            // conjunto de misiones standard
            Mission Mfront = new Mission();
            Mfront.direction =
step.DirectionEnumeration.Front;
            Mfront.distance = 0.5;
            Mfront.angle = 0;

            Mission Mright = new Mission();
            Mright.direction =
step.DirectionEnumeration.Right;
            Mright.distance = 0;
            Mright.angle = 90;

            Mission Mleft = new Mission();
            Mleft.direction = step.DirectionEnumeration.Left;
            Mleft.distance = 0;
            Mleft.angle = 90;

            Mission Mrear = new Mission();
            Mrear.direction = step.DirectionEnumeration.Rear;
            Mrear.distance = _state.contadorY - _state.meta;
            Mrear.angle = 0;

            #endregion
```

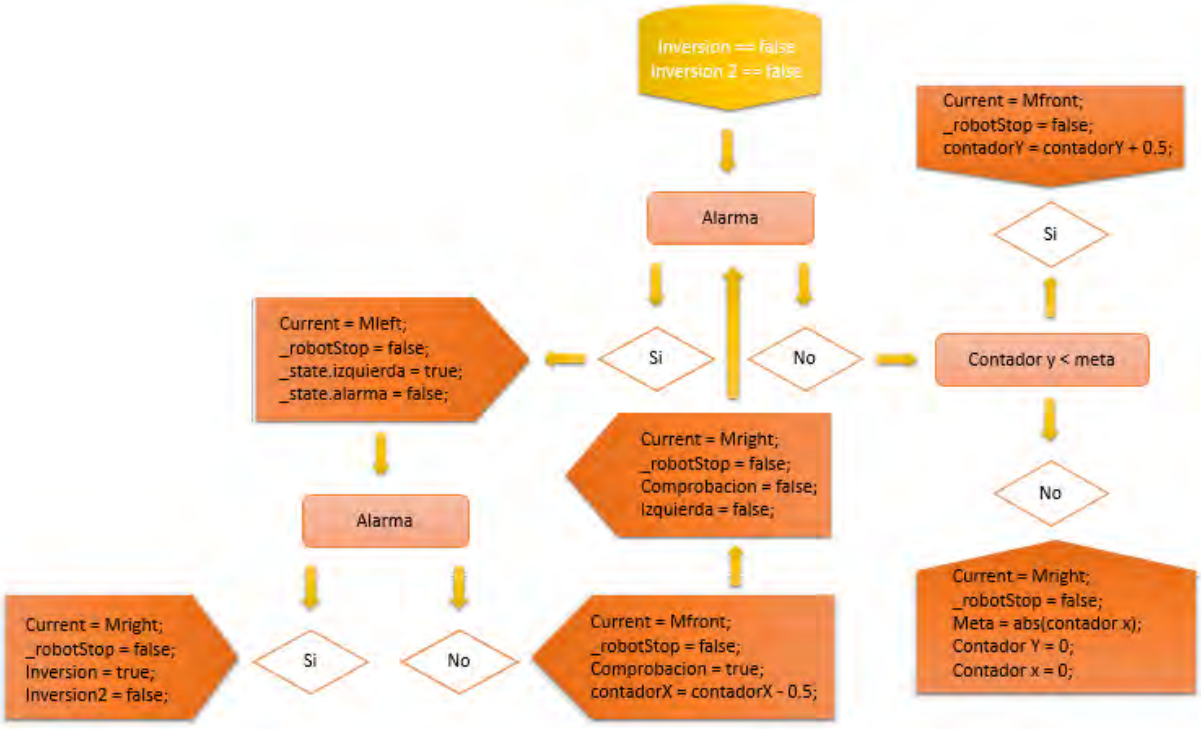
La gran comodidad que nos ofrece lo que acabamos de hacer, esque a partir de ahora, cuando nos queramos referir a una de estas misiones, lo haremos simplemente mediante su nombre, por ejemplo, Mfront.

Vemos ahora el algoritmo con la lógica de navegación.

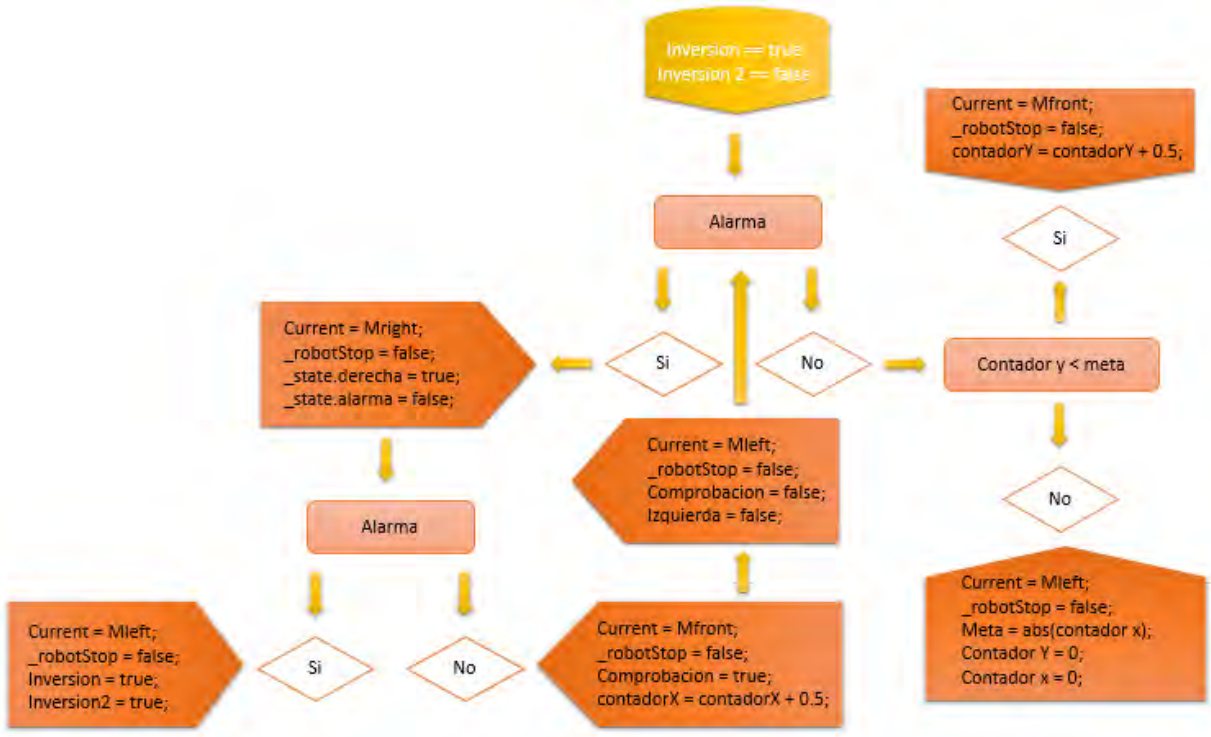
La lógica viene representada en los siguientes diagramas (figura 44, 45 y 46):



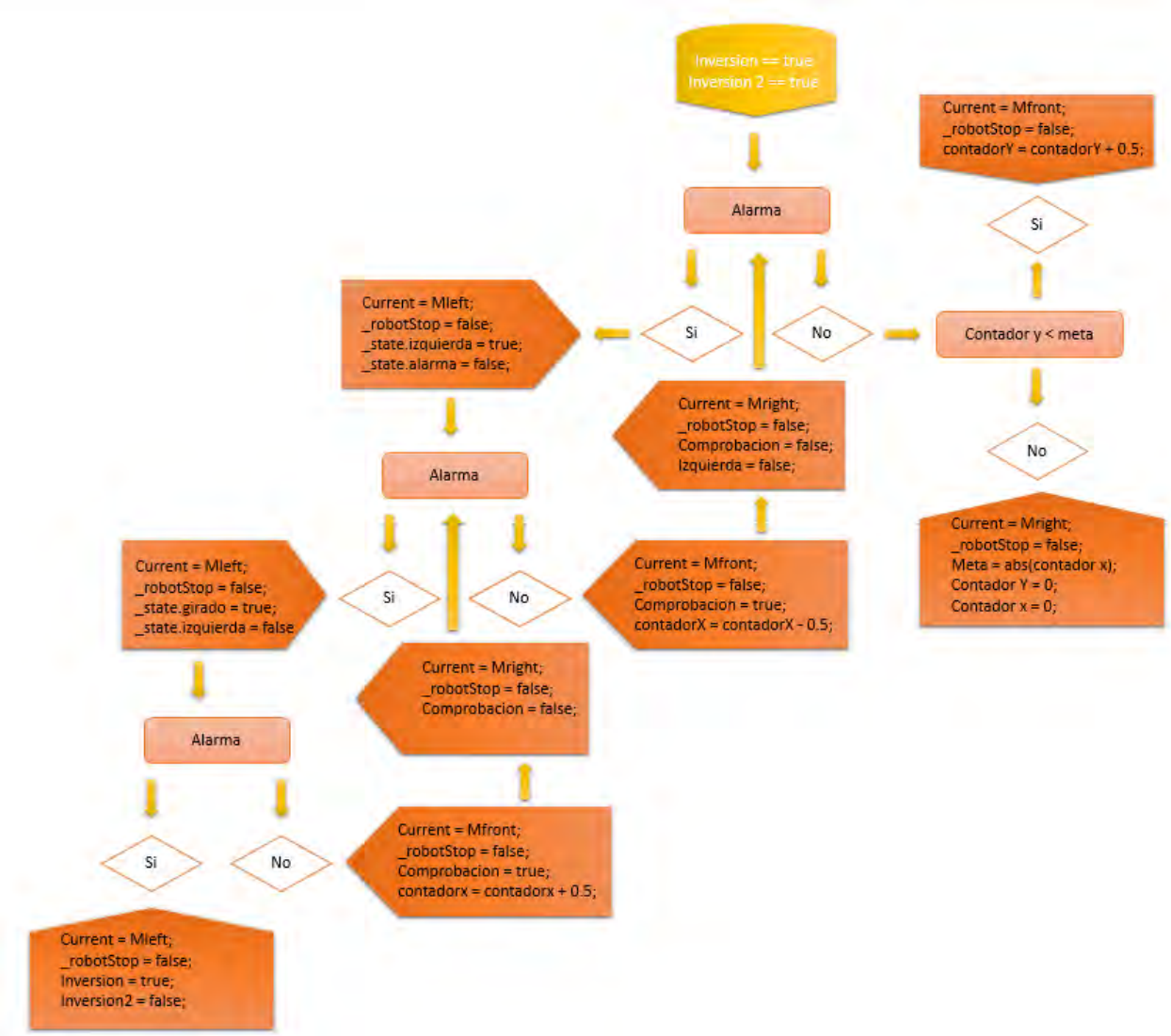
(figura 44)


Inversion == false
Inversion 2 == false
Alarma
Si
No
Current = Mleft;
_robotStop = false;
_state.izquierda = true;
_state.alarma = false;
Current = Mright;
_robotStop = false;
Comprobacion = false;
Izquierda = false;
Current = Mfront;
_robotStop = false;
contadorY = contadorY + 0.5;
Contador y < meta
Current = Mright;
_robotStop = false;
Meta = abs(contador x);
Contador Y = 0;
Contador x = 0;
Current = Mright;
_robotStop = false;
Inversion = true;
Inversion2 = false;
Current = Mfront;
_robotStop = false;
Comprobacion = true;
contadorX = contadorX - 0.5;

(figura 10)


Inversion == true
Inversion 2 == false
Alarma
Si
No
Current = Mright;
_robotStop = false;
_state.derecha = true;
_state.alarma = false;
Current = Mleft;
_robotStop = false;
Comprobacion = false;
Izquierda = false;
Current = Mfront;
_robotStop = false;
contadorY = contadorY + 0.5;
Contador y < meta
Current = Mleft;
_robotStop = false;
Meta = abs(contador x);
Contador Y = 0;
Contador x = 0;
Current = Mleft;
_robotStop = false;
Inversion = true;
Inversion2 = true;
Current = Mfront;
_robotStop = false;
Comprobacion = true;
contadorX = contadorX + 0.5;

(figura 45)



(figura 46)

Como podemos apreciar, el navegador llama al handler send coordenadas cuando se presiona el botón start, y cada vez que el robot da el paso demandado actual.

De hecho, hasta que el robot no se para de nuevo, no se lanza la siguiente misión.

Inicialmente, el robot realiza un giro para encarar al objetivo, de esta forma, la condición de llegada será que el contador y sea igual al valor de la meta, y que el contador x sea igual a 0. Una vez hecho esto, tenemos 3 estados del robot definidos por las variables booleanas inversión e inversión2:

- Estado 1
  - Inversión = false;
  - Inversion2 = false;
- Estado 2
  - Inversión = true;
  - Inversion2 = false;
- Estado 3
  - Inversión = true;
  - Inversion2 = true;

Dependiendo de en qué estado nos encontremos, tenemos uno u otro método de actuación. El estado 3 es el único infalible, siempre llega al objetivo.

Aun así, los otros dos estados se incluyen ya que hacen más rápido y eficiente el navegador.

Para definir cada opción que puede realizar el robot, definimos:

1. Misión actual (current = Mactual)
2. Muestra de la misión actual en el HMI.
3. Marcar que el robot no está parado.
4. Actualizar las variables de control.

5. Postear el paso actual en el control diferencial de pasos.

La sintaxis del código para definir el algoritmo de navegación lo definimos por lo tanto:

```
#region if (!giro)

                if (_state.giro == false)
                {
                    Mission Mgiro = new Mission();
                    Mgiro.direction =
step.DirectionEnumeration.Right;
                    Mgiro.distance = 0;
                    Mgiro.angle = _state.alpha;

                    //Send the mission at the top of the list
                    Current = Mgiro;

                    //Display missions list on HMI
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                    }));

                    //Mark the robot as moving
                    _robotStop = false;

                    //Marcar giro como realizado

                    _state.giro = true;

                    //Send order
                    step.Update updateStep = new
step.Update();
                    updateStep.Body.MotionDirection =
Current.direction;
                    updateStep.Body.AngleRequest =
Current.angle;
```

```
                    updateStep.Body.DistanceRequest =
Current.distance;
                    updateStep.Body.LinearSpeedMax =
this._state.LinearSpeedMax;
                    updateStep.Body.AngularSpeedMax =
this._state.AngularSpeedMax;
                    _stepPort.Post(updateStep);
                }

                #endregion

                else if (_state.contadorX != 0 ||
_state.contadorY != _state.meta)
                {

                    #region if (_state.contadorY >
_state.meta)

                    if (_state.contadorY > _state.meta)
                    {
                        //Send the mission at the top of the
list
                        Current = Mrear;

                        //Display missions list on HMI
                        WinFormsServicePort.Post(new
FormInvoke(delegate()
                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                        }));

                        //Mark the robot as moving
                        _robotStop = false;

                        //legada
                        _state.contadorY = _state.meta;

                        //Send order
                        step.Update updateStep = new
step.Update();
                        updateStep.Body.MotionDirection =
Current.direction;
```

```
                        updateStep.Body.AngleRequest =
Current.angle;
                        updateStep.Body.DistanceRequest =
Current.distance;
                        updateStep.Body.LinearSpeedMax =
this._state.LinearSpeedMax;
                        updateStep.Body.AngularSpeedMax =
this._state.AngularSpeedMax;
                        _stepPort.Post(updateStep);
                    }

                    #endregion


                    else
                    {

                        if (_state.inversion == false &&
_state.inversion2 == false)
                        {

                            #region if (_state.alarma == true
&& _state.izquierda == false)

                            if (_state.alarma == true &&
_state.izquierda == false)
                            {
                                Current = Mleft;

                                //Display missions list on HMI
                                WinFormsServicePort.Post(new
FormInvoke(delegate()
                                {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                }));

                                //Mark the robot as moving
                                _robotStop = false;

                                //robot mirando a la izquierda

                                _state.izquierda = true;
```

```
                                    //Marcar obstáculo como
evitado
                                    _state.alarma = false;


                                    //Send order
                                    step.Update updateStep = new
step.Update();

updateStep.Body.MotionDirection = Current.direction;
                                    updateStep.Body.AngleRequest =
Current.angle;

updateStep.Body.DistanceRequest = Current.distance;
                                    updateStep.Body.LinearSpeedMax
= this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

                                    _stepPort.Post(updateStep);
                                }

                                #endregion

                                else
                                {

                                    #region  if (izquierda &&
!comprobacion && !_state.alarma)

                                    if (_state.izquierda == true
&& _state.comprobacion == false && _state.alarma == false)
                                    {
                                        //Send the mission at the
top of the list
                                        Current = Mfront;


                                        //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);
```

```
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //Incrementar el contador
X en - recorrido

                                    _state.contadorX =
_state.contadorX - 0.5;
                                    _state.comprobacion =
true;


                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                #region  else if (izquierda &&
_state.alarma)

                                else if (_state.izquierda ==
true && _state.alarma)
                                {
                                    //Send the mission at the
top of the list
                                    Current = Mright;
```

```
                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //inversion "espejo" del
problema
                                    _state.inversion = true;
                                    _state.inversion2 = false;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                #region (izquierda &&
comprobacion && !_state.alarma)

                                else if (_state.izquierda ==
true && _state.comprobacion == true && _state.alarma == false)
                                {
                                    Current = Mright;
```

```
                                    //Display missions list on
HMI
WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {
_mainHMI.DisplayCurrentMission(Current);
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));
                                    //Mark the robot as moving
                                    _robotStop = false;
                                    //marcar comprobacion como
hecha
                                    _state.comprobacion =
false;
                                    //marcar izquierda falsa
                                    _state.izquierda = false;
                                    //Send order
                                    step.Update updateStep =
new step.Update();
updateStep.Body.MotionDirection = Current.direction;
updateStep.Body.AngleRequest = Current.angle;
updateStep.Body.DistanceRequest = Current.distance;
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;
_stepPort.Post(updateStep);
                                }
                                #endregion
                                else
                                {
```

```
                                        #region if
(_state.contadorY < _state.meta)

                                        if (_state.contadorY <
_state.meta)
                                        {
                                            //Send the mission at
the top of the list
                                            Current = Mfront;

                                            //Display missions
list on HMI
WinFormsServicePort.Post(new FormInvoke(delegate()
                                            {
_mainHMI.DisplayCurrentMission(Current);
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                            }));

                                            //Mark the robot as
moving
                                            _robotStop = false;

                                            //Incrementar el
contador y en 1

                                            _state.contadorY =
_state.contadorY + 0.5;

                                            //Send order
                                            step.Update updateStep
= new step.Update();
updateStep.Body.MotionDirection = Current.direction;
updateStep.Body.AngleRequest = Current.angle;
updateStep.Body.DistanceRequest = Current.distance;
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;
```

```
_stepPort.Post(updateStep);
                                    }

                                    #endregion

                                    #region else if
(_state.contadorY == _state.meta && _state.contadorX != 0)

                                    else if (_state.contadorY
== _state.meta && _state.contadorX != 0)
                                    {

                                        //Send the mission at
the top of the list
                                        Current = Mright;

                                        //Display missions
list on HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as
moving
                                        _robotStop = false;

                                        //giro de pi/2
radianes negativos en las coordenadas locales del robot
                                        //nuevos objeivos
                                        _state.meta =
Math.Abs(_state.contadorX);
                                        _state.contadorX = 0;
                                        _state.contadorY = 0;

                                        //Send order
                                        step.Update updateStep
= new step.Update();

updateStep.Body.MotionDirection = Current.direction;
```

```
updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                    }

                                    #endregion
                                }

                            }
                        }
                        else if (_state.inversion == true &&
_state.inversion2 == false)
                        {

                            #region if (_state.alarma == true
&& !izquierda)

                            if (_state.alarma == true &&
_state.izquierda == false)
                            {
                                Current = Mright;

                                //Display missions list on HMI
                                WinFormsServicePort.Post(new
FormInvoke(delegate()
                                {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                }));

                                //Mark the robot as moving
                                _robotStop = false;

                                //robot mirando a la izquierda

                                _state.izquierda = true;
```

```
                                    //Marcar obstáculo como
evitado
                                    _state.alarma = false;


                                    //Send order
                                    step.Update updateStep = new
step.Update();

updateStep.Body.MotionDirection = Current.direction;
                                    updateStep.Body.AngleRequest =
Current.angle;

updateStep.Body.DistanceRequest = Current.distance;
                                    updateStep.Body.LinearSpeedMax
= this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

                                    _stepPort.Post(updateStep);
                                }

                                #endregion

                                else
                                {

                                    #region if (izquierda &&
!comprobacion && !_state.alarma)

                                    if (_state.izquierda == true
&& _state.comprobacion == false && _state.alarma == false)
                                    {
                                        //Send the mission at the
top of the list
                                        Current = Mfront;


                                        //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);
```

```
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //Incrementar el contador
X en - recorrido

                                    _state.contadorX =
_state.contadorX + 0.5;
                                    _state.comprobacion =
true;


                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                #region else if (izquierda &&
_state.alarma)

                                else if (_state.izquierda ==
true && _state.alarma)
                                {
                                    //Send the mission at the
top of the list
                                    Current = Mleft;
```

```
                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //inversion "espejo" del
problema
                                    _state.inversion = true;
                                    _state.inversion2 = true;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                #region else if (izquierda &&
comprobacion && !_state.alarma)

                                else if (_state.izquierda ==
true && _state.comprobacion == true && !_state.alarma)
                                {
                                    Current = Mleft;
```

```
                                    //Display missions list on
HMI
WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {
_mainHMI.DisplayCurrentMission(Current);
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));
                                    //Mark the robot as moving
                                    _robotStop = false;
                                    //marcar comprobacion como
hecha
                                    _state.comprobacion =
false;
                                    //marcar izquierda falsa
                                    _state.izquierda = false;
                                    //Send order
                                    step.Update updateStep =
new step.Update();
updateStep.Body.MotionDirection = Current.direction;
updateStep.Body.AngleRequest = Current.angle;
updateStep.Body.DistanceRequest = Current.distance;
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;
_stepPort.Post(updateStep);
                                }
                                #endregion
                                else
                                {
```

```
                                        #region if (contadorY <
_state.meta)

                                        if (_state.contadorY <
_state.meta)
                                        {
                                            //Send the mission at
the top of the list
                                            Current = Mfront;

                                            //Display missions
list on HMI
WinFormsServicePort.Post(new FormInvoke(delegate()
                                            {
_mainHMI.DisplayCurrentMission(Current);
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                            }));

                                            //Mark the robot as
moving
                                            _robotStop = false;

                                            //Incrementar el
contador y en 1

                                            _state.contadorY =
_state.contadorY + 0.5;

                                            //Send order
                                            step.Update updateStep
= new step.Update();
updateStep.Body.MotionDirection = Current.direction;
updateStep.Body.AngleRequest = Current.angle;
updateStep.Body.DistanceRequest = Current.distance;
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;
```

```
_stepPort.Post(updateStep);
                                    }

                                    #endregion

                                    #region else if (contadorY
== _state.meta && contadorX != 0)

                                    else if (_state.contadorY
== _state.meta && _state.contadorX != 0)
                                    {

                                        //Send the mission at
the top of the list
                                        Current = Mleft;

                                        //Display missions
list on HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as
moving
                                        _robotStop = false;

                                        //giro de pi/2
radianes positivos en las coordenadas locales del robot
                                        //nuevos objeivos
                                        _state.meta =
Math.Abs(_state.contadorX);
                                        _state.contadorX = 0;
                                        _state.contadorY = 0;

                                        //Send order
                                        step.Update updateStep
= new step.Update();

updateStep.Body.MotionDirection = Current.direction;
```

```
updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                    }

                                    #endregion

                                }

                            }

                        }
                        else if (_state.inversion == true &&
_state.inversion2 == true)
                        {
                            #region if (_state.alarma == true
&& !izquierda && !girado && !comprobacion)

                            if (_state.alarma == true &&
_state.izquierda == false)
                            {
                                Current = Mleft;

                                //Display missions list on HMI
                                WinFormsServicePort.Post(new
FormInvoke(delegate()
                                {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                }));

                                //Mark the robot as moving
                                _robotStop = false;

                                //robot mirando a la izquierda
```

```
                                    _state.izquierda = true;

                                    //Marcar obstáculo como
evitado
                                    _state.alarma = false;


                                    //Send order
                                    step.Update updateStep = new
step.Update();

updateStep.Body.MotionDirection = Current.direction;
                                    updateStep.Body.AngleRequest =
Current.angle;

updateStep.Body.DistanceRequest = Current.distance;
                                    updateStep.Body.LinearSpeedMax
= this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

                                    _stepPort.Post(updateStep);
                                }

                                #endregion

                                else
                                {

                                    #region  if (izquierda &&
!comprobacion && !_state.alarma && !girado)

                                    if (_state.izquierda == true
&& _state.comprobacion == false && _state.alarma == false &&
_state.girado == false && _state.derecha == false)
                                    {
                                        //Send the mission at the
top of the list
                                        Current = Mfront;

                                        //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {
```

```
_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //Incrementar el contador
X en - recorrido

                                    _state.contadorX =
_state.contadorX - 0.5;
                                    _state.comprobacion =
true;
                                    _state.girado = false;
                                    _state.izquierda = true;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                #region else if (!izquierda &&
!comprobacion && !_state.alarma && girado)

                                else if (_state.izquierda ==
false && _state.comprobacion == false && !_state.alarma &&
_state.girado == true && _state.derecha == false)
                                {
```

```
                                    //Send the mission at the
top of the list
                                    Current = Mfront;

                                    //Display missions list on
HMI
WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {
_mainHMI.DisplayCurrentMission(Current);
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //Incrementar el contador
X en - recorrido

                                    _state.contadorY =
_state.contadorY - 0.5;
                                    _state.comprobacion =
true;
                                    _state.girado = true;
                                    _state.izquierda = false;


                                    //Send order
                                    step.Update updateStep =
new step.Update();
updateStep.Body.MotionDirection = Current.direction;
updateStep.Body.AngleRequest = Current.angle;
updateStep.Body.DistanceRequest = Current.distance;
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;
_stepPort.Post(updateStep);
                                }
```

```
                                #endregion

                                #region  else if (izquierda &&
_state.alarma)

                                else if (_state.izquierda ==
true && _state.alarma == true)
                                {
                                    //Send the mission at the
top of the list
                                    Current = Mleft;

                                    //Display missions list on
HMI
WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {
_mainHMI.DisplayCurrentMission(Current);
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    _state.girado = true;
                                    _state.izquierda = false;
                                    _state.comprobacion =
false;

                                    //Send order
                                    step.Update updateStep =
new step.Update();
updateStep.Body.MotionDirection = Current.direction;
updateStep.Body.AngleRequest = Current.angle;
updateStep.Body.DistanceRequest = Current.distance;
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;
```

```
_stepPort.Post(updateStep);
                                    }

                                    #endregion

                                    #region else if (izquierda &&
comprobacion && !_state.alarma && !girado)

                                    else if (_state.izquierda ==
true && _state.comprobacion == true && _state.alarma == false
&& _state.girado == false && _state.derecha == false)
                                    {
                                        Current = Mright;

                                        //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as moving
                                        _robotStop = false;

                                        //marcar comprobacion como
hecha
                                        _state.comprobacion =
false;

                                        //marcar izquierda falsa
                                        _state.izquierda = false;
                                        _state.girado = false;

                                        //Send order
                                        step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;
```

```
updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                    }

                                    #endregion

                                    #region else if (!izquierda &&
comprobacion && !_state.alarma && girado)

                                    else if (_state.izquierda ==
false && _state.comprobacion == true && _state.alarma == false
&& _state.girado == true && _state.derecha == false)
                                    {
                                        Current = Mright;

                                        //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as moving
                                        _robotStop = false;

                                        //marcar comprobacion como
hecha
                                        _state.comprobacion =
false;

                                        //marcar izquierda falsa
                                        _state.izquierda = true;
                                        _state.girado = false;

                                        //Send order
```

```
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }

                                #endregion

                                #region else if (girado &&
_state.alarma && !izquierda && !comprobacion)
                                else if (_state.izquierda ==
false && _state.comprobacion == false && _state.alarma == true
&& _state.girado == true && _state.derecha == false)
                                {
                                    Current = Mleft;

                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //marcar comprobacion como
hecha
                                    _state.comprobacion =
false;
```

```
                                    //marcar izquierda falsa
                                    _state.izquierda = false;
                                    _state.girado = false;
                                    _state.derecha = true;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }
                                #endregion

                                #region else if(derecha &&
_state.alarma)
                                else if (_state.izquierda ==
false && _state.comprobacion == false && _state.alarma == true
&& _state.girado == false && _state.derecha == true)
                                     {
                                    Current = Mleft;

                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;
```

```
                                    //marcar comprobacion como
hecha
                                    _state.comprobacion =
false;

                                    //marcar izquierda falsa
                                    _state.izquierda = false;
                                    _state.girado = false;
                                    _state.derecha = false;
                                    _state.inversion = false;
                                    _state.inversion2 = false;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                     }
                                #endregion

                                #region else if(derecha &&
!_state.alarma)
                                else if (_state.izquierda ==
false && _state.comprobacion == false && _state.alarma ==
false && _state.girado == false && _state.derecha == true)
                                {
                                    Current = Mfront;

                                    //Display missions list on
HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                    {

_mainHMI.DisplayCurrentMission(Current);
```

```
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                    }));

                                    //Mark the robot as moving
                                    _robotStop = false;

                                    //marcar comprobacion como
hecha
                                    _state.comprobacion =
true;

                                    //marcar izquierda falsa
                                    _state.izquierda = false;
                                    _state.girado = false;
                                    _state.derecha = true;

                                    //Send order
                                    step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                }
                                #endregion

                                #region else if(derecha &&
!_state.alarma && comprobacion)
                                else if (_state.izquierda ==
false && _state.comprobacion == true && _state.alarma == false
&& _state.girado == false && _state.derecha == true)
                                {
                                    Current = Mright;

                                    //Display missions list on
HMI
```

```
WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as moving
                                        _robotStop = false;

                                        //marcar comprobacion como
hecha
                                        _state.comprobacion =
false;

                                        //marcar izquierda falsa
                                        _state.izquierda = false;
                                        _state.girado = true;
                                        _state.derecha = false;

                                        //Send order
                                        step.Update updateStep =
new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;

updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                    }
                                    #endregion

                                    else
                                    {

                                        #region if (contadorY <
_state.meta)
```

```
                                    if (_state.contadorY <
_state.meta)
                                    {
                                        //Send the mission at
the top of the list
                                        Current = Mfront;

                                        //Display missions
list on HMI
WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {
_mainHMI.DisplayCurrentMission(Current);
_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as
moving
                                        _robotStop = false;

                                        //Incrementar el
contador y en 1

                                        _state.contadorY =
_state.contadorY + 0.5;

                                        //Send order
                                        step.Update updateStep
= new step.Update();
updateStep.Body.MotionDirection = Current.direction;
updateStep.Body.AngleRequest = Current.angle;
updateStep.Body.DistanceRequest = Current.distance;
updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;
updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;
_stepPort.Post(updateStep);
                                    }
```

```
                                    #endregion

                                    #region else if (contadorY
== _state.meta && contadorX != 0)

                                    else if (_state.contadorY
== _state.meta && _state.contadorX != 0)
                                    {

                                        //Send the mission at
the top of the list
                                        Current = Mright;

                                        //Display missions
list on HMI

WinFormsServicePort.Post(new FormInvoke(delegate()
                                        {

_mainHMI.DisplayCurrentMission(Current);

_mainHMI.DisplayCoordenadasList(_CoordenadasList);
                                        }));

                                        //Mark the robot as
moving
                                        _robotStop = false;

                                        //giro de pi/2
radianes negativos en las coordenadas locales del robot
                                        //nuevos objeivos
                                        _state.meta =
Math.Abs(_state.contadorX);
                                        _state.contadorX = 0;
                                        _state.contadorY = 0;

                                        //Send order
                                        step.Update updateStep
= new step.Update();

updateStep.Body.MotionDirection = Current.direction;

updateStep.Body.AngleRequest = Current.angle;
```

```
updateStep.Body.DistanceRequest = Current.distance;

updateStep.Body.LinearSpeedMax = this._state.LinearSpeedMax;

updateStep.Body.AngularSpeedMax = this._state.AngularSpeedMax;

_stepPort.Post(updateStep);
                                        }

                                        #endregion
                                    }

                                }
                            }
                        }

                        #region meta alcanzada
                        else
                        //If there is no missions to send
                        {
                            WinFormsServicePort.Post(new
FormInvoke(delegate()
                            {
                                _mainHMI.LogInfo("Objetivo
alcanzado");
                            }));
                        }

                        #endregion

                    }
                    #endregion
Ahora ya solo definimos lo que pasa si el robot esta en
movimiento aun y llamamos de nuevo a sendcoordenadas(), esto
es, presionamos de nuevo el botón start.

                    #region robot en movimiento aun

                    else
                    {
                        WinFormsServicePort.Post(new
FormInvoke(delegate()
                        {
```

```
                    _mainHMI.LogInfo("El objetivo no se ha
alcanzado todavia");
                }));
            }

            #endregion
        }

        #endregion
```

Ya como ultima novedad a comentar, el handler que define el funcionamiento del boton de start:
En este handler, antes activabamos el periodic handler. Pues bien, el cambio consiste simplemente en que este handler activa el handler de sendcoordenadas:

```
/// <summary>
        /// User clicks on HMI: the next mission is sent send
next mission
        /// </summary>
        /// <param name="s"></param>
        void SendCoordenadasHandler(StartMission s)
        {
            SendCoordenadas();
        }
```

3.3 Servicio explorador

3.3.1 Introducción

Tras el desarrollo del programa previo, el navegador, ya somos capaces de enviar al robot a ciertas coordenadas, incluso cuando el camino se encuentra obstaculizado por objetos, e incluso aun cuando ni siquiera conocemos como es esta distribución de objetos.

Esto adquiere gran relevancia en una amplia gama de aplicaciones, entre las que destaca la de poder llevar al robot a cierto punto en el cual se encuentre un objeto, una persona… etc, deseado y se pretenda en un

desarrollo posterior, quizás, manipular ese objeto o incluso interaccionar con la persona en cuestión.

Para ello necesitamos principalmente dos puntos fundamentales:

- Conocer el entorno en el que el robot va a operar, sus límites, su forma… en definitiva, elaborar un mapa.
- Localizar el objeto, la persona, o lo que sea con lo que queramos que nuestro robot interaccione en este entorno.

Para esta finalidad es para la que el servicio explorador se ha desarrollado.

El programa explorador utiliza el láser de rango para realizar un mapeado del entorno (SLAM) en el que el robot trabaja, e incluso para sortear posibles obstáculos que pueda haber en el mismo, utiliza los encoders de las ruedas para realizar mediante técnicas de odometría un registro de la localización y la orientación del robot en cada instante, y mediante una cámara que se le ha incorporado al robot, realiza una captura de imágenes de este entorno para ser procesadas posteriormente utilizando MATLAB y localizar el objeto que el usuario desea detectar, dibujando sobre el mapeado del entorno el lugar exacto donde se encuentra tal objeto, proporcionando sus coordenadas para que posteriormente puedan ser introducidas en el programa navegador para que el robot pueda viajar directamente a ese lugar.

3.3.2 Hardware utilizado

El hardware utilizado es el mismo que en el del programa navegador, a excepción de la incorporación de la cámara con la que cogemos las imágenes:

Se trata de una webcam, concretamente el modelo *Logitech HD Webcam C270* (figura 47).

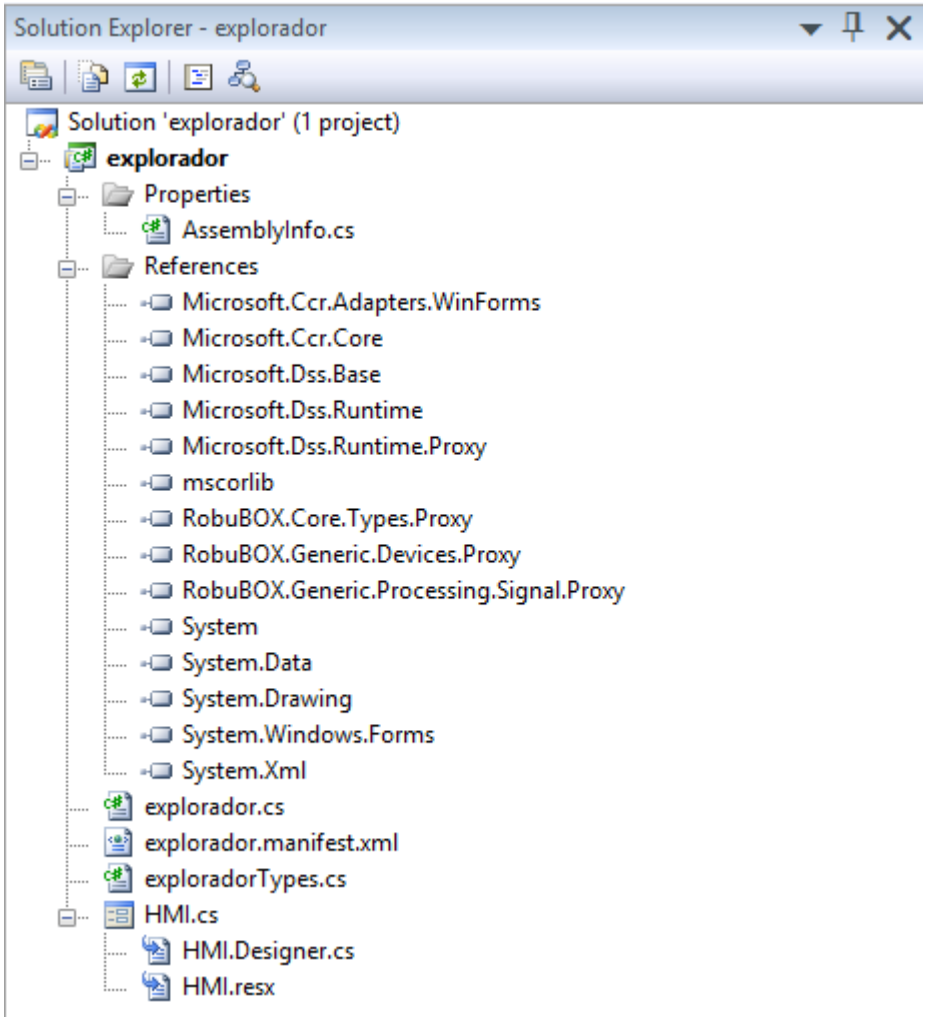
(figura 47)

Imagen tomada de la web http://www.logitech.com/es-es/product/hd-webcam-c270

Sus características están recogidas en la sección de datos de partida.

3.3.3 Navegador de soluciones

En el explorador de soluciones del explorador encontramos los siguientes apartados (figura 48):

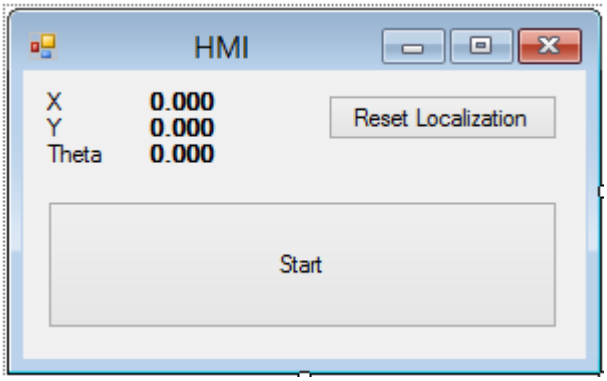


(figura 48)

Como vemos, el esquema general es el seguido por el programa navegador, aunque su contenido tiene numerosas variaciones.

3.3.4 HMI

Primeramente echaremos un vistazo al HMI de este nuevo servicio, mucho más simple ya que se trata de un servicio mucho más automático, con mucha menos intervención humana (figura 49):



(figura 49)

Como vemos, esta vez solo tenemos 2 botones, el botón que nos resetea la localización, y el botón start.

Con el primero realizamos exactamente la misma acción que en el programa navegador, esto es, ponemos las localización del robot, la cual tiene el mismo aspecto en los campos de texto que en el programa navegador, en su valor inicial (0,0,90).

Con el botón start damos comienzo al reconocimiento del entorno, sin más intervención por parte del usuario.

HMI Designer.

Esta es la parte del código que da forma al HMI.

Como podemos ver a continuación, se trata del mismo código que en el Navegador con todas las partes que ni hacían referencia al botón start, el reset localization y los campos de texto de la localización suprimidos.

```
namespace explorador
{
    partial class HMI
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
```

```
        private System.ComponentModel.IContainer components =
null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources
should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not
modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.buttonResetOdo = new
System.Windows.Forms.Button();
            this.valx = new System.Windows.Forms.Label();
            this.valy = new System.Windows.Forms.Label();
            this.theta = new System.Windows.Forms.Label();
            this.xval = new System.Windows.Forms.Label();
            this.yval = new System.Windows.Forms.Label();
            this.thetaval = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Location = new
System.Drawing.Point(12, 65);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(255,
64);
            this.button1.TabIndex = 0;
```

```
            this.button1.Text = "Start";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new
System.EventHandler(this.button1_Click);
            //
            // buttonResetOdo
            //
            this.buttonResetOdo.Location = new
System.Drawing.Point(152, 12);
            this.buttonResetOdo.Name = "buttonResetOdo";
            this.buttonResetOdo.Size = new
System.Drawing.Size(115, 23);
            this.buttonResetOdo.TabIndex = 20;
            this.buttonResetOdo.Text = "Reset Localization";
            this.buttonResetOdo.UseVisualStyleBackColor =
true;
            this.buttonResetOdo.Click += new
System.EventHandler(this.buttonResetOdo_Click);
            //
            // valx
            //
            this.valx.AutoSize = true;
            this.valx.Location = new System.Drawing.Point(9,
9);
            this.valx.Name = "valx";
            this.valx.Size = new System.Drawing.Size(14, 13);
            this.valx.TabIndex = 21;
            this.valx.Text = "X";
            //
            // valy
            //
            this.valy.AutoSize = true;
            this.valy.Location = new System.Drawing.Point(9,
22);
            this.valy.Name = "valy";
            this.valy.Size = new System.Drawing.Size(14, 13);
            this.valy.TabIndex = 22;
            this.valy.Text = "Y";
            //
            // theta
            //
            this.theta.AutoSize = true;
            this.theta.Location = new System.Drawing.Point(9,
35);
            this.theta.Name = "theta";
```

```
            this.theta.Size = new System.Drawing.Size(35, 13);
            this.theta.TabIndex = 23;
            this.theta.Text = "Theta";
            //
            // xval
            //
            this.xval.AutoSize = true;
            this.xval.Font = new
System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.xval.Location = new System.Drawing.Point(60,
9);
            this.xval.Name = "xval";
            this.xval.Size = new System.Drawing.Size(39, 13);
            this.xval.TabIndex = 24;
            this.xval.Text = "0.000";
            //
            // yval
            //
            this.yval.AutoSize = true;
            this.yval.Font = new
System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.yval.Location = new System.Drawing.Point(60,
22);
            this.yval.Name = "yval";
            this.yval.Size = new System.Drawing.Size(39, 13);
            this.yval.TabIndex = 25;
            this.yval.Text = "0.000";
            //
            // thetaval
            //
            this.thetaval.AutoSize = true;
            this.thetaval.Font = new
System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.thetaval.Location = new
System.Drawing.Point(60, 35);
            this.thetaval.Name = "thetaval";
            this.thetaval.Size = new System.Drawing.Size(39,
13);
            this.thetaval.TabIndex = 26;
```

```
            this.thetaval.Text = "0.000";
            //
            // HMI
            //
            this.AutoScaleDimensions = new
System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(281,
144);
            this.Controls.Add(this.thetaval);
            this.Controls.Add(this.yval);
            this.Controls.Add(this.xval);
            this.Controls.Add(this.theta);
            this.Controls.Add(this.valy);
            this.Controls.Add(this.valx);
            this.Controls.Add(this.buttonResetOdo);
            this.Controls.Add(this.button1);
            this.Name = "HMI";
            this.Text = "HMI";
            this.FormClosed += new
System.Windows.Forms.FormClosedEventHandler(this.HMI_FormClose
d);
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Button button1;
        public System.Windows.Forms.Button buttonResetOdo;
        private System.Windows.Forms.Label valx;
        private System.Windows.Forms.Label valy;
        private System.Windows.Forms.Label theta;
        private System.Windows.Forms.Label xval;
        private System.Windows.Forms.Label yval;
        private System.Windows.Forms.Label thetaval;
    }
}
```

HMI.cs

De nuevo nos encontramos en la misma situación que en el apartado anterior, todo lo que no hacía referencia a start o a la localización en si ha sido suprimido.

Como vemos, start sigue activando el handler "StartMisssion()".

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace explorador
{
    public partial class HMI : Form
    {

        //Member
        HMIPort _hmiPort;

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="port"></param>
        public HMI(HMIPort port)
        {
            this._hmiPort = port;

            InitializeComponent();
        }

        /// <summary>
        /// Send new mission when the button1 is clicked
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button1_Click(object sender, EventArgs e)
        {
            this._hmiPort.Post(new StartMission());
        }
```

```
        /// <summary>
        /// Display a text box
        /// </summary>
        /// <param name="s">String to display in
textBox</param>
        public void LogInfo(string s)
        {
            MessageBox.Show(s);
        }

        /// <summary>
        /// Form closed, drop the service
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void HMI_FormClosed(object sender,
FormClosedEventArgs e)
        {
            _hmiPort.Post(new DropFromGui());
        }

        public void UpdateOdo(double X, double Y, double
Theta)
        {
            xval.Text = X.ToString("0.000");
            yval.Text = Y.ToString("0.000");
            thetaval.Text = Theta.ToString("0.000");
        }

        private void buttonResetOdo_Click(object sender,
EventArgs e)
        {
            this._hmiPort.Post(new ResetLocalization());
        }

    }
}
```

3.3.5 AssemblyInfo.cs

Como dijimos en el navegador, AssemblyInfo.cs se crea automáticamente al crear el servicio:

```
using System.Reflection;
using dss = Microsoft.Dss.Core.Attributes;
using interop = System.Runtime.InteropServices;

[assembly:
dss.ServiceDeclaration(dss.DssServiceDeclaration.ServiceBehavi
or)]
[assembly: interop.ComVisible(false)]
[assembly: AssemblyTitle("explorador")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyProduct("explorador")]
[assembly: AssemblyCopyright("")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("1.0.0.0")]
```

3.3.6 ExploradorTypes.cs

En este apartado se reitera que prácticamente el código utilizado en el navegador puede ser reutilizado, pero esta vez el volumen de cosas que se han suprimido es mayor. Muchos handlers que antes eran necesarios desaparecen, junto con todos los que modelaban las funciones de los botones suprimidos en el HMI:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using W3C.Soap;
using laser =
Robosoft.RobuBOX.Generic.Devices.LaserSensor.Proxy;
using explorador = explorador;
```

```
namespace explorador
{
      public sealed class Contract
      {
            [DataMember]
            public const string Identifier =
"http://schemas.tempuri.org/2014/05/explorador.html";
      }

    [DataContract()]
    public class exploradorState
    {

        public double vel;

        public double ang;

        public double x;

        public double y;

        public double theta;

        public int contador;

    }

    [ServicePort]
    public class exploradorOperations :
PortSet<DsspDefaultLookup, DsspDefaultDrop, Get, Replace,
Update, Subscribe>
    {
    }

    /// <summary>
    /// Port to communicate between the service and its HMI
    /// </summary>
    public class HMIPort : PortSet<DsspDefaultLookup,
DsspDefaultDrop, StartMission, ResetLocalization, DropFromGui>
    {
    }

    //generic class operation
    public class Operation
    {
```

```
    }

    /// <summary>
    /// Provides Read-Access to the  state.
    /// </summary>
    public class Get : Get<GetRequestType,
PortSet<exploradorState, Fault>>
    {
    }


    public class Subscribe : Subscribe<SubscribeRequestType,
PortSet<SubscribeResponseType, Fault>>
    {
    }

    /// <summary>
    /// Used to replace state
    /// </summary>
    public class Replace : Replace<exploradorState,
PortSet<DefaultReplaceResponseType, Fault>>
    {
    }

    /// <summary>
    /// Used to update state
    /// </summary>
    public class Update : Update<exploradorState,
PortSet<DefaultReplaceResponseType, Fault>>
    {
    }

    /// <summary>
    /// Internal message from HMI to send mission when
clicking on start button
    /// </summary>
    public class StartMission
    {
    }

    public class DropFromGui
    {
    }

    public class ResetLocalization : Operation
```

```
    {
    }

}
```

Los usings que usaremos son exactamente los mismos.

Se ha suprimido la lista que se venía creando anteriormente de coordenadas al no ser utilizada en este servicio.

Igualmente, muchas de las variables de control utilizadas para realizar la navegación ya no son utilizadas, por lo que se han suprimidas.

El "state" por lo tanto queda ahora así:

```
    public class exploradorState
    {

        public double vel;

        public double ang;

        public double x;

        public double y;

        public double theta;

        public int contador;

    }
```

Vel y ang siguen siendo necesarias para que el robot pueda moverse, ya que son quienes pasan la informacion en cada momento de la velocidad lineal y angular que debe llevar en cada momento el robot.

Igualmente, x, y y theta son quienes nos daban la información obtenida por la odometría, por lo que son incluso mas fundamentales en este programa que en el navegador.

Si nos fijamos en la definición del HMI notamos que todos los handlers de los botones ausentes en este programa (up, down, delete… etc) han sido eliminados, y solo quedan el de StartMission, ResetLocalization y el que cierra el programa, DropFromGui.

```
    public class HMIPort : PortSet<DsspDefaultLookup,
DsspDefaultDrop, StartMission, ResetLocalization, DropFromGui>
```

3.3.7 Explorador.cs

Por último, nos encontramos de nuevo con el programa principal, el explorador.cs.

En esta ocasión las modificaciones con respecto al programa navegador es donde más se acusan, y esque es aquí donde realmente se define el comportamiento del programa:

```
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Xml;
using explorador = explorador;
using System.Security.Permissions;

using Microsoft.Ccr.Adapters.WinForms;
using System.Windows.Forms;
```

```
using types = Robosoft.RobuBOX.Core.Types.Proxy;

using ds = Microsoft.Dss.Services.Directory;
using submgr = Microsoft.Dss.Services.SubscriptionManager;

using laser =
Robosoft.RobuBOX.Generic.Devices.LaserSensor.Proxy;

using drive =
Robosoft.RobuBOX.Generic.Devices.DifferentialDrive.Proxy;

using loc =
Robosoft.RobuBOX.Generic.Processing.Signal.Localization.Proxy;

namespace explorador
{
    [Contract(Contract.Identifier)]
    [DisplayName("explorador")]
    [Description("explorador service (no description
provided)")]
    class exploradorService : DsspServiceBase
    {

        private exploradorState _state = new
exploradorState();

        [ServicePort("/explorador", AllowMultipleInstances =
false)]
        private exploradorOperations _mainPort = new
exploradorOperations();

        [Partner("differentialdrive", Contract =
drive.Contract.Identifier, CreationPolicy =
PartnerCreationPolicy.UseExisting)]
        drive.DifferentialDriveOperations _drivePort = new
drive.DifferentialDriveOperations();

        drive.DifferentialDriveState _driveState;

        private laser.LaserSensorOperations _laserPort;
        private laser.LaserSensorOperations _laserNotifyPort =
new laser.LaserSensorOperations();

        private loc.LocalizationOperations _locPort;
```

```
        private loc.LocalizationOperations _locNotifyPort =
new loc.LocalizationOperations();

        //internal port for HMI
        private HMIPort _hmiPort = new HMIPort();

        //Variables
        HMI _mainHMI;

        Port<bool> _periodicHandlerPort = new Port<bool>();

        string _laserSubMgr = string.Empty;
        string _laserSubscriber = string.Empty;

        string _localizationSubMgr = string.Empty;
        string _localizationSubscriber = string.Empty;

        public exploradorService(DsspServiceCreationPort
creationPort)
            : base(creationPort)
        {
        }

        protected override void Start()
        {

            base.Start();

            MainPortInterleave.CombineWith(
                Arbiter.Interleave(
                    new TeardownReceiverGroup(),
                    new ExclusiveReceiverGroup(

Arbiter.ReceiveWithIterator<bool>(true, _periodicHandlerPort,
PeriodicHandler),
                        Arbiter.Receive<laser.Update>(true,
_laserNotifyPort, UpdateLaserHandler),
                    //Arbiter.Receive<loc.Update>(true,
_locNotifyPort, LocalizationUpdateHandler),
                        Arbiter.Receive<DropFromGui>(true,
_hmiPort, DropFromGuiHandler)
                        ),
                    new ConcurrentReceiverGroup(
                        Arbiter.Receive<StartMission>(true,
_hmiPort, StartHandler),
```

```
Arbiter.Receive<ResetLocalization>(true, _hmiPort,
ResetLocalizationHandler)
                ));

            //launch Winform
            WinFormsServicePort.Post(new
RunForm(RunMainWindow));

            string texto = "X Y alpha";
            System.IO.StreamWriter sw = new
System.IO.StreamWriter(@"C:\datos\datos.txt", false);
            sw.WriteLine(texto);
            sw.Close();

            string texto2 = "Vlinear Vangular";
            System.IO.StreamWriter sw2 = new
System.IO.StreamWriter(@"C:\datosv\datosv.txt", false);
            sw2.WriteLine(texto2);
            sw2.Close();

            string texto3 = "laser central";
            System.IO.StreamWriter sw3 = new
System.IO.StreamWriter(@"C:\datosl\datosl.txt", false);
            sw3.WriteLine(texto3);
            sw3.Close();

            SpawnIterator(ConnectTodriveHandler);
            SpawnIterator(ConnectToLaserHandler);
            SpawnIterator(ConnectToLocalizationHandler);

        }
        IEnumerator<ITask> ConnectTodriveHandler()
        {
            bool error = false;
            //Informe to which sercice to connect
            //Change "drive" by the alias of the service to
connect
            ServiceInfoType info = new
ServiceInfoType(drive.Contract.Identifier);
            //Declare a port to the node/machine where the
service to connect is running, giving URI
            ds.DirectoryPort remotePort =
ServiceForwarder<ds.DirectoryPort>(new Uri("http://" +
"robuLAB10" + ":50000/directory"));
```

```
            //Declare the request
            ds.Query query = new ds.Query(new
ds.QueryRequestType(info));
            //post
            remotePort.Post(query);

            //listen for a response Display in a textbox the
sentence in red
            yield return Arbiter.Choice<ds.QueryResponseType,
W3C.Soap.Fault>(query.ResponsePort,
                delegate(ds.QueryResponseType response)
                {
                    //instanciate the service port
                    _drivePort =
ServiceForwarder<drive.DifferentialDriveOperations>(response.R
ecordList[0].Service);

                },
                delegate(W3C.Soap.Fault fault)
                {
                    error = true;
                });

            if (!error)
            {
                yield return
Arbiter.Choice<drive.DifferentialDriveState, W3C.Soap.Fault>(
                _drivePort.Get(),
                delegate(drive.DifferentialDriveState
response)
                {
                    _driveState = response;
                    WinFormsServicePort.Post(new
FormInvoke(delegate()
                    {
                        _mainHMI.LogInfo("connected to
DifferentilDrive service");
                    }));
                },
                delegate(W3C.Soap.Fault fault)
                {
                    LogError("Failed to get DifferentialDrive
State");
                });
            }
```

```
            yield break;
        }

        IEnumerator<ITask> ConnectToLaserHandler()
        {
            bool error2 = false;
            ServiceInfoType info = new
ServiceInfoType(laser.Contract.Identifier);
            //Change "laser" by the alias of the service to
connect
            info = new
ServiceInfoType(laser.Contract.Identifier);
            //Declare a port to the node/machine where the
service to connect is running, giving URI
            ds.DirectoryPort remotePort2 =
ServiceForwarder<ds.DirectoryPort>(new Uri("http://" +
"robuLAB10" + ":50000/directory"));
            //Declare request
            ds.Query query2 = new ds.Query(new
ds.QueryRequestType(info));
            //post
            remotePort2.Post(query2);

            //listen for a response
            yield return Arbiter.Choice<ds.QueryResponseType,
W3C.Soap.Fault>(query2.ResponsePort,
                delegate(ds.QueryResponseType response)
                {
                    error2 = false;
                    //instanciate services port: receive the
two intance of laser
                    _laserPort =
ServiceForwarder<laser.LaserSensorOperations>(response.RecordL
ist[0].Service);
                },
                delegate(W3C.Soap.Fault fault)
                {
                    error2 = true;
                    _laserPort = null;
                    LogInfo(LogGroups.Console, "ERROR
connecting to Laser service");
                });
            if (error2 == false)
            {
                //Subscribe with the odometry
```

```
                yield return
Arbiter.Choice<SubscribeResponseType, W3C.Soap.Fault>(
                    _laserPort.Subscribe(_laserNotifyPort),
                    delegate(SubscribeResponseType response)
                    {
                        //Make he link between the message
receive and the handler
                        LogInfo("subscribed to Laser
service...");
                        _laserSubMgr =
response.SubscriptionManager;
                        _laserSubscriber =
response.Subscriber;


Activate(Arbiter.Receive<laser.Update>(false,
_laserNotifyPort, UpdateLaserHandler));

                        WinFormsServicePort.Post(new
FormInvoke(delegate()
                        {
                            _mainHMI.LogInfo("connected to
Laser service");
                        }));
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogInfo("Failed to subscribe to
Laser...");

                    });
            }
        }

        IEnumerator<ITask> ConnectToLocalizationHandler()
        {
            bool error = false;
            //Change "loc" by the alias of the service to
connect
            ServiceInfoType info = new
ServiceInfoType(loc.Contract.Identifier);
            //Declare a port to the node/machine where the
service to connect is running, giving URI
```

```
            ds.DirectoryPort remotePort =
ServiceForwarder<ds.DirectoryPort>(new Uri("http://" +
"robuLAB10" + ":50000/directory"));
            //Declare the request
            ds.Query query = new ds.Query(new
ds.QueryRequestType(info));
            //post
            remotePort.Post(query);
            //listen for a response
            yield return Arbiter.Choice<ds.QueryResponseType,
W3C.Soap.Fault>(query.ResponsePort,
                delegate(ds.QueryResponseType response)
                {
                    //to be alert when odometry is updated
                    _locPort =
ServiceForwarder<loc.LocalizationOperations>(
                        response.RecordList[0].Service);
                },
                delegate(W3C.Soap.Fault fault)
                {
                    error = true;
                    _locPort = null;
                });

            if (!error)
            {
                SubscribeRequestType request = new
SubscribeRequestType();
                request.Subscriber = "http://" + "robuLAB10" +
":50000/directory";

                //Subscribe with the localization
                yield return
Arbiter.Choice<SubscribeResponseType, W3C.Soap.Fault>(
                    _locPort.Subscribe(_locNotifyPort),
                    delegate(SubscribeResponseType response)
                    {
                        _localizationSubMgr =
response.SubscriptionManager;
                        _localizationSubscriber =
response.Subscriber;


Activate(Arbiter.Receive<loc.Update>(true, _locNotifyPort,
LocalizationUpdateHandler));
```

```
                        WinFormsServicePort.Post(new
FormInvoke(delegate()
                        {
                            _mainHMI.buttonResetOdo.Enabled =
true;
                        }));
                        WinFormsServicePort.Post(new
FormInvoke(delegate()
                        {
                            _mainHMI.LogInfo("connected to
Localization service");
                        }));
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogInfo("Failed to subscribe to
Localization...");
                    });
            }
        }

        #region drop

        /// <summary>
        /// Drop message sent from gui
        /// </summary>
        /// <param name="d"></param>
        void DropFromGuiHandler(DropFromGui d)
        {
            _mainPort.Post(new DsspDefaultDrop());
        }

        #endregion

        public void UpdateLaserHandler(laser.Update update)
        {
            double xrobot = -_state.y;
            double yrobot = _state.x;
            int estado = 1;

            System.IO.StreamWriter sw = new
System.IO.StreamWriter(@"C:\datosl\datosl.txt", true);
            for (int i = 0; i <= 170; i++)
            {
```

```
                sw.WriteLine(update.Body.Echos[i].Distance + "
" + update.Body.Echos[i].Angle + " " + i);
            }
            sw.Close();

            System.IO.StreamWriter sw2 = new
System.IO.StreamWriter(@"C:\datos\datos.txt", true);
            sw2.WriteLine(_state.x + " " + _state.y + " " +
_state.theta + " " + update.Body.TimeStamp.Hour + " " +
update.Body.TimeStamp.Minute + " " +
update.Body.TimeStamp.Second + " " +
update.Body.TimeStamp.Millisecond);
            sw2.Close();

            for (int j = 0; j <= update.Body.Echos.Count - 85;
j++)
            {
                types.Point2D pointCartesian = new
types.Point2D();
                pointCartesian.X =
update.Body.Echos[j].Distance *
Math.Cos(update.Body.Echos[j].Angle);
                pointCartesian.Y =
update.Body.Echos[j].Distance *
Math.Sin(update.Body.Echos[j].Angle);

                if (pointCartesian.Y < 0.25 &&
pointCartesian.Y > -0.25 && pointCartesian.X > 0 &&
pointCartesian.X <= 0.7)

                    estado = 3;

                else if (_state.contador == 1)
                {
                    if (pointCartesian.Y < 0.25 &&
pointCartesian.Y > -0.25 && pointCartesian.X > 0.7 &&
pointCartesian.X <= 1.1)
                    {
                        estado = 2;
                        _state.contador = 2;
                    }

                }
                else if (_state.contador == 2)
                {
```

```
                    if (pointCartesian.Y < 0 &&
pointCartesian.Y > -1 && pointCartesian.X > 0.0 &&
pointCartesian.X <= 1.1)
                    {
                        estado = 2;
                        _state.contador = 2;
                    }

                }

            }

            if (estado == 2)
            {
                _state.vel = 0.2;
                _state.ang = 0.2;
                _state.contador = 2;
            }
            else if (estado == 3)
            {
                _state.vel = 0.0;
                _state.ang = 0.2;
                _state.contador = 1;
            }
            else if (update.Body.Echos[85].Distance < 0.5)
            {

                _state.vel = 0.2;
                _state.ang = 0.0;
                _state.contador = 1;
            }

            else
            {
                _state.vel = 0.2;
                _state.ang = -0.2;
                _state.contador = 1;
            }



            System.IO.StreamWriter sw3 = new
System.IO.StreamWriter(@"C:\datosv\datosv.txt", true);
            sw3.WriteLine(_state.vel + " " + _state.ang);
```

```
            sw3.Close();


            /* pruebas
            if (_state.estado == 1)
            {
                _state.vel = 0.2;
                _state.ang = 0;
            }
            else
            {
                _state.vel = 0;
                _state.ang = 0;
            }
            */

        }

        //Function to instatiate the HMI window
        HMI RunMainWindow()
        {
            _mainHMI = new HMI(_hmiPort);
            return _mainHMI;
        }

        IEnumerator<ITask> PeriodicHandler(bool b)
        {
            drive.DriveRequest request = new
drive.DriveRequest();

            request.LinearSpeed = _state.vel;
            request.AngularSpeed = _state.ang;


            yield return
Arbiter.Choice<DefaultSubmitResponseType, W3C.Soap.Fault>(
                 _drivePort.Drive(request),
                 delegate(DefaultSubmitResponseType response)
                 {
                 },
                 delegate(W3C.Soap.Fault fault)
                 {
                 });
```

```
            yield return Arbiter.Receive(false,
TimeoutPort(80), delegate(DateTime d) { });

            _periodicHandlerPort.Post(true);

            yield break;
        }

        #region Dss handlers

        /// <summary>
        /// Get Handler
        /// </summary>
        /// <param name="get"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Concurrent)]
        public virtual IEnumerator<ITask> GetHandler(Get get)
        {
            get.ResponsePort.Post(_state);
            yield break;
        }

        /// <summary>
        /// Replace Handler
        /// </summary>
        /// <param name="replace"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Exclusive)]
        public virtual IEnumerator<ITask>
ReplaceHandler(Replace replace)
        {
            _state = replace.Body;

replace.ResponsePort.Post(DefaultReplaceResponseType.Instance)
;
            yield break;
        }

        /// <summary>
        /// Drop Handler
        /// </summary>
        /// <param name="replace"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Teardown)]
```

```
        public virtual IEnumerator<ITask>
DropHandler(DsspDefaultDrop drop)
        {
            DirectoryDelete();
            Shutdown();

drop.ResponsePort.Post(DefaultDropResponseType.Instance);

            if (_laserSubMgr != string.Empty)
            {
                //unsubscribe to laser
                submgr.DeleteSubscription delete = new
submgr.DeleteSubscription();
                delete.Body = new
submgr.DeleteSubscriptionMessage(_laserSubscriber);

ServiceForwarder<submgr.SubscriptionManagerPort>(_laserSubMgr)
.Post(delete);
                yield return
Arbiter.Choice<DefaultDeleteResponseType, W3C.Soap.Fault>(
                    delete.ResponsePort,
                    delegate(DefaultDeleteResponseType
response)
                    {
                        _laserSubMgr = string.Empty;
                        _laserSubscriber = string.Empty;
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogError("Error deleting subscription
to laser");
                    });
            }
            if (_localizationSubMgr != null)
            {
                //unsubscribe to localization
                submgr.DeleteSubscription delete = new
submgr.DeleteSubscription();
                delete.Body = new
submgr.DeleteSubscriptionMessage(_localizationSubscriber);

ServiceForwarder<submgr.SubscriptionManagerPort>(_localization
SubMgr).Post(delete);
                yield return
Arbiter.Choice<DefaultDeleteResponseType, W3C.Soap.Fault>(
```

```
                    delete.ResponsePort,
                    delegate(DefaultDeleteResponseType
response)
                    {
                        _localizationSubMgr = string.Empty;
                        _localizationSubscriber =
string.Empty;
                    },
                    delegate(W3C.Soap.Fault fault)
                    {
                        LogError("Error deleting subscription
to localization");
                    });
            }

            yield break;
        }

        #endregion

        #region StartMission, reset and update loc

        /// <summary>
        /// Reset odometry
        /// </summary>
        /// <param name="r"></param>
        void ResetLocalizationHandler(ResetLocalization r)
        {
            loc.Replace reset = new loc.Replace();
            reset.Body.X = 0;
            reset.Body.Y = 0;
            reset.Body.Theta = 0;
            this._locPort.Post(reset);

            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
                _mainHMI.UpdateOdo(0, 0, 90);
            }));

        }

        /// <summary>
        /// Occure when odometry is updated
        /// </summary>
        /// <param name="up">Odometry</param>
```

```
        void LocalizationUpdateHandler(loc.Update up)
        {
            _state.y = up.Body.X;
            _state.x = - up.Body.Y;
            _state.theta = up.Body.Theta * (180 / Math.PI) +
90;

            WinFormsServicePort.Post(new FormInvoke(delegate()
            {
                _mainHMI.UpdateOdo(_state.x, _state.y,
_state.theta);
            }));
        }

        /// <summary>
        /// User clicks on HMI: the next mission is sent send
next mission
        /// </summary>
        /// <param name="s"></param>
        void StartHandler(StartMission s)
        {
            _periodicHandlerPort.Post(true);
        }

        #endregion
    }
}
```

De nuevo, los usings utilizados son los mismos que en el programa navegador, por lo tanto, todas las referencias agregadas también son las mismas que en este mismo.

Posteriormente al definir el estado, vemos que no están definidas la lista de coordenadas, pero si tenemos los puertos del láser y la notificación, y posteriormente el del HMI y el del periodic handler (el que lleva a cabo el movimiento del robot).

A continuación tenemos el servicio explorador. Después de ": base(creationPort)" vemos que no se ha agregado un estado inicial por

defecto como se hizo en el navegador, y es que en este caso ya no se nos hace necesario definirlo, al no haber ningún problema o conflicto en el HMI con la necesidad de tener un estado presente.

Por ello, se define directamente el interleave justo a continuación del "base(start)":

```
base.Start();

            MainPortInterleave.CombineWith(
                Arbiter.Interleave(
                    new TeardownReceiverGroup(),
                    new ExclusiveReceiverGroup(

Arbiter.ReceiveWithIterator<bool>(true, _periodicHandlerPort,
PeriodicHandler),
                        Arbiter.Receive<laser.Update>(true,
_laserNotifyPort, UpdateLaserHandler),
                    //Arbiter.Receive<loc.Update>(true,
_locNotifyPort, LocalizationUpdateHandler),
                        Arbiter.Receive<DropFromGui>(true,
_hmiPort, DropFromGuiHandler)
                        ),
                    new ConcurrentReceiverGroup(
                        Arbiter.Receive<StartMission>(true,
_hmiPort, StartHandler),

Arbiter.Receive<ResetLocalization>(true, _hmiPort,
ResetLocalizationHandler)
                    )));
```

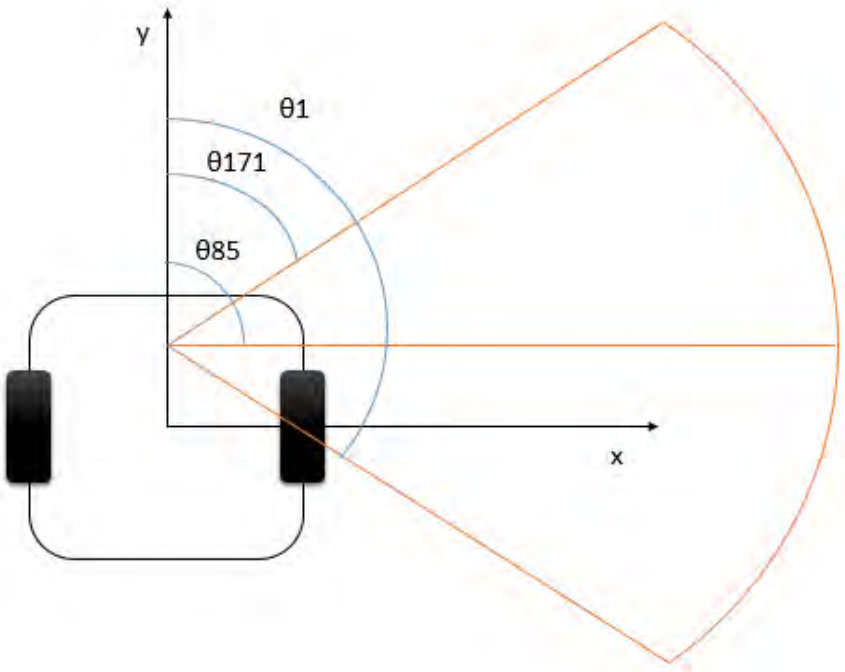
Como se puede apreciar, el interleave se ha visto bastante reducido. Esto se debe a la cantidad de handlers que se han suprimido en este programa. Ahora ya solo tenemos los siguientes (sin contar los propios de DSS)

- Periodic handler
- Laser handler

- Drop from Gui handler
- Start handler
- Reset localization handler

Posteriormente se lanza el HMI y se abren y escriben los primeros datos en todos los ficheros de texto. Aquí encontramos la primera novedad que consiste en la inclusión del fichero datosl.txt, el cual recoge los datos del láser.

La información del láser está contenida en un vector de 682 componentes. La parte recogida en el fichero se corresponde con los valores del 1 al 171, esto es, toda la parte derecha del robot. Para ser más exactos, el primer valor del vector (el primer "echo") forma un ángulo con el eje y de coordenadas de 2,0923879737534 radianes, el 85 de pi/2 radianes, y el 171 por tanto de 1,04314511483962 radianes (figura 50).



(figura 50)

Este es el fichero que recogerá los datos de todos los componentes del vector por tanto, contenidos entre el 1 y el 171.

Su objeto es la posterior reconstrucción del entorno recorrido, el mapeado (SLAM).

Una vez hechas todas las conexiones (procedimiento idéntico al realizado en el navegador), definimos el handler del láser:

Después de pasarle los datos de la odometría, recogemos en datosl los datos del láser mencionados mediante un bucle for:

```
double xrobot = -_state.y;
            double yrobot = _state.x;
            int estado = 1;

            System.IO.StreamWriter sw = new
System.IO.StreamWriter(@"C:\datosl\datosl.txt", true);
            for (int i = 0; i <= 170; i++)
            {
                sw.WriteLine(update.Body.Echos[i].Distance + "
" + update.Body.Echos[i].Angle + " " + i);
            }
            sw.Close();
```

Para acceder a ellos escribimos "Update.body.Echos[posición del elemento en el vector].característica requerida". Las características que necesitamos son distance, que es la distancia del echo, y angle, que será constante, y que es el ángulo que forma con el eje y. recogeremos también la variable i, para ver que elemento del vector estamos recogiendo.

No cerramos el fichero hasta terminar el bucle para ahorrar tiempo de y computo en el algoritmo, factor determinante con la inmensa cantidad de datos a manejar.

De hecho, no se recogerán todos los componentes del láser por este mismo motivo. Si intentáramos hacerlo, la cantidad de datos sería tan alta en cada iteración del Update que no nos daría tiempo si quiera a actualizar las variables del movimiento vel y ang, por lo que el robot jamás llegaría si quiera a moverse.

Registramos ahora los datos de la Odometría, pero en esta ocasión añadimos el tiempo absoluto (horas, minutos, segundos y milisegundos) en el que la medida fue recogida.

Esto se hace para poder sincronizar de forma offline la localización del robot con las imágenes captadas con la cámara, las cuales también recogerán el tiempo absoluto en el que se tomaron.

Los estados que toma el láser para evitar los obstáculos o los límites del entorno se determinan con el mismo procedimiento que en el navegador, solo que para no tener que computar todos los componentes del láser, se cambia el bucle foreach por un for simple en el que suprimimos los últimos 85 componentes del láser:

```
for (int j = 0; j <= update.Body.Echos.Count - 85; j++)
            {
                types.Point2D pointCartesian = new
types.Point2D();
                pointCartesian.X =
update.Body.Echos[j].Distance *
Math.Cos(update.Body.Echos[j].Angle);
                pointCartesian.Y =
update.Body.Echos[j].Distance *
Math.Sin(update.Body.Echos[j].Angle);
```

```
                if (pointCartesian.Y < 0.25 &&
pointCartesian.Y > -0.25 && pointCartesian.X > 0 &&
pointCartesian.X <= 0.7)

                    estado = 3;

                else if (_state.contador == 1)
                {
                    if (pointCartesian.Y < 0.25 &&
pointCartesian.Y > -0.25 && pointCartesian.X > 0.7 &&
pointCartesian.X <= 1.1)
                    {
                        estado = 2;
                        _state.contador = 2;
                    }

                }
                else if (_state.contador == 2)
                {
                    if (pointCartesian.Y < 0 &&
pointCartesian.Y > -1 && pointCartesian.X > 0.0 &&
pointCartesian.X <= 1.1)
                    {
                        estado = 2;
                        _state.contador = 2;
                    }

                }

            }
```

Ahora se define el movimiento del robot.

Su comportamiento por defecto será el de ir buscando la pared que se encuentre a su derecha, para ello ang y vel tienen 0.2 como valor.

Cuando el láser que forma pi/2 radianes con el eje y "(update.Body.Echos[85])" tenga como distancia un valor menor de 0.5 metros, el robot irá en trayectoria recta hacia delante. Esto nos indica que el robot ha encontrado una pared y la está recorriendo. (vel = 0.2, ang = 0).

Los otros dos casos corresponden a los estados 2 y 3 que se utilizan para la evasión de obstáculos.

```
if (estado == 2)
            {
                _state.vel = 0.2;
                _state.ang = 0.2;
                _state.contador = 2;
            }
            else if (estado == 3)
            {
                _state.vel = 0.0;
                _state.ang = 0.2;
                _state.contador = 1;
            }
            else if (update.Body.Echos[85].Distance < 0.5)
            {

                _state.vel = 0.2;
                _state.ang = 0.0;
                _state.contador = 1;
            }

            else
            {
                _state.vel = 0.2;
                _state.ang = -0.2;
                _state.contador = 1;
            }
```

Las velocidades que se dan son bajas para que la captura de imágenes tenga la mejor calidad posible y no sea distorsionada.

Posteriormente se toman los valores de la velocidad igual que en el navegador.

Estos valores se toman para asegurarse que todo ha funcionado correctamente en un post-análisis de la actuación del robot.

El resto del programa es idéntico al navegador, suprimiendo todos los handlers mencionados que no forman parte de este servicio de exploración.

3.4 SLAM

3.4.1 Introducción al SLAM realizado

A continuación, una vez recogidos los datos en el programa explorador, el cual utilizaba el láser para hacer un reconocimiento del entorno, mediante la recogida de las distancias desde el robot a los diferentes objetos o límites, procesamos esa información para realizar una reconstrucción del mapa del entorno explorado.

El SLAM realizado, por tanto, tiene las siguientes características:

- **Offline**. (se realiza SLAM sobre un conjunto de datos que previamente fueron recuperados con algún robot, tanto de la medida de sus sensores como las medidas de odometría (movimiento).
- **Métrico**. (se provee información métrica entre los lugares).
- **Pasivo**. (En los algoritmos de SLAM pasivos, es otra entidad quien se encarga de controlar el robot, mientras que el algoritmo de SLAM es puramente observador).
- **Estático**. (En el caso del SLAM estático se asume que el entorno no cambia con el tiempo).
- **Volumétrico**. (En SLAM volumétrico, el mapa es muestreado a una resolución que permite una reconstrucción fotográfica del entorno).

- **SLAM con un solo robot**. (El SLAM es realizado con un solo robot que realiza solo la medición de todos los datos).
- **Manejo de mucha incertidumbre**. (para los mapas los cuales tienen lugares que se pueden alcanzar de muchas maneras, es necesario que el robot pueda manejar mucha incertidumbre en su posición).

Por lo tanto, el siguiente paso es realizar un pre-procesado de los datos recogidos para prepararlos antes de construir el mapa.

3.4.2 Tratamiento de los datos recogidos

Se han confeccionado hasta este punto 3 archivos de texto de datos, que son datos.txt, datosl.txt y datos.txt, que se corresponden respectivamente con la posición del robot y su orientación, los datos del láser del robot, y su velocidad en cada momento.

Los datos, cuyo código de recogida se encuentra en el programa explorador, se estructuran de la siguiente manera:

Datos contiene 7 columnas en las que cada fila representa una nueva medición, en las cuales figuran:

- Columna 1: coordenada x de la posición del robot con respecto al sistema de coordenadas absoluto definido por la posición inicial y orientación del robot en el instante en el que arranca el programa.
- Columna 2: coordenada y de la posición del robot
- Columna 3: orientación del robot.
- Columna 4: hora absoluta en la que se registraron los datos
- Columna 5: minuto absoluto en el que se registraron los datos
- Columna 6: segundo absoluto en el que se registraron los datos
- Columna 7:milisegundo absoluto en el que se registraron los datos

Datosl, por su parte, contiene los datos del láser, los cuales se estructuran en 3 columnas, que representan cada 171 filas, las mediciones de los 171 primeros echos del láser. Las columnas se corresponden con:

- Columna 1: distancia registrada del echo
- Columna 2: ángulo que forma el echo
- Columna 3: número del echo que se está recogiendo.

Por último, datos tan solo contiene 2 columnas que registran los datos recogidos en cada fila consecutiva.

- Columna 1: velocidad lineal
- Columna 2: velocidad angular

Para pasar estos datos a matlab y tenerlos preparados para la construcción del mapa, se introdujo una parte específicamente dedicada de código en el programa que realiza el mapa, llamado SLAM.m

Esta parte abre los ficheros de texto, los lee, y posteriormente reordena los datos, ya que al leerlos los toma como una única columna.

No todos los ficheros tienen el mismo número de filas, no solo porque el que contiene los datos del láser tenga 171 filas por cada fila de cualquier otro de los dos archivos de datos, sino porque los datos del fichero de posiciones y orientaciones esta recortado con el fin de eliminar datos indeseados.

Cuando se recogen los datos, la forma de proceder es la de hacer un reset a la orientación antes de presionar start y que dé comienzo la exploración, por lo que todos los datos recogidos antes del reset han de ser eliminados, esto es, todos aquellos que se encontraban antes de que aparezca (0,0,90), es decir, la posición y orientación reiniciadas.

Como consecuencia, ahora tenemos que recortar los datos correspondientes a los otros dos ficheros de texto, pero dado que esto representa una ardua tarea de conteo, se propone la solución que podemos apreciar en el código de matlab, en el que a través de una sencilla comparación en las longitudes de ambos ficheros, podemos quedarnos simplemente con los que nos interesan.

El código completo de esta primera parte del programa SLAM.m (el cual se verá al completo en los siguientes apartados) es el siguiente:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          %
%          SLAM            %
%                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%

fid = fopen('POSICION.txt');

posicion = fscanf(fid,'%f');

fid2 = fopen('VELOCIDAD.txt');

velocidad = fscanf(fid2,'%f');

fid3 = fopen('LASER.txt');

laser = fscanf(fid3,'%f');

posicion = [posicion(1:7:length(posicion-6))
posicion(2:7:length(posicion-5)) posicion(3:7:length(posicion-
4)) posicion(4:7:length(posicion-3))
posicion(5:7:length(posicion-2)) posicion(6:7:length(posicion-
1)) posicion(7:7:length(posicion))];
laser = [laser(1:3:length(laser-2)) laser(2:3:length(laser-1))
laser(3:3:length(laser))];
velocidad = [velocidad(1:2:length(velocidad-1))
velocidad(2:2:length(velocidad))];

laser = laser(length(velocidad)*171-
length(posicion)*171:end,:);
```

```
velocidad = velocidad(length(velocidad)-
length(posicion):end,:);
```

Pasamos ahora a explicar el código paso a paso para entender que es lo que se pretende:

Primero abrimos los ficheros con la instrucción fopen, y los leemos con fscan. Cabe añadir que debemos indicar que archivo estamos abriendo con su extensión:

```
fid = fopen('POSICION.txt');

posicion = fscanf(fid,'%f');

fid2 = fopen('VELOCIDAD.txt');

velocidad = fscanf(fid2,'%f');

fid3 = fopen('LASER.txt');

laser = fscanf(fid3,'%f');
```

A continuación reordenamos los datos, ya que estos han quedados estructurados como una única columna al ser leídos por MATLAB.

Esta labor es sencilla, ya que como sabíamos de cuantas columnas disponíamos anteriormente, nos basta con contar a lo largo de esta nueva columna un número de veces igual al número de columnas que teníamos, e indicar que esos elementos conformaran una de las nuevas columnas de los datos que estamos tratando. Repetimos este proceso pero avanzando una posición en esta única columna, un número de veces también igual al número de columnas de las que disponíamos, de forma que cada vez que repetimos el proceso estamos conformando una nueva columna:

```
posicion = [posicion(1:7:length(posicion-6))
posicion(2:7:length(posicion-5)) posicion(3:7:length(posicion-
4)) posicion(4:7:length(posicion-3))
posicion(5:7:length(posicion-2)) posicion(6:7:length(posicion-
1)) posicion(7:7:length(posicion))];

laser = [laser(1:3:length(laser-2)) laser(2:3:length(laser-1))
laser(3:3:length(laser))];

velocidad = [velocidad(1:2:length(velocidad-1))
velocidad(2:2:length(velocidad))];
```

Ahora simplemente comparamos la longitud de los datos con el de posición (que es el que habíamos recortado anteriormente), y nos quedamos con los últimos datos de los datos del láser y velocidad, de tal manera que ahora la longitud (es decir, el número de filas) de todos los datos disponibles sea la misma.

Por supuesto, con el láser, al tener un número de filas 171 veces más grande que el de los otros dos ficheros, debemos realizar la comparación multiplicando el número de filas de la posición por 171 para indicar cuantas filas es necesario que nos quedemos:

```
laser = laser(length(velocidad)*171-
length(posicion)*171:end,:);
velocidad = velocidad(length(velocidad)-
length(posicion):end,:);
```

Con esto tenemos los datos preparados para construir el mapa y la trayctoria del robot.

3.4.3 Elaboración de los algoritmos de matlab

En este apartado se aborda la representación de las posiciones y orientaciones alcanzadas por el robot durante la exploración, así como la construcción del mapa del entorno.

3.4.3.1 Representación de la posición y orientación del robot en el espacio

En esta sección representaremos la posición y orientación que el robot fue alcanzando a lo largo de la exploración, o más adelante, a lo largo de la navegación a uno de los puntos explorados.

Las posiciones se representaran con una serie de puntos unidos por líneas azules, y las orientaciones por unas pequeñas líneas con una longitud de 0.5 uds en la representación, que apuntaran hacia aquello a lo que el robot estuviera orientado en cada momento.

El código completo de esta sección es el siguiente.

```
% posicion del robot
x = posicion(:,1);
y = posicion(:,2);
alpha = posicion(:,3)*pi/180;
z = 1:length(x);

plot3(x,y,z)
hold on

% orientacion del robot
for (i=1:1:length(x))
    plot3([x(i) x(i)+0.5*cos(alpha(i))],[y(i)
y(i)+0.5*sin(alpha(i))],[z(i) z(i)])
end
```

Primero definiremos cuales van a ser las coordenadas x e y alcanzadas en cada instante de tiempo, su orientación Alpha, y una tercera coordenada z que representara el progreso temporal de estas coordenadas alcanzadas.

La matriz posición es donde tenemos los datos de las posiciones y orientaciones alcanzadas ordenadas en sendas columnas.

No tenemos más que definir qué columna representa cada coordenada o ángulo para separar esta matriz en los vectores x, y, Alpha y z.

```
x = posicion(:,1);
y = posicion(:,2);
alpha = posicion(:,3)*pi/180;
z = 1:length(x);
```

Como dijimos, z no es más que un vector que representa la progresión temporal de los puntos alcanzados, es decir, podríamos definirlo como el tiempo, por lo que lo podremos definir como un vector que avanza una unidad con cada nuevo dato.

Pero esta unidad no es el segundo, por lo que la representación que obtengamos no estará expresada en unidades del SI, si no que será el tiempo que el robot tardo en tomar una nueva de datos.

Este tiempo puede ser variable, es decir, puede que no sea el mismo entre un dato y otro, aunque de todos modos, siempre bastante parecido, y dependerá del tiempo que se tardó en computar el nuevo Update como vimos en la información recibida por el láser.

Se podría incluso regular esta pequeña diferencia de tiempo entre datos recogidos, ya que se recogió también el momento absoluto en el que se recogió cada medida, pero no es interesante incluso computacionalmente

ya que lo que se busca no es representar fidedignamente el tiempo en esta coordenada z, sino la progresión temporal como se ha indicado.

Para representar la posición, simplemente dibujamos x e y en función de z, es decir, las posiciones cartesianas del robot en cada instante medido:

```
plot3(x,y,z)
hold on
```

A continuación, para la orientación, debemos recorrer con un bucle for todas y cada una de las orientaciones que el robot alcanzo para representarlas.

Para ello dibujamos un vector con origen en cada una de las posiciones que el robot tomo, y tomando como hipotenusa un segmento con valor 0.5 uds, calculamos el extremo final del vector de orientación como este punto inicial sumado a la proyección de esta hipotenusa en el eje x o y según corresponda, ya que al disponer del ángulo en cada caso (dato recogido de la orientación), bastara por multiplicar el segmento de 0.5 por el coseno o el seno para x e y respectivamente.

```
% orientacion del robot
for (i=1:1:length(x))
    plot3([x(i) x(i)+0.5*cos(alpha(i))],[y(i)
y(i)+0.5*sin(alpha(i))],[z(i) z(i)])
end
```

3.4.3.2 Mapeado del entorno explorado por el robot

En esta sección se creara el mapa del entorno y se propondrán diversas formas de representarlo. Algunas de ellas requieren bastante capacidad computacional debido al enorme número de datos con los que trabajar, aunque también se proponen otras alternativas con precisos resultados con una carga computacional más liviana.

El código completo de esta sección es el siguiente:

```
% slam
px = [];
py = [];
for (i=1:1:171)
    px = [px x+laser(i:171:end-
171+i,1).*cos(alpha+laser(i:171:end-171+i,2))];
    py = [py y+laser(i:171:end-
171+i,1).*sin(alpha+laser(i:171:end-171+i,2))];
end

% slam laser central
% plot3(px(:,85),py(:,85),z,'r')

% slam todo
% for (i=1:1:171)
%     plot3(px(:,i),py(:,i),z,'r')
% end

% slam con saturacion
% k = 1;
% m = true;
% for i=1:length(x)
%     for j=1:171
%         if norm([px(i,j);py(i,j)]-[x(i);y(i)]) > 1.5
%             while m
%                 if norm([px(i-k,j);py(i-k,j)]-[x(i-k);y(i-
%k)]) < 1.5
%                     px(i,j) = px(i-k,j);
%                     py(i,j) = py(i-k,j);
%                     k = 1;
%                     break;
%                 else
```

```
%                     k = k+1;
%                 end
%             end
%         end
%     end
% end
% for (i=1:1:171)
%     plot3(px(:,i),py(:,i),z,'r')
% end
```

Lo primero que hay que hacer para la construcción del mapa es reordenar los datos del láser de nuevo, ya que tenemos 171 filas seguidas con los datos de los echos hasta que encontramos una nueva medición.

Por lo tanto, lo que nos interesa es tener los datos de cada medición en cada fila, y cada echo medido en cada columna.

Paralelamente, a la vez que reordenamos los datos, construimos el mapa.

Primeramente definimos dos vectores vacíos, que será donde introduzcamos los valores que representan el entorno explorado:

```
px = [];
py = [];
```

A continuación recorremos con un bucle for todos los datos que conforman una medición, es decir, los 171 echos.

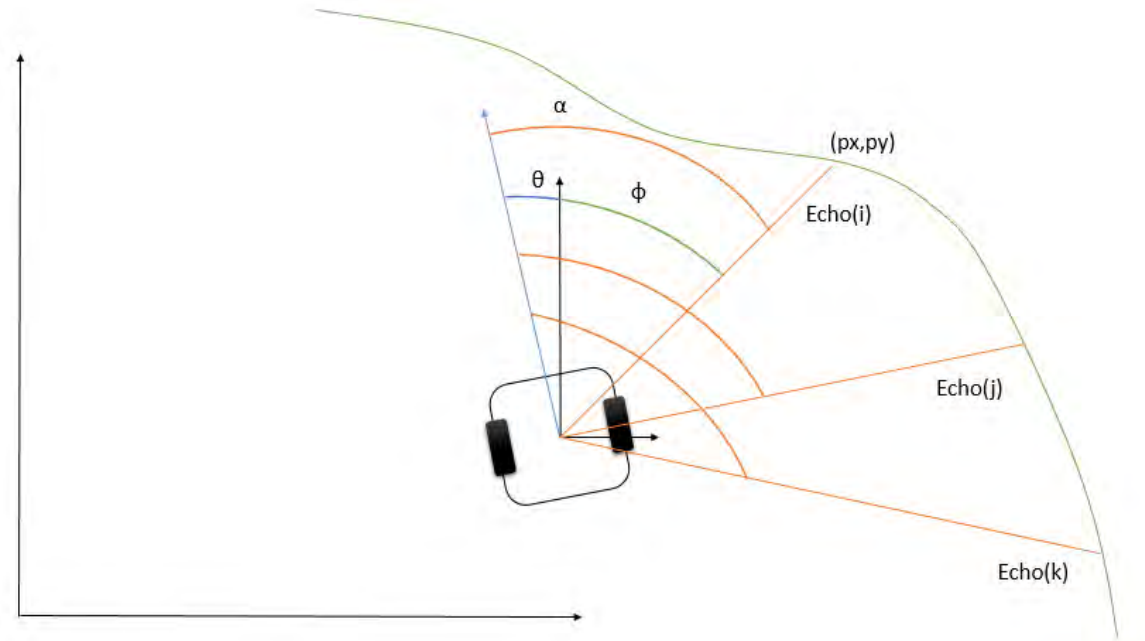
Con esto se pretende calcular en cada bucle el entorno explorado por todas las mediciones de uno de los echos cada vez.

La forma de calcularlo es obteniendo las coordenadas de cada punto en px y en py, de forma separada, para ello guardamos en cada bucle el px y py previo y redimensionamos el mismo con otra columna (px = [px (...)]).

Como estamos teniendo todos las mediciones de un mismo echo cada bucle, cuando hagamos referencia a los datos del láser que queremos seleccionar, deberemos realizar saltos de 171 pasos para evitar el resto de echos, empezando por i, siendo i el bucle actual, y acabando en (end − 171 + i), que hace referencia a la última medición del echo con el que se trabaja.

Ahora bien, la forma de calcular el punto cada vez que define el entorno consiste en, para px y py, sumar a la posición actual en el momento de cada medición del robot la distancia medida por el echo, y multiplicarla por, según sea px o py, el coseno o el seno respectivamente de la suma del ángulo que tiene en ese instante la orientación del robot, y el ángulo del echo con el que se están haciendo los cálculos en cada bucle.

Para ilustrarlo gráficamente, el cálculo pretende lo siguiente (figura 51):



(figura 51)

Podemos apreciar en esta figura los diferentes ángulos presentes para calcular px y py.

Como vemos, nuestro objetivo es obtener el ángulo ϕ, el cual es el resultado de sumar el ángulo θ, que determina la orientación del robot, y el ángulo, α, que determina el ángulo del echo correspondiente.

```
for (i=1:1:171)
    px = [px x+laser(i:171:end-
171+i,1).*cos(alpha+laser(i:171:end-171+i,2))];
    py = [py y+laser(i:171:end-
171+i,1).*sin(alpha+laser(i:171:end-171+i,2))];
end
```

Una vez calculados los puntos que forman el mapa, queda representarlo.

Como ya se ha comentado, debido a la gran cantidad de datos de los que se dispone, podemos optar por diferentes formas de representarlo.

Una de ellas, la más básica pero no por ello menos precisa, consiste en representar simplemente los datos obtenidos por el echo central, es decir, el que forma pi/2 radianes con la parte frontal del robot.

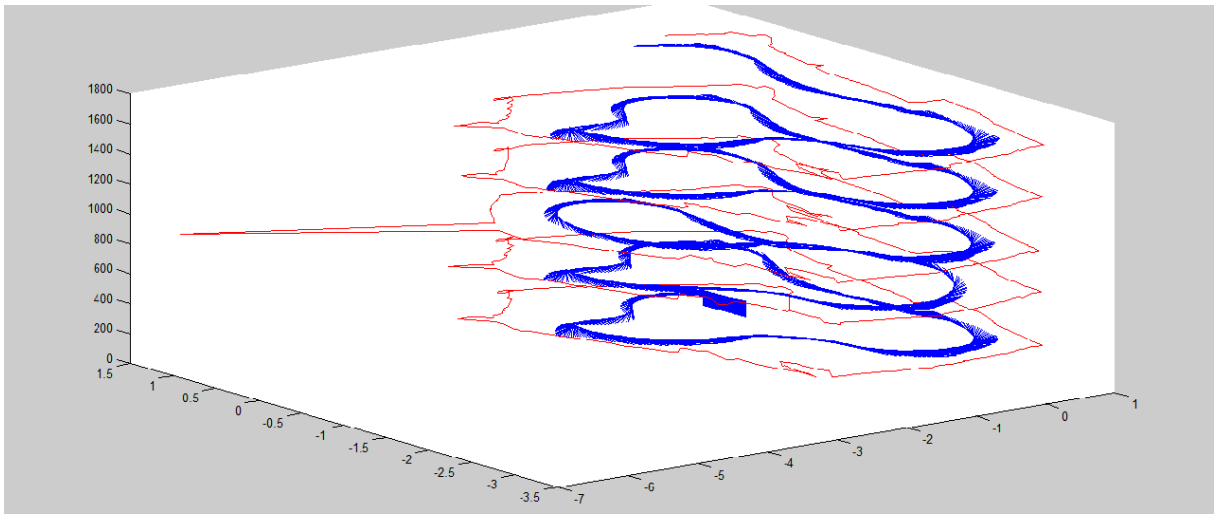
Para hacerlo, simplemente representamos la columna correspondiente a este echo, es decir, la numero 85 (ya que tenemos de 170 echos en las mediciones)

```
% slam laser central
plot3(px(:,85),py(:,85),z,'r')
```

Si lo hacemos obtendremos unos resultados bastante precisos que nos mostraran el entorno de forma precisa.

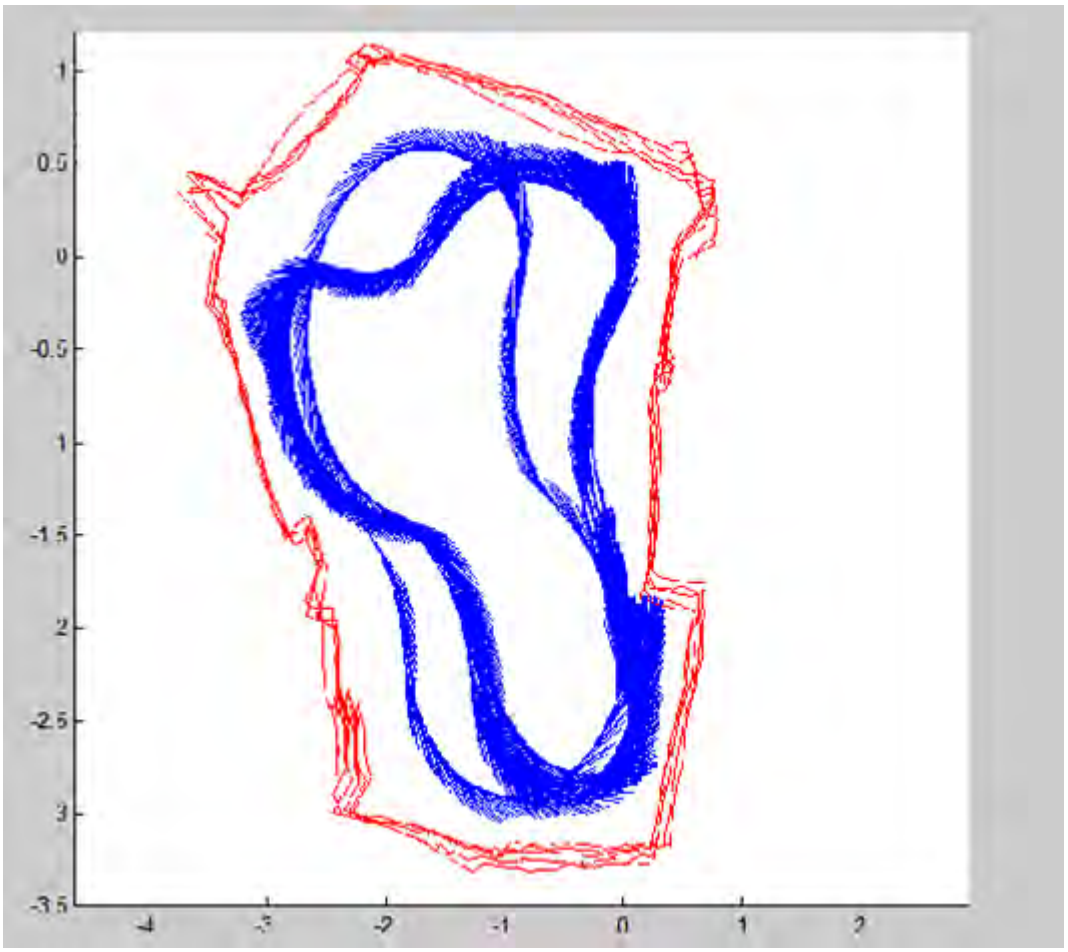
Se realizaron varias pruebas eligiendo como entorno a explorar el descrito en la sección correspondiente con los siguientes resultados:

En la siguiente representación podemos apreciar las posiciones y orientaciones del robot vistas desde una perspectiva que nos permite también visualizar como las fue alcanzando a lo largo del progreso en la exploración, además de poder visualizar el mapa del entorno recorrido (figura 52):



(figura 52)

A continuación, la misma representación vista en planta para poder apreciar el mapa construido (figura 53):



(figura 53)

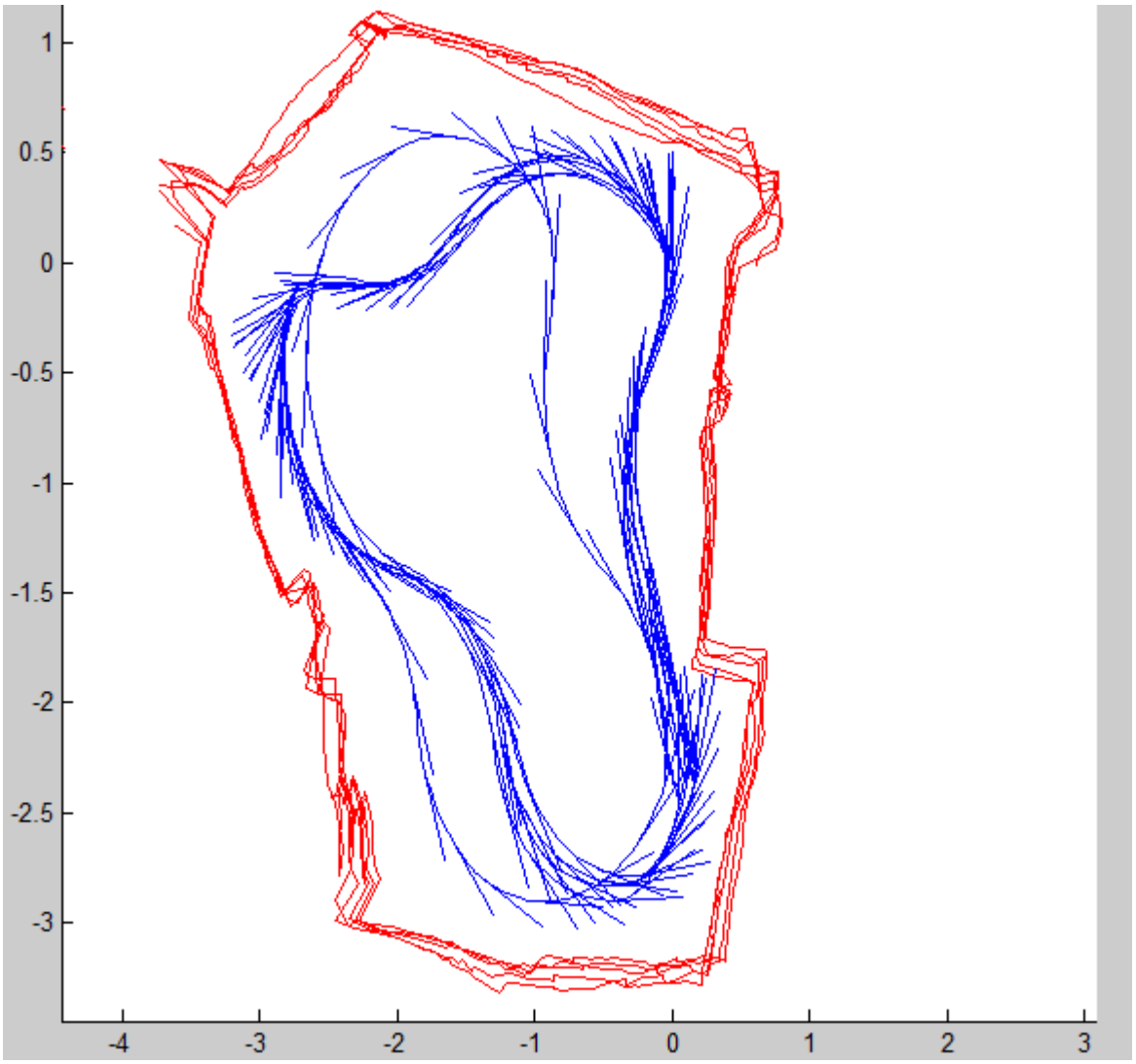
Se pude reducir el número de vectores que representan la orientación para poder visualizarla más claramente de la siguiente forma:

En el código en el que representamos la orientación realizamos el siguiente cambio:

En el bucle for indicamos que no queremos avanzar en los datos de la orientación de 1 en 1, sino que lo haremos de x en x, siendo x un valor deseado por el usuario, por ejemplo, con 10 (figura 54):

```
% orientacion del robot
```

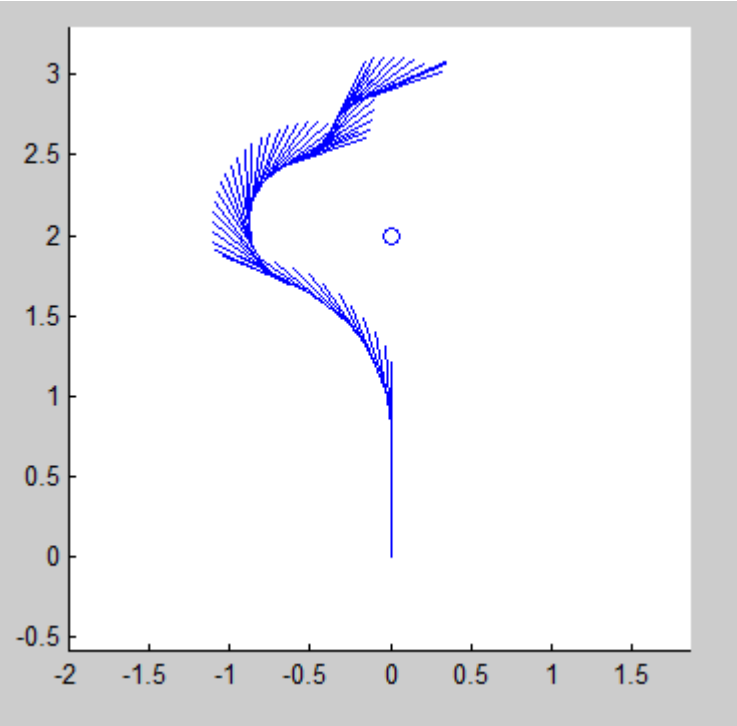
```
for (i=1:10:length(x))
    plot3([x(i) x(i)+0.5*cos(alpha(i))],[y(i)
y(i)+0.5*sin(alpha(i))],[z(i) z(i)])
end
```



(figura 54)

Vemos cómo podemos apreciar más claramente la orientación, no obstante, al haber realizado el robot en su tarea de exploración varias vueltas puede que los vectores se superpongan en la planta y no veamos claramente los detalles.

Para ello, podemos representar una trayectoria más simple del robot, por ejemplo, indicando al programa navegador que queremos movernos a la coordenada (0,3) y poniendo un obstáculo entre medias para que el robot haga un giro y lo corrija (figura 55):



(figura 55). El circulo representa el obstáculo.

Otra alternativa para representar el mapa es la de mostrar todos los echos del láser y no solo el central al mismo tiempo.

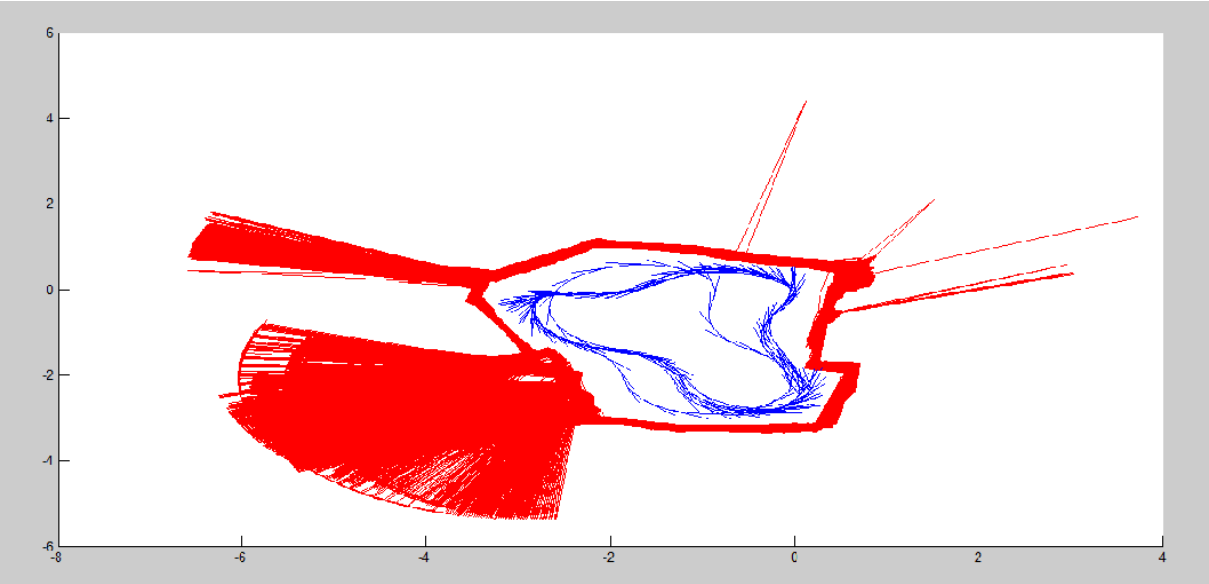
Esta alternativa presenta la ventaja de poseer una representación más robusta y fidedigna del entorno, pero también presenta desventajas. Por un lado, al representar todos los echos, la carga computacional es muy elevada, y por otro lado, al tener tantas mediciones, puede que algunas de ellas nos muestren valores demasiado elevados fruto de zonas con

huecos o similares que llevan a que el echo no encuentre paredes o cuerpos y alcance su valor máximo (4 metros aprox).

El código es tan sencillo como representar todos los puntos del entorno, sin restricciones.

```
% slam todo
for (i=1:1:171)
    plot3(px(:,i),py(:,i),z,'r')
end
```

La representación vista nuevamente de planta es la siguiente (figura 56):



(fugra 56)

Como vemos tenemos tantos datos que muchos se han ido al máximo, y es de aquí de donde surge la última representación del mapeado que se propone.

Esta última representación consiste en saturar los valores a un valor dado por el usuario. Se propone en la exposición del proyecto un valor límite de 1.5 metros.

La saturación se realiza recorriendo con dos bucles todos los valores de px y py, y observando el valor de la distancia que tomo el echo. Para esto realizamos la norma del valor obtenido de las coordenadas del límite encontrado por el láser (px(i,j) y py(i,j) en cada instante) menos la posición (x(i), y(i)) del robot en cada instante.

Esto significa, observar el valor del segmente que va desde el robot hasta la pared observada, es decir, cada echo del láser.

Este valor se compara con un valor límite (1.5 metros), y si es mayor, se asigna al punto de la pared el valor anterior, siempre que este cumpla la condición de ser un echo con un valor inferior a 1.5 metros. Si el echo anterior tampoco lo cumpliese, seguiríamos buscando hacia atrás hasta que encontráramos uno que si lo cumpliese con el bucle while.

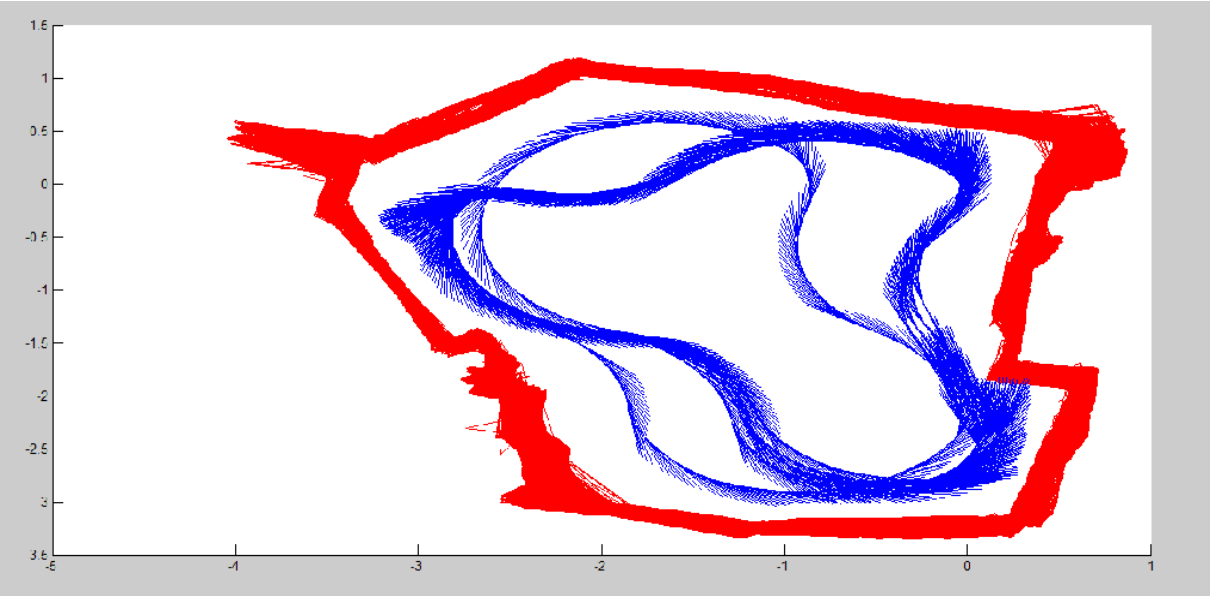
Una vez lo encontramos, reiniciamos la variable k a 1, que es la que va buscando valores hacia atrás en los echos, y damos la instrucción break para salir del bucle.

El resultado es unas nuevas matrices px y py saturadas:

```
slam con saturacion
k = 1;
m = true;
for i=1:length(x)
    for j=1:171
        if norm([px(i,j);py(i,j)]-[x(i);y(i)]) > 1.5
            while m
                if norm([px(i-k,j);py(i-k,j)]-[x(i-k);y(i-k)])
< 1.5
                    px(i,j) = px(i-k,j);
                    py(i,j) = py(i-k,j);
```

```
                    k = 1;
                    break;
                else
                    k = k+1;
                end
            end
        end
    end
end
for (i=1:1:171)
    plot3(px(:,i),py(:,i),z,'r')
end
```

El resultado, el cual es el más preciso y el que mayor grado de exactitud tiene con el entorno explorado es el siguiente (figura 57):

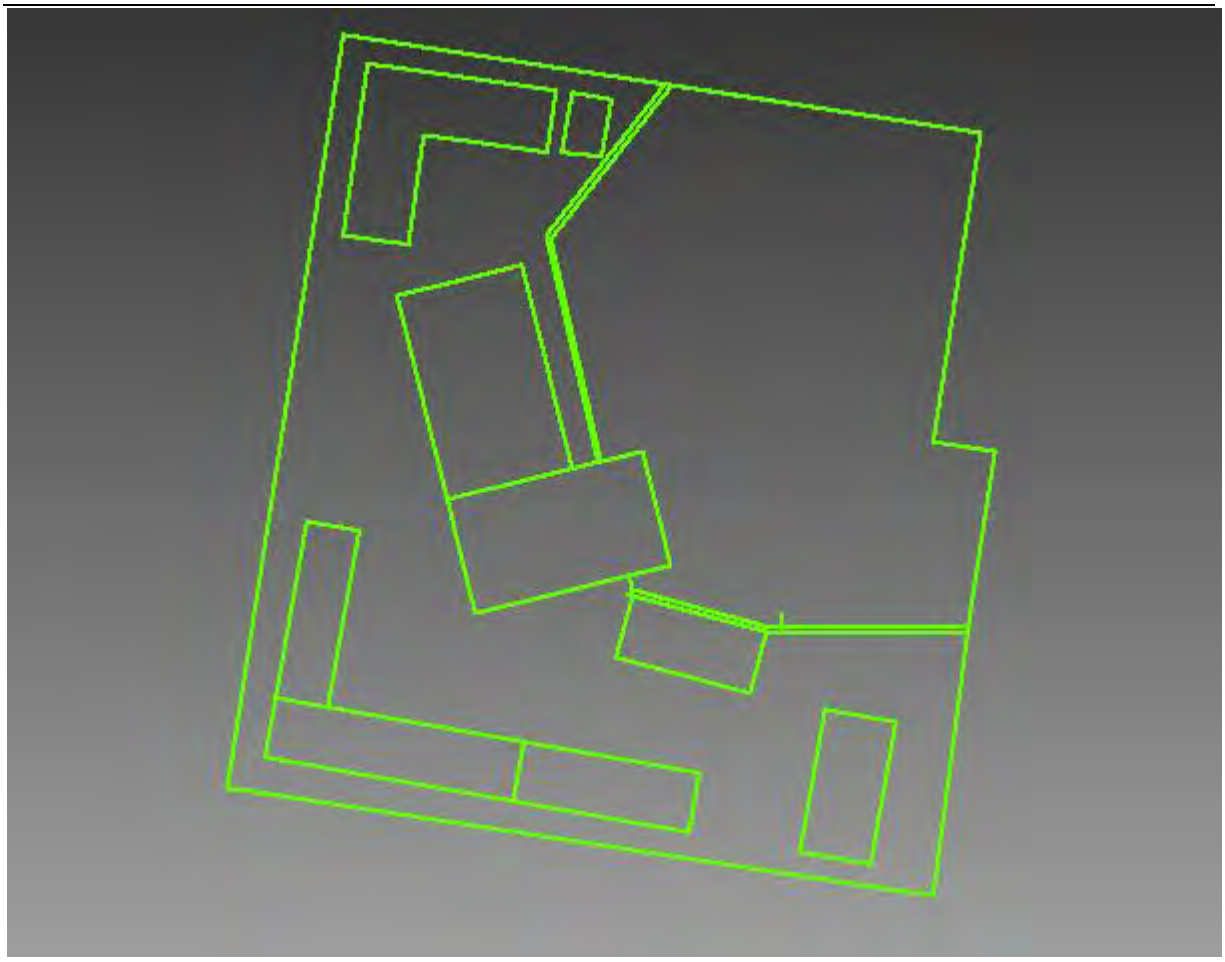


(figura 57)

Para poder ver los resultados más claramente, es decir, como representa este desarrollo de la técnica de SLAM el entorno explorado, vamos a

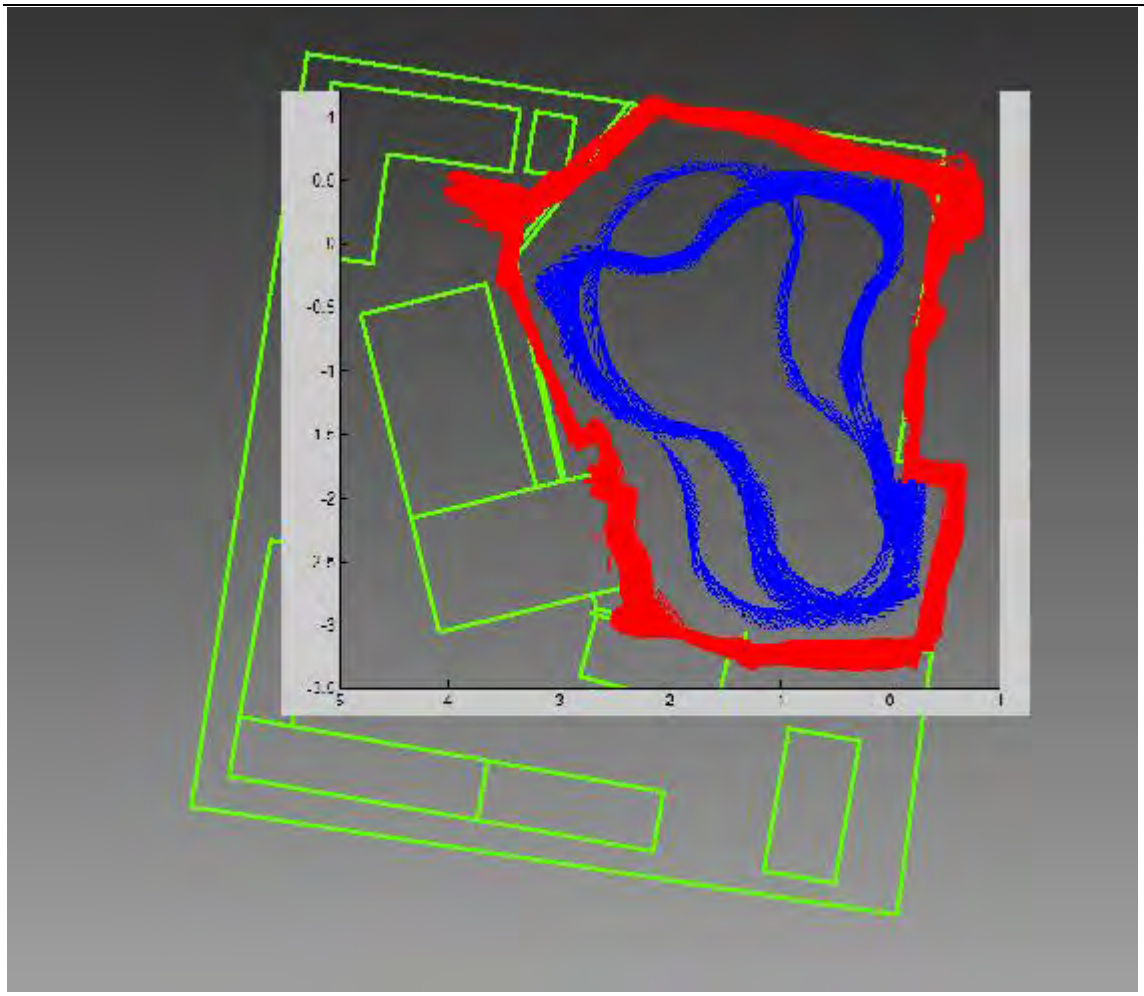
comparar un plano de este mismo con el mapa construido para ver que es de lo que realmente el robot nos está informando.

El plano en cuestión es el siguiente (figura 58):



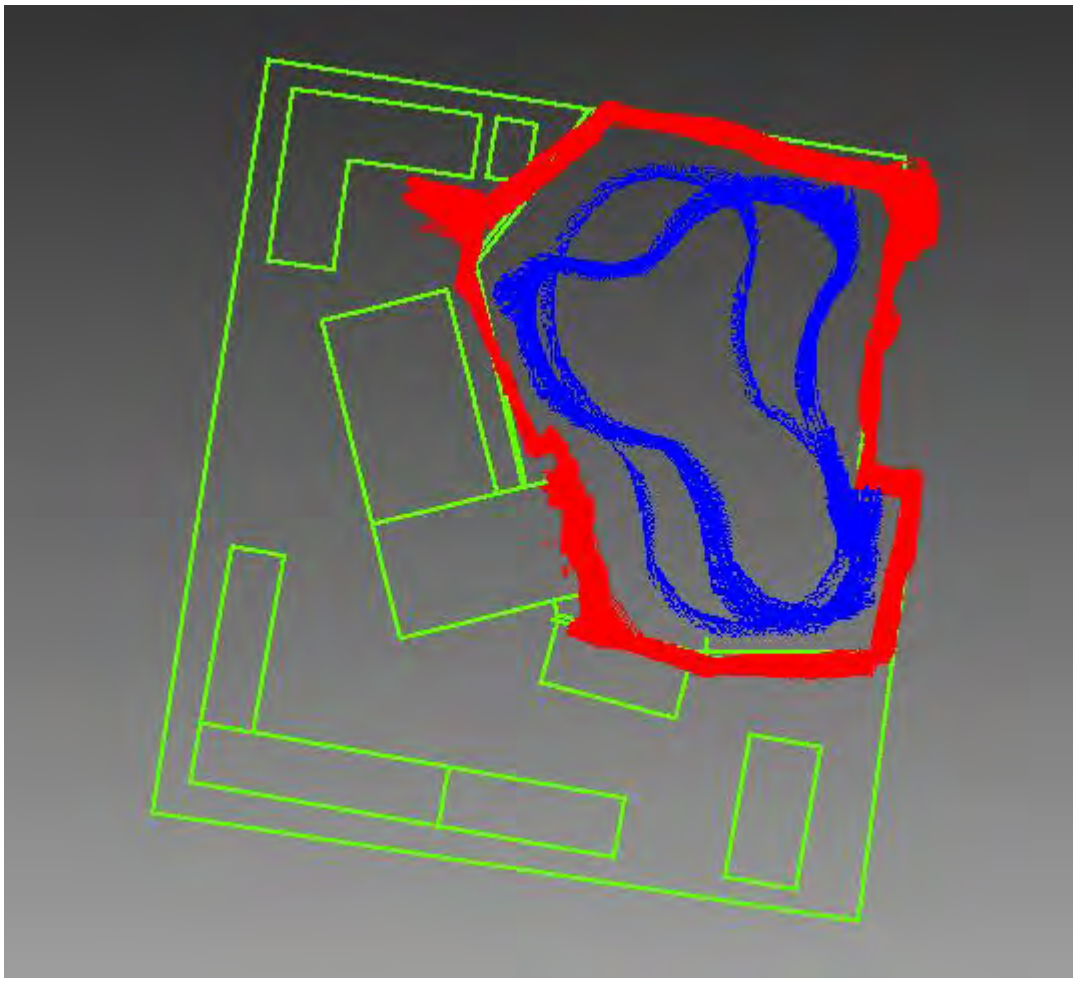
(figura 58)

Comparando este plano con el mapa explorado, obtenemos el siguiente resultado (figura 59):



(figura 59)

Sin las cotas (figura 60):



(figura 60)

3.5 Reconocimiento de objetos mediante visión artificial

3.5.1 Introducción

Con el fin de añadir la aplicación para el reconocimiento de objetos deseados por el usuario, se incluye una cámara externa situada a un extremo del robot, centrada con el láser, con el fin de conocer la posición

de los objetos reconocidos gracias a la información de la distancia a la que se encuentran del robot.

El montaje propuesto, por tanto, se ilustra en las siguientes imágenes (figura 61 y 62)

(figura 61)

(figura 62)

3.5.2 Elaboración de los algoritmos de MATLAB

Primeramente, es preciso elaborar un programa en MATLAB para la captura de imágenes para el posterior reconocimiento de los objetos deseados haciendo uso de técnicas de visión artificial.

Por tanto, creamos el programa captura_imagenes.m.

El código completo es el siguiente:

```
vid = videoinput('winvideo',2);
vid.ReturnedColorspace = 'rgb';
cond = true;
```

```
datos = [];
info=[];
i = 0;

while(cond)
    start(vid);
    [data, time, abstime] = getdata(vid,1);
    imaqmontage(data)
    C= struct2cell(abstime);
    A = cell2mat(C(1));
    info = [info; A(4:end)];
    datos = [datos;data];
    i = i + 1;
    stop(vid)
end
```

Lo primero que hay que hacer es declarar el objeto de video:

Para trabajar con la cámara de vídeo hay que crear un objeto que controle la cámara, conectando ésta con Matlab. Aparecerá un sumario de las características del objeto de vídeo creado, indicando el número de imágenes adquiridas, el tipo de disparo para la adquisición (trigger), el estado del objeto, etc.

```
vid = videoinput('winvideo',2);
```

Como vemos, definimos el objeto de video con la instrucción videoinput.

Como argumentos de entrada indicamos 'winvideo', el cual hace referencia a que cámara queremos utilizar, y 2, ya que en el momento de seleccionar la cámara en cuestión, se tenían 2 con el mismo nombre (la webcam incluida con el ordenador en el cual se realiza el procesado, y la cámara externa conectada al mismo mediante cable USB).

Si no se tiene claro cuál es el nombre de la cámara que se quiere usar, se puede utilizar el comando imaqhwinfo, que nos devuelve las cámaras que tenemos disponibles.

Ya tenemos por tanto el objeto de video que hemos llamado vid.

A continuación, indicamos que queremos que la cámara capte las imágenes en color (formato "Red, Green and Blue" 'RGB').

```
vid.ReturnedColorspace = 'rgb';
```

Ahora ya estamos en condiciones de comenzar con la captura de imágenes, y para ello definimos una serie de variables para controlar y almacenar los datos recogidos:

```
cond = true;
datos = [];
info=[];
i = 0;
```

cond será la variable que nos permita permanecer de forma infinita en el bucle que capta las imágenes, por lo que solo saldremos de él cuando paremos el programa.

Datos es la matriz donde almacenamos los datos de las imágenes recogidas e info es donde se guarda el momento exacto en el que estas se captaron.

Por otro lado, el valor final de i será el número de imágenes recogidas.

El bucle que se encarga de captar las imágenes es el siguiente:

```
while(cond)
    start(vid);
    [data, time, abstime] = getdata(vid,1);
    imaqmontage(data)
    C= struct2cell(abstime);
    A = cell2mat(C(1));
    info = [info; A(4:end)];
    datos = [datos;data];
    i = i + 1;
    stop(vid)
end
```

while es una instrucción que indica que hasta que no se cumpla su condición, el bucle continuara ejecutándose. Como la condición es cond, y esta es siempre true conseguimos nuestro objetivo, que era una ejecución infinita hasta el momento en el que paremos el programa.

Start es la instrucción que inicia al objeto de video que se le indique, en nuestro caso, vid. Para captar la información de la imagen utilizamos getdata.

Getdata es un comando que nos devuelve, por este orden, los datos de la imagen captada, el tiempo que transcurrió entre ambas capturas, y el tiempo absoluto en el que se captaron. En nuestro caso, el tiempo que tardara cada bucle en ejecutarse, es decir, el valor del tiempo entre dos capturas sucesivas cualesquiera, oscilara alrededor de los 2,5 segundos.

Posteriormente, con la instrucción imaqmontage vemos en tiempo real la imagen capturada en el bucle actual (data) en el que nos encontramos, y guardamos la imagen y el tiempo en el que se capturo en datos e info.

Para ello, primero tenemos que preparar el tiempo en el que se capturo la imagen, ya que los datos con los que trabajamos no son números, sino una estructura que contiene la hora, minutos, segundos y milisegundos.

Los pasos a seguir son pasar la estructura a celdas, ya que tenemos la opción en matlab de pasar los datos de celdas a datos de tipo numérico.

Para ello utilizamos dos instrucciones: struct2cell, que pasa los datos del formato estructura a celda, y cell2mat, que pasa los datos de formato celda a formato numérico. Los argumentos de entrada de estas instrucciones están seleccionadas para obtener únicamente la hora, minuto, segundo y milisegundo exactos en los que se tomaron las imágenes.

Lo que nos queda es almacenar todos estos datos, ya preparados, en matrices para poder realizar el procesado de las imágenes.

Para ello, redimensionamos las ya nombradas matrices datos e info. Además aumentaremos el valor de i en uno en cada bucle, de forma que tengamos guardado en esa misma variable las imágenes captadas.

Para terminar, paramos el objeto de video (stop(vid)) antes de finalizar el bucle.

Ahora que tenemos las imágenes capturadas, damos la siguiente instrucción en la ventana de comandos de MATLAB:

>>"save datos". Con este comando guardamos los datos en la variable datos, y dispondremos de ellos siempre que queramos escribiendo >>"load datos".

>>"save info". Necesitamos también guardar el tiempo en el que se tomaron los datos.

Los datos del tiempo entre mediciones consecutivas son prescindibles, y además podemos estimarlos en 2,5 segundos como se dijo.

>>"save i". Es muy importante saber cuántas imágenes se capturaron para el siguiente programa.

Por último, se presenta a continuación el programa separación.m, que es el encargado del reconocimiento de los objetos deseados mediante el uso de técnicas de visión artificial.

Este programa recibe como argumentos de entrada las variables datos, info e i, y nos devuelve la imagen captada por la cámara, la imagen binarizada por el programa con el objeto encontrado, y el momento exacto en el que se halló de tal forma que podamos representar su localización en el entorno explorado sincronizando ese instante absoluto de tiempo con el correspondiente a la localización que tenía el robot.

Los objetos, como ya se ha mencionado en apartados anteriores, son cuadrados verdes. Por lo tanto, si el usuario desea encontrar otro tipo de objetos, este es el programa que debe modificar, concretamente, la parte del reconocimiento que se indica a continuación.

El código completo del programa separacion.m es:

```
imagenes = zeros(length(datos)/i,640,3,i);
t = [];
Ir = zeros(length(datos)/i,640,1,i);
Ib = zeros(length(datos)/i,640,1,i);
Ig = zeros(length(datos)/i,640,1,i);
num = [];

for j = 1:i
    imagenes(:,:,:,j) = datos((length(datos)*(j-
1)/i)+1:length(datos)*j/i,:,:);
    Ir(:,:,:,j) = imagenes(:,:,1,j);
    Ib(:,:,:,j) = imagenes(:,:,3,j);
    Ig(:,:,:,j) = imagenes(:,:,2,j);

    H = histograma(Ir(:,:,:,j));
    % Binarizado
    c = [];
    for q=2:256
```

```
        if H(q-1)-H(q) < -1500
            c = [c q];
        end
    end

        I = not(Ir(:,:,:,j)>c(1));
    % Guardar la imágen I y etiquetar
    I = imdilate(I,eye(3));
    I = imdilate(I,eye(3));
    I = imdilate(I,eye(3));
    I = imerode(I,eye(3));
    I = imerode(I,eye(3));
    I = imerode(I,eye(3));

    I = imfill(I,'holes');
    Y = bwlabel(I,8);
    I2 = I;
    [B,L] = bwboundaries(I2);
    char = regionprops(L,'Area');
    threshold1 = 0.7;
    threshold2 = 1.8;
    idx = [];

    for k = 1:length(B)

      % Obtenemos las coordinadas x y del limite
correspondiente.
      boundary = B{k};

              % PERIMETRO
              f = diff(boundary).^2;
              perimeter = sum(sqrt(sum(f,2)));

              % AREA
              area = char(k).Area;

      metric = perimeter^2/(area*16);

      % Seleccionar objetos cuadrados
      if metric > threshold1 && metric < threshold2
          if 100000 > char(k).Area && char(k).Area > 1200
            idx = [idx k];
          end
      end
```

```
    end

    I = ismember(Y,idx);

% H = histograma(Ir(:,:,:,j));
%     % Binarizado
%     c = [];
%     for q=2:256
%         if H(q-1)-H(q) < -1500
%             c = [c q];
%         end
%     end
%     I = not(Ir(:,:,:,j)>c(1));
% % Guardar la imágen I y etiquetar
% I = imdilate(I,eye(3));
% I = imdilate(I,eye(3));
% I = imdilate(I,eye(3));
% I = imerode(I,eye(3));
% I = imerode(I,eye(3));
% I = imerode(I,eye(3));
%
% I = imfill(I,'holes');
% Y = bwlabel(I,8);
% I2 = I;
% % imshow(I2)
% [L,B] = bwlabel(I2,8);
% char = regionprops(L,'Centroid');
% char2 = regionprops(L,'BoundingBox');
% threshold1 = 0.7;
% threshold2 = 1.3;
% idx = [];
%
% for k = 1:length(B)
%
%     %esquinas del bounding box
%     esq = [char2(k).BoundingBox(1) char2(k).BoundingBox(2);
%            char2(k).BoundingBox(1)+char2(k).BoundingBox(3)
char2(k).BoundingBox(2);
%            char2(k).BoundingBox(1)
char2(k).BoundingBox(2)+char2(k).BoundingBox(4);
%            char2(k).BoundingBox(1)+char2(k).BoundingBox(3)
char2(k).BoundingBox(2)+char2(k).BoundingBox(4)];
%
%     %Centroide
```

```
%     cent = [char(k).Centroid(1) char(k).Centroid(2)];
%     %distancias
%     d1 = cent - esq(1,:);d1 = sqrt(d1(1)^2 + d1(2)^2);
%     d2 = cent - esq(2,:);d2 = sqrt(d2(1)^2 + d2(2)^2);
%     d3 = cent - esq(3,:);d3 = sqrt(d3(1)^2 + d3(2)^2);
%     d4 = cent - esq(4,:);d4 = sqrt(d4(1)^2 + d4(2)^2);
%
%     if threshold1 < d1/d2 < threshold2
%         if threshold1 < d1/d3 < threshold2
%             if threshold1 < d1/d4 < threshold2
%                 idx = [idx k];
%             end
%         end
%     end
% end
%
% I = ismember(Y,idx);



    if length(idx) ~= 0
        figure
        subplot(1,2,1)
        imshow(I)
        subplot(1,2,2)
        imshow(imagenes(:,:,:,j)/255)
        j
        resp = input('¿Objeto encontrado? (s/n)','s')
        if resp == 's'
            num = [num;j];
        end
    end
    close all
    t = [t;j info(j,:)];
end
```

Si echamos un primer vistazo al código, podemos apreciar que se han propuesto 2 caminos para hallar el objeto deseado, que consisten en dos técnicas de visión artificial diferentes. Aunque ambas se comentaran, se propone para la realización de la búsqueda de los objetos deseados aquella que no está comentada ya que proporciona mejores resultados.

Cabe también añadir, antes de entrar a comentar el código, que debemos valorar siempre los resultados obtenidos y corroborarlos, ya que en los campos de la ingeniería siempre es necesario poner en tela de juicio cualquier solución adoptada hasta que sus resultados, una vez analizados, se dan como válidos. Es por ello que se añade en el código esta opción, en la que el programa pregunta al usuario si el objeto que ha encontrado es el que se deseaba encontrar cada vez que localiza alguno.

Para comenzar, se pretende separar la gran cantidad acumulada de datos que tenemos en imágenes individuales, ya que en este momento forman una gran matriz en la que las imágenes se apilan una debajo de otra, es decir en "filas".

Una vez separadas, queremos también separarlas en las matrices básicas de rojo, verde y azul de la imagen.

Pues bien, lo primero que hacemos es definir las siguientes variables:

```
imagenes = zeros(length(datos)/i,640,3,i);
t = [];
Ir = zeros(length(datos)/i,640,1,i);
Ib = zeros(length(datos)/i,640,1,i);
Ig = zeros(length(datos)/i,640,1,i);
num = [];
```

Imágenes será una matriz de ceros con el tamaño justo de una sola imagen, pero con una cuarta dimensión que representara en cada uno de sus valores a cada imagen captada separada del resto.

Analicemos pues sus dimensiones:

Es una matriz que contendrá exactamente el número de filas de una sola imagen, es decir, el número de filas de todas las imágenes apiladas en filas (length(datos)), dividido del número de imágenes (i).

Contendrá el número de columnas de una sola imagen, es decir, 640 columnas.

Por otra parte, tendrá 3 componentes en la tercera dimensión, que son las matrices de rojo, verde y azul.

Por último, como se ha comentado, la cuarta dimensión representa cada una de las imágenes captadas, es decir, el total de las i imágenes.

Las variables t y num las definimos como vectores vacíos, ya que habrá que redimensionarlos en posteriores bucles.

Nos queda por definir cada una de las matrices de rojo, verde y azul, que tendrán unas dimensiones iguales a la de la de la matriz imágenes, con la diferencia de que en estas matrices la tercera dimensión, la de las componentes de los colores, tendrá como valor 1, ya que solo vamos a trabajar con uno de los tres colores primarios en cada una de ellas.

Seguimos ahora entrando ya en el bucle que nos va a permitir trabajar con cada imagen por separado, consiguiendo de este modo realizar el análisis individual de cada una de ellas.

Las primeras líneas nos separan cada una de las imágenes en cuestión cada ejecución del bucle.

```
for j = 1:i
    imagenes(:,:,:,j) = datos((length(datos)*(j-
1)/i)+1:length(datos)*j/i,:,:);
    Ir(:,:,:,j) = imagenes(:,:,1,j);
    Ib(:,:,:,j) = imagenes(:,:,3,j);
    Ig(:,:,:,j) = imagenes(:,:,2,j);
```

Como vemos, es un bucle for que va desde 1 hasta i, es decir, el número de imágenes capturadas.

Primero, vamos armando la matriz imágenes especificando que cada una de las componentes de la cuarta dimensión que habíamos mencionado, va a corresponder con cada imagen captada.

Para ello, tenemos que seleccionar los datos de la matriz que contiene todas las imágenes (matriz datos) que corresponden a cada imagen. Es obvio que las columnas y los colores que corresponden a cada imagen son todas las de la matriz datos, pero esto no ocurre con las filas, ya que las imágenes están recogidas de forma consecutiva.

La forma de seleccionar las filas correspondientes es multiplicar todas las filas de la matriz datos por la iteración correspondiente menos uno, (j - 1), y sumar uno para ubicarnos en el primer elemento de la imagen correspondiente a la iteración mencionada, y vamos a contar elementos de uno en uno hasta la misma multiplicación mencionada pero sin restar esa unidad, de tal manera que las filas que hemos seleccionado se corresponden en cada iteración con cada una de las imágenes captadas (((length(datos)*(j-1)/i)+1:length(datos)*j/i).

El siguiente paso consistente en seleccionar las matrices de rojo, azul y verde de esta imagen es tan simple como definir el elemento correspondiente de las matrices Ir, Ig e Ib como el correspondiente al de la matriz imágenes seleccionando que componente queremos de su tercera dimensión en cada caso: (1 para rojo, 2 para verde y 3 para azul).

Ya tenemos las imágenes separadas y preparadas individualmente en cada iteración del bucle para realizar su análisis en búsqueda de los objetos deseados.

Lo primero que hacemos es sacar el histograma de cada imagen, por lo que creamos una función a parte llamada histograma.

El motivo de hacer esto es que con el comando disponible en MATLAB llamado "hist", tan solo obtenemos una representación del histograma de la imagen y lo que buscamos son los valores del mismo, por lo que se hace necesaria la creación de esta función.

El código completo de la función histograma.m es el siguiente:

```
function H = histograma(I)

    % Cálculo de histograma con matriz

[n m] = size(I);
H = zeros(1,256);

    for i=1:n
        for j=1:m
            H(I(i,j)+1) = H(I(i,j)+1)+1;
        end
    end
```

Histograma es una función que recibe como argumento de entrada una matriz (una imagen) y nos devuelve un vector con los valores que conforman su histograma.

El histograma de la imagen es la representación de la frecuencia con la que se dan los diferentes valores de intensidad de cada pixel.

Como vemos, el programa calcula primero las dimensiones de la matriz que le hemos pasado y las guarda en las variables n y m y predefine un vector H de ceros de dimensiones (1,256) que hacen referencia a todos los posibles valores de intensidad que puede tomar un pixel (255 valores).

Posteriormente inicia 2 bucles for que recorren la imagen que hemos pasado al programa completa gracias a la información de sus dimensiones, n y m.

Cada valor que hemos ido recorriendo de la imagen, es decir, el valor de cada uno de sus pixels, se lo pasamos al vector H como índice y le sumamos 1 (para evitar el índice 0, que en MATLAB no existe, se empieza en 1), e indicamos que ese valor se incremente en 1. Por ejemplo, si el pixel en cuestión que se está dando en esa iteración de los bucles, el I(45,87) por ejemplo, tiene como valor de intensidad 167, el elemento 167 de H, el cual define la frecuencia de los pixeles de la imagen con un valor igual a 167, se incrementa en 1.

Al final, tendremos un vector H que nos indica cuantas veces se ha repetido cada valor de intensidad en la imagen, es decir, su histograma.

Lo que hacemos entonces es calcular el histograma de la matriz Ir(:,:,:,j), es decir, de la matriz de rojos de la imagen j, y binarizarla.

```
H = histograma(Ir(:,:,:,j));
    % Binarizado
    c = [];
    for q=2:256
        if H(q-1)-H(q) < -1500
            c = [c q];
        end
    end

        I = not(Ir(:,:,:,j)>c(1));
```

Para binarizar la imagen tenemos que seleccionar un límite lo suficientemente bueno para cada imagen como para que los objetos se diferencien del fondo.

Este es un gran problema en proyectos de investigación, ya que a diferencia de entornos en los que se lleve a cabo una producción industrial, los que en estos casos se contemplan tienen una iluminación cambiante, la cual dificulta en este caso la binarización de las imágenes.

Puede que en determinadas imágenes un límite x sea suficientemente bueno, pero quizá en la siguiente imagen, la cual tenga unas condiciones de una iluminación más intensa, este límite tenga que ser inferior.

Para ello se propone la solución adoptada, que consiste en recorrer el histograma con un bucle for, buscando un cambio brusco en el histograma (entre valores sucesivos), lo cual significara que se está cambiando entre el fondo y los objetos.

Vamos a fijar el valor de este salto brusco en 1500 pixeles de diferencia entre un valor de intensidad y el siguiente.

Guardamos la posición del salto y binarizamos con ese mismo límite (que hemos llamado c).

Los resultados son bastante buenos, permitiéndonos diferenciar bastante bien los objetos del fondo.

C es un vector, pero solo buscamos el primer salto brusco (el que corresponde a la diferencia fondo − objetos), por lo que marcamos como limite el primero de sus componentes (c(1)).

Otro aspecto importante esque el fondo no es negro, sino blanco, por lo que una vez binarizada la imagen, la negamos.

Ahora vamos a realizar el preprocesado de la imagen binarizada para dejarla a punto para la extracción de características.

Primeramente se realizara una operación de cierre con 3 dilataciones y 3 erosiones, con la que se pretende unir todas las partes de los objetos que se han quedado separadas, después, si aún queda algún "agujero" en el objeto, se utilizara la instrucción imfill para rellenarlo:

```
    % Guardar la imágen I y etiquetar
    I = imdilate(I,eye(3));
```

```
    I = imdilate(I,eye(3));
    I = imdilate(I,eye(3));
    I = imerode(I,eye(3));
    I = imerode(I,eye(3));
    I = imerode(I,eye(3));

    I = imfill(I,'holes');
```

A continuación etiquetamos la imagen con el comando bwlabel utilizando la opción de vecindad de 8 pixeles en la variable Y.

Copiamos la imagen original en I2 y volvemos a etiquetar la imagen con el comando bwboundaries, que nos devuelve las etiquetas y la imagen [B,L]

```
Y = bwlabel(I,8);
    I2 = I;
    [B,L] = bwboundaries(I2);
    char = regionprops(L,'Area');
    threshold1 = 0.7;
    threshold2 = 1.8;
    idx = [];
```

Definimos como vemos la característica área, que hace referencia al área de los objetos de las etiquetas, en la variable char mediante la instrucción regionprops de las etiquetas de L, haciendo referencia a la opción 'Area'.

Ahora definimos dos límites que utilizaremos a continuación, 0.7 y 1.8.

Estos límites son una tolerancia que se mueve alrededor de 1, y que usaremos para la identificación:

Lo que se pretende es buscar objetos cuadrados, los cuales tienen unas propiedades geométricas características muy importantes.

Cuando dividimos el área de un cuadrado por su perímetro al cuadrado, obtenemos una constante, sean cuales sean las dimensiones del cuadrado: 1/16.

Esto se comprueba a continuación (figura 63)

(figura 63). Cuadrado de lado l

Como vemos, el cuadrado de la figura tiene un lado de valor l, es decir, un valor cualquiera.

Su área será:

$$Area = l\,l = l^2$$

Y su perímetro al cuadrado:

$$perímetro^2 = 4l\ 4l = 16l^2$$

Por lo que su área dividida por su perímetro al cuadrado será:

$$metric = \frac{1}{16}$$

Por lo tanto, si dividimos esta variable métric por 16, tenemos una constante cuyo valor a de, al menos, oscilar en torno a 1.

Idx, la cual la definimos como un vector vacío que redimensionaremos, hace referencia a los objetos que encontramos dentro de una imagen que cumplen las características.

Ahora comenzamos un bucle que vaya de 1 hasta la longitud de B, es decir, de la imagen

Lo que hacemos es definir los límites de los objetos en boundary para hallar su perímetro y su área:

El perímetro lo hallamos como el sumatorio de la raíz cuadrada del valor que obtenemos al hacer el sumatorio, en sus 2 dimensiones, de la derivada de los límites del objeto, elevados cada uno al cuadrado.

El área ya le habíamos obtenido con el comando regionprops.

```
  for k = 1:length(B)

      % Obtenemos las coordinadas x y del limite
correspondiente.
      boundary = B{k};

               % PERIMETRO
               f = diff(boundary).^2;
               perimeter = sum(sqrt(sum(f,2)));

               % AREA
               area = char(k).Area;

      metric = perimeter^2/(area*16);
```

Por tanto, la variable metric es la resultante de dividir el perímetro al cuadrado por el área y por 16.

Ahora es cuando entran en juego los límites que definimos:

```
% Seleccionar objetos cuadrados
 if metric > threshold1 && metric < threshold2
     if 100000 > char(k).Area && char(k).Area > 1200
       idx = [idx k];
     end
 end
```

Si la variable metric se encuentra entre los dos limites, hemos encontrado un cuadrado.

Además, al estar trabajando con la matriz de rojos, los objetos verdes serán más oscuros, y destacaran más con el fondo blanco, por lo que estaremos hallando principalmente cuadrados verdes.

Además se añade una segunda condición para descartar objetos ridículamente grandes, consecuencia probablemente de unas condiciones de iluminación tan desfavorables que llevan a un binarizado no valido.

Estas condiciones pueden consistir en una intensidad de luz demasiado alta que desemboca en una imagen quemada y demasiado clara en su totalidad, dando la sensación de tener un fondo negro al negar la imagen.

Lo mismo puede pasar cuando la imagen pierda toda la iluminación en rincones extremadamente oscuros, los cuales se confundirán con objetos cuadrados gigantes al negar la imagen.

Con esta condición del tamaño de los objetos, descartamos estos objetos "gigante" y objetos diminutos, como podrían ser pequeñas conexiones en las paredes por ejemplo.

Por tanto, si se pasan estas 2 dobles condiciones, hemos encontrado uno de los objetos deseados, por lo que lo metemos en la variable idx.

Para finalizar, indicamos que es el objeto deseado:

```
end

    I = ismember(Y,idx);
```

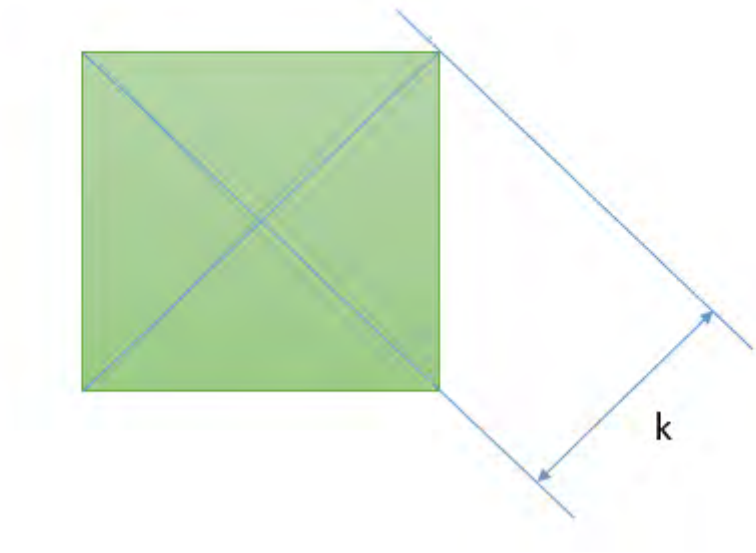
Esta constituye la primera alternativa para encontrar el objeto en cuestión, pero se propone otra que también obtiene resultados aunque no tan robustos como la anterior:

```
% H = histograma(Ir(:,:,:,j));
%     % Binarizado
%     c = [];
%     for q=2:256
%         if H(q-1)-H(q) < -1500
%             c = [c q];
%         end
%     end
%     I = not(Ir(:,:,:,j)>c(1));
% % Guardar la imágen I y etiquetar
% I = imdilate(I,eye(3));
% I = imdilate(I,eye(3));
% I = imdilate(I,eye(3));
% I = imerode(I,eye(3));
% I = imerode(I,eye(3));
% I = imerode(I,eye(3));
%
% I = imfill(I,'holes');
% Y = bwlabel(I,8);
% I2 = I;
% % imshow(I2)
% [L,B] = bwlabel(I2,8);
% char = regionprops(L,'Centroid');
% char2 = regionprops(L,'BoundingBox');
% threshold1 = 0.7;
% threshold2 = 1.3;
% idx = [];
%
% for k = 1:length(B)
```

```
%
%     %esquinas del bounding box
%     esq = [char2(k).BoundingBox(1) char2(k).BoundingBox(2);
%            char2(k).BoundingBox(1)+char2(k).BoundingBox(3)
char2(k).BoundingBox(2);
%            char2(k).BoundingBox(1)
char2(k).BoundingBox(2)+char2(k).BoundingBox(4);
%            char2(k).BoundingBox(1)+char2(k).BoundingBox(3)
char2(k).BoundingBox(2)+char2(k).BoundingBox(4)];
%
%     %Centroide
%     cent = [char(k).Centroid(1) char(k).Centroid(2)];
%     %distancias
%     d1 = cent - esq(1,:);d1 = sqrt(d1(1)^2 + d1(2)^2);
%     d2 = cent - esq(2,:);d2 = sqrt(d2(1)^2 + d2(2)^2);
%     d3 = cent - esq(3,:);d3 = sqrt(d3(1)^2 + d3(2)^2);
%     d4 = cent - esq(4,:);d4 = sqrt(d4(1)^2 + d4(2)^2);
%
%     if threshold1 < d1/d2 < threshold2
%         if threshold1 < d1/d3 < threshold2
%             if threshold1 < d1/d4 < threshold2
%                 idx = [idx k];
%             end
%         end
%     end
% end
%
% I = ismember(Y,idx);
```

En esta alternativa, el reconocimiento se realiza mediante el siguiente principio.

Todos los cuadrados que vamos captar no tienen ninguna rotación (problema que no afectaría a la propuesta anterior), por lo que se pueden encerrar los objetos encontrados en "cajas" y medir las líneas surgen desde los vértices de esta caja hasta el centroide del objeto (figura 64):



(figura 64)

Sean cuales sean las dimensiones del cuadrado, si este lo es, todas estas líneas que van desde su centroide hasta la caja que lo encierra tendrán un valor constante k.

Igual que antes, se aplica una cierta tolerancia (0.7 − 1.3)

Después de realizar el mismo binarizado y el mismo preprocesado, se determinan el centroide y la caja que lo encierra o bounding box con la instrucción regionprops:

```
char = regionprops(L,'Centroid');
```

```
char2 = regionprops(L,'BoundingBox');
```

Se definen las esquinas de las bounding boxes y las distancias al centroide.

Una vez hecho esto, solo hay que aplicar las condiciones con las tolerancias. Si las cumplen, son los objetos deseados, igual que antes.

Para terminar, sea cual sea la alternativa escogida, se realiza la comprobación de cada resultado obtenido:

```
if length(idx) ~= 0
        figure
        subplot(1,2,1)
        imshow(I)
        subplot(1,2,2)
        imshow(imagenes(:,:,:,j)/255)
        j
        resp = input('¿Objeto encontrado? (s/n)','s')
        if resp == 's'
            num = [num;j];
        end
    end
    close all
    t = [t;j info(j,:)];
end
```

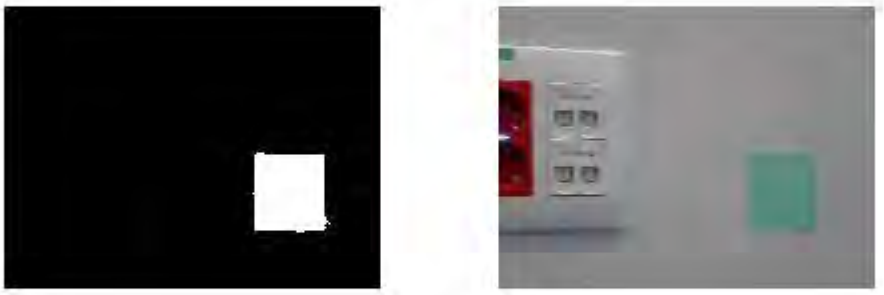
Como vemos, esta parte del código nos indica que, si idx no es nulo, es decir, se ha encontrado al menos un objeto de los deseados en la imagen procesada, se muestra la imagen original por pantalla, la imagen binarizada y procesada con el objeto que el programa propone como deseado, y se pide por pantalla una respuesta a la pregunta si el objeto a sido encontrado.

La respuesta aparentemente solo puede ser (s/n) (si o no), pero en realidad es s o cualquier otro símbolo que significara que no.

Si la respuesta a sido si (s), se guardara que componente de la matriz imágenes es, es decir, que imagen concreta es, en el vector num, se cerraran los bucles, se cerraran todas las ventanas, y se guardara en el vector t el momento exacto en el que se captó esa imagen.

Para apreciar la precisión del programa, se muestran a continuación los resultados obtenidos en una de las pruebas realizadas.

En concreto, en la prueba de la que se obtuvo el mapa que anteriormente se mostró:



(figura 65)

(figura 66)

(figura 67)

(figura 68)

(figura 69)

(figura 70)

3.5.3 Representación de la ubicación del objeto en el entorno

Por último, solo nos queda ubicar estos objetos en el entorno previamente explorado.

Para ello nos servimos de la última parte del código del programa SLAM.m:

```
for co=1:length(num)
    pos = find(posicion(:,5)==info(num(co),2));
    ext = find(posicion(pos,6)==round(info(num(co),3)));

plot3(px(ext(1),85),py(ext(1),85),z(end),'gs','LineWidth',10)
end
```

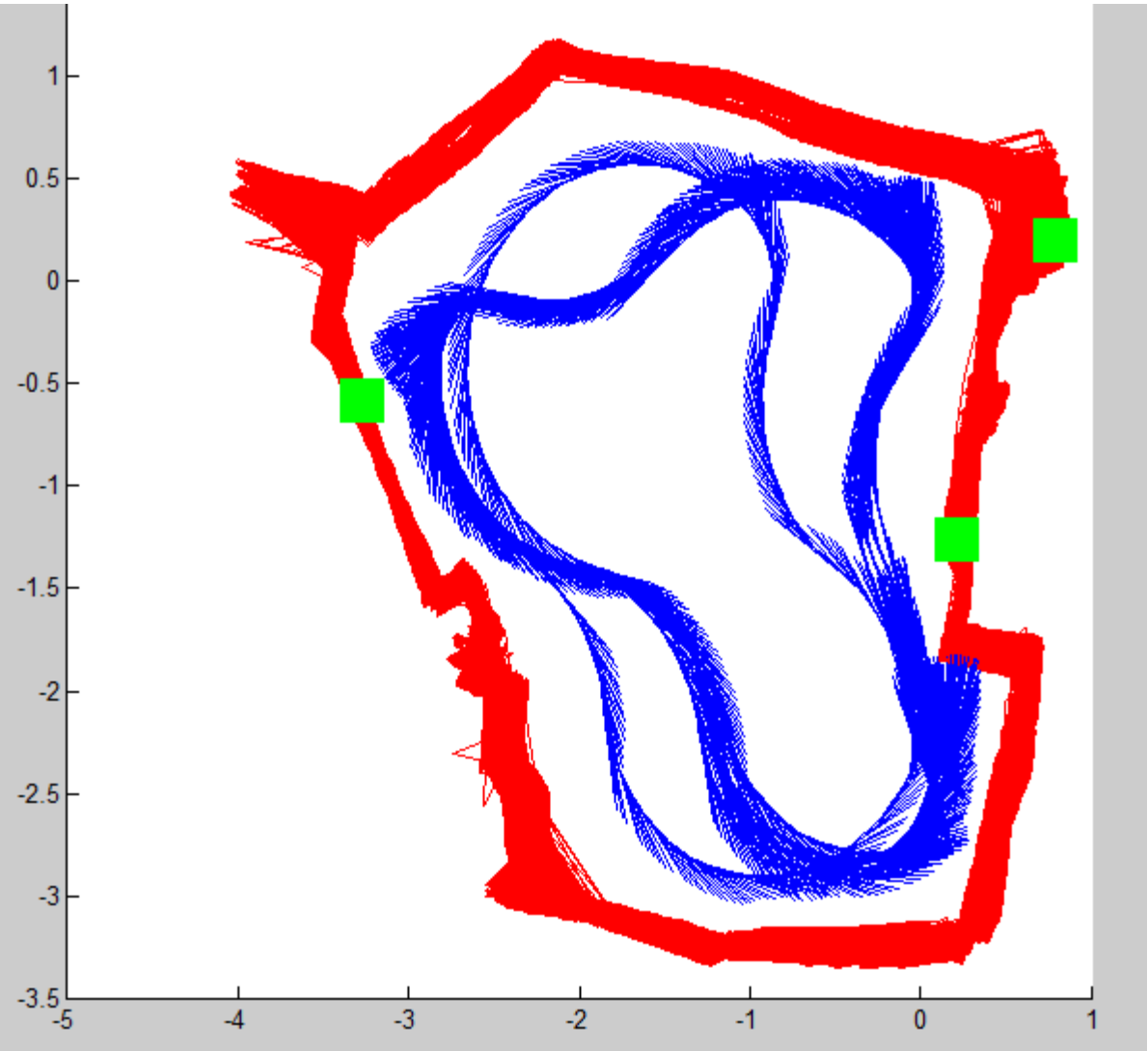
Con este último bucle, recorremos todos los datos que hemos obtenido referentes al tiempo absoluto en el que se recogieron los datos de la

posición del robot, y se captaron las imágenes que contienen el objeto deseado.

Por lo tanto, estamos sincronizando estos datos, de tal modo que podamos saber el lugar en el que se encontraba el robot cuando capto estas imágenes.

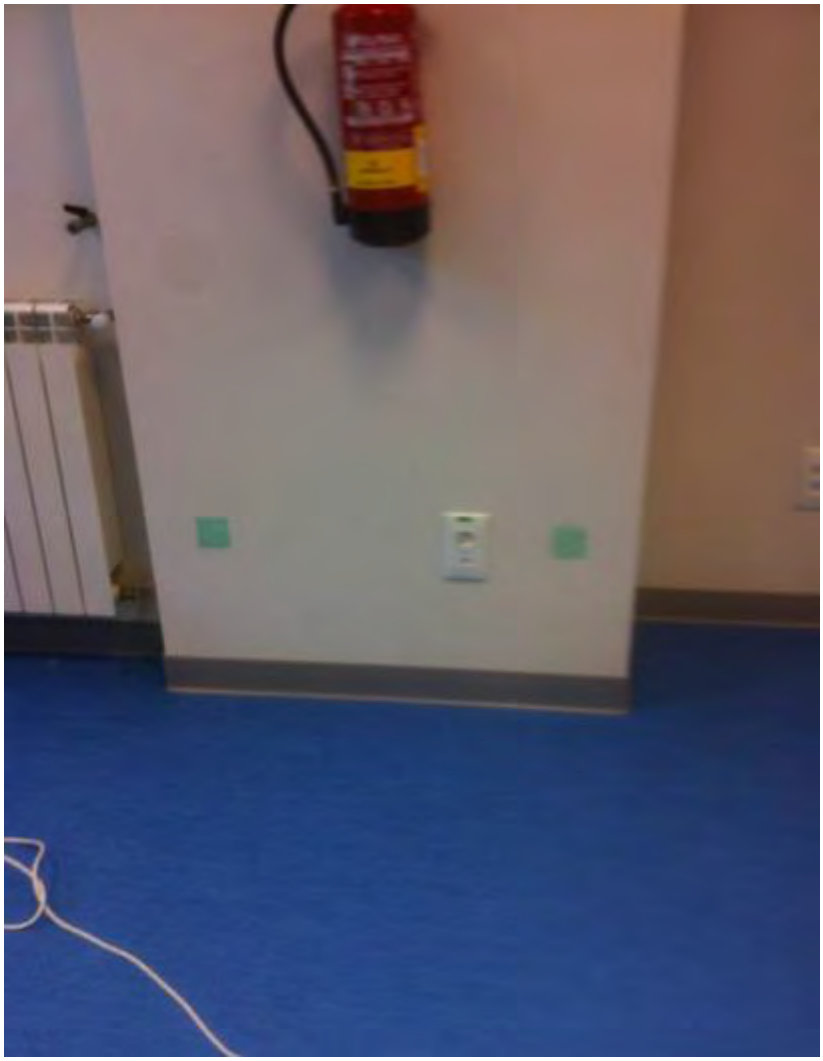
Por lo tanto, como sabemos que los objetos se encuentran pegados a la pared del entorno, los representamos ayudándonos de las coordenadas que detecto el láser central en ese instante.

El resultado es la localización en el mapa generado de estos objetos con bastante precisión (figura 71):



(figura 71)

Para poder apreciar esta precisión, se muestran las fotografías de la localización de los 3 objetos distribuidos por el entorno (figuras 72 y 73)



(figura 72)

(figura 73)

4 Conclusiones y trabajos futuros

Podemos extraer una serie de conclusiones del estudio realizado en este proyecto.

- El avance que la tecnología ha experimentado en las últimas décadas nos permite hoy en día adoptar unas soluciones a diferentes problemas del día a día con herramientas muy complejas.
- A día de hoy, la fuerte automatización de las herramientas actuales, hace prácticamente imposible la posibilidad de que los operarios de nuestro tiempo resulten heridos en diferentes procesos.
- Las aplicaciones de los autómatas en los campos que abarca el SLAM se amplían día a día, con técnicas cada vez más precisas, que nos proveen de una alta cantidad de información de los entornos aun no explorados.
- Los estudios de lugares inhóspitos o incluso imposibles para el hombre, se facilitan enormemente mediante la exploración autónoma.
- En situaciones extremas de peligro, cada vez son más las ocasiones en las que se plantea como opción la utilización de autómatas.

En las soluciones aportadas en este proyecto, hemos podido apreciar que los resultados de las técnicas utilizadas de SLAM no son bastos y aproximados, si no que recrean con bastante exactitud y precisión los entornos explorados, permitiéndonos incluso la localización de objetos concretos en el mismo. Estas tareas, arduas en muchos de los casos para el hombre, son un mero trámite de programación y puesta a punto para

una serie de robots que pueden desempeñar de forma muy robusta este tipo de misiones.

Se proponen como futuras líneas de estudio y trabajos a realizar, así como otro tipo de proyectos, la modificación de los programas de visión artificial para el reconocimiento de otro tipo de objetos de amplio interés en distintos ámbitos de aplicaciones, como puede ser reconocimiento de extintores, reconocimiento de salidas de emergencia, planes de evacuación, reconocimiento de minas, reconocimiento de minerales o incluso reconocimiento de personas concretas.

La adición a la plataforma móvil de un manipulador robótico, para que esta pueda realizar cierta clase de tareas más elaboradas, como no solo el desplazamiento hasta el lugar del objeto concreto deseado, si no la manipulación del mismo.

Se proponen también la adición al robot de accesorios para una mayor interacción externa con el mismo, como pueden ser altavoces, micrófonos o incluso iluminación propia para la exploración de entornos oscuros.

5 Anejos

5.1 Modelo cinemático

5.1.1 Introducción

La cinemática de un robot es el estudio de los movimientos de un robot. En un análisis cinemático la posición, velocidad y aceleración de cada uno de los elementos del robot son calculadas sin considerar las fuerzas que causan el movimiento. La relación entre el movimiento y las fuerzas asociadas son estudiadas en la dinámica de robots.

5.1.2 Restricciones no holónomas

El robot puede moverse instantáneamente adelante o atrás pero no lateralmente por el deslizamiento de las ruedas.

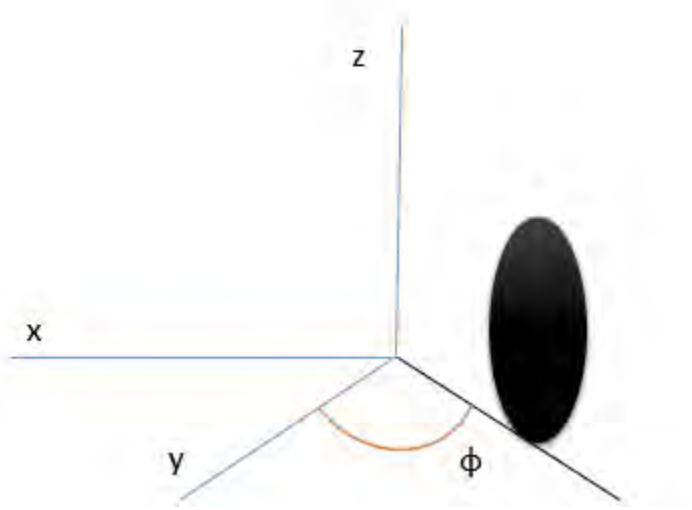
Expresado matemáticamente:

(figura 74)

$$G_{(p,\dot{p},t)=0}$$

$$\dot{x} = R \cdot \dot{\theta}$$

$$\int dt \rightarrow x - R \cdot \theta = cte$$



(figura 75)

R Holónoma (no depende de $\dot{p}$)

$$-\dot{x} \cdot \sin\phi + \dot{y}\cos\phi = \dot{\theta} \cdot R$$

$$\dot{x} \cdot \cos\phi + \dot{y}\sin\phi = 0$$

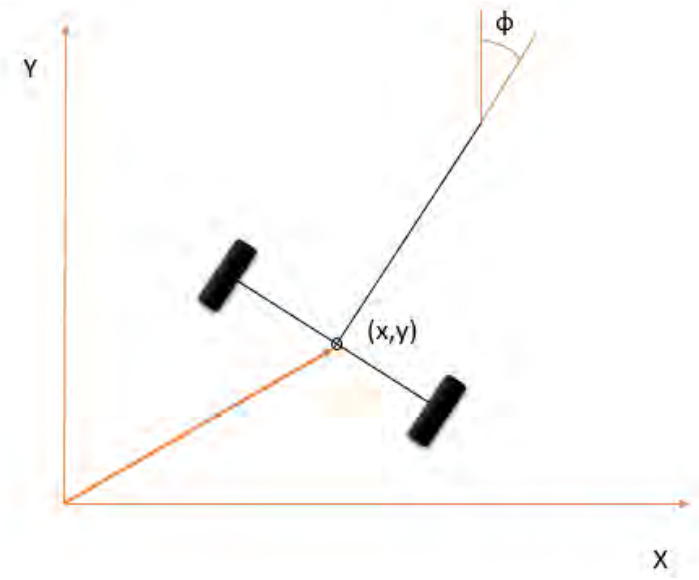
(no integrable)

R no holónoma (depende de $\dot{p}$ y no es integrable)

5.1.3 Representación de robots con una configuración de movimiento de tipo diferencial

La configuración del robot utilizado en este proyecto (RobuLAB10) es de tipo diferencial, por lo que cabe desarrollar el modelado cinemático de robots con este tipo de configuración.

Las coordenadas (x,y) representan la posición del robot con respecto a las coordenadas globales y el ángulo φ su orientación con respecto a un eje paralelo a y (figura 76)



(figura 76)

Nótese que para especificar la configuración hay que indicar el valor de las 3 variables (x, y, φ). Se dice que se tiene una restricción holónoma y 2 grados de libertad.

5.1.4 Cinemática directa

Hipótesis previas:

- El Robot se mueve sobre una superficie plana
- Los ejes de guiado son perpendiculares al suelo
- Rodadura pura (No hay deslizamiento)
- Robot como sólido rígido (no hay flexión)
- Las trayectorias se pueden aproximar como arcos de circunferencia entre dos periodos de muestreo consecutivos.

Las variables de control en un robot de locomoción con guiado diferencial son las velocidades de las ruedas laterales. Sean pues ω_i y ω_d las velocidades de giro de las ruedas izquierda y derecha respectivamente. Si el radio de las ruedas es c, las velocidades lineales serán:

$$v_i = \omega_i \cdot c$$

$$v_d = \omega_d \cdot c$$

Por lo que la velocidad lineal y angular vienen dadas por:

$$v = \frac{v_d + v_i}{2} = \frac{(w_d + w_i)}{2} \cdot c$$

$$\omega = \frac{v_d - v_i}{b} = \frac{(w_d - w_i)}{b} \cdot c$$

Siendo b la vía del vehículo (espacio entre las 2 ruedas).

Por lo tanto, si se especifica la velocidad lineal y angular del vehículo, las velocidades de giro que hay que aplicar a las ruedas izquierda y derecha son:

$$w_i = \frac{v - \left(\frac{b}{2}\right) \cdot \omega}{c}$$

$$\omega_d = \frac{v + \left(\frac{b}{2}\right) \cdot \omega}{c}$$

Atendiendo a esto, la variación diferencial de las variables necesarias para representar el modelo (x, y, φ) pueden representarse en función de las variables de control de forma matricial como:

$$\begin{bmatrix} x' \\ y' \\ \phi' \end{bmatrix} = \begin{bmatrix} -(c \cdot \sin\phi)/2 \\ (c \cdot \cos\phi)/2 \\ -c/b \end{bmatrix} \cdot w_i = \begin{bmatrix} -(c \cdot \sin\phi)/2 \\ (C \cdot \cos\phi)/2 \\ c/b \end{bmatrix} \cdot \omega_d$$

$$= \begin{bmatrix} -(c \cdot \sin\phi)/2 & -(c \cdot \sin\phi)/2 \\ (c \cdot \cos\phi)/2 & (c \cdot \cos\phi)/2 \\ -c/b & c/b \end{bmatrix} \cdot \begin{bmatrix} \omega_i \\ w_d \end{bmatrix}$$

O en función de las velocidades lineal y angular:

$$\begin{bmatrix} x' \\ y' \\ \phi' \end{bmatrix} = \begin{bmatrix} v_{(t)} \cdot \cos\theta \\ v_{(t)} \cdot \sin\theta \\ \omega \end{bmatrix}$$

Para estimar pues la posición, se ha de integrar esta expresión, es decir, las variaciones infinitesimales de x, y y φ.

$$\begin{bmatrix} x_{(t)} \\ y_{(t)} \\ \phi_{(t)} \end{bmatrix} = \begin{bmatrix} x_0(t) \\ y_0(t) \\ \phi_0(t) \end{bmatrix} + \begin{bmatrix} \int_{\Delta t} v(t) \cdot \cos\phi(t)\, dt \\ \int_{\Delta t} v(t) \cdot \sin\phi(t)\, dt \\ \int_{\Delta t} \omega(t) \cdot dt \end{bmatrix}$$

Como ampliación, es interesante contemplar la posibilidad de tener un periodo de observación Δt, situación en la cual, las integrales anteriores pueden ser reemplazadas por los desplazamientos diferenciales Δx, Δy, Δϕ.

Por tanto, si mantenemos la frecuencia de muestreo constante y elevada, se puede estimar la posición y orientación del móvil mediante las siguientes ecuaciones en diferencias:

$$x_k = x_{k-1} + \Delta x_k$$

$$y_k = y_{k-1} + \Delta y_k$$

$$\phi_k = \phi_{k-1} + \Delta \phi_k$$

5.1.5 Ecuaciones y cálculo de distancia con encoders incrementales

El cálculo del desplazamiento de las ruedas a la cual van acoplados los encoders se realiza tomando el diámetro de la rueda el cual es de 14.5 cm.

Se determina la resolución en centímetros de paso entre ranuras del encoder así:

Distancia recorrida en una vuelta:

$$(\pi \times 14.5 \text{ cm}) = 45{,}5530 \text{ cm}$$

Para determinar la distancia recorrida por el robot diferencial se debe tener en cuenta las siguientes variables:

- Distancia recorrida = S
- Radio de las llantas = R
- Pulsos por vuelta = PPV
- Número de pulsos recorridos= C

$$s = \frac{R \cdot 2\pi}{PPV} \cdot c$$

5.1.6 Modelo cinemático Jacobiano

En general la cinemática de un robot móvil puede expresarse:

$$\dot{p} = J(p) \cdot \dot{q}$$

Donde:

- p → Coordenadas generalizadas

- q → Variables de actuación
- J (p) → Matriz Jacobiana

Para nuestro modelo diferencial:

$$\begin{bmatrix} x' \\ y' \\ \phi' \end{bmatrix} = \begin{bmatrix} v_{(t)} \cdot \cos\theta \\ v_{(t)} \cdot \sin\theta \\ \omega \end{bmatrix}$$

p → $(x, y, \phi)^T$

q → $(v, \omega)^T$

J (p) → $\begin{bmatrix} \cos\phi & 0 \\ \sin\phi & 0 \\ 0 & 1 \end{bmatrix}$

5.1.7 Cinemática inversa

Permite obtener las variables de actuación necesarias para seguir una determinada trayectoria.

Si partimos del modelo Jacobiano podemos obtener el modelo inverso como:

$$\dot{p} = J(p) \cdot \dot{q} \rightarrow \dot{q} = J^*(p) \cdot \dot{p}$$

Con:

$$J^*(p) = \left(J^T(p) \cdot J(p)\right)^{-1} \cdot J^T(p)$$

Esta expresión es conocida como la pseudoinversa de una matriz (no cuadrada).

Tenemos por tanto:

$$J^T(p) = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$J^T(p) \cdot J(p) = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\phi & 0 \\ \sin\phi & 0 \\ 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos^2\phi + \sin^2\phi & 0 \\ 0 & 1 \end{bmatrix}$$

$$\left|J^T(p) \cdot J(p)\right| = \begin{vmatrix} \cos^2\phi + \sin^2\phi & 0 \\ 0 & 1 \end{vmatrix} = \cos^2\phi + \sin^2\phi$$

$$\left(J^T(p) \cdot J(p)\right)^{Adj} = \begin{pmatrix} 1 & 0 \\ 0 & \cos^2\phi + \sin^2\phi \end{pmatrix}$$

$$\left(J^T(p) \cdot J(p)\right)^{-1} = \begin{bmatrix} \cos^2\phi + \sin^2\phi & 0 \\ 0 & 1 \end{bmatrix}^{-1}$$

$$= \frac{1}{\cos^2\phi + \sin^2\phi} \cdot \begin{pmatrix} \cos^2\phi + \sin^2\phi & 0 \\ \sigma & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\cos^2\phi + \sin^2\phi} \end{pmatrix}$$

$$J^*(p) = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\cos^2\phi + \sin^2\phi} \end{pmatrix} \cdot \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$J^*(p) = \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ 0 & 0 & \frac{1}{\cos^2\phi + \sin^2\phi} \end{pmatrix}$$

Una vez hallada la pseudoinversa podemos expresar las variables de actuación en función de una determinada trayectoria.

$$\begin{bmatrix} v_{(t)} \\ \omega_{(t)} \end{bmatrix} = \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ 0 & 0 & \frac{1}{\cos^2\phi + \sin^2\phi} \end{pmatrix} \begin{pmatrix} x'_{(t)} \\ y'_{(t)} \\ \phi'_{(t)} \end{pmatrix}$$

5.2 Robosoft.RobuBOX.Generic.Devices

5.2.1 Introducción

Robosoft.RobuBOX.Generic.Devices es el grupo de librerías que se utilizó en el servicio navegador para tener acceso al control diferencial, la Odometría y el láser.

No obstante, no solo tenemos esas funcionalidades con esta librería, sino que también nos ofrece las siguientes opciones:

- CarDrive
- DifferentialDrive
- Gripper
- IOCard
- GPS
- GPSGGA
- LaserSensor
- Odometry
- PanTiltUnit
- SerialManipulator
- UltraSoundSensor

5.2.2 Descripción

Los contracts de las librerías de "Devices" ponen a nuestra disposición interfaces genéricos para los diferentes tipos de sensores y actuadores que podemos encontrar más comúnmente en robótica.

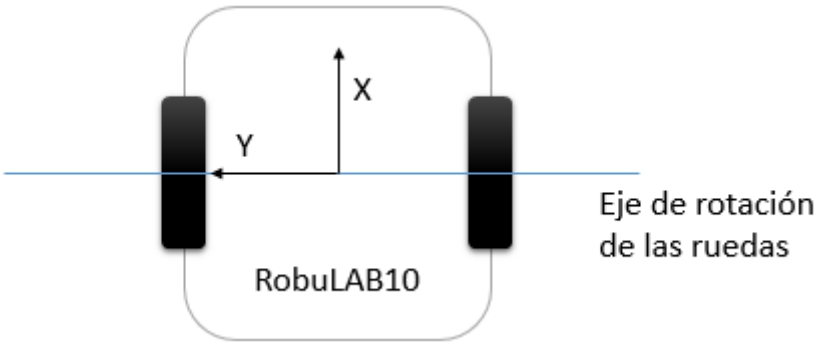
Para cada contract de la librería Devices se detallará una descripción de que es físicamente el sensor, y se explicará la estructura del estado.

Los contracts utilizados en el servicio navegador son: DifferentialDrive, Laser, Odometry (localization).

5.2.3 DifferentialDrive

Descripción

Este contract provee una interface genérica para robots de tipo diferencial, como es el caso de robuLAB10. Podemos modelar estos robots con 2 únicas ruedas con los mismos ejes de rotación (figura 76):



(Figura 76)

Si utilizamos el contract del control diferencial de RobuBOX tenemos que tener en cuenta que es totalmente diferente al definido en Microsoft Robotics Developer Estudio.

MRDS toma como inputs las velocidades lineales de las ruedas izquierda y derecha por separado, mientras que en robuBOX esta tarea ya está realizada y usa la velocidad lineal y la velocidad angular en la dirección

positiva del eje Z (Según están definidas las coordenadas en la figura 1) del robot.

Vamos a ver como se traducen estas velocidades lineales que demandamos a las ruedas izquierda y derecha a la velocidad lineal y angular que requerimos al robot:

El robot físicamente consigue una velocidad lineal deseada añadiendo velocidad a ambas ruedas, y angular restando esas velocidades.

Las velocidades lineales y angulares se aplican al centro del segmento definido por el eje de rotación de las ruedas y la intersección con las mismas.

Llamaremos v a la velocidad lineal, y ω a la velocidad angular. L Será el ancho del robot y D el diámetro de las ruedas

La velocidad lineal Ω_L de la rueda izquierda y la velocidad lineal Ω_R de la rueda derecha se calculan de la siguiente manera:

$$\begin{cases} \Omega_R = v + \dfrac{\omega \cdot D}{2} \\ \Omega_L = v - \dfrac{\omega \cdot D}{2} \end{cases}$$

Es fácil entonces demostrar que las velocidades lineal y angular se calculan para unas velocidades lineales dadas de las ruedas de la siguiente forma:

$$\begin{cases} \omega = \dfrac{\Omega_R - \Omega_L}{D} \\ v = \dfrac{\Omega_R + \Omega_L}{2} \end{cases}$$

Contract del control diferencial

A continuación se muestra la definición del contract del DifferentialDrive

```
[DataContract()]
public class DifferentialDriveState
{
/// <summary>
/// Describes the differential robot model.
/// </summary>
[DataMember]
public string Description;
/// <summary>
/// Identifies an instance of a robot. Useful when using
multiple instances.
/// </summary>
[DataMember] public int Identifier;
/// <summary>
/// Actual linear speed demand in meters/second.
/// </summary>
[DataMember]
public double TargetLinearSpeed;
/// <summary>
/// Actual angular speed demand in radians/second.
/// </summary>
[DataMember] public double TargetAngularSpeed;
/// <summary>
/// Actual linear speed in meters/second.
/// </summary>
[DataMember] public double CurrentLinearSpeed;
/// <summary>
/// Actual angular speed in radians/second.
/// </summary>
[DataMember] public double CurrentAngularSpeed;
```

```
/// <summary>
/// Maximum linear speed in meters/second.
/// </summary>
[DataMember] public double SecurityMaxLinearSpeed;
/// <summary>
/// Minimum linear speed in meters/second.
/// </summary>
[DataMember]
public double SecurityMinLinearSpeed;
/// <summary>
/// Maximum angular speed in radians/second.
/// </summary>
[DataMember] public double SecurityMaxAngularSpeed;
/// <summary>
/// Minimum angular speed in radians/second.
/// </summary>
[DataMember] public double SecurityMinAngularSpeed;
/// <summary>
/// Properties of the robot.
/// </summary>
[DataMember] public DifferentialDriveProperties Properties;
}
```

Descripción de los campos del estado differentialDrive

- **Description**

  Tipo: string

  Este string contiene una breve descripción del hardware al que se está aplicando este servicio.

- **Identifier**

  Tipo: integer

  Este campo puede ser usado para identificar instancias del servicio.

- **TargetLinearSpeed**

  Tipo: double

Este campo es el que contiene la información de la velocidad lineal demandada. Esta en las unidades del SI, metros / segundos ($\frac{m}{s}$).

- **TargetAngularSpeed**

  Tipo: double

  Este campo es el que contiene la información de la velocidad angular demandada. Esta en las unidades del SI, radianes / segundos ($\frac{rad}{s}$).

  La rotación se realiza en la dirección positiva del eje z (dirección contraria a las agujas del reloj).

- **CurrentLinearSpeed**

  Tipo: double

  En este campo está contenida la información de la velocidad linear actual de los drivers del robot. Metros /segundo.

- **CurrentAngularSpeed**

  Tipo: double

  En este campo está contenida la información de la velocidad angular actual de los drivers del robot. radianes /segundo.

- **SecurityMaxLinearSpeed**

  Tipo: double

  Este campo contiene la velocidad lineal máxima permitida en el robot, establecido por los servicios de seguridad de detección o de colisión.

- **SecurityMinLinearSpeed**

Tipo: double

Este campo contiene la velocidad lineal mínima permitida en el robot, establecido por los servicios de seguridad de detección o de colisión.

- **SecurityMaxAngularSpeed**

  Tipo: double

  Este campo contiene la velocidad angular máxima permitida en el robot, establecido por los servicios de seguridad de detección o de colisión.

- **SecurityMinAngularSpeed**

  Tipo: double

  Este campo contiene la velocidad angular mínima permitida en el robot, establecido por los servicios de seguridad de detección o de colisión.

- **Properties**

  Tipo: Robosoft.RobuBOX.Generic.Devices.DifferentialDrive.DifferentialDriveProperties

  Este campo contiene los parámetros para los robots de movimiento diferencial.

Operaciones y mensajes

- **Drive (DSS action: Submit)**

Este mensaje se utiliza para establecer la velocidad lineal y velocidad angular requeridas. Aun así, siempre podrán ser reemplazadas por las peticiones de seguridad.

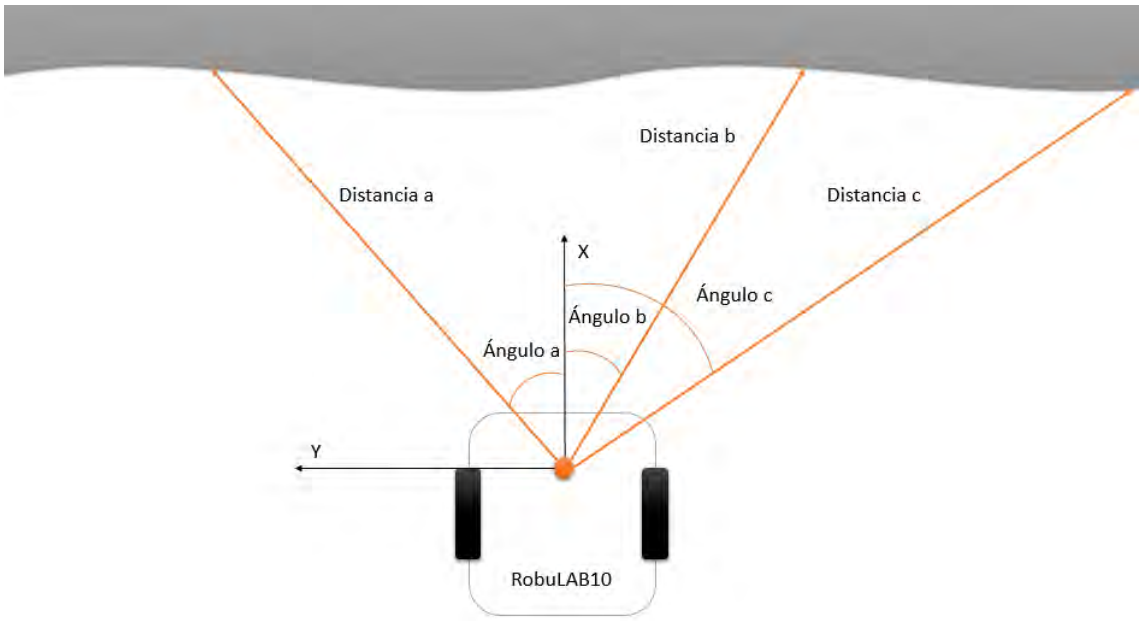
- **Seguridad (DSS action: Submit)**

  Este mensaje proporciona la seguridad relacionada con el servicio para limitar la velocidad máxima en el instante concreto. Una vez que la velocidad se ha limitado, debe reestablecerse explícitamente con este mensaje para volver a su valor típico, definido en las propiedades.

5.2.4 Sensor Laser

Descripción

Este contract es el que se corresponde con el láser de rango bidimensional. Cada muestra que proporciona es una colección de mediciones de rango en un plano dado, pero con diferentes ángulos con respecto al punto de medición donde se encuentra el propio laser (figura 77).



(Figura 77).

Las mediciones se computan desde el tiempo de vuelo del rayo del láser. Estos dispositivos también pueden determinar si hay un reflector, basado en una medida de la reflectividad.

Los ángulos se miden en la dirección positiva de giro del eje z, siendo el eje X el ángulo nulo. En cada período de muestreo, conseguimos una lista de "los ecos". Un eco es un par (ángulo, distancia).

Estado del laser

A continuación se muestra la implementación de la definición del estado del láser de rango:

```
[DataContract()]

public class LaserSensorState
{

[DataMember]
```

```
public string Description;
[DataMember]
public int Identifier;
[DataMember] public DateTime TimeStamp;
[DataMember] public Pose Pose;
[DataMember] public List<Echo> Echoes;
}
```

- **Description**

  Tipo: string

  Esta cadena tiene una breve descripción del hardware. Normalmente se establece por el fabricante del sensor y la referencia del producto.

- **Identifier**

  Tipo: integer

  Este campo puede utilizarse para identificar diferentes instancias de dos o más servicios de laser montado en el mismo robot, o compartidos en una aplicación.

- **TimeStamp**

  Tipo: System.DateTime

  Es la fecha en la que se tomaron las medidas

- **Pose**

  Tipo: Microsoft.Robotics.PhysicalModel.Proxy.Pose

  Esta es la posición de la estructura del sensor láser en el marco del robot (vector posición + cuaternión orientación

- **Echoes**

  Tipo: List of Robosoft.RobuBOX.Generic.Devices.Echo

Esta es la lista de las mediciones. La medición se almacena en una estructura de datos denominada Echo.

Tipo Echo

El contract del sensor láser define un tipo personalizado llamado echo para almacenar las medidas. Su definición es la siguiente:

```
[DataContract]
public class Echo
{
[DataMember]
public double Distance;
[DataMember]
public double Angle;
[DataMember]
public bool Overflow;
[DataMember]
public bool Reflector;
}
```

- **Distance**

  Tipo: double

  Este campo corresponde a la distancia en la figura 2.

- **Angle**

  Tipo: doble

  Este campo se corresponde con el ángulo especificado en la figura 2.

- **Overflow**

  Tipo: bool

  Es utilizado como una señal que indica que la medida está fuera del rango del sensor cuando su valor cambia a true

- **Reflector**

  Tipo: bool

  Esta señal indica que se ha detectado un reflector cuando su valor cambia a true. Esta funcionalidad no está disponible en todos los sensores láser, por lo que de forma predeterminada, el valor es false.

Operaciones y mensajes

Subscribe (DSS Action: Subscribe)

Este mensaje se utiliza para subscribirse a los updates del estado, los cuales son notificados a través de los mensajes del tipo Update (los activados cuando hay nueva información disponible en el servicio).

Update (DSS Action: Update)

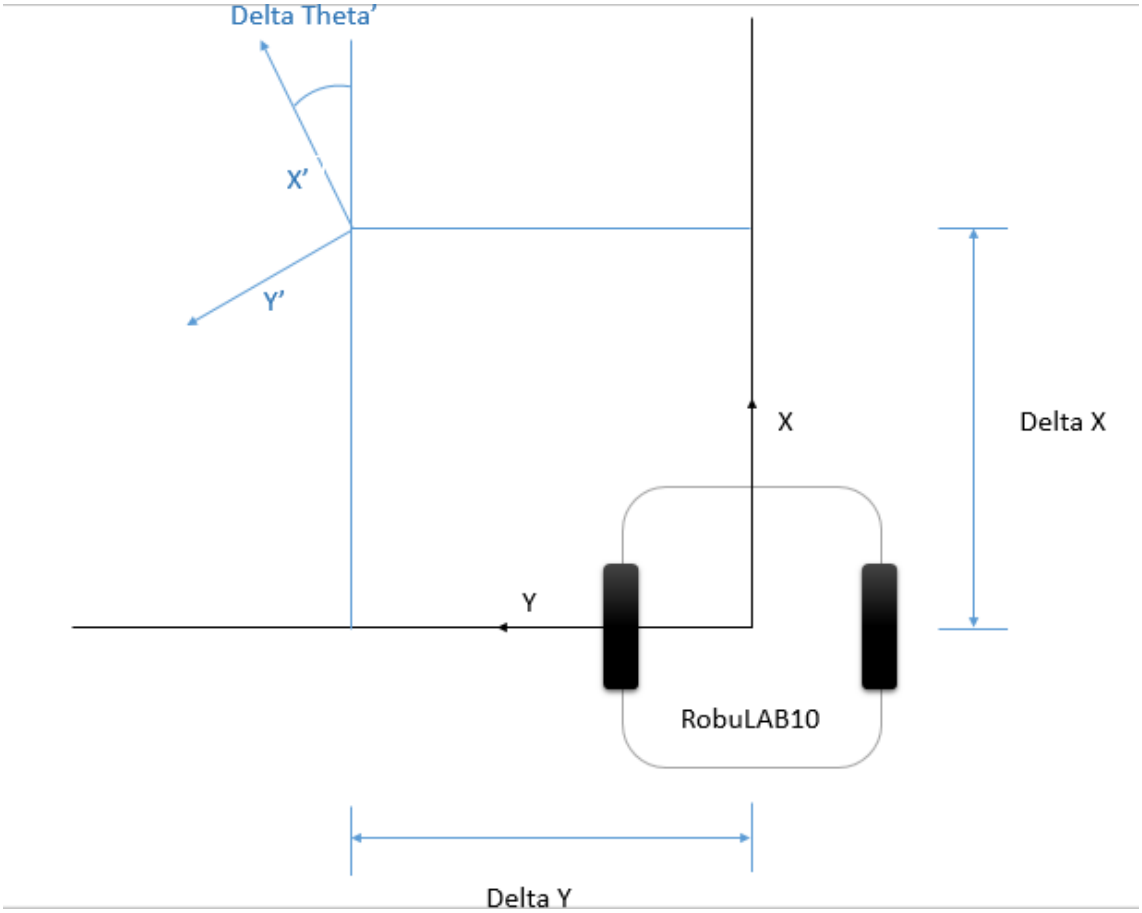
Este mensaje se utiliza solo para notificaciones a los servicios subscritos. Las notificaciones son enviadas cada vez que el servicio en el que se encuentra el contract del sensor laser recibe un nuevo conjunto de medidas de rango. La suscripción a estas notificaciones es la mejor manera de procesar la información de este servicio.

5.2.5 Localization

Descripción

Este contract consiste en la estimación de la localización del robot mediante técnicas de odometría, consistentes en la integración de la información provista por los encoders.

Generalmente se utiliza por servicios que también utilizan el contrato DifferentialDrive. La forma genérica de parametrizar un movimiento bidimensional se describe en la figura (figura 78).



(Figura 78)

El movimiento se describe con tres parámetros:

- El ángulo DeltaTheta de la rotación del marco inicial a la orientación final.
- Las distancias DeltaX y DeltaY que definen la translación desde la posición inicial a la final.

Los robots móviles usan la odometría para estimar (y no determinar) su posición relativa a su localización inicial. Es bien sabido que la odometría proporciona una buena precisión a corto plazo, es barata de implantar, y permite tasas de muestreo muy altas. Sin embargo la idea fundamental de la odometría es la integración de información incremental del movimiento a lo largo del tiempo, lo cual conlleva una inevitable acumulación de errores.

Como podemos observar, la odometría consiste en un sistema de lazo abierto, sin ninguna realimentación de referencia. El hecho de que la odometría produzca errores es algo común con cualquier otro sistema real físicamente realizable, ya que es algo intrínseco en la práctica que hasta las condiciones más favorables de funcionamiento no estén exentas de algún tipo de error con respecto a la referencia. No obstante, en el caso de sistemas de lazo abierto, como la odometría, los errores no se corrigen, sino que de hecho, se acumulan. En concreto, la acumulación de errores de orientación, causa grandes errores en la estimación de la posición, los cuales van aumentando proporcionalmente con la distancia recorrida por el robot.

A pesar de estas limitaciones, muchos investigadores están de acuerdo en que la odometría es una parte importante del sistema de navegación de un robot

La odometría se basa en ecuaciones simples que se pueden implementar fácilmente y que utilizan datos de encoders situados en las ruedas del robot. Sin embargo, la odometría también está basada en la suposición de que las revoluciones de las ruedas pueden ser traducidas en un desplazamiento lineal relativo al suelo. Esta suposición no tiene una validez absoluta. Un ejemplo extremo es cuando las ruedas patinan: si

por ejemplo, una rueda patina sobre una mancha de aceite y la otra no, entonces el encoder asociado registrará revoluciones en la rueda, a pesar de que éstas no correspondan a un desplazamiento lineal de la rueda.

Además de este ejemplo hay muchas otras razones más sutiles por las cuales se pueden producir imprecisiones en la traducción de las lecturas del encoder de la rueda a un desplazamiento lineal. Todos estos errores se pueden agrupar en dos categorías: errores sistemáticos, y errores no sistemáticos.

Entre los errores sistemáticos destacan

- Los diámetros de las ruedas no son iguales.
- La media de los diámetros de las ruedas difieren del diámetro de fábrica de las ruedas.
- Mal alineamiento de las ruedas.
- Resolución discreta (no continua) del encoder.
- La tasa de muestreo del encoder es discreta.

Entre los errores no sistemáticos se encuentran:

- Desplazamiento en suelos desnivelados.
- Desplazamiento sobre objetos inesperados que se encuentren en el suelo.
- Patinaje de las ruedas debido a:
    - Suelos resbaladizos.
    - Sobre-aceleración.
    - Derrapes (debidos a una rotación excesivamente rápida).
    - Fuerzas externas (interacción con cuerpos externos).
    - No hay ningún punto de contacto con el suelo.

Para poder estimar la posición y orientación del robot pues, se ha usado este sistema odométrico, que aun produciendo este tipo de errores, nos da una estimación bastante fiable de su localización. Además, el objetivo del proyecto son recorridos de no demasiada distancia, en los cuales el error en la odometría es totalmente despreciable.

Para calcular por tanto la localización del robot gracias a la información provista por los encoders, se ha dedicado un anejo para el modelado cinemático inverso y directo de la plataforma móvil utilizada RobuLAB.

Volviendo de nuevo al sistema de localización implementado en el Navegador, observaremos ahora el estado del contract utilizado.

State

Aquí podemos ver la implementación del estado del contract de la localización.

```
[DataContract()]
public class OdometryState
{
[DataMember]
public string Description;
[DataMember] public int Identifier;
[DataMember] public DateTime TimeStamp;
[DataMember] public TimeSpan Duration;
[DataMember] public double DeltaX;
[DataMember] public double DeltaY;
[DataMember] public double DeltaTheta;
}
```

- **Description**

  Tipo: string

  Esta cadena tiene una breve descripción del hardware subyacente. Normalmente se establece con el modelo de robot.

- **Identifier**

  Tipo: integer

  Este campo puede utilizarse para identificar una instancia del servicio.

- **Timestamp**

  Tipo: System.DateTime

  Este campo contiene la fecha de la medición.

- **Duration**

  Tipo: System.TimeSpan

  Esta es la duración del movimiento descrito por los siguientes parámetros.

- **DeltaX**

  Tipo: double

  Este es el desplazamiento en el eje x local del robot.

- **DeltaY**

  Tipo: double

  Este es el desplazamiento en el eje Y local del robot.

- **DeltaTheta**

  Tipo: double

  Este es el cambio de rumbo.

Operaciones y mensajes

Subscribe (DSS Action: Subscribe)

Este mensaje se utiliza para realizar una suscripción. Permite recibir información del movimiento a través de mensajes del Update.

Update (DSS Action: Update)

Este mensaje se utiliza sólo para enviar notificaciones a los servicios subscritos. Se reciben notificaciones en el NotifyPort con la información de la localización del robot cada vez que esta cambia (se produce un Update).

5.3 Librería RobuBOX.Core.Types

El grupo de librerías Types son muy utilizadas en RobuBOX.

En Types podemos encontrar 5 grupos que constituyen las funcionalidades de esta librería, que son Point2D, Pose2D, Trajectory, DriveParameters, LambertParameters:

5.3.1 Point2D

Este type describe un punto en un espacio bidimensional.

La clase completa Types.Point2D es la siguiente:

```
[DataContract]
public class Point2D
{
/// <summary>
/// X coordinate of the point.
/// </summary>
[DataMember]
public double X;
/// <summary>
/// Y coordinate of the point.
/// </summary>
[DataMember]
public double Y;
}
```

5.3.2 Pose2D

Este Type describe la posición de un objeto (Coordenadas x, y y orientación) en un espacio bidimensional. Hace uso de la clase Point2D:

```
[DataContract]
public class Pose2D
{
/// <summary>
/// Position of the object.
/// </summary>
[DataMember]
public Point2D Position;
/// <summary>
/// Orientation of the object.
/// </summary>
[DataMember]
public double Orientation; }
```

5.3.3 Trajectory

Esta clase describe una trayectoria, esto es, una lista de puntos (utilizando una lista de puntos definidos utilizando la clase Point2D) con un identificador y una descripción almacenada en una cadena de caracteres (string).

```
[DataContract]
public class Trajectory
{
/// <summary>
/// Numeric identifier.
/// </summary>
[DataMember]
public int Id;
/// <summary>
/// Name of the trajectory.
/// </summary>
```

```
[DataMember] public string Name;
/// <summary>
/// List of points that constitute the trajectory.
/// </summary>
[DataMember]
public List<Point2D> Points;
}
```

5.3.4 DriveParameters

Son los parámetros del sistema de movimiento diferencial del motor:

```
[DataContract]
public class DriveParameters
{
/// <summary>
/// CANOpen node Id of the motor drive.
/// </summary>
[DataMember]
public byte DriveNode;
/// <summary>
/// Diameter of the wheel.
/// </summary>
[DataMember]
public double WheelDiameter;
/// <summary>
/// Ratio of the gear box.
/// </summary>
[DataMember]
public double GearBoxRatio;
/// <summary>
/// Output of the gearBox
/// </summary>
[DataMember]
public double GearBoxOutput;
/// <summary>
/// Number of encoder lines per revolution of drive.
/// </summary>
[DataMember]
public double EncoderLines;
/// <summary>
```

```
/// Flag to invert drive consign
/// </summary>
[DataMember] public bool InvertConsign;
/// <summary>
/// Flag to invert encoder data
/// </summary>
[DataMember] public bool InvertEncoder;
/// <summary>
/// Target speed of the wheel
/// </summary>
[DataMember] public double TargetWheelSpeed;
/// <summary>
/// Current speed of the wheel
/// </summary>
[DataMember] public double CurrentWheelSpeed;
}
```

6 Bibliografía y referencias

- Robótica: manipuladores y robots móviles. Anibal Ollero, marcombc, 2001
- Smith, Randall; Self, Matthew; Cheeseman, Peter (1987). «Estimating uncertain spatial relationships in robotics». 1987 IEEE International Conference on In Robotics and Automation. Proceedings.
- Montemerlo, Michael (2003). «FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association». doctoral dissertation, tech. report CMU-RI-TR-03-28, Robotics Institute, Carnegie Mellon University.
- Proceedings of IEEE Robotics and Automation, 1988
- Mathworks
- Robosoft. Technical support.
- Craig J, John (2006). Robótica. México: pearson education.