Universidad
de La Laguna

Escuela Superior de
Ingeniería y Tecnología
Sección de Ingeniería Informática

Trabajo de Fin de Grado

Robótica móvil colaborativa

Collaborative mobile robotics

Guillermo Rivero Rodríguez

La Laguna, 4 de marzo de 2015

D. **José Demetrio Piñeiro Vera,** con N.I.F. 43.774.048-B profesor Titular de Universidad adscrito al Departamento de Ingeniería informática de la Universidad de La Laguna, como tutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

"Robótica móvil colaborativa."

ha sido realizada bajo su dirección por D. **Guillermo Rivero Rodríguez,** con N.I.F. 54.113.314-H.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de marzo de 2015.

Agradecimientos

El informe que expone en este documento no es sólo una actividad más que representa doce créditos. Representa mucho más de lo que mucha gente puede imaginar, el final de una etapa y el comienzo de otra en un largo camino que es el aprendizaje. Pero para llegar hasta donde he llegado, he necesitado únicamente esfuerzo y trabajo, sino una serie apoyos que me han animado y motivado para ir poniendo cada día un ladrillo nuevo que construirá mi formación.

Me gustaría agradecer en primer lugar a mi familia, en especial a mis padres Emilio Rivero y Alicia Rodríguez y a mi hermana Paula Rivero, por aguantarme y entender mi peculiar manera de afrontar los estudios. También me gustaría agradecer a mi compañero Boris Ballester por estar a mi lado programando muchas de las prácticas que se han de presentar a lo largo de la carrera.

Por último me gustaría agradecer a Gisela Yohana Luis por avanzar conmigo a lo largo de mi andadura universitaria, ayudándome con cosas que en su momento ni entendía y siempre brindándome el mejor ánimo y apoyo que una persona te puede ofrecer.

Licencia

Resumen

El objetivo de este trabajo ha sido principalmente aprender sobre la robótica móvil y los principales problemas que la rodean.

En este informe plantean cuales son simuladores que se utilizan en la actualidad, para abarcar los temas que corresponden con la robótica de una manera más sencilla y económica, así como sus puntos positivos y negativos.

Además se establecerá cuál es el mejor de los simuladores comentados para la representación de un problema real de robótica.

Por último quedará detallado cómo crear uno o varios robots e incluirles algunas funcionalidades como navegación hasta un punto, cálculo de rutas, evasión de obstáculos o comunicación entre robots.

Palabras clave: Robótica, móvil, colaborativa, navegación, comunicación, evasión de obstáculos

Abstract

The aim of this study is to learn about mobile robotics and the main problems that surround it.

This report shows the moderns simulators that can solve the main robotics problems in an easy and cheap way. Also, it will describe the strengths and weaknesses points of the simulators.

In addition it will be established the best of the commented simulators that can represent a real robotics problem.

Finally, it will detail how to code one or more robots with some features like navigation to a point , path planning, obstacle avoidance or communication between robots .

Keywords: Robotics, mobile, collaborative , navigation, communication , obstacle avoidance

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1 Definición

La robótica una rama tecnológica que se dedica al diseño, construcción, operación y programación de robots. La robótica combina muchas ciencias como la informática, la física, la mecánica, la electrónica etc...

1.2 Clasificación

Los robots se pueden clasificar de varias maneras, aunque vamos a comentar

1.2.1 Según su generación:

Primera generación: Son sistemas mecánicos multifuncionales con un sencillo sistema de control de secuencia fija o manuales, comúnmente llamados manipuladores

Segunda generación: Son dispositivos mecánicos que imitan una serie de movimientos que ha ejecutado un operario antes, de manera que el robot va aprendiendo los movimientos o acciones que los humanos van realizando.

Tercera generación: Robots con control sensorizado. Los ordenadores captan información del medio a través de sensores y mediante un procesador deciden como utilizar sus manipuladores.

Cuarta generación: Son sistemas que también son capaces de recibir datos del medio mediante sus sensores, pero que antes de realizar una acción envía los datos a un

ordenador que procesa los datos y decide de qué manera responder mediante los manipuladores.

1.2.2 Según su aplicación:

Industriales: Son aquellos que se usan para la manufacturación de productos, principalmente suelen ser robots de manipulación.

De servicio: Son robots que operan de forma autónoma o semiautónoma con el fin de proporcionar alguna utilidad a personas u otros equipos.

1.2.3 Según su morfología:

Manipuladores: Principalmente son brazos robot con algún manipulador en un extremo, ya sean pinzas, soldadores, pistolas de pintura, etc.

Móviles: Son artilugios provistos de ruedas, extremidades o algún otro mecanismo que les permita moverse de forma autónoma. El control de este movimiento lo hacen mediante datos que obtienen de un sensado.

Híbridos: Son robots cuya estructura se compone de una mezcla de los dos tipos anteriores.

Dentro de los robots móviles, en los que se centra este trabajo, se puede encontrar otra clasificación interesante, según el tipo de movimiento que tiene el robot.

Se dice que un robot es holonómico cuando tiene los mismo grados de libertad efectivos que controlables, es decir, si es capaz de controlar todos los grados de libertad que le afectan. A partir de esta afirmación se puede decir que es holonómico si puede cambiar su dirección sin necesidad de rotar previamente.

1.3 Características

1.3.1 Navegación

Es el conjunto de métodos o técnicas que utiliza el robot para moverse por su entorno. La navegación se realiza comúnmente mediante la implementación de mapas, ya sean de marcas de ocupación de espacios libres o de rejillas.

1.3.2 Localización

Consiste en determinar en que posición del mapa se encuentra el robot. Hay varios métodos para resolver este problema pero principalmente se realiza mediante filtros de partículas o filtros de Kalman.

1.3.3 Planificación

Consiste en establecer una ruta para llegar hasta un punto sin colisiones e idealmente de una manera óptima, encontrando el mejor camino entre todos los posibles. La planificación puede ser guiada, enseñando previamente al robot a que puntos debe desplazarse o automática, en la que se utilizan algoritmos como A* o RRT.

1.3.4 Sensado

El sensado consiste en obtener datos del entorno que puedan ser utilizados por el robot. Se pueden obtener diversos tipos de datos, desde distancias gracias a sensores de ultrasonidos o laser, hasta posiciones mediante GPS, pasando por una gran cantidad de sensores como cámaras codificadores ópticos sensores de luz...

1.3.5 Actuadores

Son elementos que permiten al robot interactuar con el mundo que les rodea, en el caso de los robots móviles los actuadores más conocidos son las ruedas, aunque hay muchos otros cómo hélices, orugas...

1.4 Descripción del problema

1.4.1 Objetivos del proyecto

En la actualidad existen muchos campos de estudio dedicados a la robótica, este hecho dificulta bastante que problema tratar debido a la gran cantidad de posibilidades. Aunque en un principio se pensó en hacer un sistema de mapeado y localización simultanea, el proyecto fue derivando en varias ideas, que aunque en un principio eran sencillas, a medida que se iban acumulando iban mostrando comportamientos más interesantes.

Este trabajo quiere abordar algunos de los temas de los que hemos hablado en puntos anteriores del informe, pero como son ideas que se pueden implementar por separado, se ha ido realizando el proyecto de forma parcial, y para que quede más claro se ha subdividido en hitos.

1.4.2 Subdivisión en hitos

Cada uno de estos hitos debe ser desarrollado e implementado en un simulador, con el fin de comprobar su correcto funcionamiento.

En la Tabla 1 se puede observar la manera en la que se va a descomponer el problema, cada uno de los apartados que aparecen serán explicados en Capitulo 3 de forma detallada.

HITO 1	Diseñar e implementar un robot capaz de seguir una ruta.
HITO 2	Diseñar e implementar un robot capaz de evitar colisiones.
HITO 3	Diseñar e implementar un robot capaz de seguir una ruta evitando colisiones.
HITO 4	Diseñar e implementar un robot capaz de comunicarse con otros robots.
HITO 5	Diseñar e implementar un robot capaz de mapear el entorno en el que se encuentra.
HITO 6	Diseñar e implementar un robot que sea capaz de comunicarse con otros robots para compartir información sobre el entorno.
HITO 7	Diseñar e implementar un robot que sea capaz de aprovechar su propio mapa del entorno para desplazarse en el mundo real.

Tabla 1 Hitos del proyecto.

Capítulo 2. Simuladores

En robótica es común el uso de programadas como herramienta de ayuda a la hora de crear robots. Gracias a estas aplicaciones se pueden obtener resultados muy similares al de un robot real sin tener que disponer del robot en sí. De esta manera se puede abaratar costes, depurar errores que en un robot real serían más difíciles de identificar o probar como afectaría una modificación a un robot ya existente. En términos de robótica, un simulador no sólo tiene que representar al robot que se quiere estudiar, sino que también engloba a otros aspectos como pueden ser el medio en el que se encuentran, ya sean objetos físicos, personas, luces, o cualquier otro elemento que pueda interactuar directa o indirectamente con el robot. Cabe destacar que algunos de los simuladores que se van a tratar a continuación, no sólo dan soporte para robot móviles, sino que también se lo brindan a robots industriales como pueden ser los brazos mecánicos.

Para este proyecto se han estudiado y testeado tres simuladores, Player/Stage, Simbad, y Vrep, y aunque finalmente el proyecto se desarrolla sobre Vrep, se van a comentar los puntos fuertes y débiles de cada simulador, así como los motivos por los que se ha escogido este último para realizar el proyecto.

2.1 Player/Stage

2.1.1 Player

Player no es un simulador, es un programa que permite al usuario controlar un robot mediante algoritmos de control. Actúa como servidor y se comunica con otros mediante sockets TCP/IP. Unos de los puntos más interesantes de este programa es que ofrece la posibilidad de abstraerse de los detalles hardware del robot, es decir, el programador no tiene que conocer los detalles de implementación concretos para un tipo de motor o un tipo de sensor, sino que permite simularlo de forma genérica. La manera de controlar el robot es enviando mensajes que sigan un protocolo de comunicación estándar, o si se desea abstraer también esta capa, es posible controlar el robot haciendo llamadas a librerías. Esto es un punto positivo ya que hay librerías definidas en diversos lenguajes como C++, Java o Phyton.

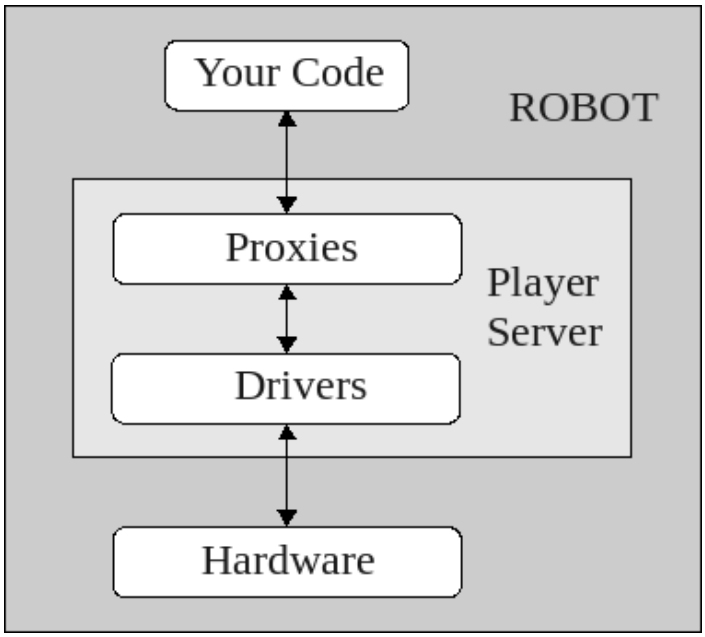

Ilustración 1 Funcionamiento de player.

2.1.2 Stage

Stage es un simulador de robots móviles. Es un principio fue desarrollado para entornos 2D, aunque después de ciertas actualizaciones también es capaz de simular entornos 3D bastante sencillos. Puede comunicarse con player para simular los detalles del robot que se haya definido, por lo que ambas herramientas en conjunto son bastante potentes a la hora de realizar simulaciones que se correspondan en la mayor medida de lo posible con el mundo real.

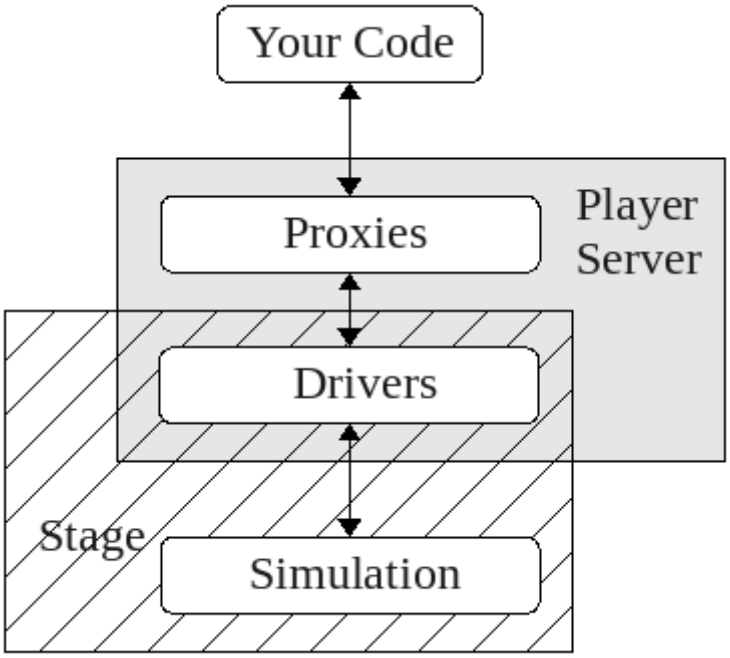

Ilustración 2 Funcionamiento de stage.

Ambos programas fueron desarrollados como proyectos cerrados, más adelante fue liberado y mantenido por una amplia comunidad, y actualmente, su desarrollo se ha visto mermado por el uso de otros simuladores 3D y programas de control de robots más complejos como pueden ser Gazebo y ROS.

2.1.3 Ventajas e inconvenientes

Ventajas:

- El controlador del robot es bastante completo.
- Permite la programación en diversos lenguajes.

- Al ser un entorno principalmente 2D, permite hacer simulaciones con muchos robots sin que baje el rendimiento del programa.

Inconvenientes:

- Muchos problemas a la hora de realizar la instalación de los módulos, requiere muchas dependencias.

- El soporte para sistemas operativos actuales como Ubuntu14 deja bastante que desear.

2.2 Simbad

Simbad permite hacer simulaciones de robots en un ordenador. Según la propia web de los desarrolladores, Simbad "permite a los programadores escribir su propio controlador del robot, modificar el mundo que le rodea, y utilizar una serie de sensores. Está dirigido principalmente a investigadores/programadores que quieren una base simple para estudiar situaciones de inteligencia artificial, aprendizaje automático, algoritmos relacionados con la robótica en general. En el contexto de la robótica y los Agentes Autónomos".

Simbad está escrito en el lenguaje Java por Louis Hugues y Nicolas Bredeche. El proyecto, organizado en SourceForge.net, está libre y se puede utilizar y modificar bajo las condiciones de la Licencia Pública General de GNU.

2.2.1 Detalles del simulador.

Un mundo Simbad puede contener agentes (robots) y los objetos inanimados (cajas, paredes, luces, etc). El tiempo en Simbad se divide en momentos discretos (ticks) de manera que tiene un gestor de tareas que distribuye las acciones de los agentes en cada tick. Al igual que los robots físicos, agentes Simbad tienen sensores (distancia, tacto, luz, etc.) y actuadores (generalmente

ruedas). En cada instante de tiempo, un robot puede actuar.

En los agentes se sobrescribe el método performBehavior() para determinar su comportamiento. En este método un robot puede leer datos de los sensores y utilizarlos para establecer las velocidades de sus motores de manera que no hay problemas a la hora de realizar la odometría aún así, cada agente lleva un registro de estado y también puede utilizar una variable contador de tiempo para realizar un seguimiento de la cantidad de ticks del reloj que han pasado desde el comienzo de la simulación.

2.2.2 API

La API Simbad consta de los siguientes paquetes principales:

simbad.sim Las clases de este paquete representan tanto el robot y como el mundo en el que se encuentra:

- **Agent** Son los robots.
- **Arch** Un obstáculo con forma de arco por el que el robot puede pasar.
- **Box** Un obstáculo con forma de cuboide.
- **CameraSensor** Simula una cámara que se puede incorporar al robot.
- **EnvironmentDescription** Representa el "mundo" al que se agregan robots y objetos tales como paredes o cajas.
- **LampActuator** Una lámpara que se puede añadir a su robot con el fin de emitir luz.
- **LightSensor** Detecta la intensidad de la luz.
- **RangeSensorBelt** Contiene un conjunto de sensores de rango alrededor del robot.
- **RobotFactory** Esta opción sirve para agregar sensores a su robot.
- **Wall** Un obstáculo con forma de pared.

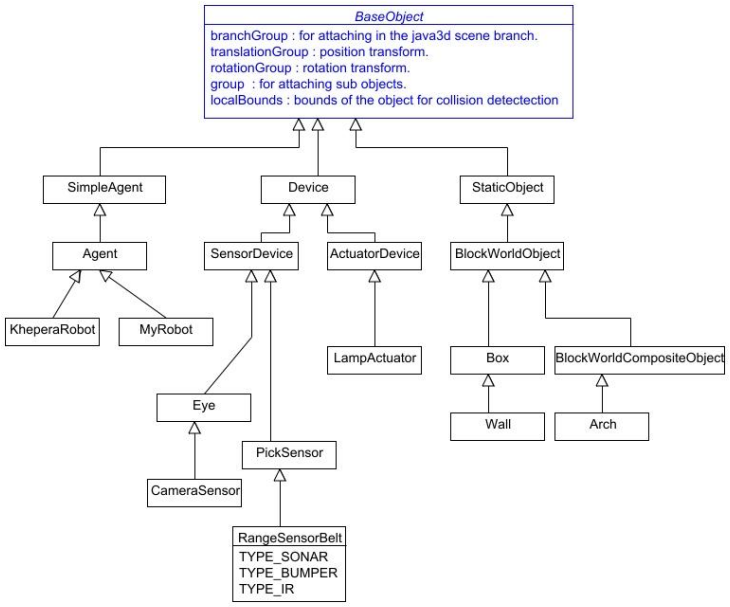

Ilustración 3 Funcionamiento de simbad.

En una primera instancia este simulador parece bastante bueno para afrontar los problemas que se quieren resolver en este trabajo. No obstante a medida que se iba incrementando la complejidad de los algoritmos a implementar iban quedando a la vista las debilidades del simulador.

En un principio se realizó un algoritmo para seguir paredes, con el fin de entender un poco el uso del navegador. Y el resultado fue bueno utilizando el algoritmo descrito a continuación:

1). Ir recto
2). Cuando chocas con una pared, girar a la izquierda.
3). Si ves un pasillo a la derecha, girar a la derecha.
4). Parar cuando llegues al final.

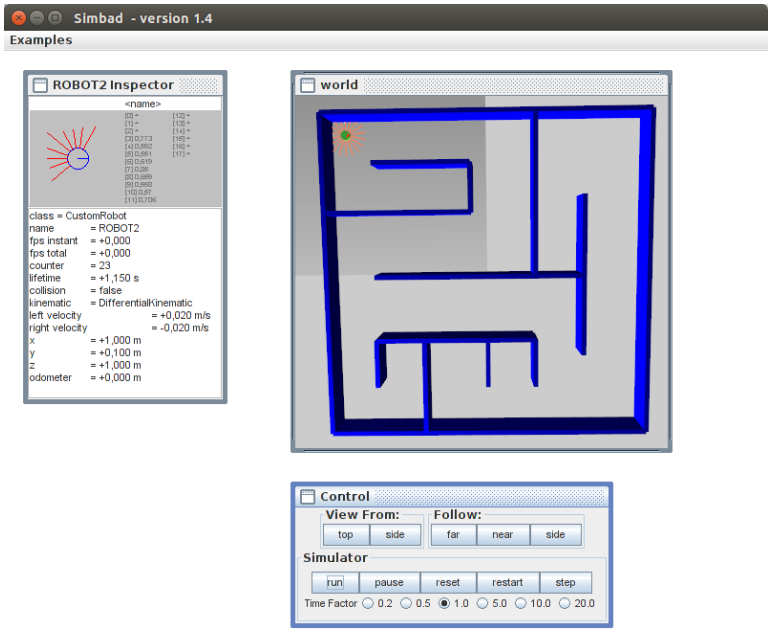

Ilustración 4 Simulador Simbad.

Sin embargo cuando se intentó implementar un algoritmo para mapear el mundo, (el propio simulador no permite realizar el mapeado) queda a la vista que la potencia del simulador no es suficiente para afrontar el problema, obteniendo lecturas bastante malas y teniendo problemas con el funcionamiento de la aplicación que se trababa regularmente. Como se puede observar en la Ilustración 5 el mapa de obstáculos generado tras 500 iteraciones de la simulación es erróneo y poco preciso.

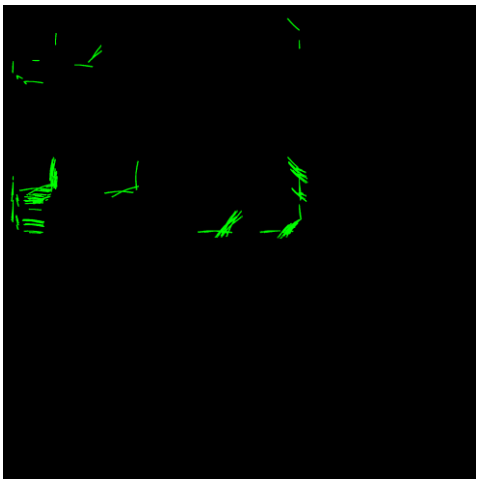

Ilustración 5 Mapa de obstáculos simbad.

2.2.3 Ventajas e inconvenientes

Ventajas:

- La API está bien documentada y el código es legible y fácil de entender.
- Está programado en Java sin muchas dependencias externas, por lo que es relativamente fácil adaptar y mejorar el código.

Inconvenientes:

- La API no es muy completa, hay muchas funciones que serían útiles y no están implementadas.

- El simulador permite entornos 3D pero son muy básicos, y los movimientos de cámara son muy limitados.

- Al simular algoritmos complejos se queda trabado con frecuencia o algunas funciones se comportan de manera inesperada.

2.3 VREP

V-REP es un simulador altamente personalizable: cada aspecto de la simulación se puede personalizar. Además, del propio simulador que se puede personalizar y adaptar con el fin de que se comporte de la manera deseada. Esto es posible gracias a una API (Application Programming Interface) muy elaborada. Se admiten seis enfoques de programación diferentes, cada uno con ventajas particulares (y, obviamente, también desventajas) sobre los demás, pero los seis son compatibles entre sí (es decir, se pueden utilizar al mismo tiempo, o incluso conjuntamente). Vrep puede ser programado en C/C++, Phyton, Java, Matlab, Octave, y LUA. Y además puede ser controlado a través de una inerfaz para ROS.

2.3.1 Manejo del simulador

El control de la escena y los agentes se pueden realizar de las siguientes maneras:

Una secuencia de comandos incrustada: Este método, que consiste en escribir guiones LUA, es muy fácil y flexible, con compatibilidad garantizada con todas los demás instalaciones por defecto en V-REP. Este método permite la personalización de una simulación completa, desde la escena hasta el propio simulador, por ejemplo, con este método puedes añadir nuevos paneles o ventanas al simulador. Este es el enfoque de programación más fácil y más utilizado.

Un add-on: Este método, que consiste en escribir guiones LUA, permite personalizar rápidamente el propio simulador. Los complementos pueden iniciarse automáticamente y se ejecutan en segundo plano, o pueden ser llamados como funciones. Los Add-ons no deben ser específicos para un determinado modelo de simulación sino que deberían más bien ofrecer una funcionalidad más genérica.

Un plug-in: Este método consiste básicamente en escribir un plugin para V-REP. A menudo, los plugins sólo se utilizan para proporcionar una simulación con los comandos LUA personalizados, y así se utilizan de manera conjunta al primer método. Otras veces, los plugins se utilizan para proporcionar a V-REP una funcionalidad especial que requiere la capacidad de cálculo rápido (los generalmente más lentos que lenguajes compilados), una interfaz específica a un dispositivo de hardware (por ejemplo, un robot real), o una interfaz especial de comunicación con el mundo exterior.

Una API remota cliente: (Este método permite una aplicación externa conectarse a V-REP de manera muy fácil, usando comandos remotos. Esto hace que sea posible tener exactamente el mismo código situado en

un robot y en el simulador VREP.

Un nodo de ROS: Este método permite una aplicación externa conectarse a V-REP a través de ROS, un conjunto de librerías y herramientas que te ayudan a construir aplicaciones para robots

Un cliente/servidor personalizado: La aplicación de cliente/servidor tiene que trabajar junto a VREP, ya sea mediante una secuencia de comandos o un plugin, con algún medio de comunicación determinado la comunicación (por ejemplo, sockets, pipes, etc.). Esta es la manera más flexible de trabajar, aunque conlleva mucho más esfuerzo que las anteriores.

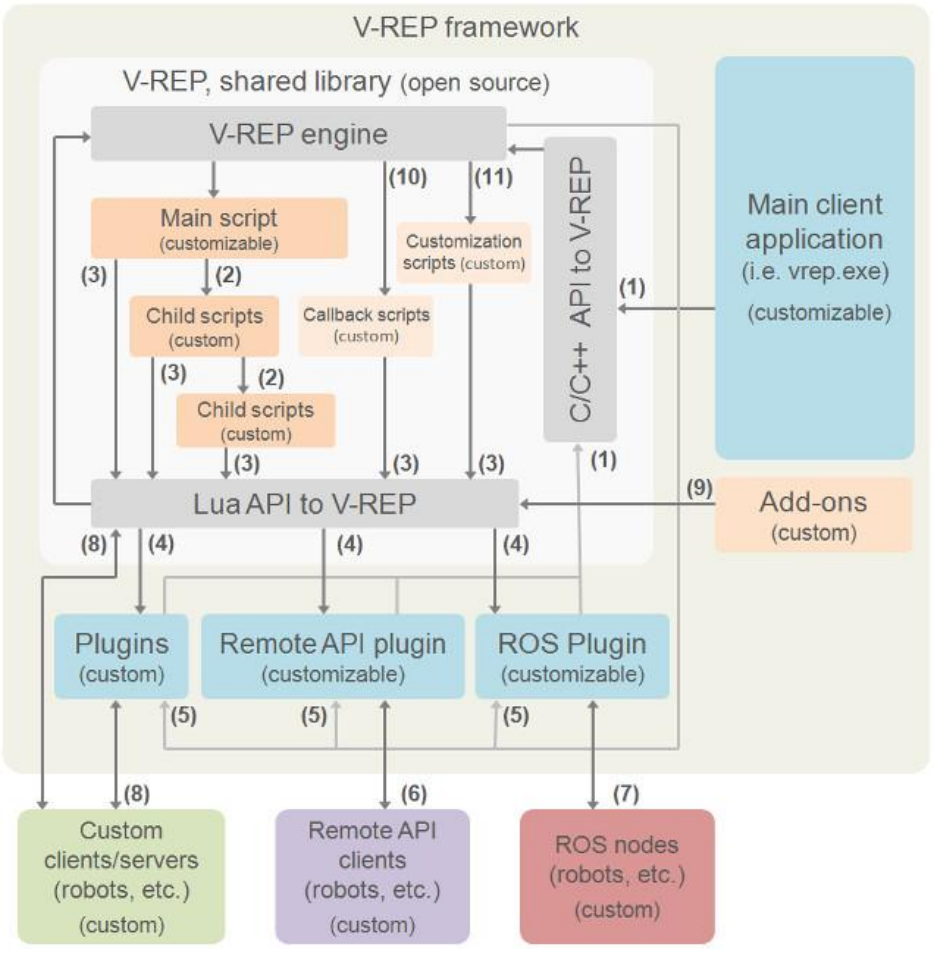

Ilustración 6 Funcionamiento de VREP.

2.3.2 Principales características

VREP dispone de un motor para interpretar y simular condiciones reales incluyendo físicas complejas. También dispone de funcionalidades para hacer cálculos de cinemática tanto directa como inversa. Ofrece un motor para gestionar las colisiones y simulaciones de partículas. Este punto es muy interesante para simular escenas como robots que pintan o el efecto que genera el viento producido por una hélice sobre algún objeto.

Es capaz de simular una gran cantidad de sensores, y en especial destaca por su capacidad de simular sensores de visión con filtros de procesamiento de imagen que pueden ser personalizados y extendidos.

Tiene un motor para la generación de rutas y la planificación de movimientos.

Permite la ampliación del simulador mediante interfaces de usuario fácilmente personalizables.

Simula un escenario 3D de gran complejidad permitiendo importar modelos propios. Además, permite alterar la escena mientras la simulación está en marcha.

2.3.3 Ventajas e inconvenientes

Ventajas:

- La API tiene una documentación muy extensa y es bastante completa.
- Permite la programación en diversos lenguajes.
- Permite la simulación en entornos 3D complejos y en varios medios, como aire, agua...
- Tiene un foro soporte atendido por profesionales.
- Es un programa multiplataforma.

Inconvenientes:

- Para entornos complejos con muchos robots las simulaciones pueden ir lentas.

2.4 Elección del simulador

Después de analizar los diferentes simuladores y sus características, se ha optado por escoger VREP para desarrollar el proyecto. Esta decisión se ha tomado en base a múltiples motivos.

En primer lugar se descartó Player/Stage debido a la gran dificultad que conlleva su instalación y puesta en marcha de una manera estable.

Posteriormente, tras elegir simbad como nuevo simulador, y estar testeándolo y haciendo programas de prueba durante varios días, se puede afirmar que no es lo suficiente potente para realizar las tareas que se disponen en este proyecto, y por lo tanto se debe encontrar un simulador más potente.

En este momento es en el que se decide trabajar con VREP, debido a su facilidad de instalación, a su extensa documentación y su soporte, su capacidad para simular entornos 3D complejos y a la gran cantidad de sensores que permiten simular un robot real de una manera fiel y efectiva.

Capítulo 3. Desarrollo del proyecto.

3.1 Diseño del entorno

El entorno en el que se van a encontrar los robots va a ser un entorno limitado mediante paredes, es decir, se van a encontrar dentro de una habitación. En esta habitación se van a encontrar una serie de obstáculos, sillas, mesas, personas...

Según el problema que estemos tratando el robot dispondrá de determinada información sobre el entorno, que se irá detallando en el desarrollo de cada subproblema.

3.2 Diseño del robot

Se ha decido simular un robot real, con el fin de poder implementar estos algoritmos en dicho robot en un futuro si fuera necesario.

En concreto se ha optado por el pioneer3dx ya que es uno de los más populares a nivel mundial, es bastante fiable y compacto. Puede realizar odometría mediante los codificadores de sus ruedas y tiene 16 sensores de ultrasonidos rodeando su chasis. Funciona mediante baterías y se le pueden añadir otros sensores de una manera cómoda debido al espacio libre que tiene la parte superior.

Ilustración 7 Pioneer p3dx

Puede alcanzar velocidades de 1.6 metros por segundo y soporta una carga de hasta 17 kilos. Dispone de un SDK propio para ser programado, lo que facilita la tarea que supone portar el código del simulador elegido al robot.

3.3 Hito 1: Diseñar e implementar un robot capaz de seguir una ruta.

3.3.1 Especificaciones del problema

El robot simulado debe de ser capaz de seguir una serie de puntos que se le han establecido.

Para este problema el robot tiene conocimiento de su posición en todo momento (Se considera una odometría perfecta). Además se supone que la superficie por la que se desplaza es completamente plana y la única fuerza que mueve el robot es la ejercida por el movimiento de las ruedas.

Se deben considerar que el robot es no-holonómico, por lo que no puede desplazarse en cualquier dirección en cualquier momento, de manera que puede desplazarse hacia delante o detrás pero para ir hacia los lados debe rotar.

Además se debe conocer el radio de las ruedas y la distancia entre las mismas.

3.3.2 Desarrollo

Para conseguir el objetivo, el robot debe definir qué velocidad aplicar a cada rueda (la izquierda y la derecha) con el fin de que la velocidad y la orientación final sean las adecuadas para alcanzar cada punto de la ruta.

Por lo tanto hay que considerar dos puntos principales, la velocidad lineal, y la angular. Ambas se pueden obtener con los datos disponibles.

$$v = \frac{x}{t}$$

Donde v es la velocidad lineal, R es el radio de las ruedas, V_d es la velocidad del motor derecho, V_i es la velocidad del motor izquierdo

$$\omega = R\frac{V_d - V_i}{L}$$

Donde ω es la velocidad angular y L es la distancia entre las dos ruedas.

Para conseguir seguir la ruta es necesario definir la cinemática inversa del robot, es decir, hay que encontrar un modelo matemático que permita obtener las velocidades para cada rueda a partir de las velocidad lineal y angular.

Para ello se despejan las variables de interés:

$$\begin{bmatrix}\omega_d \\ \omega_i\end{bmatrix} = \begin{bmatrix} R\frac{\cos\theta}{2} & R\frac{\cos\theta}{2} \\ R\frac{\sin\theta}{2} & R\frac{\sin\theta}{2} \\ \frac{R}{L} & -\frac{R}{L}\end{bmatrix}^{-1} \begin{bmatrix}dx \\ dy \\ d\varphi\end{bmatrix}$$

El primer problema que se encuentra es que hay que realizar el cálculo de la inversa de una matriz que no es cuadrada, aunque se soluciona hallando la matriz pseudoinversa.

$$A^{+}\begin{bmatrix}\frac{\cos\theta}{R} & \frac{\sin\theta}{R} & \frac{L}{R} \\ \frac{\cos\theta}{R} & \frac{\sin\theta}{R} & \frac{-L}{R}\end{bmatrix}$$

Una vez resuelto este problema se puede definir la cinemática del robot como:

$$\begin{bmatrix}\omega_d \\ \omega_i\end{bmatrix} = \begin{bmatrix}\frac{\cos\theta}{R} & \frac{\sin\theta}{R} & \frac{L}{R} \\ \frac{\cos\theta}{R} & \frac{\sin\theta}{R} & \frac{-L}{R}\end{bmatrix}\begin{bmatrix}dx \\ dy \\ d\varphi\end{bmatrix}$$

Como resultado se obtiene la expresión matricial que permite calcular la velocidad que debe tener cada rueda.

3.4 Hito 2: Diseñar e implementar un robot capaz de evitar colisiones.

3.4.1 Especificaciones del problema

El robot debe ser capaz de evitar obstáculos que se encuentren en su camino (no planificados), por ejemplo un objeto que se ha colocado después de planificar la ruta o evitar obstáculos que se aproximen hacia su posición, como por ejemplo una persona caminando.

Para este problema se establecen las mismas condiciones que para el problema anterior.

3.4.2 Desarrollo

En primer lugar se debe establecer la técnica para evitar los obstáculos, para este caso se ha elegido por el algoritmo del explorador Braitenberg. Este comportamiento se basa en evitar obstáculos de manera que se navegue hacia los lugares con menor densidad de obstáculos, es decir espacios libres.

Este algoritmo se basa en los sensores de ultrasonido del robot, de manera que se le asigna un peso a cada sensor, haciendo que el robot pueda reaccionar de manera diferente a los obstáculos que encuentra según el lugar relativo al robot en el que se encuentren.

Para este problema se han seleccionado los siguientes pesos:

```
    braitenbergL={-0.2,-0.4,-0.6,-0.8,-1,-1.2,-1.4,-1.6,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0}
    braitenbergR={-1.6,-1.4,-1.2,-1,-0.8,-0.6,-0.4,-0.2,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0}
```

De esta manera sólo tenemos en cuenta la mitad de los sensores ya que nuestro robot va a navegar principalmente hacia delante, por lo que será poco probable que encontremos un objeto a por la parte trasera. No obstante se le podrían dar unos pequeños valores para contemplar el caso en el que otro vehículo o un humano que pueda envestir por detrás al robot.

Una vez seleccionado los pesos se deben hacer mediciones de todos los sensores para poder calcular la contribución de cada sensor al movimiento final del robot.

```
  for i=1,16,1 do
        res,dist=simReadProximitySensor(usensors[i])
        if (res>0) and (dist<noDetectionDist) then
            proxSensDist[i] = dist
            if (dist<maxDetectionDist) then
                dist=maxDetectionDist
            end
            detect[i]=1-((dist-
maxDetectionDist)/(noDetectionDist-maxDetectionDist))
        else
            detect[i]=0
        end
  end
```

La contribución de cada sensor es la siguiente:

$$1 - \frac{Dist - MaxDist}{MaxRange - MaxDist}$$

Donde "Dist" es la distancia al obstáculo encontrado, "MaxDist" es la distancia más cercana a la que el sensor puede leer y "MaxRange" es la distancia más lejana a la que el sensor puede leer.

Una vez se tienen las contribuciones de todos los sensores deben ser utilizados para definir la velocidad final de las ruedas. Esta velocidad se define según las siguientes fórmulas:

$$\omega_d = V_{Crucero} + \sum_{i=0}^{numeroSensores} (\text{braitenbergR[i]} * \text{detect[i]})$$

$$\omega_i = V_{Crucero} + \sum_{i=0}^{numeroSensores} (\text{braitenbergL[i]} * \text{detect[i]})$$

Lo que en el código se traduce a:

```
for i=1,16,1 do
      v_l=v_l+braitenbergL[i]*detect[i]
      v_r=v_r+braitenbergR[i]*detect[i]
end
```

Suponiendo que en las variables v_l, y v_r se encontrara previamente establecida la velocidad de crucero.

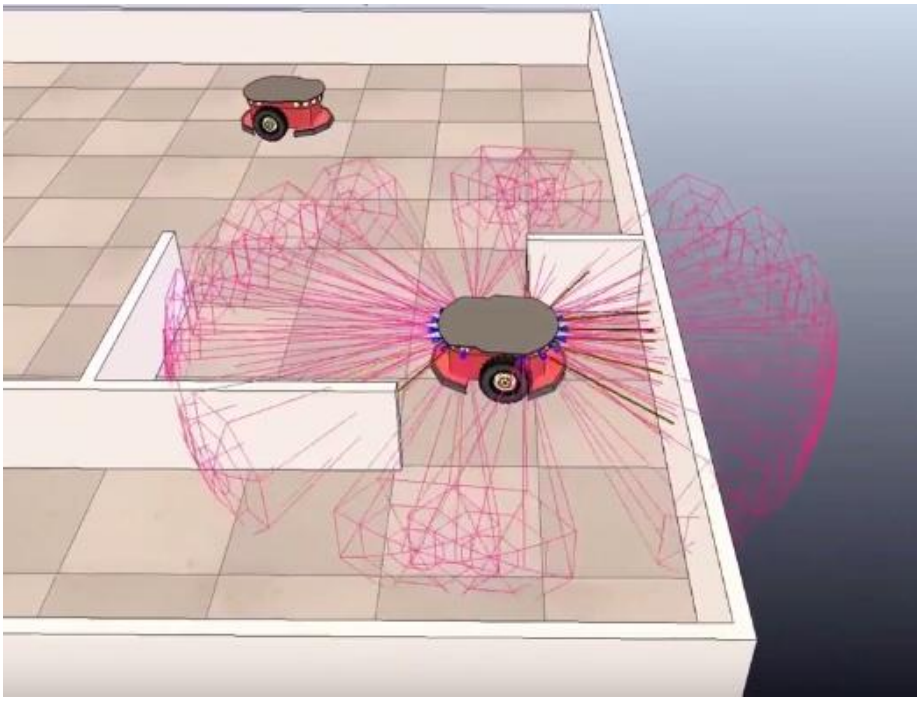

Ilustración 8 Algoritmo Baitenberg.

3.5 Hito 3: Diseñar e implementar un robot capaz de diseñar y seguir una ruta evitando colisiones.

3.5.1 Especificaciones del problema

Para este problema se establecen todas las condiciones explicadas anteriormente, y además se determina que el robot tiene conocimiento en todo momento de el lugar en el que se encuentra, es decir, tiene un mapa (idealmente perfecto) del entorno y puede consultarlo en cualquier momento.

3.5.2 Desarrollo

Teniendo el mapa del entorno, hay varias maneras de calcular rutas hacia un punto establecido mediante una gran cantidad de algoritmos como pueden ser A*, búsquedas en profundidad, o el que se va a utilizar para el problema RRT.

El algoritmo RRT(Rapidly Exploring Random Trees) sirve para hacer búsquedas de manera eficiente en espacios no convexos de altas dimensiones. Funciona construyendo un árbol aleatorio que va rellenando el espacio vacío.

Es bastante útil para problemas en los cuales hay que evitar obstáculos y existen limitaciones diferenciales, como es en el caso de nuestro robot no holonómico.

El árbol generado por el algoritmo se construye pseudoaleatoriamente a partir de una serie de muestras, este árbol tiende a expandirse hacia espacios no ocupados. Este fenómeno tiene lugar debido a que la probabilidad de expandir el árbol a un lugar es proporción al tamaño de su región de Voronoi.

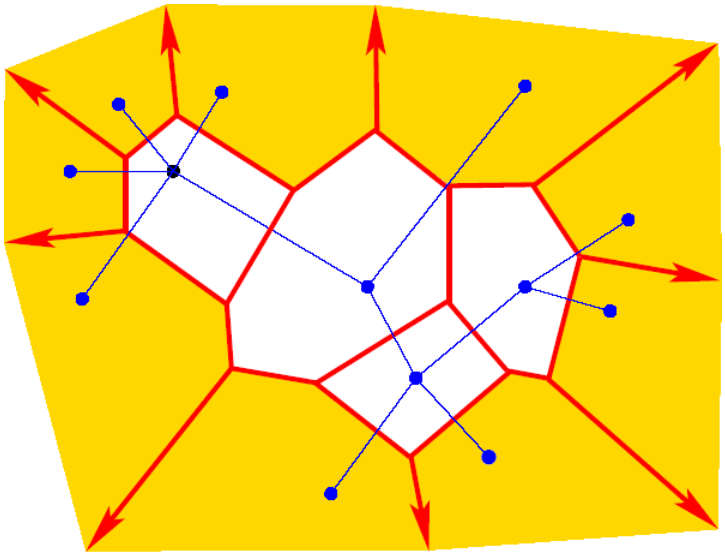

Ilustración 9 Región de Voronoi para el RRT.

Como se comentó anteriormente, VREP tiene un motor para la generación de rutas, que simplifica enormemente este trabajo.

Una vez generada la ruta, queda el problema de evitar posibles obstáculos dinámicos no contemplados al generar la ruta que puedan encontrarse en el camino, y que habría que esquivar.

Para solucionar este problema se ha implementado un sistema de modos en el robot. Gracias a este mecanismo el robot puede decidir si debe seguir la ruta, cuando no existen obstáculos, o esquivarlos en casa de que exista alguno.

Para establecer el modo de comportamiento del robot se toma como referencia las mediciones del sensor de ultrasonidos, si hay algún obstáculo que esté más cerca del robot que la distancia de seguridad se entra en el modo de evitar obstáculos, en otro caso se sigue la ruta planificada.

3.6 Hito 4: Diseñar e implementar un robot capaz de comunicarse con otros robots.

3.6.1 Especificaciones del problema

Para este problema se establecen todas las condiciones explicadas anteriormente. El robot intentará comunicarse con otro que entre en el rango de su antena, con el fin de que no interfieran en sus rutas.

3.6.2 Desarrollo

La primera tarea que hay que hacer para poder solucionar este problema es añadirle un mecanismo de comunicación al robot, ya que de por si no lo tiene. Por lo tanto se le ha añadido una antena en la parte superior con un rango de acción variable que puede ser definido al principio de cada simulación. Además se le ha añadido al robot un pequeño LED que se utilizará para comprobar visualmente que acción está realizando el robot.

Una vez instaladas las antenas a cada robot se debe definir un modelo para evitar que interfieran en sus rutas. para este problema cada robot tendrá asignada una prioridad, de manera que el robot con mayor prioridad recalculará su ruta teniendo en cuenta al otro robot, y el otro robot esperará en el lugar en el que se encuentra

hasta que el robot con mayor prioridad haya salido de su radio de acción.

Para el envío de mensajes entre robot se utiliza un protocolo de comunicación interno de VREP mediante los métodos.

```
simReceiveData(0, "c", antenna_handle)

simSendData(sim_handle_all, 0, "c",  "MESSAGE",
                            antenna_handle, ANTENNA_RANGE)
```

El protocolo que se utiliza el siguiente, ambos robots están enviando mensajes con su prioridad en cada iteración, en caso de que alguno reciba un mensaje, comprueba la prioridad del otro robot, si es mayor que la propia, se pone en modo espera encendiendo el led rojo y manda un mensaje de confirmación al otro robot.

En caso de que el mensaje recibido lleve una prioridad menor que la del propio robot, este lo ignora.

Si el mensaje recibido tiene una etiqueta de confirmación, el robot sabe que tiene prioridad sobre el otro, que le ha confirmado que está esperando, por lo que es libre para planificar su nueva ruta, y continuar sin problemas.

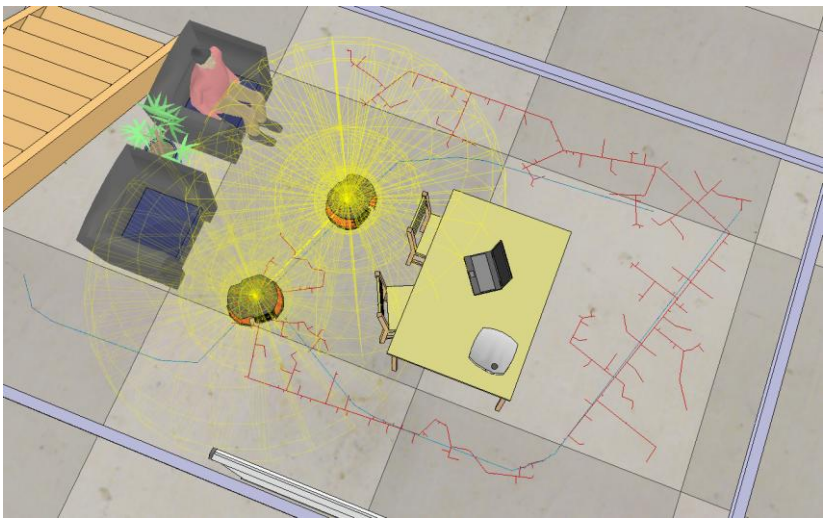

Ilustración 10 Comunicación entre robots.

3.7 Hito 5: Diseñar e implementar un robot capaz de mapear el entorno en el que se encuentra.

3.7.1 Especificaciones del problema

Para este problema se establecen todas las condiciones explicadas anteriormente. El robot recopilará datos de su entorno para formar un mapa de la escena en la que se encuentra. Además el mapa que realizará será en 3D de manera que represente más fielmente la realidad.

3.7.2 Desarrollo

Lo primero que hay que hacer para resolver este problema es dotar al robot de un sensor que permita recopilar datos en 3D. Como los sensores de ultrasonidos son fijos, no valen para esta tarea, por lo que se simulará un sensor laser Hokuyo con un motor que permita hacer mediciones en varias direcciones.

Ilustración 11 Sensor laser Hokuyo.

VREP no simula de manera directa sensores láser, pero sí que simula sensores de visión que pueden obtener la profundidad de la imagen. Se puede aprovechar esto para simular el sensor láser a partir de un sensor de visión.

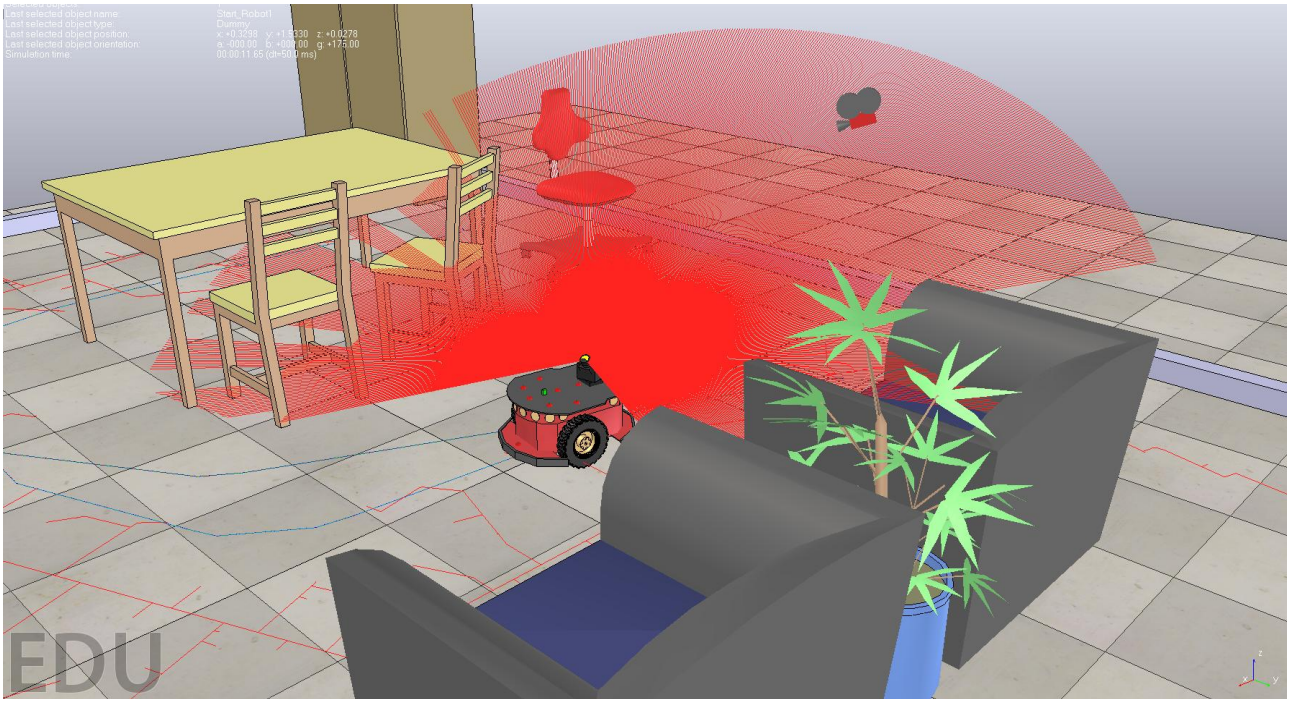

Ilustración 12 Sensor láser VREP.

Para ello se leen los datos del sensor de visión, si existen debemos ajustar esos puntos a la posición en el que se encuentra el sensor utilizando matrices de transformación. Por último sólo se deben añadir los puntos sin están más cercanos que la distancia de lectura máxima del sensor.

Para cada punto:

```
if (v4<maxScanDistance_) then
        p={v1,v2,v3}
        p=simMultiplyVector(m01,p)
        table.insert(measuredData,p[1])
        table.insert(measuredData,p[2])
        table.insert(measuredData,p[3])
end
```

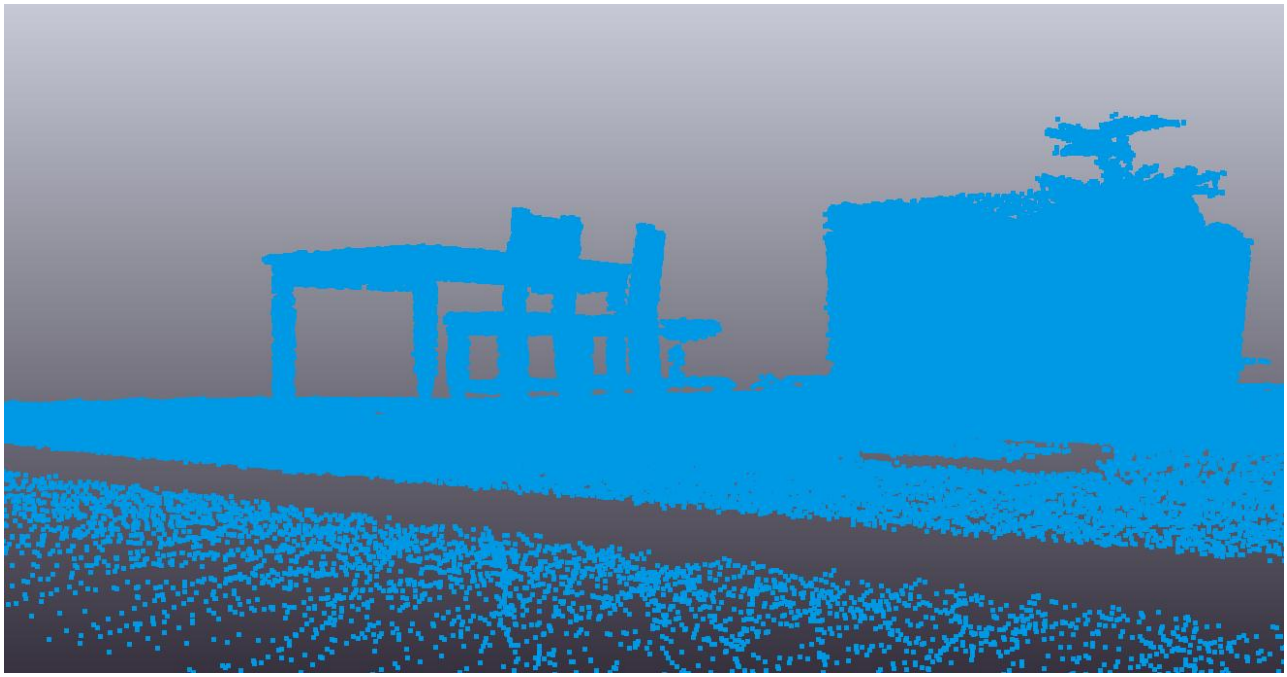

Ilustración 13 Mapeado de puntos VREP.

Como se puede observar en las imágenes el mapeado es bastante bueno, obteniendo un punto por cada posición que detecta el sensor laser del robot.

3.8 Hito 6: Diseñar e implementar un robot que sea capaz de comunicarse con otros robots para compartir información sobre el entorno.

3.8.1 Especificaciones del problema

Para este problema se establecen todas las condiciones explicadas anteriormente. En este caso, en vez de un único robot habrán varios que aportaran la información recopilada a un mapa conjunto para ambos, por lo que se supone que en todo momento los robots se están comunicando entre sí.

3.8.2 Desarrollo

Para solventar este problema no hace falta añadir componentes diferentes a los que el robot tenía en versiones anteriores, simplemente hay que modificar el sensor para que en vez de enviar la información al robot la envíe a un mapa global.

```
modelHandle=simGetObjectAssociatedWithScript(sim_handle_self)
objName=simGetObjectName(modelHandle)
communicationTube=simTubeOpen(0,objName..'_SHARED_HOKUYO',1)
```

Así los puntos que vayan descubriendo los robots pueden ser consultados por ambos a la hora de realizar el mapa. Para poder distinguir que puntos ha mapeado cada robot se les asigna un color en la imagen. Como se puede ver en la Ilustración 14 Mapeado colaborativo VREP los puntos mapeados por un robot se representan en azul, y los mapeados por el segundo en verde.

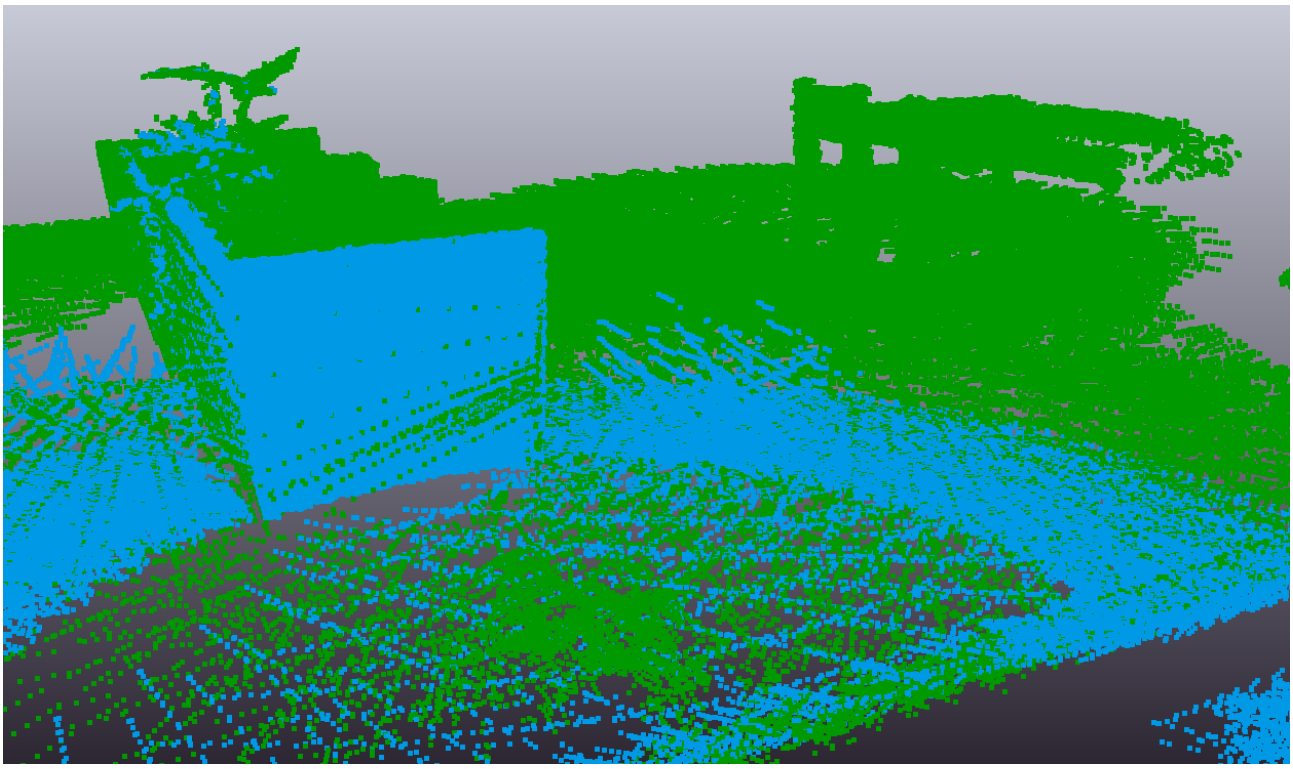

Ilustración 14 Mapeado colaborativo VREP

3.9 Hito 7: Diseñar e implementar un robot que sea capaz de aprovechar su propio mapa del entorno para desplazarse en el mundo real.

3.9.1 Especificaciones del problema

Este problema cambia algunas de las condiciones anteriores. El robot ya no tiene acceso al mapa global, únicamente tiene acceso a su mapa de obstáculos propio.

3.9.2 Desarrollo

Para solucionar este problema el robot calcula las rutas a partir de su mapa de obstáculos (inicialmente vacio) y una vez detecta algún obstáculo por los sensores de ultrasonido, recalcula la ruta a partir de los nuevos obstáculos que haya ido descubriendo gracias a su sensor laser.

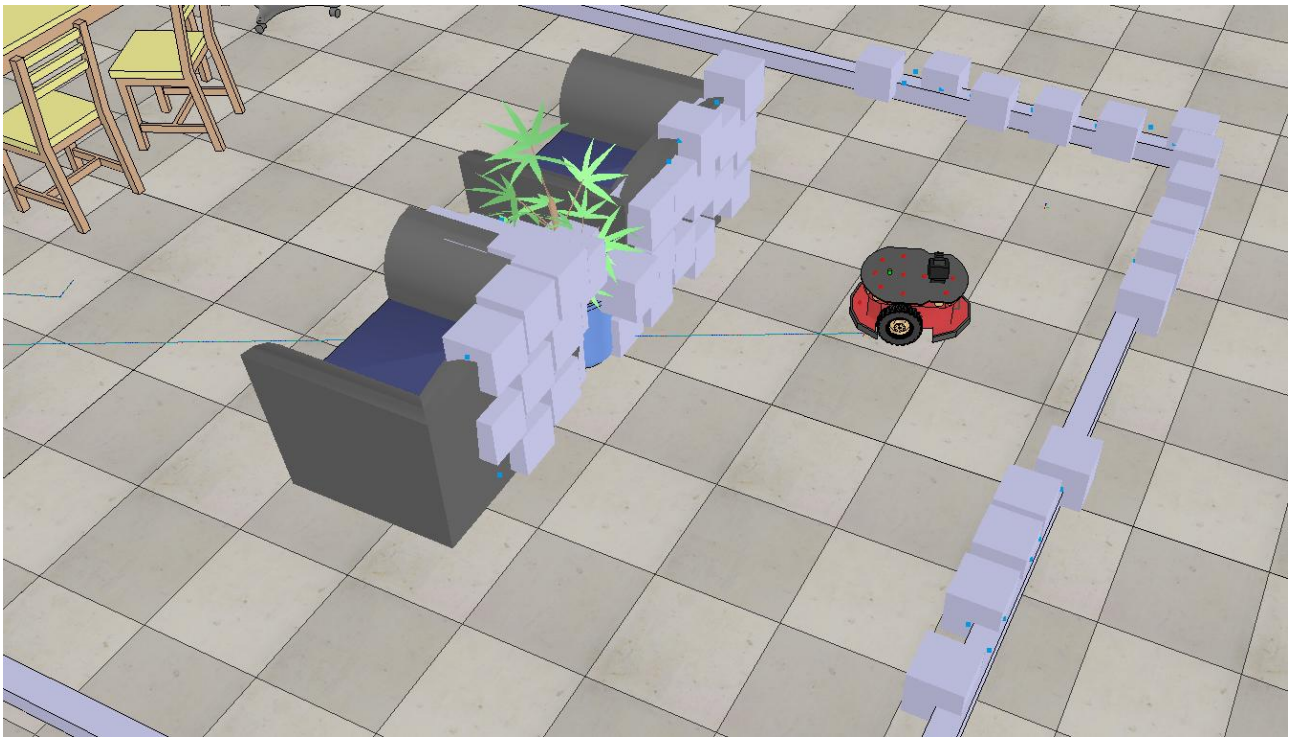

Ilustración 15 Generación de rutas con mapa vacío.

Este método tiene un problema, y es que en un entorno con un mínimo de obstáculos, se van a mapear demasiados puntos, haciendo que el robot no pueda procesar tanta información. Cómo solución se ha optado por discretizar el entorno, es decir, tomar como obstáculos puntos cuando más grandes mejor, pero como máximo del tamaño del robot, ya que si se eligieran tamaños más grandes podrían estarse tapando rutas que el robot realmente podría seguir. Además se evitaran mapear los puntos del suelo ya que ocupan espacio en el mapa del robot y hagan que los cálculos sean más lentos.

VREP utiliza objetos y no puntos para el cálculo de rutas, habrá que crear un objeto por cada punto mapeado, respetando las restricciones del párrafo anterior.

Como se puede apreciar en la Ilustración 15 Generación de rutas con mapa vacío. la ruta inicial del robot es directa hacia su objetivo, ya que cuando la calculó no tenia obstáculos en su mapa, una vez empieza a seguir la ruta y detecta un obstáculo con el sensor de ultrasonido, recalcula la ruta utilizando los nuevos obstáculos mapeados por el sensor laser, consiguiendo así una nueva ruta que le acerca más a su destino.

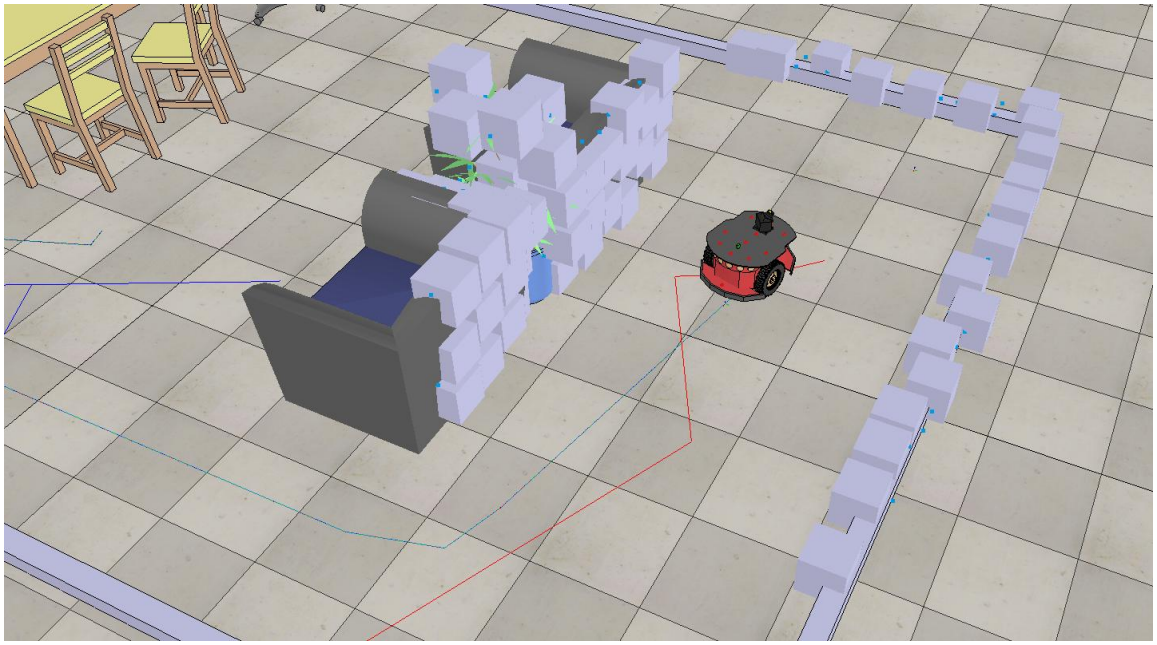

Ilustración 16 Generación de rutas con obstáculos.

Capítulo 4. Conclusiones y líneas futuras

La robótica en general es un campo de estudio que se encuentra en auge actualmente, una gran cantidad de empresas y universidades investigan para avanzar y desarrollar las técnicas que existen actualmente. Este proyecto no sólo me ha servido para ampliar mis conocimientos de robótica, aprender nuevas técnicas, y descubrir el mundo de los simuladores, que facilitan enormemente el diseño e implementación de algoritmos. Además me ha dado la motivación suficiente como para querer seguir desarrollando el proyecto, ideando nuevas técnicas, mejorando las existentes y probando otro tipo de vehículos como los drones, para los que hay muchos problemas que resolver y afortunadamente, están soportados por el simulador. Entre las técnicas que me gustaría desarrollar se encuentras los problemas de mapeado y localización simultáneos o SLAM, ya que no se han visto técnicas de localización en el proyecto y es un tema que me parece muy interesante.

Desde seguir una simple ruta, hasta crear rutas a partir de un mapa generado por el propio robot he ido pasando por una serie etapas que aunque en algunos momento me han hecho querer rendirme, me han dado la motivación suficiente para seguir adelante, ya que una vez resuelves uno, quieres continuar y mejorar lo que tienes, ya que nunca estará todo lo bien que te gustaría.

Además me ha servido para mejorar un poco mis conocimientos sobre electrónica, ya que aunque no se presentó en el proyecto, se realizó un pequeño robot con el fin de implementar estos algoritmos, aunque por falta de tiempo no fue posible llevar la idea a cabo.

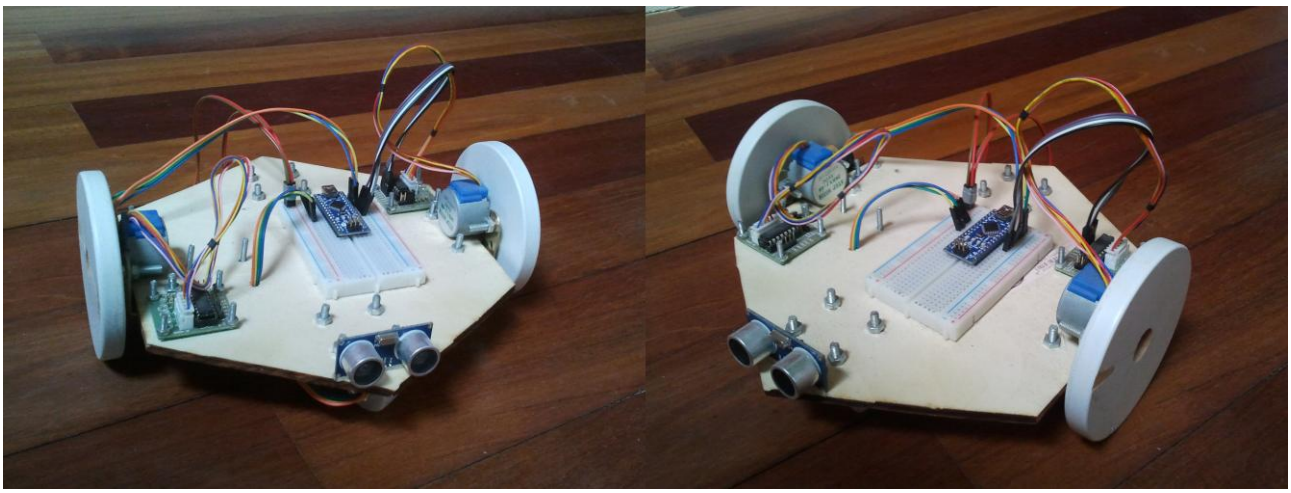

Ilustración 17 Robot para pruebas.

En un futuro me gustaría seguir trabajando sobre este campo, y extender los algoritmos descritos en el proyecto a un Pioneer p3dx real, observando cómo se comporta en un entorno no tan controlado donde hay variables que no están contempladas en el simulador.

Capítulo 5.
Summary and Conclusions

Robotics in general is a field of study that is currently booming, a lot of companies and universities research to develop techniques that currently exist. This project helped me to expand my knowledge of robotics, learn new techniques, and discover the world of simulators, which greatly facilitate the design and implementation of algorithms. He has given me the enough motivation to further develop the project, with new techniques that improves the existing one and testing other vehicles such as drones, for which there are many problems that has not been solved and fortunately, are supported by the simulator. Among the techniques that I would develop the an algorithm of simultaneous localization and mapping or SLAM, because there are a lot of localization techniques that I find very interesting.

I've been going through a series of stages that sometimes makes me want to give up. But once the problem is solved got the enough motivation to keep going, because when you solve one, you want to continue and improve what you have, because you are never as good as you would like.

It has helped me to improve my knowledge of electronics, because I built, a small robot in order to implement the algorithms I coded, but due to time constraints it was not possible to take the idea out.

In the future I would like to continue working on this field, and extend the algorithms described in the project to a real Pioneer Real p3dx, observing how they behave in a less controlled environment where a lot of variables that are not covered in the simulator exist.

Capítulo 6. Presupuesto

El coste del proyecto radica principalmente en el trabajo de los empleados asignados a su desarrollo, debido a que ha sido realizado íntegramente en un simulador. En este caso, el simulador es gratuito para estudiantes, por lo que no supone ningún coste. Teniendo en cuenta que el sueldo en España para un recién titulado corresponde a unos 1200 euros mensuales, y que a cada mes le corresponden 160 horas de trabajo. Se podría decir que el sueldo por hora es de 7.5 euros. El tiempo medio de ejecución del trabajo ha sido de unas 300 horas para una sola persona, por lo que entre dos deberían hacerlo en la mitad de tiempo y de una manera más eficiente.

Personal	Horas	Precio
Ingeniero Informático	150h	1125€
Ingeniero Informático	150h	1125€
TOTAL	**300h**	**2250€**

Tabla 2 Presupuesto de personal.

En caso de que se quisiera implementar el sistema en robots reales el presupuesto sería diferente. Ya que habría que adquirir mínimo dos robots y dos sensores laser. Además del tiempo de desarrollo para programar dichos robots.

Personal	Cantidad	Precio
Robot pioneer p3dx	2 u	8000€
Sensor laser Hokuyo	2 u	1700€
TOTAL		**9700€**

Tabla 3 Presupuesto hardware.

Por lo tanto se pueden establecer dos tipos de presupuesto:

Sólo el programa simulado, cuyo precio final total sería de 2.250€.

Implementación del sistema real, cuyo precio final total sería de 11.950€.

Bibliografía

[1] Simbad Guide.

http://simbad.sourceforge.net/guide.phpHOYUKO.

[2] Hoyuko sensors. https://www.hokuyo-aut.jp/

[3] Seguimiento de trayectorias con un robot móvil de configuración diferencial.

http://web.usbmed.edu.co/usbmed/fing/v5n1/v5n1a3

[4] Foro Coppeliarobotics. www.forum.coppeliarobotics.com/

[5] Player-Stage based simulator for simultaneous multi-robot exploration and terrain coverage problem. http://airccse.org/journal/ijaia/papers/1011ijaia10.pdf

[6] V-REP: A versatile and scalable robot simulation framework. http://www.researchgate.net/publication/261352390_V-REP_A_versatile_and_scalable_robot_simulation_frameworkP,

[7] Pioneer P3-DX.

http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx