ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA EN SISTEMAS AUDIOVISUALES Y MULTIMEDIA

TRABAJO FIN DE GRADO

Nuevas Prácticas en el Entorno Docente de Robótica JdeRobot-Academy

Autora: Irene Lope Rodríguez

Tutor: José María Cañas Plaza

Curso académico 2017/2018

Agradecimientos

En primer lugar, quiero darle las gracias a mi familia por todo su interés y apoyo. En especial a mis padres, por estar a mi lado en todo momento, apoyarme en todas mis decisiones y darme todo su cariño. Gracias por todo lo que habéis hecho por mi a lo largo de toda mi vida.

También, quería agradecer a mi tutor José María la oportunidad de realizar este proyecto así como toda su ayuda, dedicación e interés durante los meses de desarrollo del trabajo.

Por otro lado, me gustaría darle las gracias a mi compañera proyecto Vanessa por toda su ayuda y ánimo cuando las cosas parecían que no salían y por ser un gran apoyo en estos meses de trabajo. Además, quería darle las gracias a mis compañeros de laboratorio, en especial a Fran por su infinita ayuda, a Aitor por sacarnos de más de un atasco y por su puesto, a Nacho y Carlos por los buenos ratos que nos han hecho pasar.

Muchas gracias a mis amigos, a los de siempre: Ayrton, Emilio, Ana, Jumana, y en especial a mi mejor amigo Pablo, por animarme cuando más lo necesitaba, por hacerme reír cada día y por seguir a mi lado durante tantos años. Y no me puedo olvidar de mis amigas de la universidad: Carol, María, Chris y Natalia, por hacer que las clases fueran menos duras y las horas de estudio más divertidas.

Y por último, mi más sincero agradecimiento a una de las personas más importantes de mi vida, a mi pareja Carlos, porque sin él, habría tirado la toalla hace mucho tiempo. Simplemente, gracias por todo.

¡Muchas gracias a todos!

Resumen

La robótica es una rama de la ingeniería que emplea la informática para diseñar y desarrollar sistemas y máquinas que faciliten la vida del ser humano, e incluso sustituirle en determinadas tareas. Se pueden encontrar robots en diferentes áreas y entornos como la industria, la automovilística, la medicina o la domótica, entre otros. Debido a que actualmente la robótica está en continua expansión, es necesario formar a personas para que sean capaces de programar estos robots.

El objetivo de este proyecto es la creación de dos nuevas prácticas en el entorno educativo JdeRobot-Academy. Estas prácticas están programadas en el lenguaje Python y se ha hecho uso del simulador Gazebo para realizar las distintas pruebas hasta conseguir un resultado final satisfactorio. También se ha utilizado la librería OpenCV para el tratamiento digital de las imágenes y la herramienta PyQt5 para el desarrollo de la interfaz gráfica de cada una de las prácticas.

La primera práctica, llamada *"Coche autónomo negocia un cruce"*, trata de conseguir que un coche autónomo negocie con éxito un cruce. Primero, que sea capaz de reconocer una señal de STOP. Una vez detecte la señal, el coche tendrá que frenar en un cruce y detectar si hay otros coches circulando por la carretera. En el caso de que no aparezcan coches, deberá realizar un giro a la izquierda o a la derecha y continuar su camino. Para ello, se utilizarán tres cámaras de vídeo instaladas en el robot (una en el techo y dos en los faros del coche).

La segunda práctica, *"Aspiradora autónoma con autolocalización"*, consiste en que una aspiradora autónoma barra la mayor superficie posible de un apartamento. Dicha aspiradora posee el mapa de la casa y tiene instalado un sistema de autolocalización que le permite saber en todo momento cual es su ubicación en el mundo.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este capítulo se definirá el contexto en el cual se sitúa este proyecto así como la motivación principal que ha llevado a desarrollarlo. Se explicará de forma general qué es la robótica, sus aplicaciones y algunas piezas software utilizado para la programación de robots. También se expondrá su uso en docencia hoy en día y se comentará de manera general la plataforma JdeRobot-Academy en la que se encuadra este Trabajo Fin de Grado (TFG).

1.1. Robótica

La robótica es una rama de la ingeniería que emplea la informática para diseñar y desarrollar sistemas que permitan facilitar la vida del ser humano, e incluso sustituirle en determinadas tareas. Esta rama usa conceptos de diversas disciplinas, tales como la física, las matemáticas, la electrónica, la mecánica, la inteligencia artificial o la ingeniería de control. Mediante todas estas disciplinas crea diversas máquinas que ejecutan diferentes comportamientos en función de su propósito. Estas máquinas se denominan "robots". En el futuro, dominar esta disciplina será clave debido a que cada vez de forma más habitual se implantan robots en diferentes empleos.

Los robots pueden tener formas variadas. Según su arquitectura física los robots pueden ser:

- Brazos robotizados: Son robots de base fija, aunque en algunas ocasiones pueden realizar desplazamientos limitados, y mover sus extremidades en un espacio de

trabajo concreto mediante algún sistema de coordenadas y con un limitado número de grados de libertad. Pueden ser muy diferentes en su forma y configuración. Se emplean habitualmente en zonas de trabajo amplias o alargadas. Los robots industriales, manipuladores y cartesianos son algunos ejemplos.

- Robots móviles: Tienen una importante capacidad de movimiento. Son capaces de realizar un cierto desplazamiento, mediante la información que les proporcionan sus sensores del entorno o mediante tele-mando. Suelen tener un sistema locomotor de tipo rodante. Estos robots son capaces de evitar obstáculos y tienen un nivel de inteligencia considerablemente alto. Se suelen emplear para transportar piezas en una cadena de fabricación.
- Humanoides: Estos robots intentan imitar de manera parcial o total la forma y el comportamiento del movimiento humano.
- Zoomórficos: La principal característica de estos robots es su sistema de locomoción, el cual pretende imitar a los distintos seres vivos.

1.1.1. Aplicaciones de la robótica

Actualmente la robótica está en continua expansión. Se pueden encontrar robots en diferentes áreas y entornos. Una de las principales áreas donde se encuentran robots es en la industria. Gracias a los robots se pueden abordar tareas peligrosas y complejas. El robot industrial, debido a su naturaleza multifuncional, puede llevar a cabo un gran número de tareas, totalmente inalcanzable si la mano de obra es humana, de manera que se abarata mucho el coste de producción.

En el mundo del automóvil se han introducido robots tanto para su construcción, utilizando brazos mecánicos en las cadenas de montaje, como para lograr que los coches sean autónomos. Una de las empresas pioneras en este ámbito es Google que junto con Fiat-Chrysler han desarrollado el proyecto “Waymo” (Figura 1.1). Estos vehículos autónomos tienen sensores y software diseñado para detectar personas, ciclistas, vehículos, ciclistas y carreteras a una distancia mayor que dos campos de fútbol en todas las direcciones.

La empresa Tesla también ha desarrollado coches autónomos (Figura 1.1). Sus coches tienen instaladas ocho cámaras que ofrecen una visión de 360 grados alrededor del vehícu-

lo en un área de hasta 250 metros. Además, dispone de doce sensores ultrasónicos capaces de detectar objetos de todo tipo y tamaño alrededor del coche, y de un radar delantero que ofrece datos adicionales.

El objetivo de los coches autónomos es evitar los accidentes de tráfico y disminuir los atascos.

Figura 1.1: Coches autónomos Waymo y Tesla

También se pueden encontrar robots en los hogares de las personas, como por ejemplo aspiradoras autónomas. Estas aspiradoras mediante sensores y el desarrollo de tecnologías robóticas como la construcción de mapas y los sistemas de navegación son capaces de limpiar las casas de una manera fiable y robusta ante distintos tipos de obstáculos como muebles o escaleras. La empresa iRobot es pionera mundial en este sector con su aspiradora Roomba, aunque también se pueden encontrar otras aspiradoras potentes en el mercado como el robot aspirador Dyson (Figura 1.2) que calcula un patrón de modo sistemático y de esa forma sabe por dónde ha pasado y dónde tiene que ir a limpiar.

Amazon, en sus almacenes, también ha introducido robots para que la logística sea mucho más rápida y eficaz (Figura 1.2). Utiliza la automatización para almacenar y retirar los productos. Sus robots son capaces de cargar hasta 1300 Kg y mediante el uso de un láser y una cámara en la parte delantera, son capaces de detectar obstáculos. Se mueven a 1,7 metros por segundo y satisfacen un pedido en 15 minutos en vez de en 70, disminuyendo así el tiempo de entrega de sus productos.

Figura 1.2: Robot aspirador Dyson y robot de Amazon Drive

También se pueden encontrar robots en otra gran variedad de ámbitos:

- Robots médicos: La aplicación fundamental de estos robots se sitúa en el campo de la cirugía. Es fundamental que los diversos brazos robóticos que se emplean en alguna operación quirúrgica sean lo suficientemente precisos. Pueden ser controlados a distancia.

- Robots industriales: Son robots automáticos, reprogramables y con múltiples funciones. Poseen tres o más ejes para poder orientar y colocar en la posición correcta diferentes piezas, materiales, dispositivos o herramientas. Son empleados en la realización de diferentes tareas de la producción industrial en sus diversas etapas. Su entorno de trabajo suele estar controlado, esto hace que las funciones de los robots se simplifiquen de manera notable.

- Robots militares: Estos robots tienen aplicaciones militares concretas, para las cuales pueden actuar de forma autónoma o estar controlados de forma remota. Presentan diferentes morfologías en función de su uso. Asisten o guían al ejército en operaciones especiales. Sus funciones pueden ser la búsqueda, el transporte, el rescate o el ataque.

- Robots educativos: Estos robots se crearon con el fin de emplearse en la enseñanza, especialmente en escuelas e institutos. Los robots educativos de LEGO Mindstorms son un buen ejemplo.

- Robots de servicio: De forma habitual, se emplean para reemplazar al ser humano en entornos no controlados, hostiles y donde puede ser necesario un cambio de forma del robot. Son dispositivos electromecánicos controlados por ordenador y normalmente

dotados de movimiento. Suelen poseer uno o varios brazos mecánicos independientes. No realizan tareas industriales.

- Robots de investigación: son empleados habitualmente en los laboratorios de las Universidades. Están destinados a la investigación y por ello pueden ser de muy diversas formas. Pueden tener un fin concreto en algún proyecto de investigación o no tener ninguna aplicación concreta.

1.2. Software para robots

Además del hardware, el otro ingrediente fundamental de los sistemas robóticos es su software. Mediante el software se le indica al robot las acciones que tiene que realizar, dotándole así de inteligencia y autonomía. Este desarrollo de software suele ser complicado y arduo. En la actualidad, se han propuesto muchos entornos (middleware) y bibliotecas para hacer más fácil la programación de los robots.

1.2.1. Middleware

Los robots autónomos son sistemas complejos que requieren de la interacción entre numerosos componentes heterogéneos. Debido al aumento de la complejidad de las aplicaciones robóticas y la diversa gama de hardware, el middleware robótico está diseñado para gestionar la complejidad y la variedad del hardware y las aplicaciones. Permiten a una aplicación interactuar o comunicarse con otras aplicaciones, sistemas operativos, redes o dispositivos. Los middlewares robóticos proporcionan una API (Interfaz de Programación de Aplicaciones) que simplifica la comunicación entre las aplicaciones robóticas y el hardware subyacente (como los sensores y actuadores), simplificando así el diseño del software y mejorando su calidad.

El middleware más extendido, y que se ha convertido en un estandard de facto es ROS (Robot Operating System) [1]. Es un entorno de programación de código abierto mantenido por la Open Source Robotics Foundation (OSRF). Ofrece una colección de herramientas, bibliotecas y convenciones que tienen como objetivo simplificar la tarea de crear un comportamiento robótico complejo y robusto en una amplia variedad de plataformas robóticas.

[1] `http://www.ros.org/`

Está orientado a sistema UNIX (Ubuntu (Linux)), aunque también se está adaptando a otros sistemas operativos como Fedora, MacOS-X, Arch, Gentoo, OpenSUSE, Slackware, Debian o Microsoft Windows, considerados como "experimentales".

Existen otros middleware como Orca, JdeRobot, Player-Stage, Orocos, etc. pero su ámbito de utilización es más reducido que el de ROS.

1.2.2. Bibliotecas

En informática, una biblioteca es un conjunto de funciones, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca. Su fin es ser utilizada por otros programas.

Una de las bibliotecas de visión artificial utilizadas en robótica es OpenCV [2]. Contiene más de quinientas funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos y reconocimiento facial. Otro ejemplo es PLC [3]. Se utiliza para el procesamiento digital de imágenes mediante el tratamiento de nubes de puntos aleatorios. Contiene numerosos algoritmos de última generación que incluyen filtrado, estimación de características, reconstrucción de superficies y segmentación entre otros.

1.2.3. Simuladores

Los simuladores son herramientas muy utilizadas en robótica debido a que, normalmente, los robots suelen ser costosos. Mediante el uso de estos simuladores se consigue hacer todo tipo de pruebas con los robots sin ningún riesgo de dañar o romper el equipo. Estos simuladores representan de manera realista el entorno y los robots, por lo que es recomendable usarlos antes de probar los programas o aplicaciones creados en robots reales.

Uno de los simuladores más utilizados es Gazebo [4].Es un simulador 3D de código abierto distribuido bajo licencia Apache 2.0. Accede a múltiples motores de física de alto

[2] `http://opencv.org/`
[3] `http://pointclouds.org/`
[4] `http://gazebosim.org/`

rendimiento, incluidos ODE [5] y Bullet [6] entre otros. Al utilizar el motor de renderizado 3D OGRE [7], Gazebo proporciona una representación realista de entornos que incluyen iluminación, sombras y texturas de alta calidad. Permite probar rápidamente algoritmos, diseñar robots, simular cámaras, coches y drones, realizar pruebas de regresión y entrenar sistemas de inteligencia artificial utilizando escenarios realistas. Da soporte a Linux, Solaris, * BSD y Mac OSX (Darwin).

También hay otros simuladores como V-REP, Webots, Stage, etc.. Pero no están tan extendidos como Gazebo.

1.3. Robótica en docencia universitaria

En la docencia universitaria se imparten clases de robótica en los Grados y los Postgrados, en concreto en escuelas de ingeniería. En España, se puede ver la robótica integrada en el "Grado en Ingeniería Robótica" de la Universidad de Alicante, y los Grados de "Electrónica industrial y automática" o en "Ingeniería Electrónica, Robótica y Mecatrónica" en diversas universidades. En los estudios de Postgrado existen Másteres destacados como el "Máster de Visión Artificial" en diferentes universidades. En el ámbito internacional se pueden destacar universidades especializadas en robótica como el MIT, Stanford, Georgia Institute of Technology, etc.

Cabe destacar como iniciativa privada el entorno de enseñanza robótica TheConstructSim [8] cuyo objetivo es enseñar a programar con el middleware ROS. Contiene una serie de tutoriales ROS en línea vinculados a simulaciones en línea, que brindan las herramientas y el conocimiento necesario para comprender y crear cualquier desarrollo de robótica basado en ROS. Usa robots reales simulados y solo se necesita un navegador web por lo que no requiere instalación.

[5] `http://opende.sourceforge.net/`
[6] `http://bulletphysics.org/wordpress/`
[7] `https://www.ogre3d.org/`
[8] `http://www.theconstructsim.com/`

1.3.1. JdeRobot-Academy

La Universidad Rey Juan Carlos cuenta con el proyecto de software libre en robótica JdeRobot, que incluye el entorno académico JdeRobot-Academy [9] [10] . Este entorno educativo se ha empleado con éxito en diferentes asignaturas, como son "Visión en Robótica" del Máster de Visión Artificial, o "Robótica" del Grado de Ingeniería Telemática. Asimismo, la Universidad ofrece cursos de introducción a la robótica y a los drones, empleando dicho entorno.

Este TFG se encuadra justo dentro de este entorno JdeRobot-Academy y aborda la creación de dos ejercicios nuevos. Está diseñado para que las prácticas desarrolladas por los alumnos se ejecuten en robots reales y también en simulados sin modificar el código fuente. El lenguaje de programación que se utiliza prácticas es Python debido a su sencillez y la potencia que ofrece. Para ejecutar dichas prácticas en robots, se usa el simulador Gazebo. Este simulador permite aprender robótica aunque no se tengan los robots reales ya que dispone de una gran variedad de robots, como pueden ser drones, coches, aspiradoras o brazos mecánicos, entre otros, pudiendo abarcar distintos aspectos de la robótica.

Cada práctica consta de una aplicación académica, que resuelve tareas como la interfaz gráfica (GUI) o la conexión con sensores y actuadores concretos, y que quedan simplificados para el alumno. También contiene el código del estudiante, que simplemente rellena un sencillo fichero plantilla, llamado `MyAlgorithm.py`, con la lógica del robot, logrando así que se centre solamente en la creación y desarrollo de los algoritmos de percepción, planificación y control habituales en los robots.

[9] `https://jderobot.org/JdeRobot-Academy`
[10] `https://github.com/JdeRobot/Academy`

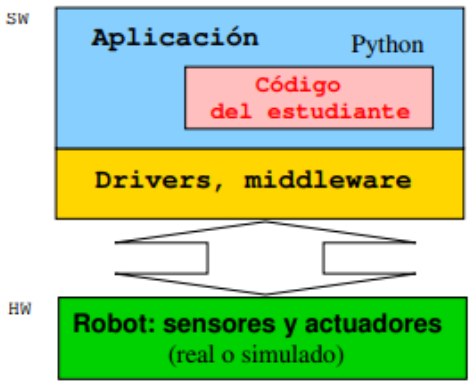

Figura 1.3: Diseño de una práctica robótica

En la Figura 1.3 se puede apreciar cómo está diseñada una práctica en el entorno de JdeRobot-Academy. En la capa inferior se encuentra el robot con sus actuadores (ruedas, motores...) y sensores (láseres, cámaras...) y puede ser tanto simulado como real. En la capa intermedia se tienen los drivers y middlewares necesarios para la comunicación entre la aplicación y el robot. Y por último, en la capa superior, está la aplicación donde se analizan los datos captados por los sensores y se sitúa el código del estudiante, tomando decisiones de actuación y planificación.

En la Figura 1.4 se puede ver la estructura que tiene cada una de las prácticas y como estan relacionados los distintos componentes.

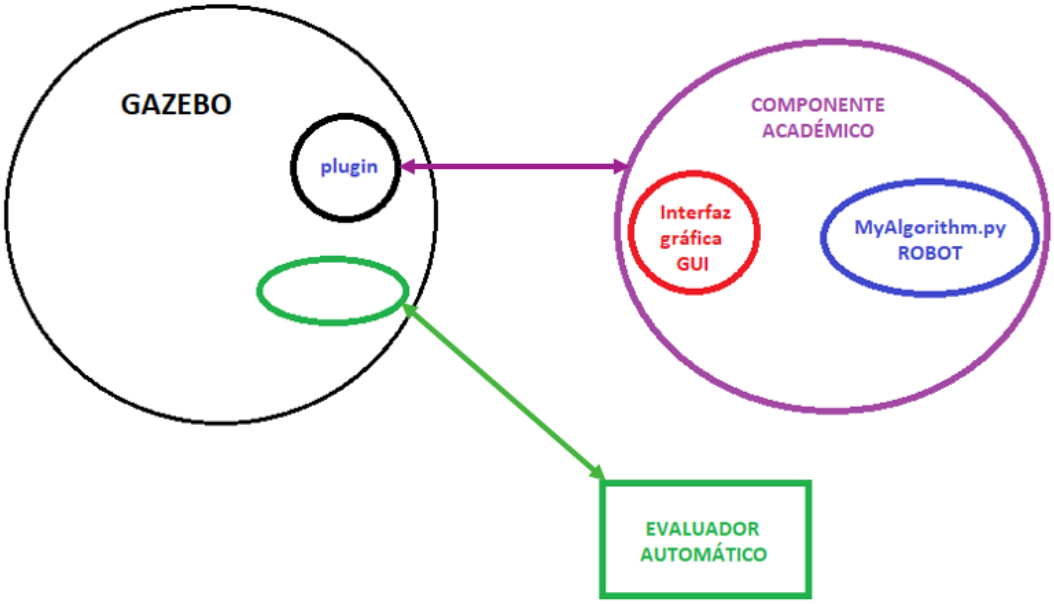

Figura 1.4: Estructura de una práctica robótica en JdeRobot-Academy

Los cuatro ámbitos principales en los que actualmente se agrupan los ejercicios disponibles son:

- Visión: Aquí se pueden encontrar las prácticas `Color filter`, cuyo principal objetivo es el uso de filtros de color, y `Visual 3D reconstruction from a stereo pair of RGB cameras` en la que se reconstruye una escena en 3D a partir de dos cámaras color a bordo de un robot (Figura 1.5).

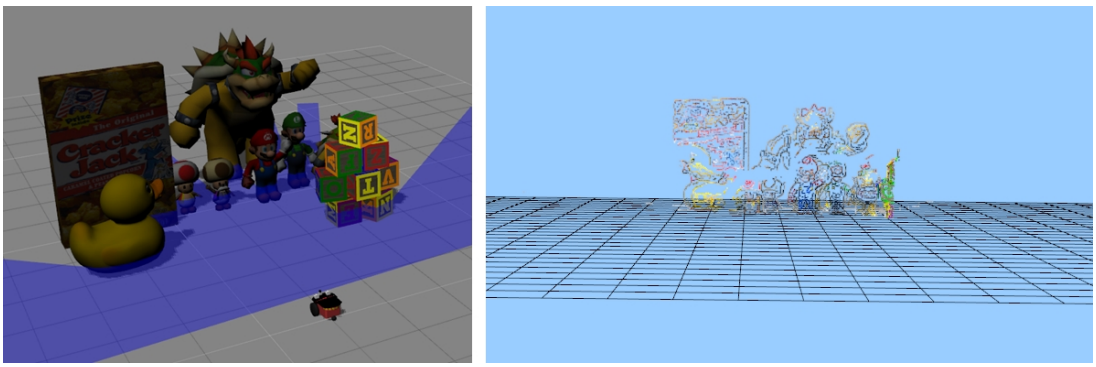

Figura 1.5: Práctica Visual 3D reconstruction

- Coches autónomos: En este ámbito se han desarrollado las prácticas `Visual follow-line behavior on a Formula 1` en la cual los alumnos tienen que conseguir que un co-

che de Fórmula 1 (Figura 1.6) logre seguir una línea roja pintada en un circuito de carreras ejercitando para ello el control visual; `Local navigation of a Formula 1 with VFF`, su objetivo es que el alumno programe un algoritmo de navegación local que consiga que un coche Fórmula 1 recorra un circuito evitando chocar con obstáculos (que son otros coches estacionados a lo largo del circuito); y `Global navigation of a TeleTaxi with GPP`, en la cual el alumno debe programar un algoritmo de planificación y otro de pilotaje para que un taxi consiga llegar a un punto del mundo en el que se encuentra clicando en una parte del mapa de dicho mundo.

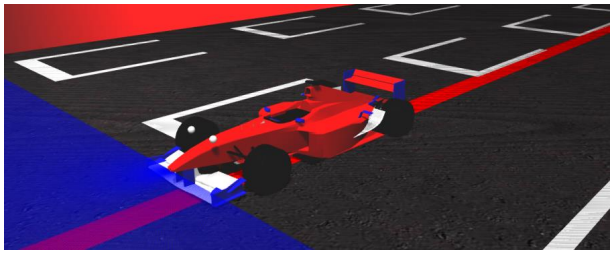

Figura 1.6: Coche de Fórmula 1

- Robots móviles: La práctica realizada es `Bump and go`, que consiste en programar un autómata finito de estados reactivo con un robot TurtleBot.

- Drones: Es el ámbito que posee más prácticas desarrolladas. `Drone position control navigation` se basa en el uso de controladores PID basados en posición; `Follow the ground robot`, cuyo objetivo es conseguir que un dron siga a un robot moviéndose en tierra; `Follow the road`, en la cual, un dron sigue una carretera; `Drone cat and mouse` que consiste en que un dron juega el papel de gato y tiene que atrapar al dron autónomo que tiene el papel de ratón (Figura 1.7); `Landing on a moving car`, en esta práctica hay que lograr que un dron aterrice sobre un coche en movimiento mediante control visual; `Escaping from a labyrinth using visual clue`, el objetivo es conseguir que un dron salga de un laberinto siguiendo las flechas ubicadas a lo largo de dicho laberinto que se perciben desde visión; y `People rescue after an earthquake` que consiste en que un dron detecte personas y marque la posición en la que se encuentran.

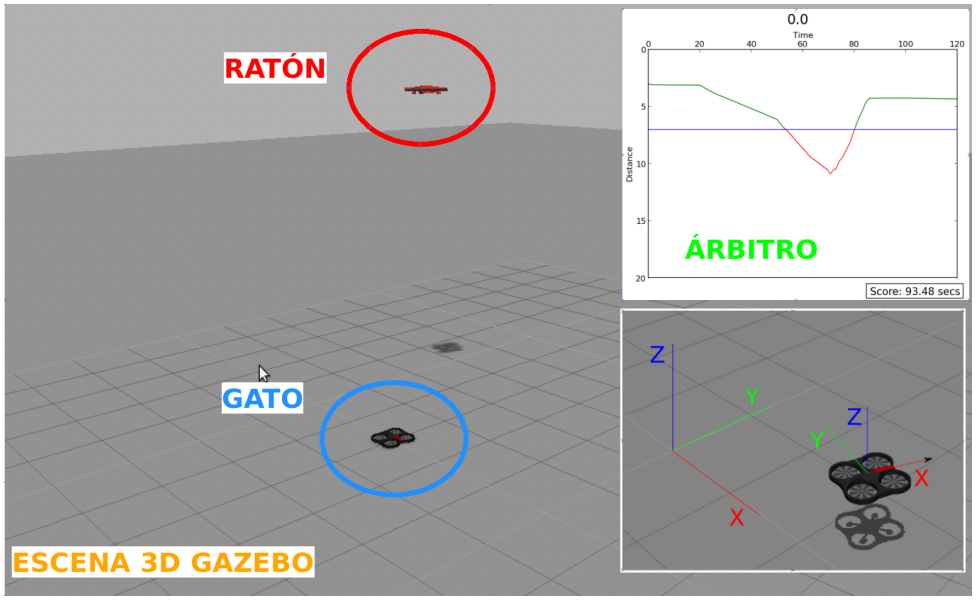

Figura 1.7: Práctica Drone cat and mouse

Capítulo 2

Objetivos y metodología

En este capítulo se expondrá de manera detallada el objetivo principal de este proyecto así como la metodología utilizada para su desarrollo. También se incluirá un plan de trabajo donde se explican las diferentes etapas seguidas en su realización.

2.1. Objetivos

El objetivo principal de este trabajo es llevar a cabo dos nuevas prácticas para la plataforma JdeRobot-Academy. Para cada práctica se creará el enunciado, la infraestructura necesaria en Gazebo, un nodo académico donde los estudiantes empotrarán su código y una solución tentativa.

El objetivo de la práctica "Coche autónomo negocia un cruce" es conseguir que un coche autónomo sea capaz de reconocer una señal de stop situada a lo largo de una carretera, gracias a la cámara que lleva situada en el techo, y después, frene. Además, tiene que ser capaz de reconocer si se acercan otros coches y en caso negativo volver a arrancar. Y por último, tiene que hacer un giro a la izquierda o a la derecha de manera aleatoria.

En la práctica "Aspiradora autónoma con autolocalización", el objetivo es lograr que un robot aspirador sea capaz de limpiar la mayor superficie posible de un apartamento en un tiempo limitado. Para ello, el alumno hará uso de la capacidad de autolocalización de la aspiradora y del mapa de la casa.

Uno de los requisitos de las prácticas es que los alumnos sólo tengan que concentrarse

en la parte de programación de robots no en funcionalidades auxiliares necesarias, como el interfaz gráfico o el mecanismo particular de acceso a sensores y actuadores. El nodo académico de cada práctica debe resolver eso y ofrecérselo de manera sencilla al estudiante. Además, gracias a la realización de estas prácticas el alumno adquirirá tanto conocimientos sobre programación de Python en robótica como conocimientos relacionados con el tratamiento digital de la imagen y la toma de decisiones.

2.2. Metodología

El desarrollo de trabajo que se ha seguido es el modelo en espiral. Este modelo consiste en una serie de ciclos o iteraciones que se repiten en forma de espiral como se muestra en la Figura 2.1.

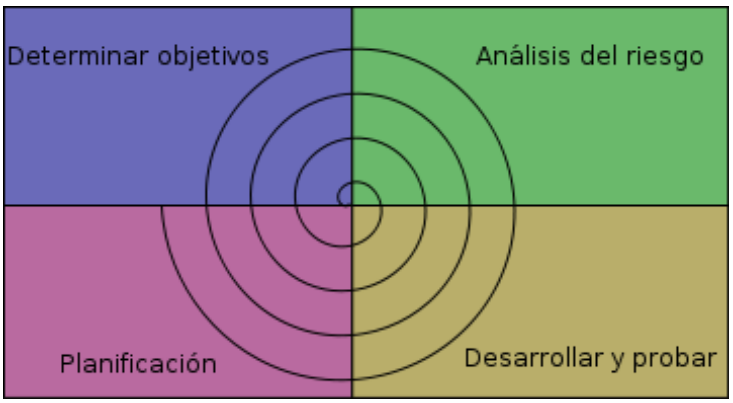

Figura 2.1: Metodología en espiral

Cada ciclo consta de cuatro fases:

- Determinar objetivos que se deben de llevar a cabo para que una vez completados se pueda dar por finalizado el ciclo.
- Análisis del riesgo: se evalúa qué problemas es posible encontrarse al empezar el desarrollo.
- Desarrollar y probar: en la tercera fase, una vez evaluados los riesgos, se procede al propio desarrollo del trabajo realizando distintas pruebas para conseguir el mejor resultado.

- Planificación: en esta última fase se valoran los resultados obtenidos y se planifican los siguientes etapas del proyecto.

No existe un número fijo de iteraciones, deben de llevarse a cabo tantas como sean necesarias hasta completar el trabajo.

Una ventaja del modelo de espiral en la ingeniería de software es que se adapta a cualquier número de cambios que pueden ocurrir durante cualquier fase del proyecto, permitiendo minimizar los riesgos y una buena evolución del trabajo.

Se han ido acordando reuniones semanales con el tutor. En estas reuniones, el tutor iba revisando el trabajo realizado y marcando nuevos objetivos para la semana siguiente. Además, servían para corregir fallos y comentar las dudas que iban surgiendo a lo largo de los meses de trabajo. Gracias a esto, se podía avanzar de manera más fluida en la realización del proyecto.

También se ha desarrollado una bitácora en la Wiki de JdeRobot [1] donde cada semana se redactaban los avances realizados y se añadían vídeos para mostrar los resultados obtenidos.

Conjuntamente a estas herramientas se ha utilizado un repositorio de Git Hub [2] para almacenar el software creado. En este repositorio se encuentra el código desarrollado en las prácticas, tanto el código de la infraestructura como el de las soluciones.

2.3. Plan de trabajo

Para lograr los objetivos descritos, se han seguido las siguientes etapas temporales:

- Familiarización del entorno JdeRobot, mediante la descarga e instalación de ese software además de las distintas dependencias necesarias. En esta etapa también se llevó a cabo la resolución de algunas prácticas anteriores de JdeRobot-Academy relacionadas con las prácticas a desarrollar.

[1] `http://jderobot.org/Ilope-tfg`

[2] `https://github.com/RoboticsURJC-students/2016-tfg-irene-lope`

- Familiarización del simulador Gazebo. En esta etapa se han estudiado distintos ejemplos disponibles en la web de Gazebo y de JdeRobot. Además, se han comprendido los distintos *plugins* (son los drivers de los robots simulados) necesarios para el desarrollo del trabajo, lo que implicó un aprendizaje básico del lenguaje de programación C++.

- Crear los mundos y *plugins* necesarios de Gazebo. Se modificaron distintos *plugins* ya creados para adaptarlos a las prácticas que se iban a desarrollar. También se diseñaron los mundos que se han utilizado en cada ejercicio incluyendo en dichos mundos tanto los modelos de objetos y obstáculos como los de los robots y coches.

- Desarrollar la infraestructura académica para la práctica del coche autónomo, que consiste principalmente en crear la interfaz gráfica, utilizando la herramienta PyQt5, para que el alumno pueda resolver las prácticas de manera más sencilla e intuitiva.

- Realizar la solución para la práctica del coche autónomo.

- Desarrollar la infraestructura académica y un evaluador automático para la práctica de la aspiradora autónoma, para que el alumno sepa de manera automática qué nota ha obtenido con el código que programe. Este evaluador medirá distintos parámetros y calculará una nota final. También ha sido necesario el uso de la herramienta PyQt5.

- Realizar la solución para la práctica la aspiradora autónoma.

- Redactar los enunciados, donde se explica el contenido de cada práctica.

Capítulo 3

Infraestructura

En este capítulo se explicarán los principales ingredientes software en los que nos hemos apoyado para desarrollar el trabajo. Tales como el simulador Gazebo, el entorno JdeRobot, la biblioteca de OpenCV (de tratamiento de imagen), la biblioteca PyQt (para interfaces gráficas) y Python como lenguaje de programación.

3.1. Simulador Gazebo

Es un simulador usado en robótica que permite utilizar diversos escenarios tridimensionales virtuales donde probar nuestro software robótico. A la hora de desarrollar el software es necesario hacer pruebas, las cuales saldrían muy costosas si se probaran en robots reales (podría no funcionar correctamente y que el robot se rompiera). Por esta razón es muy útil el empleo de simuladores, pues se pueden realizar las pruebas que se quieran sin peligro de que el robot se estropee. Con los simuladores se pueden diseñar robots y escenarios realistas donde ejecutar los algoritmos creados.

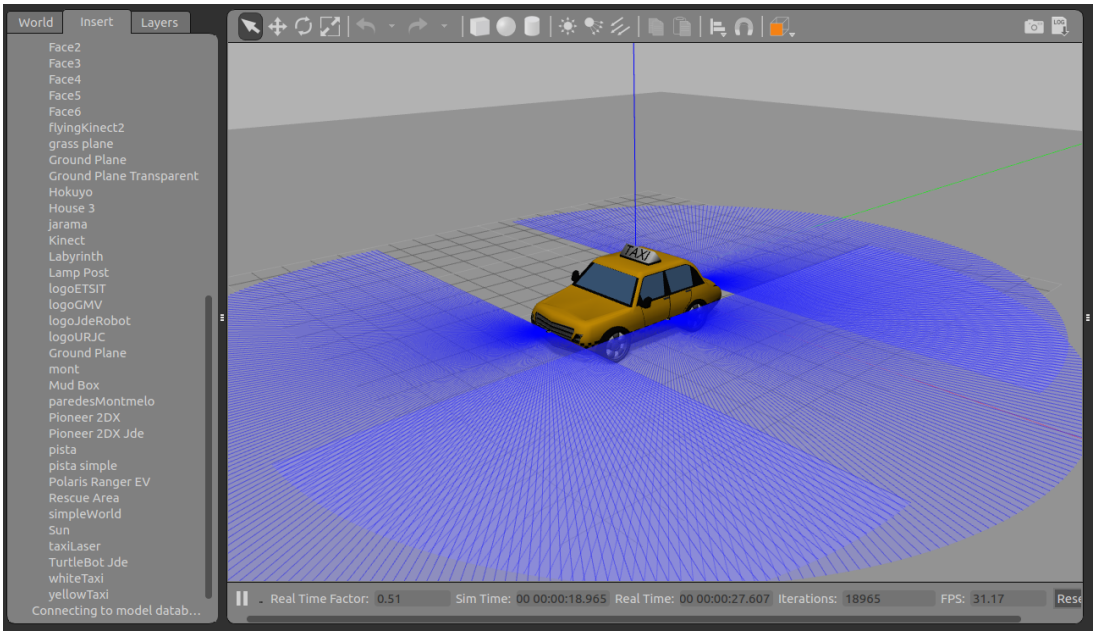

Figura 3.1: Simulador Gazebo

Gazebo [1] es un programa de código abierto distribuido bajo licencia de Apache 2.0. Se emplea en el desarrollo de aplicaciones robóticas y en inteligencia artificial. Fue elegido para realizar el DARPA Robotics Challenge (2012-2015) y está mantenido por la Fundación Open Robotics [2]. Es capaz de simular robots, objetos y sensores en entornos complejos de interior y exterior. Tiene gráficos de gran calidad y un robusto motor de física (masa del robot, rozamiento, inercia, amortiguamiento, etc.).

El simulador Gazebo es uno de los ejes principales de JdeRobot-Academy. En este proyecto se emplea la versión 7 de Gazebo, la cual se usará para crear los diferentes entornos de las dos prácticas y para probar nuestras soluciones de referencia.

Gracias a Gazebo se pueden incluir texturas, luces y sombras en los escenarios, así como simular física como por ejemplo choques, empujes, gravedad, etc. Además, incluye diversos sensores, como pueden ser cámaras y lásers, los cuales podrán ser incorporados en los robots que empleemos. Todo ello hace que sea una herramienta muy potente y de gran ayuda en el mundo de la robótica.

[1] `http://gazebosim.org/`
[2] `https://www.openrobotics.org/`

Los mundos simulados con Gazebo son 3D, que se cargan a partir de ficheros con extensión ".world". Son ficheros definidos en SDF (Simulation Description Format) que es un formato XML (Extensible Markup Language) que describe objetos y entornos para simuladores, visualización y control de robots. Originalmente fue desarrollado como parte del simulador Gazebo pero con los años se ha convertido en un formato estándard estable, robusto y extensible capaz de describir todos los aspectos de los robots, los objetos estáticos y dinámicos, la iluminación, el terreno e incluso la física.

Se puede describir con precisión todos los aspectos de un robot usando SDF, ya sea un robot que sea un simple chasis con ruedas o un humanoide. Además de los atributos cinemáticos y dinámicos, se pueden definir sensores, propiedades de superficie, texturas, fricción de las juntas y muchas más propiedades de un robot. Estas características permiten usar SDF para simulación, visualización, planificación de movimiento y control de robot.

Los modelos de robots que se emplean en la simulación pueden ser creados mediante algún programa de modelado 3D como Blender o Sketchup. Estos robots simulados necesitan también ser dotados de inteligencia para lo cual se emplean *plugins*. Éstos, pueden dotar al robot de comportamiento autónomo u ofrecer la información de sus sensores a aplicaciones externas y recibir de éstas comandos para los actuadores de los robots.

3.2. Entorno JdeRobot

JdeRobot [3] es un middleware de software libre para el desarrollo de aplicaciones con robots y visión artificial. Esta plataforma fue creada por el Grupo de Robótica de la Universidad Rey Juan Carlos en 2003 y está licenciada como GPLv3 [4].

Está desarrollado en C y C++, aunque contiene componentes desarrollados en lenguajes como Python y Java. Proporciona un entorno de programación distribuido basado en componentes, donde el programa de aplicación se compone de una colección de varios

[3] `http://jderobot.org/Main_Page`

[4] `https://www.gnu.org/licenses/quick-guide-gplv3.html`

componentes concurrentes asincrónicos. Dichos componentes operan entre sí mediante el middleware de comunicación ICE(Internet Communications Engine) o ROS messages. Los componentes pueden tener su propia GUI (Graphical User Interface).

La obtención de mediciones del sensor o el envío de órdenes al motor se realizan llamando a funciones locales. La plataforma conecta esas llamadas a componentes del driver que están conectados a dispositivos hardware del robot (reales o simulados, remotos o locales). Esas funciones conforman la API de la capa de abstracción de hardware.

Las aplicaciones constan de uno o varios componentes. Los que interactúan directamente con los sensores y actuadores del robot se llaman drivers, que son los encargados de controlar los robots y ofrecer interfaces de red con mensajes ICE o con ROS messages. Otros llevan en su código las funciones perceptivas, procesamiento de señales o la lógica de control e inteligencia del robot. En la Figura 3.2 se puede ver un ejemplo de esta comunicación con un AR Drone empleando interfaces ICE. La misma lógica de comportamiento se puede conectar al driver del drone real o al drive del drone simulado, basta con cambiar la configuración.

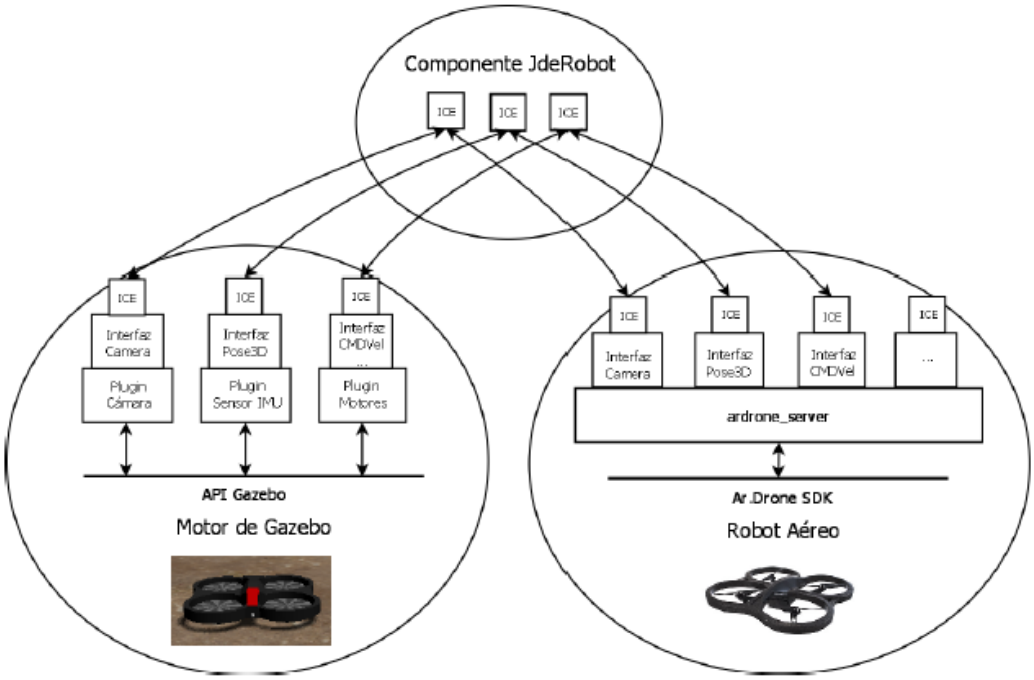

Figura 3.2: Ejemplo de componentes JdeRobot

Algunos de los dispositivos y robots actualmente soportados en JdeRobot son:

- Sensores RGBD: Kinect y Kinect2 de Microsoft, Asus Xtion tanto reales como simulados en Gazebo.
- Robots de ruedas: TurtleBot de Yujin Robot y Pioneer de MobileRobotics Inc. tanto en real como en Gazebo.
- ArDrone quadrotor de Parrot tanto real como en Gazebo.
- Escáneres láser: LMS de SICK, URG de Hokuyo y RPLidar tanto reales como en Gazebo.
- Cámaras Firewire, cámaras USB, archivos de vídeo (mpeg, avi ...), cámaras IP (como Axis) tanto en real como en Gazebo.

JdeRobot también incluye varias herramientas y bibliotecas de programación de robots como:

- Teleoperadores para varios robots, viendo sus sensores y controlando sus motores.
- Herramienta "VisualStates" para programar el comportamiento del robot empleando autómatas de estado finito.
- Herramienta "Scratch2JdeRobot" para programar robots (incluidos los drones) con el lenguaje gráfico de bloques estándar.
- Un calibrador de cámara.
- Una herramienta para ajustar filtros de color llamada "ColorTuner".
- Una biblioteca para desarrollar controladores borrosos y una biblioteca para la geometría proyectiva y el procesamiento de visión artificial.

En el desarrollo del proyecto se empleará la versión 5.6.2 de JdeRobot, ya que es la última versión estable. El nodo académico que se desarrollará para cada una de las prácticas es una aplicación JdeRobot.

3.3. Lenguaje de programación Python

Python [5] es uno de los ejes principales de JdeRobot-Academy. Es el lenguaje de programación en el que están escritos los componentes académicos y las soluciones por lo que es el lenguaje utilizado para el desarrollo de este proyecto. Se trata de un lenguaje fácil de aprender y de alto nivel, es decir, se caracteriza por expresar los algoritmos de una manera adecuada a la capacidad cognitiva humana. Su creador fue Guido van Rossum, un investigador holandés que trabajaba en el centro de investigación CWI (Centrum Wiskunde & Informatica). Actualmente Python es un lenguaje de código abierto administrado por Python Software Foundation.

Algunas funcionalidades incluidas en Python son la programación orientada a objetos, el manejo de excepciones, listas y diccionarios entre otras. A pesar de todo lo que soporta, se creó con el objetivo de que fuera un lenguaje sencillo de entender, sin perder todas las funcionalidades que pueden ofrecer lenguajes complejos tales como C. Incluye módulos que permiten la entrada y salida de ficheros, sockets, llamadas al sistema e incluso interfaces gráficas como Qt. Además, permite dividir el programa en módulos reutilizables y no es necesario compilarlo, pues es un lenguaje interpretado.

La última versión ofrecida por Python Software Foundation es la 3.6.2, pero en nuestro caso se empleará la 2.7.12 por compatibilidad con JdeRobot 5.5.2, que a su vez, sigue en esa versión de Python para ser compatible con ROS Kinetic.

3.4. Biblioteca OpenCV

OpenCV [6] es una librería de código abierto desarrollada por Intel y publicada bajo licencia de BSD. Sus siglas provienen de los términos anglosajones “Open Source Computer Vision Library”. Esta librería implementa gran variedad de herramientas para la interpretación de la imagen.

La biblioteca cuenta con más de 2500 algoritmos optimizados, que incluyen un conjunto completo de algoritmos de visión artificial y de aprendizaje automático, tanto clásicos

[5] `https://www.python.org/`
[6] `http://opencv.org/`

como avanzados. Estos algoritmos se pueden usar para detectar y reconocer rostros, identificar objetos, rastrear movimientos de la cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, unir imágenes para producir una alta resolución imagen de una escena completa, encontrar imágenes similares de una base de datos de imágenes, etc. También tiene una librería de aprendizaje automático (MLL, Machine Learning Library) destinada al reconocimiento y agrupación de patrones estadísticos.

Fue diseñado para tener una alta eficiencia computacional. Está escrito en C/C++ y puede aprovechar las ventajas de los procesadores multinúcleo. Los algoritmos se basan en estructuras de datos flexibles acopladas con estructuras IPL (Intel Image Processing Library), aprovechándose de la arquitectura de Intel en la optimización de más de la mitad de las funciones. También se puede aprovechar del uso de tarjetas gráficas para acelerar el procesamiento.

Desde su aparición OpenCV ha sido usado en numerosas aplicaciones entre las cuales se encuentran la unión de imágenes de satélites y de mapas web, la reducción de ruido en imágenes médicas, los sistemas de detección de movimiento, la calibración de cámaras, el manejo de vehículos no tripulados o el reconocimiento de gestos. OpenCV es empleado también en reconocimiento de música y sonido, mediante la aplicación de técnicas de reconocimiento de visión en imágenes de espectrogramas del sonido. OpenCV ha sido usada en el sistema de visión del vehículo no tripulado Stanley de la Universidad de Stanford, el ganador en el año 2005 del Gran desafío DARPA. También hay una gran cantidad de empresas y centros de investigación que emplean estas técnicas como IBM, Microsoft, Intel, SONY, Siemens, Google, Stanford, MIT, CMU, Cambridge e INRIA.

Esta librería puede ser usada en distintos sistemas operativos como MAC, Windows, Android y Linux, y existen versiones para lenguajes de programación como C#, Python y Java, a pesar de que originalmente era una librería en C/C++. Además, hay interfaces en desarrollo para Ruby, Matlab y otros lenguajes.

En este trabajo se ha empleado la versión 3.2 de OpenCV en Python. Esta librería se empleará para realizar todo lo relacionado con el tratamiento digital de imágenes en el ejercicio del coche autónomo. Con ello se extraerán datos que puedan emplearse a la hora

de tomar decisiones para que los robots funcionen correctamente.

3.5. Bliblioteca PyQt

PyQt [7] es un conjunto de enlaces Python para el conjunto de herramientas Qt, las cuales se emplean para el desarrollo de interfaces gráficas. Fue desarrollado por Riverbank Computing Ltd y es soportado por Windows, Linux, Mac OS/X, iOS y Android.

Qt es un entorno multiplataforma orientado a objetos desarrollado en C++ que permite desarrollar interfaces gráficas e incluye sockets, hilos, Unicode, bases de datos SQL, etc. PyQt combina todas las ventajas de Qt y Python, pues permite emplear todas las funcionalidades ofrecidas por Qt con un lenguaje de programación tan sencillo como Python. En concreto PyQt5 es un conjunto de enlaces Python para Qt5, disponible en Python 2.x y 3.x. Tiene más de 620 clases y 6000 funciones y métodos. PyQt5 dispone de una licencia dual, es decir, los desarrolladores pueden elegir entre una licencia GPL (General Public Licence) o una licencia comercial.

La interfaz gráfica de los componentes académicos creados en las prácticas de este TFG está escrita usando PyQt5. Las clases se dividen en ciertos módulos, tales como QtCore, QtGui, QtWidgets, QtXml o QtSql. En las prácticas desarrolladas se ha hecho uso de los siguientes módulos:

- QtCore: contiene las funcionalidades principales que no tienen que ver con la interfaz gráfica. Este módulo se emplea para trabajar con archivos, diferentes tipos de datos, hilos, procesos, url, etc.
- QtGui: contiene clases para el desarrollo de ventanas, gráficos 2D, imágenes y texto.
- QtWidgets: dispone de clases que proporcionan un conjunto de elementos de interfaz de usuario para crear interfaces de usuario clásicas de escritorio.

[7] `https://riverbankcomputing.com/software/pyqt/intro`

Capítulo 4

Práctica: Coche autónomo negocia un cruce

En este capítulo se expondrá la creación de una nueva práctica para la plataforma de JdeRobot-Academy, llamada "Coche autónomo negocia un cruce". Se explicará el desarrollo de su infraestructura, su componente académico correspondiente, así como la solución de referencia realizada.

4.1. Enunciado

El objetivo principal de esta práctica es conseguir que un coche autónomo sea capaz de negociar un cruce en una carretera. Deberá reconocerla señal de STOP, frenar a tiempo en un cruce y después, si no detecta coches, volver a arrancar y realizar un giro a la izquierda o a la derecha de manera aleatoria. El modelo del coche utilizado está dotado de cámaras de vídeo para visualizar el entorno, un sensor de posición y actuadores de movimiento que permiten controlar tanto la velocidad lineal como la de giro.

En esta práctica el alumno tiene que programar un algoritmo que cumpla todos los puntos descritos, evitando que el coche autónomo choque con otros coches u obstáculos de su entorno. En la interfaz gráfica del componente académico se pueden visualizar las imágenes captadas por las cámaras.

El algoritmo responde a un control reactivo, por lo que en cada instante actuará en función de los datos captados por los sensores permitiendo controlar el movimiento del

coche y reaccionar a distintos imprevistos.

4.2. Infraestructura desarrollada

En este apartado se describirán los elementos del simulador Gazebo que se han creado o integrado para poder realizar esta práctica "Coche autónomo negocia un cruce". Primero se describirá el modelo del coche utilizado, incluyendo sus sensores y actuadores. Después, se explicará el mundo simulado por el cual se moverá el coche.

4.2.1. Modelo coche Opel

Para esta práctica se ha creado un nuevo modelo de coche autónomo. El robot está basado en un modelo de coche Opel. Este modelo de coche se denomina *"Opel"* y se puede ver en la Figura 4.1. Posee tres cámaras de vídeo que se usarán para la detección de la señal de STOP y la detección de otros coches; un sensor de posición, que se utilizará para obtener su orientación al realizar los giros, tanto a la izquierda como a la derecha; y motores que le permiten moverse por el mundo de Gazebo.

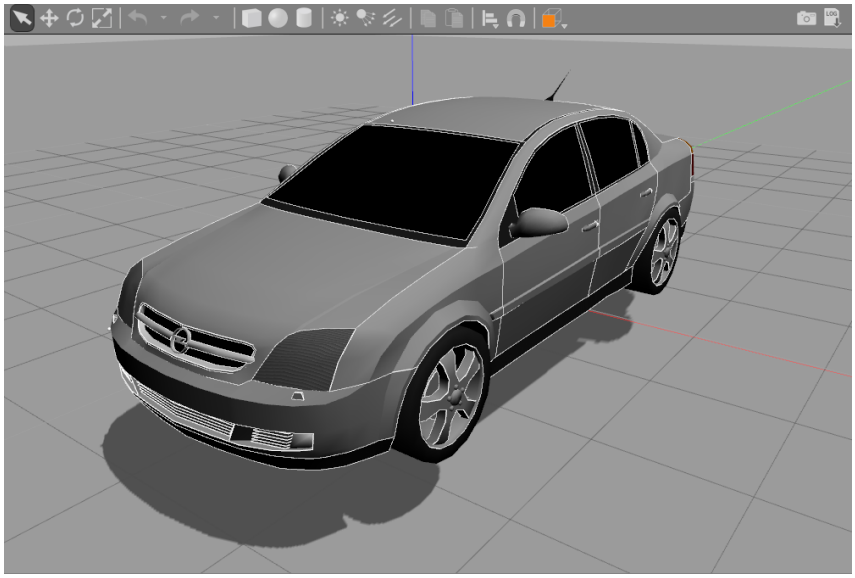

Figura 4.1: Modelo coche Opel

En esta práctica se han empleado los siguientes *plugins* para este modelo, todos ellos hacen de drivers del coche simulado y con ellos interactúa el nodo académico que aloja el código escrito por el estudiante para este ejercicio:

- *OpelMotors*: Este *plugin* permite ordenar velocidad al coche, tanto velocidad de tracción como velocidad de rotación.
- *camera_dump*: Este *plugin* será empleado por los componentes para tener visión del entorno. En el coche hay tres, uno para la cámara central y dos para las laterales.
- *Pose3D*: Este *plugin* se emplea para obtener la posición del coche en tiempo real y su orientación.

En este coche se han instalado 3 cámaras de vídeo. Las imágenes captadas tienen un tamaño de 640 x 480 píxeles y siguen el formato de color RGB (Red Green Blue)(cada uno de estos tres canales está codificado con 1 byte). El rango de profundidad de visión de las cámaras abarca desde 10 centímetros hasta 35 metros. Una de las cámaras está situada en el centro del techo (ligeramente a la derecha), otra al lado del faro derecho (orientada hacia a la derecha) y la última, a lado del faro izquierdo (orientada hacia a la izquierda). De esta manera, conseguimos un campo de visión más amplio. La cámara central se usará para el proceso de detección de la señal de STOP y la carretera, y las cámaras laterales, para la detección de otros coches.

4.2.2. Modelo de coche figurante

Se han añadido dos coches adicionales que se mueven de manera automática a lo largo de una de las carretera del cruce para simular el tráfico de una calle. Ambos siguen el modelo *"car"* creado y utilizan la misma malla visual que el modelo Opel por lo que tienen el mismo aspecto.

Para que estos coches se muevan de manera automática se ha creado un nuevo *plugin* llamado *"carplugin"*. Este *plugin* dota de velocidad lineal constante al modelo en su eje X. Dependiendo de su posición en el simulador Gazebo, tomará dirección positiva o negativa.

4.2.3. Modelo señal de STOP

Debido a que el objetivo principal de la práctica es reconocer una señal de STOP, se ha añadido el modelo *"stop_sign"*. Este modelo se puede ver en la Figura 4.2. Es un modelo estático, sin *plugins*.

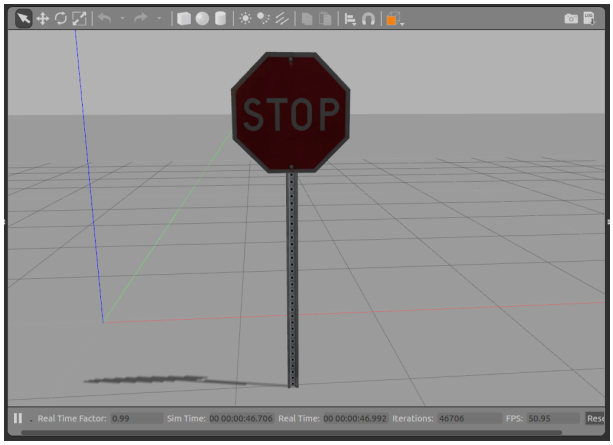

Figura 4.2: Modelo stop_sign

4.2.4. Modelo cruce de carreteras

Ha sido necesario crear el modelo de un cruce de carreteras por donde circularán los coches utilizados en esta práctica. Este modelo se ha denominado *"StopW"* y se puede ver en la Figura 4.3. Es un modelo estático, sin *plugins*.

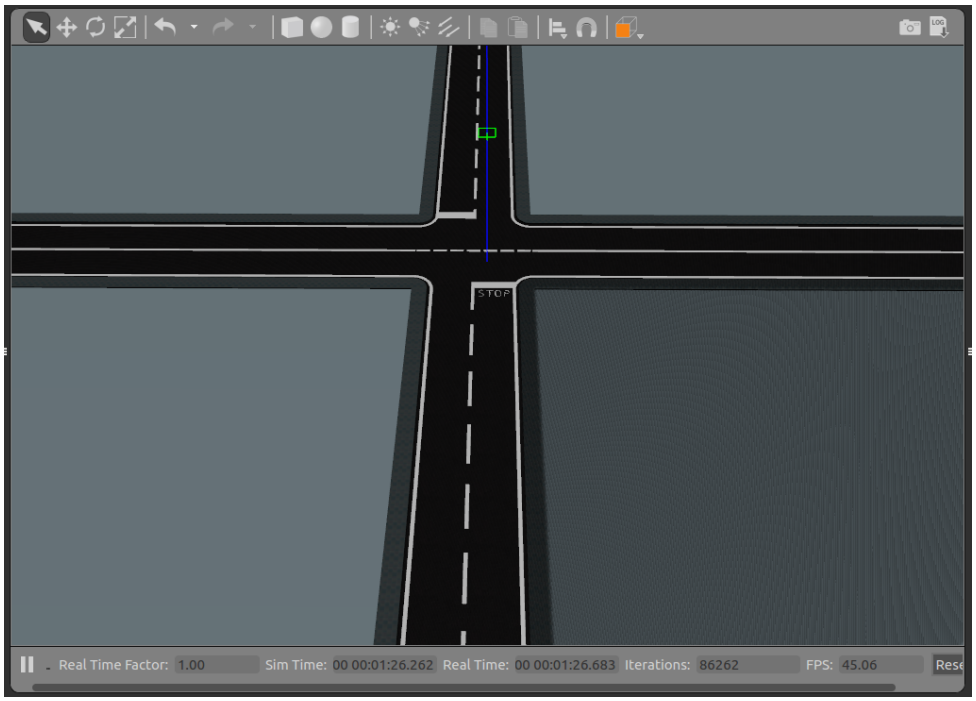

Figura 4.3: Modelo stopW

4.2.5. Mundo de Gazebo

El mundo creado en Gazebo para este ejercicio está formado por un modelo de cruce de carreteras *"stopW"*, dos señales de STOP con el modelo *"stop_sign"*, posee 2 coches que se mueven de manera autómatica empleando el modelo *"car"* y se incluye el modelo del coche *"Opel"*, que ejecutará la solución desarrollada. Por útimo, para dotar de realismo al mundo se han añadido como decoración algunos modelos que tiene Gazebo:

- Modelo *sun*: Añade una fuente de luz global a la escena.
- Modelo *house_1*, modelo *house_2*, modelo *house_3*: Se han utilizado distintos modelos de casas.
- Modelo *gas_station*: Se ha añadido una gasolinera.
- Modelo *lamp_post*: Se han añadido un total de 14 farolas situadas a lo largo de las carreteras.

Para tener este escenario se ha creado un mundo en Gazebo llamado `stop.world` [1]. En la Figura 4.4 se puede observar el mundo creado en Gazebo.

```
<?xml version="1.0"?>
<sdf version="1.4">
  <world name="default">

    <!-- Stop signs -->
    <include>
      <static>true</static>
      <uri>model://gazebo/models/stop_sign</uri>
      <pose>3.5 -3.5 0 0 0 0</pose>
    </include>

    <!-- Houses -->
    <include>
      <uri>model://house_1</uri>
```

[1] https://github.com/RoboticsURJC-students/2016-tfg-irene-lope/blob/master/Stop_Practice/stop.world

```
      <pose>-9.5 8.5 0 0 0 0</pose>
    </include>

    <!-- A gas station -->
    <include>
      <uri>model://gas_station</uri>
      <pose>10 14 0 0 0 1.55</pose>
    </include>

    <!-- Lamps -->
    <include>
      <uri>model://lamp_post</uri>
      <pose>-3 13 0 0 0 1.55</pose>
    </include>

    <!-- A opel car -->
    <include>
      <uri>model://opel</uri>
      <pose>1.5 -30 0 0 0 3.14</pose>
    </include>

    <!-- Cars -->
    <include>
      <uri>model://car</uri>
      <pose>-30 -1.5 0 0 0 1.57</pose>
    </include>

    <!-- Roads -->
    <include>
      <uri>model://stopW</uri>
      <pose>0 0 0 0 0 0</pose>
    </include>

  </world>
</sdf>
```

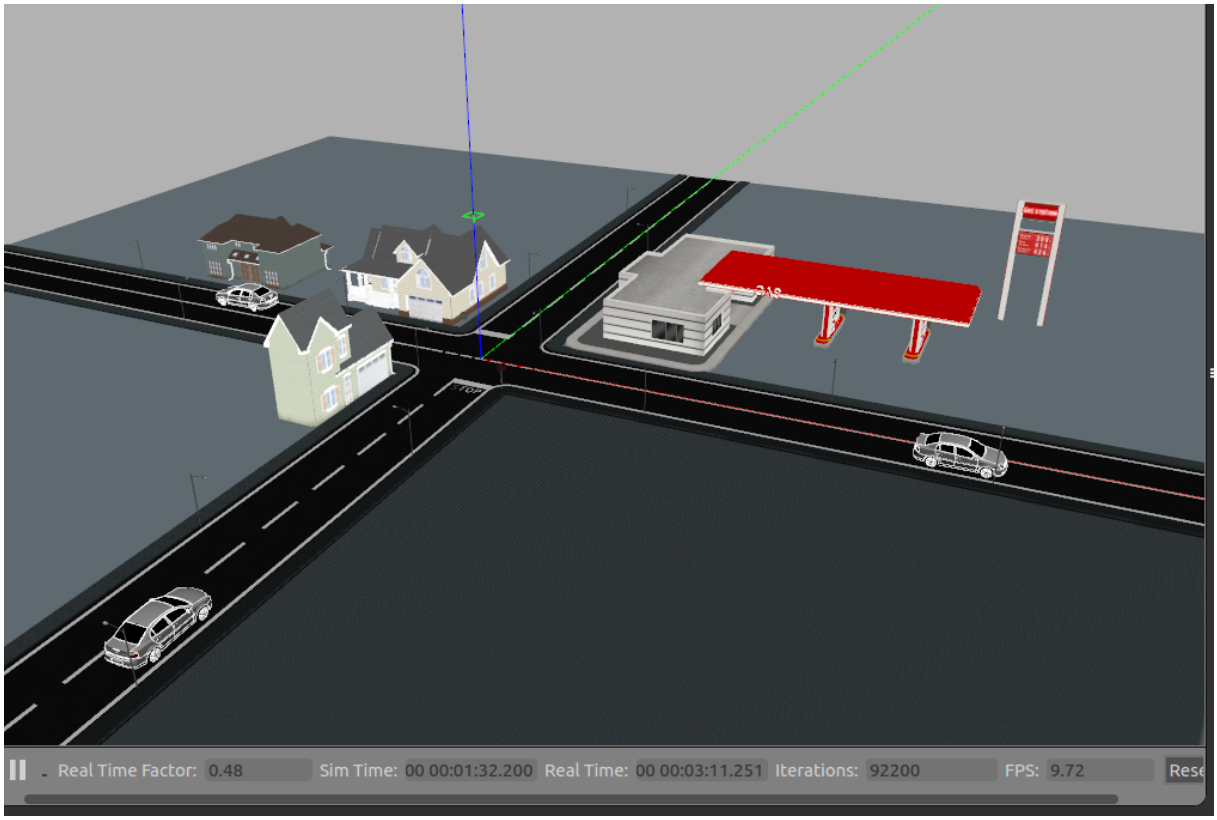

Figura 4.4: Mundo stop.world en Gazebo

4.3. Nodo académico

Se ha creado un componente académico para esta práctica que resuelve diversas funcionalidades: (a) muestra una interfaz gráfica al usuario, con distintos elementos, que permiten al estudiante depurar su código de manera más sencilla; (b) ofrece acceso a sensores y actuadores en forma de métodos simples ocultando el *middleware* de comunicaciones; (c) incluye código auxiliar que ayuda a programar la solución. El componente deja todo listo para que el estudiante sólo tenga que incorporar su código rellenando el método *"execute"* en el fichero `MyAlgorithm.py`.

Para realizar todas las tareas necesarias para el funcionamiento de la práctica se emplean dos hilos de ejecución:

- Hilo de percepción y control: se encarga de actualizar de manera constante los datos captados por los sensores y de decidir los datos enviados a los actuadores. Debido a

que se trata de una solución reactiva es necesario que el intervalo de actualización de dichos datos sea un tiempo muy corto, en este caso, 50 ms. Si se fijara un tiempo muy grande podría ocasionar errores en la trayectoria del robot. De este modo se puede crear un control reactivo, donde el coche irá tomando decisiones según los datos que vaya obteniendo de los sensores en tiempo real.

- Hilo de la GUI: este hilo actualiza la interfaz gráfica que se muestra al estudiante. Este intervalo de actualización debe de ser también corto ya que la interfaz gráfica es una herramienta que utiliza el programador para depurar su código y debe de mostrar de manera fiable los datos del robot en tiempo real. Este intervalo también es de 50 ms.

4.3.1. Interfaz de acceso al hardware

Este componente ofrece al programador del algoritmo este API de sensores y actuadores:

- *pose3d.getX()*: Permite obtener la posición absoluta del robot en el eje X.
- *pose3d.getY()*: Permite obtener la posición absoluta del robot en el eje Y.
- *pose3d.getYaw()*: Permite obtener la orientación del robot con respecto al sistema de referencia de Gazebo.
- *motors.sendV()*: Para establecer la velocidad lineal.
- *motors.sendW()*: Para establecer la velocidad de giro.
- *cameraC.getImage()*: Permite obtener las imágenes captadas por la cámara situada en el techo del coche.
- *cameraL.getImage()*: Permite obtener las imágenes captadas por la cámara situada en el faro izquierdo.
- *cameraR.getImage()*: Permite obtener las imágenes captadas por la cámara situada en el faro derecho.

4.3.2. Fichero de configuración

Es necesario crear un archivo de configuración para este nodo (`stop.cfg`) donde se indican los puertos utilizados por los distintos *plugins*, la velocidad lineal máxima y la velocidad angular máxima de los motores.:

```
Stop.CameraC.Proxy = cam_opel_center:default -h localhost -p 8995
Stop.CameraL.Proxy = cam_opel_left:default -h localhost -p 8996
Stop.CameraR.Proxy = cam_opel_right:default -h localhost -p 8997
Stop.Motors.Proxy = Motors:default -h localhost -p 9999
Stop.Pose3D.Proxy = Pose3D:default -h localhost -p 9989

Stop.Motors.maxV = 250
Stop.Motors.maxW = 20
```

En esta práctica, los motores emplean el puerto 9999; el sensor de posición el puerto 9989; y las cámaras central, izquierda y derecha utilizan los puertos 8995, 8996 y 8997, respectivamente.

4.3.3. Interfaz gráfica

Para que la realización de la práctica sea más fácil, se ha creado una interfaz gráfica que muestra al usuario datos importantes, relativos al robot. Además, esta interfaz permite de manera sencilla ejecutar el código donde se desarrolla la solución correspondiente. Se ha empleado la herramienta PyQt5 para su desarrollo.

En la parte izquierda de la interfaz gráfica, se muestran las imágenes captadas por las tres cámaras de vídeo instaladas en el robot (Figura 4.5). Mediante estas imágenes el programador podrá verificar si está realizando correctamente el tratamiento digital de las imágenes necesario para detectar tanto la señal de STOP, como la carretera o los otros coches. Las imágenes de la cámara izquierda se reproducen a la izquierda de la interfaz, las de la cámara central en el centro y las de la cámara derecha a la derecha.

A la derecha del GUI, hay un teleoperador bidimendional que permite mover el coche

manualmente. La velocidad lineal se puede controlar con el eje vertical: cuanto más se suba el *joystick* más velocidad tendrá el coche hacia delante, y si se baja, más velocidad tendrá el robot hacia atrás. La velocidad angular del coche se controla moviendo el *joystick* en sentido horizontal, según se mueva a la izquierda o a la derecha, el robot girará en un sentido u otro.

En la parte inferior aparecen dos botones. El botón situado bajo las imágenes captadas por las cámaras permite tanto ejecutar como detener el código alojado en el archivo `MyAlgorithm.py`. Con el botón que está bajo el teleoperador, se puede detener al robot si se está controlando con el teleoperador.

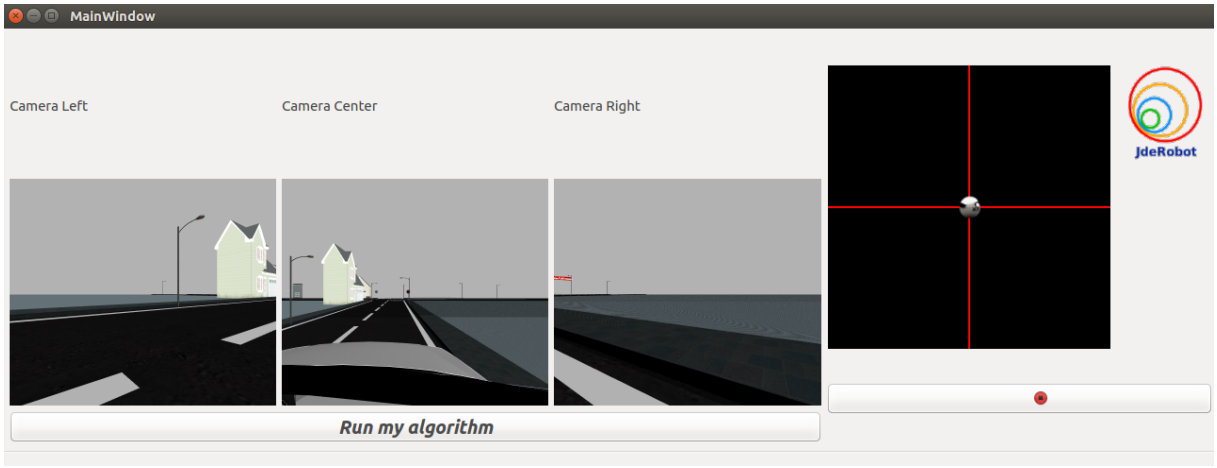

Figura 4.5: Interfaz gráfica

4.4. Solución de referencia

La solución desarrollada para esta práctica resuelve el problema planteado y se ha programado en el fichero `MyAlgorithm.py`, en el método *"execute"*, que es el método principal de la solución. Para esta práctica se ha realizado el pilotaje del robot sin una planificación de ruta previa. La solución puede dividirse en tres partes principales: (a) reconocimiento de la señal de STOP y frenado del coche; (b) detección de otros coches; (c) realización del giro.

4.4.1. Reconocimiento de la señal de STOP y frenado del coche

Para detectar la señal de STOP, nos basaremos en dos de sus características principales: el color y la forma. Si sólo nos centráramos en el color podría llevarnos a error, ya que existen más señales viales de color rojo, por lo que necesitamos fijarnos en otra característica adicional. En este caso nos hemos decantado por la forma ya que la señal de STOP es la única señal vial que tiene forma octogonal.

Para llevar acabo el proceso de detección, utilizaremos las imágenes captadas por la cámara frontal situada en el techo del coche. A estas imágenes les aplicamos un filtro de color para detectar el color rojo específico de la señal. Para ello, hemos creado la función *"filterHSV"* que realiza el cambio de modelo de color de RGB al modelo HSV (Hue Saturation Value) (Figura 4.6), el proceso de segmentación (Figura 4.7) y aplica la operación morfológica de cierre (Figura 4.8) para eliminar la palabra "STOP" de la señal. Para fijar los valores del filtro, hemos utilizado la herramienta "ColorTuner" de JdeRobot.

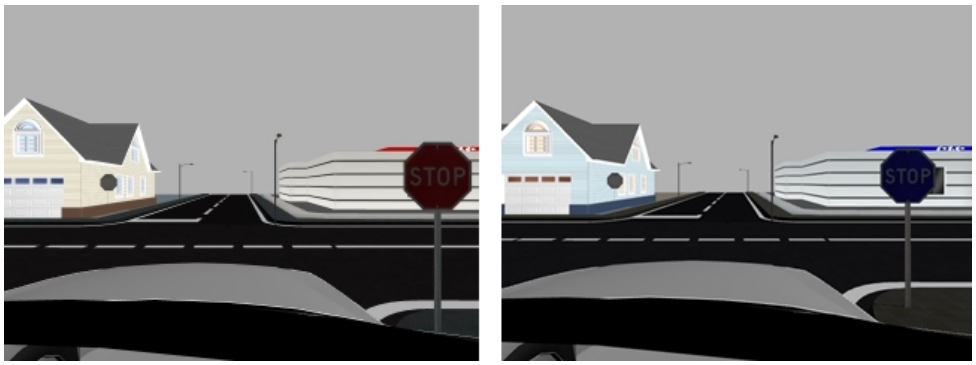

Figura 4.6: Imagen original en RGB y en HSV

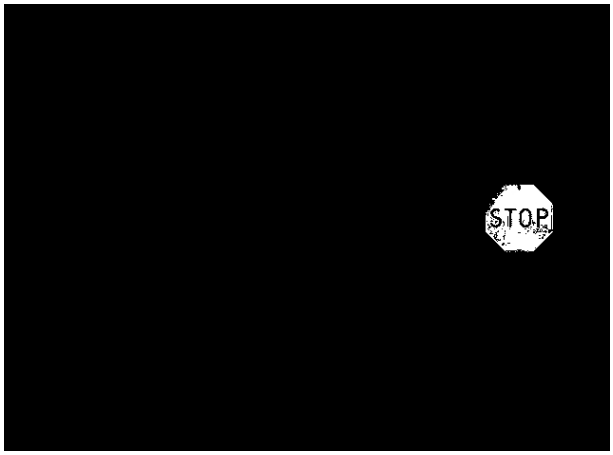

Figura 4.7: Imagen tras el filtro de color rojo

Figura 4.8: Imagen con cierre

De esta manera, obtenemos una imagen binaria con la forma de la señal de STOP en primer plano (blanco) (Figura 4.8). Esta imagen la compararemos con una plantilla de referencia para comprobar que se trata de la forma que queremos (octogonal). Para realizar esta comparación recortamos la imagen filtrada (Figura 4.9) para quedarnos sólo con la parte de la imagen que contiene la señal y la redimensionamos para que, tanto esta imagen como la plantilla (Figura 4.10), tengan el mismo tamaño y poder realizar correctamente el cotejamiento.

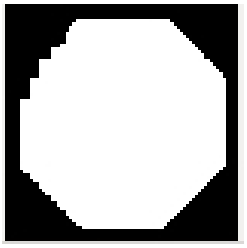

Figura 4.9: Señal de STOP recortada

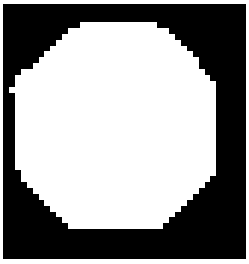

Figura 4.10: Plantilla de referencia

Una vez que hemos detectado la señal, el coche deberá frenar hasta situarse en el cruce de las dos carreteras. Para realizar el frenado hemos creado la función *"brake"*, mediante la cual, el coche irá reduciendo su velocidad lineal según el tamaño que vaya adquiriendo la señal de STOP detectada. Al principio, esta señal será pequeña ya que estará a lo lejos e irá aumentando de tamaño según el coche se vaya acercando hacia ella (Figura 4.11). Para saber el tamaño de la señal, realizamos un *bounding* alrededor de la señal (Figura 4.12) y, según el tamaño del recuadro, se llevarán a cabo las distintas fases del frenado, disminuyendo paulatinamente la velocidad.

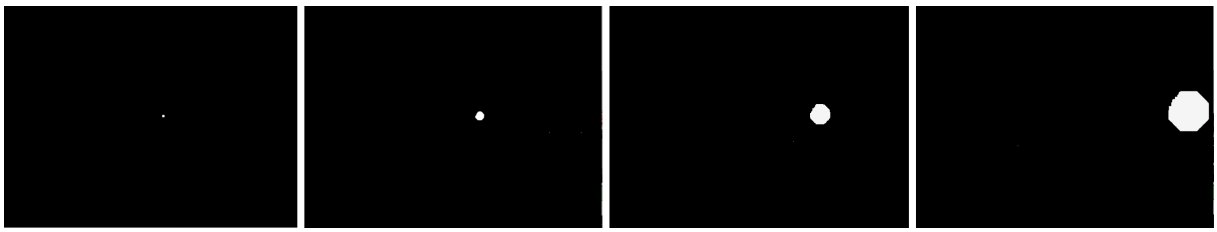

Figura 4.11: Aumento del tamaño de la señal de STOP

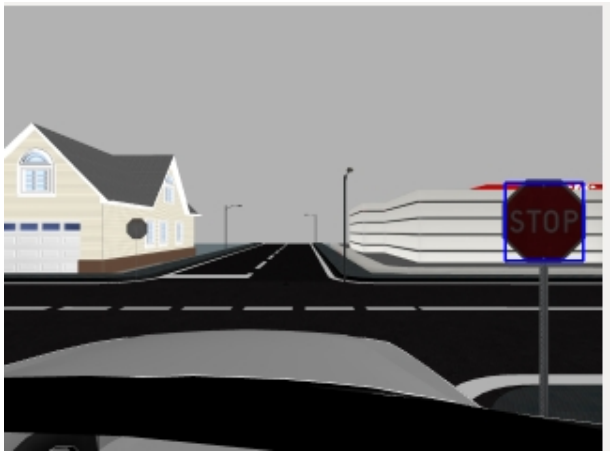

Figura 4.12: Señal de STOP recuadrada

4.4.2. Detección de otros coches

Después de frenar, el coche deberá saber si vienen o no otros vehículos por la otra carretera del cruce y luego reanudar su camino cuando no haya otros vehículos. Para distinguir a los otros coches que circulan por la carretera perpendicular, nos basamos en la resta de imágenes para detectar su movimiento (ya que en este escenario serán los únicos elementos que se moverán). Usamos las imágenes captadas por las cámaras situadas en los faros de los coches (tanto la izquierda como la derecha), las transformaremos a escala de grises y les aplicamos un suavizado mediante un filtro de Gauss para que la detección de movimiento sea más sencilla (Figura 4.13). Hemos creado la función *"motionDetection"* que se encarga de calcular la diferencia entre los distintos fotogramas; aplicar un umbral para segmentar la imagen y sólo quedarnos con las zonas de mucho movimiento (puede que existan vibraciones en las imágenes debido a la inercia del robot y este movimiento no interesa); y dilatar la imagen resultante para intentar eliminar el mayor número de partes negras dentro del coche detectado (Figura 4.14). Realizamos la resta de fotogramas con una diferencia de 5 fotogramas entre el actual y el anterior.

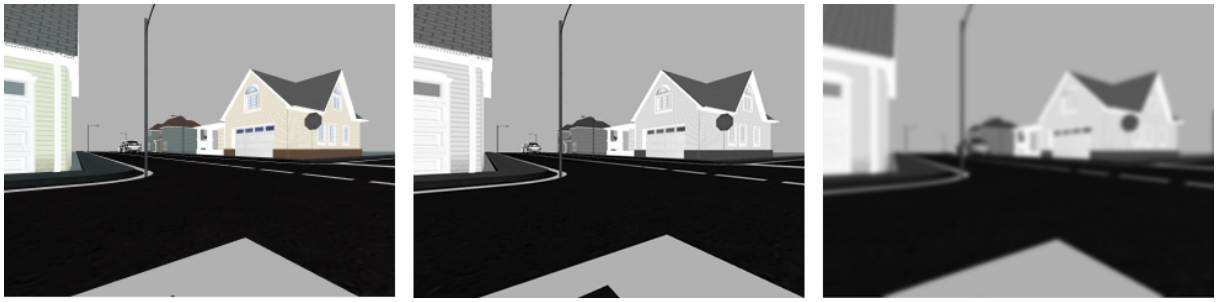

Figura 4.13: Imagen original, en escala de grises y suavizada

Figura 4.14: Proceso de detección de movimiento

4.4.3. Realización del giro

Para decidir si el coche comienza o no a moverse de nuevo hemos creado la variable dinámica *"detectionCar"*. Ésta irá disminuyendo su valor a lo largo del tiempo y aumentará si se detectan coches mediante la función *"findCar"* que usa la técnica descrita en la sección anterior. Cuando esta variable sea menor que el umbral establecido, el coche considerará que ha pasado un tiempo razonable sin detectar a otros vehículos y reanudará su ruta.

El coche deberá girar a la izquierda o a la derecha del cruce. La dirección del giro se escogerá de manera aleatoria mediante la función creada para ello *"chooseDir"*. Una vez elegida la dirección el coche llevará a cabo un giro de 45º y después comenzará a detectar la carretera para centrarse en carril derecho. Para realizar la detección de la carretera utilizaremos de nuevo la cámara frontal. A las imágenes captadas volveremos a aplicarles la función *"filterHSV"* pero esta vez los valores del filtro serán los correspondientes al color gris de la carretera (Figura 4.15). Para seleccionar estos valores también utilizamos

la herramienta "ColorTuner" de JdeRobot.

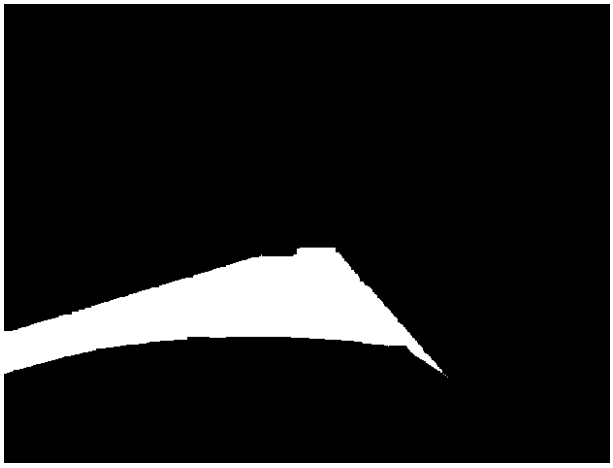

Figura 4.15: Carretera filtrada

Una vez hemos realizado el filtrado de color de la imagen y hemos obtenido la carretera en primer plano, necesitamos hallar los límites de la carretera para que, posteriormente, el coche sea capaz de posicionarse correctamente en el carril derecho. Para ubicar los bordes, hemos creado la función *"findRoad"* que recorrerá las columnas de la imagen y buscará los cambios de color que aparezcan en la fila 300 de la imagen. Hemos elegido esta fila para asegurar que contenga a la carretera (que estará en la parte inferior de la imagen). Si los píxeles cambian de negro a blanco se tratará del borde izquierdo y si cambian de blanco a negro se tratará del borde derecho. Después, deberemos encontrar la mitad de la carretera ya que marcará la separación de los dos carriles, y posteriormente, hallaremos la mitad del carril derecho (Figura 4.16). Estas operaciones las llevará a cabo la función creada *"findMidLane"*.

Figura 4.16: Punto central del carril derecho de color verde

Para saber si el coche está circulando por el carril adecuado, haremos un control del pilotaje basándonos en la desviación que haya entre el centro de la imagen y el punto que marca el centro del carril derecho. Para ello, hemos creado la función *"controlDesviation"* que aplicará una velocidad de giro a los motores directamente proporcional al valor de la desviación del vehículo en el carril por el que circula. También tendrá en cuenta el signo de la desviación ya que indica hacia qué lado se está moviendo el coche. Si es negativa, se estará desviando hacia la derecha por lo que tendremos que corregir girando a la izquierda, y si es positiva, haremos el giro a la derecha. Si la desviación es pequeña, el coche simplemente irá recto sin girar.

4.5. Experimentación

Para ejecutar esta práctica, es necesario abrir dos terminales y ejecutar los siguientes comandos:

1. Lanzar el simulador Gazebo:

```
gazebo stop.world
```

1b. Se puede arrancar solo el simulador sin la interfaz gráfica:

```
gzserver stop.world
```

2. Ejecutar la práctica y lanzar la interfaz gráfica (GUI):

```
python2 stop.py -- --Ice.Config=stop.cfg
```

En la Figura 4.17 se puede observar una ejecución típica, que sirve para validar experimentalmente la infraestructura, el nodo y la solución desarrollados. Ejecutando el algoritmo explicado anteriormente, el coche es capaz de frenar cuando reconoce la señal de STOP y después, en este caso, realizar el giro hacia la derecha.

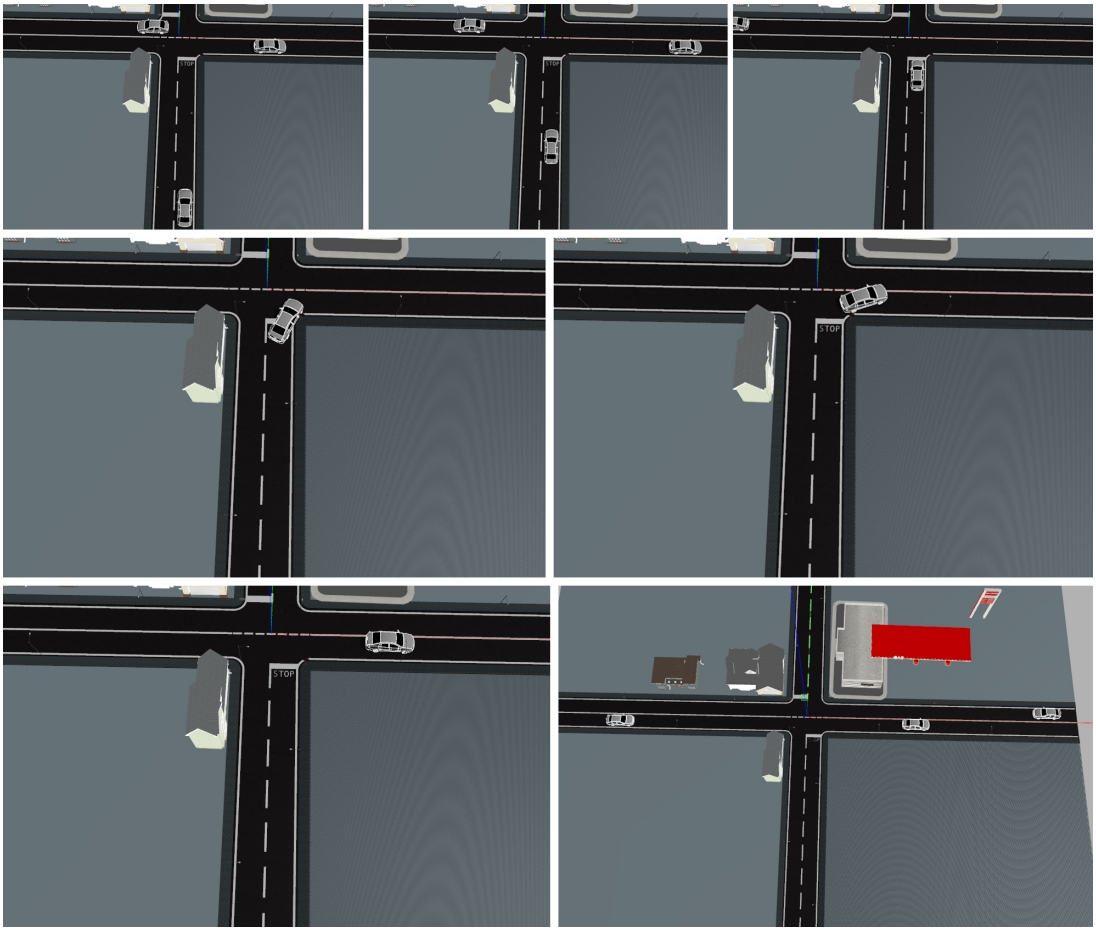

Figura 4.17: Ejecución típica (giro a la derecha)

Se han grabado varios videos representativos del comportamiento logrado, con el giro hacia la izquierda se puede ver en este vídeo [2] y con el giro a la derecha en este otro [3].

[2] https://www.youtube.com/watch?v=hF2i0rdlIqE
[3] https://www.youtube.com/watch?v=VXZtfHTGsW4

Capítulo 5

Práctica: Aspiradora autónoma con autolocalización

En este capítulo se expondrá el desarrollo de una nueva práctica para la plataforma de JdeRobot-Academy, denominada "Aspiradora autónoma con autolocalización". Se aborda el desarrollo de su infraestructura, su componente académico correspondiente, así como el evaluador automático creado y la solución de referencia programada.

5.1. Enunciado

En esta nueva práctica el objetivo principal es lograr que un robot aspirador autónomo consiga barrer la mayor superficie posible de un apartamento. Para ello deberá hacer uso de su capacidad de autolocalización y del mapa de dicho apartamento. Esta aspiradora además de poseer un sensor de posición que le permite saber su posición y orientación, tiene un motores con los que controla su velocidad lineal y su velocidad de giro. Además, posee un sensor láser que le permite medir la distancia a la que se sitúan los obstáculos y un sensor de choque que no se permiten en este ejercicio y no serán utilizados en el desarrollo de la solución de referencia.

En esta práctica el alumno deberá programar un algoritmo capaz de recorrer un gran porcentaje de la casa usando la capacidad de autolocalización del robot, que deberá ofrecer primero la planificación de una ruta en forma de zigzag. Y segundo un algoritmo de pilotaje para que la aspiradora siga dicha ruta. En la interfaz gráfica creada se puede observar el mapa de la casa, así como la posición actual de la aspiradora en dicho mapa y la superficie

recorrida. Este mapa estará disponible para realizar el algoritmo de control y también se usará para la medida de la nota final.

5.2. Infraestructura

En este apartado se describirá el entorno creado para poder realizar la práctica "Aspiradora autónoma con autolocalización". Primero se describirá el modelo del robot aspirador utilizado, incluyendo sus sensores y actuadores. Después se explicará el mundo simulado (un apartamento típico, con distintas habitaciones y muebles) por el cual se moverá la aspiradora.

5.2.1. Modelo Roomba

El modelo del robot aspirador encargado de ejecutar el algoritmo programado está basado en los robots Roomba de la empresa iRobot. Inicialmente, este modelo se diseñó para realizar un algoritmo de navegación sin autolocalización por lo que está inspirado en el Roomba de la serie 500 (ver Figura 5.1).

Figura 5.1: Roomba 500 de iRobot

El modelo Roomba de JdeRobot tiene instalados un sensor láser y un sensor de choque que no serán necesarios para la realización de la solución de esta práctica. Además, posee un sensor de posición que se utilizará para obtener tanto la orientación del robot como sus coordenadas en el sistema de referencia absoluto del simulador Gazebo. También dispone de un actuador de movimiento que permite que se desplace por el mundo de Gazebo.

En esta práctica se han empleado los siguientes *plugins* para este modelo:

- pose3di: Este *plugin* se emplea para obtener las coordenadas del robot en tiempo real y su orientación.
- motorsi: Este *plugin* interactúa con el componente permitiendo dotar al robot de velocidad lineal y velocidad de giro.

En la Figura 5.2 se puede observar el modelo *"Roomba"* de JdeRobot que mide aproximadamente 330 mm de ancho, 90 mm de altura, y un peso de 2.5 kg.

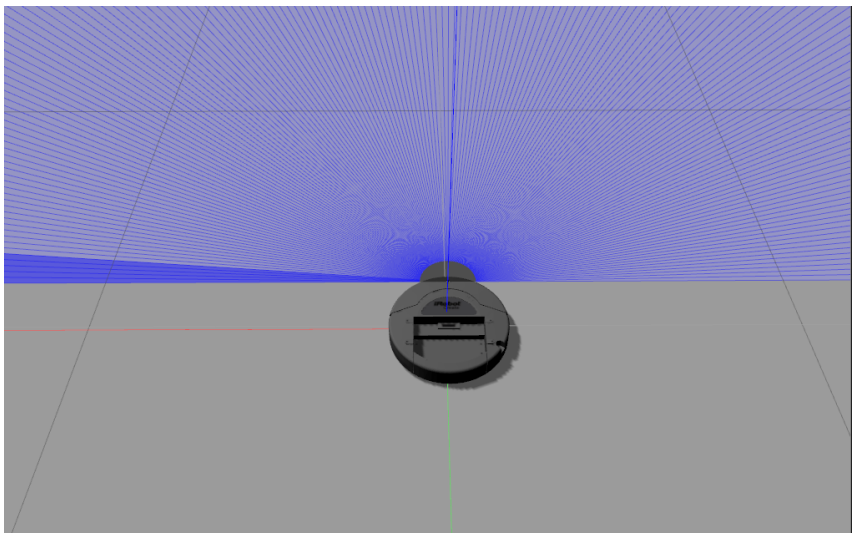

Figura 5.2: Modelo Roomba

5.2.2. Modelo house_int2

Ha sido necesario crear el modelo de un apartamento para que la aspiradora pueda navegar por ella. El modelo está basado en el modelo de casa que empleó Juan Navarro en su proyecto Fin de Carrera [36] en el mundo `GrannyAnnie.world` (Figura 5.3).

Inicialmente este modelo se utilizó para una navegación sin autolocalización por lo que fue necesario realizar algunas modificaciones. Debido a que el pilotaje de la aspiradora se basaba en un algoritmo pseudoaleatorio, cabía la posibilidad de que la aspiradora se saliese de la casa, por lo que se eliminaron las puertas exteriores la misma. También, se modificó la malla de colisiones de algunas paredes ya que la aspiradora podía atravesarlas.

Por último, se aumentó la masa del mobiliario de la casa ya que la masa que tenían inicialmente no era suficiente por lo que la aspiradora al chocar con los muebles los desplazaba. El nuevo modelo de casa se llama *"house_int2"* y se puede ver en la Figura 5.3, donde aparecen los cambios marcados en rojo.

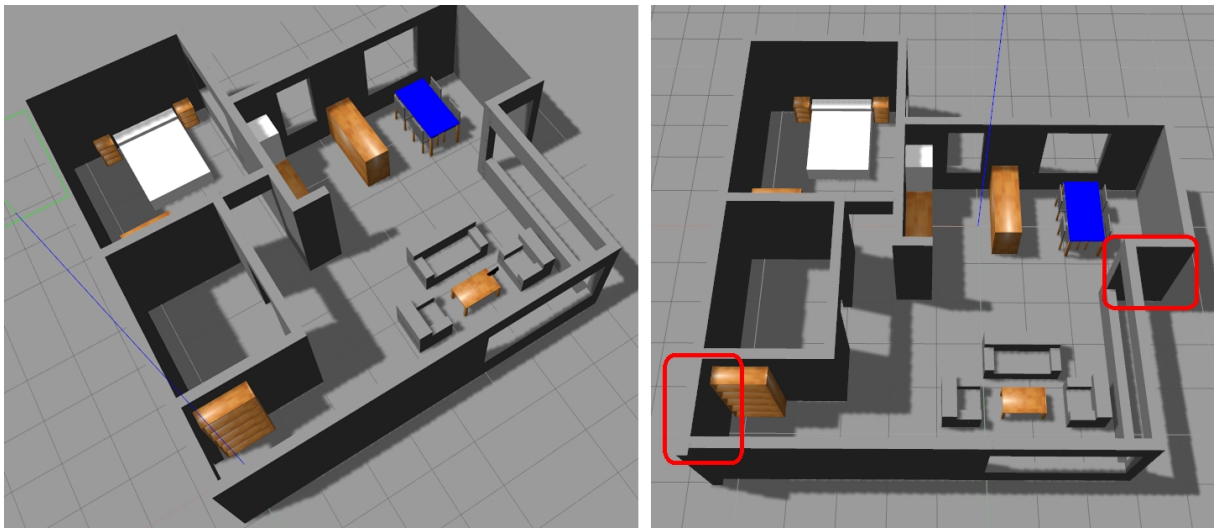

Figura 5.3: Modelo GrannyAnnie y modelo house_int2

5.2.3. Mundo de Gazebo

Para esta práctica se ha creado un mundo 3D en Gazebo formado por el modelo de la casa *"house_int2"* y por el robot aspirador *"Roomba"*, que ejecutará la solución desarrollada. También se ha añadido una fuente de luz mediante la etiqueta `<light>`.

Para tener este escenario se ha creado el siguiente mundo en Gazebo llamado `vacuum.world` [1]:

```
<?xml version="1.0" ?>
<sdf version='1.4'>
    <world name='Vacuum'>
        <include>
            <uri>model://roomba</uri>
```

[1] https://github.com/RoboticsURJC-students/2016-tfg-irene-lope/blob/master/Vacuum_Practice/vacuum.world

```
            <pose>5.04 3.80 0 0 0 -1.57 </pose>
        </include>
        <include>
            <uri>model://house_int2</uri>
            <pose>0 0 0 0 0 0</pose>
        </include>
        <include>
            <uri>model://ground_plane_transparent</uri>
        </include>

        <light name='sun' type='directional'>
        </light>

        <scene>
            <ambient>0.4 0.4 0.4 1</ambient>
            <background>0.7 0.7 0.7 1</background>
            <shadows>1</shadows>
        </scene>

        <gui fullscreen='0'>
          <camera name='user_camera'>
            <pose>0.126197 6.13852 18.8314 0 1.08764 -2.14299</pose>
            <view_controller>orbit</view_controller>
          </camera>
        </gui>
  </world>
</sdf>
```

En la Figura 5.4 se puede observar el mundo creado en Gazebo.

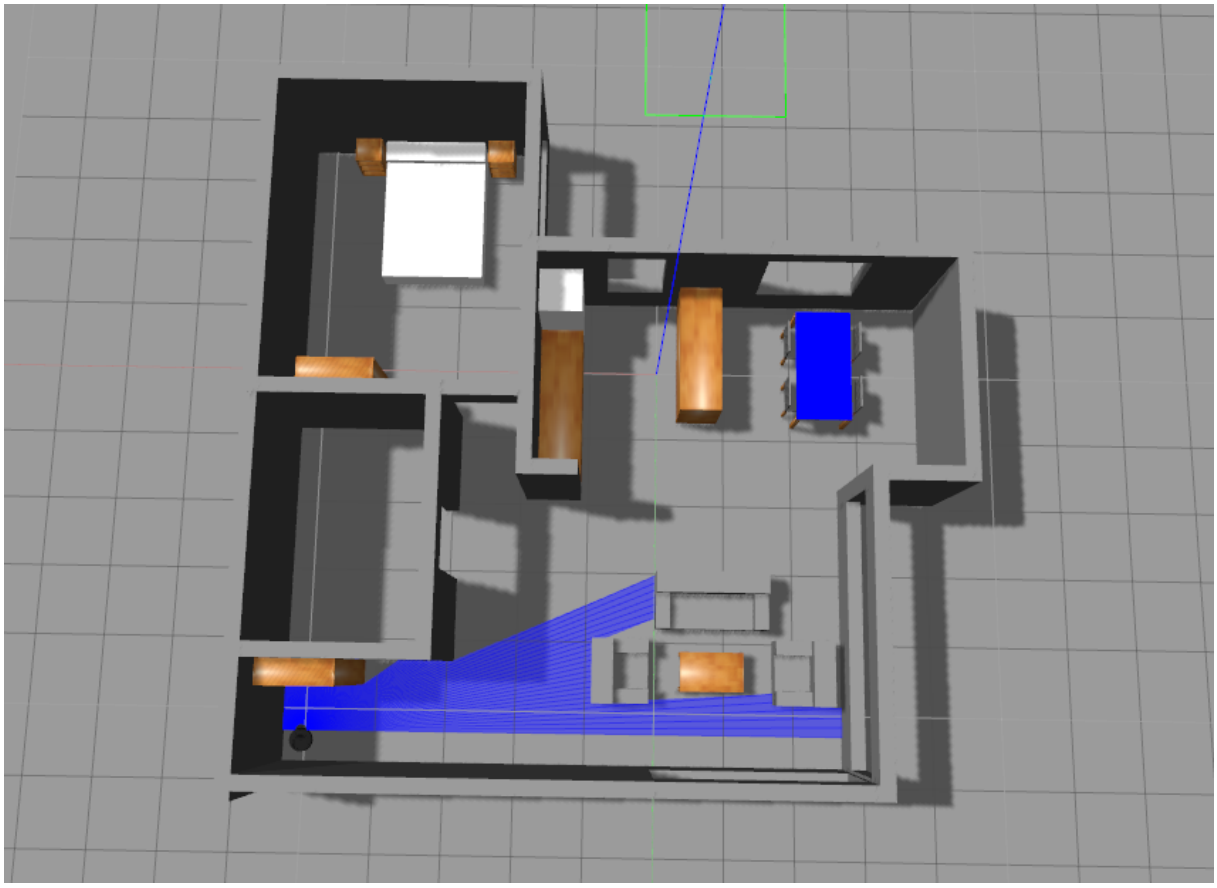

Figura 5.4: Mundo vacuum.world en Gazebo

5.3. Nodo académico

Se ha creado un componente académico para esta práctica que resuelve diversas funcionalidades: (a) muestra una interfaz gráfica al usuario, con distintos elementos que permiten depurar el código de manera más sencilla; (b) ofrece acceso a sensores y actuadores en forma de métodos simples ocultando el *middleware* de comunicaciones; (c) incluye código auxiliar que ayuda a programar la solución. El componente deja todo listo para que el estudiante sólo tenga que incorporar su código rellenando el método *"execute"* en el fichero `MyAlgorithm.py`.

El nodo académico emplea dos hilos de ejecución, de esta manera se pueden realizar todas las tareas necesarias para el funcionamiento correcto de la práctica. Los hilos utilizados son los siguientes:

- Hilo de control: se encarga de actualizar de manera constante los datos captados por los sensores y los datos enviados a los actuadores. Debido a que se trata de una solución reactiva es necesario que el intervalo de actualización de dichos datos sea un tiempo muy corto, en este caso, 50 ms. Si se fijara un tiempo muy grande podría ocasionar errores en la trayectoria del robot.
- Hilo de la GUI: este hilo actualiza la interfaz gráfica que se muestra al usuario. Este intervalo de actualización debe de ser también corto ya que la interfaz gráfica es una herramienta que utiliza el programador para depurar su código y debe de mostrar de manera fiable los datos del robot en tiempo real. Este intervalo también es de 50 ms.

5.3.1. Interfaz de acceso al hardware

Este componente ofrece al programador del algoritmo este API de sensores y actuadores:

- *pose3d.getX()*: Permite obtener la posición absoluta del robot en el eje X.
- *pose3d.getY()*: Permite obtener la posición absoluta del robot en el eje Y.
- *pose3d.getYaw()*: Permite obtener la orientación del robot con respecto al sistema de referencia de Gazebo.
- *motors.sendV()*: Para establecer la velocidad lineal.
- *motors.sendW()*: Para establecer la velocidad de giro.

5.3.2. Fichero de configuración

Es necesario crear un archivo de configuración (`vacuum.cfg`) para el nodo donde se indican los puertos utilizados por los distintos *plugins*, la velocidad lineal máxima y la velocidad angular máxima de los motores:

```
Vacuum.Motors.Proxy = Motors:default -h localhost -p 9003
Vacuum.Pose3D.Proxy = Pose3D:default -h localhost -p 9003
Vacuum.Laser.Proxy = Laser:default -h localhost -p 9003
```

```
Vacuum.Bumper.Proxy = Bumper:default -h localhost -p 9003
Vacuum.Motors.maxV = 5
Vacuum.Motors.maxW = 20
```

Gracias a la nueva configuración de JdeRobot en su versión actual, se puede usar el mismo puerto para diferentes *plugins*. En esta práctica, se emplea el puerto 9003 para todos los plugins y se ha establecido la velocidad máxima de tracción y de rotación.

5.3.3. Interfaz gráfica

La interfaz gráfica de usuario (GUI) de la práctica sirve para representar información importante de ayuda para resolver de manera más sencilla el algoritmo. Además, esta interfaz permite ejecutar el código que desarrolla la solución correspondiente. Se ha empleado la herramienta PyQt5 para su desarrollo.

En la parte izquierda de la interfaz se muestra el mapa del apartamento que la aspiradora tiene que recorrer (ver Figura 5.5). Es una imagen binaria en la que los obstáculos de la casa (muebles, paredes, etc) aparecen coloreados en negro (valor 0) y la superficie por la que la aspiradora puede pasar libremente aparece en blanco (valor 255). Este mapa se ha realizado manualmente, por lo que si se utilizara otra casa, habría que realizar un nuevo mapa. En este mapa también aparece un triángulo con las aristas en color rojo que indica la posición y la orientación de la aspiradora en tiempo real. Por último, también se marca en color azul, las zonas de la casa por las que la aspiradora ya ha pasado.

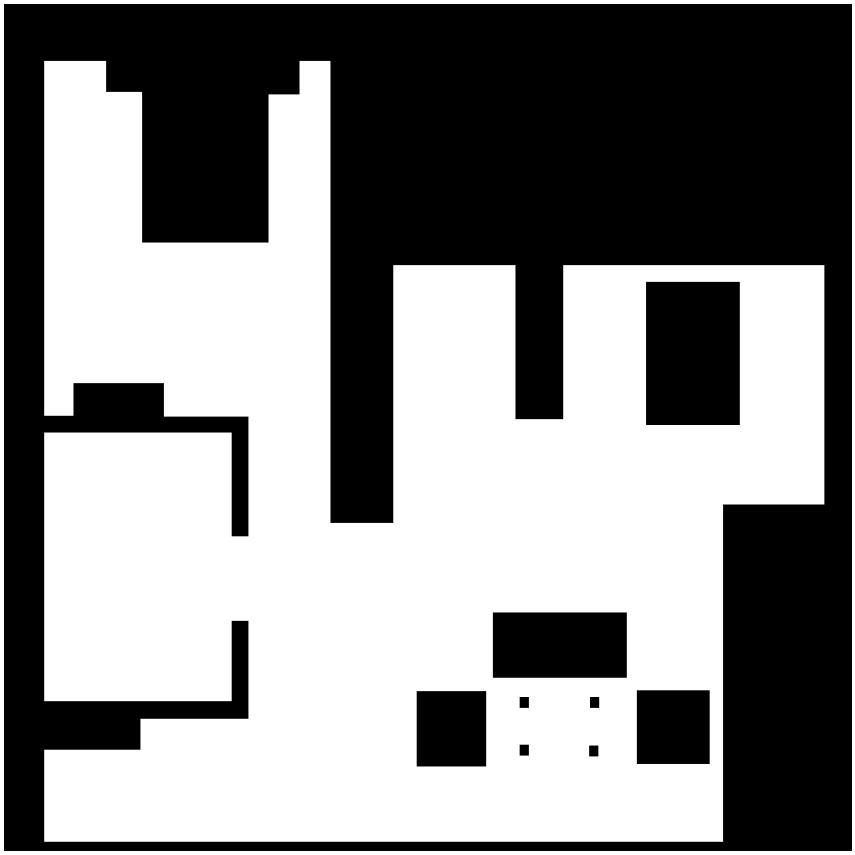

Figura 5.5: Mapa de la casa

Tanto para representar a la aspiradora en el mapa como las zonas por las que ha pasado es necesario pasar el sistema de coordenadas del mundo en Gazebo (3D) al sistema de coordenadas de la imagen del mapa (2D). En la Figura 5.6 se pueden ver el sistema de referencia de Gazebo y del mapa.

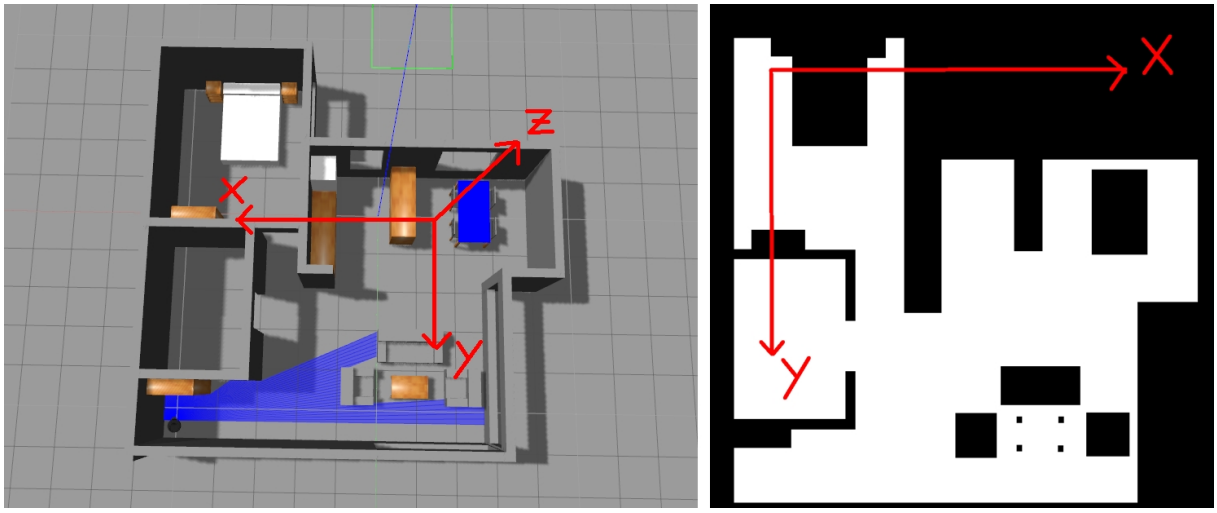

Figura 5.6: Sistema de referencia de Gazebo y sistema de referencia del mapa

Para realizar la conversión de un sistema a otro hay que pasar de una coordenada (x, y, z) en Gazebo a una coordenada (x', y') en el mapa. Para ello se ha aplicado una rotación de π grados sobre el eje Y y también ha sido necesario añadir una traslación para conseguir tener el origen de coordenadas (0,0) de la imagen en la esquina superior izquierda. En concreto, se ha realizado una traslación de 0,6 en el eje X y de -1 en el eje Y. La siguiente matriz de rotación y traslación del eje Y (ecuación 5.1) es la matriz utilizada. El ángulo α de la matriz será el ángulo de rotación en el eje Y; tx será la traslación en el eje X; ty será la traslación en el eje Y; y tz será la traslación en el eje Z.

$$\begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & tx \\ 0 & 1 & 0 & ty \\ -\sin(\alpha) & 0 & \cos(\alpha) & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

También es necesario multiplicar el resultado de aplicar la matriz por la escala, que indica el número de píxeles de la imagen que equivalen a un metro en el mundo de Gazebo. En este caso se ha establecido una escala de 50 píxeles por metro, por lo que la ecuación final (5.2) quedaría de la siguiente manera:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & tx \\ 0 & 1 & 0 & ty \\ -\sin(\alpha) & 0 & \cos(\alpha) & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} * scale \tag{5.2}$$

Gracias a esta transformación se puede pintar el triángulo, que representa a la aspiradora, aplicando la matriz de rotación y traslación a la coordenada de la posición de la aspiradora que se obtiene mediante el *pose3d*. Esta coordenada será el centro del triángulo, que es isósceles para mostrar cuál es la orientación del robot que estará orientado hacia el vértice del triángulo de ángulo menor.

Se pintarán en cada iteración en azul los puntos del mapa por donde ha pasado la aspiradora en cualquier momento. Se pintará un círculo en cada iteración dado que la aspiradora ocupa un determinado volumen. Los puntos por los que ha pasado la aspiradora se guardan en un array para saber por dónde ha pasado y pintarlos todos en cada iteración.

Por otro lado, a la derecha, hay un teleoperador bidimendional que permite mover el robot manualmente. La velocidad lineal se puede controlar moviendo el *joystick* en el eje vertical y la velocidad angular se controla moviendo el *joystick* en sentido horizontal.

En la parte inferior aparecen dos botones. El botón situado bajo el mapa de la casa permite tanto ejecutar como parar el código alojado en el archivo `MyAlgorithm.py`. Con el botón que está debajo se puede detener al robot si se está controlando manualmente con el teleoperador.

En la Figura 5.7 se muestra el resultado de la interfaz gráfica que verá el usuario en todo momento y en ella, en la esquina superior izquierda, el logo de JdeRobot.

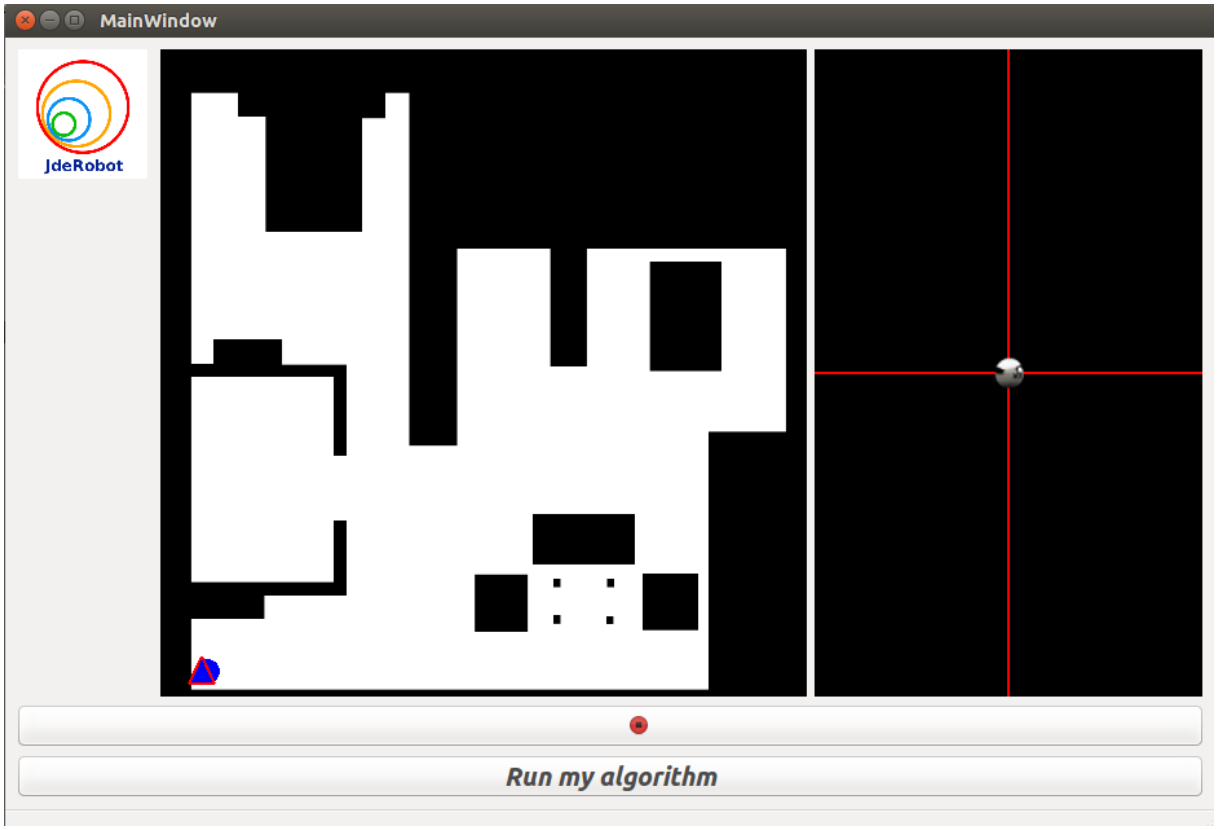

Figura 5.7: Interfaz gráfica

5.4. Solución de referencia

En esta práctica se plantea un problema de barrido de superficie en el que un robot aspirador tiene que ser capaz de recorrer la mayor parte posible del suelo de una casa haciendo uso de su capacidad de autolocalización y del mapa de dicha casa. La solución se ha programado en el fichero `MyAlgorithm.py`, en el método *"execute"*, que es el método principal de la solución.

La solución puede dividirse en dos partes principales: (a) planificación de la ruta; (b) pilotaje del robot.

5.4.1. Planificación de la ruta

Para calcular la ruta de la aspiradora será necesario crear un algoritmo en zigzag [1] [2] [3]. Para llevarlo a cabo se creará una rejilla o malla de navegación sobre el mapa de

la casa. Idealmente, el tamaño de las celdas que forman la malla es del mismo tamaño que el robot, pero en esta solución es ligeramente inferior al tamaño de la aspiradora (17 x 17 píxeles), en concreto, miden 16 x 16 píxeles. Esto es debido a que las medidas de la celda deben ser un número par para poder realizar correctamente el cálculo y de esta manera, nos aseguramos de que no queden zonas sin barrer ya que si la celda ocupase más que la aspiradora quedarían zonas sin recorrer entre los trayectos del zigzag.

El mapa que la aspiradora utilizará tanto para comprobar el valor de las celdas como para ir marcando las zonas por las que ya ha pasado, será un mapa de la casa con los obstáculos dilatados (Figura 5.8). Utilizando la operación morfológica de erosión de las zonas blancas conseguimos que la parte negra del mapa, es decir, los obstáculos, se amplíen ligeramente más que la mitad del tamaño de la aspiradora para asegurar que se puede tratar como un punto sin que se choque con ningún obstáculo (ya que la aspiradora ocupa más que las celdas y tenemos que dar un margen de error para la navegación). En la Figura 5.8 aparece la diferencia entre el mapa original y el mapa con la dilatación de los obstáculos.

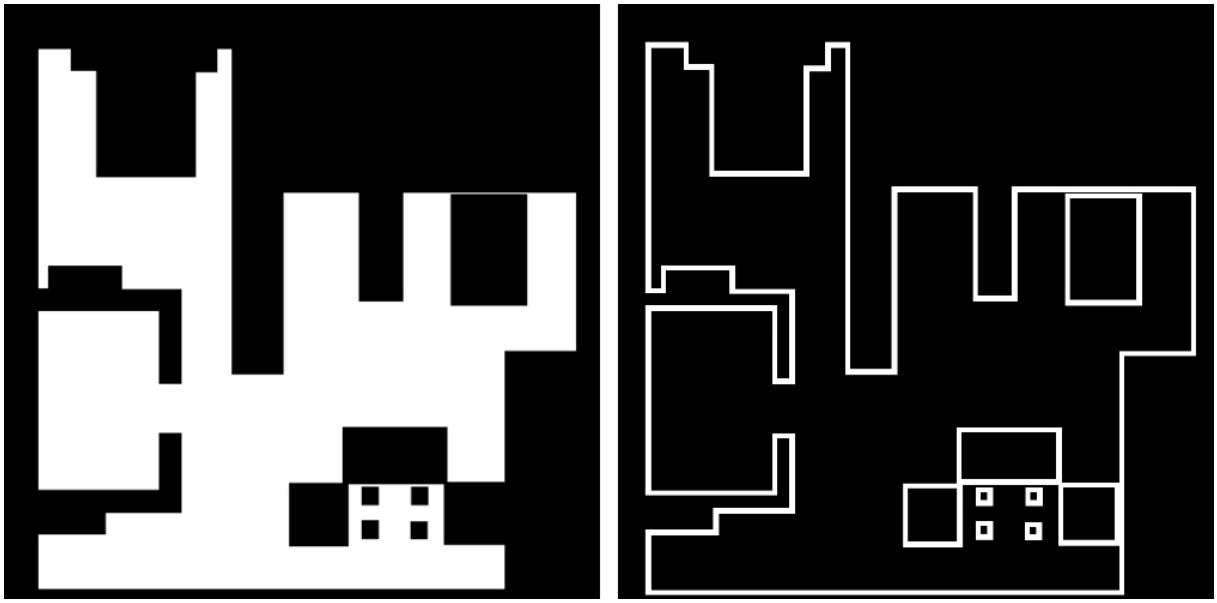

Figura 5.8: Mapa con los obstáculos dilatados y diferencia con el mapa original

En primer lugar calcularemos la celda en la que se encuentra la aspiradora a partir de su posición inicial. Para realizar la transformación de coordenadas de Gazebo a píxeles en el mapa hemos creado la función *"coord2pix"* que aplicará la ecuación (5.2), y para realizar

la transformación inversa, es decir, de píxeles a coordenadas, hemos creado la función *"pix2coord"* que aplicará la matriz de rotación y traslación inversa. Una vez ubicada la aspiradora en el mapa y creada la primera celda, se irán calculando las celdas contiguas a ésta mediante la función *"calculateNeigh"* que calculará las celdas vecinas: Norte, Este, Oeste y Sur. En la Figura 5.9 se puede ver cómo queda la disposición de las celdas en el mapa, siendo C la celda donde se encuentra la aspiradora, N la celda Norte, E la Este, O la Oeste y S la Sur.

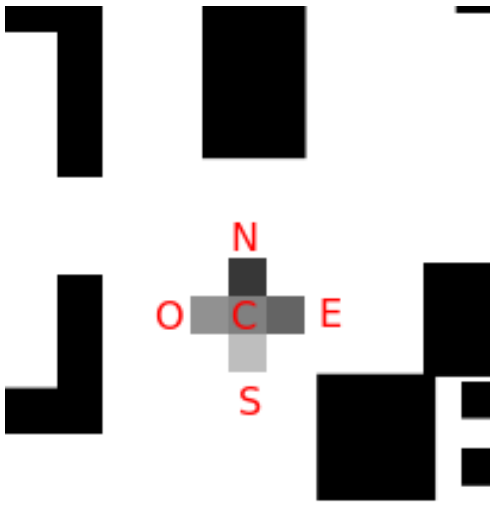

Figura 5.9: Ejemplo de vecindad de las celdas

Las celdas podrán tener tres valores:

- Obstáculo: Al menos uno de los píxeles que forman la celda es negro, es decir forma parte de un obstáculo de la casa.
- Obstáculo virtual: La aspiradora ya ha pasado por esta casilla y la marcará de color gris en el mapa.
- Libre: Todos los píxeles de la celda son blancos, es decir, no contienen ningún tipo de obstáculo (ni normal ni virtual).

Para saber el valor que tiene cada casilla hemos creado la función *"checkCell"* que comprobará todos los píxeles de la celda e indicará cuál es su valor correspondiente.

Mediante la función *"zigzag"* calcularemos la siguiente celda a la que tiene que moverse la aspiradora llevando a cabo la lógica del zigzag, y hasta que no haya conseguido

llegar a ese punto, no se calculará la siguiente casilla. Este algoritmo consistirá en que la aspiradora avanzará hacia adelante siempre y cuando la casilla Norte esté libre. Si encuentra un obstáculo (normal o virtual), la aspiradora se moverá hacia su derecha, es decir, hacia la celda Este y después avanzará hacia el sur hasta que vuelva a encontrar otro obstáculo, en cuyo caso, volverá a desplazarse a la celda Este y después hacia el norte y así sucesivamente. En el caso de que la celda Este estuviera ocupada, la aspiradora avanzará hacia la celda Oeste. En la Figura 5.10 se muestra un ejemplo sencillo de cómo quedaría idealmente el algoritmo donde la aspiradora inicialmente estaría en la celda con el punto azul y terminaría en la del punto verde.

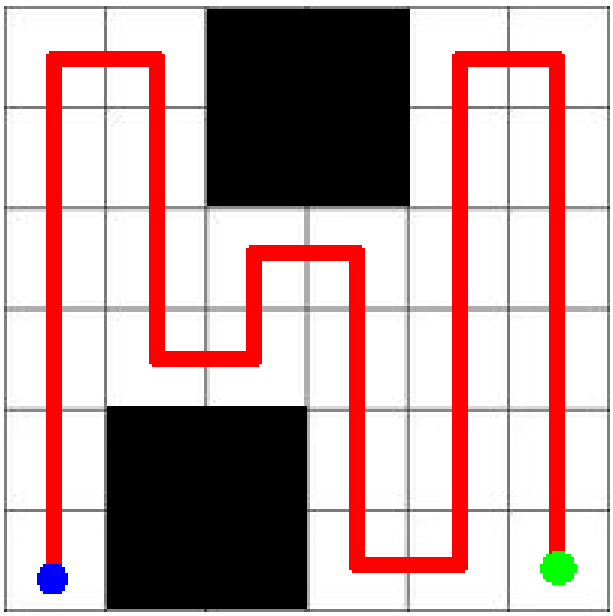

Figura 5.10: Ejemplo de zigzag

Seguirá realizando estos pasos hasta que llegue a una casilla en la que todas las celdas adyacentes sean obstáculos tanto normales como virtuales. A esta situación la denominaremos punto crítico.

5.4.1.1. Ruta de retorno

Si la aspiradora llega a un punto crítico y no puede avanzar más porque está rodeada de obstáculos, deberá ser capaz de calcular una ruta para retornar a otra zona del mapa y continuar con un nuevo zigzag. La función creada *"isCriticalPoint"* comprobará, cada vez que la aspiradora llegue a una celda nueva, si se trata o no de un punto crítico chequeando

todos los vecinos de dicha celda.

El retorno se calculará a partir de las celdas que ya ha recorrido la aspiradora. De esta manera aseguramos que todas las celdas están libres de obstáculos. Según vaya avanzando la aspiradora, guardará todas las celdas por las que ha ido pasando mediante la función *"savePath"*. Además, guardará las celdas que posiblemente puedan considerarse puntos de retorno. Un punto de retorno lo definimos como aquella celda que tiene uno o más vecinos libres. Para ello hemos creado la función *"isReturnPoint"* que indica si la celda correspondiente es un punto de retorno o no.

En el momento en que la aspiradora llega a un punto crítico, deberá comprobar todos los puntos de retorno que ha ido guardando ya que es posible que a lo largo del trayecto la aspiradora haya recorrido las celdas vecinas de algunos de estos puntos. Para ello hemos creado la función *"checkReturnPoints"* que comprobará si los puntos de retorno guardados tienen o no celdas vecinas libres, en cuyo caso se eliminarán.

Para decidir a qué punto de retorno dirigirse, nos basaremos en elegir el punto más cercano a la posición de la aspiradora. Hemos creado la función *"checkMinDist"* que mide la distancia euclídea (ecuación 5.3) entre la aspiradora y todos los puntos de retorno que haya y elige el punto más cercano.

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{5.3}$$

Una vez elegido el punto al que retornar, la aspiradora mediante la función *"goToReturnPoint"* se dirijirá a dicho punto. Esta función llevará a cabo el cálculo de las distintas celdas a las que la aspiradora tiene que desplazarse y de verificar si la aspiradora ha llegado a ellas. Para calcular la trayectoria de retorno hemos creado la función *"searchReturnPath"* que se encarga de comprobar si existe visibilidad entre la celda de retorno y la celda donde se encuentra la aspiradora. Dos puntos tienen visibilidad entre sí si los puntos de la recta que los une están libres de obstáculos. Si existe visibilidad directa entre el punto de retorno y la posición de la aspiradora, la trayectoria de retorno consistirá sólo en estas dos celdas ya que habría vía libre de obstáculos entre ambos. En el caso contrario, sería necesario encontrar celdas intermedias. Estas celdas se buscarían en el camino que ha seguido la aspiradora ya que son celdas sin obstáculos. Cuando se encontrase una celda que tenga

visibilidad con la aspiradora también se comprobaría si tiene visibilidad con la celda de retorno. Si hay visibilidad con ambas sólo sería necesario una celda intermedia, pero si no, se volvería buscar la visibilidad entre las celdas del camino y la celda intermedia y así sucesivamente hasta conseguir encontrar la ruta con celdas que tengan visibilidad entre sí.

Para calcular la visibilidad entre dos celdas hemos creado la función *"visibility"*. Esta función comprobará si la recta que une a las dos celdas está libre de obstáculos. Como es imposible comprobar todos los puntos que contienen a la recta, se comprobarán los que hay cada 10 cm. Dichos puntos se calcularán mediante la ecuación (5.4) donde A y B son los puntos de inicio y final la recta, P es el punto intermedio y S es el paso, es decir, la distancia entre A y P. Esta ecuación la llevará a cabo la función *"pointOfLine"*. Para constatar si alguno de los puntos de la recta forma parte de un obstáculo, hemos creado la función *"isObstacle"*, que comprobará si el punto corresponde a un píxel blanco o negro en el mapa.

$$P = A + S(B - A) \tag{5.4}$$

Para comprobar la visibilidad entre dos celdas, utilizamos un mapa de la casa con los obstáculos dilatados el tamaño de la aspiradora para evitar que ésta se choque con las esquinas de los obstáculos (Figura 5.11).

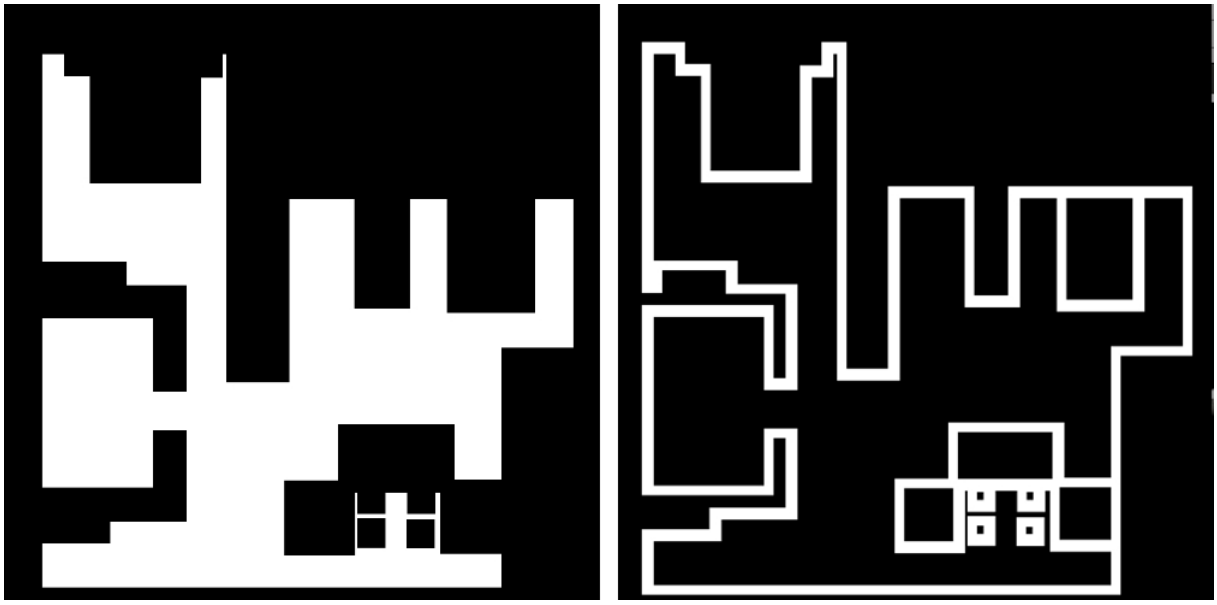

Figura 5.11: Mapa usado para la visibilidad entre dos celdas y diferencia con el mapa original

En el mapa de la izquierda de la Figura 5.12 se puede ver un ejemplo de cómo el algoritmo va buscando las celdas intermedias que forman el camino de retorno ya que no existe visibilidad entre las celdas de retorno y la que contiene a la aspiradora. En el mapa de la derecha se muestra en color gris las celdas por las que la aspiradora ya ha pasado, en azul la celda donde se encuentra la aspiradora, en verde la celda de retorno, en amarillo la celda intermedia y en rojo la trayectoria calculada.

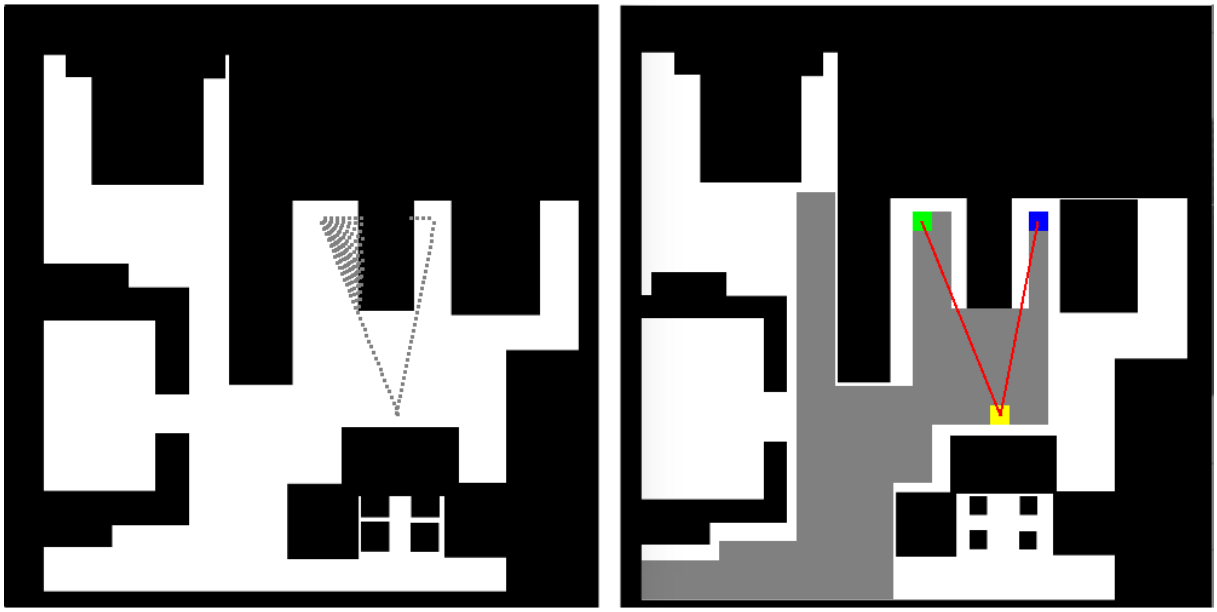

Figura 5.12: Cálculo de la ruta de retorno

El algoritmo de barrido finalizará cuando la aspiradora esté en un punto crítico y no queden puntos a los que retornar.

5.4.2. Pilotaje del robot

Una vez calculada la celda a la cual la aspiradora se tiene que dirigir, es necesario crear un algoritmo que se encargue del movimiento del robot. Para ello hemos creado la función *"goToCell"* que recibirá la celda destino y calculará la desviación (con la función *"calculateDesv")* de dicha celda respecto de la aspiradora. Para realizar este cálculo, necesitamos convertir las coordenadas absolutas de la celda a coordenadas relativas al robot mediante la función *"abs2rel"*. Esta función aplicará una rotación a las coordenadas según la orientación de la aspiradora.

Tanto la velocidad de giro como la velocidad lineal de la aspiradora se determinarán según la desviación que haya entre la celda y el robot. Si la desviación es elevada, estableceremos una velocidad de giro alta y pararemos la aspiradora para que se alinee rápidamente con la celda y no avance desviada. Según vaya disminuyendo la desviación, aumentaremos de manera gradual la velocidad lineal y reduciremos la de giro hasta conseguir que la aspiradora esté alineada con la celda. Dichos cambios de velocidad se llevarán

a cabo mediante las funciones *"controlDrive"* y *"controlDesv"*.

En el caso de que la aspiradora tenga que recorrer varias celdas consecutivas (en la dirección norte o sur), aumentaremos la velocidad lineal de manera proporcional a las casillas libres que tenga delante, disminuyendo así el tiempo que tarda la aspiradora en recorrer la casa entera. Para ello, hemos creado la función *"setV"*, que calcula las cuatro celdas siguientes en la dirección correspondiente y según las que haya libres, aumentará o reducirá la velocidad lineal de la aspiradora. En el caso de estar realizando una ruta de retorno, la velocidad se ajustará según la distancia entre la celda objetivo y la aspiradora, yendo más rápido cuanto más lejos se encuentren entre sí.

La función *"driving"* irá comprobando si la aspiradora ha llegado a la celda destino. Para realizar esta comprobación hemos creado la función *"checkArriveCell"* que dará por conseguida la celda si la aspiradora se encuentra en el centro de dicha celda. Es necesario establecer un margen de error ya que la aspiradora se encuentra en movimiento. Este umbral será mayor cuando la aspiradora tenga que recorrer varias celdas consecutivas (en la dirección norte o sur). El umbral será menor en el caso en el que la aspiradora tenga que girar o si en la siguiente celda hay un obstáculo (normal o virtual), para ajustar lo máximo posible la ruta en forma de zigzag.

5.5. Evaluador automático

Para esta práctica se ha creado un evaluador automático que, basándose en diferentes parámetros, brinda una calificación final a la solución realizada por el alumno. Al igual que en el componente académico, también se ha utilizado la herramienta PyQt5 para su creación. El evaluador muestra una interfaz gráfica en la que aparecen los parámetros que mide para el cálculo de la nota final.

En la esquina superior izquierda aparece una barra de progreso que se irá rellenando de color rojo proporcionalmente a la superficie recorrida por el robot. Para calcular el porcentaje recorrido, se tiene en cuenta que el 100 % de la superficie son todos los píxeles blancos del mapa. Según la aspiradora vaya moviéndose por la casa, se irá guardando el número de píxeles correspondientes a las zonas ocupadas y se calculará el porcentaje de la

casa al que correspondan. Esta información también aparece de manera numérica encima de la barra de progreso.

Al igual que en la interfaz gráfica del nodo académico, a la izquierda se muestra el mapa de la casa y en él, las zonas por las que va pasando la aspiradora pintadas en azul. Igualmente, ha sido necesario utilizar la ecuación (5.2) y aplicar una rotación de π grados sobre el eje Y, una traslación de 0,6 en el eje X y de -1 en el eje Y, y usar la misma escala. En este caso, no se muestra la orientación de la aspiradora.

En la parte derecha aparece un reloj digital que indica el número de segundos restantes hasta la finalización de la prueba. Al comenzar la prueba aparecerán 900 segundos (15 minutos). Se ha elegido este tiempo ya que se ha considerado el más idóneo para evaluar la práctica. Si se escogiera un tiempo mayor la evaluación de la práctica sería muy lenta y si se optase por un tiempo menor, no habría datos suficientes para calcular la nota realistamente.

Debajo de este reloj digital se muestra un reloj analógico que va avanzando cada segundo que pasa.

Por último, una vez que el tiempo de ejecución finaliza, a la derecha del reloj digital aparecerá un mensaje con la nota que ha obtenido el alumno. La calificación obtenida será directamente proporcional al porcentaje de superficie recorrida. Si pasados los 15 minutos, el alumno ha conseguido que la aspiradora recorra el 35 % de la superficie o más, obtendrá la máxima nota, un 10. Si por el contrario recorre menos superficie se realizará una regla de tres sabiendo que el 35 % es un 10.

Este evaluador se ha desarrollado en el archivo llamado `referee.py`. En la Figura 5.13 se muestra el resultado de la interfaz gráfica que se mostrará después de un tiempo ejecutando la solución creada y en ella, en la parte inferior, el logo de JdeRobot después de un tiempo ejecutando la solución creada.

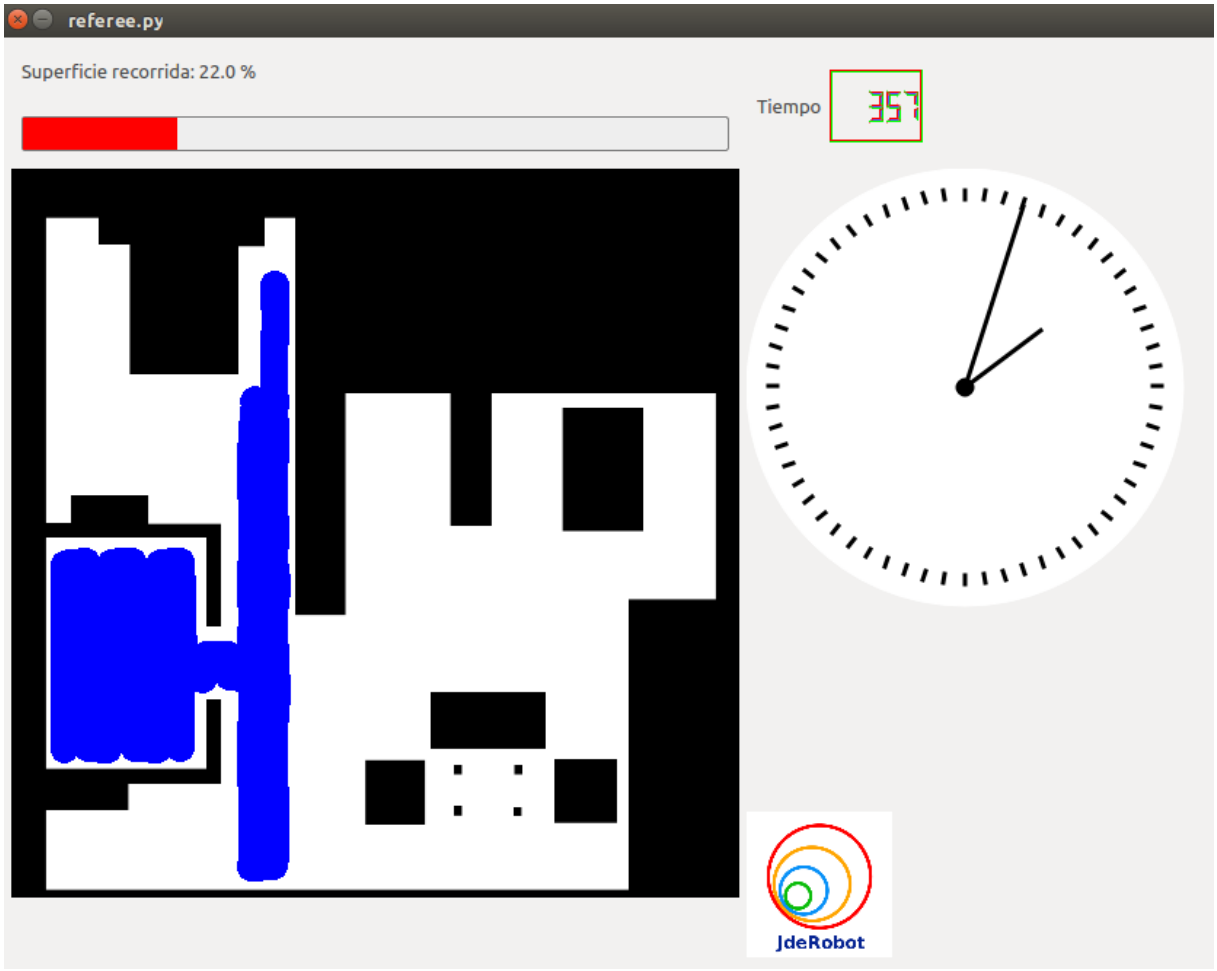

Figura 5.13: Evaluador automático

5.6. Experimentación

Para validar experimentalmente todas las piezas software desarrolladas, es decir, los drivers, el nodo académico, la solución de referencia y el evaluador automático se han realizado numerosas pruebas que se detallan en esa sección.

5.6.1. Ejecución típica

Para ejecutar esta práctica, es necesario abrir tres terminales y ejecutar los siguientes comandos:

1. Lanzar el simulador Gazebo:

```
gazebo vacuum.world
```

1b. Se puede arrancar solo el simulador sin la interfaz gráfica:

```
gzserver vacuum.world
```

2. Ejecutar la práctica y lanzar la interfaz gráfica (GUI):

```
python2 vacuum.py -- --Ice.Config=vacuum.cfg
```

3. Ejecutar el evaluador automático:

```
python2 referee.py -- --Ice.Config=vacuum.cfg
```

En la Figura 5.14 se puede observar el resultado de una ejecución típica de 15 minutos.

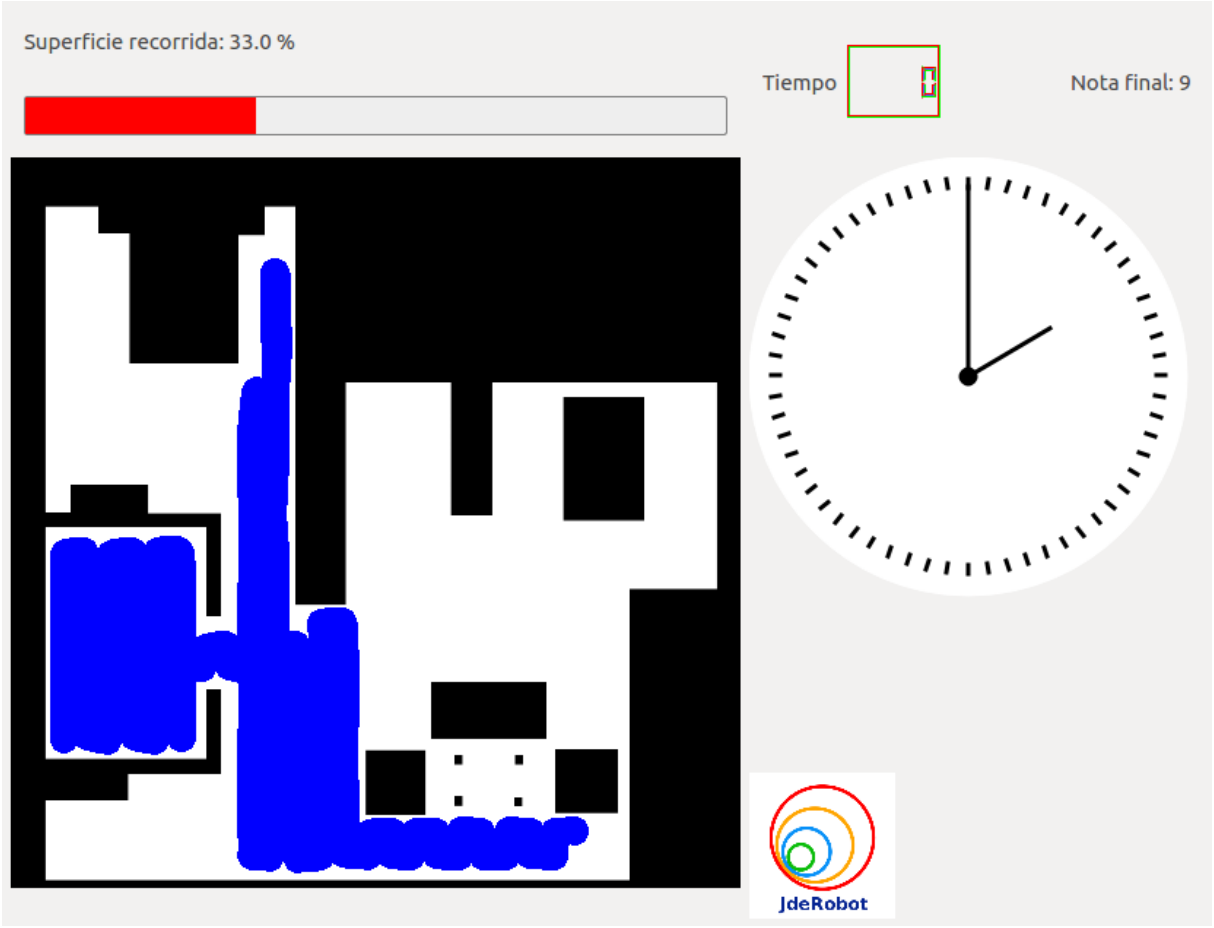

Figura 5.14: Ejecución típica

5.6.2. Experimentos de larga duración

Con una ejecución típica no se puede apreciar si realmente el algoritmo programado recorre la casa entera o no, por lo que se ha realizado una ejecución de larga duración para que de tiempo a llevar a cabo todo el algoritmo y que el robot recorra la casa entera.

En la Figura 5.15 se puede ver cómo con el algoritmo programado la aspiradora es capaz de recorrer la mayor parte de la superficie disponible, en concreto, el 74 %.

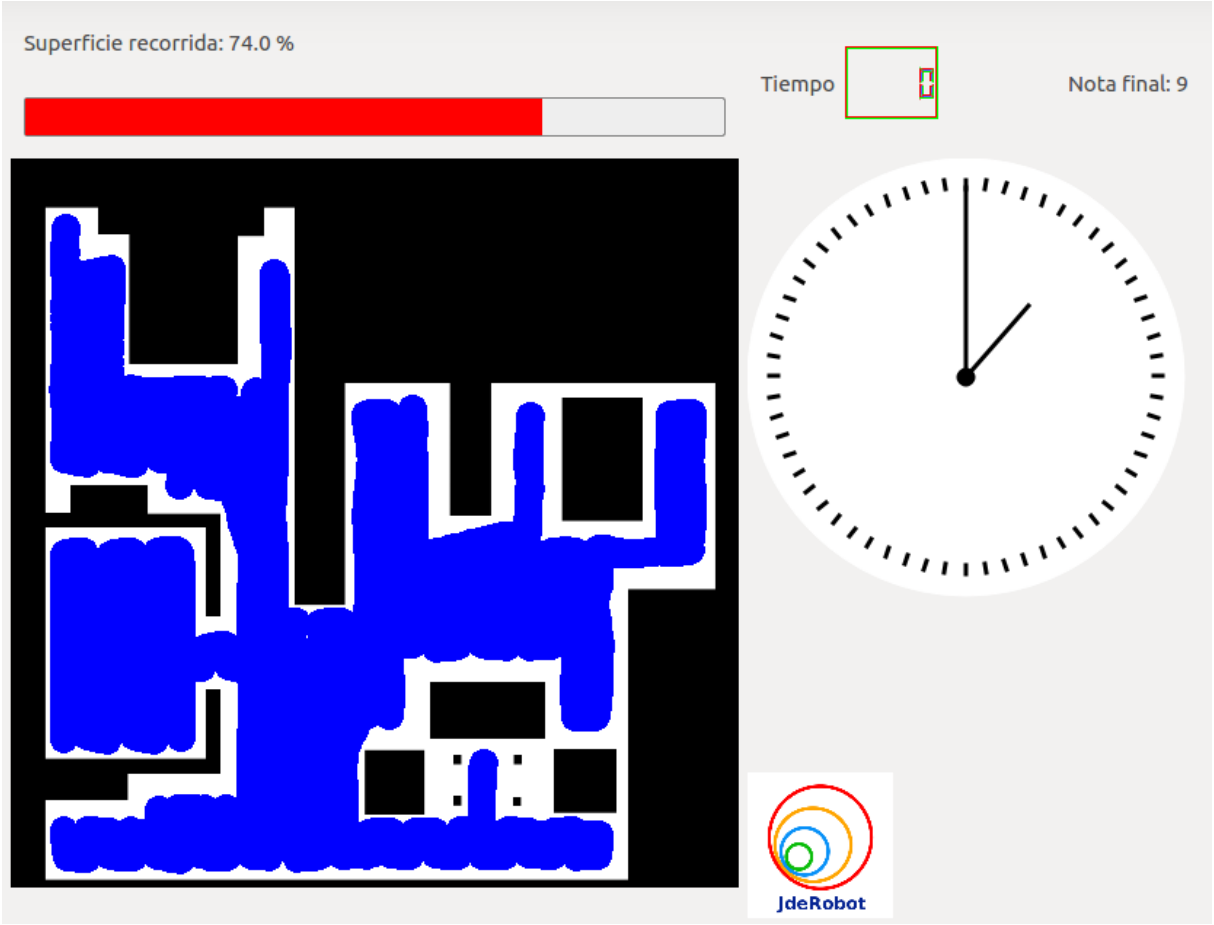

Figura 5.15: Ejecución de larga duración

La ejecución completa del algoritmo se puede ver en este vídeo [2].

[2] https://www.youtube.com/watch?v=sUT5ru4Ew_E

5.6.3. Inicio en distintos puntos de la casa

La posición inicial del robot influye en la cantidad de superficie recorrida, en la nota obtenida y en el tiempo que tarda en ejecutar completamente el algoritmo programado. Esto es debido a que la rejilla de navegación se va creando según las coordenadas iniciales del robot por lo que si por ejemplo, empieza más cerca de las paredes, puede que las celdas de la rejilla se ajusten más o menos a los obstáculos y sea capaz de recorrer más superficie. También influye en el tiempo porque puede que sea necesario hacer más rutas de retorno y esto aumentará el tiempo que tarda en recorrer la casa entera ya que pasará varias veces por zonas ya barridas.

Se han realizado varios experimentos cambiando la posición de inicio de la aspiradora. En la Tabla 5.1 se puede observar cómo la posición de inicio afecta a la superficie recorrida y al tiempo de ejecución necesario para barrer la casa entera.

Tabla 5.1: Resultados con distintas posiciones iniciales

Posición inicial	**Superficie recorrida**	**Tiempo de ejecución**	**Nº rutas de retorno**
Sala (5.04, 3.80)	74 %	48 min 01 seg	15
Dormitorio (5.04, 0.00)	75 %	47 min 07 seg	15
Salón (-2.35, 5.44)	75 %	51 min 10 seg	16
Comedor (-3.50, 1.45)	69 %	42 min 13 seg	14

En la Figura 5.16 aparecen marcadas las zonas de inicio seleccionadas. En verde aparece la posición denominada Sala, en rosa Dormitorio, en rojo Salón y en azul Comedor.

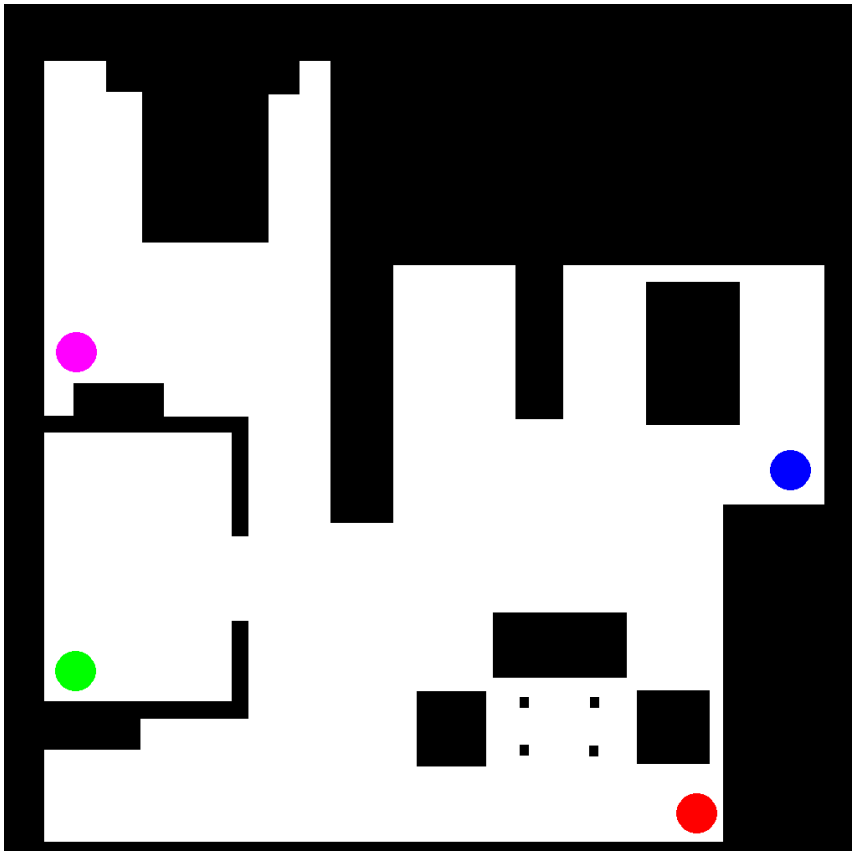

Figura 5.16: Posiciones de inicio

A continuación se muestran los resultados obtenidos con la ejecución del algoritmo en las distintas posiciones marcadas anteriormente.

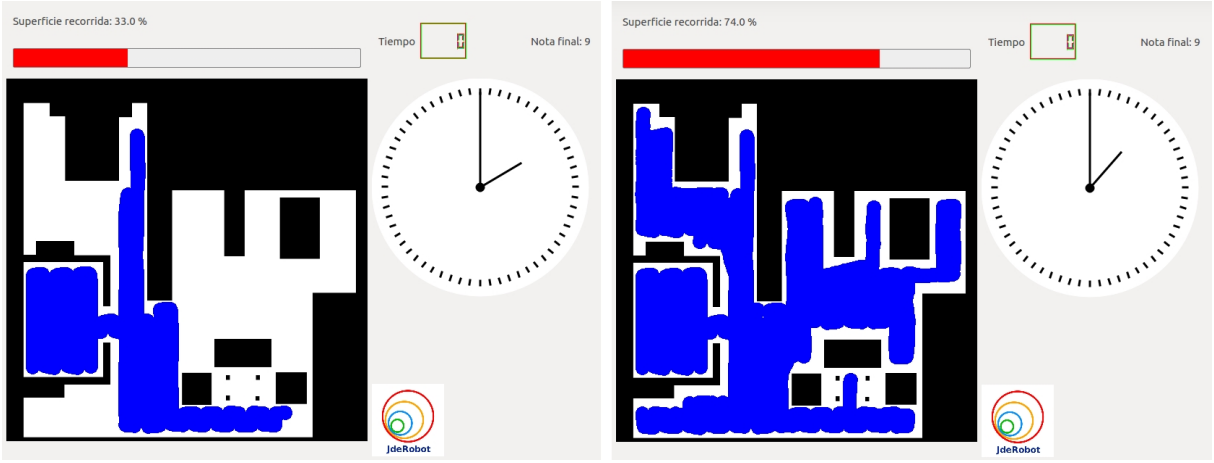

Figura 5.17: Ejecución típica y de larga duración en la posición inicial Sala

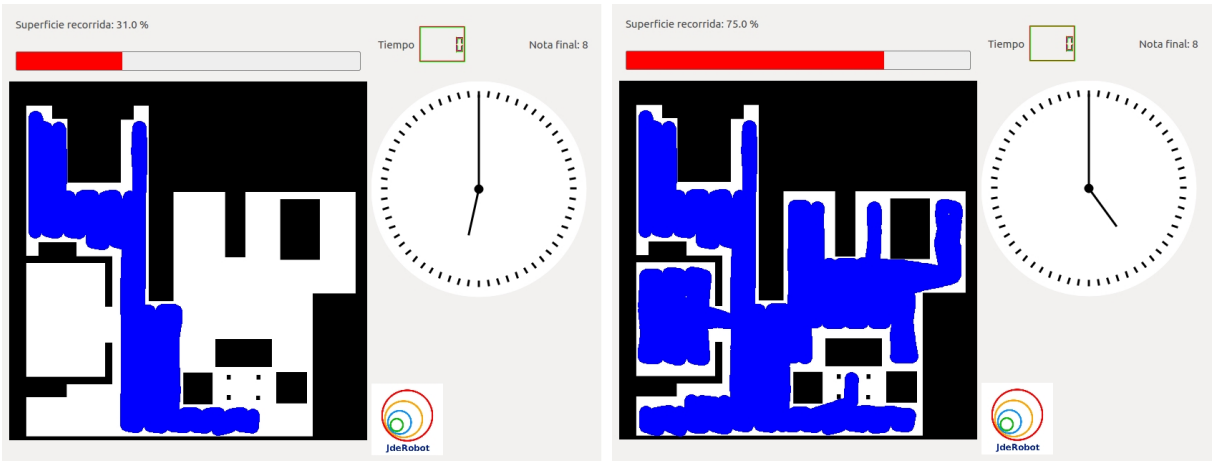

Figura 5.18: Ejecución típica y de larga duración en la posición inicial Dormitorio

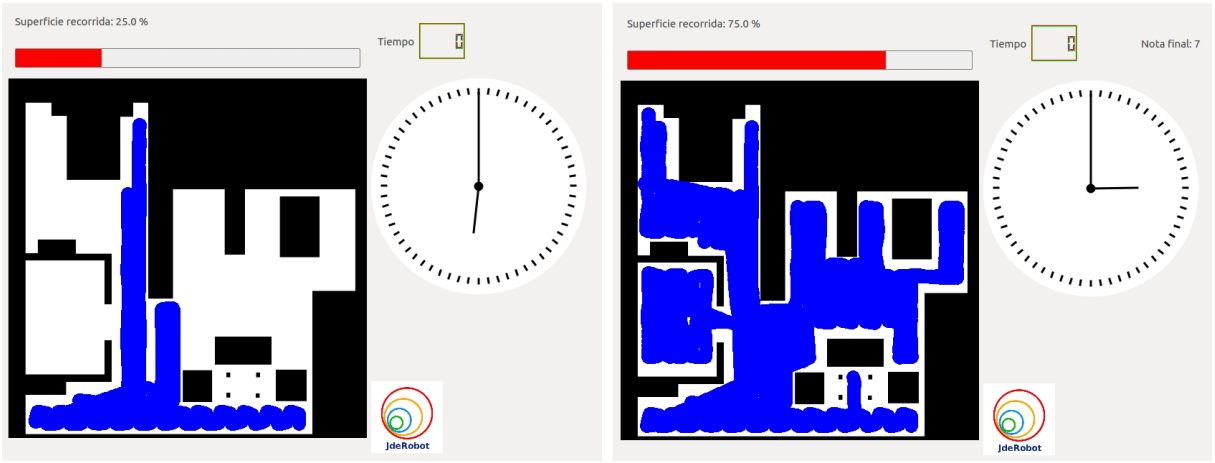

Figura 5.19: Ejecución típica y de larga duración en la posición inicial Salon

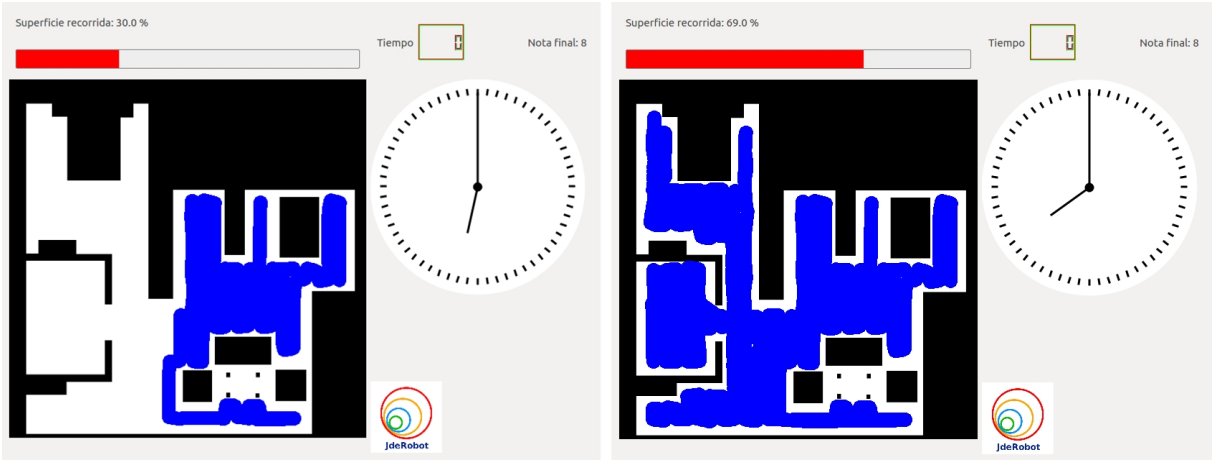

Figura 5.20: Ejecución típica y de larga duración en la posición inicial Comedor

Es interesante observar cómo el área barrida tras un periodo largo es más o menos el mismo, mientras que el área barrida en los quince primeros minutos sí depende significativamente de la posición de arranque.

Capítulo 6

Conclusiones

En este capítulo se recapitulan las conclusiones que se han obtenido realizando este proyecto. También se expondrán posibles mejoras que aplicar a las distintas prácticas desarrolladas.

6.1. Conclusiones

Con la realización de este proyecto se ha cumplido el objetivo global de crear dos nuevas prácticas destinadas a la docencia de robótica en el entorno de JdeRobot Academy. El objetivo de dichas prácticas es conseguir que el estudiante que las realice adquiera conocimientos relacionados con la programación de diferentes robots.

Por cada práctica se ha desarrollado una infraestructura y un nodo académico que simplifican la resolución del algoritmo permitiendo a los alumnos se abstraigan de problemas secundarios que conlleva la práctica, tales como la programación de la interfaz gráfica o la comunicación con el simulador Gazebo, con los sensores y actuadores del robot. Además de la interfaz gráfica y el código auxiliar, se ha desarrollado una solución de referencia por práctica.

En primer lugar, se ha creado la práctica "Coche autónomo negocia un cruce". Para esta práctica se ha creado un nuevo mundo en el simulador Gazebo que consiste principalmente en un cruce de carreteras donde se encuentra la señal de STOP que hay que detectar. Además, también aparecen dos coches que se mueven automáticamente a lo largo de una de las carreteras. También se ha programado una interfaz gráfica que permite

ver las imágenes captadas por las cámaras instaladas en el coche, consiguiendo que resulte más sencillo el tratamiento digital de las imágenes. La solución de referencia elaborada consta de tres partes principales. La primera consiste en el reconocimiento visual de la señal de STOP usando un filtro de color para detectar el color rojo y depués realizando una comparación con una plantilla de referencia para detectar la forma octogonal. En esta primera parte también se realiza el frenado del coche que paulatinamente reduce la velocidad de manera proporcional al tamaño que tenga la señal de STOP detectada. La segunda parte aborda la detección de otros coches mediante la detección de movimiento en las imágenes captadas. Por último, en la tercera parte, el coche realiza un giro a la izquierda o a la derecha (se elige de manera aleatoria) y, mediante el reconocimiento de la carretera se sitúa en el centro del carril derecho.

En segundo lugar, se ha creado la práctica "Aspiradora autónoma con autolocalización". Para esta práctica se ha modificado el mundo `GrannyAnnie.world`, que consiste en un apartamento básico con distintas habitaciones y muebles. También se ha desarrollado una interfaz gráfica que muestra el mapa de la casa y marca las zonas por las que la aspiradora ha pasado, así como la orientación del robot. La solución llevada a cabo consta de dos partes principales: la planificación de la ruta y el pilotaje de la aspiradora. La planificación de la ruta está basada en un algoritmo de barrido de superficies en zigzag. Esta planificación utiliza el mapa de la casa para ir calculando las casillas a las que tiene que ir llegando la aspiradora. De esta manera, cada vez que la aspiradora llega a una nueva celda, se calcula la siguiente según si las casillas que rodean a la aspiradora (con vecindad a 4) son obstáculos, están libres o ya se ha pasado anteriormente por ese punto. Además, esta planificación calcula el punto al que retornar si la aspiradora no puede avanzar más en el zigzag actual. Para el pilotaje del robot se calcula la desviación que hay entre la aspiradora y la celda a la que tiene que dirigirse y según dicha desviación, aumentará o disminuirá su velocidad lineal y de giro. Para esta práctica también se ha creado un evaluador automático que, dependiendo del porcentaje recorrido de la casa, otorgará una nota final para el alumno.

A nivel presonal, hemos aprendido a manejar el simulador Gazebo, creando nuevos mundos y modelos, y a utilizar la plataforma JdeRobot para programar el comportamiento de diferentes robots autónomos. Uno de los elementos fundamentales de aprendizaje de

esta plataforma es cómo se comunican los robots con los sensores y actuadores que poseen. Además, hemos aprendido a usar distintas bibliotecas de Python para desarrollar las prácticas. Al realizar este trabajo, también hemos comprendido cómo un problema de gran envergadura se puede resolver dividiéndolo en objetivos más pequeños y a solucionar problemas típicos de ingeniería realizando dsitintas pruebas y experimentos para afinar el algoritmo (aplicando conocimientos adquiridos durante el grado o bien buscando y entendiendo nueva información).

6.2. Trabajos futuros

Como posibles mejoras a las prácticas y trabajos futuros relacionados con éstas, se proponen las siguientes opciones. Posibles trabajos y mejoras relacionadas con la práctica "Coche autónomo negocia un cruce":

- Cuando se lleva a cabo el reconocimiento de la señal de STOP se podría detectar, a parte del color rojo y la forma octogonal, la palabra 'STOP'. De este modo, el reconocimiento de la señal sería todavía más fiable.
- Utilizar otras técnicas de percepción de movimiento más avanzadas para la detección de los coches como el uso de vectores de movimiento o el flujo óptico.
- Para disminuir el tiempo que el coche está parado en el cruce, se podría detectar el tamaño y el sentido de movimiento de los coches. De esta manera, si por ejemplo, se detectan coches pequeños significaría que éstos están lejos y que al coche le daría tiempo a realizar el giro en el cruce. También si se detecta un coche por la cámara izquierda que se dirige hacia la izquierda, aunque sea grande, indica que el coche ya ha pasado el cruce y nuestro coche podría ir detrás sin chocarse.
- Se podría aumentar la dificultad de la práctica acelerando la velocidad e incrementando el número de los coches que circulan automáticamente por la carretera.

Posibles trabajos y mejoras relacionadas con la práctica "Aspiradora autónoma con autolocalización":

- Durante la etapa de planificación de la ruta se podrían utilizar distintos tipos de algoritmos para el barrido de superficies. En vez de un recorrido en zigzag, se podría

utilizar un recorrido en espiral. Al principio, la aspiradora iría al lado de la pared y realizaría la espiral hacia el interior de cada habitación. Seguiría guardando los puntos de retorno de la misma manera que en el zigzag y el punto crítico para comenzar una nueva espiral sería cuando la aspiradora tuviese todas las casillas que la rodean barridas o con obstáculos.

- No proporcionar el mapa de la casa para realizar el algoritmo de barrido. De esta manera, la aspiradora iría creando el mapa según su posición y los datos obtenidos por el sensor láser que tiene instalado.
- Optimizar el algoritmo de retorno para que encuantre la ruta más corta posible.
- Se podría probar el algoritmo creado en una aspiradora real. Así, se podría observar de manera exacta como funcionaría la solución realizada. Se probaría en distintas habitaciones con diferentes obstáculos para lograr que el algoritmo fuera lo más robusto posible.

Bibliografía

[1] E. González, A. Suárez, C. Moreno, F. Artigue *Complementary Regions: a Surface Filling Algorithm.* Proceedings of IEEE International Conference on Robotics and Automation, 1996.

[2] Enrique González, Oscar Álvarez, Yul Díaz, Carlos Parra, Cesar Bustacara *BSA: A Complete Coverage Algorithm.* Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005.

[3] Abdallah Ntawumenyikizaba, Hoang Huu Viet, TaeChoong Chung *An Online Complete Coverage Algorithm for Cleaning Robots based on Boustrophedon Motions and A* search.* 2012 8th International Conference on Information Science and Digital Content Technology (ICIDT2012) , 2012.

[4] Maria Raquel L. de Guizado Robótica. `https://nextcomrobotics.wordpress.com/campo-de-aplicacion/aplicacion-de-la-robotica/`, 2012.

[5] Waymo (Google). `https://waymo.com`.

[6] Tesla. `https://www.tesla.com/es_ES/`.

[7] dyson. `https://www.dyson.es`.

[8] iRobot Corporation. `http://www.irobot.es/`.

[9] Félix Cerezo. *Esto es lo que es capaz de hacer el coche autónomo de Tesla.* `http://www.elmundo.es/motor/2016/10/24/580ddaa9468aeb0b2b8b45cf.html`, 2016.

[10] Michael Pooler. *Los robots de Amazon revolucionan la logística.* `http://www.expansion.com/economia-digital/innovacion/2017/08/29/59a458d7ca4741cf138b4622.html`, 2017.

[11] Miguel López. *Este es el primer almacén robotizado de Amazon en España.* `https://www.xataka.com/robotica-e-ia/este-es-el-primer-almacen-robotizado-de-amazon-en-espana`, 2017.

[12] Wiki de robótica. http://wiki.robotica.webs.upv.es/wiki-de-robotica/introduccion/historia/.

[13] ROS (Robot Operating System). http://www.ros.org.

[14] Orca Robotics. http://orca-robotics.sourceforge.net/.

[15] Ayssam Elkady and Tarek Sobh. *Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography.* https://www.hindawi.com/journals/jr/2012/959013/, 2012.

[16] Player Project, *Wikipedia.* https://en.wikipedia.org/wiki/Player_Project.

[17] The Player Project. http://playerstage.sourceforge.net/.

[18] JdeRobot. http://jderobot.org.

[19] Gazebo Simulator. http://gazebosim.org/.

[20] PCL (Points Cloud Library. http://pointclouds.org.

[21] The Construct. http://www.theconstructsim.com/.

[22] Arduino. https://www.arduino.cc/.

[23] Arduino, *Wikipedia.* https://es.wikipedia.org/wiki/Arduino.

[24] Simone Ceriani and Martino Migliavacca. Middleware in robotics. *Advanced Methods of Information Technology for Autonomous Robotics.*

[25] Herman Bruyninckx and Peter Soetens. The OROCOS Project. https://people.mech.kuleuven.be/~orocos/pub/stable/documentation/rtt/v1.10.x/doc-xml/orocos-overview.html, 2007.

[26] Jorge Coronado Vallés. *Desarrollo de aplicaciones embebidas de control en robots móviles.* PhD thesis, Universidad Politécnica de Valencia, 2013.

[27] J. Baillie, A. Demaille, Q. Hocquet, M. Nottale, and S Tardieu. The Urbi Universal Platform for Robotics. *Simulation, Modeling and Programming for Autonomous Robots*, pages 580–591, 2008.

[28] Alberto Manuel Mireles Suárez. Re-identificación de personas en redes de sensores RGBD,*Universidad de las Palmas de Gran Canaria*. file:///C:/Users/jessi/Downloads/0710907_00000_0000.pdf, 2015.

[29] Javier Nuño Simón. *Reconocimiento de objetos mediante sensor 3D Kinect*. PhD thesis, Universidad Carlos III de Madrid, 2012.

[30] AForge.NET Framework, *AForge.NET*. http://www.aforgenet.com/framework/.

[31] Galo Fariño R. Modelo Espiral de un proyecto de desarrollo de software, *Administración y Evaluación de Proyectos*. http://www.ojovisual.net/galofarino/modeloespiral.pdf, 2011.

[32] Modelo Espiral. http://modeloespiral.blogspot.com.es/, 2009.

[33] Detección de movimiento con OpenCV. https://programarfacil.com/blog/vision-artificial/deteccion-de-movimiento-con-opencv-python/.

[34] What is PyQt?, *Riverbank Computing Limited*. https://riverbankcomputing.com/software/pyqt/intro, 2016.

[35] Jan Bodnar. Introduction to PyQt5. http://zetcode.com/gui/pyqt5/introduction/, 2017.

[36] Juan Navarro Bosgos. *Construcción de Mapas 3D Compactos desde Sensores RGBD*. PhD thesis, Universidad Rey Juan Carlos.Ingeniería de Telecomunicación, 2015.

[37] Florencia Ucha. Definición de Robótica. https://www.definicionabc.com/tecnologia/robotica.php, 2009.

[38] Definición de Robot. http://conceptodefinicion.de/robot/, 2014.

[39] Marilyn Perdigón. El padre de la robótica, *Los primeros robots*. http://marilynpg.blogspot.com.es/, 2012.

[40] Historia. http://wiki.robotica.webs.upv.es/wiki-de-robotica/introduccion/historia/.

[41] Yanet Maritza Sagua Alanguía. Clasificación de los Robots, *Robótica Puno*. http://roboticapuno.blogspot.com.es/2013/01/clasificacion-de-los-robots.html.

[42] Clasificación de robots. http://wiki.robotica.webs.upv.es/wiki-de-robotica/introduccion/clasificacion-de-robots/.

[43] Generaciones de la Robótica, *Todo sobre la Robótica*. http://conozcamoslarobotica.blogspot.com.es/p/generaciones-de-la-robotica.html. [Accedido 5 de Agosto de 2017].

[44] Robots Militares, *Actualidad Gadget*. https://www.actualidadgadget.com/tag/robots-militares/.

[45] Víctor R. González. Definición del Robot Industrial, *Robots industriales*. http://platea.pntic.mec.es/vgonzale/cyr_0204/ctrl_rob/robotica/industrial.htm, 2002.

[46] Importancia de la Robótica en la Educación, *Educatronics. Ciencia, arte y arquitectura*. http://educatronics.com/publicaciones/importancia-de-la-rob%C3%B3tica-en-la-educacion.

[47] I. Moreno, L. Muñoz, J. Rolando, J. Quintero, K. Pittí, and J. Quiel. La robótica educativa, una herramienta para la enseñanza-aprendizaje de las ciencias y las tecnologías. *Revista Teoría de la Educación: Educación y Cultura en la Sociedad de la Información.*, 13(2):74–90, July 2012.

[48] J.M. Cañas, A. Martí, E. Perdices, F. Rivas, and R. Calvo. Entorno docente para la programación de la inteligencia de los robots. *Iberoamericana de Automática e Informática Industrial*, pages 1–12, 2016.

[49] L.F Bello and J.Gutierrez. Modelo en Espiral y Modelo Basado en Prototipos. https://prezi.com/y7slahvparel/modelo-en-espiral-y-modelo-basado-en-prototipos-para-el-desarrollo-de-software/, 2017.

[50] Sonia Posligua. Modelo en Espiral. https://es.slideshare.net/soniaposligua/modelo-enespiral, 2013.

[51] Carlos Santana. Historia de Python. https://www.codejobs.biz/es/blog/2013/03/03/historia-de-python, 2013.

[52] Python Software Foundation. Tutorial de Python. http://docs.python.org.ar/tutorial/3/real-index.html, 2017.

[53] Python, *EcuRed*. https://www.ecured.cu/Python, 2017.

[54] Python Software Foundation. Tutorial de Python. http://docs.python.org.ar/tutorial/3/real-index.html, 2017. https://programarfacil.com/blog/vision-artificial/deteccion-de-movimiento-con-opencv-python/

[55] Adrián González. *Planificación de Movimiento en Robótica Móvil utilizando retículas de estados*. PhD thesis, Universidad de Santiago de Compostela.Escola Técnica Superior de Enxeñaria, 2011.

[56] Julio Manuel Vega. *Navegación visual del robot Pioneer*. PhD thesis, Universidad Rey Juan Carlos.Ingeniería Técnica en Informática de Sistemas, 2005.

[57] Navegación autónoma, *Cuentos Cuánticos*. https://cuentos-cuanticos.com/2011/11/12/navegacion-autonoma/, 2011.

[58] Mariano Gómez Plaza. *Planificación óptima de movimiento y aprendizaje por refuerzo en vehículos móviles autónomos*. PhD thesis, Universidad de Alcalá. Escuela Politécnica Superior, 2009.

[59] J.M Cañas. *Programación de robots con la plataforma Jderobot*. PhD thesis, Universidad de Málaga, 2009. [Accedido 11 de Agosto de 2017].

[60] Lucas Martín. Gazebo, simulador de robótica, *Automatismos Mar del Plata*. http://www.automatismos-mdq.com.ar/blog/2017/01/gazebo-simulador-de-robotica.html, 2017.

[61] Gazebo Simulator: simular un robot nunca fue tan fácil, *Robologs*. https://robologs.net/2016/06/25/gazebo-simulator-simular-un-robot-nunca-fue-tan-facil/, 2016.

[62] Grupo SIRP. Tipos de Movimiento y Grados de Libertad. https://es.slideshare.net/EducaredColombia/tipos-de-movimiento-y-grados-de-libertad.

[63] Lego NXT Holonómico, *Lego NXT Mindstorms*. http://www.lejosconlego.com/2013/02/lego-nxt-holonomico.html.

[64] SDF. http://sdformat.org/, 2017.

[65] Simuladores. https://eva.fing.edu.uy/pluginfile.php/127423/mod_resource/content/1/simuladores.pdf.

[66] iRobot Corporation. *Manual Roomba 500. Robot de Limpieza Aspirador*, 2010.

[67] iRobot Corporation. *Manual Roomba 800*, 2014.

[68] Elena Rodríguez. *Sistema de Control de Aspiradora Automática.* PhD thesis, Universidad Pontificia Comillas. Escuela Técnica Superior de Ingeniería, 2008.

[69] Lucía Ruiz. Las Mejores Aspiradoras Robot del 2017 – Guía de Compra, *Aspiradoras de mano.* http://www.aspiradorasdemano.com/mejores-aspiradoras-robot/, 2017.

[70] Guido Zunino. *Simultaneous Localization and Mapping for Navigation in Realistic Environments* . PhD thesis, Royal Institute of Technology, 2002.

[71] Julia Layton. How Robotic Vacuums Work, *HowStuffWorks* . http://electronics.howstuffworks.com/gadgets/home/robotic-vacuum2.htm.

[72] Robot aspirador LG Hombot Turbo. Especial casas con mascotas, niños y alfombras, *LG.* http://www.lg.com/es/aspiradoras/lg-VR65710LVMP, 2017.

[73] Roomba navigation algorithm, *Random Sequence.* http://www.randseq.org/2012/11/roomba-navigation-algorithm.html, 2012.

[74] Página Oficial de OpenCV. http://opencv.org/, 2017.

[75] Jose Cabrera Lozano. Definición de robótica educativa*Edukative, Robótica educativa.* https://edukative.es/definicion-robotica-educativa/, 2014.

[76] Iniciativas que promueven las competencias 4C en los alumnos: creatividad, pensamiento crítico, colaboración y comunicación,*innedu.* http://www.innedu.es/iniciativas-que-promueven-las-competencias-4c-en-los-alumnos-creatividad-pensamie#.WbbxB8hJbIV.

[77] Importancia de los videojuegos para las habilidades del siglo XXI,*ÁrbolABC.com.* https://arbolabc.com/blog/importancia-de-los-videojuegos-para-las-habilidades-del-siglo-xxi/.

[78] Las Cuatro Habilidades De Una Educación Del Siglo 21,*Fundación SM*. http://www.seminariointernacional.com.mx/blog/Las-cuatro-habilidades-de-una-educacion-del-siglo-21, 2015.

[79] G. Ocaña, I. Romero, F. Gil, and A. Codina. Implantación de la nueva asignatura "Robótica" en Enseñanza Secundaria y Bachillerato. *Investigación en la Escuela*, 13(87):65–79, 2015.

[80] JdeRobot-kids,*JdeRobot*. http://jderobot.org/Robotica-en-secundaria, 2017.

[81] Tutoriales de robóticaTutoriales de robótica. https://www.solidworks.es/sw/education/9931_ESN_HTML.htm.

[82] Ricardo Tellez. A thousand robots for each student: using cloud robot simulations to teach robotics. *The Construct Sim*, 2016.

[83] Robótica en la Universidad de Alcalá. Proyecto TuBot 2015,*Alcabot*. http://asimov.depeca.uah.es/robotica/mod/resource/view.php?id=1188, 2015.

[84] Grado en Ingeniería Robótica,*Universidad de Alicante*. https://cvnet.cpd.ua.es/webcvnet/planestudio/planestudiond.aspx?plan=C211#, 2016.

[85] Plan de Estudios Electrónica, Robótica y Mecatrónica,*Universidad de Málaga*. http://www.uma.es/grado-en-ingenieria-electronica-robotica-y-mecatronica/info/9857/plan-de-estudios-electronica-robotica-y-mecatronica/, 2017.

[86] 471 Másters oficiales de ingeniería electrónica robotica mecatroóica,*educaweb*. http://www.educaweb.com/masters-oficiales-de/ingenieria-electronica-robotica-mecatronica/, 2017.

[87] Robotics middleware,*Wikipedia*. https://en.wikipedia.org/wiki/Robotics_middleware, 2017.

[88] Open Source Robotics Software/Hardware List,*Learn Robotix*. http://learnrobotix.com/open-source-robotics-software.html.

[89] V. M. Arévalo, J. González, and G. Ambrosio. *La Librería de Visión Artificial OpenCV, Aplicación a la Docencia e Investigación*. PhD thesis, Dpto. De Ingeniería de Sistemas y Automática, Universidad de Málaga, 2004.

[90] BoofCV. https://boofcv.org/index.php?title=Main_Page.