Actas de las Jenui, vol. 5. 2020.
Páginas: 285-292

R2P2: Un simulador robótico para la enseñanza de Inteligencia Artificial

Mario Cobos María D. R-Moreno David F. Barrero
Departamento de Automática, Universidad de Alcalá
mario.cobos@edu.uah.es malola.rmoreno@uah.es david.fernandezb@uah.es

Resumen

La abstracción y complejidad inherente a los diversos algoritmos existentes para el desarrollo de Inteligencia Artificial (IA), desde algoritmos deterministas clásicos hasta técnicas de Aprendizaje Automático, dificulta su comprensión por parte del alumnado. Nuestra propuesta es R2P2, un sencillo simulador robótico escrito en Python cuyo objetivo es facilitar el estudio de la IA. Para ello, presenta al alumno un problema de control robótico en un entorno de fácil comprensión y permite tanto practicar como visualizar distintas técnicas de IA de aplicación en el ámbito de la Robótica. R2P2 es un simulador abierto basado en Python fácil de programar y que aporta un entorno altamente motivador para el estudiante.

Abstract

Artificial Intelligence (AI) algorithms are inherently abstract and complex, independently of their nature. From classic deterministic techniques to Machine Learning algorithms, this complexity makes understanding them increasingly hard for the students. Thus, we introduce R2P2, a simple robotics simulator written entirely in Python, with the goal of simplifying the process of learning about AI. For that purpose, the student is given an easy to understand robotics control problem, allowing them to visualize and practice a variety of AI techniques used in Robotics. R2P2 is an open, easy to use, Python-based robotics simulator, which provides a highly motivating environment for the student.

Palabras clave

Simulador, Robótica, Inteligencia Artificial, Recurso docente, Python

1. Introducción

La impartición de asignaturas relacionadas con Inteligencia Artifical (IA) suele requerir la visualización de las técnicas explicadas para facilitar su comprensión.

Una posible aplicación de la IA es el control de sistemas robóticos de complejidad variable. La planificación de las tareas que un sistema ha de realizar, así como el desempeño de las mismas, se realiza comúnmente mediante técnicas de complejidad que se engloban en dicho campo. El carácter físico de esta aplicación de la IA la convierte en una herramienta apta para su enseñanza.

No obstante, el hardware robótico real presenta costes y riesgos adicionales que, habitualmente, son difícilmente aceptables en el contexto de un aula. Es por ello que, normalmente, el medio elegido para su uso son simuladores robóticos tales como ROS (*Robot Operating System*) [5] o V-REP [6], por permitir aplicar el contenido estudiado a un caso real.

Pese a la existencia de numerosos simuladores robóticos a disposición del profesorado, la mayoría de ellos han sido diseñados para su uso en un contexto de investigación o desarrollo, y habitualmente resultan excesivamente complejos para su uso en el aula en el contexto de la enseñanza de la IA. En este contexto se busca que el simulador no añada más complejidad que la imprescindible para evitar que el alumno no distraiga la atención del núcleo de la enseñanza. Por otra parte, la IA actual utiliza de manera mayoritaria Python como lenguaje de programación, por lo que sería deseable que el simulador estuviera en ese mismo lenguaje.

Por ello, presentamos R2P2, un simulador robótico sencillo y robusto, escrito exclusivamente en Python, que permite al alumno practicar diversas técnicas de IA, sin para ello requerir un largo proceso de aprendizaje referido a la herramienta.

En primer lugar, examinaremos brevemente los simuladores robóticos ya existentes, tratando de dilucidar sus características principales. A continuación, describiremos de manera general el simulador R2P2, para entonces detallar su modelo de programación y su aplicación a nivel docente. Finalmente, describiremos las conclusiones extraidas del desarrollo y uso de esta herramienta.

2. Simuladores robóticos

El mundo de la Robótica cuenta con numerosos simuladores y medios para facilitar el desarrollo y depuración de software de control robótico. Dichos medios, si bien potentes, resultan excesivamente complejos suponiendo una curva de aprendizaje elevada relacionada con la propia herramienta a utilizar, y restando protagonismo a los conceptos básicos que debería ser el centro de atención.

Las plataformas disponibles son vastamente numerosas, pero entre las más utilizadas en el aula se encuentran ejemplos notorios como ROS, V-REP, OpenAI GYM [1], Player/Stage/Gazebo[4], Modular open robots simulation engine (MORSE)[3] y Bullet[2].

Con el fin de establecer las diferencias de R2P2 con sus predecesores, se procede a continuación a resumir las características más fundamentales de los simuladores que, probablemente, más se usan con fines docente: ROS, Player/Stage y Bullet.

2.1. Robot Operating System

ROS fue desarrollado por Willow Garage y Stanford, y liberado al público en enero de 2009. La última versión del sistema, Lunar Loggerhead, fue liberada en mayo de 2017.

Diseñado originalmente con el objetivo de crear una plataforma robótica modular, dicha filosofía de diseño se ha mantenido a lo largo de los años. Así, ROS no ofrece un simulador robótico, sino, funcionalidades para el control de plataformas robóticas reales desde un ordenador.

Las marcadas diferencias entre versiones de esta tecnología acrecentan aún más la ya de por sí notable dificultad de su aprendizaje y dominio, llegando incluso a invalidar por completo secciones completas de su documentación entre versiones.

Así, si bien la aplicabilidad de ROS para labores de investigación robótica, así como la implementación de sistemas funcionales en el mundo real, es innegable, resulta difícil implementar su uso durante el desarrollo de una asignatura que no sea de Robótica, especialmente sin garantías de que gozará de continuidad como herramienta en asignaturas posteriores que el alumnado deberá cursar.

2.2. Player/Stage/Gazebo

Player es una plataforma robótica orientada a la investigación tanto robótica como sobre sensores. Para facilitar la depuración de los controladores generados se apoya en dos simuladores robóticos distintos: Stage y Gazebo.

Stage se centra en la simulación robótica bidimensional, y provee soporte para la simulación de físicas y sensores en entornos bidimensionales. No puede ser empleado por si mismo, y la precisión de la simulación es limitada, pero el relativamente bajo consumo computacional involucrado permite la simulación de numerosas plataformas robóticas en ejecución de manera simultánea.

Gazebo, por su parte, es el simulador tridimensional típicamente asociado a Player o ROS como plataformas robóticas mediante el uso de módulos externos que hagan de intermediarios. Ofrece funciones de simulación de alta fiabilidad para plataformas robóticas, empleando para ello simulación de físicas tridimensionales de alta calidad. No obstante, su alto consumo computacional implica la imposibilidad de simular numerosos robots al mismo tiempo.

Finalmente, y estableciendo un nuevo paralelismo con ROS, la herramienta Player/Stage/Gazebo se beneficiaría de la existencia de continuidad en su uso en el plan de estudios del alumno que la emplea, permitiendo explotar sus posibilidades en caso de existir dicha continuidad.

2.3. Bullet

Bullet es un motor de físicas 3D escrito en Python, aplicable preeminentemente a los campos de la Robótica y los videojuegos. Su enfoque principal es la generación de simulaciones físicas realistas, que permitan transferir el resultado obtenido en la simulación a la realidad de la forma más directa posible.

Bullet incluye soporte directo con múltiples ejemplos de robots ya generados y funcionales, de modo que se puedan aplicar de manera directa en el contexto

de la investigación. No obstante, generar un nuevo robot para su uso resulta un proceso largo y laborioso, debido a la cantidad de parámetros a configurar, haciendo inviable la adaptación del motor a nuevos casos de uso en el contexto de un aula de clase. Lo que es más, la necesidad de trabajar con los parámetros físicos del modelo en gran parte de los casos eleva la complejidad de aplicar algoritmos clásicos para planificación de tareas o planificación de caminos.

La documentación de las funcionalidades incluidas es extensa y clara, recibiendo actualizaciones regulares para garantizar su aplicabilidad, y su infraestructura lo convierte en un motor adecuado para la realización de pruebas robóticas en un entorno controlado.

No obstante, su potencia lo hace difícilmente aplicable en una asignatura. Del mismo modo que ROS y Player/Stage/Gazebo requieren una extensa comprensión de sus complejos modelos de programación, Bullet requiere profundos conocimientos de Python para el apropiado empleo de su motor de físicas.

Adolece, por tanto, de las mismas desventajas que los otros dos motores ya presentados. Su aplicabilidad en un contexto de desarrollo e investigación es innegable, pero resulta difícil transferir las mismas ideas a un contexto puramente educativo. Resulta deseable, por tanto, buscar una alternativa más sencilla, que permita al alumnado centrar sus esfuerzos en la materia a estudiar.

3. R2P2

Nuestro simulador[1] se basa en dos premisas fundamentales: simplicidad de uso y versatilidad; de este modo, el alumno puede dedicar la mayor parte del tiempo invertido en el uso del mismo en la comprensión e interiorización de los conceptos que se están trabajando en el aula.

La figura 1 muestra el simulador en su modo de funcionamiento más básico. En este modo, se pude controlar el robot utilizando el teclado o un joystick compatible con el sistema operativo local. Se puede observar que R2D2 simula un escenario bidimensional y robots circulares con sensores de distancia (sónar o láseres) situados a lo largo de su perímetro. Estas suposiciones permiten simplificar en gran medida tanto el simulador como su modelo de programación. En las siguientes secciones, se detallarán el lenguaje de programación empleado y las dependencias software del simulador, las funcionalidades definidas por defecto, las opciones de configuración y los métodos de extensión disponibles. Finalmente, se procederá a comparar el simulador R2P2 con ROS, Player/Stage y Bullet.

[1]Disponible como software libre con licencia GPL en `https://github.com/ISG-UAH/r2p2`.

3.1. Lenguaje de implementación y dependencias

Para conseguir el objetivo de crear un simulador robótico sencillo de comprender y utilizar, se ha escrito la totalidad del código fuente en el lenguaje Python. Todas las dependencias se corresponden con librerías estándar, fácilmente descargables desde el repositorio de la herramienta PIP.

El motor gráfico está construido sobre Pygame, apoyándose en PIL y Numpy para facilitar el tratamiento de imágenes para su utilización como escenario en la simulación. El nivel de los colores empleados en la creación de la imagen determinan la pasabilidad de cada sección de la misma, oscilando desde un obstáculo infranqueable con el negro a terreno fácilmente superable en la blanco. Cualquier tonalidad intermedia puede emplearse para establecer terreno que presenta un determinado grado de dificultad para ser recorrido, lo que se puede utilizar para algoritmos de planificación de caminos. Así, los escenarios pueden quedar definidos como una imagen en formato PNG.

3.2. Funcionalidades de R2P2

El simulador incluye múltiples controladores que pueden ser configurados mediante la modificación de los correspondientes archivos en formato JSON. Entre los controladores incluidos existen paradigmas clásicos en control de robots, como la teleoperación o los controladores PID, pero también se incluyen casos más sofisticados como controladores basados en planificación de caminos, ejecutores de planificaciones PDDL y soporte para la creación de controladores neuroevolutivos funcionales. Todos ellos se integran de manera transparente con el robot simulado, permitiendo utilizar el mismo controlador o robot en multitud de situaciones distintas.

Resulta conveniente realizar una revisión de los controladores incluidos por defecto en el simulador, así como funcionalidades accesorias que puedan ser dignas de mención, por tratarse de uno de los elementos definitorios del mismo.

Teleoperación

El modo más básico de funcionamiento del simulador es mediante la simulación de un robot teleoperado. Este modo es también la configuración por defecto de la aplicación, permitiendo al alumno acostumbrarse

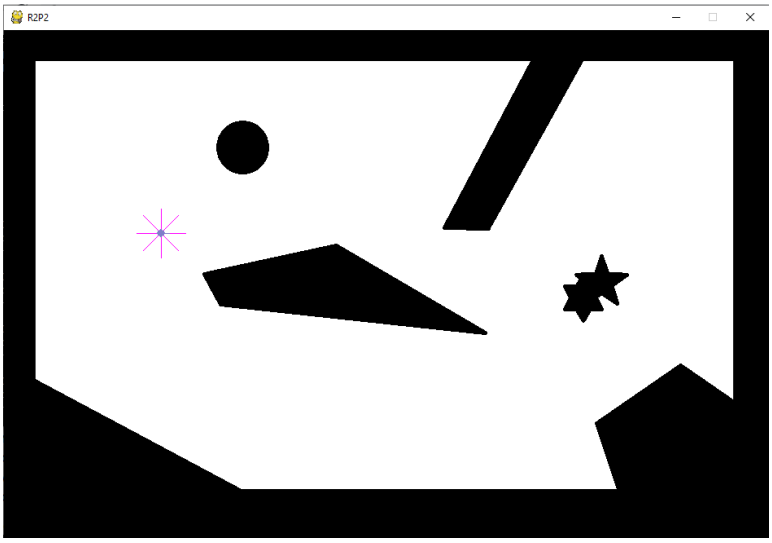

Figura 1: Captura de pantalla del simulador en funcionamiento. El punto azul representa el robot, y los haces magenta los sensores del mismo. Las paredes del escenario se representan en color negro, facilitando su identificación y su uso de manera programática.

a la interfaz gráfica antes de pasar a casos de uso más complejos. El alumno puede utilizar el teclado o un joystick para teleoperar el robot. Por supuesto, el uso de un joystick garantiza mayor precisión en la conducción del robot.

Adicionalmente, el robot definido por defecto incluye visualización de sus sensores, de modo que es posible para el alumno observar su comportamiento antes de proceder a trabajar con funciones más complejas.

PID

También incluido con fines demostrativos es un controlador PID con un camino definido manualmente.

Este controlador regula el funcionamiento interno de los controladores relacionados con planificación, tanto de caminos como de tareas, de modo que poder estudiar su comportamiento en un entorno aislado proporciona al alumno una oportunidad de familiarizarse con una herramienta básica en el desarrollo de las prácticas que es previsible que desarrolle.

Controlador neuroevolutivo

De manera nativa, R2P2 ofrece soporte para la implementación de controladores neuroevolutivos. Este soporte se basa en el uso de la librería Scikit-Learn, e incluye mecanismos para actualizar de manera dinámica el controlador que se está utilizando.

Posee dos modos de operación: entrenamiento y ejecución. El modo de entrenamiento requiere la implementación externa de un algoritmo evolutivo para entrenar el controlador, mientras que el modo de ejecución requiere la correcta parametrización de la red neuronal en el archivo de configuración del controlador.

De este modo, es posible estudiar el uso de algoritmos genéticos sin necesidad de prestar atención a la infraestructura intermedia necesaria para su aplicación al campo de la Robótica, lo que permite al alumno focalizar su atención en el problema central.

Planificación de caminos

R2P2 viene equipado con controladores compatibles con algoritmos de planificación de caminos. Estos controladores se basan en la aplicación de un algoritmo de planificación de caminos a un controlador PID previamente parametrizado.

Además, como ejemplos, R2P2 incluye implementaciones del algoritmo de Dijkstra y A*, a efectos de demostración. Las heurísticas usadas por estos algoritmos, no obstante, han de ser implementadas por el alumno. El controlador funcionando con el algoritmo A* se presenta en la Figura 2. El punto azul representa el robot, los haces magenta provenientes del mismo representan los sensores del mismo. Las paredes aparecen representadas en negro, y el camino planificado se muestra en rojo. Las regiones en distintas tonalidades de gris presentan dificultades de acceso acordes al tono que se ha utilizado para representarlas.

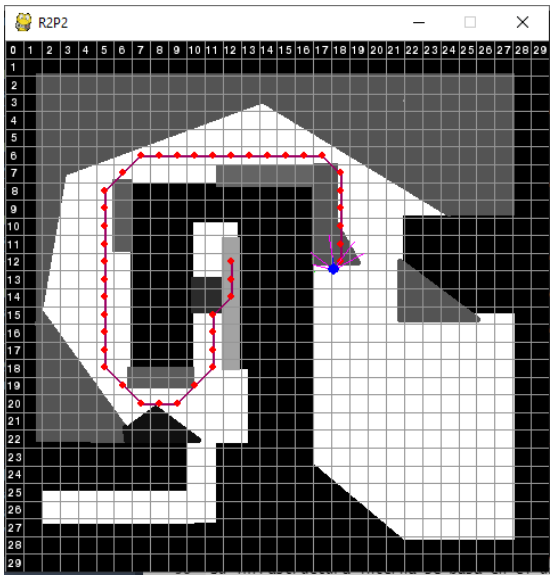

Figura 2: Captura de pantalla del simulador en modo de planificación de caminos. El simulador incluye una cuadrícula para facilitar la visualización del algoritmo empleado.

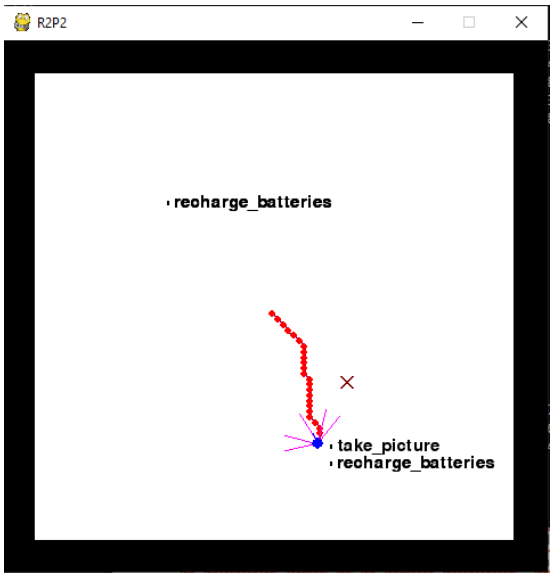

Figura 3: Captura de pantalla del simulador en modo de ejecución de PDDL.

Por supuesto, la filosofía general referente a la extensibilidad prevalente en la implementación del simulador está presente en las funcionalidades de planificación de caminos. Añadir un nuevo algoritmo o heurística resulta trivialmente sencillo, permitiendo dedicar la totalidad de los esfuerzos a escribir el código del algoritmo propiamente dicho.

Se trata, por tanto, de una funcionalidad versátil, que facilita la comprensión de los conceptos generales de planifiación de caminos por parte del alumno.

Planificación de tareas

La última funcionalidad básica incluida como parte de R2P2 es un ejecutor de planes de tareas PDDL. Reflejando la filosofía general de diseño adoptada de cara a la creación del simulador, este módulo se basa en el uso del módulo de planificación de caminos, y extiende aún más sus posibilidades.

El ejecutor de PDDL puede llevar a cabo multitud de tareas genéricas, y monitoriza los recursos disponibles en el robot durante el desarrollo de un plan. Además, cuenta con funcionalidades para simular tareas genéricas, lo que se traduce en mayor flexibilidad sin necesidad de que el alumno escriba código adicional. Un ejemplo del simulador ejecutando este modo se presenta en la figura 3. El punto azul representa el robot, y los haces magenta provenientes de él representan sus sensores. La línea roja representa un camino planificado de acuerdo a las necesidades de desplazamiento. Las etiquetas textuales representan acciones realizadas por el robot en un punto determinado y el aspa granate representa una fuente de CO_2.

Así, el foco es la creación del dominio PDDL, así como la correcta definición del problema, para poder generar un archivo con la planificación que deberá ejecutarse, y permite al estudiante visualizar un versión simplificada del desempeño de la misma, facilitando la interiorización del flujo de trabajo en el tipo de problema correspondiente.

3.3. Configuración

Es posible configurar los distintos parámetros que regulan una simulación en R2P2 mediante el uso de archivos JSON. Se distinguen tres tipos de JSON, que interactúan en conjunto para construir la simulación: Archivos de escenario, de robot y de controlador. A continuación describimos su formato.

- **Archivos de definición de escenario**. Estos archivos JSON codifican las rutas a los archivos que contienen los datos que definen la simulación, así como la configuración referente al uso de una interfaz gráfica de usuario (GUI). Así, estarán compuestos por:
  - Una ruta a una imagen, bajo el nombre de parámetro `stage`. Esta ruta puede definirse como una ruta relativa o absoluta dentro del sistema, y servirá para poder cargar la imagen que se usará para generar el entorno del

robot en la simulación.

- Una ruta, o colección de rutas, a archivos JSON en los que se configuren robot, bajo el nombre de `robot`.
- Una ruta, o colección de rutas, a archivos JSON en los que se configuren controladores, bajo el nombre de `controller`. Nótese que, en caso de establecerse menos controladores que robots, estos se asignarán de manera secuencial según su orden de aparición en la lista, y que la lista se repetirá tantas veces como sea necesario para lograr asignar un controlador a todos los robots. Asimismo, en caso de haber más controladores que robots, aquellos controladores que no puedan ser asignados a un robot serán ignorados.
- Un parámetro `gui`, cuyo valor debe estar asignado a `True` o `False`, y que indica si se debe o no usar una GUI en la simulación.

- **Archivos de definición de robot**. Este archivo contiene todos los atributos necesarios para generar un robot en las simulación. Los parámetros aceptados son los siguientes:
  - Un parámetro `id` que se corresponde con el identificador del robot.
  - Un parámetro `x` y un parámetro `y`, que en conjunto definen la posición inicial del robot.
  - Un parámetro `orientation` que define la orientación inicial del robot en grados, sobre un sistema de coordenadas cartesiano.
  - Un parámetro `sonar_range`, compuesto por dos elementos, que establece las distancias mínima y máxima para que un obstáculo sea detectado por los sónares del robot.
  - Un parámetro `radius` que establece el radio del robot.
  - Un parámetro `sonars`, que acepta un número entero o una lista de valores numéricos expresados en grados, y establece los sónares con los que cuenta el robot. En caso de emplearse un número entero, los sónares se distribuyen equitativamente por todo el perímetro del robot, mientras que en caso de utilizarse una lista, los valores incluidos establecen, en grados, la orientación de los sónares respecto al eje central del robot.
  - Un parámetro `max_speed` que establece la velocidad máxima de desplazmiento para el robot, permitiendo simular los límites impuestos por un sistema físico sobre la velocidad del conjunto.
  - Un parámetro `color`, que especifica el color con el que representar al robot.
  - Numerosos parámetros opcionales, asociados a la ejecución de planes definidos en PDDL.
- **Archivos de definición de controlador**. Dada la variedad de necesidades de configuración de un controlador en función de su funcionamiento interno, resulta imposible describir una pauta general que abarque todas las opciones. No obstante, todo archivo de configuración de controlador debe incluir un atributo `class`, que especifique la clase a la que pertenece el controlador en cuestión.

3.4. Extensión

Mediante el uso de herramientas estándar en el lenguaje Python, y un modelo de programación orientado a ser lo más intuitivo posible, el simulador permite generar nuevos simuladores de formar rápida y clara. Basta con definir el controlador en un archivo externo, escribir una función de factoría que configure el controlador en base al contenido de un archivo de configuración JSON, y registrar la función para su uso en la factoría abstracta que centraliza la instanciación de los controladores.

Además, al emplear librerías habituales en Python, y poder definir nuevos módulos externamente de forma sencilla, es posible utilizar librerías no contempladas en las depencias del simulador para la generación de nuevos elementos. Por ejemplo, resultaría posible utilizar Keras para generar un nuevo modelo de control neuronal en caso de ser necesario, y la práctica totalidad del tiempo de desarrollo del nuevo módulo estaría dedicada a la implementación del propio controlador, y no a la comprensión del funcionamiento interno del programa en su conjunto.

Esta idea tiene un alcance incluso mayor al inicialmente aparente, al permitir al usuario definir sus propias librerías de código y utilizarlas directamente sobre el simulador para proporcionar funcionalidades específicas a sus propios controladores, lo que se traduce en un elevadísimo grado de flexibilidad a nivel de código.

3.5. Comparación con otros simuladores

El cuadro 1 presenta una comparativa entre las características de R2P2 y alguno de los simuladores robóticos mencionados anteriormente. En general, se puede afirmar que la mayoría de las plataformas robóticas están diseñadas para poder ser utilizadas en un

robot físico, además de en simulación, por lo que se introduce un grado de complejidad alto.

4. Modelo de programación

Como se ha mencionado en el apartado anterior, el modelo de programación del simulador se centra en equilibrar simplicidad y flexibilidad. Esta idea informa el diseño de la API del simulador, y establece los requisitos fundamentales que rigen su implementación.

4.1. Diseño de la API

La mayor parte de utilidades que el simulador emplea para el funcionamiento de su motor permanecen ocultas, con el objetivo de que solo aquellas que el usuario final puedan necesitar queden expuestas al público. De este modo, no es necesario tomar en consideración la mayor parte de decisiones de diseño interno del simulador, y es posible concentrar toda la atención en elaborar nuevo código funcional.

Concretamente, las funcionalidades expuestas de forma directa al usuario son aquellas relacionadas con la estructura básica de un controlador. Su corazón es el método `control`, que recibe como entrada los datos de distancia procedentes de los sensores y espera como salida las velocidades lineal y angular que el robot debe tomar a continuación.

El siguiente ejemplo de código muestra cómo implementar un controlador trivial, que muestra por pantalla las lecturas recibidas de los sensores y mantiene una velocidad lineal constante de tres unidades por segundo:

```
def control(self, dst):
        print(dst)
        return 3, 0
```

Para facilitar la toma de esta decisión, la clase padre `Controller` define un parámetro interno, establecido por el robot al que se asigna el controlador, que regula la orientación de cada uno de los sensores, por lo que es posible conocer en todo momento la distancia y dirección en que se está detectando un obstáculo potencial.

Complementando a la información referente a la orientación de los sensores, es posible acceder a la odometría del robot mediante un acceso interno a los atributos `self.robot.x`, `self.robot.y` y `self.robot.orientation`, que establecen la posición actual del robot en formato `(x, y)` dentro del mapeado, así como la dirección en que se encuentra orientado el mismo en ángulos.

4.2. Salidas del simulador

Además, todos los controladores almacenan, por defecto, las lecturas registradas por los sensores, lo que permite representar la percepción del controlador de forma gráfica, y puede emplearse, también, para informar determinadas decisiones en caso de que el controlador lo requiera. Esta opción pueder ser fácilmente desactivada en nuevos controladores, implementando el método `has_cur_detected_edge_list` y haciendo que devuelva `False` al ser llamado.

Además, el simulador produce un archivo log para cada robot en ejecución, en el que registra información relevante al mismo, así como información específica referente al controlador asignado para su gestión en caso de ser necesario. Esta información permite depurar el trabajo desarrollado por el alumno de manera sencilla y directa, al poder registrar cualquier información relevante en un formato claro y legible.

Además, el paquete de ejecución de PDDL incluye múltiples funcionalidades adicionales para simular la ejecución de diversas tareas planificadas previamente. En primer lugar, este controlador genera un log adicional en el que almacena información específica sobre la ejecución del plan, incluyendo las tareas realizadas, ordenadas en el tiempo, así como parámetros propios del robot que las llevó a cabo e información específica referente a cada tarea específica. Así, si el robot llevó a cabo una lectura de CO_2 en un punto específico del mapeado, el valor registrado se encontrará como parte del registro de dicha actividad en el log creado por el ejecutor.

Adicionalmente, el propio ejecutor de PDDL es capaz, también, de tomar capturas de pantalla del estado del simulador, para simular la posibilidad de un robot tomando una fotografía durante su exploración. Esta imagen se almacena junto al resto de logs, utilizando como nombre la marca de tiempo del momento en que se realizó la captura de la misma, en formato PNG.

5. Aplicación docente

El simulador R2P2 ha sido diseñado para su uso en ejercicios prácticos de asignaturas relacionadas con la Inteligencia Artificial, especialmente desde la óptica de la robótica. Todas las funcionalidades incluidas por defecto se implementaron con el objetivo de ser utilizadas en el aula, fundamentalmente con fines demostrativos, y permitir al alumnado profundizar en los conceptos fundamentales de las asignaturas estudiadas sin necesidad de de dominar una herramienta compleja.

	R2P2	ROS	Player/Stage	Bullet	Rock
Lenguaje	Python	C++/Python/Matlab	C++	Python	C++/Ruby
Configuración	Archivos JSON	Archivos XML	Archivos CFG	Funciones Python	C++ y GUI
Salidas	Imagen, logs	Logs	Imagen, log	Imagen, log	Imagen, log

Cuadro 1: Comparativa entre los simuladores robóticos existentes y R2P2.

Puede ser utilizado de manera directa para estudiar conceptos relacionados con algoritmos de planificación de caminos, búsqueda, planificación de tareas, redes neuronales y algoritmos genéticos sin necesidad de realizar modificaciones al núcleo del motor, o de que el alumno posea grandes conocimientos previos de programación.

No obstante, su aplicabilidad en aquellos contextos en que el alumno tenga conocimientos de programación es incluso mayor. Algunas de las aplicaciones propuestas incluyen:

- Estudio de técnicas más avanzadas de neuroevolución, como NEAT[7].
- Estudio de técnicas de aprendizaje por refuerzo.
- Generación de datasets para la implementación de neurocontroladores clásicos.
- Estudio de controladores borrosos.
- Estudio de políticas de control concretas aplicadas al contexto de la Robótica.
- Estudio de técnicas de neuroevolución relacionadas con modelos adversarios.

Por supuesto, la gran extensibilidad del simulador permite su fácil aplicación a multitud de casos que no se han contemplado en esta lista.

6. Trabajo futuro

El simulador R2P2 se encuentra en la actualidad en mantenimiento, y se está implementando en diversas asignaturas, con resutados, por ahora, positivos.

7. Conclusiones

El simulador realizado permite la aplicación directa y limpia de técnicas de Inteligencia Artificial tanto clásicas como avanzadas, lo que lo hace, a priori, apto para su empleo en el aula.

La constatación de dicha eficacia, prestando especial atención al proceso de adaptación del alumnado, permitirá informar los pasos a tomar en la ampliación y mantenimiento de la herramienta en el futuro, resultando en su refinado y, por consiguiente, la mejora de los resultados obtenidos por la misma.

Referencias

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[2] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

[3] Gilberto Echeverria, Nicolas Lassabe, Arnaud Degroote, and Séverin Lemaignan. Modular open robots simulation engine: Morse. In *2011 IEEE International Conference on Robotics and Automation*, pages 46–51. IEEE, 2011.

[4] Brian Gerkey, Richard T Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323, 2003.

[5] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[6] Eric Rohmer, Surya PN Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE, 2013.

[7] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.