Simulador 3D de robótica distribuido en Unity para enseñanza de programación e inteligencia artificial

» **Néstor Balich, Tupac Ocampo, Franco Balich y Berenice Balich**
nestor.balich@uai.edu.ar y tupac.ocampo@alumnos.uai.edu.ar y francoadrian.balich@uai.edu.ar
BereniceLourdes.Balich@alumnos.uai.edu.ar
[1] Laboratorio de Robótica Física – Centro de Altos Estudios en Tecnología informática – Universidad Abierta Interamericana – CABA – Argentina – Montes de OCA 745

Abstract

El uso en los últimos años de nuevas e innovadoras herramientas tecnologías ha mejorado el modo en que se dictan las clases universitarias. El contexto de las restricciones en pandemia ocasiono que estos cambios se aceleran. Considerando que la materia paradigmas tecnológicos II (Robótica) integra y enseña en su cursada nuevas tecnologías y en sus talleres se realizan prácticas con robots reales, dispositivos IoT y de realidad aumentada, y sumado al dictado virtual de las clases. Este trabajo expone la investigación y creación de un simulador 3D de robótica con programación distribuida, con un modelo físico real, que permite el trabajo colaborativo, la programación en Python y la inclusión de inteligencia artificial con robot simulados y reales en un laboratorio remoto que cumpla con las necesidades de la materia y las cuales no podían ser resultas con una sola aplicación existente.

KEYWORDS: SIMULADORES ROBÓTICOS, UNITY, PROGRAMACIÓN DISTRIBUIDA, IOT, INTELIGENCIA ARTIFICIAL, PYTHON, EDUCACIÓN, VIDEOJUEGOS, METAVERSO.

1 Introducción

El contexto en pandemia, durante el 2020 y 2021, nos llevó a buscar formas innovadoras para el dictado de clases de grado en la carrera de ingeniería en sistemas de la Universidad Abierta Interamericana (UAI). Para ello era necesario modificar y buscar herramientas equivalentes para las prácticas y talleres en los cuales el uso de los robots físicos como herramienta educativa eran fundamental con la ventaja de los entonos lúdicos de aprendizaje Infante [1] y el aprovechamiento de entornos visuales para enseñanza de programación 2 expuesto por Sáez [2].

Este trabajo es el resultado de una investigación sobre las diferentes opciones de simuladores libres evaluados para la materia Paradigmas Tecnológicos II (Robótica), simulación de circuitos eléctricos, diseño 3D, circuitos electrónicos y programación para Arduino y en Python.

Recibido: Marzo 2023 / Aceptado: Mayo 2023 / Publicado: Junio 2023

Durante el proceso de búsqueda de un simulador que cumpla con los requisitos propuestos, una de las líneas de investigación fue evaluar la viabilidad del desarrollo de un simulador propio.

Entre los motores de video juegos evaluados se encontraron Unity, Unreal Engine, y OpenGl, al igual que sus requisitos funcionales y no funcionales, tales como la velocidad y complejidad de desarrollo, la curva de aprendizaje, facilidad de implementación de interfaces externas, y existencia de herramientas complementarias de código libre.

Este diseño debía contemplar el funcionamiento en un entorno de escritorio o en formato web, para que los alumnos de la universidad pudieran acceder en forma remota, articular con programas de control descentralizados en Python (y a futuro otros lenguajes que respeten el protocolo de comunicación), el desarrollo de un entorno IDE WEB de programación en Python para el aprendizaje y programación de algoritmos aplicados a robótica de manera distribuida desde el hogar de los alumnos 2. Sáez [2], otro requisito fundamentes era el de vincularse con una plataforma de inteligencia artificial que utilizamos para algoritmia avanzada de navegación y controla de robots.

Debía también permitir configurar de forma rápida diferentes escenarios con motor de simulación física con entornos 3D realistas similar a los juegos de video.

El desarrollo final fue llevado a cabo utilizando la plataforma de Unity, puesto que es un motor gráfico liviano y gratuito para fines no comerciales que utiliza el lenguaje de programación C#, permitiendo aplicar lo aprendido en materias de la universidad, y la gran cantidad de herramientas de desarrollo en forma de paquetes descargables. Los protocolos de comunicación explorados y desarrollados fueron: 1) Modelo de comunicación asincrónica con WebSocket. 2) Modelo con comunicación por protocolo MQTT utilizando el broker de código abierto Mosquitto.

También se desarrollaron programas cliente en Python para un robot que esquiva obstáculos de manera reactiva Atenea.py, un cliente de gestión y visualización de paquetes de información que llamamos Hermes.py, un robot simulado con 5 Lidar (Sensores láseres de distancia) que llamamos oRobby, y un escenario 3D simulado que llamamos campus de prueba UAI con objetos varios y un laberinto que el robot debe recorrer controlado en forma autónoma por un programa Python desarrollado por los alumnos.

2 Objetivo del proyecto

El objetivo original era encontrar un simulador de robots de software libre fácil de utilizar que sean web y se pueda programar y configurar en Python, tanto a nivel escenarios como de robots. Luego de la evaluación de ellos y dado que ninguno cumplía con todos requisitos, nos propusimos crear nuestro propio simulador, con lineamientos ya expresados agregando la posibilidad de programación de robots tanto virtuales como reales.

3 Tareas realizadas

3.1 Evaluación y prueba de simuladores de robótica e IA

Se evaluaron varios simuladores que utilizamos en 2020 en base a las mejores opciones recomendadas en varios artículos como Silicon Valey[3], y 4. Pitonakova [4] o de uso profesional como los citados por Michel [5], y pensando en la realización de nuestro entorno de trabajo para inteligencia artificial basado en otras experiencias con el uso framework para IA Casañ [6]

	Simulación electrónica	Simulación Arduino	Simulación robots	Lenguajes programación	Web	Complejidad	Observaciones
Tinkercad	SI	SI	-	C	SI	Básico y medio	
Gazebo	-	-	SI	C, Python, JavaScript	-	Avanzado	
Webots	-	-	SI	Python	-	Avanzado	
VR.VEX	-	-	SI	Python, Bloques	SI	Básico y medio	Escenarios acotados
Copelia Sim	-	-	SI	C y Python	-	Medio	

Fig. 1. Tabla comparativa de simuladores

3.2 Webots: Robot con Python y redes neuronales

Retomando un proyecto realizado en el año 2020 con Webot basado en el trabajo de Kirtas [7] en donde creamos un robot que utilizaba una red neuronal para desplazarse en un ambiente simulado evitando los obstáculos, si bien fue buena la experiencia de realizar inteligencia artificial con este simulador, notamos que necesitábamos dos clases para enseñar a usar el simulador. Comunicándonos con el exterior mediante un programa basado en MqTT a la escucha que se vincula con otro programa remoto con interfaz web desarrollado en JavaScript sobre node.js para el control y programación remota.

3.3 Implementación de un simulador de robots en Unreal y Python

En cuanto a la evaluación de motores de videojuegos comenzamos con Unreal dada su alta calidad de imagen. Realizamos varios programas y testeos descartándolo rápidamente, debido a que la programación en Python se realiza por medio de un plugin en su versión beta, que está diseñado para ser programado en C++ o por Blueprints (programación en bloques), posee menor documentación y que los ejemplos de programación en Python que en su mayoría corresponden a scripts de organización de materiales.

3.4 Desarrollos propios para simulación de robots y pruebas

Motor	Lenguaje	Interfaz control remoto	Permite Python	Observaciones
Unity	C#	SI	No lo permite	Mucha documentación, curva de aprendizaje rápida.
Unreal	C++	SI	Con plugin, solo para cosas auxiliares.	Menor documentación, mayor curva de aprendizaje

Fig. 2. *Comparación de lenguaje de programación para los simuladores*

Se optó por dos líneas desarrollo: 1) que permita la conexión externa desde un cliente Python con el objetivo que generar una interfaz web y propiciar la escalabilidad para desarrollar algoritmos

Revista Abierta de Informática Aplicada/vol 7 Nº1 (2023): 24-36
ISSN 2591-5320

con inteligencia artificial conectado a Google Colab y 2) desarrollar el programa del simulador en C para disminuir la curva de aprendizaje.

4 Desarrollo de entorno de simulación en Unity y cliente Python

4.1 Desarrollo del simulador en Unity

Unity hace uso de C# como lenguaje nativo con el cual los alumnos se encontraban familiarizados ya que es utilizado en las materias de programación 1, Programación Orientada a Objetos, y Lenguajes de Última Generación dictadas en la universidad.

5 Implementación

Los diferentes elementos por operar en Unity son tratados como objetos, a los cuales se les puede aportar diferentes componentes. Cada uno de estos, a su vez, posee diferentes propiedades. De tal forma, un objeto X tendrá un componente que responda al motor de físicas, otro para detectar las colisiones y otro para las coordenadas en el espacio y su desplazamiento, entre otros. Por ejemplo, el objeto "Script" será aquel que posea el código de C# propio a ejecutar.

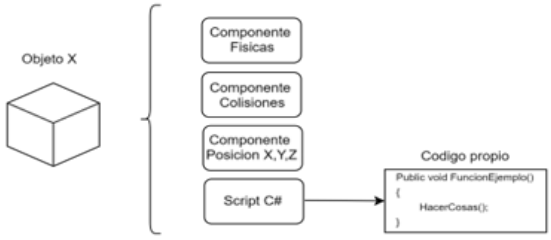

Fig. 3. *Representación de la relación entre objetos, componentes y código propio*

Al mismo tiempo, un objeto podrá referenciar otro objeto o componente diferente desde el script, permitiendo así vincularlos y conseguir un funcionamiento coordinado.

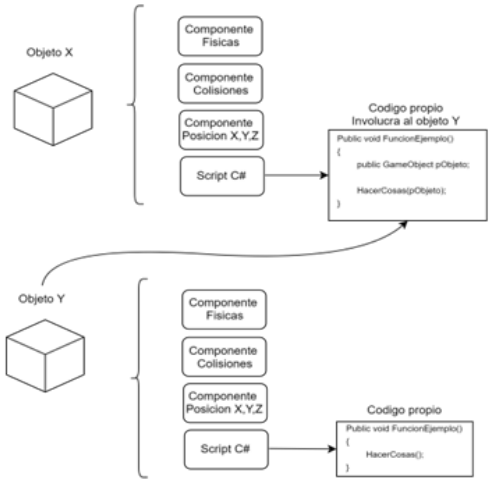

Fig. 4. *Forma en que diferentes objetos de relacionan entre sí.*

De esta forma el objeto X ejecutará funciones dentro del script, el cual referencia al objeto Y. Ellos también presentan relaciones de agregación jerárquica con contención física, en donde un conjunto de objetos formará parte de un objeto padre. Los comportamientos de este último serán reflejados en los objetos hijos.

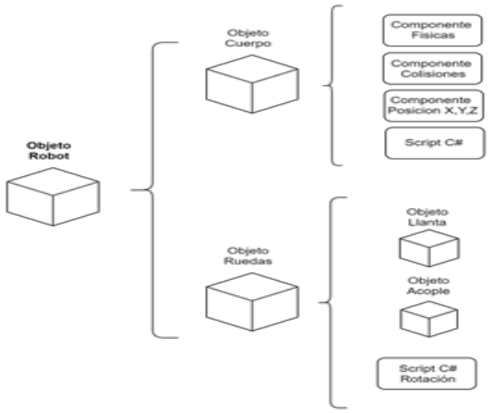

Fig. 5. *Relación de jerarquía entre objetos y componentes.*

Llevando este desarrollo a la realidad nos encontramos con un robot formado por diferentes componentes, los cuales, a su vez, mediante scripts en C#, implementan funcionalidades específicas a la vez que referencian otros objetos, con un consiguiente funcionamiento sincronizado. Aplicando todas estas reglas será posible el desarrollo de objetos que posean diferentes componentes y funcionalidades.

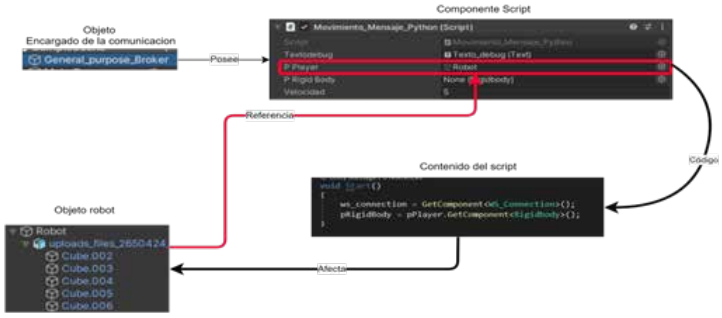

Fig. 6. *Relación entre objetos y código dentro de la interfaz de Unity.*

Dentro del entorno de Unity tendremos múltiples componentes representando los diferentes objetos, tales elementos de la interfaz de usuario, plano, paredes, obstáculos, y el mismo robot. A su vez, cada elemento poseerá diferentes componentes, los cuales aportarán funcionalidades y lógica extra.

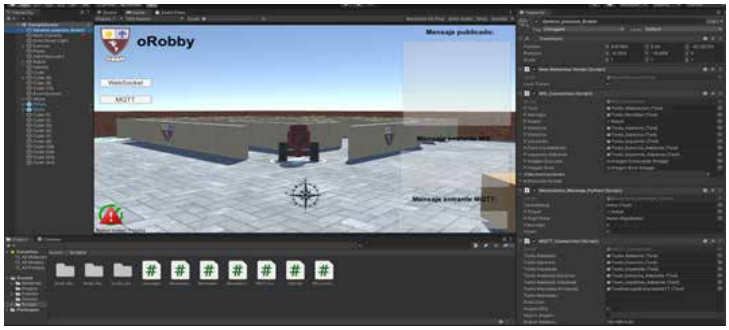

Fig. 7. *Vista de la interfaz en Unity.*

Revista Abierta de Informática Aplicada/vol 7 Nº1 (2023): 24-36
ISSN 2591-5320

De izquierda a derecha podremos ver la lista de objetos existentes, una vista del mundo generado, y finalmente la lista de componentes y objetos referenciados. En el margen inferior, la lista de scripts creados, separados entre aquellos de carácter general, y aquellos con funciones en común agrupados en carpetas.

Luego del estudio del entorno, herramientas, y la realización de varios modelos de prueba, nos encontramos con el resultado de esta investigación, oRobby, el robot simulador desarrollado por los integrantes del laboratorio.

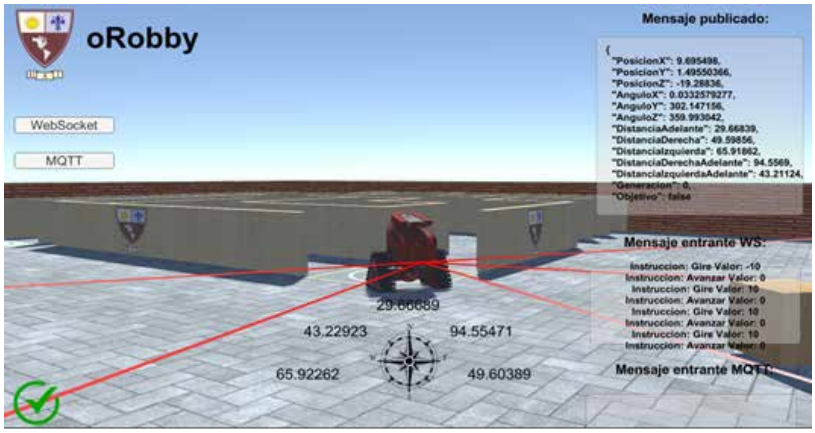

Fig. 8. *oRobby, robot desarrollado en el laboratorio, funcionando.*

5.1 Funcionalidades buscadas:

Al desarrollar nuestro propio simulador nos encontramos con algunos problemas desde el inicio: Por un lado, al guiar nuestro propio desarrollo, la dificultad de selección, definición y priorización de aquellas funciones necesarias para el laboratorio y los alumnos y, por otro lado, la dificultad de elegir entre diferentes tecnologías de acuerdo con sus ventajas y desventajas.

El objetivo principal en el desarrollo desde el principio fue la implementación de una herramienta que permita escribir código Python y que el robot lo pueda ejecutar, debido a que este es un lenguaje ampliamente adoptado y utilizado en el área de la robótica y la inteligencia artificial, con gran cantidad de librerías especializadas que permiten gran complejidad funcional a la vez que simpleza en la implementación. En consecuencia, optamos por un desarrollo iterativo e incremental, con punto de partida en el manejo de un robot básico, desarrollado en Unity

Ya que Unity no es compatible con Python, resultó necesario el desarrollo de un sistema de comunicación desde Unity con el exterior y viceversa.

Al comenzar con esta parte del desarrollo nos encontramos con una escasa cantidad de herramientas para facilitar tal función, como un solo intérpretes de Python para el entorno de Unity, "Python for Unity" Unity [8] actualmente carente de mantenimiento, e incompatible con versiones actuales de Python.

Centrados en sistemas abiertos de comunicación que permitan la compatibilidad con múltiples herramientas y lenguajes de programación para enviar al robot las instrucciones a ejecutar.

Para realizar algoritmos activos y proactivos oRobby también necesitaba conocer datos acerca de su posición con relación al entorno que lo rodea, colisión con otros objetos y otros tipos de sensores virtualizados.

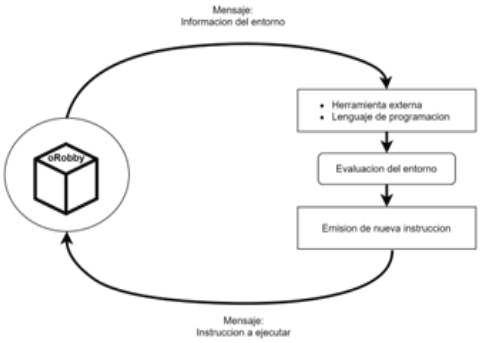

Fig. 9. *Envío y recepción de nuevos mensajes por parte de oRobby.*

Funcionalidades implementadas en oRobby:

- Movimiento utilizando izquierda, derecha, adelante y atrás.
- Implementación WebSocket para comunicación con Python
  - Movimiento mediante comunicación WebSocket.
  - Publicación de posición, ángulo, distancia al objeto más próximo en 5 direcciones, generación y llegada al objetivo en paquete JSON.
- Implementación de protocolo de comunicación MQTT.
  - Movimiento mediante suscripción a Topics.
  - Publicación de posición, ángulo, distancia al objeto más próximo en 5 direcciones, generación y llegada al objetivo.
- Sistema de reset y cuenta ante colisiones.
- Laberinto dentro del entorno con un objetivo a alcanzar.
- Lectura grafica de la distancia al objetivo más cercano en 5 direcciones y mensajes publicados y recibidos por WS y MQTT

Actualmente existen dos versiones de oRobby: Una primera versión ejecutable local, y una versión web, actualmente en proceso de implementación, ofreciendo funcionalidades extras con interfaces de conexión con Raspberry Pi y Arduino.

Fig. 10. *oRobby en su versión WebGl, para web.*

Revista Abierta de Informática Aplicada/vol 7 Nº1 (2023): 24-36
ISSN 2591-5320

Fig. 11. *Plataforma oRobby Web actualmente en desarrollo, con Python Flask.*

5.2 Protocolos de comunicación utilizados

Durante la etapa de investigación descubrimos que los protocolos explorados nos brindaron diferentes experiencias entre las cuales podemos diferenciar:

WebSocket.

El desarrollo de la comunicación utilizando WebSocket resultó simple gracias al uso de diferentes librerías y plugins para C# y Unity tales como <WebSocket-sharp>.

La dificultad de este método se encontró en la administración de los mensajes de forma asíncrona, y el tratamiento dado a los mensajes para el movimiento de oRobby.

Metodología aplicada para la comunicación con Python

Para la implementación con Python se optó por un sistema automático de navegación, capaz de evaluar la posición de oRobby con su entorno e indicarle el siguiente movimiento emitiendo un mensaje informando su estado, compuesto por una lectura de distancia de cinco lasers ubicados al frente y a los costados, posiciones X, Y, Z, el ángulo en relación con el norte del mapa, detección de contacto con el objetivo ubicado a la salida del laberinto.

El cliente comunicado evalúa esta información, y formular una nueva instrucción, la cual será devuelta a oRobby de acuerdo con la estrategia de navegación.

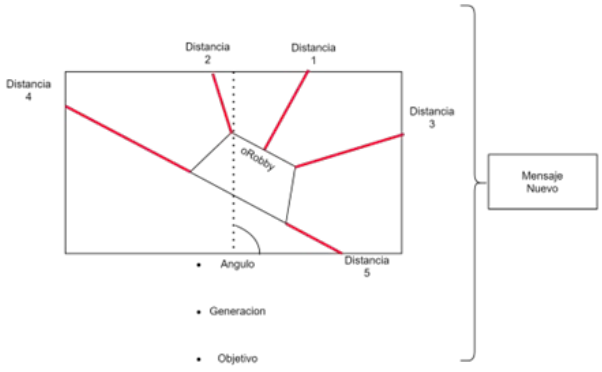

Fig. 12. *Información que compone los mensajes emitidos por oRobby y sensor lidar.*

El cliente evaluará esta información para la emisión del siguiente mensaje.

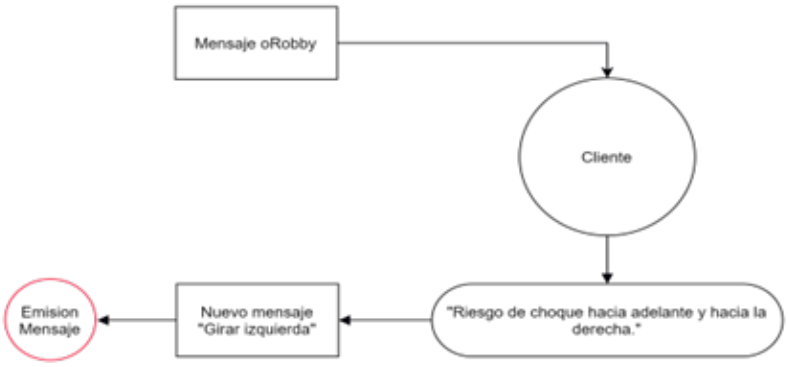

Fig. 13. *Método de emisión de nuevos mensajes por parte del cliente.*

5.3 Velocidad de comunicación

Para poder tratar el procesamiento de los mensajes, se implementó un envío constante de mensajes desde oRobby, y una lectura de mensajes entrantes cada un segundo. De esta forma se consigue tener información actualizada en tiempo real de la posición de oRobby en relación con el mundo que lo rodea, y la implementación de una cola de prioridades de los mensajes recibidos desde el cliente Python.

En lo que a eficiencia respecta, WebSocket presenta ciertos inconvenientes en instancias en las que los mensajes dejaron de ser emitidos y recibidos en intervalos de alto procesamiento por parte de Unity.

5.4 Complejidad de la implementación

El proceso por el cual se obtiene la comunicación entre oRobby y el cliente resultó ser compleja. Por un lado, resulta necesario iniciar el cliente desde el lado de Unity, y conectar un cliente desde Python. Por otro lado, Unity deberá poseer código encargado de recibir, interpretar, almacenar y desplazar al robot de la forma indicada, y del lado de Python será necesario código para recibir, interpretar, formular la instrucción indicada, y enviarlo a oRobby.

Este proceso poco intuitivo para el usuario promedio requiere de gran cantidad de acciones previas a cada ejecución, y conocimientos de programación para poder modificar el comportamiento de oRobby.

Por otro lado, al agregar mensajes con nuevas funcionalidades será necesario escribir código nuevo en ambos extremos de la comunicación, tanto en Unity como en Python.

5.5 Escalabilidad del servicio

Durante el desarrollo nos vimos en una situación particular, la cual fue que para poder realizar pruebas de cualquiera de los extremos de la comunicación era necesario inicializar el sistema completo (Unity y Python) debido a que la comunicación depende de ambos extremos funcionando al mismo tiempo. Por otro lado, en caso de querer tener múltiples clientes conectados realizando diferentes cálculos nos encontramos con problemas del lado de Unity en la asignación de prioridades a la hora de tratar los mensajes.

Revista Abierta de Informática Aplicada/vol 7 Nº1 (2023): 24-36
ISSN 2591-5320

Por tal razón, modificamos el desarrollo para implementar una pieza de software intermedia con función de servidor, encargada de recibir todo mensaje emitido por WebSocket, para luego emitir al resto de clientes conectados. De tal forma, múltiples clientes serían capaces de enviar mensajes a un servidor, el cual los remitirá para oRobby. A su vez, los mensajes emitidos por oRobby también alcanzarán a todos los clientes conectados.

Fig. 14. *Servidor creado en Node, runtime de JavaScript.*

Esta nueva implementación permite una mayor escalabilidad en la comunicación entre oRobby y clientes.

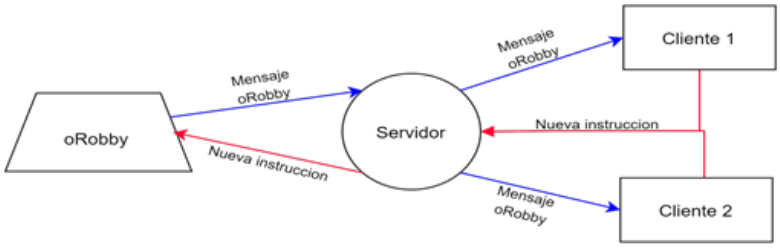

Fig. 15. *Comunicación entre múltiples clientes y oRobby, mediante un servidor intermediario.*

5.6 Compatibilidad

En lo que a la compatibilidad del servicio respecta, al ser comunicación mediante WebSocket, resulta posible implementar un cliente con cualquier herramienta o lenguaje de programación que permita implementar comunicación por WebSockets.

El servidor descrito se encuentra desarrollado en JavaScript. Al mismo tiempo, durante el desarrollo se utilizaron herramientas tales como <WebSocketTesting> y <Browser WebSocketClient>.

5.7 Modelo de la implementación

Desde Unity:

Desde el lado de Unity, mediante un script se publica constantemente mediante la función Update() un JSON, formado por un objeto "Mensaje" que posee:

- Float Posicion X, Posicion Y, Posicion Z.
- Float Angulo X, Angulo Y, Angulo Z.
- Float Distancia Adelante, Derecha, Izquierda, Adelante Derecha, Adelante Izquierda.
- Int Generacion.
- Bool Objetivo.

Al mismo tiempo se hace uso del evento OnMessage(), el cual captará el mensaje recibido, y lo procesa desde .json a un objeto mensaje. Luego compara este mensaje entrante con el último mensaje ingresado a una lista de mensajes. Solo en caso de que este sea diferente al último, el mensaje será introducido a la lista. Tomando este último mensaje para realizar el movimiento.

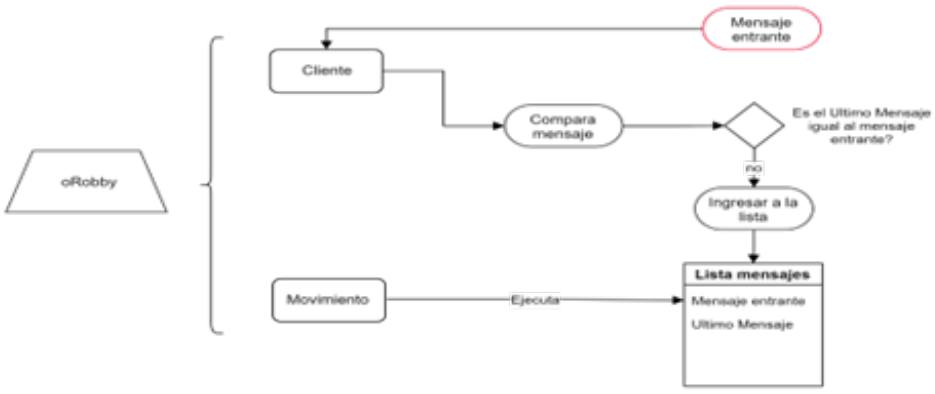

Fig. 16. *Movimiento del robot*

Server:

El servidor creado para la recepción y broadcast de mensajes fue desarrollado en JavaScript. Es una simple aplicación encargada de captar los mensajes de todos los clientes, y emitirlos nuevamente a todos los clientes menos el emisor.

Cliente Python:

Es el responsable de abrir la conexión con funciones asíncronas de captura de mensajes. Estos serán Avanzar (), Retroceder (), Girar () con indicación de giro (valores negativos para la izquierda, y positivos para la derecha).

MQTT con Paho y Mosquitto Broker

Otro de los protocolos probados fue MQTT Eclipse Foundation [9], utilizando Mosquitto como Broker de comunicación encargado de recibir dichos mensajes, y publicarlo mediante la suscripción a tópicos.

Metodologías aplicadas para la comunicación por MQTT

Puesto que es un protocolo que implementa un broker, no resulta necesario gran cantidad de código. Desde el lado de Unity será necesario inicializar de un cliente tomando como ejemplo el realizado en Giovanni [10], la suscripción a los tópicos, y la publicación de los mensajes deseados. Siendo posible la implementación desde cualquier herramienta capaz de suscribir y emitir en un tópico.

Complejidad de la implementación

MQTT resulta simple de implementar gracias al uso de diversas herramientas tales como Mosquitto Broker, y librerías tales como <M2Mqtt> para C# en su adaptación para Unity.

Una gran ventaja de este sistema es lo intuitivo de su uso, ya que el usuario promedio podrá publicar mensajes al robot con una simple instrucción o línea de código de cualquier sistema habilitado para publicar por MQTT.

A pesar de ser un sistema más fácil de implementar comparado con WebSocket, cada cliente deberá suscribir a todos y cada uno de los tópicos utilizados para el cálculo de la nueva instrucción a emitir, resultando esto menos cómodo de usar.

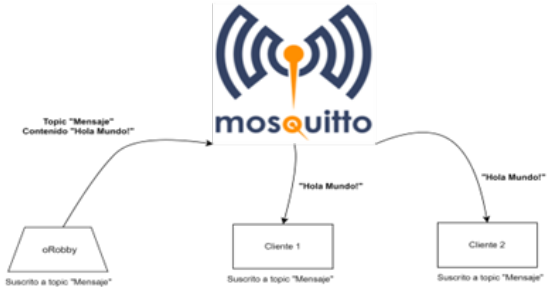

Fig. 17. *Publicación y subscripción de topics.*

Escalabilidad del servicio.

Una gran ventaja del protocolo MQTT es la facilidad de publicación de mensajes desde cualquier cliente con conexión MQTT.

Por tal, resulta posible la publicación desde cualquier dispositivo, tal como un celular, Smartwatch, Arduino, Raspberry Pi, ESP8266, etc.

Por otro lado, múltiples robots pueden suscribir a un mismo tópico y publicar en estos, logrando plantear nuevos desafíos para programación de comportamiento grupal o emergente.

Conclusiones

El simulador desarrollado cumplió con los objetivos propuestos y permitió una integración en el dictado de las clases, superando lo esperado en cuando a curva de aprendizaje de los alumnos. Unity nos permitió rápidamente desarrollar un modelo simulado con un motor de física real, creación de un robot e interfaces con aplicaciones externas en la nube. En cuanto a la compatibilidad con robots diseñamos un protocolo de comunicación con nuestros robots reales te tal forma que sea transparente el control de un robot real o virtual. Logramos integrar las prácticas en inteligencia artificial y los talleres y trabajos prácticos al crear una interfaz de comunicación entre las notebooks y el simulador. A futuro estamos explotando el plugin para ROS con la idea de vincular los robots más avanzados.

Referencias

» Infante Jiménez, C.: Propuesta pedagógica para el uso de laboratorios virtuales como actividad complementaria en las asignaturas teórico-prácticas. *Revista mexicana de investigación educativa*, *19*(62), 917–937. (2014).

» Sáez-López, J.-M., Román-González, M., & Vázquez-Cano, E.: Visual programming languages integrated across the curriculum in elementary school: A two year case study using “Scratch” in five schools. *Computers & Education*, *97*, 129–141. (2016).

» Silicon Valley Robotics: What is the best simulation tool for robotics?: https://robohub.org/what-is-the-best-simulation-tool-for-robotics/ (2021)

» Pitonakova, L., Giuliani, M., Pipe, A., & Winfield, A.: Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators. *Annual Conference Towards Autonomous Robotic Systems*, 357–368. (2018).

» Michel, O. Cyberbotics Ltd. Webots™: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, *1*(1), 5. (2004).

» Casañ Núñez, G. A., & Cervera, E.: *The Experience of the Robot Programming Network Initiative*. (2018).

» Kirtas, M., Tsampazis, K., Passalis, N., & Tefas, A. Deepbots: A Webots-Based Deep Reinforcement Learning Framework for Robotics. *IFIP International Conference on Artificial Intelligence Applications and Innovations*, 64–75. (2020).

» Unity.: Python for Unity API: https://docs.unity3d.com/Packages/com.unity.scripting.python@2.0/manual/inProcessAPI.html

» Eclipse Foundation M2Mqtt: https://github.com/eclipse/paho.mqtt.m2mqtt

» Giovanni Paolo Viganò MQTT-Unity: https://github.com/gpvigano/M2MqttUnity