



11°
Congreso
TUCUMÁN
2023

20
23



CoNaII SI

XI Congreso Nacional
de Ingeniería en Informática
y Sistemas de Información



XI Congreso Nacional de Ingeniería en Informática y Sistemas de Información.

CoNaISI 2023

Universidad Tecnológica Nacional.
Facultad Regional Tucumán

XI Congreso Nacional de Ingeniería en Informática y Sistemas de Información :
CoNalSI 2023 / Gabriela Ivana Aparicio ... [et al.] ; Compilación de José Augusto
Nasrallah ; Gabriela Ivana Aparicio ; Editado por Fernando Cejas. - 1a ed - Ciudad
Autónoma de Buenos Aires : edUTecNe, 2024.
Libro digital, PDF

Archivo Digital: descarga y online
ISBN 978-987-8992-38-9

1. Ingeniería. 2. Sistemas de Información. 3. Tecnología Informática. I. Aparicio, Gabriela
Ivana, comp. II. Nasrallah, José Augusto, comp. III. Cejas, Fernando, ed.
CDD 629.801

Diseño de Tapa e interior: Fernando Cejas



Universidad Tecnológica Nacional – República Argentina
Rector: Ing. Rubén Soro
Vicerrector: Ing. Haroldo Avetta
Secretaría Cultura y Extensión Universitaria: Ing. Federico Olivo Aneiros



Universidad Tecnológica Nacional – Facultad Regional Tucumán
Decano: Mgter Ing Walter Fabián Soria
Vicedecano: Ing. Juan Esteban Campos
Secretario de Vinculación Tecnológica: Ing. Mario Augusto Madariaga
Secretario de relaciones institucionales: Ing. Jorge Molina



edUTecNe – Editorial de la Universidad Tecnológica Nacional
Coordinador General a cargo: Fernando Cejas
Dirección General: Mg. Claudio Véliz
Dirección de Cultura y Comunicación: Ing. Pablo Lassave

Queda hecho el depósito que marca la Ley N° 11.723

© edUTecNe, 2024

Sarmiento 440, Piso 6 (C1041AAJ)

Buenos Aires, República Argentina

Publicado Argentina – Published in Argentina

ISBN 978-987-8992-38-9



Reservados todos los derechos. No se permite la reproducción total o parcial de esta obra, ni su incorporación a un sistema informático, ni su transmisión en cualquier forma o por cualquier medio (electrónico, mecánico, fotocopia, grabación u otros) sin autorización previa y por escrito de los titulares del copyright. La infracción de dichos derechos puede constituir un delito contra la propiedad intelectual.

Simulador de robótica en la nube para prácticas de programación colaborativa

Berenice Lourdes Balich, Tupac Ocampo, Franco Adrián Balich, Néstor Adrián Balich
Laboratorio de Robótica Física e Inteligencia Artificial
Centro de Altos Estudios en Tecnología Informática (CAETI)
Universidad Abierta Interamericana
Ciudad Autónoma de Buenos Aires, Argentina
{berenicelourdes.balich, tupac.ocampo}@alumnos.uai.edu.ar, {francoadrian.balich, nestor.balich}@uai.edu.ar

Resumen

La utilización de nuevas e innovadoras herramientas tecnológicas ha cambiado en estos últimos años el modo en que se dictan las clases universitarias mediante nuevos abordajes pedagógicos y tecnológicos. En el contexto de las restricciones que ocasionó la pandemia estos cambios se aceleraron en gran medida. 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, fue necesario adaptar los contenidos, evaluaciones y talleres a un entorno de clase virtual, muchos de ellos vinculados a elemento físicos y de hardware. Este trabajo expone la investigación y creación de un simulador 3D de robótica con programación distribuida, con un modelo físico realista, que permite el trabajo colaborativo, la programación en Python y la inclusión de inteligencia artificial online en tiempo real. Aplicado y testeado con alumnos de la materia robótica de 3ro de ingeniera en sistemas informáticos

Palabras clave: simulador de robótica, unity, educación virtual, iot, inteligencia artificial.

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 fundamentales, con la ventaja de los entornos lúdicos de aprendizaje [1] y el aprovechamiento de entornos visuales para enseñanza de programación [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.

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[3], Unreal Engine[4], y OpenGL[5]. De los cuales se evaluaron los siguientes requisitos funcionales y no funcionales, velocidad de procesamiento, curva de aprendizaje y complejidad de desarrollo, facilidad de implementación de interfaces externas, y fundamentalmente la existencia de herramientas de código libre y que contara con una gran comunidad de desarrolladores con documentación en línea.

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]. Otro requisito fundamental era el de vincularse con una plataforma de inteligencia artificial para enseñar y probar diferentes algoritmos de navegación, aprendizaje de maquina y redes neuronales para el control de los robots.

Con respecto al docente, el simulador debía permitir elegir rápidamente y de forma sencilla, diferentes escenarios con un motor de simulación física en un entorno 3D realistas similar a los juegos de video.

De dicha comparación y prueba de testeo, se seleccionó la plataforma de Unity, puesto cuenta con un motor gráfico liviano y gratuito para fines no comerciales que utiliza el lenguaje de programación C#, permitiendo aplicar lo aprendido en varias materias correlativas de las carreras de la facultad de tecnologías informáticas de la universidad, y que cuenta con una gran cantidad de herramientas de desarrollo libre en forma de paquetes descargables. La elección de los protocolos de comunicación es muy importante, ya con experiencia el uso de ellos y comparándolos[6] seleccionamos dos modelos para este proyecto WebSocket [7] y MqTT[8]:

- 1) Modelo de comunicación asíncrona con WebSocket.

2) Modelo con comunicación por protocolo MQTT utilizando el broker de código abierto Mosquitto[9].

También se desarrollaron varios programas clientes en Python para un robot que esquivaba 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 Lidars (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.

	Electrónica Arduino	Robot	Lenguajes	Web	Detalle
Tinkercad	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
Copelia Sim	-	SI	C y Python	-	Medio

Líneas de investigación y desarrollo

Los ejes principales de este trabajo son:

- Implementar un simulador de robótica que permita la conexión sincrónica de los alumnos.
- Implementar un simulador que utilice tecnologías web y sea compatible con algoritmos de inteligencia artificial (IA).
- Conexión a las principales plataformas libre para desarrollo en IA, como Google Colab y Jupyter.
- Permitir a los alumnos trabajar de forma colaborativa con programas individuales.
- Permitir a los alumnos crear programas distribuidos en donde cada integrante del grupo ejecuta una parte del programa principal.
- Aumentar el interés y motivación de los alumnos en el aprendizaje de programación para robótica.
- Generar un entorno de enseñanza que permita que los alumnos asimilen los conocimientos de una manera más natural desde la práctica.

Este proyecto se llevó adelante en el laboratorio de robótica física e inteligencia artificial del centro de altos estudios en tecnología informática dentro de las líneas de financiamiento y promoción de Amazon para el desarrollo de sistemas cloud que integren AWS.

Estado del arte en simuladores

A. Pruebas e investigación

Durante una primera instancia evaluativa se investigó y compararon varios simuladores que utilizado en 2020 **Tabla I.** en base a las mejores opciones recomendadas en varios artículos [10], [11] para el uso profesional [12] y educativo. También sumando algunas de las especificaciones de diseño para framework de IA presentadas en otros trabajos [13].

Tabla 1. Comparación de simuladores

Análisis de simuladores:

Tinkercad

Realizamos una primera prueba con un simulador de robótica con Arduino gratuito llamado Tinkercad [14], que utilizamos durante las clases presenciales debido a que permite simular circuitos con Arduino, pero comprobamos que no se podía integrar con la red neuronal que se planteó usar y que no se podía simular un entorno 3D, en donde realizar actividades con los robots.

Gazebo

Gazebo [15] es un simulador de robótica en 3D que funciona en el sistema operativo ROS (Robot Operating System) [16] que se puede ejecutar en las placas Raspberry Pi y se puede programar en una variedad de lenguajes de programación, como, por ejemplo: C, Python y JavaScript. Posee un alto grado de personalización, que resultó ser una limitante para nuestro objetivo, pues la gran cantidad de opciones ocasiona una curva de aprendizaje elevada para alumnos y docentes, ocasionando una experiencia poco intuitiva. Otras limitantes que encontramos fueron, que no posee una interfaz web, y que se necesitaría que cada alumno posea su propia placa Raspberry Pi o máquina virtual en la cual ejecutarlo.

Webots

Con la experiencia buena experiencia de un proyecto realizado en el año 2020 con Webot [17], [18] en donde creamos un robot que utilizaba una red neuronal para desplazarse en un ambiente simulado evitando los obstáculos, rápidamente notamos que necesitábamos dos clases para enseñar a usar el simulador. Desarrollamos también una nueva interfaz de comunicación con el exterior basada en MQTT con una interfaz de control y programación visual en sitio web desarrollado en JavaScript sobre node.js que permitía el control y programación remota. Si bien se pudo avanzar en dichos objetivos, no pudimos crear un sitio web en donde se ejecutará el simulador para evitar la instalación local en la computadora de cada alumno.

VR VEX

Otra prueba que se realizó fue con un simulador de robótica gratuito llamado “Vr Vex” [19] desarrollado en Unity, ya utilizado durante las clases presenciales, con la gran ventaja que permite simular un robot en un sitio web. Permite tanto programar en Python como en forma visual por bloques, pero comprobamos que no se podía integrar con la red neuronal que se planteó usar, ya que no podíamos agregar

funciones externas de comunicación y que lamentablemente ya la opción gratuita no permite programar en Python, además de que no permite la programación concurrente.

Copelia Sim

En esta prueba encontramos que “CopeliaSim” [20] posee una curva media de aprendizaje y es un excelente simulador con una interfaz intuitiva. Sin embargo, se necesitaron dos clases para enseñar a los alumnos a utilizarlo, tampoco posee una interface nativa de instalación en un sitio web, principales razones por la que fue descartado.

En cuanto a la evaluación de motores de videojuegos comenzamos las pruebas con Unreal dada su alta calidad de imagen y luego con Unity por su facilidad de uso y gran comunidad de desarrolladores **Tabla II**.

Tabla II. Comparación de lenguaje de programación para los simuladores

Motor	Lenguaje	Interfaz remota	Python	Detalle
Unity	C#	SI	NO	Mucha documentación, curva de aprendizaje rápida.
Unreal	C++	SI	Con plugin, solo para cosas auxiliares	Menor documentación, mayor curva de aprendizaje

Unreal vs Unity

Se evaluaron dos motores físicos empleados para videojuegos. Entre las necesidades se buscó un motor que se programara en Python, que sea gratuito y con mucha documentación, permitiera conexión TCP, se ejecutara en entorno web y con un motor de física real.

UnReal indicaba que se podía programar en Python, por ello fue el primer evaluado. Las pruebas se realizaron creando programas de movimiento y conexión tcp, pero rápidamente fue descartando, debido a que la programación en Python se realiza solo por medio de un plugin en su versión beta y es solo para algunos scripts dentro del motor, el cual está diseñado para ser programado en C++ o por Blueprints (programación en bloques), posee pocos ejemplos de programación en Python y en su mayoría corresponden a scripts de organización de materiales.

Unity fue la segunda opción pues se programa en C# pero en este punto era igual a Unreal. Rápidamente encontramos una gran comunidad de programadores con documentación, capacidad para desarrollo en WebGL entorno web, y permitía conexión TCP con variadas librerías para websocket y MQTT. Con lo cual lo seccionamos para el proyecto.

Desarrollos propios para simulación de robots y pruebas

Se optó por dos líneas de desarrollo para la interfaz de comunicación a testear:

Cliente externo

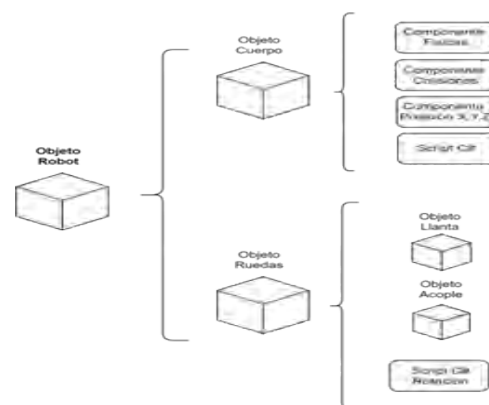


Figura 1. Relación de jerarquía entre objetos y componentes.

La primera que el simulador permita la conexión externa con un cliente Python con el objetivo de crear una interfaz web y propiciar la escalabilidad para desarrollar algoritmos con inteligencia artificial conectados a Google Colab [21] o Jupyter notebook [22], en donde cada cliente puede enviar ordenes al robot mediante un protocolo creado a tal efecto y una jerarquía de objetos (**Figura 1**).

Programación en simulador

La segunda que la programación se realice dentro del simulador a través de una pantalla especial.

Encontramos que no fue posible crear una pantalla con un compilador Python dentro de Unity, con lo cual el desarrollo se enfocó en el primer lineamiento de desarrollo “Cliente externo”.

Desarrollo en Unity e implementación

Unity hace uso de C# como lenguaje nativo con el cual el equipo de desarrollo ya estaba familiarizado pues es enseñado en las materias de Programación I, Programación Orientada a Objetos, y Lenguajes de Última Generación dictadas en la universidad.

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 física, 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.

Al mismo tiempo, un objeto podrá referenciar otro objeto o componente diferente desde el Script, permitiendo así vincularlos y conseguir un funcionamiento coordinado (ver **Figura 2**).

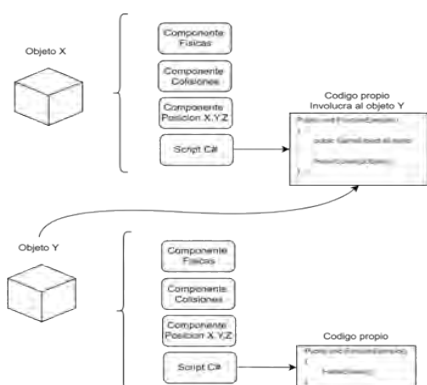


Figura 2 Coordinación entre objetos.

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 (ver **Figura 2**), en donde un conjunto de objetos formará parte de un objeto padre. Los comportamientos de este último serán reflejados en los objetos hijos (ver **Figura 3**).

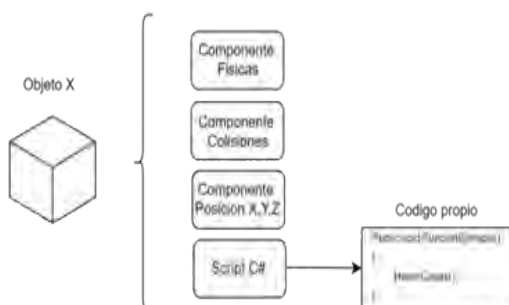


Figura 3. Representación de la relación entre objetos, componentes y código propio

Llevando este desarrollo a la realidad nos encontramos con un robot formado por diferentes componentes, los cuales, a su vez, mediante programación en C#, implementan funcionalidades específicas a la vez que referencian otros objetos (ver **Figura 4**), con un consiguiente funcionamiento sincronizado. Aplicando todas estas reglas será posible el desarrollo de objetos que posean diferentes componentes y funcionalidades.

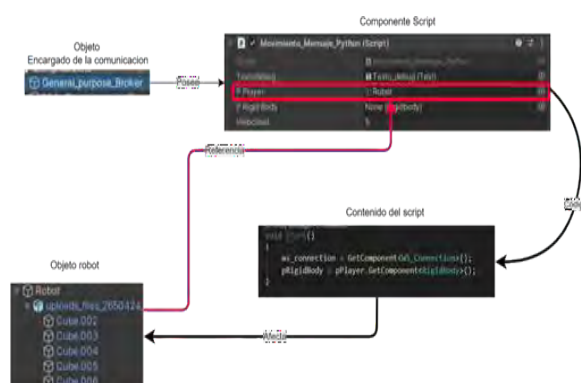


Figura 4. Representación de la relación entre objetos, componentes y código propio

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.

En la **Figura 5**, 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.



Figura 5. Vista de la interface unity.

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 (ver **Figura 6**).

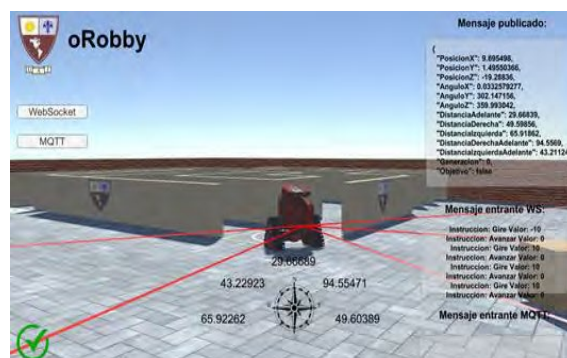


Figura 6. oRobby, robot desarrollado en el laboratorio, funcionando.

Funcionalidades buscadas

El desarrollo de nuestro propio simulador requirió un enfoque general, desde la infraestructura, modelo de comunicación y formar nuestro propio equipo de desarrollo, enfocados en las necesidades del laboratorio, docentes y alumnos.

El objetivo central consistió en la implementación de una herramienta que permita realizar programas en Python, c# y

javascript para integrarlo con las materias de programación que el robot debe ejecutar en el simulador. Ampliando luego a los cursos más avanzados de inteligencia artificial. Optamos por un desarrollo iterativo e incremental mediante tecnologías ágiles, con punto de partida en el manejo de un robot básico, desarrollado en Unity

Debido a 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 intérprete de Python para el entorno de Unity, “Python for Unity” [23] actualmente carente de mantenimiento, e incompatible con versiones actuales de Python.

Se enfocó el desarrollo 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.

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 (ver Figura 7).



Figura 7. Plataforma oRobby Web actualmente en desarrollo, con Python Flask.7. Envío y recepción de nuevos mensajes.

- 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 implementada, que ofrece funcionalidades extras con interfaces de conexión con Raspberry Pi y Arduino.

Protocolos de comunicación utilizados

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

Modelo de implementación 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.

Comunicación

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 formula una nueva instrucción, la cual será devuelta a oRobby de acuerdo con la estrategia de navegación.

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

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

Velocidad

Para poder procesar en tiempo real 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.

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.

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.

Por tal razón, modificamos el desarrollo para implementar una pieza de software intermedia con función de servidor utilizando Node.JS con JavaScript, 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.

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

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>.

Modelo de la implementación MQTT

Comunicación

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.

- 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 de Eclipse Foundation [24], utilizando Mosquitto como Broker de comunicación encargado de recibir dichos mensajes, y publicarlo mediante la suscripción a tópicos.

Velocidad

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 MqTT [25], 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. Su velocidad es elevada y fue creado para un servicio de publicación/subscripción de mensajes pequeños haciendo ideal para IoT.

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.

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.

Compatibilidad

En lo que, a la compatibilidad del servicio respecta, al ser comunicación mediante MQTT, resulta posible implementar un cliente con cualquier herramienta o lenguaje de programación que permita implementar comunicación por MQTT, en el caso de Google Colab se notó un retraso por la forma en que su servidor administra los puertos de comunicación.

Formación de recursos humanos

El equipo está formado por 2 profesores e investigadores del laboratorio de robótica física e inteligencia artificial (LRFIA) de la Universidad Abierta Interamericana (UAI), 1 alumno de 3to año de ingeniería en sistemas (UAI) y 1 alumna de 2to año de la licenciatura en producción de simuladores y videojuegos (UAI).

En este proyecto también colaboraron alumnos pasantes del laboratorio.

Se realizaron cursos internos de programación en Unity, programación en C#, IoT, procesamiento distribuido y desarrollo web, como así también charlas y consultas con la cátedra de video juegos a los cuales agradecemos por la colaboración.

RESULTADOS OBTENIDOS

El simulador desarrollado cumplió con los objetivos esperados y permitió una integración en el dictado de las clases, logrando los tiempos fijados 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 webs externas en la nube como es google colab. En cuanto a la compatibilidad con los robots diseñamos un protocolo de comunicación con nuestros robots reales de tal forma que sea transparente el control de un robot real o virtual. Logramos integrar las prácticas en inteligencia artificial, los talleres y trabajos prácticos al crear una interfaz de comunicación entre las notebooks de Google Colab, Jupyter y el simulador base Api rest, con lo cual se desarrollaron prácticas de programación en javascript, Python, c# y node.js. En cuanto a la conexión Mqtt nos permitió integrar el simulador con plataformas como Adafruit y con NodeRed.

El primer grupo de 9 alumnos desarrollo varios programas no solo para alcanzar los objetivos de la materia, sino que

continuaron con desarrollos propios, la dinámica de clase colaborativa fue excelente pues en las exposiciones y presentaciones grupales, se intercambiaron experiencias entre equipo de trabajo.

Conclusiones

En conclusión, este paper comparte las experiencias y recursos utilizados para la creación del simulador OROBBY comenzado en 2020 en base al primer prototipo mínimo viable que no permitía conexión en línea pues solo tenía un protocolo de comunicación WebSockets y con conexión de un alumno a la vez.

En este trabajo concretamos y validamos el uso de los protocolos de comunicación Mqtt y Socket, la conexión múltiple de alumnos y la implementación de inteligencia artificial mediante una clase Python que se agrega a la notebook de Google Colab, Jupyter o programa cliente en Python todo gracias a la nueva funcionalidad mediante un api de comunicación API Rest.

Se diseñaron varias actividades para el dictado de las clases de programación y robótica con Python y C# a los alumnos de la materia “Paradigmas tecnológicos II” (robótica) en el año 2021 un año después de la fecha del CIITI. Validando la efectividad del abordaje pedagógico y del funcionamiento del simulador. Los alumnos mostraron especial interés y se logró realizar 10 actividades en el tiempo en que antes se realizaban 2. Se formaron grupos de estudio y competencia, ya que los desafíos incluían, resolver laberinto en equipo en el menor tiempo posible, navegación reactiva y proactiva, navegación por redes neuronales empleando Google Colab, y actividades de programación escalonada por turno, en donde cada integrante del equipo ejecutaba su parte del código desde su computadora en forma virtual, al tiempo que el resto de la clase observaba en simulador en tiempo real. También se realizaron desafíos de algoritmia en modalidad rescate.

Como objetivo para la próxima etapa, se plantea un nuevo modelo sobre la base de este trabajo, que permita que cada alumno incorporar modelos y patrones de trabajo colaborativo, juegos en equipo por ejemplo fútbol de robot, aprendizaje de máquina y la incorporación de más sensores de ambiente para los robots y la creación de diferentes escenarios al estilo de olimpiadas de robótica.

En este escenario el profesor muestra en tiempo real el simulador mientras los alumnos programa su robot y el resto de los alumnos los ve en para permitir tiempo real, por ello la próxima versión del simulador plantea la presencia de 31 robots en tiempo real (uno por alumno y profesor) y la posibilidad que cada alumno descargue una versión local para no depender del simulador centralizado que ejecuta el profesor.

BIBLIOGRAFIA

- [1] 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).
- [2] 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).

- [3] Unity <https://unity.com/>
- [4] Unreal <https://www.unrealengine.com/es-ES/>
- [5] OpenGL <https://en.wikipedia.org/wiki/OpenGL>
- [6] Carsten Gregersen, WebSocket vs. MQTT vs. CoAP: Which is the Best Protocol? <https://www.nabto.com/websocket-vs-mqtt-vs-coap/>, 2021
- [7] WebSocket <https://www.ibm.com/docs/es/was/9.0.5?topic=applications-websocket>
- [8] MQTT <https://aws.amazon.com/es/what-is/mqtt/>
- [9] Mosquito <https://mosquitto.org/Tinkercad> <https://www.tinkercad.com/>
- [10] Silicon Valley Robotics: What is the best simulation tool for robotics?: <https://robohub.org/what-is-the-best-simulation-tool-for-robotics/> (2021)
- [11] 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).
- [12] Michel, O. Cyberbotics Ltd. WebotsTM: Professional mobile robot simulation. International Journal of Advanced Robotic Systems, 1(1), 5. (2004).
- [13] Casañ Núñez, G. A., & Cervera, E.: The Experience of the Robot Programming Network Initiative. (2018).
- [14] TinkerCad
- [15] Gazebo <https://classic.gazebosim.org/>
- [16] ROS Robot Operating System <https://www.ros.org/>
- [17] Webots <https://www.cyberbotics.com/#webots>
- [18] 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).
- [19] Vr vex <https://vr.vex.com/>
- [20] Copelia sim <https://www.coppeliarobotics.com/>
- [21] Google colab <https://colab.research.google.com/>
- [22] Jupyter notebook <https://jupyter.org/>
- [23] Unity docs, Python for Unity API API is meant for scripts that are stateless, which means that they do not keep information between runs <https://docs.unity3d.com/Packages/com.unity.scripting.python@2.0/manual/inProcessAPI.html>
- [24] Eclipse Foundation, MQTT Client Library for .Net and WinRT <https://github.com/eclipse/paho.mqtt.m2mqtt>
- [25] Giovanni Paolo Viganò, M2MQTT for Unity <https://github.com/gpvigano/M2MqttUnity>