# MODELO PARA LA SIMULACIÓN DE SISTEMAS DE MULTI-AGENTES ROBÓTICOS EN PYTHON

## MODEL FOR THE SIMULATION OF MULTI-AGENT ROBOTIC SYSTEMS IN PYTHON

ANDRÉS CAMILO JIMÉNEZ
JOHN PETEARSON ANZOLA
GIOVANNY MAURICIO TARAZONA
SANDRO JAVIER BOLAÑOS

**Resumen**

En la etapa de diseño de Sistemas Multi-Agentes Robóticos, la validación de algoritmos y la verificación del modelo cinemático inverso y directo, es importante para la detección de problemas o errores antes de implementarlos en el agente físico. En este artículo se presenta el diseño de un modelo en Pyhton para la simulación de agentes robóticos de dos ruedas compuesto por sub sistemas de agentes, validando el modelo cinemático del agente robótico dentro de un laberinto.

**Palabras clave:** Modelo, Multi-Agentes Robóticos, Python, Sistema.

**Abstract**

In the design stage in Multi-Agent Robotic Systems, the validation of algorithms and the verification of the inverse and forward kinematic models of the robot, is important to detect problems or errors before implementing them into the physical platform. This paper describes the design of a model in Python for the simulation of a two-wheeled robot agent that is designed by sub-agent, validating the kinematic model of the robotic agent in a maze.

**Key words:** Model, Multi-Agent Robotics, Python, System.

## Introducción

La verificación de los algoritmos que se van a desarrollar en el agente robótico es una de las etapas más importantes para el diseño de Sistemas Multi-Agentes (por sus siglas en Inglés: MAS) orientados a la robótica, esto permite la detección temprana de errores antes que sean implementados en el agente físico, para esto se han diseñado arquitecturas en el diseño de Sistemas Multi-Agentes Robóticos (por sus siglas en Inglés: MARS). Un modelo de arquitectura por capas, permite representar el comportamiento de un agente con el sistema por medio de módulos de percepción, toma de acciones usando lógica difusa y de aprendizaje aplicando redes neuronales, separando cada uno de estos módulos en dos niveles de comportamiento uno alto y uno bajo, el nivel bajo es la representación de movilidad y el nivel alto la comunicación entre agentes, los cuales al ser aplicados en un sistema robótico logran obtener resultados satisfactorios en la robo-copa del año 2000, más no permite la verificación de algoritmos de SLAM o planificar rutas en espacios no controlados [1].

Por otro lado, se han clasificado las arquitecturas de los MAS como estructuras de organización, aplicando la "estructura-en-5"[2] y la estructura "join venture"[3], en el control de un robot móvil por medio del modelamiento en i*, demostrando una enorme ventaja frente a las arquitecturas clásicas (arquitectura por capas, lasos de control y tareas en forma de árbol) en temas de coordinación, previsibilidad, tolerancia a fallos y adaptabilidad, aunque siendo esta arquitectura basada en modelos MAS necesita de un árbitro para la toma de decisiones del sistema [4]. En [5] desarrollan otra arquitectura para MARS basándose en el estilo "arm's-length"[4] para el diseño de una arquitectura de control del robot junto con el enfoque de control colaborativo, en este trabajo el MAS es propuesto como una arquitectura de control dividida en 4 diferentes categorías de agentes como se muestra en el mapa mental de la Figura 1.


**Figura 1.** Arquitectura de control colaborativo
**Fuente**: [5]

Las arquitecturas orientadas a los MARS permiten el desarrollo de modelos para la especificación, el diseño o simulación de agentes de software, capaces de implementarlos directamente en los agentes robóticos físicos, sea para emular los agentes o para programarlos directamente evaluando su comportamiento. En [6] se propone un kit de referencia de diseño (por sus siglas en Inglés: RDK) para el diseño de robots con hardware no especificado, pero el framework nombrado SPQR-RDK no es comercial. En [7] se presenta un framework denominado BABEL que se basa en el enfoque de ingeniería de software para el diseño de aplicaciones en la robótica sin importar el lenguaje que se desee implementar al final, pero este se limita a generar el modelo con plataformas comerciales. Por otro lado, se tienen frameworks orientados a la aplicación directa en el robot a nivel multi-agente, como en el caso de [8] que usa una red de sensores inalámbricos (por sus siglas en Inglés: WSN) para la localización y en [9] para el diseño del módulo de comunicaciones en drones.

Este artículo presenta un modelo de simulación para agentes robóticos desarrollado en Python y soportado por el motor de juegos PyGame[10], capaz de ser ejecutado en cualquier sistema operativo para la simulación y evaluación de una plataforma móvil de dos ruedas, basándose en el modelo de multi-agente presentado en [5], con la

diferencia que se incluyen a otros agentes robóticos en el sistema de manera descentralizada, además a diferencia de otros frameworks o modelos, al hacer uso del motor de juegos se pueden simular variables físicas como el ruido, la fricción, la temperatura y la aceleración gravitacional, siendo posible ser modificadas por el usuario.

El artículo está estructurado de la siguiente manera: primero, el modelamiento en UML de los agentes que componen el sistema se detalla en la sección 2, proseguido en la sección 3 se evidencia el modelo matemático del agente robótico y por último, en la sección 4 se muestran los resultados de la implementación del agente en un laberinto

## Diseño de los agentes del sistema

El modelo de simulación propuesto en este artículo se compone de cuatro agentes como se muestra en la Figura 2, de los cuales, tres de estos son principales y uno es un sub-agente. Los tres agentes principales son: el agente robótico encargado del procesamiento del entorno, el agente obstáculo encargado de generar el entorno y el agente objetivo encargado de finalizar la tarea de búsqueda. El agente robótico a su vez contiene al sub-agente denominado agente sensor, el cual se encarga de capturar los cambios en el entorno. Previo del diseño general del sistema se debe plantear los requisitos de cada uno de los cuatro agentes y la relación entre estos, exponiendo sus requerimientos funcionales y no funcionales.

## Diseño del agente sensor

El agente sensor (AS) se encarga de capturar los cambios o detectar los obstáculos en el entorno, este agente envía directamente al agente robot (AR) los cambios de distancia, sin ningún tipo de acople al modelar un sensor digital, de otra manera, es obligatorio que el AS sepa la posición en donde se instala en el agente robot con el fin de no generar redundancia en la información capturada, los requerimientos del agente son mostrados en la tabla 1.

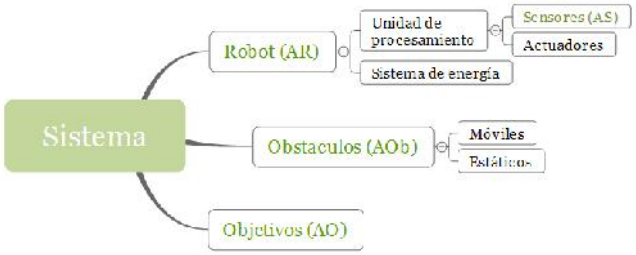



**Figura 2.** Arquitectura propuesta para el modelo de simulación.
**Fuente**: Autores

**Tabla 1.** Requerimientos del Agente Sensor.

| Funcionales | No-Funcionales |
|---|---|
| • Medir la distancia entre el entorno y el agente robot.<br>• Reportar la detección de obstáculos o posibles agentes objetivos. | • Múltiples AS pueden pertenecer a un AR.<br>• Tener rangos establecidos de distancia.<br>• Conocer la posición angular al ser instalados en un AR. |

## Diseño del agente robot

El AR a diferencia del modelo presentado en [5] se encarga de realizar las tareas de procesamiento, comunicarse con los demás agentes, modificar los actuadores y capturar la señal de los AS. Para las tareas de navegación el agente se encarga de almacenar el trayecto, realizar el control de los motores y procesar el entorno, permitiendo la instalación de estos de manera distribuida o en un punto fijo. En la tabla 2 se muestran los requerimientos del AR.

**Tabla 2.** Requerimientos del Agente Robot.

| Funcionales | No-Funcionales |
|---|---|
| • Recorrer el entorno.<br>• Guardar la ruta.<br>• Procesar la información de los sensores.<br>• Permitir exportar los datos del recorrido.<br>• Procesar los algoritmos de navegación y evaluar los objetivos asignados. | • Tener un número de identificación único del sistema.<br>• Capacidad de comunicación con otros agentes.<br>• Conocer la posición en el entorno.<br>• Interpretar el entorno para diferenciar entre agentes obstáculo y objetivo. |

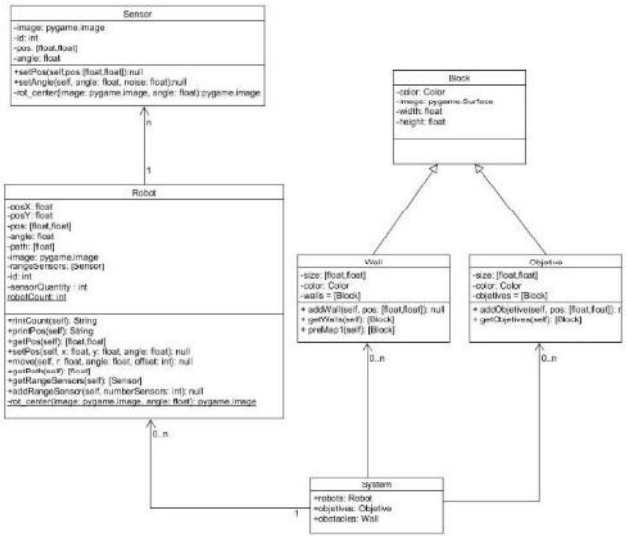



**Figura 3.** Arquitectura propuesta para el modelo de simulación
**Fuente**: Autores

## Diseño del agente obstáculo y objetivo

Los Agentes Obstáculo (AOb) y Objetivo (AO) se modelan de la misma manera, siendo diferenciados por su comportamiento cuando son detectados por el AS; de ser AOb los AS reaccionan rechazándolos, mientras que los AO atraen al AR. Estos dos agentes para ser agregados en el sistema deben tener la posición en el entorno y saber si son estáticos o dinámicos. En la tabla 3 se muestran los requerimientos del AOb y del AO. La integración de todo el modelo de la figura 2 en UML[11] se muestra en la figura 3.

**Tabla 3.** Requerimientos del Agente Obstáculo y Objetivo.

| Funcionales | No-Funcionales |
|---|---|
| • Permitir la adición de múltiples objetos a una lista, sean obstáculos u objetivos. | • Cada elemento objetivo debe tener un identificador propio que los diferencie del sistema. |
| • Debe comportarse como un objeto estático o dinámico según se instancie. | • El agente AOb debe tener un método de entorno predeterminado para hacer pruebas. |

## Modelo matemático del sistema

El modelo cinemático de un robot de dos ruedas es descrito en la ecuación (1), siendo la inversión del el eje *y* obligatoria al manejar la librería PyGame tal como se muestra e la figura 4, otros modelos cinemáticos no aplican este modelo al no tener los ejes invertidos en $R^2$ [12]–[14].

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix} cos(\theta) & 0\\ \sin(-\theta) & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix} r\\ \theta\end{bmatrix} + \begin{bmatrix} x\\ y\\ \phi\end{bmatrix} \quad (1)$$

En donde:

$\dot{x}$ = Nueva posición con respecto al eje *x*.

$\dot{y}$ = Nueva posición con respecto al eje y.

$\dot{\theta}$ = Nueva posición del ángulo con respecto al eje *x*.

$\vec{r}$ = Vector velocidad del robot.

$\theta$ = Ángulo de giro con respecto al eje *x*.

$x$ = Posición actual del eje *x*.

$y$ = Posición actual del eje *y*.

$\phi$ = Posición actual del ángulo con respecto al eje *x*.

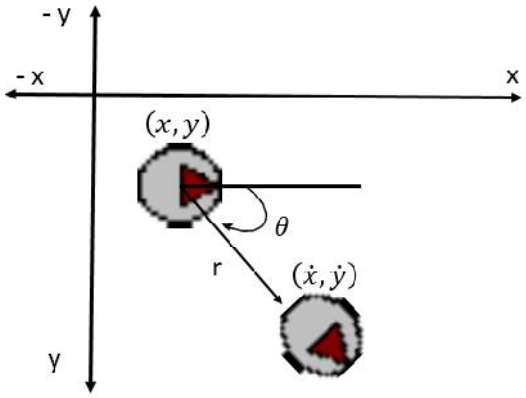



**Figura 4.** Plano cartesiano manejado en el modelo de simulación.
**Fuente**: Autores

Para agregar los sensores de manera distribuida en el AS se aplica la ecuación (2), siendo $\theta_a$ la apertura del ángulo entre sensores y *n* la cantidad de sensores que se desean agregar al robot ver figura 5a.

$$\theta_a = \frac{180}{n+1}$$

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}cos(\theta + i * \theta_a) & 0\\ \sin(-\theta - i * \theta_a) & 0\end{bmatrix}\begin{bmatrix}x_{rad}\\ y_{rad}\end{bmatrix} \quad (2)$$

$$i = 1 \dots n$$

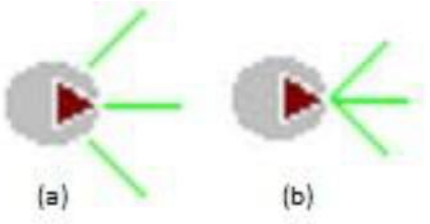



**Figura 5**. (a) 3 sensores distribuidos en el robot, (b) 3 sensores en un solo punto.
**Fuente**: Autores

Si los sensores no se desean distribuidos sino en un solo punto la ecuación (2) se puede reescribir dando como resultado la ecuación (3) como se ve en la figura *5b*.

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}cos(i * \theta_a) & 0\\ \sin(-i * \theta_a) & 0\end{bmatrix}\begin{bmatrix}x_{rad}\\ y_{rad}\end{bmatrix} \quad (3)$$

$$i = 1 \dots n$$

Las ecuaciones (1), (2) y (3) son modelos ideales que se pueden implementar en el modelo, pero además de tener estos modelos, se pueden agregar parámetros de ruido Gaussiano durante la ejecución de la simulación generando una nueva ecuación cinemática y otra para la lectura de los sensores como se muestra en las ecuaciones (4) y (5), en donde $\sigma$ es la desviación estándar que varía dependiendo del modelo de los motores y los sensores, esto permite modelar problemas generados por la lectura de la odometría mecánica y el movimiento del AR durante el trayecto [13], [15].

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}cos(\theta) + \frac{1}{\sigma 2\pi} * e^{-\frac{(x-\theta)^2}{2\sigma^2}} & 0\\ \sin(-\theta) + \frac{1}{\sigma 2\pi} * e^{\frac{(x-\theta)^2}{2\sigma^2}} & 0\\ 0 & \frac{1}{\sigma 2\pi} * e^{\frac{(x-\phi)^2}{2\sigma^2}}\end{bmatrix}\begin{bmatrix}r\\ \theta\end{bmatrix} + \begin{bmatrix}x\\ y\\ \phi\end{bmatrix} \quad (4)$$

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}cos(i * \theta_a) + \frac{1}{\sigma 2\pi} * e^{-\frac{(x-\theta)^2}{2\sigma^2}} & 0\\ \sin(-i * \theta_a) + \frac{1}{\sigma 2\pi} * e^{-\frac{(x-\theta)^2}{2\sigma^2}} & 0\end{bmatrix}\begin{bmatrix}x_{rad}\\ y_{rad}\end{bmatrix} \quad (5)$$

$$i = 1 \dots n$$

Por otro lado, al estar en un espacio euclidiano $R^2$ y $R^3$, es posible encontrar la distancia entre dos o más agentes robóticos aplicando la ecuación (6) tal como se ve en la figura 6, donde $x_n$ y $y_n$ son referencias en el espacio euclidiano de distancia, siendo las unidades elegidas por el usuario a la hora de crear el escenario desde el sistema. Es importante resaltar que esta distancia se puede encontrar en cualquier instante en este modelo, a menos que el usuario modifique el AR y el AS para generar modelos de sensores inalámbricos, de ser

así, se debe crear una restricción para encontrar la distancia si están en radio de comunicación, y la distancia se calcula usando el Indicador de Potencia de la Señal Recibida (RSSI, por sus siglas en Inglés), tal y como se evidencia en [16]–[19].

$$d = \sqrt{(x_1 + x_2)^2 + (y_1 + y_2)^2} \quad (6)$$

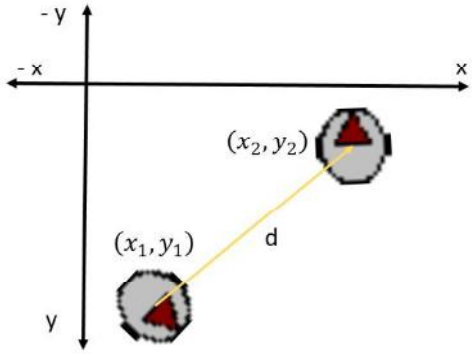


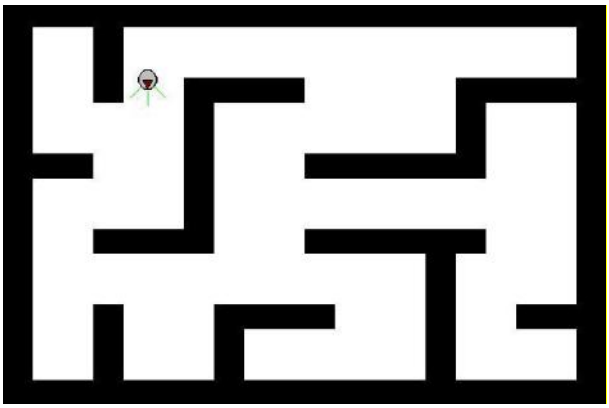

**Figura 6.** Distancia Euclidiana entre dos agentes robóticos.
**Fuente**: Autores

## Implementación del modelo

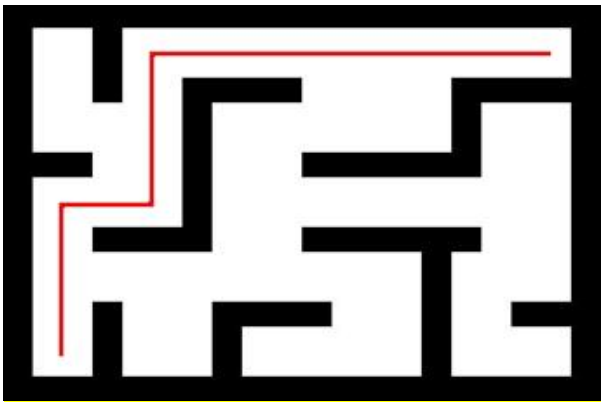
En esta sección se muestra el modelo de simulación en ejecución, en la figura 7a se puede apreciar cómo se usa la función *preMap1* de la clase *Wall* (figura 3) para generar un mapa con un área de 20x16 bloques de color blanco, mientras que el agente robótico es instanciado por medio de la clase *Robot* (figura 4) con tres sensores de distancia distribuidos como se aprecia en la figura 8 por medio de la ecuación (2) siendo la función de estos sensores la detección de los muros y dar prioridad para evadirlos por izquierda haciendo giros de 90 grados como se puede apreciar en la figura 7b, sin embargo se pueden implementar algoritmos para la resolución de laberintos como se exponen en [20] e implementar el árbol de decisión expresado en [21] o en algoritmos de comunicación [22].

De otra manera este modelo al estar diseñado para trabajar con muli-agentes se pueden agregar *n* robots con diferentes atributos en su sub-sistema de agentes como se muestra en la figura 8, en donde se tienen 3 agentes robóticos con una cantidad diferente de sensores

Se compara e interpreta los resultados obtenidos de manera clara y precisa, los autores deben exponer sus propias opiniones sobre el tema, tendencias, ventajas, limitaciones entre otros.



(a)



(b)

**Figura 7.** (a) Uso del modelo con un agente robótico en un laberinto, (b) Resultado del recorrido al solo evadir obstáculos por la izquierda.
**Fuente**: Autores

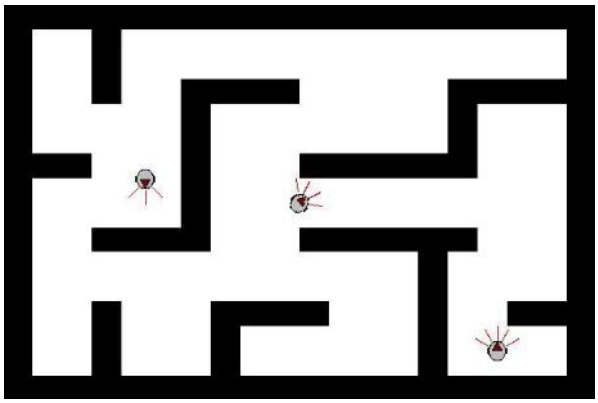


**Figura 8.** Uso del modelo con múltiples agentes robótico en un laberinto.
**Fuente**: Autores

De otra manera este modelo al estar diseñado para trabajar con muli-agentes se pueden agregar *n* robots con diferentes atributos en su sub-sistema de agentes como se muestra en la figura 8, en donde se tienen 3 agentes robóticos con una cantidad diferente de sensores.

Siendo importante analizar los resultados de la navegación en un entorno, se aplicaron tres diferentes experimentos sin obstáculos para demostrar la aplicación de los métodos *getPath* en los agentes Robot, en los dos primeros casos se deja al agente robótico realizar treinta movimientos en 90 y 45 grados para al final mostrar el recorrido de las 30 instrucciones por medio de una línea verde y una leyenda mostrando el indicativo de la acción como se puede apreciar en la figura 9, mientras que en el tercer caso se ponen dos agentes a ejecutar diez instrucciones en las cuales deben incrementar diez grados por paso con un 10% más de velocidad de un agente con respecto al otro, los resultados de este trayecto son modelados por una librería externa al modelo conocida como *Matplot* de Python[23] como se puede apreciar en la figura 10, demostrando la versatilidad del modelo de simulación en importar los resultados.

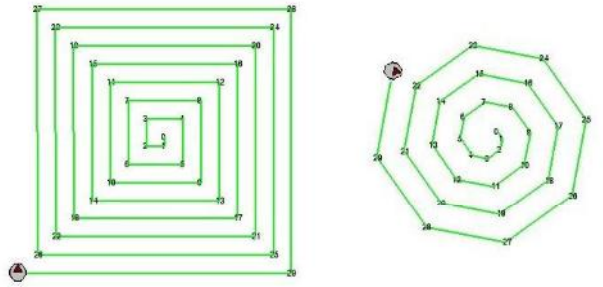


**Figura 9**. Generación de rutas en dos agentes robóticos.
**Fuente**: Autores

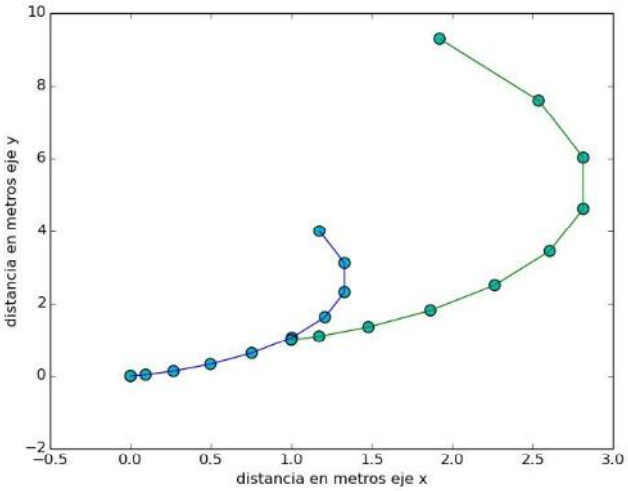



**Figura 10**. Generación de rutas importadas por el modelo en *Matplot*
**Fuente**: Autores

Por último, en la figura 11 se muestran los resultados al usar la ecuación (4) para la creación de los agentes robóticos con una velocidad constante en las dos llantas de 10 $cm/s$, el "Robot 1" presenta un $\sigma = 0$, es decir, tiene un error del 0% respecto a su movimiento mecánico de los motores, el "Robot 2" un $\sigma = 3$, en este caso representa un error del 3%, el "Robot 3" un $\sigma = 5$ y el "Robot 4" un $\sigma = 10$.

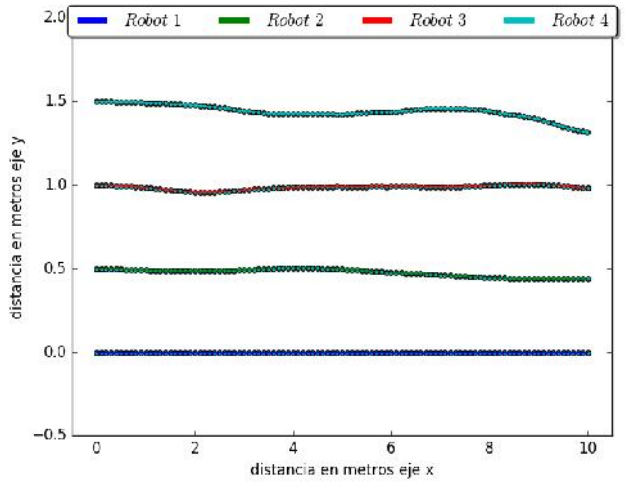



**Figura 11.** Generación de rutas usando el modelo de la ecuación (4) en *Matplot*
**Fuente**: Propia

## Resultados

Luego de realizar diferentes pruebas con el modelo de simulación en diferentes sistemas operativos Ubuntu, Fedora y Windows, se compararon los resultados con diferentes frameworks obteniendo los resultados de la tabla 4

**Tabla 4.** Resultados frente a otros modelos o frameworks

| Nombre Framework/modelo | Multiplataforma | Multi-agente | Comercial/Disponible |
|---|---|---|---|
| Arquitectura modular [1] | Si | Si. Solo se lograron resultados en la robo-copa. | No/No |
| Modelo MARS basado en estructuras de organización [2] | No | Si. Se debe tener una unidad central para realizar las tareas de procesamiento. | No/Si |
| Control colaborativo MARS | No | Si, pero solo a nivel sub-agente | No/No |
| SPQR-RDK [4] | Si, pero solo con plataformas físicas comerciales como el robot AIBO de sony | Si. Resultados en robo-copa y robo-copa modalidad de rescate | No/No |
| BABEL [5] | Si. El framework se centra en resolver problemas en la integración de software robótico. | No | Si/Si |
| Modulo propuesto en el artículo | Si. Se realizaron pruebas en Ubuntu, Fedora, Raspbian y Windows sin presentar ningún conflicto. Tambien tiene la posibilidad de exportar los resultados de navegación para procesarlos en diferentes herramientas como Matplot. | Si, a nivel de sistema y de sub-agentes con una plataforma móvil en dos ruedas. | Si/Si |

## Conclusiones

El modelo presentado en este artículo se puede implementar en cualquier sistema operativo al ser elaborado en Python, logrando desarrollar algoritmos de navegación con un sistema de múltiples agentes robóticos de dos ruedas para

navegar un laberinto. Los agentes robóticos están diseñados como un sub sistema de múltiples agentes, de tal manera que se pueda modificar la posición de los agentes sensores al igual que la cantidad de estos en los agentes robóticos de manera independiente, además permite importar de manera sencilla los resultados del recorrido a otras librerías de Python como Matplot. Por otro lado, al utilizar el motor de juegos PyGame se logra modelar variables físicas en la simulación logrando obtener resultados más cercanos al modelo real.

Como trabajos futuros se deben agregar nuevos módulos para el agente robot y más tipos de sensores, con el fin, que el usuario pueda implementar sus propios modelos dinámicos y cinemáticos de robot como se puede evidenciar en [13], [15]. Además, se está elaborando un módulo de comunicaciones para monitorear en tiempo real el comportamiento con un sistema multi agente robótico y compararlo con el modelo presentado en el artículo, esto permitirá validar de manera más precisa el funcionamiento del modelo de simulación.

## Referencias

[1] H. Hu and D. Gu, "A Multi-Agent System for Cooperative Quadruped Walking Robots," pp. 1–5, 2000.

[2] H. Mintzberg, "STRUCTURE IN 5'S: A SYNTHESIS OF THE RESEARCH ON ORGANIZATION DESIGN," *Manag. Sci.*, vol. 26, no. 3, p. 322, 1980.

[3] L. Ostrowski, M. Helfert, and S. Xie, "A Conceptual Framework to Construct an Artefact for Meta-Abstract Design Knowledge in Design Science Research," *2012 45th Hawaii Int. Conf. Syst. Sci.*, pp. 4074–4081, 2012.

[4] M. Kolp, P. Giorgini, and J. Mylopoulos, "Multi-agent architectures as organizational structures," *Auton. Agent. Multi. Agent. Syst.*, vol. 13, no. 1, pp. 3–25, 2006.

[5] B. Innocenti, B. López, and J. Salvi, "A multi-agent architecture with cooperative fuzzy control for a mobile robot," *Robot. Auton. Syst. Robot. Auton. Syst. 50th Anniv. Artif. Intell. Campus Multidiscip. Percept. Intell.*, vol. 55, pp. 881–891, 2007.

[6] A. Farinelli, G. Grisetti, and L. Iocchi, "Design and implementation of modular software for programming mobile robots," *Int. J. Adv. Robot. Syst.*, vol. 3, no. 1, pp. 037–042, 2006.

[7] J.-A. Fernández-Madrigal, C. Galindo, J. González, E. Cruz-Martín, and A. Cruz-Martín, "A software engineering approach for the development of heterogeneous robotic applications," *Robot. Comput. Integr. Manuf.*, vol. 24, pp. 150–166, 2008.

[8] F. Caballero, L. Merino, P. Gil, I. Maza, and A. Ollero, "A probabilistic framework for entire WSN localization using a mobile robot," *Rob. Auton. Syst.*, vol. 56, no. 10, pp. 798–806, Oct. 2008.

[9] G. P. Sadrollah, J. C. Barca, A. I. Khan, J. Eliasson, and I. Senthooran, "A distributed framework for supporting 3D swarming applications," in *2014 International Conference on Computer and Information Sciences (ICCOINS)*, 2014, pp. 1–5.

[10] P. Shinners, "Pygame." [Online]. Available: http://pygame.org/.

[11] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. 2004.

[12] J. Kwon, J. H. Kim, and J. Seo, "Consensus-based obstacle avoidance for robotic swarm system with behavior-based control scheme," in *2014 14th International Conference on Control, Automation and Systems (ICCAS*

*2014)*, 2014, no. Iccas, pp. 751–755.

[13] A. Jha and M. Kumar, "Two wheels differential type odometry for mobile robots," *Proc. - 2014 3rd Int. Conf. Reliab. Infocom Technol. Optim. Trends Futur. Dir. ICRITO 2014*, 2015.

[14] R. Siegwart, I. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots, Second Edition*, Second. The MIT Press, 2011.

[15] C. Jung and W. Chung, "Accurate calibration of two wheel differential mobile robots by using experimental heading errors," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 4533–4538, 2012.

[16] C. Alippi and G. Vanini, "A RSSI-based and calibrated centrali zed localization technique for Wireless Sensor Networks," *Fourth Annu. IEEE Int. Conf. Pervasive Comput. Commun. Work.*, pp. 1–5, 2006.

[17] E. Menegatti *et al.*, "Autonomous discovery, localization and recognition of smart objects through WSN and image features," *2010 IEEE Globecom Work. GC'10*, pp. 1653–1657, 2010.

[18] N. Zhou, X. Zhao, and M. Tan, "RSSI-based mobile robot navigation in grid-pattern wireless sensor network," *2013 Chinese Autom. Congr.*, pp. 497–501, Nov. 2013.

[19] J. Xiong, Q. Qin, and K. Zeng, "A Distance Measurement Wireless Localization Correction Algorithm Based on RSSI," in *2014 Seventh International Symposium on Computational Intelligence and Design*, 2014, no. 2, pp. 276–278.

[20] J. Su, H. Huang, and C. Lee, "A simple and efficient diagonal maze-solver for micromouse contests and intelligent mobile robot education," in *The 26th Chinese Control and Decision Conference (2014 CCDC)*, 2014, pp. 2407–2411.

[21] G. Parodi, A. Raska, M. Izquierdo, and E. Volentini, "Robot para resolución de laberintos," in *2014 IEEE Biennial Congress of Argentina (ARGENCON)*, 2014, pp. 821–826.

[22] A. C. Jimenéz, J. P. Anzola, and S. Bolaños, "Decentralized Model for Autonomous Robotic Systems Based on Wireless Sensor Networks," *ARPN J. Eng. Appl. Sci.*, vol. 11, no. 19, pp. 11378–11382, 2016.

[23] H. John, D. Darren, F. Eric, and D. Michael, "Matplot." [Online]. Available: http://matplotlib.org/.

## INFORMACIÓN DE LOS AUTORES

**Andrés Camilo Jiménez:** Ingeniero Electrónico – Fundación Universitaria los Libertadores – Colombia. Magister en Ingeniería Electrónica – Pontificia Universidad Javeriana – Colombia.
E-mail: acjimeneza@libertadores.edu.co

**John Petearson Anzola:** Ingeniero Electrónico – Universidad Manuela Beltrán – Colombia. Magister en ciencias de la información y el conocimiento– Universidad Distrital Francisco José de Caldas – Colombia. E-mail: jpanzolaa@libertadores.edu.co

**Giovanny Mauricio Tarazona:** Ingeniero Industrial – Universidad Distrital Francisco José de Caldas – Colombia. Doctor en Sistemas Informáticos para Internet – Universidad de Oviedo – España. E-mail: gtarazona@udistrital.edu.co

**Sandro Javier Bolaños:** Ingeniero de Sistemas – Universidad Distrital Francisco José de Caldas – Colombia. Master en Teleinformática – Universidad Distrital Francisco José de Caldas – Colombia. Doctor en Informática – Universidad Pontifica de Salamanca – España. E-mail: sbolanos@udistrital.edu.co