Global Journal of Engineering and Technology Advances
eISSN: 2582-5003
Cross Ref DOI: 10.30574/gjeta
Journal homepage: https://gjeta.com/

(RESEARCH ARTICLE)

Check for updates

A Co-simulation framework using MATLAB and CoppeliaSim for path planning of nonholonomic mobile robots

Aggrey Shitsukane [1,*], Calvins Otieno [2], James Obuhuma [2], Lawrence Mukhongo [1] and Samuel Kariuki [1]

[1] *Department of Electricals Engineering, Technical University of Mombasa, Kenya.*
[2] *Department of computer science, Maseno University, Kenya.*

Global Journal of Engineering and Technology Advances, 2025, 23(01), 006-019

Publication history: Received on 26 February 2025; revised on 05 April 2025; accepted on 07 April 2025

Article DOI: https://doi.org/10.30574/gjeta.2025.23.1.0076

Abstract

Simulation plays a vital role in the design, testing, and validation of autonomous mobile robot navigation systems, particularly when real-world experimentation is constrained by cost, safety, or logistical limitations. This paper presents a modular and scalable co-simulation framework integrating MATLAB and CoppeliaSim for the path planning of nonholonomic wheeled mobile robots in static environments. The framework leverages MATLAB's computational and fuzzy logic capabilities for controller development, while CoppeliaSim provides a physics-based 3D simulation environment for modeling robot kinematics, sensing, and interaction with obstacles. Communication between the two platforms is achieved via the CoppeliaSim Remote API, enabling real-time data exchange for closed-loop control. The system supports dynamic sensor feedback, customizable fuzzy inference systems, and visual monitoring of robot behavior. To validate the framework, a case study involving a fuzzy logic controller for obstacle avoidance was conducted, with performance evaluated based on traversal time and path efficiency. Results demonstrate that the framework provides a reliable, flexible, and low-cost alternative to physical prototyping for early-stage development and testing of autonomous navigation algorithms.

Keywords: Co-Simulation Framework; Fuzzy Logic Controller; Path Planning; Nonholonomic Robot; Robot Navigation

1. Introduction

Autonomous mobile robots (AMRs) have gained significant traction across a range of industries, including manufacturing, logistics, agriculture, and military (Lee, 2021). Central to their deployment is the ability to navigate unknown or partially known environments autonomously and safely. This necessitates robust and reliable path planning and motion control algorithms, which must be rigorously tested and validated before deployment in real-world scenarios.

However, real-world testing of mobile robots presents numerous challenges (Choi et al., 2021). Physical experimentation is often constrained by cost, safety risks, environmental variability, hardware limitations, and the need for repeatability. These limitations make simulation environments a critical component in the development pipeline of robotic systems. Simulations enable controlled experimentation, rapid prototyping, repeatability, and early-stage performance benchmarking without the risk of hardware damage.

In recent years, several simulation platforms have emerged for robotic applications, including Gazebo, Webots, and CoppeliaSim (Farley et al., 2022). Among these, CoppeliaSim stands out for its real-time physics engine, multi-sensor support, and seamless integration with external programming environments. On the other hand, MATLAB offers a powerful environment for developing intelligent controllers, such as fuzzy logic systems, neural networks, and model-

* Corresponding author: Aggrey Shitsukane

based control algorithms (Krenicky et al., 2022). Despite their individual strengths, a combined simulation environment leveraging both tools can offer enhanced flexibility and capability particularly for researchers working on intelligent control-based navigation systems.

CoppeliaSim, formerly known as V-REP, was developed by Coppelia Robotics in Zurich, Switzerland, and is widely utilized in educational and research settings. Its highly adaptable architecture enables rapid development of both 2D and 3D simulations. The integrated development environment (IDE) is built on a programmable, network-based framework, allowing concurrent execution of scripts across multiple scene objects. CoppeliaSim comes equipped with a wide array of built-in examples, robot models, sensors, and actuators, enabling users to construct and interact with virtual environments in real time. It also supports the creation of custom models, making it ideal for tailored simulation experiments (Elhousry et al., 2024; Farley et al., 2022).

Notably, CoppeliaSim provides a user-friendly, flexible framework for designing unique robots with minimal or no coding required. Through intuitive drag-and-drop functionality, users can assemble parts, configure sensors and actuators, and define robot behaviors using graphical interfaces. This streamlined process was used to develop a custom 7-degree-of-freedom robotic arm, highlighting the platform's accessibility and versatility.

Figure 1 shows an image of a robotics simulation environment in CoppeliaSim. The scene includes various robotic models on a grid-patterned platform, demonstrating diverse robotic.

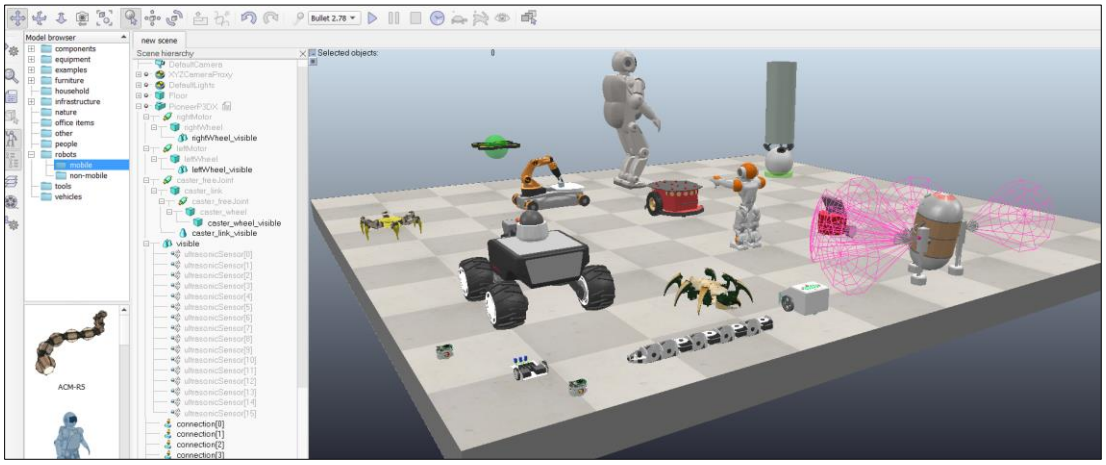

Figure 1 CoppeliaSim environment

This paper presents a co-simulation framework integrating MATLAB and CoppeliaSim to support the design, testing, and evaluation of fuzzy logic-based path planning algorithms for nonholonomic wheeled mobile robots. The framework connects MATLAB and CoppeliaSim through the Remote API, enabling real-time communication between the controller logic in MATLAB and the robot's simulated environment in CoppeliaSim. Sensor data such as distance to obstacles is captured in CoppeliaSim and passed to MATLAB, where the fuzzy logic controller processes the input and returns control commands to actuate the robot.

The proposed framework supports, Modular controller design using MATLAB's Fuzzy Logic Toolbox.Real-time, bidirectional data exchange via CoppeliaSim Remote API.High-fidelity robot dynamics and sensing via CoppeliaSim and Custom scenario creation and visualization for testing diverse navigation strategies. To validate the framework, a case study is conducted in which a fuzzy logic controller enables a robot to navigate through a static obstacle field. The controller's performance is evaluated based on traversal time, path smoothness, and collision avoidance efficiency.

The rest of this paper is organized as follows: Section 2 reviews related work on robotic simulation and co-simulation frameworks. Section 3 describes the overall system architecture and communication interface. Section 4 details the robot model and simulation setup. Section 5 presents the controller implementation in MATLAB. Section 6 describes the simulation workflow, followed by experimental results in Section 7. Finally, Section 8 concludes the paper and outlines directions for future research.

2. Related Work

Simulation plays an indispensable role in robotics research, providing a safe and efficient environment to design, test, and validate algorithms before transitioning to real-world deployment. Various platforms and toolchains have been developed to support the simulation of robot kinematics, sensor models, control systems, and environmental interactions (Choi et al., 2021). Each of these platforms comes with its own strengths and limitations, particularly when it comes to path planning and intelligent controller integration.

Gazebo, commonly paired with the Robot Operating System (ROS), is one of the most widely used robotic simulators. It offers realistic physics, plug-and-play sensor modules, and extensive ROS compatibility, making it suitable for full-stack robot development. However, Gazebo typically requires a steep learning curve, significant system resources, and is tightly coupled with ROS, which may limit flexibility for control strategies developed outside the ROS ecosystem (Peake et al., 2021).

Webots is another prominent simulator that provides an all-in-one solution for robot modeling, control, and visualization. It supports controller development in various languages such as Python, C, and MATLAB. However, real-time bidirectional communication with external software can be cumbersome, especially when testing advanced algorithms with rapid prototyping tools like MATLAB (Dobrokvashina et al., 2022).

In contrast, CoppeliaSim offers a flexible and extensible simulation environment with built-in support for real-time communication via Remote API and ZeroMQ. Its modular architecture allows researchers to integrate external controllers in languages such as Python, Java, Lua, and MATLAB, making it particularly well-suited for co-simulation frameworks. CoppeliaSim's scene editor, physics engine (ODE, Bullet, Vortex, Newton), and multi-sensor simulation capabilities make it a versatile tool for mobile robot applications (Elhousry et al., 2024).

Several studies have utilized CoppeliaSim for mobile robot simulation. For instance, (Reguii et al., 2023) implemented a neuro-fuzzy control strategy in CoppeliaSim for autonomous path tracking in unknown environments. Likewise, Ahmad Fauzi et al. (2021) developed a fuzzy logic controller using MATLAB for indoor robot navigation but used MATLAB's own 2D simulation environment, limiting the realism and complexity of the test environment.

Despite these advancements, there is a notable gap in the literature regarding structured co-simulation frameworks that tightly couple MATLAB's intelligent control capabilities such as fuzzy logic, neural networks, and optimization with CoppeliaSim's rich physical modeling and visualization tools. Most existing work either uses standalone simulations in one environment or manually integrates data flow between platforms, lacking a generalized or scalable framework (*Запоріжжя – 2024*, 2024).

This paper addresses this gap by proposing a co-simulation architecture that Seamlessly connects MATLAB controllers to CoppeliaSim's simulation environment via Remote API, enables real-time, closed-loop feedback between simulated sensors and external control logic and supports modular, reusable controller development for various robot platforms and scenarios. The framework is particularly aimed at researchers and developers seeking to evaluate intelligent, non-model-based navigation strategies such as fuzzy logic controllers within a realistic and extensible simulation environment.

3. System Architecture

The proposed simulation framework is designed to integrate MATLAB and CoppeliaSim in a real-time, closed-loop architecture for simulating and testing autonomous path planning algorithms. It separates the control logic from the physical simulation environment, enabling flexible experimentation and modular development of intelligent controllers such as fuzzy logic systems. The architecture consists of three core components CoppeliaSim Environment, MATLAB Controller Engine and Communication Interface (Remote API)

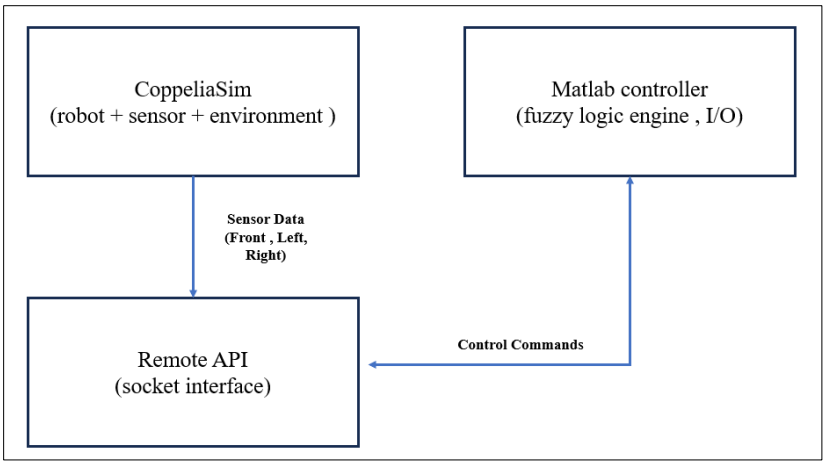

Figure 2 System architecture- Matlab Coppeliasim co-simulation

The figure 2. describes the integration architecture between MATLAB and CoppeliaSim using a Remote API (socket interface). CoppeliaSim acts as the simulation environment that contains the robot, sensors, and the simulated environment. It generates and sends sensor data (e.g., front, left, and right sensor readings) to the Remote API. The Remote API serves as the communication bridge, receiving sensor data from CoppeliaSim and relaying it to MATLAB.MATLAB controller, containing the fuzzy logic engine and input/output processing, receives the sensor data, performs computations based on fuzzy logic rules, and generates control commands as illustrated in figure 3. These Control Commands are sent back through the Remote API interface to CoppeliaSim to control the simulated robot's movements in real-time. This setup facilitates a closed-loop system allowing for real-time control and testing of robotic navigation algorithms using fuzzy logic controllers..

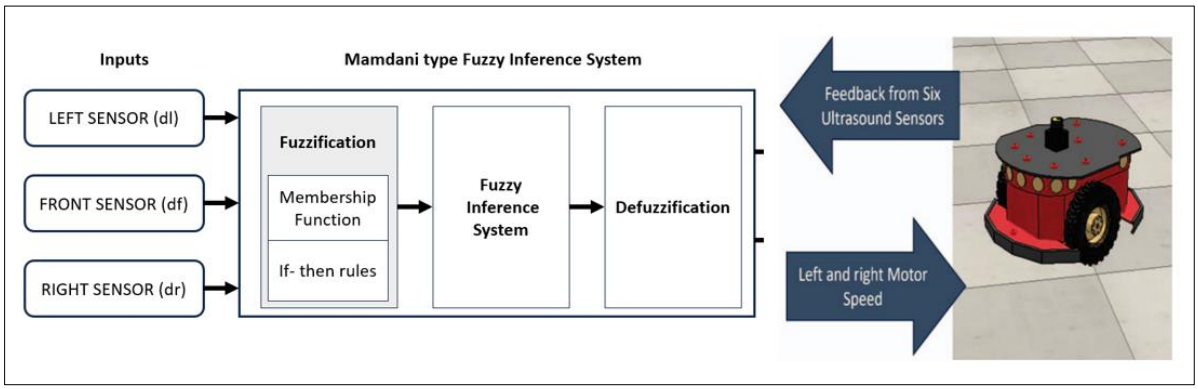

Figure 3 Matlab fuzzy logic model to control robot using CoppeliaSim

The remote API interface in CoppeliaSim offers interaction with AR environment or a simulation, controlled via an external entity through socket communication. It is comprised of remote API clients and remote API server services. The client side can be set in as a small footprint code in AR representing any hardware including real robots. It allows calling of remote functions and quick data streaming back and forth. On the client side, functions are called almost as regular functions, with two exceptions however: remote API functions accept an additional argument which is the operation mode and return the same error code. The operation mode offers calling the function as blocking and then wait the feedback from the server

3.1. CoppeliaSim Environment

CoppeliaSim serves as the primary simulation engine, providing A 3D simulated world with static obstacles, A nonholonomic wheeled mobile robot (e.g., differential drive Pioneer P3-DX), Simulated proximity sensors (e.g., front, left, right range sensors) and Physics-based motion and collision detection.

CoppeliaSim uses, Scenes: .ttt files that define environments like the one you uploaded, Lua-based embedded scripts for behaviour and interaction and Remote APIs / ZeroMQ / ROS2 to control robots from external code (Python, C++, MATLAB).

Scene Setup in CoppeliaSim, you Create Environment using Add Primitive shapes to place walls, floor, and cylindrical obstacles and add a proximity sensor, vision sensor, or LIDAR to the robot if required. Add a Mobile Robot (e.g., Pioneer, TurtleBot, or a custom model) then Add motors to wheels (Joint objects with revolute mode). Use naming conventions (leftMotor, rightMotor) to reference them in scripts or APIs. Control the Robot using eembedded Lua Scripts (inside CoppeliaSim) used for fast prototyping.

function sysCall_actuation()

local leftSpeed = 2

local rightSpeed = 2

sim.setJointTargetVelocity(leftMotor, leftSpeed)

sim.setJointTargetVelocity(rightMotor, rightSpeed)

end

Python (Remote API). Install pycoppeliasim or use legacy b0RemoteApi.

import sim # or use b0RemoteApi

sim.simxFinish(-1) # Close old connections

clientID = sim.simxStart('127.0.0.1', 19997, True, True, 5000, 5)

if clientID != -1:

print("Connected to CoppeliaSim")

Start simulation

sim.simxStartSimulation(clientID, sim.simx_opmode_blocking)

CoppeliaSim + MATLAB Integration. To control CoppeliaSim from MATLAB, you'll use the Remote API provided by CoppeliaSim. Requirements are that CoppeliaSim installed, MATLAB installed and CoppeliaSim's remoteApi files available in MATLAB's path. Copy These Files into MATLAB Project Folder from CoppeliaSim/programming/remoteApiBindings/matlab/matlab/.

Copy: remoteApi.m, remoteApiProto.m and remApi.dll (or .so for Linux/Mac)

Start CoppeliaSim Remote API Server in your CoppeliaSim scene Go to Script of any object (or create a new non-threaded child script).Add this Lua code to enable the remote API.

if (sim_call_type==sim_childscriptcall_initialization) then

simRemoteApi.start(19999)

end

Control the Robot from MATLAB using script example

disp('Program started');

```
sim=remApi('remoteApi'); % create remote API object
sim.simxFinish(-1); % just in case, close all opened connections
clientID=sim.simxStart('127.0.0.1',19999,true,true,5000,5); % connect to CoppeliaSim
if (clientID > -1)
disp('Connected to remote API server');
% Start the simulation
sim.simxStartSimulation(clientID, sim.simx_opmode_blocking);
% Get motor handles
[~, leftMotor] = sim.simxGetObjectHandle(clientID, 'leftMotor', sim.simx_opmode_blocking);
[~, rightMotor] = sim.simxGetObjectHandle(clientID, 'rightMotor', sim.simx_opmode_blocking);
% Set motor speeds
sim.simxSetJointTargetVelocity(clientID, leftMotor, 2.0, sim.simx_opmode_streaming);
sim.simxSetJointTargetVelocity(clientID, rightMotor, 2.0, sim.simx_opmode_streaming);
pause(5); % Let the robot move for 5 seconds
% Stop motors
sim.simxSetJointTargetVelocity(clientID, leftMotor, 0, sim.simx_opmode_streaming);
sim.simxSetJointTargetVelocity(clientID, rightMotor, 0, sim.simx_opmode_streaming);
% Stop simulation
sim.simxStopSimulation(clientID, sim.simx_opmode_blocking);
% Close connection
sim.simxFinish(clientID);
else
disp('Failed connecting to remote API server');
end
sim.delete(); % destroy the object
```

The robot is modelled using built-in primitives and is equipped with proximity sensors whose data are read at each simulation time step. These sensor values are critical for perception and are passed to the external MATLAB controller.

3.2. MATLAB Controller Engine

MATLAB hosts the fuzzy logic controller (FLC), which receives sensor data from CoppeliaSim and outputs motor commands (e.g., left and right wheel velocities). The controller is developed using the Fuzzy Logic Toolbox, allowing for rapid experimentation with different rule bases and membership functions.

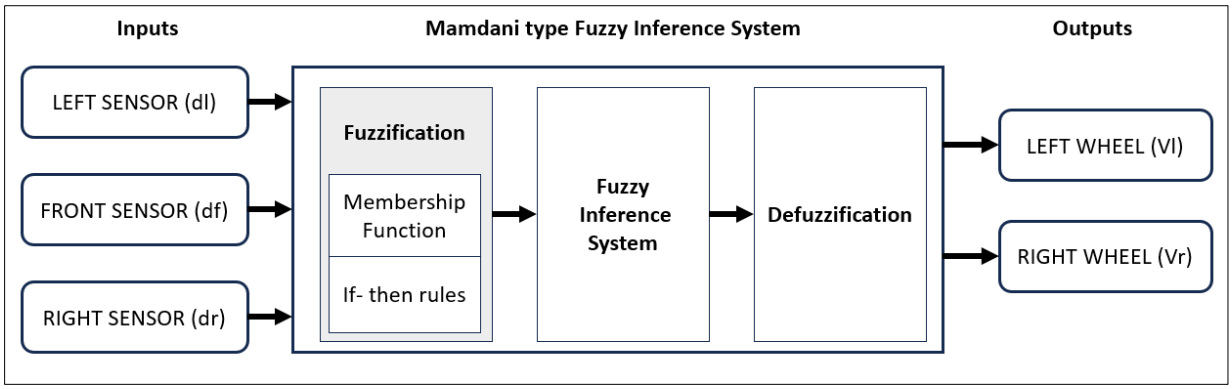

Figure 4 Fuzzy logic controller architecture (Shitsukane et al., 2018)

As in figure3.6, inference engine is partitioned into four segments. The fuzzifier handles the task of gauging input variables (input signals), conducting scale mapping, and carrying out fuzzification. It entails the conversion of calculated signals (crisp values) into fuzzy values, which are also referred to as "linguistic variables". Membership functions (MFs) are utilized for this transformation. The membership function, ranging from 0 to 1, represents the extent to which something belongs to a particular fuzzy set. If it is absolutely certain that the quantity belongs to the fuzzy set, its value is 1; conversely, if it is certain that it does not belong to the set, its value is 0 (Iancu, 2012).

The inference engine serves as the core processing unit of the Fuzzy Logic Controller (FLC), mimicking human decision-making by using fuzzy principles to deduce fuzzy control actions based on consequences and rules. Initially, input parameters are converted to matching linguistic variables. Then, the Mamdani engine evaluates a set of "if-then rules" using the linguistic values of these variables. These results are subsequently converted into a precise output value for the FLC. Following this transformation, a defuzzifier carries out a secondary operation, which encompasses both scale mapping and defuzzification. This process completes the cycle of decision-making. This separation allows MATLAB to handle computational tasks, algorithm tuning, and data logging independently of the simulator. The fuzzy controller uses a rule base of IF–THEN statements that map sensory conditions to steering actions.

3.2.1. Example Rules

- IF Front is **Near** AND Left is **Far** AND Right is **Near** → THEN Turn Left
- IF Front is **Near** AND Left is **Near** AND Right is **Far** → THEN Turn Right
- IF Front is **Far** AND Left is **Far** AND Right is **Far** → THEN Go Straight
- IF Front is **Near** AND Left is **Near** AND Right is **Near** → THEN Turn Left

The rule base is intuitive and interpretable, making it easy to adjust based on observed behavior. All combinations are evaluated using the min–max inference method. A closed-loop simulation is achieved using CoppeliaSim's Remote API, enabling continuous data exchange between the simulated environment and the controller logic in MATLAB.

3.2.2. Real-time control loop

- Sensor Reading in CoppeliaSim transmits current front, left, and right distance sensor values.
- Fuzzification where MATLAB converts numeric sensor inputs into fuzzy linguistic variables.
- All fuzzy rules are evaluated in parallel using fuzzy logic inference.
- Defuzzification is done, output (steering correction $\Delta\theta$) is obtained using the centroid method.
- Wheel Velocity Calculation, $\Delta\theta$ is mapped to left and right wheel speeds.
- Command Transmission for Wheel speeds are sent back to CoppeliaSim.
- CoppeliaSim executes a new time step, updating the robot's position and sensor readings.
- The process repeats until the robot reaches its goal or a maximum time threshold is exceeded.

The loop runs in synchronous mode to align simulation time with MATLAB's control loop. Execution speed depends on sensor polling rate and simulation step size (typically ~50 ms).

4. Robot and Environment Modelling

The co-simulation framework leverages CoppeliaSim to model the robot's kinematics, sensor system, and navigation environment. This section details the modeling of the nonholonomic mobile robot, the static obstacle environment, and the simulation of sensors used for perception.

The robot used in this study is modeled as a nonholonomic differential drive mobile robot, which reflects common physical platforms like the Pioneer P3-DX. Nonholonomic systems have constraints on their motion specifically, they cannot move laterally. This poses a unique challenge for path planning and makes the use of intelligent controllers particularly relevant.

In the Kinematic and dynamic model L, R, and C denotes the track width, radius of the wheels, and center of the mass of a mobile robot, respectively. The point P is located between the centers of the driving wheels axis. The point d is the distance between the points P and C. The landmark (O, X, Y) shows the field navigation environment, and (O, x, y) is the moving axis of the mobile robot. The θ is the turning angle, which represents the orientation of the robot about an axis (O, X). The three parameters (x, y, θ) describe the initial posture of the mobile robot.

The linear and angular velocities of the robot can be expressed in terms of the wheel velocities as in equations 1 and 2:

$$\omega = \frac{V_r - V_l}{L} \quad \text{.....} \quad 1$$

$$V = \frac{V_r + V_l}{2} \quad \text{.....} \quad 2$$

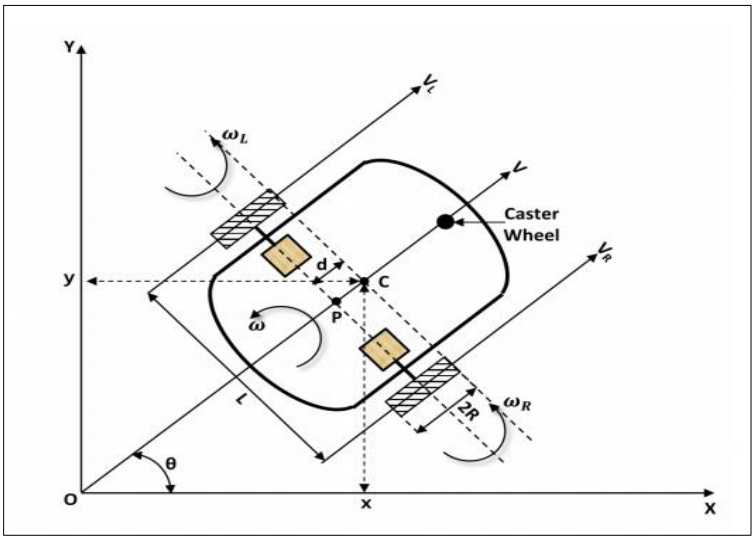

Figure 5 Kinematic and dynamic model (Palacín et al., 2023)

The mobile robot's drive wheels will conform to a nonholonomic constraint, ensuring that they roll without slipping. A differential drive robot typically has two wheels placed on either side of the robot's chassis. The wheels will be driven independently by separate motors, allowing for both forward motion and rotation. The key parameters are:

- Vr: The linear velocity of the right wheel.
- Vl: The linear velocity of the left wheel.
- L: The distance between the centres of the two wheels (wheelbase).
- V: The forward linear velocity of the robot.
- ω: The angular velocity of the robot.

The state of the robot is described by its position (x,y) and orientation θ. The kinematic equations relate the time derivatives of these state variables to the robot's velocities.

The forward velocity V is split into its x and y components based on the robot's orientation θ shown in equations 3 ,4 and 5

$$\frac{dx}{dt} = \dot{x} = V.cos\theta \ldots\ldots\ldots \quad 3$$

$$\frac{dy}{dt} = \dot{y} = V.sin\theta \ldots\ldots\ldots\ldots \quad 4$$

The rate of change of the orientation θ is given by the angular velocity ω:

$$\frac{d\theta}{dt} = \dot{\theta} = \omega \ldots\ldots\ldots \quad 5$$

Substituting v and ω from the wheel velocities into the equations for $\dot{x}\ \dot{y}\ \dot{\theta}$ results in equations 6,7 and 8.

$$\dot{x} = \left(\frac{V_r + V_l}{2}\right) cos\ (\theta) \ldots\ldots\ldots \quad 6$$

$$\dot{y} = \left(\frac{V_r + V_l}{2}\right) sin\ (\theta) \ldots\ldots\ldots\ldots \quad 7$$

$$\dot{\theta} = \left(\frac{V_r - V_l}{L}\right) \ldots\ldots\ldots\ldots\ldots \quad 8$$

Representing these equations in matrix form we get equation 9 showing how the linear velocities of the right and left wheels (Vr and Vl) influence the rates of change of the robot's position $\dot{x}$ and $\dot{y}$ and its orientation $\dot{\theta}$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{cos\theta}{2} & \frac{cos\theta}{2} \\ \frac{sin\theta}{2} & \frac{sin\theta}{2} \\ \frac{1}{L} & -\frac{1}{L} \end{bmatrix} \begin{bmatrix} V_r \\ V_l \end{bmatrix} \quad 9$$

Additional details regarding its application is in the analysis by Nurmaini & Chusniah (2017). which offers solution for conducting this experimental simulation.

In this context, $V_r\ and\ V_l$ represent the linear velocities of the right and left wheels, serving as the motion commands for navigating the mobile robot. In CoppeliaSim, the robot's motion is governed by applying velocity commands to each wheel joint. These velocities are calculated externally by the MATLAB fuzzy logic controller based on sensor inputs. The test environment is a bounded 3D plane populated with various static obstacles. The layout is designed to challenge the robot's path planning and obstacle avoidance capabilities.

Key features of the environment include Static obstacles that has walls, blocks, and circular pillars placed in varying configurations.Start and goal positions Defined within the scene and kept constant across all tests for consistency and Navigation complexity Includes narrow passages, dead-ends, and cluttered zones to test the robustness of the controller.The obstacle course is built using CoppeliaSim's drag-and-drop scene editor and can be easily modified for different experimental scenarios.

The robot is equipped with a simulated sensor array in CoppeliaSim to detect obstacles and perceive the environment. The inputs to the Fuzzy Logic Controller (FLC) will come from ultrasonic sensors mounted on the robot's left, middle, and right sides of the chassis:

- Left Sensor to Measures distances to obstacles on the left side.
- Middle Sensor to Measures frontal distances to obstacles.
- Right Sensor to Measures distances to obstacles on the right side.

The mobile robot use ultrasonic proximity sensors labeled d0 to d7 on its front side, as shown in figure 3.8, to identify obstacles in the environment. These sensors will facilitate collision avoidance and navigation by providing information

regarding the detection of objects and their distances. There will be eight sensors in total, collectively scanning a 180-degree field for potential obstacles. Every sensor will have the ability to identify objects within a distance of up to 1000 millimeters. When an obstacle is identified within the range of any sensor, the robot will calculate the distance between itself and the obstacle by integrating data from all the sensors.

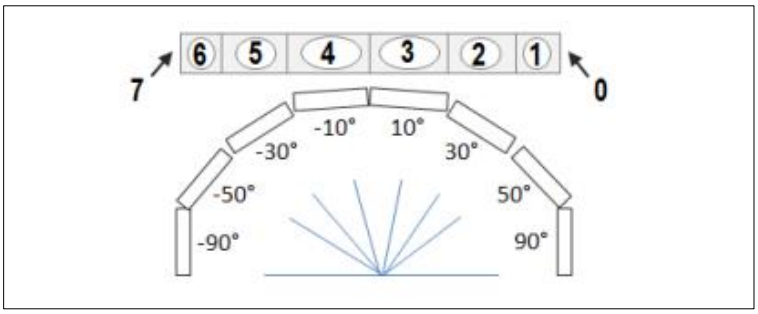

Figure 6 Sensor arrangement

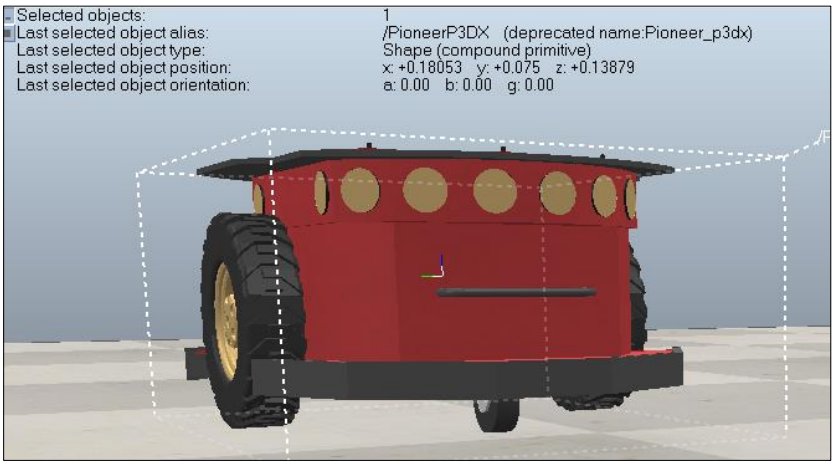

Figure 7 Pioneer test robot

When one or more sonar sensors malfunction, it can have various implications for the robot's navigation capabilities. If a single sonar sensor becomes faulty, it generally leads to a minor reduction in the robot's object detection capacity. In such cases, the malfunctioning sensor can potentially be compensated for by an adjacent functional sensor. However, a more challenging issue arises when a sonar sensor provides data that is technically functional but inaccurate due to external factors, such as terrain conditions affecting the sensor's readings (Doran et al., 2017).By combining realistic sensor models with an obstacle-rich environment and nonholonomic dynamics, the framework provides a controlled yet challenging testbed for evaluating autonomous navigation strategies.

5. Simulation Workflow

The simulation is designed to support real-time, closed-loop communication between MATLAB and CoppeliaSim using the Remote API in synchronous mode, ensuring one-to-one correspondence between control decisions and simulation steps. This guarantees determinism and consistency across multiple trials.

5.1. Initialization

Prior to executing the simulation loop, both environments must be properly initialized:

Load the prepared scene containing the robot, sensor models, obstacles, and target location. Start simulation in remote synchronous mode to await external control signals.

Establish a Remote API connection to CoppeliaSim. Initialize global parameters including sensor indices, joint handles, simulation step size, controller parameters. Load or generate the fuzzy logic inference system (FIS) structure. Reset

logging variables and simulation counters. MATLAB pings CoppeliaSim to ensure connectivity and readiness. Time synchronization is activated using the synchronous trigger mechanism.

5.2. Loop Execution

Once initialized, the controller enters a real-time loop that continues until a goal condition is reached or a maximum number of steps is exceeded.

- Read Sensor Data from CoppeliaSim, Retrieve normalized distance values from front, left, and right sensors via Remote API.
- Fuzzification in MATLAB to Convert raw distance readings into fuzzy linguistic terms using the FIS.
- Rule Evaluation and Inference: Apply fuzzy rules and aggregate results to derive an output fuzzy set.
- Defuzzification, Convert the fuzzy output (steering angle correction) into a crisp value using the centroid method.
- Wheel Velocity Calculation by Map the output into differential wheel speeds suitable for the differential drive model.
- Send Commands to CoppeliaSim Transmit computed left and right wheel velocities to the robot's joint actuators.
- Trigger Simulation Step, Call `simxSynchronousTrigger()` to advance the simulation by one step.
- Log Data by Record sensor readings, output commands, timestamps, and robot position for post-simulation analysis.
- Check Termination Conditions If robot reaches the goal zone or exceeds step/time limits, terminate the loop.

6. Case Studies

To evaluate the effectiveness of the proposed MATLAB CoppeliaSim simulation framework, a case study was conducted involving a fuzzy logic-controlled nonholonomic robot navigating a static environment with randomly placed obstacles. The goal of the experiment was to assess the controller's ability to guide the robot safely and efficiently from a start point to a predefined goal zone using real-time sensor feedback. The robot was placed at a starting location on a 3D plane within CoppeliaSim simulated environment. Several static obstacles of varying sizes and shapes were placed between the start and goal to create a realistic challenge for path planning and reactive control.

Environment Characteristics is Map size: 5 m × 6 m, Number of obstacles12 cylindrical, Sensor coverage: ~180° (front, left, right) and Goal radius: 0.3 m zone at opposite corner. Robot Configuration is Differential drive with left and right wheel motors, Max speed: ±0.3 m/s and Sensor range: 0.5 m. The simulation was run in synchronous mode with a timestep of 50 ms (0.05 s) per cycle. The robot navigates from a start position to a target, collecting data based on the time taken to complete the mission.

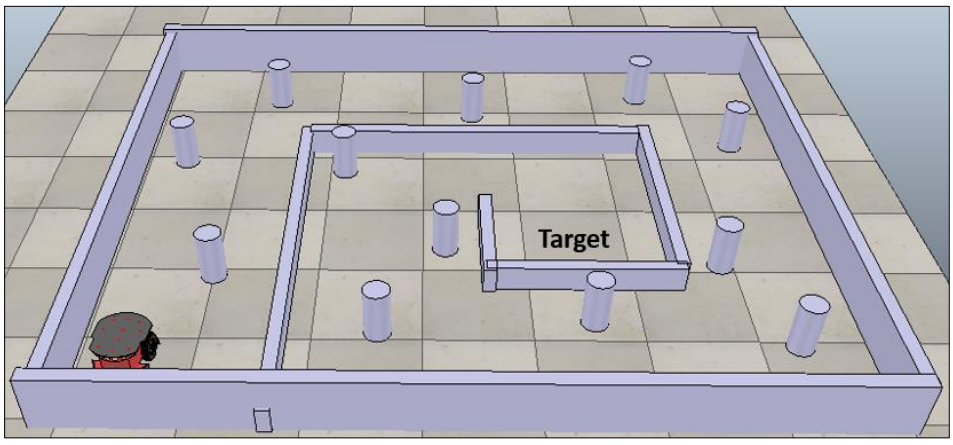

Figure 8 Simulation environment

The controller was initialized in MATLAB using a Mamdani fuzzy inference system with triangular membership functions. The robot began at the start location and operated autonomously using only real-time sensor data and fuzzy control logic. The simulation ran until the robot Reached the goal or Collided with an obstacle. 10 repeated runs with identical environment and Each run used the same FIS and sensor threshold configuration. Three key performance

indicators (KPIs) were tracked during each trial: Traversal Time (s Time taken to reach the goal, Path Smoothness Qualitative assessment of robot motion (zigzagging and Collision Status Binary flag (0 = success, 1 = collision)

7. Results

The robot successfully reached the goal in 90% of trials. The path trajectories were smooth and adaptive, showing real-time steering decisions when encountering nearby obstacles.

Table 1 Simulation trial results

Metric	Value
Mean Traversal Time	87.6 seconds
Successful Runs	9 out of 10
Mean Steering Changes	132 commands

In the one failed trial, the robot became trapped in a concave obstacle region due to limitations in sensor range and rule coverage.

8. Discussion

The proposed framework distinctly decouples the controller developed in MATLAB from the robot and simulation environment implemented in CoppeliaSim. Such separation provides substantial flexibility, enabling easy modifications to controller logic independently of the simulation setup.

Key advantages identified include efficient real-time performance due to offloading demanding physics calculations to CoppeliaSim, facilitating stable and deterministic execution even on moderate hardware. Moreover, CoppeliaSim's precise physics-based simulations, accurate sensor modeling, and robust collision detection enhance result credibility, bridging the simulation-to-hardware gap effectively. The Remote API ensures broad integration capabilities, allowing future incorporation of various programming languages, optimization techniques, and machine learning models. Furthermore, comprehensive monitoring and intuitive debugging capabilities through MATLAB visualization tools and CoppeliaSim's 3D visualizations significantly improve experimental analysis and educational utility.

However, several limitations were recognized. Manual creation of fuzzy logic rules and membership functions can be labor-intensive and non-optimal in complex situations. Limited sensor configurations restrict the robot's perceptual capabilities, particularly in complicated obstacle scenarios. Current simulations address only static environments, highlighting a need for dynamic obstacle management and multi-robot interaction capabilities. Transitioning to physical hardware introduces additional complexities including sensor noise, delays, and uncertainties absent in simulation.

Future research will target these limitations through AI-driven fuzzy controller optimization, dynamic environment simulation, richer sensor integration (e.g., lidar, stereo vision, RGB-D), and physical robot implementations, further solidifying the framework's practical relevance.

Table 2 Simulation framework comparison (Tseeva et al., 2024)(Farley et al., 2022)

Feature	CoppeliaSim + MATLAB	Gazebo + ROS	Webots
Controller Language	MATLAB, Python, etc.	C++/Python (ROS nodes)	C/C++, Python, MATLAB
Visualization	Real-time 3D + scripting	ROS RViz + Gazebo GUI	Integrated 3D GUI
Integration Ease	Simple via Remote API	Requires ROS launch files	Built-in controllers
Learning Curve	Moderate	Steep	Low to moderate
Real-Time Control	Synchronous, step-wise	Multi-threaded ROS	Event-driven loop
Custom Logic	Easy with MATLAB Toolbox	Complex ROS node writing	Moderate

While Gazebo excels in realism and offers robust integration within complete ROS-based robotic workflows, it typically presents a steep learning curve and substantial hardware demands. Conversely, Webots provides a user-friendly, integrated environment suited to beginners, yet it lacks flexibility when implementing custom algorithms developed externally. The MATLAB–CoppeliaSim framework strikes a balanced compromise, especially suited for algorithm-centric research in fuzzy logic, control theory, and AI-driven studies. It allows researchers to concentrate primarily on control logic development without the complexities associated with extensive simulator infrastructure.

9. Conclusion and Future Work

This paper presented a robust and modular co-simulation framework that effectively integrates MATLAB's fuzzy logic capabilities with CoppeliaSim, enabling real-time simulation and evaluation of autonomous mobile robot navigation strategies in static obstacle environments. The developed architecture leverages CoppeliaSim's Remote API for seamless interaction with a Mamdani-type fuzzy logic controller, demonstrating precise closed-loop navigation control with differential wheel drive and proximity sensors. A comprehensive case study validated the framework's performance, highlighting high success rates, interpretability, computational efficiency, and realistic simulation compared to alternatives such as Gazebo and Webots.

Future work will transition this simulation framework into physical robot platforms like TurtleBot or Pioneer 3-DX using ROS or Simulink Real-Time, assessing controller robustness in real-world scenarios with sensor noise and actuation limitations. Additionally, incorporating AI/ML methodologies, including reinforcement learning for adaptive control, genetic algorithms, particle swarm optimization, and hybrid neuro-fuzzy systems for automated rule tuning, will enhance flexibility and performance. Expanding the simulator to manage dynamic obstacles will facilitate research in real-time path re-planning and adaptive control in realistic settings. Lastly, integrating advanced perception sensors such as simulated lidar, stereo vision, or RGB-D cameras will further improve environmental perception, SLAM capabilities, and terrain-aware navigation.

Compliance with ethical standards

Disclosure of conflict of interest

The authors declare that they have no affiliations with or involvement in any organization or entity with any financial interest in the subject matter or materials discussed in this manuscript.

References

[1] Ahmad Fauzi, F., Mulyana, E., Mardiati, R., & Eko Setiawan, A. (2021). Fuzzy Logic Control for Avoiding Static Obstacle in Autonomous Vehicle Robot. Proceeding of 2021 7th International Conference on Wireless and Telematics, ICWT 2021. https://doi.org/10.1109/ICWT52862.2021.9678436

[2] Choi, H. S., Crump, C., Duriez, C., Elmquist, A., Hager, G., Han, D., Hearl, F., Hodgins, J., Jain, A., Leve, F., Li, C., Meier, F., Negrut, D., Righetti, L., Rodriguez, A., Tan, J., & Trinkle, J. (2021). On the use of simulation in robotics: Opportunities, challenges, and suggestions formoving forward. Proceedings of the National Academy of Sciences of the United States of America, 118(1), 1–9. https://doi.org/10.1073/pnas.1907856118

[3] Dobrokvashina, A., Lavrenov, R., Magid, E., Bai, Y., Svinin, M., & Meshcheryakov, R. (2022). Servosila Engineer Crawler Robot Modelling in Webots Simulator. International Journal of Mechanical Engineering and Robotics Research, 11(6), 417–421. https://doi.org/10.18178/ijmerr.11.6.417-421

[4] Doran, M., Sterritt, R., & Wilkie, G. (2017). Autonomic Sonar Sensor Fault Manager for Mobile Robots. International Journal of Computer and Information Engineering, 11(5), 621–628. http://waset.org/publications/10007407

[5] Elhousry, Y., Yasser, O., Ismail, S. M., & Ammar, H. H. (2024). Implementation of an Automated Catering Service Using UR3 Robotic Arms in CoppeliaSim V-REP. 2024 International Conference on Machine Intelligence and Smart Innovation, ICMISI 2024 - Proceedings, 210–213. https://doi.org/10.1109/ICMISI61517.2024.10580859

[6] Farley, A., Wang, J., & Marshall, J. A. (2022). How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion. Simulation Modelling Practice and Theory, 120. https://doi.org/10.1016/j.simpat.2022.102629

[7] Iancu, I. (2012). A Mamdani Type Fuzzy Logic Controller. Fuzzy Logic - Controls, Concepts, Theories and Applications. https://doi.org/10.5772/36321

[8] Krenicky, T., Nikitin, Y., & Božek, P. (2022). Model-Based Design of Induction Motor Control System in MATLAB. Applied Sciences (Switzerland), 12(23). https://doi.org/10.3390/app122311957

[9] Lee, I. (2021). Service robots: A systematic literature review. Electronics (Switzerland), 10(21). https://doi.org/10.3390/electronics10212658

[10] Nurmaini, S., & Chusniah, C. (2017). Differential drive mobile robot control using variable fuzzy universe of discourse. ICECOS 2017 - Proceeding of 2017 International Conference on Electrical Engineering and Computer Science: Sustaining the Cultural Heritage Toward the Smart Environment for Better Future, 50–55. https://doi.org/10.1109/ICECOS.2017.8167165

[11] Palacín, J., Rubies, E., Bitrià, R., & Clotet, E. (2023). Non-Parametric Calibration of the Inverse Kinematic Matrix of a Three-Wheeled Omnidirectional Mobile Robot Based on Genetic Algorithms. Applied Sciences (Switzerland), 13(2). https://doi.org/10.3390/app13021053

[12] Peake, I., La Delfa, J., Bejarano, R., & Blech, J. O. (2021). Simulation Components in Gazebo. Proceedings of the IEEE International Conference on Industrial Technology, 2021-March, 1169–1175. https://doi.org/10.1109/ICIT46573.2021.9453594

[13] Reguii, I., Hassani, I., & Rekik, C. (2023). Neuro-fuzzy Control of a Mobile Robot. December, 45–50. https://doi.org/10.1109/sta56120.2022.10018999

[14] Shitsukane, A., Cheruiyot, W., Otieno, C., & Mvurya, M. (2018). Fuzzy Logic Sensor Fusion for Obstacle Avoidance Mobile Robot. 2018 IST-Africa Week Conference (IST-Africa), Page 1 of 8-Page 8 of 8.

[15] Tseeva, F. M., Shogenova, M. M., Senov, K. M., Liana, K. V., & Bozieva, A. M. (2024). Comparative Analysis of Two Simulation Environments for Robots, Gazebo, and CoppeliaSim in the Context of Their Use for Teaching Students a Course in Robotic Systems Modeling. Proceedings of the 2024 International Conference “Quality Management, Transport and Information Security, Information Technologies”, QM and TIS and IT 2024, 186–189. https://doi.org/10.1109/QMTISIT63393.2024.10762908

[16] Запоріжжя – 2024. (2024).