IAES International Journal of Robotics and Automation (IJRA)
Vol. 15, No. 4, December 2026, pp. 883∼901
ISSN: 2722-2586, DOI: 10.11591/ijra.v15i4.pp883-901

Automatic instruction generation for mobile robot in CoppeliaSim from flowcharts for programming learning

Junior Gomez-Contreras, Jharold Alonso Mayorga Villena, Andrea López Condori, Yuber Velazco-Paredes, Roxana Flores-Quispe
Universidad Nacional de San Agustín de Arequipa, Arequipa, Perú

Article Info

Article history:

Received Nov 10, 2026
Revised Aug 17, 2026
Accepted Feb 13, 2026

Keywords:

Computer vision
CoppeliaSim
Educational robotics
Flowchart recognition
Hand-drawn diagrams
Mobile robotics
Optical character recognition
Programming education

ABSTRACT

Flowcharts are widely used in programming and robotics education to represent the logical structure of algorithms, especially for beginners. However, manually translating these diagrams into code is often tedious and error-prone for learners, especially when the task involves instructions to control a mobile robot. While visual tools can support engagement and learning, the complexity of programming-language syntax continues to create barriers to developing algorithmic reasoning. Therefore, this paper presents a system based on computer vision and machine learning techniques to automatically convert hand-drawn linear flowcharts—composed of predefined movement commands, including single-state loops—into executable instructions for the CoppeliaSim simulator to control the Pioneer P3DX mobile robot, representing a significant advancement in programming education. The system was validated through flowchart recognition, kinematic accuracy analysis, and path-efficiency metrics across controlled experimental scenarios with varying environmental complexity. Under these evaluated conditions, the results showed consistent execution performance ($CV < 21\%$) and a high correlation between instruction count and execution time ($r > 0.985$). The recognition pipeline achieved an accuracy rate of 95.8%, and the ablation study identified adaptive thresholding as the most influential preprocessing component for robustness. In addition, an evaluation with 12 computer science students showed that the system reduced manual coding time by 77% compared with Python programming, while also lowering perceived task difficulty. The mobile robot reached the target in all tested navigation scenarios. These findings support the potential use of the proposed system in introductory robotics education, although its current scope remains limited to the flowchart structures and simulation conditions evaluated in this study.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Junior Gomez-Contreras
Universidad Nacional de San Agustín de Arequipa
Arequipa, Perú
Email: jgomezcon@unsa.edu.pe

1. INTRODUCTION

Robotics has become an area of great importance in recent decades, thanks to the development of numerous robotic tools and simulators for industry, education, and other fields. In educational contexts, robotics fosters the development of technical skills, such as programming and robot design, and transversal skills, such as critical thinking, creativity, and teamwork [1]. A systematic review of educational robotics simulators highlights that these tools improve engagement by reducing costs and increasing accessibility, allowing students

to practice skills such as algorithm design in controlled environments [2]. Furthermore, advances in computer science have enabled the development of skills and concepts that help people solve problems, design systems, and understand human behavior [3], [4], supporting their integration into multiple disciplines [5]. Nevertheless, learning to program remains difficult for beginners [6], [7]. Robot programming has evolved from manual coding of motion scripts, which requires syntactic and low-level control knowledge, toward visual programming environments that reduce syntax barriers [8]. In parallel, flowcharts have been widely used to represent algorithmic control flow before implementation [9], a context in which handwriting can further support logical organization and focus during early learning [10]. However, hand-drawn flowcharts are still difficult for machines to interpret automatically [11], so learners often retype the same logic into executable robot code.

On the other hand, several robotic simulators have been developed to support robotics programming and training, offering virtual environments without the need for physical hardware. For example, Gazebo, integrated with the robot operating system (ROS), is a realistic physics simulation environment that supports multi-robot scenarios [12]. Webots offers an intuitive interface with cross-platform support and extensive sensor models [13]. MORSE focuses on modular simulations for inquiry-oriented learning, while CoppeliaSim stands out for its accurate motion simulation, robust integration with the Python API, and educational features such as ease of scripting and visualization, making it ideal for beginners learning programming through robotics [14]. However, these tools require users to learn specific development environments. There are currently no tools capable of automating the process of recognizing movement and directional instructions from hand-drawn patterns and converting them into executable control commands for mobile robots. Such automation would substantially reduce the cognitive and technical load associated with traditional programming for novice users. Moreover, the variability of human handwriting in critical elements (such as return arrows in loops) prevents conventional programming methods from directly interpreting the control logic. This leads to a reliance on manual code transcription, eliminating the fluency necessary for mission prototyping.

Existing tools typically focus either on flowchart design and recognition or on robot simulation, but rarely integrate both for automatic instruction generation. Flowchart-recognition approaches mainly detect diagram symbols and structures [15], [16], whereas robotic simulation approaches usually require control instructions to be written through programming interfaces [17], [18]. As a result, the automatic conversion of hand-drawn flowcharts into executable mobile-robot instructions within a single educational framework remains limited.

To address this gap, this study develops a computer vision and machine learning framework that translates linear hand-drawn flowcharts into executable movement instructions for a Pioneer P3DX robot in CoppeliaSim. The framework is intended to reduce programming barriers by allowing learners to design robot behaviors on paper and immediately observe the resulting execution in simulation, without requiring conventional programming code. To evaluate the proposed approach, this study addresses the following research questions:

a. Better recognition and translation accuracy of linear hand-drawn flowcharts into executable movement sequences for the Pioneer P3DX mobile robot within the CoppeliaSim environment,
b. Performance enhancement resulting from optimized computer vision preprocessing techniques in recognizing linear hand-drawn flowcharts, which eventually improves overall system reliability, and
c. Minimize the corresponding generation latency and instruction execution errors, which usually affect the robotic navigation process.

The framework follows five stages: image preprocessing, element analysis, output and export, reading and preprocessing of the TXT instruction file, and robot command execution in CoppeliaSim. Its feasibility is evaluated through recognition accuracy, preprocessing ablation, kinematic and navigation metrics, and student testing of the proposed tool. The results demonstrate that linear hand-drawn flowcharts can be reconstructed into ordered movement sequences and executed by the Pioneer P3DX in CoppeliaSim under controlled conditions. The proposed approach also provides a simple way for beginners to connect visual algorithm design with robot programming.

Recent advances in flowchart recognition and educational robotics have explored multiple approaches to bridge visual algorithm design with program execution. For the recognition of hand-drawn flowcharts, Schäfer *et al.* [15] proposed arrow R-CNN, a deep learning system extending Faster R-CNN with arrow keypoint prediction and diagram-aware post-processing, improving offline recognition of scanned flowcharts substantially. Earlier, Chen *et al.* [16] addressed online handwritten flowchart recognition using logical structure

and graph grammar with random-forest classification.

In the educational context, Zhang *et al.* [19] demonstrated that progressive flowchart-development scaffolding significantly improves university students' computational thinking and programming self-efficacy compared with non-progressive approaches. Andrzejewska *et al.* [20] used eye-tracking to compare the cognitive strategies used by learners when analysing algorithms expressed as flowcharts versus pseudocode, finding measurable advantages of structured flowcharts.

Regarding robotic simulation for educational purposes, Rani and Chauhan [17] presented a Python-controlled robot simulation in V-REP, illustrating how scriptable simulation environments can support robot-configuration and behavior-testing workflows. Shitsukane *et al.* [18] developed a MATLAB-CoppeliaSim co-simulation framework for nonholonomic mobile-robot path planning with fuzzy-logic obstacle avoidance.

While these works demonstrate significant advances in flowchart recognition, diagram-based programming education, and robotic simulation, there remains a gap in systems that integrate hand-drawn flowchart interpretation with direct mobile robot control inside an educational simulator. Most existing solutions focus either on digital flowchart processing or require complex programming knowledge from the learner. The present work addresses this gap by proposing an end-to-end system that automatically translates linear hand-drawn flowcharts, including a single-state loop, into executable mobile robot movements within CoppeliaSim, providing an accessible tool for teaching programming concepts through robotics.

Although existing visual programming environments reduce the syntactic difficulty of conventional programming, they require users to construct programs directly within a digital interface. Scratch provides a block-based environment designed primarily for introductory programming [21], while Blockly is a framework for creating customized block-based programming languages [22]. LabVIEW uses graphical dataflow diagrams for measurement, automation, and control applications [23]. Unlike these environments, the proposed system accepts paper-based hand-drawn flowcharts, recognizes their symbols, connections, and commands through computer vision and OCR, and automatically generates movement instructions for a mobile robot in CoppeliaSim. Table 1 summarizes this distinction.

Table 1. Novelty comparison with visual programming tools (evaluating input modalities, recognition capabilities, and execution environments; existing tools rely on digital blocks or dataflow, whereas the proposed method handles physical hand-drawn flowcharts)

Tool	Input	Automatic recognition	Robot execution
Scratch [21]	Digital blocks	No	Extension dependent
Blockly [22]	Digital blocks	No	Implementation dependent
LabVIEW [23]	Digital dataflow	No	Hardware dependent
Proposed method	Hand-drawn flowchart	Vision and OCR	CoppeliaSim

The proposed approach connects paper-based algorithm design with mobile-robot execution by recognizing the flowchart, reconstructing its structure, interpreting its commands, and executing them on the robot. The current framework is limited to linear flowcharts, predefined movement commands, and a single-state loop.

2. PROPOSED METHOD

Figure 1 summarizes the proposed five-stage framework: i) image preprocessing, ii) element analysis, iii) output and export, iv) reading and preprocessing of the TXT instruction file, and v) robot command execution in CoppeliaSim. Together, these stages transform a hand-drawn flowchart into structured movement instructions that are subsequently executed by the Pioneer P3DX mobile robot.

2.1. Flowcharts dataset

A new dataset of 120 linear hand-drawn flowcharts with movement instructions, including a single-state loop, for mobile robots was created. Each flowchart follows a linear structure with approximately 4 to 10 sequential movement instructions for the tests. To ensure visual consistency and facilitate processing, specific design features were considered, as summarized in Table 2. Figure 2 shows examples with 4 (Figure 2(a)) and 7 instructions (Figure 2(b)). Loops are identified by repetition numbers located on the right in each block, which replace consecutive identical blocks.

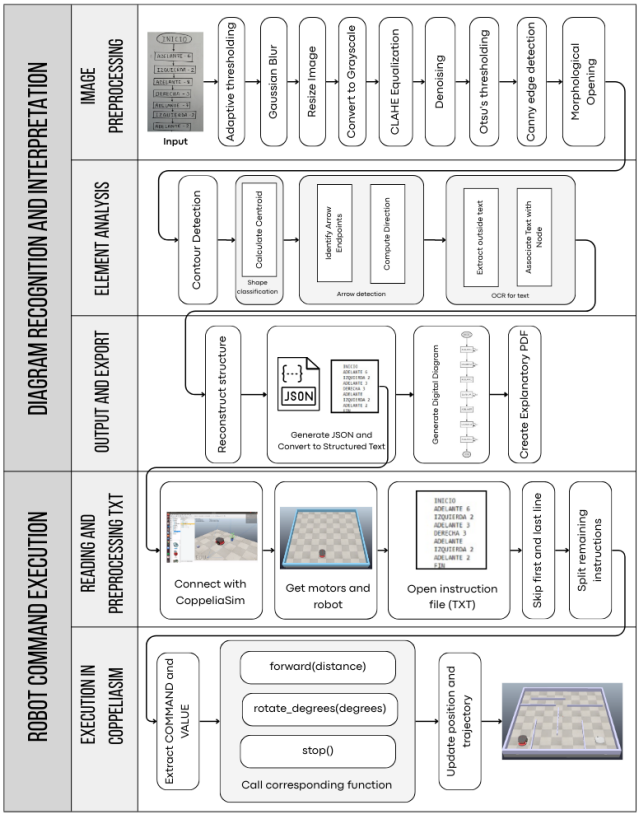
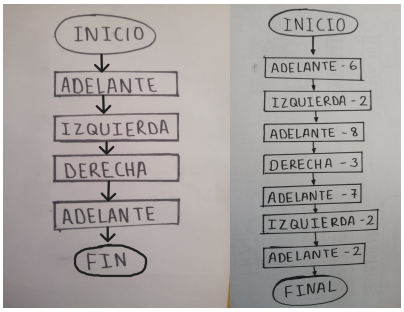


Figure 1. Proposed method for the automatic generation of movement instructions for mobile robots from flowchart recognition

Table 2. Characteristics of the proposed hand-drawn flowchart dataset

Feature	Description
Diagram structure	Linear (sequential execution)
Instructions per diagram	Between 4 and 10
Start/End shape	Rounded (oval shape)
Instruction shape	Rectangular (square or rectangle)
Drawing style	Hand-drawn with pen or pencil
Image format	Scanned (PNG or JPEG)
Total diagrams	120



(a) (b)

Figure 2. Representative hand-drawn flowcharts from the proposed dataset (a) simple instructions and (b) single-state loop instructions

2.2. Flowcharts recognition

This section describes how instruction blocks are recognized and converted from hand-drawn flowcharts into data that can be processed by the system. The process produces a JSON file containing the structure of each block and a text file used for robot execution, as shown in Figure 3.

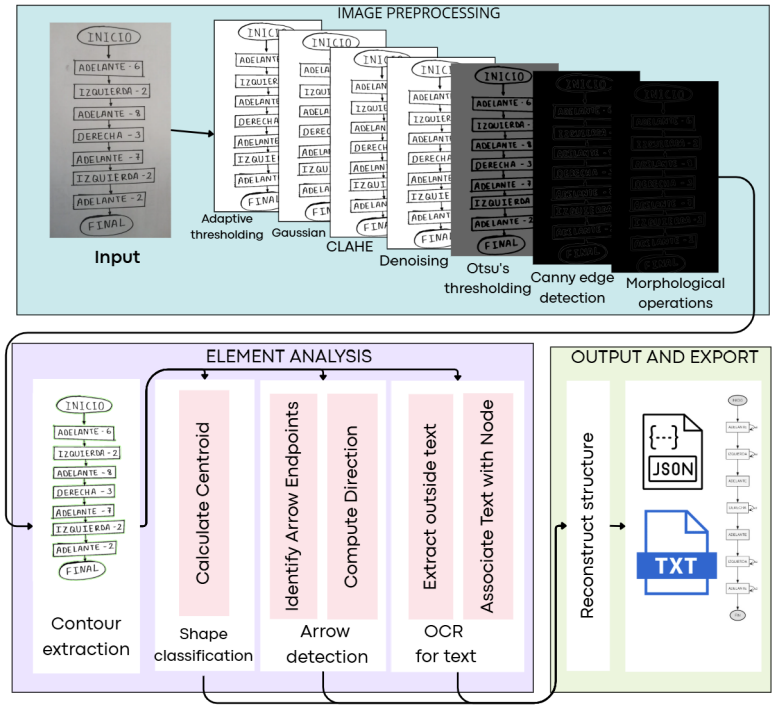


Figure 3. Proposed process to convert hand-drawn flowcharts into structured text and digital diagrams

2.2.1. Processing flowchart image

Hand-drawn flowcharts require preprocessing to ensure quality input [24], which includes eight stages:

a. Adaptive thresholding: Converts images to binary format using local thresholds based on mean intensity. For pixel (x, y), threshold $T(x, y)$ is computed as in equation 1, where $\mu(x, y)$ is local mean intensity in a 111×111 block and $C = 48$ [25], [26].

$$T(x, y) = \mu(x, y) + C \tag{1}$$

b. Resizing: To improve the accuracy of boundary detection, the images are scaled by a factor 2.0 using bilinear interpolation [27], with coordinates remapped using ratio 0.5.

c. Grayscale conversion: The image is converted to grayscale from the image in the RGB channel using (2) [28].

$$I = 0.299R + 0.587G + 0.114B \tag{2}$$

d. CLAHE: contrast-limited adaptive histogram equalization was applied to enhance local contrast and improve contour detection. The image was divided into 4×4 tiles with a clip limit of 4.0 [29], [30]. Figure 4 shows the resulting image.

e. Noise reduction: non-local means (NLM) was applied to reduce noise while preserving edges and structural details. The parameters were filter strength $h = 270$, template window 7×7, and search window 21×21 [31].

f. Gaussian blur & Otsu's thresholding: A Gaussian kernel 25×25 followed by Otsu's automatic thresholding [32] was used to convert the image to a clean binary format with an automatically selected threshold. The left side of Figure 5 shows the binary image obtained.

g. Edge detection (Canny): To highlight object boundaries and significant transitions, the binary image was processed through the Canny edge detector with thresholds of 0 and 255 and an aperture size of 3.

h. Morphological opening: Finally, a morphological opening operation using a 3 × 3 kernel is applied to remove small noise and artifacts while preserving structural contours. This operation consists of erosion followed by dilation.



Figure 4. Result of using the CLAHE method on a grayscale image



Figure 5. Binary image using Gaussian blur and Otsu's thresholding with contours detected using cv2.findContours

2.2.2. Component detection and classification

Once the diagram image has been preprocessed, it is now ready to detect and identify instruction types and their components, such as nodes, arrows, and text.

a. Contour detection: The outer contours of the flowchart elements, including nodes, arrows, and text regions, are detected using `cv2.findContours()` with `RETR_EXTERNAL` and `CHAIN_APPROX_SIMPLE`. The detected contours are shown in green on the right side of Figure 5.

b. Shape detection: Contours are classified using three geometric features: (1) number of vertices, where the Douglas–Peucker algorithm determines the number of vertices of each contour (3 = triangle, 4 = rectangle/square, and > 4 = circle); (2) compactness (area-perimeter ratio), where $K = \text{area}/\text{perimeter}$ separates arrows ($K < 30$) from nodes ($K \geq 30$); and (3) moments, which are used for centroid calculation and spatial relationship analysis. The process is shown in Algorithm 1 and Figure 6.

Algorithm 1. Shape classification

Input: Contour C, image I
Output: Shape label S
Compute area A, perimeter P, and polygonal approximation vertices V of C
Calculate compactness $K = A/P$
if $K < 30$ then
    $S \leftarrow$ 'arrow'
else if $|V| = 3$ then
    $S \leftarrow$ 'triangle'
else if $|V| = 4$ then
    $S \leftarrow$ 'square'
else
    $S \leftarrow$ 'circle'
end if
Return S

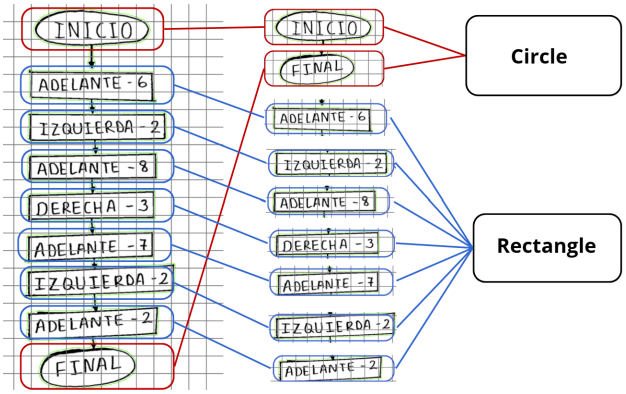


Figure 6. Identified shapes with labels assigned by contour analysis. Each shape is annotated with its type (triangle, rectangle, arrow)

2.2.3. Text recognition

Optical character recognition (OCR) and text region isolation were used to recognize block content with Tesseract configured for Spanish (lang='spa') on contour-based regions.

a. Text region isolation: Cropped regions use 25-pixel padding for nodes and 5 pixels for external text.
b. Heuristic rules for text association: External text is associated with nodes based on proximity, searching adjacent contours in ascending and descending order until a text region is found.

2.2.4. Flowchart structure reconstruction

The flowchart was reconstructed from the identified nodes using Algorithm 2, following three steps.

Algorithm 2. Arrow detection and connection

Input: Edge image I_{edge}, contours C, nodes N
Output: List of arrows A with connections
Initialize empty list A
for all contour $c \in C$ do
    if c is classified as arrow then
        Extract start point P_s and end point P_e
        Compute distances to nodes in N
        Assign P_s to source node
        Assign P_e to destination node
        Add arrow (P_s, P_e) to A
    end if
end for
Return A

a. Identifying arrows: Arrows are identified by a low area-perimeter ratio ($K < 30$), while their direction is determined by comparing the number of contour points near the start and end points of the bounding rectangle.
b. Distance calculation: Distances between arrow endpoints and node boundaries are calculated using the perpendicular distance from a point to a line segment, as expressed in (3).

$$d(P, P_s, P_e) = \frac{|(P_e - P_s) \times (P - P_s)|}{\|P_e - P_s\|} \tag{3}$$

c. Linking nodes: Connections are established by linking arrows to the closest nodes based on endpoint distances.

2.2.5. Semantic processing

The flowchart conversion process is carried out in six steps:

a. Preprocess the image: Apply the image-processing pipeline to the hand-drawn flowchart.
b. Detect components: Detect and classify nodes, arrows, and text regions.
c. Parse the flowchart: Use OCR and spatial relationships to reconstruct the execution order from "INICIO" to "FIN".
d. Build the JSON representation: Convert movement labels into Spanish command tokens, including repetition factors, and store them with the node information and connections in the JSON IR (Figure 7).
e. Generate the instruction file: Export the ordered commands as "COMMAND n" (*e.g.*, "ADELANTE 2", "IZQUIERDA 1") in a plain-text file (Figure 8).
f. Execute the commands: The external Python controller maps each command to its corresponding CoppeliaSim motion routine.


```
{"Node": [
  {"Id": 0, "Name": "INICIO\n", "Position": "Inside", "
      Shape": "circle", "Line": "ADELANTE-2\n"},
  {"Id": 1, "Name": "ADELANTE-2\n", "Position": "Inside",
       "Shape": "square", "Line": "IZQUIERDA-1\n"},
  {"Id": 2, "Name": "IZQUIERDA-1\n", "Position": "Inside
      ", "Shape": "square", "Line": "ADELANTE-6\n"},
  {"Id": 3, "Name": "ADELANTE-6\n", "Position": "Inside",
       "Shape": "square", "Line": "IZQUIERDA-1\n"},
  {"Id": 4, "Name": "IZQUIERDA-1\n", "Position": "Inside
      ", "Shape": "square", "Line": "ADELANTE-2\n"},
  {"Id": 5, "Name": "ADELANTE-2\n", "Position": "Inside",
       "Shape": "square", "Line": "DERECHA-1\n"},
  {"Id": 6, "Name": "DERECHA-1\n", "Position": "Inside",
      "Shape": "square", "Line": "ADELANTE-1\n"},
  {"Id": 7, "Name": "ADELANTE-1\n\nFIN\n", "Position": "
      Inside", "Shape": "circle", "Line": ""}
]}
```


Figure 7. Architecture of JSON file


```
INICIO
ADELANTE 2
IZQUIERDA 1
ADELANTE 6
IZQUIERDA 1
ADELANTE 2
DERECHA 1
ADELANTE 1
FIN
```


Figure 8. Text output file

2.3. Building the scenario in CoppeliaSim

The simulation environment was developed in CoppeliaSim to evaluate the proposed approach under controlled and reproducible conditions. As shown in Figure 9, the scene includes the Pioneer P3DX robot, its control script, boundary walls, static obstacles, and a target, along with the corresponding 3D workspace.

The Pioneer P3DX was used to execute the movement commands generated from the flowcharts. The robot operates inside a rectangular workspace enclosed by four static cuboid walls. Additional cuboid obstacles with different positions and sizes were included to create different navigation conditions. The simulation environment was configured as shown in Figure 8, including the boundary wall dimensions (Figure 10(a)) and the activation of collision and detection properties for the walls and obstacles (Figure 10(b)).

A base scenario containing the robot and boundary walls was duplicated to create eleven scenarios: one reference environment without obstacles and ten environments with different obstacle configurations. Only the positions of the robot, target, and obstacles were changed, while the environment structure and physical properties remained the same. The robot started from the same position and orientation in all scenarios, and the simulation was reset before each execution to avoid positional drift. Figure 11 shows the different scenarios.

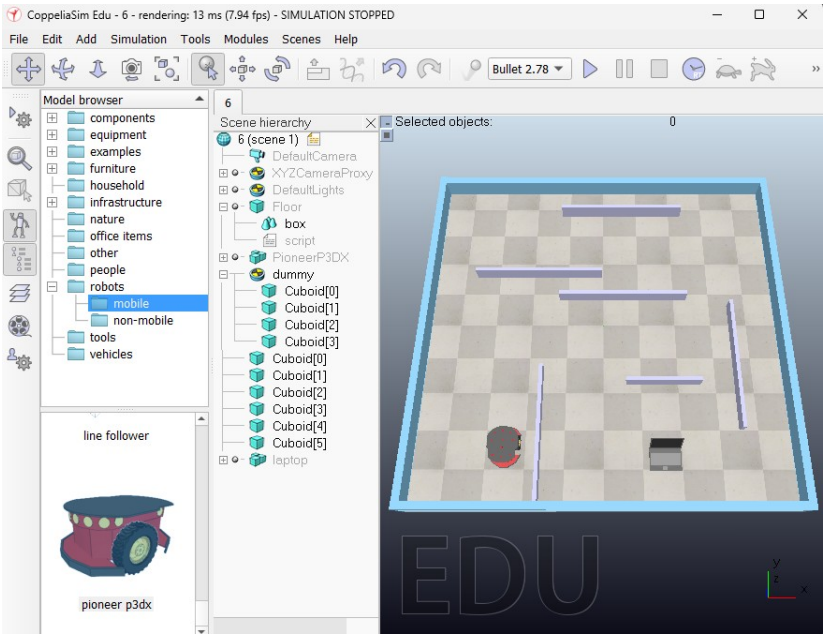
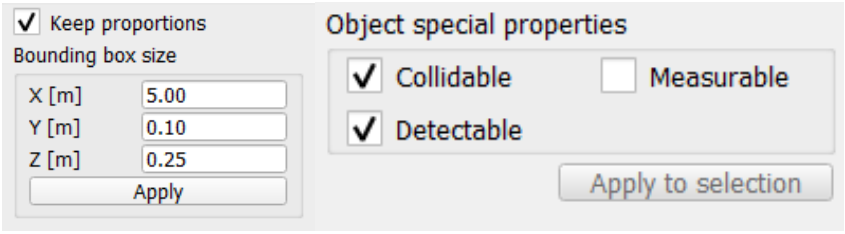


Figure 9. Scenario construction in CoppeliaSim



(a) (b)

Figure 10. Simulation environment configuration (a) configuration of boundary wall dimensions and (b) activation of the collidable and detectable options for boundary wall and obstacles

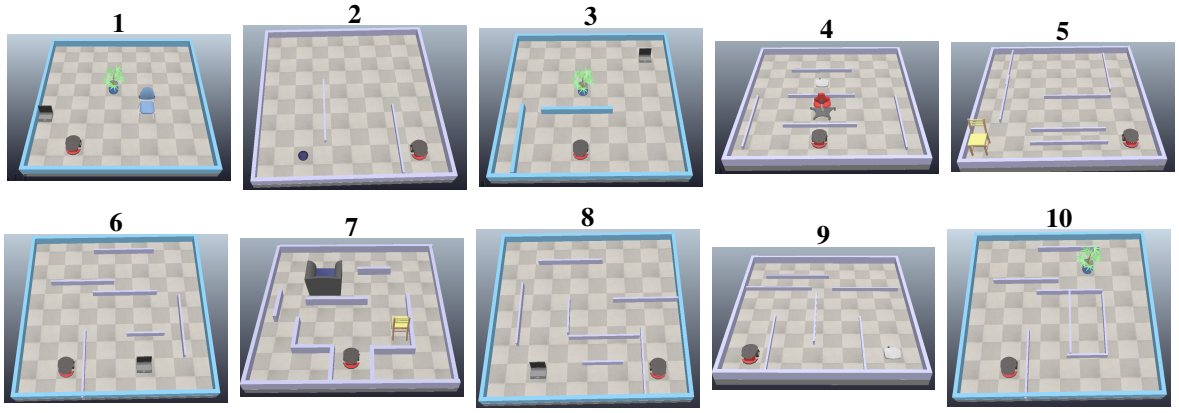


Figure 11. Ten obstacle courses in CoppeliaSim with different complexity levels

Before the experiments, each scenario was tested using basic forward and rotation commands to verify the robot's behavior under the defined boundary and collision conditions. These tests also confirmed that the environment remained stable during execution.

2.4. Mobile robot control in CoppeliaSim

Communication with CoppeliaSim is established through the Remote API over TCP/IP, as illustrated in Figure 12. The external Python controller reads the generated instruction file, obtains the required robot and motor handles, and receives the robot's position and orientation data. It then uses this information to execute the motion routine associated with each command.

Figure 13 shows the main part of the connection and initialization process. The `simxStart` function establishes the TCP/IP connection and returns the client identifier used for subsequent remote API calls.

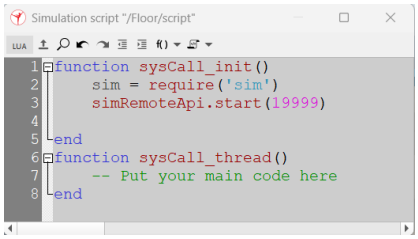


Figure 12. Remote API connection script in CoppeliaSim

```
# Connect to CoppeliaSim Remote API
clientID = sim.simxStart('127.0.0.1', 19999, True, True, 5000, 5)
# Retrieve robot handles
_, left_motor_handle = sim.simxGetObjectHandle(clientID, 'Pioneer_p3dx_leftMotor', sim.
    simx_opmode_oneshot_wait)
_, right_motor_handle = sim.simxGetObjectHandle(clientID, 'Pioneer_p3dx_rightMotor', sim.
    simx_opmode_oneshot_wait)
_, pioneer_handle = sim.simxGetObjectHandle(clientID, 'Pioneer_p3dx', sim.simx_opmode_oneshot_wait)
# Initialize position and orientation streaming
sim.simxGetObjectPosition(clientID, pioneer_handle, -1, sim.simx_opmode_streaming)
sim.simxGetObjectOrientation(clientID, pioneer_handle, -1, sim.simx_opmode_streaming)
#Constants
SQUARE_SIZE = 0.48
TURN_DEGREES = 90
```

Figure 13. Connection and handle initialization in external Python script

Position and orientation data are continuously obtained using `simxGetObjectPosition` and `simxGetObjectOrientation` in streaming mode. This feedback is used by `forward()` and `rotate_degrees()` to track linear and angular movement during execution.

Two constants define the movement increments: `SQUARE_SIZE = 0.48m` for each linear step and `TURN_DEGREES = 90°` for rotations. These values keep the robot's movements consistent with the workspace and the flowchart instructions.

The control layer includes `STOP()`, which stops both wheels, `FORWARD(d)`, which moves the robot forward, and `ROTATE_DEGREES(angle)`, which performs the specified rotation. These functions convert the generated commands into motor actions, as summarized in Algorithm 3.

Algorithm 3. Linear and rotational motion

```
procedure FORWARD(d)
    Read initial position
    Set wheel speeds (±1.4 depending on d)
    while distance < |d| do
        Read current position
    end while
    Stop motors
end procedure
procedure ROTATE_DEGREES(angle)
    Read initial orientation
    Calculate the opposite wheel speeds (±0.4)
    while rotation < target angle do
        Read current orientation
    end while
    Stop motors
end procedure
```

Finally, the `map_instructions(command, count)` function converts each text command into the corresponding CoppeliaSim instruction, as shown in Algorithm 4. It supports forward, rotation, and stop commands.

Algorithm 4. Instruction mapping

```
procedure MAP_INSTRUCTIONS(command, count)
    if command = "ADELANTE" then
        for i = 1 to count do
            forward(SQUARE_SIZE)
        end for
    else if command = "IZQUIERDA" then
        for i = 1 to count do
            rotate_degrees(-TURN_DEGREES)
        end for
    else if command = "DERECHA" then
        for i = 1 to count do
            rotate_degrees(TURN_DEGREES)
        end for
    else if command = "DETENER" then
        stop()
    else
        Print "Unknown instruction"
    end if
end procedure
```

3. RESULTS AND DISCUSSION

The experimental evaluation was carried out in three stages: i) validation of the correct execution of the generated instructions by the Pioneer P3DX in CoppeliaSim, ii) evaluation of the impact of each process applied in hand-drawn flowchart recognition, and iii) evaluation of the system by a group of students.

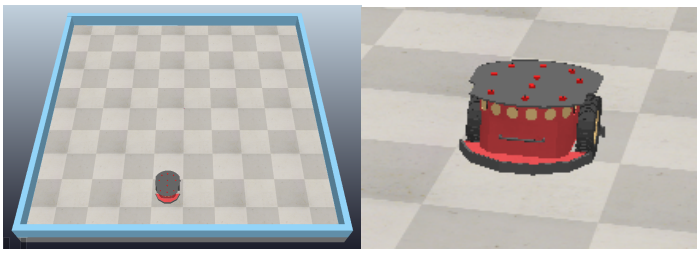
3.1. Evaluation of the correct execution of the generated instruction

The experimental evaluation aimed to validate the correct execution of movement instructions automatically generated from a flowchart in a controlled simulation environment. Two groups of scenarios were considered. Empty scenarios were used to verify the consistency of instruction interpretation and execution, while obstacle scenarios evaluated navigation robustness under increasing environmental complexity. This assessment covered two levels: flowchart recognition and structuring, and robotic execution of the generated commands. For each scenario, the instruction file generated from the recognized flowchart was executed in CoppeliaSim using the external Python controller. Performance was evaluated using three metrics: i) number of executed instructions, ii) total execution time, and iii) success rate in reaching the goal.

3.1.1. Evaluation in empty scenarios

Ten instruction files containing between 7 and 11 commands were evaluated in an environment without obstacles using the dashboard shown in Figure 14(a). The mobile robot shown in Figure 14(b) successfully executed all instructions, demonstrating accurate control and reliable instruction interpretation. To assess consistency and reliability, a statistical analysis was conducted, whose results are reported in Table 3, where the number of instructions (M=9.30, SD=1.89) and execution time (M=8.87 s, SD=1.74 s) were evaluated. The coefficient of variation was below 21% in both cases, indicating high consistency. These results suggest that the mobile robot shows a reasonably consistent relationship between the number of instructions and execution time, and that performance in an empty scenario is predictable within an acceptable range.

A Pearson correlation was also calculated, which confirms a strong positive relationship between instruction count and execution time ($r = 0.98$) with a significance level of $p < 0.001$, indicating predictable scaling with complexity. The linear model Time $= 0.92 \times$ Instructions $- 0.68$ ($R^2 = 0.96$) shows an average cost of 0.92 time units per instruction and explains 96% of the execution time variance. This confirms a stable and scalable system with strong engineering qualities in terms of predictability.



(a) (b)

Figure 14. Experimental setup for obstacle-free evaluation (a) empty CoppeliaSim scenario and (b) Pioneer P3DX mobile robot used in the experiments

Table 3. Statistical analysis of empty scenario results

Metric	Instructions	Time (s)
Mean (μ)	9.30	8.87
Std. Deviation (σ)	1.89	1.74
Coefficient of Variation (CV)	20.3%	19.6%

3.1.2. Evaluation in obstacle scenarios

The system was evaluated in ten obstacle environments of varying complexity, shown in Figure 11. For each scenario, the flowchart recognition module generated a movement instruction file; Figure 8 shows a representative sequence. Figure 15 illustrates the step-by-step execution of scenario 5 by the Pioneer P3DX. This medium-complexity task required 15 movement steps and 20.61 seconds to complete, reaching the target without collisions.

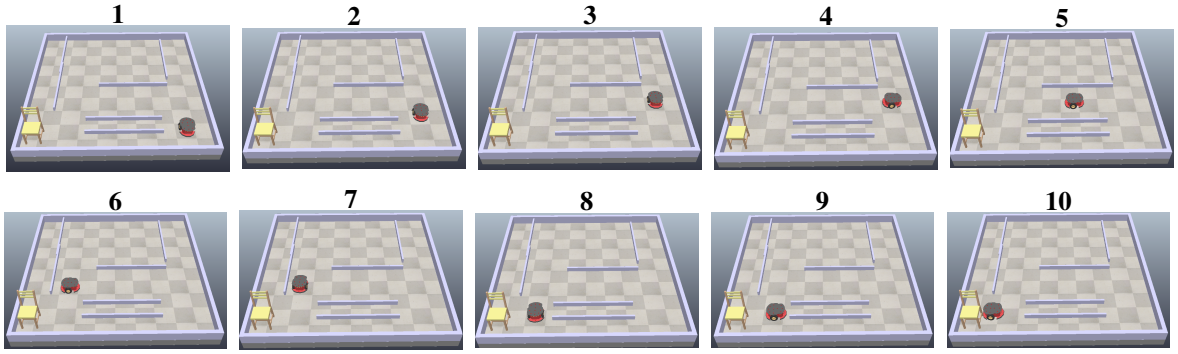


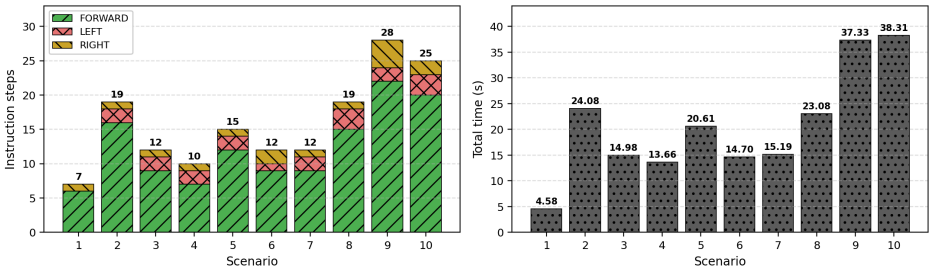
Figure 15. Execution of scenario 5 for the movement instructions shown in Figure 8

All remaining scenarios reached the target without collisions, maintaining a 100% success rate. Table 4 summarizes the results, showing that higher-complexity environments required more movements and longer execution times.

Table 4. Evaluation summary in scenarios with obstacles

ID	Cmplx.	Position Initial	Position Target	Instruct.	Time (s)	Success
1	Low	x: -1.269, y: -1.750, z: +0.138	x: -1.925, y: -1.224, z: +0.827	7	4.580	Yes
2	High	x: +1.730, y: -1.725, z: +0.138	x: -1.250, y: -1.800, z: +0.005	19	24.077	Yes
3	Medium	x: -0.244, y: -1.725, z: +0.138	x: -0.225, y: +0.175, z: +0.165	12	14.981	Yes
4	Low	x: -0.219, y: -1.750, z: +0.138	x: -0.174, y: -0.616, z: +0.460	10	13.659	Yes
5	Medium	x: +1.780, y: -1.725, z: +0.138	x: -2.075, y: -1.925, z: +0.450	15	20.609	Yes
6	Low	x: -1.244, y: -1.725, z: +0.138	x: +0.675, y: -1.924, z: +0.827	12	14.704	Yes
7	Medium	x: +0.005, y: +1.750, z: +0.138	x: +0.825, y: -0.827, z: +0.350	12	15.194	Yes
8	High	x: +1.730, y: -1.725, z: +0.138	x: -1.050, y: -1.949, z: +0.827	19	23.078	Yes
9	High	x: -1.769, y: -1.825, z: +0.138	x: +1.525, y: -2.425, z: +0.816	28	37.334	Yes
10	High	x: -1.244, y: -1.725, z: +0.138	x: +0.800, y: +1.200, z: +0.165	25	38.305	Yes

Figure 16 summarizes instruction composition and execution time across scenarios. Figure 16(a) shows the distribution of FORWARD, LEFT, and RIGHT commands, while Figure 16(b) presents total execution time. Simpler scenarios (1–4) mainly required FORWARD commands, whereas more complex scenarios (5–10) involved more turning maneuvers to avoid obstacles.



(a) (b)

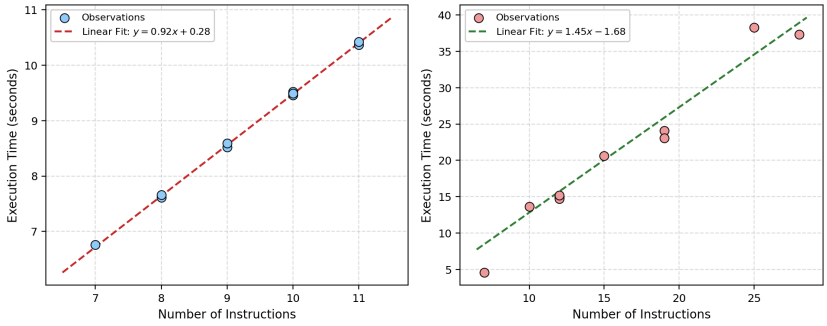
Figure 16. Instruction composition and execution time per obstacle scenario (a) instruction steps by command type and (b) total execution time

Table 5 presents the results grouped by complexity level. As complexity increases, the number of instructions and execution time also increase, although the relationship is not perfectly proportional. At the Medium and High levels, variability remains within 17-25%, which is close to that observed in the empty scenario. These results show that the system maintains consistent robot behavior across environments with different obstacle configurations, despite the higher variability observed at lower complexity levels.

Figure 17 shows a strong linear relationship between the number of instructions and execution time. Figure 17(a) shows the correlation in scenarios without obstacles, where the near-perfect relationship ($r = 0.998$, $R^2 = 0.997$) confirms that execution time increases linearly with the instruction sequence generated when there are no obstacles. Figure 17(b) presents the correlation in the scenario with obstacles, where the relationship remains high ($r = 0.985$, $R^2 = 0.969$), indicating that the framework maintains consistent and predictable performance even when additional maneuvers are required. This predictable scalability supports the validity and reproducibility of the proposed approach at different levels of complexity.

Table 5. Statistical analysis by complexity level

Metric	Low	Medium	High
Mean Time (s)	10.98	16.93	30.70
Standard deviation (s)	4.96	2.91	7.67
Mean Instructions	9.67	13.00	22.75
CV (%)	45.2%	17.2%	25.0%



(a) (b)

Figure 17. Correlation analysis between instructions and execution time (a) empty scenarios ($r = 0.998$, $R^2 = 0.997$) and (b) obstacle scenarios ($r = 0.985$, $R^2 = 0.969$)

In addition, other performance navigation metrics were calculated (see Table 6). A position error of 4.2±1.8 cm after 28 instructions demonstrates high precision. Path efficiency (0.87±0.09) shows near-optimal trajectories with 13% overhead. Processing time (3.2±0.7 s/diagram) confirms real-time applicability. Despite an OCR character error rate of 8.3%, text normalization ensured a 100% success rate.

Table 6. Performance metrics for obstacle navigation

Metric	Value
Avg. position error (cm)	4.2 ± 1.8
Path efficiency ratio	0.87 ± 0.09
Processing time/diagram (s)	3.2 ± 0.7
OCR character error rate (%)	8.3
Success rate (overall)	100.0

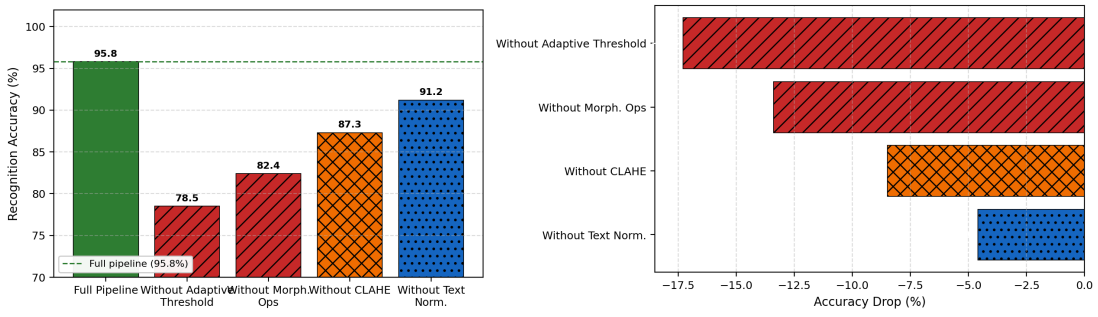
3.2. Ablation study

To validate the contribution of each image processing stage in the flowchart recognition pipeline, an ablation study was conducted by systematically removing key components. Table 7 presents the recognition accuracy and execution success rate measured on 30 representative hand-drawn flowcharts.

Table 7. Ablation study: impact of removing components

Configuration	Accuracy (%)	Success (%)
Full Pipeline	95.8	100.0
Without Adaptive Threshold	78.5	76.7
Without Morphological Ops	82.4	80.0
Without CLAHE	87.3	86.7
Without Text Normalization	91.2	93.3

Figure 18 visualizes component importance. Figure 18(a) compares the recognition accuracy obtained by the full pipeline and by each reduced configuration, showing that the complete preprocessing sequence provides the best performance. Figure 18(b) ranks the contribution of each component, highlighting adaptive thresholding as the most influential stage because its removal produces the largest accuracy decrease. The results demonstrate the critical role of each preprocessing step. The full pipeline achieved 95.8% recognition accuracy and 100% execution success. When adaptive thresholding was removed, accuracy dropped dramatically to 78.5% and success rate to 76.7%, representing a 17.3% accuracy loss, the most significant drop among all components. Removing morphological operations caused the second-largest impact, with accuracy falling to 82.4% and success to 80.0% (13.4% drop). CLAHE removal resulted in 87.3% accuracy and 86.7% success (8.5% drop), while text normalization showed the smallest impact at 91.2% accuracy and 93.3% success (only 4.6% drop), suggesting relative OCR robustness. These findings validate that adaptive thresholding and morphological operations are indispensable for reliable flowchart recognition, while all preprocessing steps contribute meaningfully to the overall system performance.



(a) (b)

Figure 18. Ablation study results (a) recognition accuracy per configuration and (b) ranked component importance showing adaptive thresholding as most critical (-17.3%)

3.3. Qualitative error analysis

To complement the quantitative evaluation, the incorrectly processed flowcharts were examined to identify the main causes of recognition failure. Four types were identified: ambiguous symbols, unclear arrow connections, illegible commands, and incomplete flowchart logic. Ambiguous symbols affected node classification, unclear arrows affected execution-order reconstruction, illegible text affected OCR, and missing nodes or connections prevented a complete movement sequence. Table 8 summarizes these cases and their effects on the recognition pipeline.

Table 8. Main error cases in flowchart recognition

Error case	Cause	Effect
Ambiguous symbol	Irregular shape	Incorrect node classification
Unclear arrow	Missing arrowhead or disconnected line	Incorrect execution order
Illegible command	Unclear handwriting	Incorrect OCR output
Incomplete logic	Missing node or connection	Invalid instruction sequence

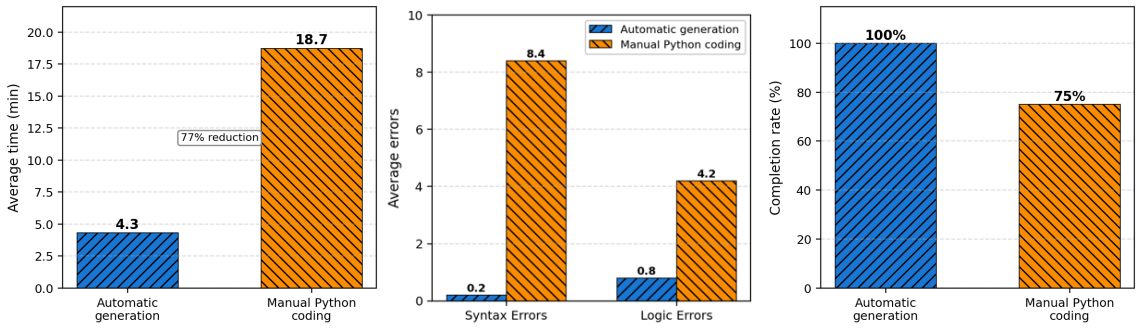
This analysis indicates that reliable recognition depends on distinguishable geometric symbols, readable commands, explicit arrow connections, and logically complete flowcharts. Flowcharts that do not satisfy these conditions may require manual correction before an executable movement sequence can be generated.

3.4. Baseline Comparison

A controlled comparison was conducted with 12 beginner Computer Science students aged 19–23, with basic programming knowledge but limited experience in mobile-robot control. All participants completed the same medium-complexity navigation task under two conditions: manual Python coding and the proposed flowchart-based workflow. Task-completion time, syntax and logic errors, completion rate, and perceived difficulty on a 1–5 scale were measured. Table 9 summarizes the results, while Figure 16 compares the performance of both approaches in terms of task-completion time (Figure 19(a)), syntax and logic errors (Figure 19(b)), and task-completion rate (Figure 19(c)).

Table 9. Manual coding versus automatic generation (n=12 students)

Metric	Proposed	Manual Python
Avg. Time (min)	4.3	18.7
Syntax Errors	0.2	8.4
Logic Errors	0.8	4.2
Completion Rate (%)	100	75
Difficulty (1–5)	1.6	4.1



(a) (b) (c)

Figure 19. Automatic generation versus manual Python coding (a) average task-completion time (77% reduction), (b) average syntax and logic errors, and (c) task-completion rate

The proposed workflow reduced average completion time by 77% (4.3 vs. 18.7 min), decreased programming errors, and increased the completion rate from 75% to 100%. These gains can be explained by a

shift in cognitive effort: students focus on algorithmic decisions in the flowchart rather than on Python syntax and low-level robot API details, while CoppeliaSim provides immediate visual feedback that helps them detect logical mistakes earlier. Overall, the system also achieved a 100% execution success rate in the evaluated navigation scenarios, 95.8% flowchart recognition accuracy, an average final-position error of 4.2 ± 1.8 cm, and a path efficiency of 0.87 ± 0.09. Learning gains and long-term knowledge retention were not directly measured. Accordingly, the system should support, rather than replace, students' learning of programming logic

3.5. Comparison with educational robotics platforms

Educational robotics platforms such as Gazebo, Webots, and CoppeliaSim provide simulation environments for programming and testing robot behavior [12]-[14]. However, they generally require users to create control instructions through scripts or digital programming interfaces. In contrast, the proposed framework uses CoppeliaSim as the execution environment while allowing users to define robot behavior through hand-drawn flowcharts. Under the evaluated conditions, the proposed framework achieved 95.8% recognition accuracy, a 100% execution success rate, and a 77% reduction in task-completion time compared with manual Python coding. Its main contribution is not to replace existing robotics platforms, but to provide an accessible way to generate algorithms for executing robot control instructions in CoppeliaSim. Note that no direct experimental comparison with other simulators was conducted, nor was platform-level performance evaluated.

4. CONCLUSION

This research presents a system that automatically interprets linear hand-drawn flowcharts, including a single-state loop composed of predefined movement commands, and converts them into executable instructions to control the Pioneer P3DX mobile robot in CoppeliaSim for programming education. Experimental results demonstrated high consistency, a strong correlation between instruction count and execution time, 100% navigation success without collisions in the evaluated obstacle scenarios, and 95.8% recognition accuracy, with adaptive thresholding identified as the most critical preprocessing component. In addition, the evaluation with 12 novice programming students showed a 77% reduction in task-completion time compared with manual Python programming, together with fewer errors and lower perceived difficulty. These findings validate the effectiveness of computer vision and machine learning in facilitating mobile robot control through the automatic generation of instructions from hand-drawn flowcharts, reducing programming barriers, and supporting the scalable use of the proposed system in robotics curricula and IoT-based laboratories. However, some limitations remain. The educational evaluation was conducted will a small sample from a single group, the validation was performed only in CoppeliaSim, so transfer to other simulators or physical robots is not yet demonstrated. Finally, recognition and scalability may degrade as flowchart complexity increases beyond linear structures with a single-state loop and predefined movement commands. Future work should integrate sensor feedback loops and AI-based decision-making, explore multi-robot coordination, and extend the framework to physical robots, industrial robot programming, and manufacturing systems.

ACKNOWLEDGEMENTS

The authors would like to thank to Eduardo Felipe Valdivia Quispe for their contribution and initial development work on this research, which helped lay the foundation for the study.

FUNDING INFORMATION

This research was funded by the Universidad Nacional de San Agustín de Arequipa, Perú, under the Applied Research Project "Development of an Interactive Robotics-Based Tool for Learning to Program"; under contract number PI-011-2023-UNSA. This article was written as part of this project.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Junior Gomez-Contreras	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓	
Jharold Alonso Mayorga Villena		✓				✓		✓	✓	✓				
Andrea López Condori		✓	✓			✓			✓	✓				
Yuber Velazco-Paredes	✓	✓		✓		✓			✓	✓		✓	✓	✓
Roxana Flores-Quispe	✓			✓		✓	✓	✓	✓	✓		✓	✓	✓

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal Analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject Administration
Fu : **Fu**nding Acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

INFORMED CONSENT

All student participants were informed about the study purpose and voluntarily agreed to take part in the educational evaluation. No personally identifiable information was collected; only anonymized task-performance metrics (completion time, error counts, completion rate, and perceived difficulty) were recorded and analyzed.

ETHICAL APPROVAL

The educational evaluation involved anonymized performance metrics from volunteer students and did not collect sensitive personal data. Therefore, formal institutional ethics-committee approval was not required under the applicable institutional guidelines for this type of classroom usability evaluation.

DATA AVAILABILITY

The data presented in this study are available on request from the corresponding author.

REFERENCES

[1] E. Mangina, G. Psyrra, L. Screpanti, and D. Scaradozzi, "Robotics in the context of primary and preschool education: A scoping review," *IEEE Transactions on Learning Technologies*, vol. 17, pp. 342–363, 2024, doi: 10.1109/TLT.2023.3266631.

[2] S. Tselegkaridis and T. Sapounidis, "Simulators in educational robotics: A review," *Education Sciences*, vol. 11, no. 1, Art. no. 11, Jan. 2021, doi: 10.3390/educsci11010011.

[3] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, Mar. 2006, doi: 10.1145/1118178.1118215.

[4] M. Román-González, J. C. Pérez-González, and C. Jiménez-Fernández, "Which cognitive abilities underlie computational thinking? criterion validity of the computational thinking test," *Computers in Human Behavior*, vol. 72, pp. 678–691, Jul. 2017, doi: 10.1016/j.chb.2016.08.047.

[5] L. Johnson, S. Adams Becker, M. Cummins, V. Estrada, A. Freeman, and C. Hall, *NMC Horizon Report: 2016 Higher Education Edition*. Austin, TX, USA: The New Media Consortium, 2016, doi: 10.5281/zenodo.5825548.

[6] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," *ACM SIGCSE Bulletin*, vol. 37, no. 3, pp. 14–18, Sep. 2005, doi: 10.1145/1151954.1067453.

[7] Kanika, S. Chakraverty, and P. Chakraborty, "Tools and techniques for teaching computer programming: A review," *Journal of Educational Technology Systems*, vol. 49, no. 2, pp. 170–198, Dec. 2020, doi: 10.1177/0047239520926971.

[8] J. Wenkang, C. Huaben, C. Mingyang, Z. Guobin, X. Lufeng, G. Roderich, Z. Rui, C. Ming, and Z. Shiyu, "Scalable multi-robot code-policy generation and deployment via language models," *Robotics*, vol. 4, no. 5, 2026, doi: 10.1038/s44182-025-00065-w.

[9] ACM Curriculum Committee on Computer Science, "Curriculum 68," *Communications of the ACM*, vol. 11, no. 3, pp. 151–197, Mar. 1968, doi: 10.1145/362929.362976.

[10] L. Alonso and J. Blazquez, "Cognitive benefits of handwriting: Evidence from neuroscience and education," *Frontiers in Psychology*, vol. 6, Art. no. 1056, Jul. 2015, doi: 10.3389/fpsyg.2015.01056.

[11] V. Agrawal, J. Jagtap, and M. P. Kantipudi, "An overview of hand-drawn diagram recognition methods and applications," *IEEE Access*, vol. 12, pp. 19739–19751, 2024, doi: 10.1109/ACCESS.2024.3357398.

[12] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2004, pp. 2149–2154, doi: 10.1109/IROS.2004.1389727.

[13] O. Michel, "Cyberbotics Ltd. Webots: Professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004, doi: 10.5772/5618.
[14] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Japan, Nov. 2013, pp. 1321–1326, doi: 10.1109/IROS.2013.6696520.
[15] B. Schäfer, M. Keuper, and H. Stuckenschmidt, "Arrow R-CNN for handwritten diagram recognition," *International Journal on Document Analysis and Recognition*, vol. 24, nos. 1–2, pp. 3–17, Jun. 2021, doi: 10.1007/s10032-020-00361-1.
[16] Q. Chen, D. Shi, G. Feng, X. Zhao, and B. Luo, "On-line handwritten flowchart recognition based on logical structure and graph grammar," in *Proceedings of the 5th International Conference on Information Science and Technology*, Changsha, China, Apr. 2015, pp. 424–429, doi: 10.1109/ICIST.2015.7289009.
[17] P. Rani and N. R. Chauhan, "Coal mine rescue robot simulation using V-Rep and Python," in *Advances in Interdisciplinary Engineering*. Singapore: Springer, 2019, pp. 733–739, doi: 10.1007/978-981-13-6577-5_71.
[18] A. Shitsukane, C. Otieno, J. Obuhuma, L. Mukhongo, and S. Kariuki, "A co-simulation framework using MATLAB and CoppeliaSim for path planning of nonholonomic mobile robots," *Global Journal of Engineering and Technology Advances*, vol. 23, no. 1, pp. 006–019, 2025, doi: 10.30574/gjeta.2025.23.1.0076.
[19] J.-H. Zhang, B. Meng, L.-C. Zou, Y. Zhu, and G.-J. Hwang, "Progressive flowchart development scaffolding to improve university students' computational thinking and programming self-efficacy," *Interactive Learning Environments*, vol. 31, no. 6, pp. 3792–3809, 2023, doi: 10.1080/10494820.2021.1943687.
[20] M. Andrzejewska, *et al.*, "Eye-tracking verification of the strategy used to analyse algorithms expressed in a flowchart and pseudocode," *Interactive Learning Environments*, vol. 24, no. 8, pp. 1981–1995, 2016, doi: 10.1080/10494820.2015.1073746.
[21] M. Resnick, *et al.*, "Scratch: Programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009, doi: 10.1145/1592761.1592779.
[22] J. Trower and J. Gray, "Blockly language creation and applications," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, Kansas City, MO, USA, 2015, p. 653, doi: 10.1145/2676723.2691871.
[23] "LabVIEW overview," *National Instruments*, 2026. https://www.ni.com/docs/en-%0AAS/csh?context=lv_overview (accessed Aug. 08, 2026).
[24] S. Boudraa and W. Hidouci, "Enhancement of degraded document images using adaptive methods," *Pattern Recognition and Image Analysis*, vol. 29, no. 1, pp. 118–133, Jan. 2019, doi: 10.1134/S1054661819010048.
[25] T. R. Singh and S. Roy, "Adaptive thresholding techniques for document image analysis," *Journal of Information Science and Engineering*, vol. 28, no. 4, pp. 767–782, Jul. 2012, doi: 10.48550/arXiv.1201.5227.
[26] J. Sauvola and M. Pietikäinen, "Adaptive document image binarization," *Pattern Recognition*, vol. 33, no. 2, pp. 225–236, Feb. 2000, doi: 10.1016/S0031-3203(99)00055-2.
[27] K. Turkowski, "Filters for common resampling tasks," in *Graphics Gems*, A. S. Glassner, Ed. Boston, MA, USA: Academic Press, 1990, pp. 147–165, doi: 10.1016/B978-0-08-050753-8.50042-5.
[28] A. R. Smith, "Color gamut transform pairs," *Computer Graphics*, vol. 12, no. 3, pp. 12–19, Aug. 1978, doi: 10.1145/965139.807361.
[29] K. Zuiderveld, "Contrast limited adaptive histogram equalization," in *Graphics Gems IV*, P. S. Heckbert, Ed. San Diego, CA, USA: Academic Press, 1994, pp. 474–485, doi: 10.1016/B978-0-12-336156-1.50061-6.
[30] E. D. Pisano, *et al.*, "Contrast limited adaptive histogram equalization image processing to improve the detection of simulated spiculations in dense mammograms," *Journal of Digital Imaging*, vol. 11, no. 4, pp. 193–200, Nov. 1998, doi: 10.1007/BF03178082.
[31] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Diego, CA, USA, Jun. 2005, vol. 2, pp. 60–65, doi: 10.1109/CVPR.2005.38.
[32] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, Jan. 1979, doi: 10.1109/TSMC.1979.4310076.

BIOGRAPHIES OF AUTHORS



Junior Gomez-Contreras is a computer science student at UNSA, Peru. His research interests include computer vision, robotics, and educational technology. He contributes to developing tools that bridge visual algorithm design and robotic execution for educational purposes. He can be contacted at jgomezcon@unsa.edu.pe.



Jharold Alonso Mayorga Villena is a computer science student at UNSA, Peru. His research interests include robotics simulation, image processing, and educational robotics. He contributes to flowchart recognition systems and CoppeliaSim integration. He can be contacted at jmayorgav@unsa.edu.pe.



Andrea López Condori is a computer science student at UNSA, Peru. Her research interests include machine learning, image processing, and educational technology. She works on preprocessing pipelines for diagram recognition and educational datasets. She can be contacted at alopezco@unsa.edu.pe.



Yuber Velazco-Paredes is a research professor at the National University of San Agustín of Arequipa - UNSA, Peru. His research interests include artificial intelligence, educational robotics, computer vision, and programming education. He supervises projects on diagram recognition and robotic control for education. He can be contacted at yvelazco@unsa.edu.pe.



Roxana Flores-Quispe is a research professor at the National University of San Agustín of Arequipa - UNSA, Peru. Her research interests include computer engineering, microcontroller-based control systems, programming education, and computational thinking. She supervises projects on accessible programming tools through visual representations. She can be contacted at rfloresqu@unsa.edu.pe.