Optimization of Robot Vacuum Navigation Algorithms Using Webots Simulation

Ethan Guo

Received September 28, 2024
Accepted December 07, 2024
Electronic access December 15, 2024

Robot vacuums have become increasingly popular household appliances, yet significant work remains to optimize their navigation system to improve cleaning efficiency. Navigation algorithms are critical to this optimization, and simulation provides a cost-effective and safe alternative to physical testing for development and refinement. This study used the 3D robot simulation software Webots to design and evaluate optimized movement patterns and navigation algorithms in a controlled environment. Random, zigzag, rectangular spiral, and snake algorithms were developed and compared based on cleaning time, travel distance, and path overlap with heat maps. The results showed that the zigzag algorithm improved cleaning efficiency, reducing total cleaning time by approximately 60% and travel distance by over 70% compared to the random algorithm. The rectangular spiral and the snake algorithms, with less path overlap, achieved even greater efficiency, cutting cleaning time and travel distance by more than 85%. This study highlights Webots as a valuable platform for optimizing navigation algorithms. It also establishes baseline performance metrics for different navigation strategies and identifies potential paths for future work.

Introduction

The use of autonomous robotics has increased in industries such as manufacturing, healthcare, and logistics, where it has significantly improved productivity and efficiency[1,2,3,4]. Mobile robots also play an important role in modern daily life, with robot vacuum cleaners being one such example. These autonomous vacuums are designed to navigate and clean floors independently, equipped with infrared or touch sensors to avoid obstacles. Due to their ease of use and convenience, they have massively risen in popularity with 18.5 million units shipped to consumers in 2023[5,6,7].

Since their inception, robotic vacuums have evolved significantly, advancing from simple programmable devices to sophisticated systems capable of navigating complex environments autonomously[8,9]. The first commercial robot vacuum, Roomba, introduced in 2002, relied on a random movement pattern as its navigation method[10]. Since then, research has aimed to improve navigation capabilities through techniques such as manual planning and navigational waypoints. Modern algorithms such as simultaneous localization and mapping (SLAM) combine precise sensor data with mapping and movement planning, resulting in more efficient navigation[11]. In recent years, simulation software has become a crucial tool for developing and optimizing the navigation algorithms for robotic vacuums. Simulations provide a controlled and replicable environment where different control strategies can be tested without the constraints of physical testing, such as time, cost, and safety concerns. Researchers can refine robotic algorithms by leveraging simulation platforms to ensure optimal performance in real-world conditions. One widely used simulation software is Webots, a mobile robotics simulator developed by Cyberbotics Ltd. Webots provides a rapid prototyping environment for modeling, programming, and simulating mobile robots[12]. It features a physics engine that provides realistic simulations of robot interactions, making it particularly valuable for research and educational purposes[13]. Previous studies have shown that Webots enables precise robot motion control, making it a reliable platform for simulating and refining robotic algorithms[14]. Limited research has been conducted on the evaluation of robot navigation algorithms in simulated environments, particularly for robotic vacuums. In this study, Webots was used to assess the navigation and efficiency of a robot vacuum. Four different navigation algorithms were evaluated, with each algorithm analyzed for cleaning efficiency based on cleaning time, and travel distance and path overlap.

Methods

Sensors

Sensors enable robots to perceive and interact with their environment, providing the necessary data for autonomous task execution, decision-making, and responsiveness to changes in their surroundings. In this study, three types of commonly used sensors, bumper sensors, distance sensors, and inertial measurement unit (IMU), were attached to the robot vacuum to facilitate data collection and algorithm decision-making. The

same sensors were used by the robots in every algorithm. The bumper sensors are physical contact sensors that detect when the robot comes into contact with an obstacle. The distance sensors enable the robot to detect obstacles a short distance ahead, helping it avoid collisions before bumping into them. IMU can determine a robot's orientation and angular change, providing essential data for navigation and motion control. An IMU provides orientation measurements in terms of roll, pitch, and yaw, which collectively define a robot's orientation in three-dimensional space. However, for this study, which was conducted within a simulated 2D map environment, only the yaw component—representing rotation around the vertical axis—is utilized. Yaw indicates the rotation angle when an object turns left or right on a horizontal plane, similar to the steering direction of a car.

Navigation Algorithms

Random Navigation Algorithm

The random navigation algorithm uses the bumper and distance sensor to detect obstacles. When an obstacle is detected, the robot will turn at a random angle between 0° and 180° in the opposite direction. For example, if the left bumper sensor detects an obstacle, the robot will back up and turn right at a random angle up to 180°. This process helps the robot avoid collisions while unpredictably navigating the environment.

Code for Random Algorithm :

```
def random_move():
    while not all area is covered:
        get_sensor_status()
        if obstacle_detected:
            turn(random_angle)
        go_forward_one_step()
```

Zigzag Navigation Algorithm

The zigzag algorithm is a movement pattern that enables a robot to efficiently cover an area by moving back and forth in straight lines at a slight angle, forming a zigzag path. The robot moves in a straight line until it encounters an obstacle, where it makes a sharp turn at a calculated angle, typically just under 180 degrees, to avoid retracing its previous path. This slight angle adjustment allows the robot to make incremental forward progress, ensuring thorough coverage without gaps, though it may result in some overlap.

Code for Zigzag Algorithm :

```
def zigzag_move():
    # Assume the robot starts at a corner of the room.
    # Calculate the turning angle for a zigzag pattern in radians.
    angle = π - CLEANING_WITDH / MAP_SIZE
    while not all area is covered:
        get_sensor_status()
        if obstacle_front:
            if current_clean_direction is left of robot_orientation:
                turn_left(angle)
            if current_clean_direction is right of robot_orientation:
                turn_right(angle)
        elif obstacle_left or obstacle_right:
            # near wall
            follow_left_wall() if obstacle_left else follow_right_wall()
        go_forward_one_step()
```

Rectangular Spiral Navigation Algorithm (Spiral Algorithm)

The rectangular spiral algorithm enables a robot to cover an area by spiraling outward from a central starting point in an expanding rectangular pattern. Starting at the center, the robot moves straight in one direction until it reaches a boundary or completes a predetermined distance, then turns 90° to continue along the next side. After every two turns, it extends its path incrementally, forming a progressively larger rectangle around the starting point. The distance between each revolution is consistent and matches the vacuum's cleaning width, creating a pattern similar to an Archimedean spiral but with straight lines. Upon reaching a wall, the robot stops expanding in that direction but continues spiraling until all boundaries are reached, ensuring coverage with minimal gaps.

Code for Spiral Algorithm :

```
def spiral_move():
    # Assume the robot starts at the center of the room.
    # The robot moves in a rectangular spiral pattern
    # Turn 90 degrees to the left on each turn.
    # The travel distance increases linearly with every two turns.
    multiply = 2
    while not all area is covered:
        get_sensor_status()
        if current_travel_distance() > CLEANING_WITDH * (multiply // 2):
            turn_left(π/2)
            multiply = multiply + 1
            reset_current_travel_distance()
        if obstacle_front:
            # reaches the edge
            turn_left(π/2)
            follow_right_wall()
        else:
            go_forward_one_step()
```

Snake Navigation Algorithm

The snake algorithm is an optimized version of the zigzag algorithm, designed with parallel paths to create a more efficient navigation pattern. Unlike the traditional zigzag pattern, which includes a slight angle between paths, the snake pattern uses parallel back-and-forth paths. To ensure forward progress and prevent retracing, the snake algorithm consists of a small vertical shift between each parallel path. This shift matches the robot's cleaning width, enabling efficient coverage without leaving gaps. Named for its resemblance to a snake's smooth,

Code for Snake Algorithm:

```
def snake_move():
   # Assume the robot starts at the corner of the room.
   # Normally, the robot's orientation is perpendicular to the cleaning direction.
   # When the robot makes turns to follow the wall for a short distance temporarily,
the cleaning direction remains unchanged.
   # During this time, the robot's orientation parallels the cleaning direction.
   while not all area is covered:
       get_sensor_status()
       if obstacle_front:
           if obstacle_left or obstacle_right:
               # reaches corner,
               # change clean direction in case not all area is covered
               change_clean_direction()
               turn_right(π/2) if obstacle_left else turn_left(π/2)
           else:
               # temporarily follow the wall
               # travel with a short distance equal to the cleaning width
               if current_clean_direction is left of robot_orientation:
                   turn_left(π/2)
                   follow_right_wall(CLEANING_WITDH)
                   turn_left(π/2)
               if current_clean_direction is right of robot_orientation:
                   turn_right(π/2)
                   follow_left_wall(CLEANING_WITDH)
                   turn_right(π/2)
       elif obstacle_left or obstacle_right:
           # edge, follow the wall
           follow_left_wall() if obstacle_left else follow_right_wall()
       go_forward_one_step()
```

slithering motion, this algorithm ensures thorough cleaning in a streamlined pattern.

Wall-following technique

In the zigzag, spiral, and snake algorithm code, the wall-following technique supports the main navigation algorithms by guiding the robot along a wall until it travels a predefined distance or detects an obstacle ahead.

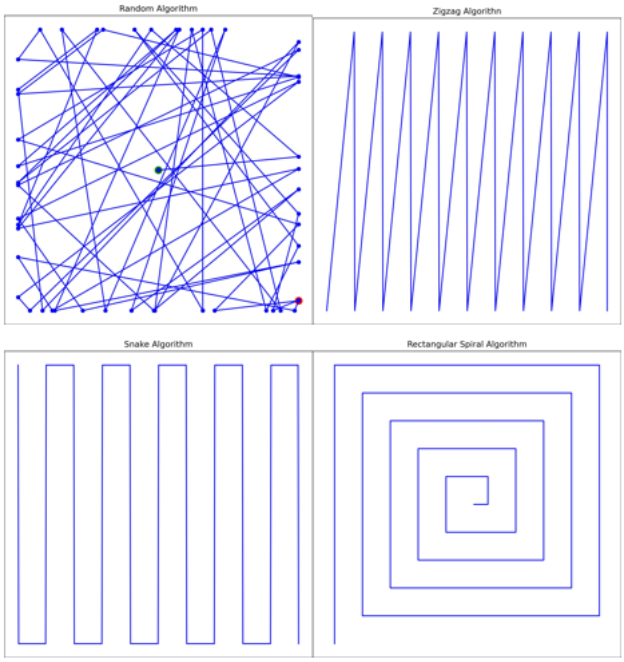
Code for Wall-Following Algorithm:

```
def follow_wall(travel_length=MAX_TRAVEL_LENGTH, follow_left_wall=True):
   reset_current_travel_distance()
   while get_current_travel_distance() < travel_length and no_obstacle_front:
       if follow_left_wall:
           distance_signal = get_distance_signal_to_left_wall()
       else:
           distance_signal = get_distance_signal_to_right_wall()
       # the bigger the distance signal, the closer to the obstacle
       if distance_signal > DISTANCE_THRESHOLD + tolerance:
           # too close to the wall, turn away from the wall slightly
           turn_right_one_step() if follow_left_wall else turn_left_one_step()
       elif distance_signal < DISTANCE_THRESHOLD:
           # too far from the wall, turn towards the wall slightly
           turn_left_one_step() if follow_left_wall else turn_right_one_step()
       go_forward_one_step()
```

Summary of Algorithms

The algorithms used in this study were selected for their computational efficiency and simplicity. They are simulated in the simplified but same environment. Figure 1 shows the theoretical path designs of the four algorithms used in this study and Table 1 provides a high-level comparison of each algorithm's strengths and limitations. While the zigzag, spiral and snake algorithms offer distinct advantages over the random algorithm, they also present challenges, particularly when room configurations change.

Figure 1. Theoretical path design of the four algorithms.



Due to the limited time and resources, this study did not consider more complex environment setups or more advanced algorithms like SLAM.

Results

In this study, the robot was tested in a 5m by 5m room with no obstacles to ensure simplicity and consistency in measurements.

Cleaning Time

The zigzag, spiral, and snake algorithms were compared to the random algorithm based on the time taken to clean the room, with the time recorded at every 10% coverage increment (Table 2). Given the random algorithm's variability, the table data represents an average of 10 runs. In contrast, the data for the other algorithms reflects a single run due to no variability. Table 3 summarizes the individual data set for each random algorithm run, with relative standard deviation (RSD) ranging from 8 to 20%. Figure 2 provides a visual comparison of cleaning time versus coverage for the four algorithms.

Travel Distance

The travel distance of each navigation algorithm was also measured at every 10% coverage increment, with the random algorithm's data again representing an average of 10 runs, while data for other algorithms represent singular runs (Table 4). Table

Table 1. High-level comparison of the navigation algorithms evaluated in this study.

Algorithm	**Description**	**Strengths**	**Limitations**
Random	Uses bumper sensors to detect obstacles, then turns randomly (0-180°) in the opposite direction when hitting an object.	Simple, reliable for obstacle avoidance	Increasingly inefficient in the larger room
Zigzag	Moves in a zigzag pattern, alternating between straight lines until encountering an obstacle, then turns with an angle less than 180°.	Fewer turns required	Some level of path overlap; worse in the larger room
Spiral	Moves in squares that grow in size, eventually reaching the size of the room.	No gaps in movement	Becomes inefficient when the room's length is longer than its width
Snake	Moves in a snakelike pattern, turning 90° after hitting a wall, moving a short distance, then turning again to travel the other way.	Least amount of overlap	Moves forward between turns when a wall is reached

Table 2. Comparison of the cleaning times of the four algorithms.

Percentage of Area Cleaned	Average Cleaning Time (seconds)			
	Random algorithm	Zigzag algorithm	Spiral algorithm	Snake algorithm
10%	18.44	20.06	24.1	17.47
20%	36.84	56.74	43.58	34.72
30%	59.31	112.03	62.4	54.82
40%	82.55	150.05	79.46	72.03
50%	116.28	171.65	98.14	92.16
60%	158.9	219.65	115.14	109.41
70%	213.38	261.79	132.7	129.5
80%	276.87	301.66	148.77	146.66
90%	411.09	355.65	176.61	166.78
100%	1,578.16	384.86	197.79	184.58

Data of the random algorithm represents an average of 10 runs.

Table 3. Cleaning times for random navigation algorithm

	No. of Runs												
Percentage of Area Cleaned	1	2	3	4	5	6	7	8	9	10	AVG	SD	RSD
10%	17.3	16.4	15.1	15.3	24.6	16.6	24.1	15.7	16.8	22.4	18.4	3.7	20.2
20%	34.6	44.3	30.3	30	41.8	35.8	38.6	33.7	34.8	44.5	36.8	5.3	14.3
30%	60.4	71.6	58.3	49.3	69.2	59.5	55.8	52.7	50	66.4	59.3	7.8	13.1
40%	91.3	95.4	83	70.7	84.1	88.7	77.4	72.2	76.6	86	82.5	8.2	9.9
50%	120.5	133.8	113.1	103	122.9	124.5	101.4	114.9	113.2	115.4	116.3	9.8	8.4
60%	154.7	227.1	173.6	144.9	150.7	162.9	130.8	149.5	148.7	146.2	158.9	26.5	16.7
70%	205.3	282	211.2	218.2	188.4	187.1	205.4	240.2	201.4	194.5	213.4	28.7	13.4
80%	294.3	347.9	313.3	256.5	248	240.2	256.9	323.4	263.8	224.4	276.9	40.5	14.6
90%	369.2	397.8	447.2	479.6	334.6	404.4	454.8	552.4	351.2	319.6	411.1	72.8	17.7
100%	1496.6	1662.4	1811	1981.5	1252.4	1895	1241.7	1805.9	1584.4	1050.7	1578.2	312.8	19.8

Figure 2. Comparison of cleaning time of the four algorithms.

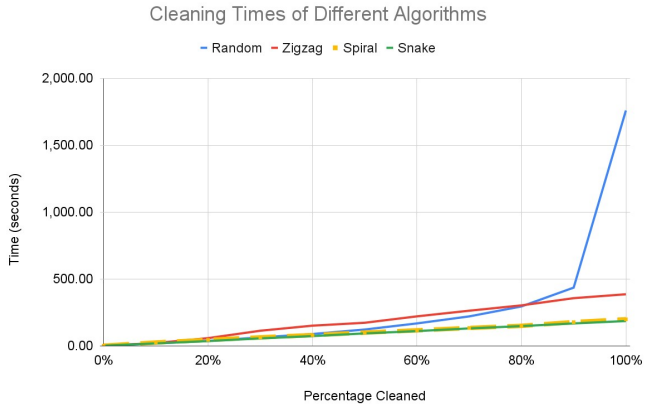
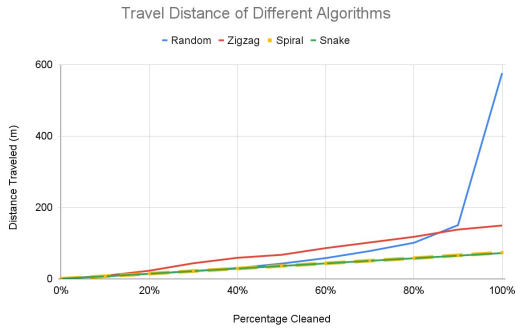


Figure 3. Comparison of travel distance of the four algorithms.



5 provides a summary of the individual data set for each random algorithm run, with RSD ranging from 8 to 20%. Figure 3 presents a visual comparison of cleaning time versus coverage for the four algorithms, and these distance data align with the cleaning time results.

Navigation Paths

The navigation path of each algorithm is shown in Figures 4-7 respectively.

Path Overlap

Figure 8 illustrates the path overlapping characteristics of the four algorithms using heatmaps. The heatmaps are created based on data from the occupancy grid, a data structure used to create a 2D map of the room by dividing the room into a grid of cells. Each cell in the grid represents how often a specific location was visited. As the robot moves through each cell in the environment, the number of times the cell was visited was updated. The heatmaps created based on the occupancy grids visually map out the robot's path and show the areas it has covered. The higher

Figure 4. Vacuum navigation path with the random algorithm (representing 20%, 40%, 60% and 100% coverage).

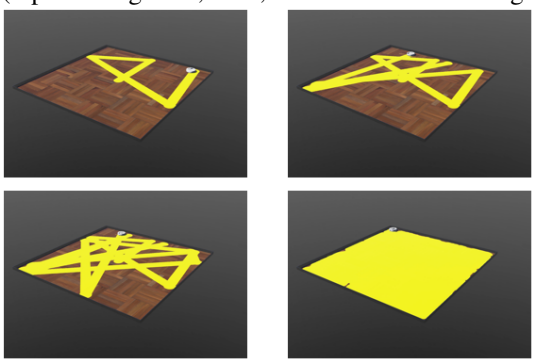


Figure 5. Vacuum navigation path with the zigzag algorithm (representing 20%, 40%, 60%, 100% coverage).

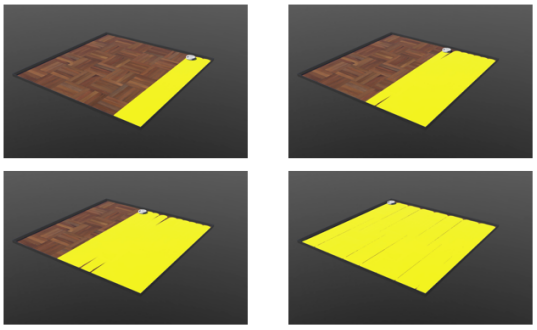


Figure 6. Vacuum navigation path with the Spiral algorithm (representing 20%, 40%, 60%, 100% coverage).

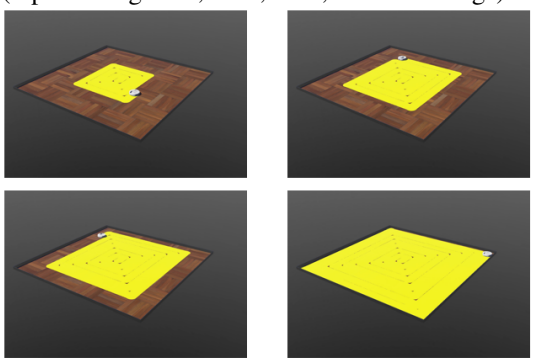


Figure 7. Vacuum navigation path with the Snake algorithm (representing 20%, 40%, 60%, 100% coverage).

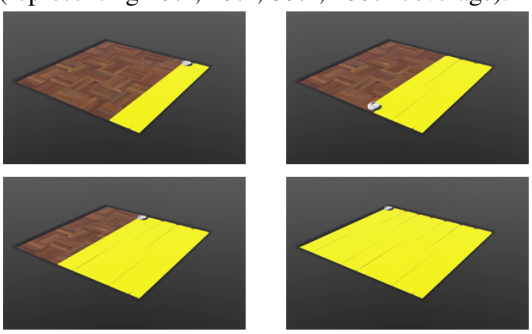


Table 4. Comparison of the travel distance of the four algorithms.

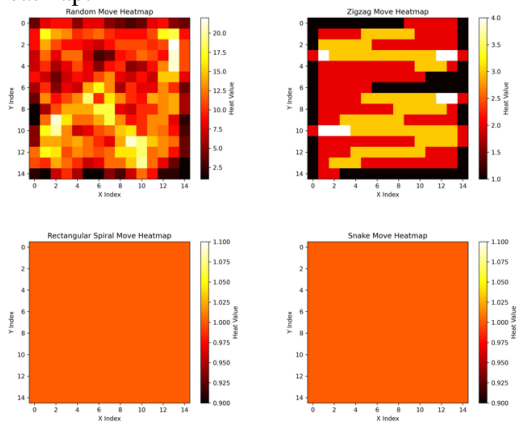
Percentage of Area Cleaned	Average Travel Distance (meters)			
	Random algorithm	Zigzag algorithm	Spiral algorithm	Snake algorithm
10%	6.7	8.22	6.99	6.88
20%	13.7	22.45	14.11	14.02
30%	21.5	43.43	21.52	21.35
40%	30.1	58.69	28.65	28.47
50%	42.4	67.06	36	35.82
60%	57.8	85.83	43.1	42.95
70%	77.7	101.64	50.48	50.28
80%	100.8	117.49	57.57	57.37
90%	150.3	137.84	65.65	64.72
100%	576.5	149.2	73.12	71.9

Data of the random algorithm represents an average of 10 runs.

Table 5. Travel distance for random navigation algorithm (10 runs).

Percentage of Area Cleaned	No. of Runs										AVG	SD	RSD
	1	2	3	4	5	6	7	8	9	10			
10%	6.4	6.3	5.8	5.6	8	6.1	9.2	6.5	6.2	7	6.7	1.1	16.6
20%	13	16.3	12.3	11.7	14.4	14.3	15.1	12.8	13.4	13.6	13.7	1.4	10
30%	22.7	25.1	22.3	17.9	23.4	22.7	21.9	20.3	19.2	19.9	21.5	2.2	10
40%	33.4	31.9	31.8	27.2	29.6	33.4	29.7	27	28.7	27.7	30.1	2.5	8.2
50%	44.3	46.8	43.4	38.6	43.2	47	37.4	41.1	43.3	38.8	42.4	3.3	7.9
60%	56.6	76.8	66.3	54.4	54.6	62	49.2	54.4	54.9	49.2	57.8	8.5	14.6
70%	74.2	96.7	77.3	82.3	69.1	69.7	76.9	88.6	76	66.2	77.7	9.4	12
80%	104	116.4	114.3	98	90.7	90	96.9	120.6	99.6	78.1	100.8	13.3	13.2
90%	132.4	135.2	165.1	176.1	122.7	150.9	170.1	205.5	133.1	111.8	150.3	28.7	19.1
100%	525.2	584.3	662.5	724.9	458.2	696.6	465.5	670.9	596.1	380.6	576.5	115.7	20.1

Figure 8. Path Overlapping of the four algorithms using Heatmap.



values indicate more frequent visits and a high degree of path overlapping.

Discussion

As summarized in the Results section, the spiral and snake algorithms demonstrated significant improvements in both cleaning time and travel distance compared to the zigzag algorithm, with even greater improvement over the random algorithm. Although minimal improvement was observed up to around 20% room coverage, the differences became much more prominent beyond that point. The spiral and snake algorithms cleaned the entire room in less than half the time required by the zigzag algorithm and approximately eight times faster than the random algorithm. The zigzag algorithm performed the worst until around 90% coverage, where it began to surpass the random algorithm. This is likely due to the zigzag algorithm overlapping itself upon reaching room edges, as illustrated in Figure 5. In contrast, the random algorithm's overlapping becomes more frequent over time, causing it to slow down significantly. The random navigation algorithm showed substantial path overlap, while the zigzag algorithm displayed moderate path overlap. These overlaps align with their longer cleaning time and travel distance. In contrast, the spiral and the snake algorithms exhibited no path overlaps thus significantly improving the cleaning efficiency. Both the spiral and the snake algorithms achieved a linear progression of coverage due to the absence of overlap and the consistency of their paths. While the differences between the spiral and snake algorithms were minimal, the slight differences were due to the time required to make turns and the different times needed to clean the edges along the walls.

Conclusion

This study evaluated the efficiency of four different robot vacuum navigation algorithms - random, zigzag, rectangular spiral, and snake - in a simulated 5m x 5m empty room using Webots. The rectangular spiral and snake algorithms, designed to minimize path overlap, significantly outperformed the random and zigzag algorithms in cleaning time, travel distance, and path overlap, highlighting their effectiveness in achieving comprehensive coverage efficiently.

It is important to note that this study was conducted in a controlled, obstacle-free environment. The presence of obstacles could alter performance, as the efficiency of the spiral and snake algorithms may decrease with more obstacles. In such scenarios, further optimization or exploration of more advanced algorithms, such as simultaneous localization and mapping (SLAM), may be necessary to maintain efficiency.

Overall, this study demonstrated Webots as a tool for developing and comparing robot navigation algorithms and established a foundational baseline for future research in optimizing robot vacuum navigation in more complex environments using simulation.

Acknowledgments

The author would like to thank Professor Robert Chun for his guidance through this research topic.

References

1 F. Fahimi, *Autonomous robots*, Springer, 2009.

2 G. P. Moustris, S. C. Hiridis, K. M. Deliparaschos and K. M. Konstantinidis, *International Journal of Medical Robotics and Computer Assisted Surgery*, 2011, **7**, 375–392.

3 A. Shukla and H. Karki, *Robotics and Autonomous Systems*, 2016, **75**, 490–507.

4 M. Shamout, R. Abdallah, M. Alshurideh, H. Alzoubi, B. Al-Kurdi and S. Hamadneh, *Uncertain Supply Chain Management*, 2022, **10**, 577–592.

5 R. Al-Khateeb and B. Mustafa, *International Journal of Responsible Artificial Intelligence*, 2022, **12**, 9–18.

6 A. Eren and H. Doğan, *Turkish Journal of Engineering*, 2022, **6**, 166–177.

7 IDC, *Worldwide smart vacuum market shipped 18.5 million units in 2023, according to IDC*, 2024, https://www.idc.com/getdoc.jsp?containerId=prUS52072324.

8 T. Asafa, T. Afonja, E. Olaniyan and H. Alade, *Alexandria Engineering Journal*, 2018, **57**, 2911–2920.

9 Q. Huang, *Applied Sciences*, 2023, **13**, 10031.

10 J. Olson, I. Kirillov and J. Sweeney, 2008.

11 I. Ayman, N. Waleed, T. El-Shabrawy and M. Ashour, 2023 2nd International Conference on Smart Cities 4.0, 2023, pp. 370–377.

12 O. Michel, *International Journal of Advanced Robotic Systems*, 2004, **1**, 39–42.

13 B. Magyar, Z. Forhecz and P. Korondi, Proceedings of the IEEE International Conference on Industrial Technology (ICIT), 2003, pp. 1–6.

14 A. Farley, J. Wang and J. Marshall, *Simulation Modelling Practice and Theory*, 2022, **120**, 102629.