Mobile Robots with Dynamic Obstacle Avoidance

Maram Ali
Centre for Environmental Mathematics,
Faculty of Environment, Science and Economy,
University of Exeter, Penryn Campus,
Cornwall TR10 9FE, United Kingdom.
Email: ma935@exeter.ac.uk

Saptarshi Das, *Member, IEEE*
Centre for Environmental Mathematics,
Faculty of Environment, Science and Economy,
University of Exeter, Penryn Campus,
Cornwall TR10 9FE, United Kingdom.
Email: S.Das3@exeter.ac.uk, saptarshi.das@ieee.org

***Abstract*—Robots with autonomous navigation capabilities have become increasingly popular after the advances in robotic automation, particularly in the area of pathfinding algorithms. These algorithms enable robots to safely traverse through complex environments with both stationary and moving obstacles. Applications of this field range from data acquisition to surveys of hazardous situations and transportation by industrial robots. The most commonly utilized approach for two-dimensional obstacle avoidance is grid-based pathfinding algorithms. These methods function by initially generating a grid consisting of nodes and edges based on the environment. In this paper, we explore an implementation of a variation of the A* pathfinding algorithm on a 15x15 grid. The A* algorithm was chosen because it guarantees finding the optimal route between starting and ending points. A* is a grid-based algorithm that falls under the category of search-based algorithms. The Maximum Velocity Obstacle (MVO) algorithm undergoes rigorous testing to evaluate its performance, and we examine how the simulation input parameters influence the algorithm's effectiveness. The experimental results indicate that the MVO algorithm is an efficient and reliable solution for dynamic obstacle avoidance in a grid-based setting. Moreover, this study demonstrates that the algorithm can be further optimized by using more advanced techniques such as combining it with existing pathfinding algorithms, like artificial neural networks. This would enable the robot to adapt to unpredictable environments in future research.**

***Index Terms*—Obstacle-avoidance, grid-based path-finding, A* algorithm, mobile robot simulation.**

I. Introduction

The operations of autonomous and semi-autonomous vehicles rely heavily on obstacle avoidance algorithms. These algorithms enable robots to determine the optimal path based on information collected about their environment. Advances in sensor technology and big data processing have led to innovative applications of these algorithms.In practice, various sensors such as lidar (light detection and ranging), sonar, radar, cameras, and infrared rangefinders are used to gather data on the robot's surroundings. This information helps the robot understand its environment and make decisions on the best course of action to avoid collisions with nearby objects [1], [2]. In nature, we can observe evidence of super-intelligence in the way that individual animals work together in a swarms. This phenomenon is exhibited by creatures such as bees, termites, and birds. Scientists have drawn inspiration from this biomimicry, developing algorithms that imitate these nature-inspired behaviors in artificial systems. One critical challenge in modeling swarm behavior is ensuring effective obstacle avoidance among swarm members [3]–[5]. Methods such as the Boid algorithm [6], [7], simulate the flocking behavior of birds using three rules namely, separation, cohesion, and alignment. Among these rules, separation deals with dynamic obstacle avoidance. Another algorithm commonly used to solve obstacle avoidance in swarm problems is the self-organizing migrating algorithm (SOMA) [3], [8]. SOMA is a population-based algorithm that uses fuzzy logic and neural networks to simulate the intelligent behavior of a group of animals in competition-cooperation with each other to look for food [9]–[11].

Obstacle avoidance is a technique used by unmanned aerial vehicles (UAVs) to avoid collisions while in flight. The UAVs must avoid collisions in three dimensions in this setting [12], using millimeter wave radar and monocular camera data fusion techniques amongst many other sensing methods. The unmanned surface vehicles (USVs), on the other hand, avoid collisions in two dimensions using algorithms like the extended Kalman filter (EKF) [13] and multiple variants of sensor data fusion, state estimators or observers, controllers, and control objectives [14]. Since the environment of the robot has multiple dimensions, different algorithms must be employed and compared in these situations. Similarly for unmanned underwater vehicles (UUVs) [15], the obstacle avoidance algorithms include recurrent neural networks (RNNs) with convolution [16], fuzzy relation-based sonar image processing for potential field and edge detection using gradient detection [17] and a combination of intelligent algorithms like A* algorithm, artificial potential field, rapid-exploring random tree, neural networks, genetic algorithms, fuzzy logic, reinforcement learning etc. [18]. In a two-dimensional space, dynamic obstacle avoidance is the main topic of this paper. Robots that detect landmines, clean houses, drive themselves, work as industrial robots in material transport, and other applications mostly use UAVs, USVs, and UUVs. Since 2D obstacle avoidance algorithms are widely used in both the military and civilian applications, it is crucial to understand them through numerical simulations. The capacity to avoid obstacles is a key component for ground mobile robots, allowing them to

979-8-3503-1997-2/23/$31.00 ©2023 IEEE

effectively and safely undertake a wide range of duties and tasks such as exploration and transportation [19].

II. Related Works on Obstacle Avoidance

Obstacle avoidance can be summarized into two distinct problems, namely, environment modelling and the obstacle avoidance algorithm [9].Environment modeling serves as the foundational step in implementing obstacle avoidance. It involves creating representations of the environment, such as spatial and topological maps, based on the specific application requirements. The accuracy and reliability of environment processing and modeling are crucial for the successful operation of obstacle avoidance algorithms, as they depend on a precise understanding of their surroundings.Various representations are commonly used for environment modeling, including metric, topological, and hybrid approaches. The metric representation employs a world coordinate system to define the locations of objects within the environment. In contrast, the topological representation relies on nodes for object location and edges to demonstrate the relationships between these nodes. By accurately modeling the environment, obstacle avoidance strategies can operate effectively and predictably. Hybrid representation combines the two (metric and topological) representations to perform a simultaneous location and mapping of the environment [9]. Other types of environment processing techniques such as the Otsu algorithm are used when an image of the environment is used as an input to an obstacle avoidance algorithm [19]. The Otsu algorithm for instance converts the input image to a binary representation of the environment [12]. This study will focus on the metric environment model, specifically, a branch of metric representation referred to as the grid map metric representation. This technique divides the environment into a grid and defines the location within the grid using the grid coordinates.

Various path-finding algorithms have been developed to solve the problem of obstacle avoidance. They are categorized into different groups based on the algorithm's approach to the problem. For instance, the genetic algorithm and the neural networks are categorized as intelligent algorithms. They are ideal for environments with unlabeled features such as the use of image recognition in model environment processing. An additional characteristic of these algorithms is that they draw inspiration from biological processes in the way they operate [20]. In real-world applications, physical properties such as the kinematics of the robot have to be considered while designing path planning algorithms. This prevents the algorithms from producing a physically impossible route compromising the safety of the robot. Circle grid trajectory cell (CGTC) algorithms consider kinematics and dynamics constraints during the computation of a safe path during USV obstacle avoidance. This is achieved in three distinct steps. The first step involves scanning the environment using the circle grid to make sure that the most efficient route and waypoints of the robot have a smooth rate of change during its search process. The second step involves incorporating the robot's dynamics and kinematics into the pathfinding algorithm. It does this by mathematically modelling the robot and simulating the proposed path on the robot to optimize the path for a given robot. Lastly, the algorithm smoothens the path based on the results in step two [21].

This section will discuss the development of pathfinding algorithms under different categorizations i.e. sampling-based and search-based algorithms since they are commonly used in graph-based high-dimensional environments. Sampling-based path-finding algorithms are probabilistic algorithms that operate by randomly sampling the environment or space to find the optimum path to the target position. The probabilistic roadmap (PRM) and the rapidly-exploring random trees (RRT) algorithms are the most preferred sampling algorithm due to their accuracy in practice. In addition, both of these algorithms are considered to be probabilistic complete which means that the probability of a collision exponentially decays to zero as the number of samples taken in the state-space increases [22]. This makes these types of path-finding algorithms ideal for optimal motion planning for large state space/environments.

On the other hand, a searching-based algorithm, collect data about their environment to decide the next move. Early versions of searching-based pathfinding algorithms were used for static obstacle avoidance. They do this by searching the whole environment and computing the most efficient path to the target. One major drawback of this technique was that the computation cost directly affects the time cost of running the algorithm i.e. the time cost linearly increases with an increase in the environment state space. In programming terms, the algorithm has a complexity of $O(n)$. Efficient variations of searching-based algorithms have been developed to tackle this problem. Localized searching-based algorithms only search their neighbor location for obstacles and through fuzzy logic decide on the next move. Agents that use this technique do not have a complete picture of the whole environment space [3]. An example of a Localized searching-based pathfinding algorithm is the A* algorithm. Similar to this, the random-walk search technique used in this study [23] is that enables the swarm of robots to investigate unknown environment while avoiding static obstacle. In complicated and dynamic contexts where a pre-defined map might not be available or might be continually changing.

The A* algorithm is a heuristic algorithm that is used in the autonomous navigation of USVs. It is preferred due to its high accuracy in obstacle avoidance, both static and dynamic obstacles, and its simplicity in implementation [20]. In addition, the A* algorithm can be used in higher dimensional pathfinding applications such as obstacle avoidance in 3D space for drones. A variation of the A* algorithm will be implemented in this paper to demonstrate dynamic obstacle avoidance. Other notable pathfinding algorithms are different neural network variants that utilize modern deep learning methods to analyze the environment and based on the experience of the model, the neural network computes an optimum path while avoiding obstacles in the environment.

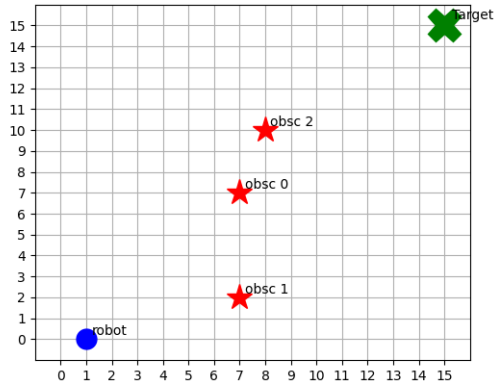

Fig. 1. A frame of the simulation animation showing a 15 by 15 grid environment with the robot, target and dynamic obstacles.

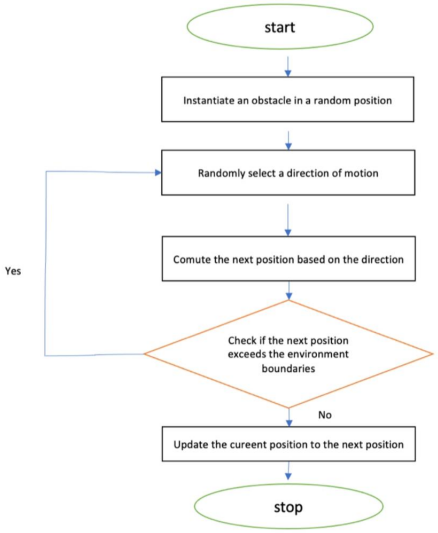

Fig. 2. flow chart showing the algorithm that defines the behavior of obstacles in its environment

III. Material and Methods

A. Implementing the Grid Environment

In the current numerical implementation, a 15 by 15 grid has been created using the Python programming language and the Matplotlib visualization library. An axis is plotted with x-limits and y-limits set to 0 and 15, which defines the size and boundaries of the environment. The lines $x = 0, x = 15, y = 0$, and $y = 15$ are considered virtual static boundaries, which means that agents within these boundaries are confined to the grid boundaries and should treat the virtual boundaries as static obstacles. This ensures that the agents do not move beyond the boundaries of the environment. Overall, this implementation provides a structured and well-defined environment for the agents to navigate through. The robot is represented by a blue circle on the plot, the obstacles are represented by a red star, and the target is represented by a green X symbol as in Fig. 1.

B. Implementing the Dynamic Obstacles

This involves setting the representation of obstacles in the matplotlib axes and defining how the obstacle agents move within the environment. Obstacles are instantiated in random positions in the environment in the first iteration. Next, obstacles move randomly within the environment in only four directions i.e., up, down, left, and right. Obstacles just like the robot can move 1 time-step in each iteration of the simulation. The algorithm movement is decided using the algorithm represented by the flowchart shown in Fig. 2.

Agents, which include the obstacles and robots, move at a constant velocity i.e. with acceleration = 0. Since the agents move one step per unit time step (iteration), the distance per iteration for each agent is calculated as: distance $= v * t$ where, t is a unit (one) time unit and v is the agent's velocity. Therefore, the distance moved in a unit time step is given by: distance $= v$ since $v * 1 = v$.

Using the distance per unit step, calculated above, an agent calculates its position in the next iteration by adding the distance to its current position in the environment i.e., For right and left movements, next position = ($x \pm$distance, y), while for up and down movement: next position $= (x, y \pm$distance).

C. Implementing an Obstacle Avoiding Robot

The MVO algorithm was used for the robot navigation. But at first, the robot had to be instantiated in the environment. The robot is initialized at position [0, 0]. The robot's target position is set in the opposite corner of the environment at [15, 15]. This maximizes the chance of collision between the robot and the randomly initialized obstacles since the robot needs to transverse the 15 by 15 grid to get to the target at the farthest corner.

The motion of the robot within the defined environment was dictated by the MVO algorithm. The obstacle avoidance algorithm works by calculating the distance of each obstacle in the environment and then determining the direction of the obstacle relative to the current position of the robot. If the distance between the robot and the obstacle is less than the sum of their velocity there is a chance of collision if the robot moves in the direction of the obstacle hence the robot is programmed to avoid moving in the direction of the obstacle if the sum of the obstacles and robot's velocity is less than the distance between them, therefore, avoiding collision with the obstacle. This process is repeated for each obstacle before deciding on the next position for each iteration. In addition to obstacle avoidance, the algorithm also prioritizes motion in the direction that brings the robot's position closer to the target position and uses the Euclidean distance between each obstacle and the robot and determines if the obstacle can reach the robot in the current step. The robot's ability to move in each direction is then checked, along with the position of each obstacle in relation to the robot. Then,it decides on

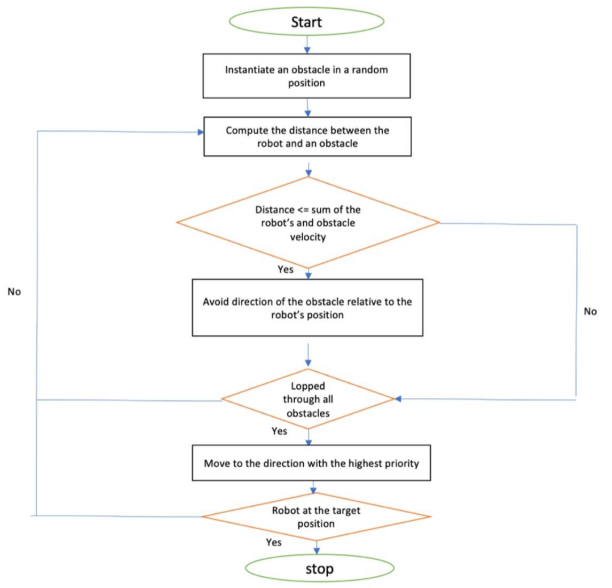

Fig. 3. flow chart showing the algorithm that defines the behaviour of the robot in its environment

the direction to move based on the current position of the robot and the distance to the target. It then updates the new position of the robot, the distance between the robot and each obstacle, was calculated using the formula:

$$d = \sqrt{(x(\text{robot}) - x(\text{obstacle}))^2 + (y(\text{robot}) - y(\text{obstacle}))^2}$$

In Figure 3, the simulation is designed to run a loop until the robot reaches its target destination. During each iteration of the simulation, each agent, including the obstacle and the robot, takes a step forward in the environment. This process continues until the robot successfully reaches its target location.

IV. Results and Discussions

The software implementing the simulation was designed in such a way that it could only run one simulation at a time. After each simulation, an animation shows the step-by-step movement of the agents during the simulation run.

Figure 4 shows a series of 4 screenshots showing dynamic obstacle avoidance.The simulation parameters in Fig. 4 are:
a) Robot velocity $= 1$,
b) Obstacle velocity $= 2$,
c) Number of obstacles $= 3$.

Fig. 5 shows a series of 4 screenshots showing dynamic obstacle avoidance. The simulation parameters in Fig. 4 are:
a) Robot velocity $= 1$,
b) Obstacle velocity $= 2$,
c) Number of obstacles $= 5$.

In the second test shown in Fig. 5 with five obstacles, the results differ slightly as compared to those obtained in the first test in Fig. 4. More specifically, there was one collision within the 30 iterations of the simulation time that took to get the robot to the target. In the first test with three obstacles, there was no collision in the 29 iterations, which the robot took

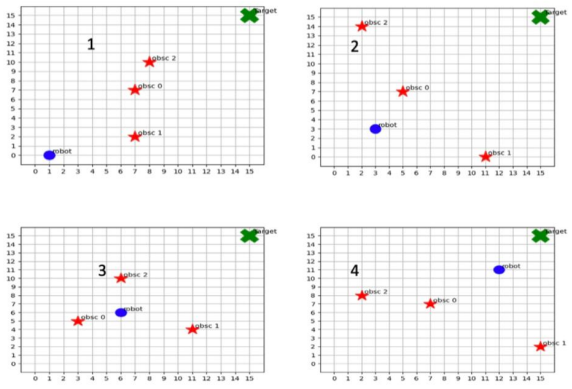

Fig. 4. A sequence of screenshots of the simulation environment demonstrating the dynamic obstacle avoidance with three obstacles.

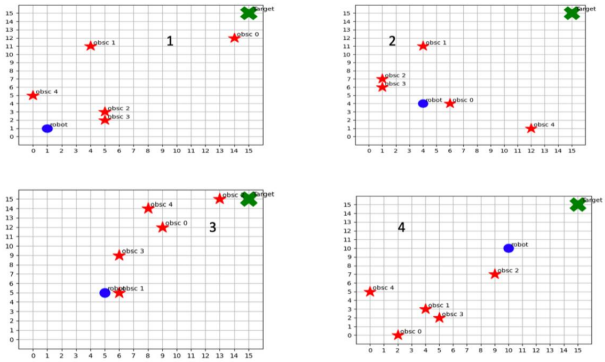

Fig. 5. A sequence of screenshots of the simulation environment demonstrating the dynamic obstacle avoidance with five obstacles.

in the simulation to reach its goal.The simulation performance indicators offer crucial data for assessing the MVO algorithm's efficacy and efficiency.The number of steps required for the robot to reach the destination without running into obstacles is shown by the step cost metric. Better performance is shown by lower step cost values since the robot can reach the target with fewer steps. The simulation performance metrics offer a quantitative framework for assessing and contrasting various iterations of the same method.By modifying the parameters and observing changes in the step cost in addition to other metrics, researchers can enhance the algorithm's performance in dynamic settings with differing amounts of obstacles.This optimization technique enables better effectiveness and flexibility.

A. Simulation Performance Metrics

To better understand the performance of the implemented MVO algorithm, a summary of the output was generated from the simulation, compiled from the 60 independent simulation

TABLE I
CORRELATION COEFFICIENT OF THE SIMULATION INPUTS VS. THE PERFORMANCE METRICS

Data Series under Investigation	Pearson Correlation coefficient
No. of obstacles, step cost	0.4644
No. of obstacles, No. of collisions	0.3880
Obstacles' velocity, step cost	0.4948
Obstacles' velocity, No. of collisions	0.0752

runs. The step cost refers to the number of steps it takes the robot to get to the target without colliding with the obstacles. The following properties of the MVO algorithm can be deduced from the simulation run data:

1) Average error

$$\% \; error = \frac{no.of \;\; sim \;\; runs \;\; with \;\; collisions}{no.of \;\; sim \;\; runs} \times 100$$
$$\% \; error = \frac{17}{60} \times 100 = 28.3\% \quad (1)$$

This means the algorithm has a 71.7% accuracy in avoiding dynamic obstacles.

2) The maximum error in a single simulation

$$\% \; error \; (single \;\; sim) = \frac{no.of \;\; collisions}{sim's \;\; step \;\; cost} \times 100$$
$$\% \; error \; (single \;\; sim) = \frac{4}{32} \times 100 = 12.5\% \quad (2)$$

3) Correlation coefficient of the number of collisions compared to the step cost

B. Correlation between Simulation Metrics

Table I shows the correlation coefficient comparing the simulation's input variables i.e. the number of obstacles vs. the obstacles' velocities and also the simulation performance metrics i.e., the step cost vs. the number of collisions per simulation iteration.

From the reported results, the following can be concluded:

1) No. of obstacles is directly proportional to the step cost.
2) No. of obstacles is directly proportional to the number of collisions.
3) The Obstacles' velocity is directly proportional to the step cost.

The correlation coefficient between the obstacles' velocity and the no. of collisions is quite small and is assumed to be negligible and thus the two factors are not proportional to each other.

C. Distribution of the Simulation Performance Metrics

From the boxplot of the 4 variables i.e. the no. of obstacles, s tep cost, no. of collisions, and obstacles' velocities shown in Fig. 8 revealed the following:

1) There is a maximum of 40 steps per iteration, a minimum of 20 steps per iteration, with an upper quartile of 31, a lower quartile 24 and a median of 26.5 steps.
2) There is a maximum of 2 collisions per iteration and a minimum of 0 collisions per iteration, with an upper quartile of 1, a lower quartile 0 and a median of 0 collisions.

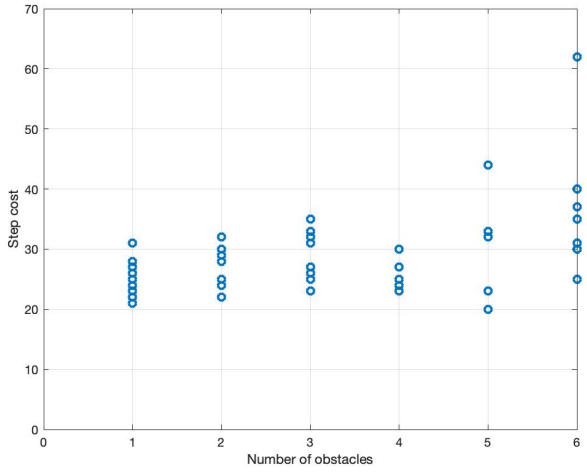

Fig. 6. Scatterplot showing the correlation between the inputs Number of obstacles compared to with the step cost over 60 simulation iterations

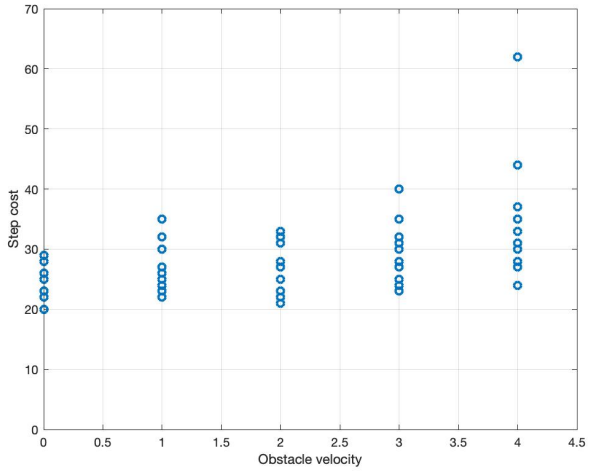

Fig. 7. Scatterplot showing the correlation between the inputs Number of obstacle's velocity compared to with step cost over 60 simulation iterations

3) There is a maximum of 5 obstacles velocity per iteration and a minimum of 1 obstacles velocity per iteration, with an upper quartile of 3, a lower quartile 2 and a median of 3.5 obstacles velocity.
4) There is a maximum of 8 obstacles per iteration and a minimum of 2 obstacles per iteration, with an upper quartile of 5, a lower quartile 3 and a median of 4.5 obstacles.

The two cases with three and four collisions in a single iteration were considered as statistical outliers during the analysis.

V. CONCLUSION AND FUTURE WORK

This study showcases the implementation of dynamic obstacle avoidance in a grid-based environment by developing a mobile robot capable of avoiding dynamic obstacles within

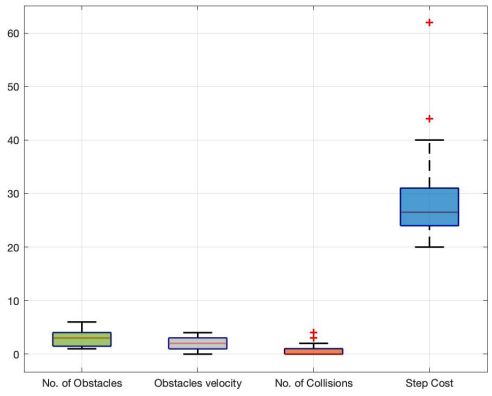

Fig. 8. Boxplot of performance metrics in 60 independent simulation runs.

a 15x15 grid with an accuracy of 71.7. The effectiveness of the MVO (Maximum Velocity Obstacle) algorithm decreases with an increase in the obstacle's speed and a slight increase in the number of obstacles in the simulation environment. Unfortunately, the program often crashes when dealing with a larger number of obstacles, indicating that the algorithm is not optimized for environments with a high potential for collision.Future improvements to the simulation could lead to better efficiency in terms of time or step cost by modifying the robot's movement rules. Currently, the robot can move in four directions (90-degree steps), but expanding its range to include eight directions (45-degree angular steps) could result in shorter and more efficient paths. Adding obstacles with various sizes and shapes to the environment would further test the MVO algorithm's efficiency as it navigates around detours and optimizes its route.Optimizing the MVO algorithm to better adapt to changes in moving obstacles' direction or velocity, as well as a dynamic environment, could enhance its performance. To achieve this, the MVO algorithm could be integrated with other navigation algorithms or augmented with an artificial neural network to observe the robot's behavior in unpredictable environments. This approach would allow for a more flexible and efficient navigation response, ultimately contributing to a more dynamic and effective obstacle avoidance algorithm.

ACKNOWLEDGMENT

MA thanks King Khalid University and Saudi Arabia Cultural Bureau in the UK, London for supporting this research. SD was partially supported by the ERDF Deep Digital Cornwall project number: 05R18P02820.

REFERENCES

[1] C. H. Everett, "Survey of collision avoidance and ranging sensors for mobile robots," *Robotics and Autonomous Systems*, vol. 5, no. 1, pp. 5–67, 1989.

[2] S. Huang, R. S. H. Teo, and K. K. Tan, "Collision avoidance of multi unmanned aerial vehicles: A review," *Annual Reviews in Control*, vol. 48, pp. 147–164, 2019.

[3] D. Q. Bao and I. Zelinka, "Obstacle avoidance for swarm robot based on self-organizing migrating algorithm," *Procedia Computer Science*, vol. 150, pp. 425–432, 2019.

[4] R. Sharma and D. Ghose, "Collision avoidance between uav clusters using swarm intelligence techniques," *International Journal of Systems Science*, vol. 40, no. 5, pp. 521–538, 2009.

[5] S. Na, H. Niu, B. Lennox, and F. Arvin, "Bio-inspired collision avoidance in swarm systems via deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 3, pp. 2511–2526, 2022.

[6] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, 1987, pp. 25–34.

[7] R. G. Braga, R. C. Da Silva, A. C. Ramos, and F. Mora-Camino, "Collision avoidance based on reynolds rules: a case study using quadrotors," in *Information Technology-New Generations: 14th International Conference on Information Technology*. Springer, 2018, pp. 773–780.

[8] Q. B. Diep, T. C. Truong, S. Das, and I. Zelinka, "Self-organizing migrating algorithm with narrowing search space strategy for robot path planning," *Applied Soft Computing*, vol. 116, p. 108270, 2022.

[9] X. Chen and Y. Li, "Smooth path planning of a mobile robot using stochastic particle swarm optimization," in *2006 International Conference on Mechatronics and Automation*. IEEE, 2006, pp. 1722–1727.

[10] J. Vaščák, "Adaptation of fuzzy cognitive maps by migration algorithms," *Kybernetes*, 2012.

[11] M. K. Muni, P. K. Pradhan, P. R. Dhal, S. Kumar, R. Sethi, and S. K. Patra, "Improving navigational parameters and control of autonomous robot using hybrid soma–pso technique," *Evolutionary Intelligence*, pp. 1–17, 2023.

[12] X. Huang, X. Dong, J. Ma, K. Liu, S. Ahmed, J. Lin, and B. Qiu, "The improved a* obstacle avoidance algorithm for the plant protection uav with millimeter wave radar and monocular camera data fusion," *Remote Sensing*, vol. 13, no. 17, p. 3364, 2021.

[13] H. Mousazadeh, H. Jafarbiglu, H. Abdolmaleki, E. Omrani, F. Monhaseri, M.-r. Abdollahzadeh, A. Mohammadi-Aghdam, A. Kiapei, Y. Salmani-Zakaria, and A. Makhsoos, "Developing a navigation, guidance and obstacle avoidance algorithm for an unmanned surface vehicle (usv) by algorithms fusion," *Ocean Engineering*, vol. 159, pp. 56–65, 2018.

[14] S. Campbell, W. Naeem, and G. W. Irwin, "A review on improving the autonomy of unmanned surface vehicles through intelligent collision avoidance manoeuvres," *Annual Reviews in Control*, vol. 36, no. 2, pp. 267–283, 2012.

[15] W. Zhang, S. Wei, Y. Teng, J. Zhang, X. Wang, and Z. Yan, "Dynamic obstacle avoidance for unmanned underwater vehicles based on an improved velocity obstacle method," *Sensors*, vol. 17, no. 12, p. 2742, 2017.

[16] C. Lin, H. Wang, J. Yuan, D. Yu, and C. Li, "An improved recurrent neural network for unmanned underwater vehicle online obstacle avoidance," *Ocean Engineering*, vol. 189, p. 106327, 2019.

[17] B. Braginsky and H. Guterman, "Obstacle avoidance approaches for autonomous underwater vehicle: Simulation and experimental results," *IEEE Journal of Oceanic Engineering*, vol. 41, no. 4, pp. 882–892, 2016.

[18] R. Kot, "Review of collision avoidance and path planning algorithms used in autonomous underwater vehicles," *Electronics*, vol. 11, no. 15, p. 2301, 2022.

[19] A. H. Vo, H. K. Yoon, J. Ryu, and T. Jin, "Modified a* algorithm for obstacle avoidance for unmanned surface vehicle," *Journal of Ocean Engineering and Technology*, vol. 33, no. 6, pp. 510–517, 2019.

[20] T. Sai Abhishek, D. Schilberg, and A. Selvakumar Arockia Doss, "Obstacle avoidance algorithms: A review," in *Materials Science and Engineering Conference Series*, vol. 1012, no. 1, 2021, p. 012052.

[21] M. Zhu, C. Xiao, S. Gu, Z. Du, and Y. Wen, "A circle grid-based approach for obstacle avoidance motion planning of unmanned surface vehicles," *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, vol. 237, no. 1, pp. 132–152, 2023.

[22] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[23] M. Ali and S. Das, "Swarms of mobile robots for area exploration," in *2023 Sixth International Conference of Women in Data Science at Prince Sultan University (WiDS PSU)*. IEEE, 2023, pp. 138–143.