ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 9, No. 2, 2024

STUDY OF PATHFINDING APPROACH BASED ON A* WITH ADAPTIVE OCCUPANCY GRID

Oleh Sinkevych, Yaroslav Boyko, Bohdan Sokolovskyy, Oleksandr Rechynskyi

Ivan Franko National University of Lviv, 50, Drahomanova Str, Lviv, 79005, Ukraine.
Authors' e-mail*: oleh.sinkevych@lnu.edu.ua, yaroslav.boyko@lnu.edu.ua, bohdan.sokolovskyy@lnu.edu.ua, oleksandr.rechynskyi@lnu.edu.ua*

https://doi.org/10.23939/acps2024.02.095

Submitted on 30.09.2024

***Abstract*: This paper presents the results of a study on the A* search algorithm applied to a two-dimensional map with obstacles. Since, in typical cases, A* is implemented on a map divided into cells of equal size, a scientific interest has lies in investigating the efficiency of this algorithm on a map with dynamically variable cell size. Such a map representation increases the "resolution" of constructing a better trajectory near obstacles. For this purpose, the paper proposes an approach to representing the search space as a dynamic adaptive grid using a QuadTree structure. Additionally, a modification of the A* algorithm has been proposed and investigated, which involves selecting the best cell in the neighborhood of the agent's current position and performing pathfinding from a starting point to a goal. The paper considers maps of various sizes and complexities for numerical experiments and compares the classical and modified A* algorithms. It has been shown that the proposed modification of the A* algorithm demonstrates better computational properties than the classical version of the algorithm on an adaptive grid.**

***Index Terms*: A*, Pathfinding, Grid map, Graph algorithms, Adaptive occupancy grid.**

I. INTRODUCTION

The A* algorithm, a widely used pathfinding algorithm in artificial intelligence and robotics, is commonly implemented on maps with uniformly sized cells. While this approach has proven effectiveness in many applications, it can be suboptimal in scenarios where the environment exhibits varying levels of complexity. For instance, in environments with dense obstacles or areas requiring high-precision navigation, a fixed-size grid may not adequately capture the underlying spatial variations [1, 2]. Several studies have explored the use of adaptive grid structures to address this limitation. These structures allow for dynamic adjustments in cell size, enabling more efficient exploration and representation of complex environments. For example, [3] proposed an approach that improves the A* algorithm using hexagonal grid mapping by addressing limitations such as low degrees of freedom, inadequate consideration of particular regions, and excessive nodes and turns. Also, the agent that searches the map to reach the target using A* can modify the grid resolution depending on the complexity of the environment.

In this paper, we extend the range of research by introducing the adaptive grid representation based on the QuadTree data structure. The QuadTree allows for an efficient hierarchical subdivision of the search space, enabling the algorithm to focus computational resources on regions with higher levels of details. We propose a modified A* algorithm incorporating the adaptive grid structure to improve the pathfinding performance in complex environments.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

The A* pathfinding algorithm in many real-world problems is viral and extensive [4]. Its popularity is based on relative simplicity and comprehensive research results. Nevertheless, due to significant computational load and issues with real-time use, the A* algorithm needs modifications like D* and D* Lite to enhance its applicability. For instance, in [5], authors present a comparative analysis of the A* and D* Lite algorithms for UV (uncrewed vehicle) path planning through simulation and experimentation, highlighting that while D* Lite generally outperforms A* in terms of faster computation time and shorter path generation, the choice between algorithms ultimately depends on specific system requirements. The paper [6] focuses on path planning for a mobile robot using a grid map, introducing several A* modifications and improvements aimed at enhancing computational time and path optimality. An interesting approach is presented in [7], where an A*-based zigzag global planner for a novel self-reconfigurable Tetris-inspired cleaning robot is designed to optimize coverage path planning in complex environments. The study [8] addresses the challenges of mapping, localization, and navigation for an autonomous mobile robot designed to assist the elderly by using RTABMap for mapping and localization, A* for path planning, and the Dynamic-Window Approach and cost map algorithms for obstacle avoidance, enabling the robot to navigate and adapt to new environments successfully.

Our study aims to extend the basic A* algorithm for the proposed adaptive occupancy grid maps and to compare their performance with the naïve implemen-

tation. In order to explore these ideas, different 2D maps with obstacles are considered. The agent (virtual robot), which is initialized at some starting point related to a specific node, should reach the predefined target dynamically, i.e., it scans the map in a short range around itself, detects obstacles, and determines the direction of movement using A* and the history of movement points. The robot can dynamically dense the grid by splitting nodes into four smaller ones. These nodes are organized in a QuadTree [9] structure to increase computational performance. Here, we set a restriction to force the robot to move only in neighborhood nodes if possible. If not, the robot picks a random optimal position from the A* set of movements.

III. SCOPE OF WORK AND OBJECTIVES

In this paper, we consider two-dimensional grid maps with multiple polygonal obstacles. The map is initially split into rectangular nodes of predefined size $r_b = \left(r_x, r_y\right)$ according to axes X and Y (fig. 1, 2). The agent, initialized in the starting coordinates $\left(x_0, y_0\right)$, is mapped to one of the nodes that hold these coordinates. Each *j-th* node n^j is defined via upper left corner coordinates, width and height $n^j = n^j\left(x^j, y^j, w, h\right)$. The agent's *task* is to reach some target node n^T which holds coordinates $\left(x_t, y_t\right)$ of the target. It does not know about the whole map in advance.

Before each agent's step, it runs the A* algorithm to obtain a new optimal node to relocate based on information about surrounding nodes. These nodes can have three statuses: *occupied* when obstacles are inside, *accessible* and *unknown* so that the agent can move only to accessible ones. The agent scans the map around itself at a certain distance *s*, significantly less than the map size. It allows him to obtain the nodes' statuses. If surrounding nodes are either occupied or unknown, the agent can split them into four smaller nodes in a recursive way using the QuadTree data structure

$$n^j \rightarrow \left[n^{j,1}, n^{j,2}, n^{j,3}, n^{j,4}\right], \qquad (1)$$

which forms a parent-children tree. These splits can be performed until the node's size reaches some predefined limit value $r_l = \left(r_l, r_l\right)$. Note that the node's size cannot be less than the agent's size because the agent is positioned within the node.

Such splits allow the agent to build a more flexible and granular trajectory. For instance, if all the nodes around it are occupied, then the agent can split nodes and find those that are accessible (Fig. 3). Note that the agent can perform splitting only within its scanning range *s*.

Based on the proposed dynamic map representation, the agent utilizes the standard and modified A* algorithms to calculate the *quasy*-optimal movement points.

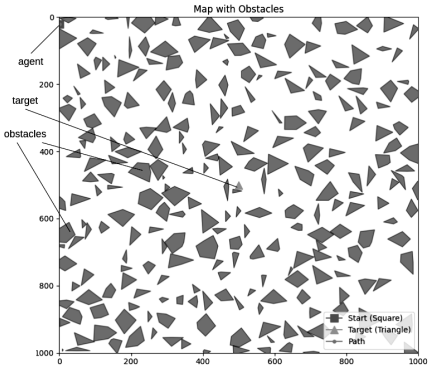


Fig. 1. Map representation

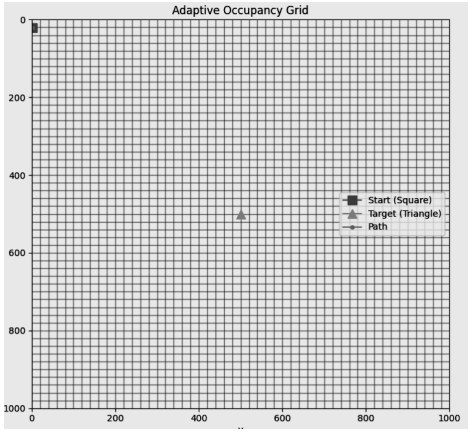


Fig. 2. Map represented as adaptive grid

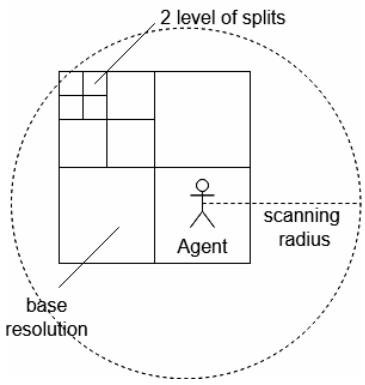


Fig. 3. Examples of map splitting

Because the performance of A* is not widely studied on such adaptive occupancy grids, we can formulate the *objective* of the study. Given the 2D map with the corresponding adaptive occupancy grid, the search agent initialized in starting point $\left(x_0, y_0\right)$ must compute sub-optimal path $\left\{n^0, n^1, \ldots, n^T\right\}$ in order to reach the target node n^T defined by coordinates $\left(x_t, y_t\right)$ using the standard and modified A* algorithms. The study's *primary goal* is to research and compare the performance of the standard and modified versions of the A* (discussed in the next section) and provide an analysis of the obtained results.

IV. PATHFINDING ALGORITHMS

Using the proposed adaptive occupancy grid and basic A*, it is possible to implement the latter modification to approximate the real-time search algorithm with the priority of moving only to the nearest nodes.

Firstly, let us briefly outline the main steps of standard A* adjusted to the proposed adaptive occupancy grid. A* is a popular pathfinding algorithm that efficiently finds the shortest path between a starting node and a goal node in a weighted graph. The algorithm utilizes two special sets: *open* and *closed*. The first is a set of nodes that have been discovered but have yet to be explored, and the latter is a set of nodes that have already been explored. Each *j-th* node is assigned three functions $g\left(n^j\right)$, $h\left(n^j\right)$ and $f\left(n^j\right)=g\left(n^j,n^k\right)+h\left(n^j\right)$. Here $g\left(n^j,n^k\right)$ is the cost of the path from the start node to node n^j and $h\left(n^j\right)$ is the heuristic estimate of the cost of the cheapest path from node n^j to the target node. In this study, we have chosen functions as follows

$$g\left(n^j,n^k\right)=\sqrt{\left(x_c^j-x_c^k\right)+\left(y_c^j-y_c^k\right)}+\sum_{i=1}^{j-1}g\left(n^i,n^{i-1}\right), \quad (2)$$

$$h\left(n^j\right)=\sqrt{2}\min\left(cdx^1+cdy^1,cdx^2+cdy^2\right), \quad (3)$$

where $g\left(n^j,n^k\right)$ is the Euclidean distance, $\left(x_c^k,y_c^k\right)$ is the center of node n^k, $\left(x_c^j,y_c^j\right)$ is the center of node n^j. The choice of the second function is more complicated and took place on the basis of conducted numerical experiments. It is a modified *diagonal distance*, where cdx^1,cdy^1,cdx^2 and cdy^2 are the diagonal distances from each corner of the current node to the center of the goal node. They are calculated as

$$\boldsymbol{cd}=\left|\boldsymbol{n}_c^j-\boldsymbol{g}_c+\boldsymbol{n}_c/2\right|, \quad (4)$$

where $\boldsymbol{n}_c^j$ is the center coordinates of n^j, $\boldsymbol{g}_c$ is the center coordinates of the target node, $\boldsymbol{n}_c$ is the size of node n^j.

The steps of A* are straightforward [10]. Firstly, during the initialization, the start node n^0 is set as the current node and is simultaneously added to the open set, and all functions are set to zero. Secondly, we remove the current node from the open set and add it to the closed set. For each neighbor of the current node:

a) *if* the neighbor is in the closed set, we ignore it;

b) *if* the neighbor is not in the open set, we add it to the open set, set its parent to the current node, and calculate its h, g, and f values;

c) *if* the neighbor is already in the open set, we check if this path to the neighbor is better than the previous one based on f value; if so, we update the neighbor's parent and recalculate its f, g, and h scores.

Algorithm 1 Mod-A* (A*modification)

Require: $openSet=\{(f,startNode)\}, startNode$
Require: $goalNode$, $finished=False$
Require: $history=\{startNode\}$
  $currentNode \leftarrow startNode$
  if $openSet=\emptyset$ and not $finished$ **then**
    $currentNode \leftarrow history.pop()$
  end if
  if $finished$ **then**
    return $currentNode$
  end if
  $agent\_node \leftarrow agent.get\_current\_node()$
  $adjacent\_leaves \leftarrow agent.get\_adjacent\_leaves(agent\_node)$
  $best\_node \leftarrow$ None
  $best\_priority \leftarrow \infty$
  for (priority, node) in $openSet$ **do**
    if node ∈ adjacent_leaves and priority < best_priority **then**
      $best\_priority \leftarrow$ priority
      $best\_node \leftarrow$ node
    end if
  end for
  if $bestNode$ **then**
    $openSet.pop(bestNode)$
    $currentNode = bestNode$
  else
    $unoccupied\_adjacent\_leaves \leftarrow adjacent\_leaves|free$
    $currentNode \leftarrow random(unoccupied\_adjacent\_leaves)$
    $openSet.pop(currentNode)$
  end if
  if $currentNode == goalNode$ **then**
    $finished \leftarrow True$ **return** $currentNode$
  end if
  $history \leftarrow currentNode$
  $leaves \leftarrow agent.get\_adjacent\_leaves(currentNode)$
  for neighbor in leaves **do**
    if neighbor.is_occupied() or not neighbor.is_leaf() **then**
      continue
    end if
    $g\_score \leftarrow currentNode.g + g(currentNode, neighbor)$
    if $g\_score < neighbor.g$ **then**
      $neighbor.g \leftarrow g\_score$
      $neighbor.h \leftarrow h(neighbor, goalNode)$
      $neighbor.f \leftarrow neighbor.g + neighbor.h$
      if $neighbor \notin openSet$ **then**
        $openSet \leftarrow (neighbor.f, neighbor)$
      else
        for item in $openSet$ **do**
          if item[1] == neighbor **then**
            $openSet.pop(item[1])$
            $openSet \leftarrow (neighbor, neighbor.f)$
            break
          end if
        end for
      end if
    end if
  end for
  return $currentNode$

*Fig.4. Pseudocode of Mod-A**

The second stage runs until the target node is found, but with a few considerations applied in this work. Because of the second A*, the agent performs two actions: 1) it scans the surrounding area and detects nodes of three types described in the previous section; b) the agent can split nodes with *occupied or unknown* statuses into four children nodes, but only if the size of the minimum node does not reach the minimal resolution $r_l=\left(r_l,r_l\right)$. To identify obstacles within the *j-th* node n^j, the agent verifies if the intersection of the scanning circle and the obstacle defined as a polygon occurs. If so, the corresponding node is marked as *occupied*. As in real-world scenarios, the agent should move only within the neighboring area and not *'jump'* over the map to the optimal node far away from the current node n^j, we propose the following modification of A* algorithm.

Firstly, the next optimal node for the movement from the current position should be picked up from the adjacent leaves. If there are no such optimal nodes, i.e., all f-values of adjacent leaves are worse than others, then a random node from *free* adjacent leaves is to be taken. This step introduces randomness, which helps avoid getting stuck. Secondly, we do not utilize the closed set used in the standard A* because we want the agent to re-explore already visited nodes. The one iteration of the proposed modification of the algorithm named *Mod-A** is given in the pseudocode (Fig. 4).

V. NUMERICAL EXPERIMENTS

To investigate and compare the performance of the standard and proposed modified A* algorithms, we generated two 2D maps with different sizes and difficulty levels.

The first map, #1, is of size 500x500. It has 200 polygonal obstacles with a maximum side length of 20. The agent can scan the map in the range of 10, and the agent's diameter is 1. The base resolution r_b is $(10,10)$ and the smallest resolution is $r_l = (2,2)$. The results of the standard and modified algorithms are shown in Fig. 5-6, where starting coordinates are $(1.2,1.2)$ and target coordinates are $(430.2,477.2)$.

As it is seen in Fig. 5-6, the agent splits surrounding nodes near obstacles to build a more refined path. Here, the standard A* algorithm is used with the closed set, i.e., the agent is forbidden to revisit explored nodes. As it is an *exploration* path, it may have detours and local circles. The refined path is smoother than the explored path; however, we have omitted it here due to the paper's limited size.

The modified *Mod-A** version (Fig. 7-8) produces similar results but with one significant improvement: the length of the explored path in all three experiments, where targets occupied coordinates $(430.2,477.2)$, $(430.2,35.2)$ and $(10.2,490.3)$ is less than the length of the standard A* implementation (Table 1).

The results of three experiments with different target's coordinates are presented in Table 1. *Mod-A** generates a reasonably shorter exploration path in each experiment, which is much better, for instance, for the agent's power savings. Nevertheless, as the agent can revisit nodes, the revised shortest paths built over the explored path are larger than the paths generated via A*.

Table 1

Path length for A* and Mod-A* (map # 1)

A*		Mod-A*	
Explored length	Shortest length	Explored length	Shortest length
1277	810	1112	817
1175	499	952	490
3311	589	1380	641

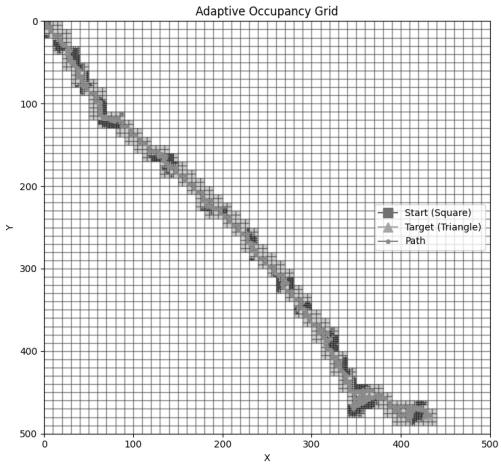


Fig. 5. Agent's adaptive grid (A)*

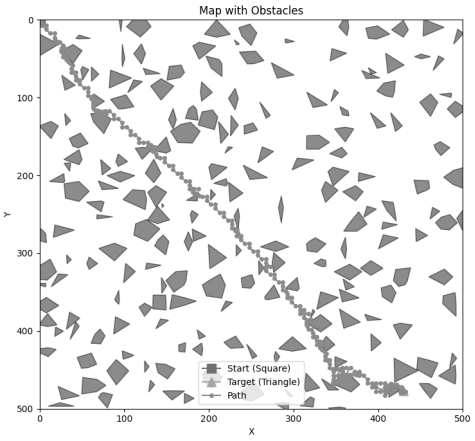


Fig. 6. Agent's exploration route (A)*

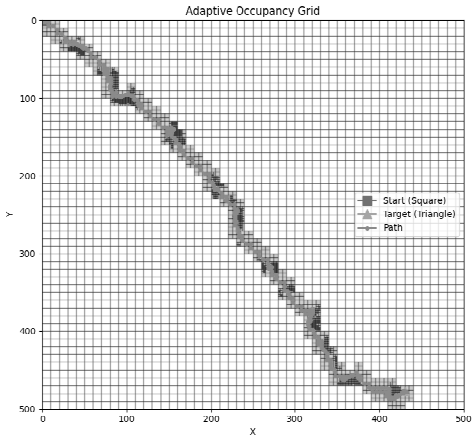


Fig. 7. Agent's adaptive grid (Mod-A)*

The second map, #2, is twice as extensive and is of size 1000x1000. It consists of 200 obstacles with a maximum side length of 45. The scanning range of the agent is the same as for map #1. The results for this configuration of the standard and modified algorithms, where target coordinates are $(970.2,820.7)$, are visualized in Fig. 9-12.

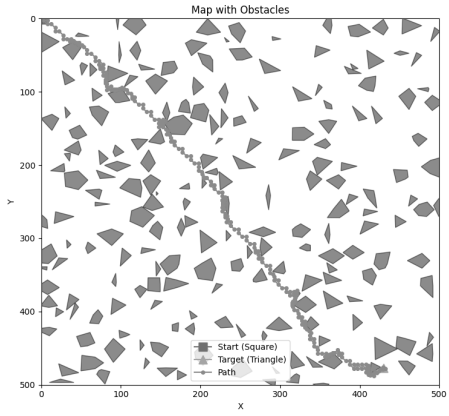


Fig. 8. Agent's exploration route (Mod-A)*

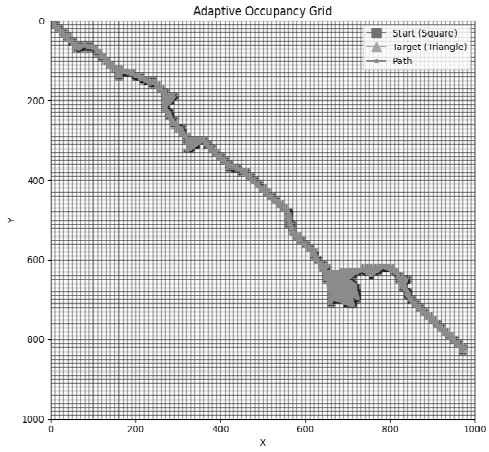


Fig. 9. Agent's adaptive grid (A)*

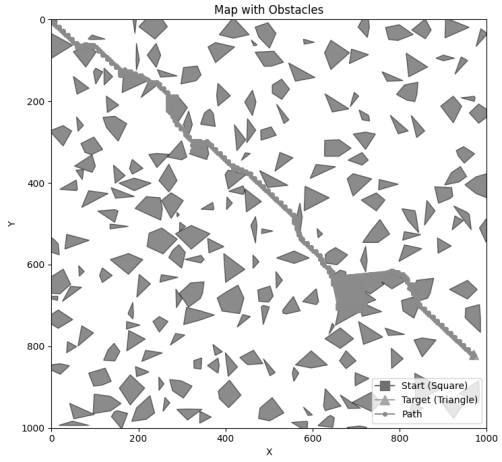


Fig. 10. Agent's exploration route (A)*

Table 2

Path length for A* and Mod-A* (map # 2)

A*		Mod-A*	
Explored length	Shortest length	Explored length	Shortest length
8644	1684	2100	1620
22156	1172	6064	1130
6486	1016	4193	1043

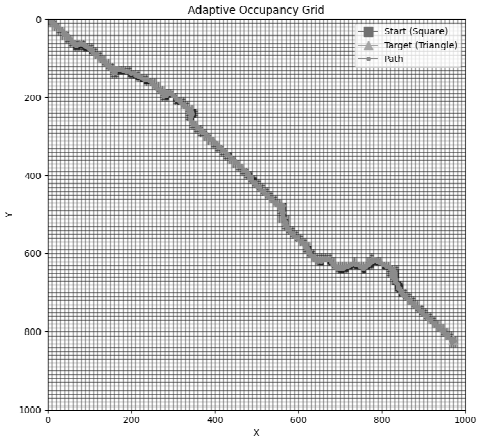


Fig. 11. Agent's adaptive grid (Mod-A)*

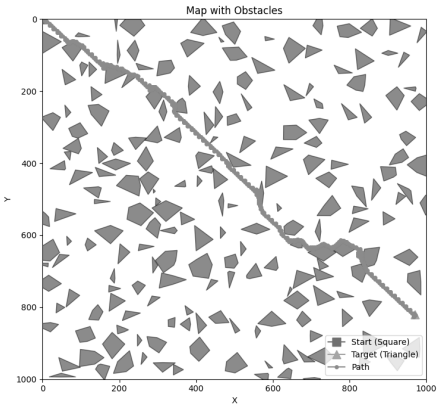


Fig. 12. Agent's exploration route (Mod-A)*

The results of the other three experiments done for map # 2, also with different target coordinates $(970.2, 820.7)$, $(970.2, 35.3)$ and $(90.2, 820.7)$ correspondingly, are presented in Table 2. With the increase in map complexity and size, *Mod-A** generates a much shorter exploration path in each experiment. Besides, the revised shortest paths built over the explored path are shorter in two of three experiments.

The conducted numerical modeling demonstrates a few intriguing outcomes. Firstly, the dynamic splitting of the searching grid impacts the A* implementation and increases the complexity. Although this modification complicates the search for the shortest path and leads to constructing a quasi-optimal trajectory, it allows more flexible processing of the map and the environment closest to the agent. Secondly, the proposed *Mod-A** modification with slight randomness, focusing only on the neighboring nodes and skipped closed set, outperforms standard A* under a given configuration.

VI. CONCLUSION

In this paper, we presented and explored an integrated approach to the dynamic pathfinding problem. As part of this approach, a scheme for combining an adaptive occupancy grid with a variable resolution,

which is superimposed on the map, and a modification of the A* algorithm was proposed.

The software agent, whose goal was to build a path from the start point to the goal, moved along a two-dimensional map with dense obstacles. The agent can scan the surrounding space in some predefined range and, if necessary, thicken the grid within the reach of the scanning radius. Since in real-world tasks the agent must move only within the range of the scanning device, we modified the standard A* algorithm and compared it with the standard implementation.

It is shown that the proposed modification of A* named *Mod-A** in the context of the track length of map exploration is superior to the standard A* algorithm. Such a modification can serve as a basis for the implementation of a multi-agent pathfinding system in the conditions of complex maps.

Also, in the future, we plan to compare the proposed modification with the family of D* algorithms.

References

[1] X. Sun, S. Deng, B. Tong, S. Wang, C. Zhang, & Y. Jiang. "Hierarchical framework for mobile robots to effectively and autonomously explore unknown environments". ISA Trans., vol. 134, pp. 1–15, Sep. 2023. DOI: https://doi.org/10.1016/j.isatra.2022.09.005.

[2] H. Ryu. Hierarchical Path-Planning for Mobile Robots Using a Skeletonization-Informed Rapidly Exploring Random Tree*. Appl. Sci., vol. 10, no. 21, p. 7846, Nov. 2020. DOI: https://doi.org/10.3390/app10217846.

[3] Z. An, X. Rui, and C. Gao. Improved A* Navigation Path-Planning Algorithm Based on Hexagonal Grid. ISPRS Int. J. Geo-Inf., vol. 13, no. 5, p. 166, May 2024. DOI: https://doi.org/10.3390/ijgi13050166.

[4] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah, & E. Gunawan. A Systematic Literature Review of A* Pathfinding. Procedia Comput. Sci., vol. 179, pp. 507–514,. DOI: https://doi.org/10.1016/j.procs.2021.01.034.

[5] Y. D. Setiawan, P. S. Pratama, S. K. Jeong, V. H. Duy, & S. B. Kim. Experimental Comparison of A* and D* Lite Path Planning Algorithms for Differential Drive Automated Guided Vehicle. AETA 2013: Recent Advances in Electrical Engineering and Related Sciences. Berlin, Heidelberg: Springer Berl. Heidelb., 2014, pp. 555–564. DOI: https://doi.org/10.1007/978-3-642-41968-3_55.

[6] F. Duchoň et al. Path Planning with Modified a Star Algorithm for a Mobile Robot. Procedia Eng., vol. 96, pp. 59–69, 2014. DOI: https://doi.org/10.1016/j.proeng.2014.12.098.

[7] A. Le, V. Prabakaran, V. Sivanantham, & R. Mohan. Modified A-Star Algorithm for Efficient Coverage Path Planning in Tetris Inspired Self-Reconfigurable Robot with Integrated Laser Sensor. Sensors, vol. 18, no. 8, p. 2585, Aug. 2018. DOI: https://doi.org/10.3390/s18082585.

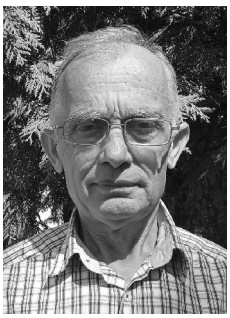
[8] Muhtadin, R. M. Zanuar, I. K. E. Purnama, & M. H. Purnomo. Autonomous Navigation and Obstacle Avoidance For Service Robot. 2019 Int. Conf. Comput. Eng., Netw., Intell. Multimedia (CENIM), Surabaya, Indonesia, Nov. 19–20, 2019. IEEE, 2019. DOI: https://doi.org/10.1109/cenim48368.2019.8973360.

[9] S. Sahni and D. P. Mehta. Handbook of Data Structures and Applications. Taylor Francis Group, 2018.

[10] R. Wang, Z. Lu, Y. Jin, and C. Liang. Application of A* algorithm in intelligent vehicle path planning. Math. Models Eng., Aug. 2022. DOI: https://doi.org/10.21595/mme.2022.22828.

Oleh Sinkevych, PhD, was born in Lviv, Ukraine, in 1988. Starting from 2023, he has been working as an Associate Professor at the Faculty of Electronics and Computer Technologies of Ivan Franko National University of Lviv. His research interests encompass machine learning, natural language processing, swarm AI algorithms, numerical optimization, and metaheuristics.

Yaroslav Boyko, PhD, was born in Lviv region, Ukraine, in 1967. Since 2017, he has been working as an Asssociate Professor at the Faculty of Electronics and Computer Technologies of Ivan Franko National University of Lviv. His research interests encompass Internet of Things and Fog/Edge computing.



Bohdan Sokolovskyy, PhD, was born in Kulykiv, Lviv region, Ukraine, in 1950. Since 2014, he has been working as an Associate Professor at the Faculty of Electronics and Computer Technologies of Ivan Franko National University of Lviv. His research interests encompass computer modeling of nonuniform semiconductor structures and methods of stochastic optimization.

Oleksandr Rechynskyi, was born in Kropyvnytskyi, Ukraine, in 2000. Starting from 2024, he has been studying as a PhD student at the Faculty of Electronics and Computer Technologies of Ivan Franko National University of Lviv. His research interests encompass machine learning, natural language processing, audio processing, swarm AI, and metaheuristics algorithms.