# Mobile Robot Navigation in Known Environment

**Alex Galčík, František Duchoň**

*Institute of Robotics and Cybernetics, Slovak University of Technology, Ilkovičova 3, 821 19 Bratislava, Slovak Republic (e-mail: frantisek.duchon@stuba.sk)*

**Abstract**: This article covers various aspects of mobile robot navigation in a known environment. In the article, we discuss different algorithms used in determining the robot's path in real time (global planner), along with the known advantages and disadvantages. The article also compares configuration spaces that are used in this type of navigation. For a better overview, we will use algorithms from the area of oriented and non-oriented global navigation, namely A * and Breadth First Search. We will also re-evaluate the modified A * algorithm in a known environment designed to avoid dynamic obstacles. At the end of the article, all these navigation methods are compared.

*Keywords*: navigation in a known environment, searching algorithms, response to a dynamic environment, A*, BFS

## 1. INTRODUCTION

Efficiently navigating a mobile robot through a known environment entails safely guiding it from point A to point B without encountering collisions. This requires employing various methodologies, including the utilization of diverse search algorithms capable of swiftly determining the optimal path for the robot. These algorithms are categorized based on their search methods into oriented and non-oriented global navigation approaches, both capable of adapting to dynamic environments. In addition to finding the optimal path, they must promptly adapt to environmental changes. In oriented navigation, the robot need not explore all potential pathways. Algorithms like Greedy, Dijkstra's algorithm, and A* (Figure 1) employ qualified heuristics or distance metrics to eliminate unnecessary paths, facilitating the identification of the optimal path (Elgabry et al., 2018).

Various configuration spaces, along with their modifications or simplifications, are crucial for ensuring the proper functioning of these algorithms (Figure 2). This diversity underscores the multifaceted nature of the pathfinding problem, which manifests in myriad similarities across different scenarios. Real-world applications of pathfinding exhibit preferences for specific circumstances, thus necessitating different algorithmic requirements. Consequently, it is imperative to conduct a thorough examination and comparison of a broad spectrum of algorithms to determine the most suitable ones for a given situation (Vinther et al., 2015). Moreover, there exist numerous limitations and potential errors in path determination that can impede successful completion. These may include misinterpretations of the environment or incorrect algorithmic choices, resulting in prolonged path times to reach the goal. Additionally, path completion may fail due to factors such as an unreachable goal or complications arising during path recalculation (Elgabry et al., 2018).

The paths generated by algorithms like A* or Breadth First Search may exhibit variations, both in their alignment and computational requirements. However, we will delve into these disparities further in their respective sections.

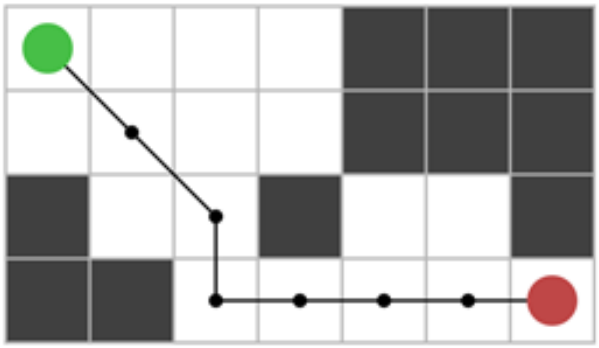

Fig. 1. The figure illustrates the path determined by the A* algorithm from point A to point B (Imms et al., 2016).

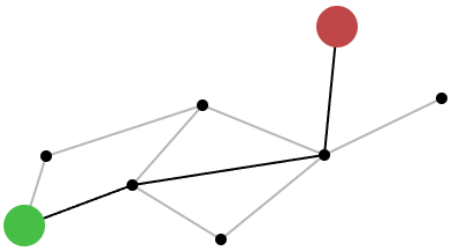

Fig. 2. The figure illustrates the path determined by A* algorithm in the graph type environment representation (Imms et al., 2016).

Global navigation has a lot of different applications. This means that the worlds and spaces used by this navigation have a wide range of rules and functions or specific features. It is therefore very important to consider them to achieve optimal solutions. However, certain environments necessitate simplification to facilitate the application of specific search algorithms (Vinther et al., 2015). One prevalent representation of such environments is **the metric or grid format** (Figure 3), which stands out as one of the most widely utilized due to its ease of implementation and rapid deployment.

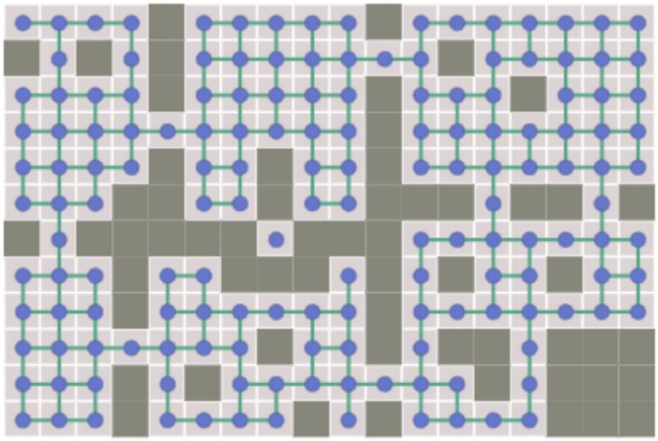

Fig. 3. A grid representation featuring a designated walking points within the environment. (Patel et al., 1997).

The metric format imposes directional constraints on the robot's movement, offering significant advantages in certain scenarios. However, compared to geometric representations where the robot is allowed to move in unrestricted angle, this limitation is often perceived as a notable drawback. Moreover, employing a metric representation for extensive environments poses challenges due to diminished detail retention. As the environment size increases, the map cells must shrink accordingly, leading to computational slowdowns (Goodwin and Menon, 2006). In contrast, geometric representations do not encounter this issue. However, they are constrained by the number of corners (representing obstacles), and to accommodate different computational algorithms, such environments necessitate simplification to a topological structure of graphs. This process is intricate as it requires mapping all visible corners (Vinther et al., 2015) (Figure 4).

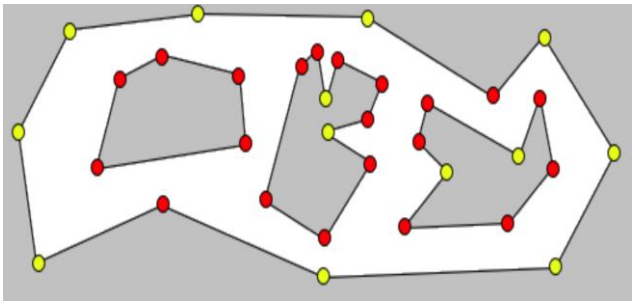

Fig. 4. Geometric representation of map (Vinther et al., 2015).

To preserve map details without compromising computational efficiency, simplification of the metric map can be employed. Initially, the entire map is partitioned into four cells, typically square in shape. Subsequently, any cell containing a combination of obstacles and free space undergoes further subdivision into four smaller cells. This recursive process continues until the cell size reaches the specified minimum threshold. An inherent advantage of this method called Quadtree (Figure 5) is the representation of large free areas as single cells. However, a drawback is the potential reduction in path quality to a certain extent, attributed to transitions through the centers of squares (Goodwin and Menon, 2006).

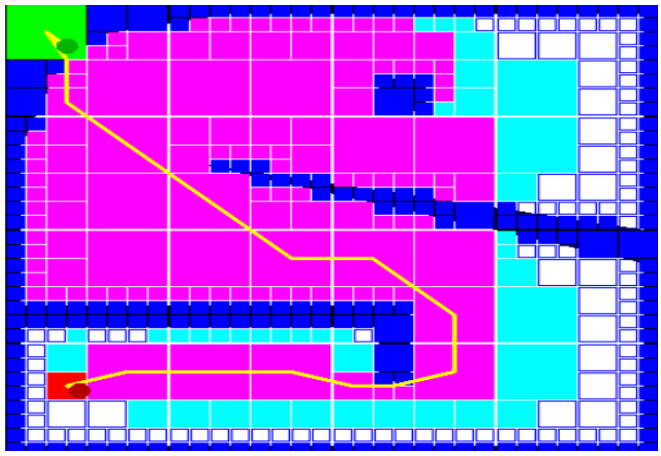

Fig. 5. Quadtree representation (Goodwin and Menon, 2006).

The metric representations can be further simplified into a **topological representation** using **graphs** (Figure 6). This approach introduces the concept of a topological map, which comprises interconnected nodes and edges within space. An edge serves to link two nodes in the environment, with each node representing a decision point between these edges. In confined spaces, a node corresponds to a small area delineated by gates that facilitate transitions between different path segments. Conversely, in extensive environments, a node is analogous to a graph node, interconnected with other nodes (Johnson and Kuipers et al., 2012). Various methodologies exist for simplifying environment representations, but generally, the fewer nodes (cells) present in the map, the faster chosen path finding algorithm operates (Patel et al., 1997).

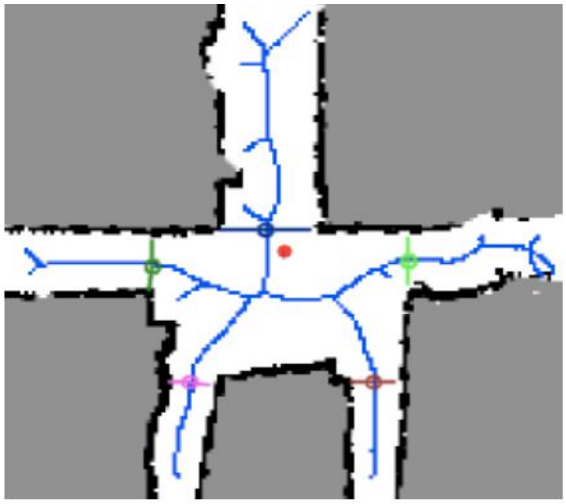

Fig. 6. A typical graph representation of the environment constructed based on the metric (grid) map (Johnson and Kuipers et al., 2012).

Another such simplification method involves converting a metric representation into waypoints on the map (Figure 7). This process determines the specific points where direction changes are necessary to navigate to another area, thereby eliminating unnecessary additional nodes between these points in the original map representation (Patel et al., 1997).

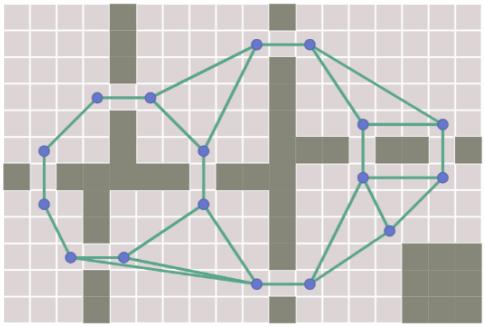

Fig. 7. Waypoints generated for the metric map (Patel et al., 1997).

Another approach is the navigation meshes approach, which changes the existing space into transitional areas of various shapes and sizes, thus reducing the number of graph nodes.

It's important to note hierarchical approaches that operate with distinct levels of graph representation. These levels could correspond to transitions between various spatial scales, for instance, at a street level in urban environments or at a city level for long-distance routes. Consequently, when determining points A and B, the selection of road levels dynamically adjusts based on the distance from the destination (Patel et al., 1997).

The algorithms employed for pathfinding typically address the challenge of determining the shortest path within the framework of graph theory. Their objective is to identify the optimal path between a given starting and ending point, guided by predefined criteria (Chopra et al., 2021). In the global navigation, these algorithms are categorized into two main groups: oriented and non-oriented.

**Non-oriented global navigation** operates by traversing between all nodes without specific directionality, akin to navigating through a complex maze with no predetermined path. In contrast, **oriented global navigation** employs predefined methods to evaluate neighboring nodes, guiding the movement towards the most heuristic-based solution (Graham and McGabe, 2003). One such algorithm is the A* algorithm, widely recognized and utilized within this domain for its efficiency in finding the shortest path between start and goal points in a relatively efficient manner. To implement A*, it is imperative to convert the configuration space into a set of nodes (Cui and Shi, 2011). Another oriented global navigation algorithm is Dijkstra's algorithm, renowned for its ability to identify the shortest path between a point and a goal without any negative costs within the graph. Unlike exhaustive search methods, Dijkstra's algorithm strategically selects which points to visit during pathfinding. It is extensively employed in GPS devices for route determination and finds widespread application in industrial contexts (Navone et al., 2020). While both algorithms are comparable, A* is often favored due to its adaptability. By simply adjusting its heuristic function, A* can emulate other pathfinding algorithms like Dijkstra's algorithm, Best First Search, Breadth First Search, or Depth First Search. Moreover, Dijkstra's algorithm is considered uninformed, meaning it doesn't require prior knowledge of the goal node. Consequently, it excels in scenarios where the node graph lacks prior information or where estimating distances to the goal is challenging. Certainly, the heuristics of Dijkstra's algorithm may not necessarily lead to finding a path to the goal, even if such a path exists—a scenario which cannot occur with the A* algorithm. Dijkstra's algorithm expands on a node only if the path appears promising, aiming solely to swiftly reach the goal node from the current node without attempting to visit every node along the way (Sukaj et al., 2020).

Conversely, there are **non-oriented global navigation algorithms**, such as Breadth First Search (BFS). This algorithm systematically explores paths by expanding to each neighboring node from the starting node and subsequently traversing to new neighboring nodes, traversing through all nodes and the paths connecting them. Akin to BFS, Depth First Search (DFS) explores nodes by delving deeper into the search space, as the name implies. While BFS searches nodes in layers, DFS explores nodes in depth (Graham and McGabe, 2003).

Given that the mentioned algorithms are typically tailored to function within static, unchanging environments, several strategies have been devised for their adaptation to **dynamic environments**, where they must navigate around objects with unpredictable trajectories. One such strategy involves programming a parallel algorithm capable of circumventing dynamic obstacles in real-time. Alternatively, path recalculation can be employed, essentially embedding dynamic objects into a static world. Here, at regular intervals or during the robot's movement around a reference point, the subsequent path is recalculated. However, this strategy is less suitable for long-distance navigation, as it may result in the loss of significant environmental data. To mitigate this issue, the approach of connecting paths is utilized. This method involves dividing the environment into multiple segments, and path recalculation occurs only when the robot approaches or enters one of these segments (Patel et al., 1997).

An alternative approach involves anticipating the movements of dynamic objects by predicting their trajectories. By considering the future positions of these dynamic objects, it becomes feasible to adjust the algorithm's functions accordingly. For instance, the A* algorithm, which incorporates a function f(n) to estimate the optimal travel cost from the current robot position to the goal, can be augmented by tracking the time required to traverse specific points in the environment. This time estimation is contingent upon the length of the path between the robot and the goal. Consequently, the algorithm can forecast when a particular point will be reached and predict the potential occupation by a dynamic object (obstacle), thus enabling the algorithm to avoid such obstacles (Silver et al., 2005).

This article offers a comparative analysis of two standard algorithms commonly used in mobile robot navigation within known environments: A* (oriented approach) and Breadth First Search (non-oriented approach). These algorithms are regarded as benchmarks in the field, and our aim is to underscore their performance concerning criteria such as path optimality and computation time. Additionally, a crucial aspect for mobile robots is their capability to navigate around moving (dynamic) obstacles. Our experiments primarily evaluate these algorithms based on path length while

considering various speeds and quantities of dynamic obstacles. While this research does not introduce novel methods or approaches to robot navigation, its significance lies in providing a comprehensive comparison. Such comparisons are invaluable for roboticists seeking to select and implement suitable navigation techniques. Moreover, they serve to validate the efficacy of these algorithms, particularly under more complex conditions than typically modeled, such as scenarios involving a substantial number of dynamic obstacles (e.g., 20 dynamic obstacles).

## 2. SIMULATION ENVIRONMENT

To assess the efficacy of the algorithms, we devised a comprehensive simulation framework incorporating various software tools such as Fusion 360, Blender, Substance Painter, Visual Studio, and the Unity 3D environment (Figure 8). Our evaluation focused on a metric configuration space featuring two pathfinding algorithms. The first algorithm employed was a non-oriented Breadth First Search, while the second was an oriented A* algorithm. The **first experiments** were designed to mimic a static maze. **Further experiments** were conducted solely on the modified A* method, with the environment modified to include moving obstacles. These obstacles were represented by objects possessing constant velocity. Upon collision with walls, these objects exhibit a reactionary behavior – they reverse their direction and pivot at a random angle before resuming their trajectory. Notably, these dynamic objects are constrained to rotate by only a small angle upon impact with the wall.

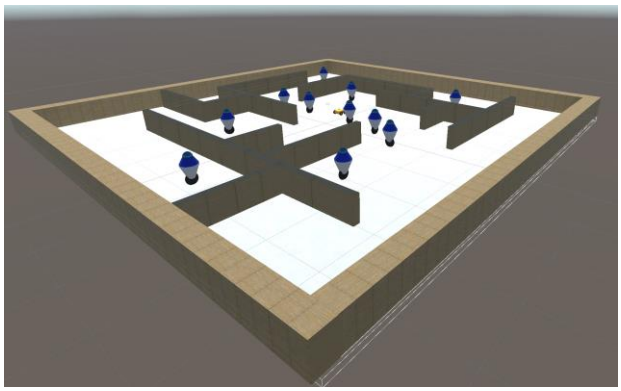

Fig. 8. Created simulation environment.

In the experiments involving **dynamic obstacles**, the robot initiates its activity from the center of the environment (maze). Upon setting the goal, the robot proceeds to recalculate the path using both the Breadth First Search (BFS) and A* algorithms. Subsequently, the calculated path is displayed on the screen. Upon simulation commencement, the robot adheres to the path specified by the A* algorithm, continuously navigating without interruption, endeavoring to avoid static or dynamic obstacles encountered along the way. Figure 9 illustrates the configuration space, where free cells are delineated in green, while non-traversable areas are highlighted in red. Additionally, dynamic objects, along with their predicted paths, are depicted in blue.

The **modified A* algorithm** for dynamic environments integrates both path recalculation and anticipation of dynamic obstacle movement directions. Anticipating the direction of obstacle movement involves labeling a node in the configuration space as reserved, along with assigning it a timestamp indicating when it will be occupied by a dynamic obstacle.

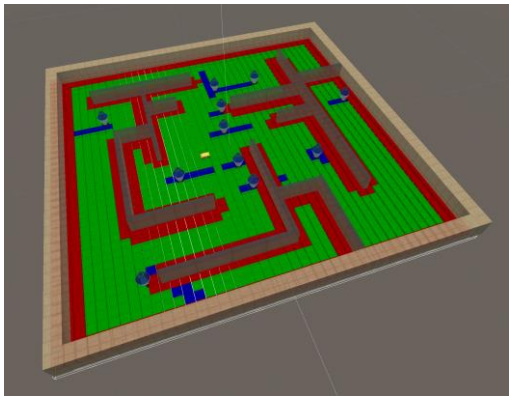

Fig. 9. Metric map with A* modification for dynamic environment.

This modification enables the algorithm to not only identify nodes that are permanently impassable but also to react to nodes that are temporarily occupied, termed as "busy" nodes. When encountering such nodes during the search, the function f(n) incorporates a temporal component, comparing the current time with the timestamp of node occupation. Consequently, the algorithm adapts its behavior to accommodate dynamic environmental changes, thus enhancing its responsiveness to dynamic obstacles.

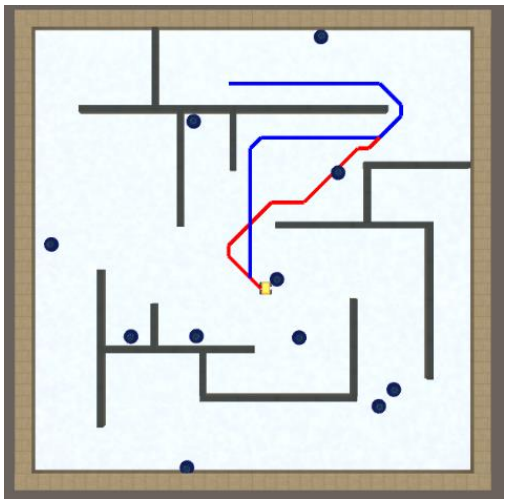

Fig. 10. Example of route BFS (red line), modified A* (blue line).

The modified A* algorithm for dynamic environments is executed in the following steps:

1. Obtain passable and impassable nodes of the configuration space.
2. If the user has set a goal, a timer is initiated.
3. Call the function to find the route.
    a. Obtain the node according to the robot's position.
    b. Obtain the goal node according to the clicked position.
    c. Create lists of open and closed nodes.
    d. Adding the starting node to the open list.

e. Repeat the cycle until the open list of nodes is empty.
   i. Mark the node with index zero from the open list as the current node.
   ii. Iterate through all nodes in the open list and mark the node with the smallest value of the function f(n) as the current node.
   iii. Remove the current node from the open list.
   iv. Add the current node to the closed list.
   v. If the current node is the goal node, the algorithm proceeds to find the reverse path from the goal node to the starting node and the cycle terminates.
   vi. For each accessible node adjacent to the current node:
      1. If the node is impassable or included in the closed list, it is disregarded.
      2. If the node is occupied, the value of the assigned function f(n) includes, among other components, a time component indicating the time required to move from the current node to the connected node.
      3. If the node is unoccupied, the value of the assigned function f(n) includes components g(n), h(n), and the reserved time component for occupancy is set to zero.
      4. If the node is not in the open list, it is added, and the current node is set as the parent node of this node.
      5. If the node is already in the open list, its cost-to-come is checked to see if it offers a better path cost, and if so, all components are recalculated.
      6. Return to e.
4. Find the reverse path and stopping the timer if it was initiated.
5. Send the new path to the mobile robot.
6. Repeat from 3. until the simulation is terminated.

## 3. RESULTS

We conducted a series of experiments in static environment aimed at assessing the time efficiency of path recalculation using both the Breadth First Search algorithm and the A* algorithm (Table 1).

In subsequent experiments, we chose to investigate the behavior of the modified A* algorithm within an environment featuring dynamic objects. Ten such objects were introduced into the environment, and various parameters such as the speed of the mobile robot or the dynamic objects were altered. We quantified the path length traversed by the mobile robot and documented the instances where it avoided collisions or executed movements without any incidents.

**Table 1. Path length and time of calculation.**

| Experiment no. | A* length [m] | BFS length [m] | Time A* [s] | Time BFS [s] |
|---|---|---|---|---|
| 1 | 42,728 | 44,042 | $2{,}62 \cdot 10^{-3}$ | $8{,}666 \cdot 10^{-3}$ |
| 2 | 18,657 | 21,385 | $1{,}232 \cdot 10^{-4}$ | $1{,}878 \cdot 10^{-3}$ |
| 3 | 43.485 | 49.284 | $2{,}112 \cdot 10^{-3}$ | $9{,}684 \cdot 10^{-3}$ |
| 4 | 34,071 | 35,042 | $6{,}903 \cdot 10^{-4}$ | $6{,}092 \cdot 10^{-3}$ |
| 5 | 48,899 | 48,213 | $3{,}699 \cdot 10^{-3}$ | $9{,}675 \cdot 10^{-3}$ |
| 6 | 15,414 | 19,556 | $8{,}725 \cdot 10^{-4}$ | $1{,}667 \cdot 10^{-3}$ |
| 7 | 14.414 | 16.899 | $4{,}27 \cdot 10^{-4}$ | $1{,}603 \cdot 10^{-4}$ |
| 8 | 18,485 | 19,799 | $5{,}347 \cdot 10^{-4}$ | $15{,}146 \cdot 10^{-4}$ |
| 9 | 39,657 | 41,455 | $6{,}73 \cdot 10^{-4}$ | $74{,}742 \cdot 10^{-4}$ |
| 10 | 15,414 | 19,556 | $8{,}37 \cdot 10^{-4}$ | $1{,}702 \cdot 10^{-3}$ |

As depicted in the Table 2, the length of the path varied for each experiment conducted, indicating the utilization of different points within the maze. Consequently, the robot was tasked with reaching distinct goals across varying distances. It was observed that longer paths posed greater challenges for the modified A* algorithm to navigate successfully to the goal without encountering collisions.

**Table 2. Results of the modified A* algorithm under different parameters of moving objects.**

| Experiment no. | Speed ratio [robot:dyn. obstacle] | Collisions | Dynamic Obstacles | Path length [m] |
|---|---|---|---|---|
| 1 | 1:1 | 1 | 10 | 20,89 |
| 2 | 1:1 | 1 | 10 | 46,38 |
| 3 | 1:1 | 1 | 10 | 73,45 |
| 4 | 1:2 | 2 | 10 | 30,21 |
| 5 | 1:2 | 1 | 10 | 31,972 |
| 6 | 1:2 | 4 | 10 | 44,071 |
| 7 | 2:1 | 0 | 10 | 29,213 |
| 8 | 2:1 | 2 | 10 | 61,071 |
| 9 | 2:1 | 0 | 10 | 69,97 |
| 10 | 3:1 | 0 | 10 | 29,627 |
| 11 | 3:1 | 0 | 10 | 53,482 |
| 12 | 3:1 | 1 | 10 | 63,384 |
| 13 | 1:1 | 0 | 20 | 29.21 |
| 14 | 2:1 | 0 | 20 | 43.556 |
| 15 | 1:2 | 3 | 20 | 57.656 |
| 16 | 3:1 | 0 | 20 | 65.627 |

Below are additional experiments illustrating (Figure 11) the disparities between the paths generated by the BFS and A* algorithms. It's noteworthy that while plotting the A* path, the algorithm dynamically responded to moving objects. In contrast, BFS simply replicated the fastest path it could identify. It is evident from the results that, in most cases, the path generated by the A* algorithm is notably superior to that of the BFS algorithm. As elucidated in preceding sections, this discrepancy arises from A* leveraging specific heuristics to derive the most optimal path feasible. Nevertheless, it's imperative to acknowledge that individual paths may occasionally converge.

Another experiment (Figure 12) highlights the disparity in the recalculated paths generated by these two algorithms. This experiment further corroborates the efficacy of the A* heuristic, as the robot opted for a less challenging path amidst dynamic obstacles. Interestingly, the path length in BFS is marginally shorter than that in the A* path.

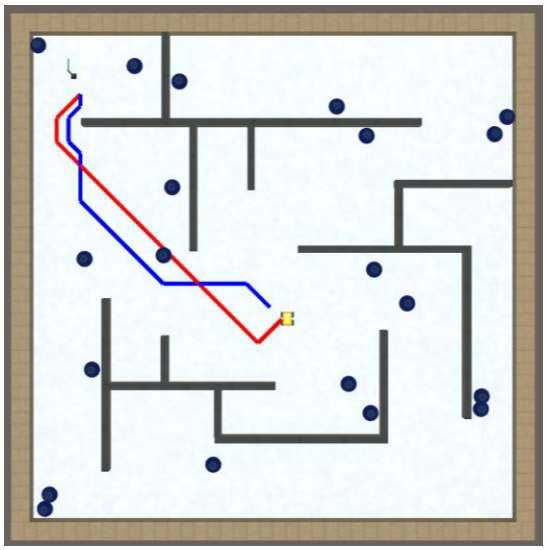

Fig. 11. Example of route BFS (red line) and A* (blue line) showing interaction with dynamic obstacles.

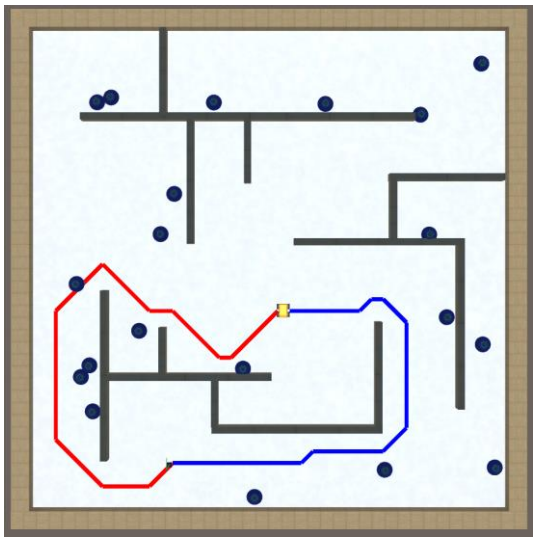

Fig. 12. Example of different routes of each algorithm.

We conducted additional experiments to validate the effectiveness of each algorithm individually. The images depict the calculated paths at various points. Figure 13 illustrates a scenario where BFS made an unnecessary detour at the onset of the path.

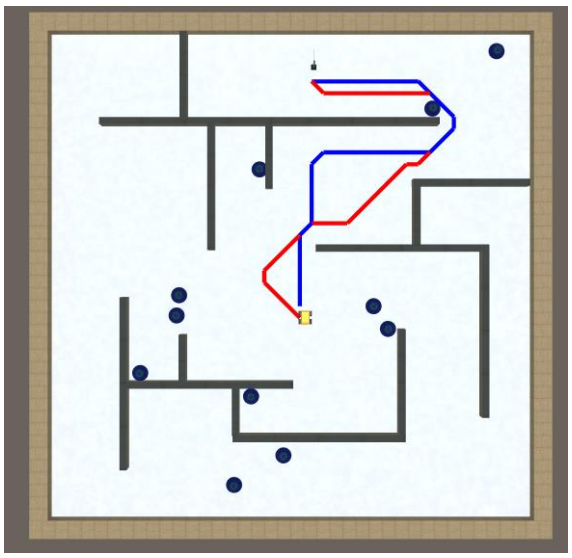

Fig. 13. Breadth First Search algorithm with unnecessary detour at the beginning of the path.

In the experiment shown in Figure 14, we can see that some parts of the path are the same for each algorithm. But BFS is again less efficient than A*, mainly due to its path length.

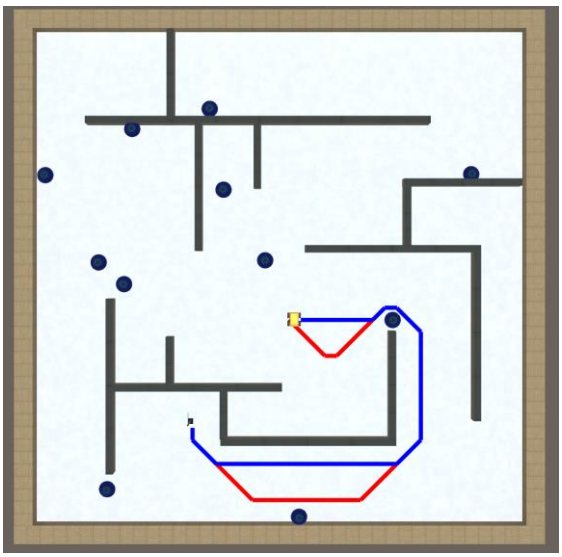

Fig. 14. A* path in compare with BFS is more efficient considering path's length.

Another experiment (Figure 15) demonstrates that the deviation observed in BFS routes stems from its undirected layer search approach. This deviation contrasts sharply with the straight path to the destination facilitated by A*.

In the last experiment (Figure 16), no dynamic objects were present, suggesting that the A* path remains unaffected by external factors. Nonetheless, even under these controlled conditions, A* consistently outperforms BFS. Notably, the A* path is significantly shorter and demonstrates superior

efficiency in terms of speed and requiring less intervention for control.

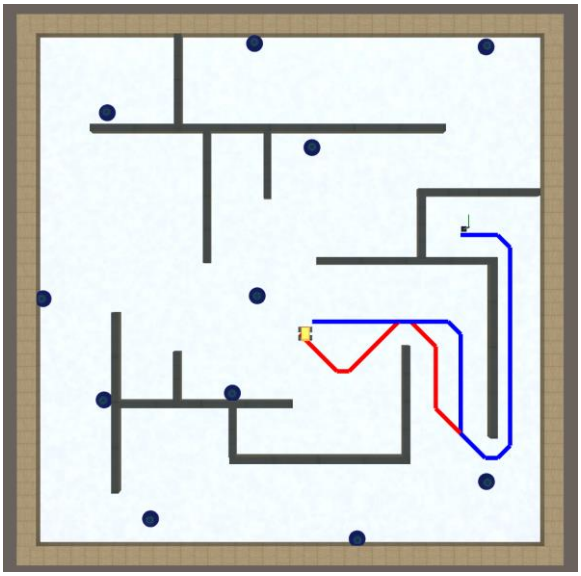

Fig. 15. A* straight path compared to BFS path with it's circumvention.

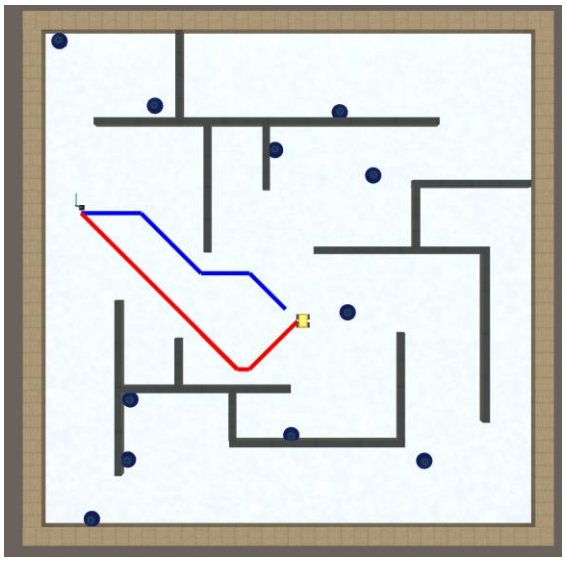

Fig. 16. Comparison of A* path (blue) with BFS path (red) in static environment.

## 3. CONCLUSIONS

In this article, our aim was to explore various aspects and opportunities in mobile navigation, elucidating the advantages and drawbacks of specific approaches to configuring the environment, path finding algorithms, and adapting them to accommodate dynamic environments. Our focus shifted towards diverse representations of known environments, ranging from basic layouts to simplified or transformed graph structures. We also delved into preparing algorithms to function within dynamic environments. Nevertheless, not every algorithm could be readily adapted, a point we thoroughly elaborated on within the article. The optimal approach often involved modifying algorithms containing heuristic elements, enabling functionalities such as path recalculations, path connections, and other adaptive strategies.

In the second part of the article, within the simulation phase, we opted for two specific algorithms: an oriented A* and a non-oriented Breadth First Search algorithm. We also tailored one algorithm to seamlessly operate within dynamic environments, unlike the other which was restricted to static conditions. Our choice of the metric (grid) space as the environment offered a convenient platform for applying both algorithms within a single structure. Despite the constraint of movement restricted to eight sides within the grid, this limitation did not impede our focus on assessing algorithm efficiency and comparison. To examine the responsiveness of the modified A* algorithm to dynamic objects, we implemented these features into the Unity 3D simulation environment, alongside other mentioned tools. Subsequently, we initiated a series of experiments to gauge algorithm efficiency, speed, path comparison, and navigation efficacy within a changing environment. Figure 8 illustrates the initial state within the maze, where the robot awaits target point determination amidst the movement of several dynamic objects. We anticipated the subsequent positions of these objects, represented by blue markings on the grid within the maze. During the simulation and experimentation phases, these objects remained in constant motion in various directions. Thus, it's crucial to note that while they appear static in the figures (resembling round dots), they are, in fact, dynamic objects. It's essential to consider that the A* algorithm's navigation is influenced by the dynamic environment.

We commenced our analysis by evaluating the efficacy of each algorithm, assigning them a common goal, and allowing them to explore the environment until they determined a path to reach this goal. Ten such experiments were conducted, as depicted in Table 1. It is evident from these results that, in most cases, the A* algorithm exhibits shorter path lengths and computation times compared to BFS. This efficiency can be attributed to the heuristic approach of A*, which consistently selects nodes closer to the goal, thus avoiding unnecessary exploration of disadvantageous nodes – a limitation inherent in the BFS algorithm.

Further simulation experiments, as displayed in Table 2, affirmed the effectiveness of mobile navigation in a known environment, particularly when anticipating potential movements of dynamic obstacles. With the introduction of dynamic obstacles moving at a constant speed, the A* algorithm demonstrated effective responsiveness unless the dynamic object unexpectedly changed its movement direction or collided with obstacles. In such instances, collisions occurred during testing, especially when a significant number of dynamic obstacles were present. The speed ratio between the mobile robot and dynamic obstacles also influenced collision avoidance. Slower dynamic obstacles allowed the robot to respond more effectively, whereas faster-moving obstacles increased the risk of collision due to the robot's inability to anticipate their movements accurately.

Path lengths were included in the table as longer paths inherently posed a higher risk of collision with dynamic obstacles. It is notable that, in individual experiments, the oriented A* algorithm consistently outperformed the non-oriented BFS algorithm in path length, yielding more adequate

and shorter paths. Figure 11 showcases completely divergent paths, with the A* path being notably more optimal. Figures 12 to 15 illustrate various experiments depicting different paths to goal. However, let's shift our attention to the Breadth First Search algorithm. Its erratic route attainment can be attributed to the fluctuations inherent in its non-oriented approach. This is particularly evident in Figure 14, where BFS changes direction multiple times unnecessarily compared to A*. Conversely, Figure 15 underscores that even when the A* algorithm is unaffected by dynamic objects, it still yields a more efficient path than Breadth First Search.

## ACKNOWLEDGMENT

This work was supported by grants VEGA 1/0775/20, VEGA 1/0599/20 and UAVLIFE (NFP313010ATR9).

## REFERENCES

Elgabry, O. (2018, May 28). Path Finding Algorithms - *OmarElgabry's Blog. Medium*.

Imms, D. (2016). A* pathfinding algorithm. *Growing With the Web.*

Patel, A. (1997). Grid Pathfinding optimizations. *Standford Theory.*

Vinther, A. S. H. (2015). Pathfinding in Two-dimensional Worlds: A Survey of Modern Pathfinding Algorithms, and a Description of a New Algorithm for Pathfinding in Dynamic Two-dimensional Polygonal Worlds (*Doctoral dissertation, Aarhus Universitet, Datalogisk Institut*).

Goodwin, S. D., Menon, S., & Price, R. G. (2006). Pathfinding in open terrain. In Proceedings of *International Academic Conference on the Future of Game Design and Technology* (Vol. 8).

Johnson, C., & Kuipers, B. (2012, October). Efficient search for correct and useful topological maps. In 2012 *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 5277-5282). IEEE.

Patel, A. (1997). Map Representations. *Standford Theory*.

Chopra, C. (2021, December 12). Pathfinding Algorithms - *The Startup. Medium*.

Cui, X., & Shi, H. (2011). A*-based pathfinding in modern computer games. International *Journal of Computer Science and Network Security*, 11(1), 125-130.

Navone, E. C. (2022, February 3). Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction. *freeCodeCamp.Org*.

Sukaj, E. (2020). Dijkstra's algorithm vs A* algorithm Detail Comparsion. *Slant.*

Graham, R., McCabe, H., & Sheridan, S. (2003). Pathfinding in computer games. *ITB Journal*, 8, 57-81.

Patel A. (1997). Dealing with moving obstacles. *In Standford Theory.*

Silver, D. (2005). Cooperative pathfinding. In *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment* (Vol. 1, No. 1, pp. 117-122).