International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V72I11P129

Volume 72 Issue 11, 294-306, November 2024

Original Article

Building a Route for a Mobile Robot Based on the BRRT and A*(H-BRRT) Algorithms for the Effective Development of Technological Innovations

Amer Abu-Jassar[1], Hassan Al-Sukhni[2], Yasser Al-Sharo[3], Svitlana Maksymova[4], Vladyslav Yevsieiev[4], Vyacheslav Lyashenko[5]

[1]*Department of Computer Science, College of Computer Sciences and Informatics, Amman Arab University, Amman, Jordan.*
[2]*Department of Computer and Cyber Security, Faculty of Information Technology, Jadara University, Jordan.*
[3]*Faculty of Information Technology, Department of Cyber Security, Ajloun National University, Jordan.*
[4]*CITAR Department of Kharkiv National University of Radio Electronics, Kharkiv, Ukraine.*
[5]*MST Department of Kharkiv National University of Radio Electronics, Kharkiv, Ukraine.*

[1]*Corresponding Author : a.abujassar@aau.edu.jo*

Received: 27 August 2024 Revised: 05 November 2024 Accepted: 15 November 2024 Published: 29 November 2024

Abstract *- The article examines the solution for the route constructing problem for a mobile robot using the BRRT (Biased Randomized Routing Table) and A*(H-BRRT) algorithms with the A*(A-star) optimizer. The use of such approaches allows to achieve the effective development of technological innovations based on mobile robots. A Python program was developed using the PhCham development environment to implement these algorithms. A study assessed the impact of changing basic parameters, such as the number of iterations and the movement step of the BRRT and A* algorithms, on the efficiency indicators of constructing a route for moving mobile robots. The study includes an analysis of execution time, length of the resulting route, route smoothness (number of turns), environmental complexity, overall route reliability and stability, and the ability to effectively deal with degenerate cases to develop technological innovation. The presented experimental results allow us to evaluate the effectiveness and applicability of the BRRT and A* algorithms for constructing optimal routes for a mobile robot in various environmental conditions. The obtained tracking results demonstrate the significant advantages of the developed H-BRRT algorithm for large maps with a size of 5000x5000 pixels compared to other algorithms developed for maps significantly smaller. The planning hour in the fragmented H-BRRT is extremely small, amounting to 0.000011 seconds, which significantly outweighs the effectiveness of other methods, where this indicator varies from 4.9 to 18.6 seconds. Wanting to expand, H-BRRT demonstrates the largest route – 24077.0 meters- determined by the map's scale and the advances to the route at great distances. Other methods, such as TG-BRRT and CW-TG-BRRT, show good results in terms of doubling down on small maps but sacrifice the calculation speed to the new H-BRRT.*

Keywords *- Mobile robot, Route planning, Algorithm BRRT, Algorithm A*, Optimization, Manufacturing innovation, Effective development, Industrial innovation.*

1. Introduction

In the era of technology and automation continuous development, the question of choosing algorithms for constructing movement routes for mobile robots (mobile systems) and studying the effectiveness of their work is becoming key in autonomous systems development. This is due to the need to introduce technological innovations into production for the purposes of effective and sustainable development [1-5]. The study of methods for selecting algorithms for autonomous systems aims to increase the efficiency and accuracy of robot movement, which is crucial for their successful implementation in various areas of human activity. In the context of developing autonomous robots capable of operating in various scenarios, research into the selection of appropriate algorithms is an integral step. This ensures an increase in the safety and efficiency of robotic systems and contributes to the creation of more flexible and adaptive solutions for a wide range of tasks as individual elements for implementing technological innovations [6,7]. One of the promising directions in this area is the study of the BRRT (Bi-directional Rapidly exploring Random Tree) algorithm with the A*(A-star) optimizer for constructing the route of a mobile robot movement [8,9]. This approach represents a symbiosis between motion planning methods and the efficiency of random search algorithms.

Thus, the relevance of such research is due to the rapid development of robotics and the need to create more intelligent and independent autonomous systems. Moreover, such systems must successfully navigate in complex scenarios and large-sized corresponding terrain maps used for such movement. Autonomous robots play a key role in many areas, from manufacturing and logistics of various sizes to medical research and security [10]. The dimensions of such areas, where the routes of mobile systems movement are compiled, can vary from several meters to several hundred meters, both vertically and horizontally. As a rule, existing developments operate in sizes 10x10, 30x30, and sometimes 100x100 [11,12].

The nodal points of such maps represent a certain configuration of the terrain map. Moreover, such nodal points can be located at different distances, not necessarily in units of measurement of the overall size of the area map. Then, these nodal points represent certain pixels of the terrain map configuration. However, even in this case, developers operate with a configuration of terrain maps of no more than several hundred pixels horizontally and vertically [13]. At the same time, the concept of such nodal points of the terrain map plays a decisive role in the study of appropriate algorithms for constructing movement routes of mobile systems (robots).

At the same time, the methods of the BRRT algorithm based on the A* optimizer make it possible to improve the accuracy and speed of planning complex and large routes, thereby ensuring more efficient functioning of autonomous systems. Consequently, the main goal of this work is to construct a route for a mobile robot based on the BRRT and A*(H-BRRT) algorithms for large-sized maps of the area to analyze and study such a symbiosis of algorithms for constructing a route for a mobile robot.

2. Related Works

W. Li et al. studied using the CC-BRRT algorithm for path planning for a mobile robot [14]. The CC-BRRT algorithm significantly improves the bidirectional fast extended random tree algorithm, offering an efficient solution to its convergence problem. The main advantages include using the center circle sampling strategy and the target offset strategy, which can reduce the number of search nodes and speed up the algorithm's convergence [14]. However, it should be noted that the CC-BRRT algorithm may require complex parameter tuning to achieve its most efficient performance. This may be a disadvantage when using it in practice. In addition, there may be problems with sensitivity to initial conditions and the risk of getting stuck in local minima, especially in complex scenarios. B. G. Jhong and M. Y. Chen proposed a navigation algorithm for motion planning of two-wheeled mobile robots, which has several advantages [15]. First, using bidirectional RRT algorithms with a path trimming and smoothing mechanism allows for obtaining a collision-free path with continuity of direction to the destination.

Secondly, speed planning based on the trapezoidal speed profile provides efficient control of linear and angular velocities, and the approximation method helps reduce the position error of the endpoint of the displacement curve, ensuring trajectory continuity [15]. Among the disadvantages, it is worth noting that the implementation of the algorithm may require complex parameter settings and high computing power, especially when working in real-time.

In addition, it is necessary to consider possible limitations on energy consumption and positioning accuracy, which may affect the algorithm's efficiency in real conditions. M. Korkmaz and A. Durdu consider improving the time to reach a task point and the efficiency of task performance of a mobile robot. As a result of various algorithm comparisons, it was shown that the A* algorithm provides the shortest path, but its time efficiency is low [16]. On the other hand, the PRM algorithm is time efficient and provides the length of possible movement close to the shortest path. Thus, the PRM algorithm may be the most suitable path-planning method for real-time mobile robots, considering the time and quality of the path.

The work by Y. Zhang and Du Gong conducted a study of the S-BRRT* algorithm, which improves global path planning under constraints [4]. Introducing a bidirectional spanning tree into the basic RRT* and considering non-holonomic constraints makes it more adaptive to different scenarios. The proposed new path shortening strategies and the use of a cubic Bezier curve for smoothness provide the most optimal and feasible moving trajectories [17]. However, it should be noted that implementing the S-BRRT* algorithm may require significant computational resources, especially in complex scenarios. Additionally, you must consider the potential difficulties in tuning algorithm parameters for optimal performance in different environments. However, experimental results on various scenes confirm the stability and improvement of the trajectory length of the S-BRRT* algorithm.

A study by X. Shu et al. proposed using the Locally Guided Multiple Bi-RRT* (LGM-BRRT*) method, significantly improving path planning in clustered environments with narrow aisles [18]. Unlike other Bi-RRT* based variants, LGM-BRRT* provides more efficient memory and time usage due to an improved bridge test and search strategy based on local guidance [18]. However, it should be noted that implementing LGM-BRRT* may require a certain degree of careful tuning to achieve the required performance in different scenarios. Additionally, in some cases, additional processing power may be required to provide a fast solution, especially in complex cluster

environments, making it more difficult to implement. In the work by the authors, P. Wang et al. it is proposed an adaptive bidirectional A* algorithm that represents a significant improvement in path planning for robots in 3D environments [19]. Unlike the standard A* algorithm, it adapts to variable conditions in the 3D environment and uses bidirectional search to find the optimal path [19]. It is worth noting that the adaptive bidirectional A* algorithm has a number of disadvantages that should be considered. When working in complex 3D environments with many obstacles, the algorithm may encounter problems processing large amounts of data, which can lead to increased path planning time. Additionally, the algorithm may be less effective when the plan needs to be updated frequently due to changing environmental or task conditions. It is also worth considering that the adaptive bidirectional A* algorithm may require more complex configuration and support, especially to work effectively in different scenarios.

The work by C. Wang and X. Yang proposed an improved Q-learning algorithm, which represents a significant improvement in the dynamic planning of obstacle avoidance routes for mobile agents [7]. Introducing a priority weight into the Q-learning algorithm improves its value estimation and increases its convergence speed and accuracy. This is especially important in complex interactive environments where other algorithms may encounter local optimization problems [20]. Despite the significant advantages, the improved Q-learning algorithm also has some disadvantages. One of the main disadvantages is that the algorithm requires a significant amount of training data to achieve high performance, which can be problematic in real-world scenarios with limited access to data.

In addition, the algorithm may encounter overfitting problems, especially if the learning parameters are not configured correctly or if the structure of the environment changes greatly. This can lead to a loss of the generalization ability of the algorithm and a decrease in its performance on new data. The examples discussed above are for planning the path of robot movement, operating with terrain maps no larger than two to three hundred pixels in size. This necessitates the study of the most effective algorithms, among which the symbiosis of BRRT and A*(H-BRRT) can be highlighted on large terrain maps. Here, it is suggested that a 5000x5000 pixel may be considered. This choice is due to the fact that such a map allows the building of a route for the mobile system (robot) movement both for production and for open areas where the corresponding robots are used.

So, the key research lies in creating a practical algorithm for determining the routes of mobile robots in large and foldable environments to correct numerical obstacles. As a rule, the original methods are limited by computational resources and do not provide the necessary accuracy and adaptability to great minds. The goal is to develop an optimized approach that will allow speed and precision to plan routes on large maps, increasing the productivity of autonomous systems.

3. Research Gap

Optimization or efficiency is often used in the reviewed works, which necessitates some clarification. In this work, the following in the form of a certain process will be meant by the efficiency of a mobile robot movement in a certain route or such movement optimization. First, it is the process of finding the best path or route from one point to another, considering certain criteria (travel time, distance of travel, etc.) or restrictions (for example, completing a path without collisions in the allotted time).

In other words, the goal of such a procedure (achieving its efficiency or optimization) may include minimizing the consumption of time, energy or other resources, as well as taking into account various factors such as obstacles on the way, speed of movement, safety, etc. [21-23]. In the framework of this research, the following key optimization (efficiency) parameters will be used: the number of iterations for constructing a route and the step size when constructing a BRRT tree. This choice is based on the fact that these parameters can affect the following parameters of robot movement:

The number of iterations allows:

- Achieve efficiency and optimization of route moving time and movement accuracy. At the same time, increasing the number of iterations can improve the accuracy of route construction, but it will also increase the computation time. On the other hand, too few iterations can lead to an incomplete and suboptimal route;
- Achieve management efficiency in a complex environment. For example, in more complex environments, more iterations may be required to find a route due to the greater number of possible paths and obstacles;

The step size allows:

- Optimization of the movement speed and convergence of the algorithm used. Increasing the step size can speed up the route-finding process, but there is a risk of missing optimal paths or encountering obstacles. A small step size may improve the accuracy of collision-free movement but will slow down the search for the desired route;
- Optimal control in a complex environment. In more complex environments, a smaller step size may be preferable to allow the robot to navigate around obstacles more accurately.

The optimization (efficiency) parameters considered for determining the route of a mobile robot's movement are key

but not exhaustive in various specific cases. Therefore, such parameters will be considered: route completion time, route length, route smoothness (number of turns), environmental complexity (number of obstacles), and overall route reliability and stability. In some way, these parameters are derived from the key ones: the number of iterations for constructing a route and the step size when constructing a BRRT tree.

4. Formalization of the Problem of Constructing a Route for a Mobile Robot Based on The Brrt And A* Algorithms

Let denote by G - a graph that is a map of the environment, where the vertices of such a graph are various points (control marks of movement), and the edges are connections between them (possible paths of movement). V – a set of graph vertices representing the points of the mobile robot movement on a specific area map; E – a set of graph edges representing possible transitions between control points of the robot's movement. q_{start} and q_{goal} are the starting and ending points of the robot's movement, respectively.

In this case, the robot configuration, its unique position and orientation in space will be in accordance with V and E will be meant. Let us introduce the movement cost function ($Cost(q_i, q_j)$). This mathematical model describes costs as distance, energy, time and environmental conditions or costs when moving from one robot configuration q_i to another at a point q_j. In the context of route planning for an autonomous mobile robot, this function plays an important role in determining the optimal path and can be represented in the following way:

$$Cost(q_i, q_j) = w_1 \cdot d(q_i, q_j) + w_2 \cdot c_{price}(q_i, q_j), \quad (1)$$

Where:

$d(q_i, q_j)$ – distance metric between configurations q_i and q_j representing the moving length;

$c_{price}(q_i, q_j)$ – a cost function that takes into account the cost of time, energy or other resources to avoid an obstacle, as well as the number of them on the way from q_i to q_j;

w_1 and w_2 – weighting factors that can be adjusted to control the impact of each component on the overall cost.

Function (1) demonstrates that the cost of movement depends on the length of the distance during movement and the presence of obstacles. Let us present a model as a heuristic function ($H_{price}(q_i, q_{goal})$) for estimating the cost of achieving the final configuration q_{goal} from the current configuration q_i. In this case, the heuristic function must be fast to evaluate and provide a feasible (lower) estimate of the path's cost between two configurations. From this study's point of view, this metric measures the direct distance between two points in space (in this case, it is the Euclidean distance). Thus, it helps speed up pathfinding by guiding the algorithm towards areas where the path is expected to be most optimal. The formalization of such a function is given below:

$$H_{price}(q_i, q_{goal}) = \|q_i - q_{goal}\|, \quad (2)$$

Where:

$\|q_1 - q_{goal}\|$ – Euclidean distance between points in configuration space. This roughly estimates how far the current configuration is off the target.

One of the function ($H_{price}(q_i, q_{goal})$) restrictions is that it must be acceptable by estimation, not overestimate the cost of the path and guarantee the correct operation of the A* algorithm [24]. Let us represent through $f(q_i)$ the combined cost function for a vertex using the A* algorithm. This function combines the cost of the path from the start vertex to the current vertex ($g(q_i)$) and a heuristic estimate of the cost from the current vertex to the final target vertex ($h(q_i)$). The representation of this function is as follows:

$$f(q_i) = g(q_i) + h(q_i), \quad (3)$$

Where:

($g(q_i)$) – the cost of the path from the initial vertex to the current vertex q_i. Typically represents the accumulated cost of moving from the starting vertex to the current vertex. As the algorithm runs, this cost is updated as the graph G expands. In particular, the cost for each vertex is calculated as the sum of the cost to the current vertex and the heuristic estimate of the cost to the goal. When updating the cost for a vertex, if the new cost is less than the previous one (that is, the most optimal path is found), then the vertex is added to the open list for further consideration;

($h(q_i)$) – a heuristic estimate of the cost from the current vertex q_i to the final target vertex, with the restriction that this is the cost of reaching the target vertex from the current vertex, and, as a rule, it is an approximate (acceptable) estimate that does not overestimate the real cost. Expression (3) allows us to select vertices to expand the description of the terrain map. Those with a lower combined cost $f(q_i)$ are the first. This ensures that the search is directed towards the least expensive paths and speeds up the algorithm's convergence.

It should be noted. Using the A* algorithm allows us to minimize the total cost $f(q_i)$, which means finding the optimal path from the initial vertex to the target vertex. Now, the initialization of robot route planning will be described. This initialization involves creating two trees T_{start} and T_{goal} (for the starting and ending configurations of the robot's movement) and defining a set of vertices V. Moreover, each vertex $v \in V$ represents a unique configuration of the robot in space, consisting of a grid of points. The tree T_{start} is initialized with a single vertex corresponding to the initial

Configuration q_{start}, and may be represented in next way:

$$T_{start} = \{v_{start}\}, \quad (4)$$

Where:

v_{start} – vertex corresponding q_{start}.

The tree T_{goal} is initialized by one vertex corresponding to the final configuration q_{goal}, and may be represented in next way:

$$T_{goal} = \{v_{goal}\}, \quad (5)$$

Where:

v_{goal} – vertex corresponding to the final configuration q_{goal}.

For a more specific description of how the initial values of trees T_{start} and T_{goal} are set using the operation of adding vertices, let us imagine that each vertex in the tree describes an object with certain characteristics, such as:

$$Vertex: v = (q, parent), \quad (6)$$

Where:

q – robot configuration corresponding to a given vertex;

$parent$ – a reference to a parent vertex - a vertex that generates other vertices (except for the initial vertex).

Then, the operation of adding a vertex to a tree T_{start} can be described as follows:

$$T_{start.}AddNode(v_{start}), \quad (7)$$

Where:

$v_{start} = (q_{start}, null)$ The initial vertex v_{start} contains the initial Configuration $q_{start}$and does not have a parent since it is the initial vertex.

Adding a vertex to a tree T_{goal} can be described as follows:

$$T_{goal.}AddNode(v_{goal}), \quad (8)$$

Where:

$v_{goal} = (q_{goal}, null)$, that is, the final vertex v_{goal} contains the final configuration q_{goal}.

Thus, the add vertex operation (7, 8) creates a vertex object with the specified characteristics and adds it to the corresponding tree. A parent reference to $null$ indicates that it is the starting or ending vertex and does not have a parent in the given context.

The vertex search and selection loop in the BRRT algorithm is an iterative process in which the algorithm seeks to expand the trees T_{start} and T_{goal} in the direction of a random vertex belonging to the robot's configuration space. This can be represented as choosing a random configuration from the configuration space $q_{rand} \in V$.

Let us find the nearest vertices in each tree for the current random configuration q_{rand}, which may be represented as the following entry:

$$q_{near\_start} = argmin_{q_i \in T_{start}} C_{price}(q_i q_{rand}), \quad (9)$$

$$q_{near\_goal} = argmin_{q_i \in T_{goal}} C_{price}(q_i q_{rand}), \quad (10)$$

Where:

q_{near_start} – this is the closest vertex in the tree T_{start} to a random configuration q_{rand}, found by minimizing the cost function;

q_{near_goal} – is the closest vertex in the tree T_{goal} to a random configuration q_{rand}, found by minimizing the cost function;

T_{start} – is a tree whose initial vertex corresponds to the initial configuration q_{start}, and each subsequent vertex is added as the tree expands towards random configuration;

T_{goal} – is a tree whose final vertex corresponds to the initial configuration q_{start};

$C_{price}(q_i, q_{rand})$ – the cost function measures the distance between vertices q_i and current random configuration q_{rand}.

Let us expand the tree in this direction q_{rand}. For this, it is proposed that the Function [25] be used, which acts as an expansion of the current vertex towards another configuration. It generates a new vertex in the tree, which is close to a random configuration. It may be represented as follows:

$$q_{new\_start} = Extend(q_{near\_start}, q_{rand}), \quad (11)$$

$$q_{new\_goal} = Extend(q_{near\_goal}, q_{rand}), \quad (12)$$

Where:

q_{new_start}, q_{new_goal} – new vertex resulting from trees T_{start} and T_{goal} expansion respectively;

q_{near_start} – current closest vertex in the tree T_{start} to a random configuration;

q_{near_goal} – current closest vertex in the tree T_{goal} to a random configuration;

q_{rand} – a random configuration to which it is planned to expand the current vertex.

The next step is to check whether the new vertex collides with environmental obstacles. Collision testing involves assessing whether a new vertex q_{new_start} or q_{new_goal} is consistent with a safe location in the environment. Let us say that a collision function takes a robot's configuration q as input and returns a boolean value: true if the configuration is collision-free and false otherwise [26].

$$CollisionFree(q_{new}), \quad (13)$$

Where:

q_{new} – a new vertex resulting from expanding the current vertex towards a random configuration.

Let us check whether the new vertex collides with obstacles in the environment; if there is no collision, add a new vertex to the corresponding tree:

$$T_{start.}AddNode(q_{new\_start}), \quad (14)$$

$$T_{goal.}AddNode(q_{new\_goal}), \quad (15)$$

Where:

T_{start} and T_{goal} – a tree representing a set of vertices describing the space of robot configurations, starting from the initial configuration q_{start} and the final configuration q_{goal}, respectively;

$AddNode$ – function to add a new vertex q_{new_start} and q_{new_goal} to trees T_{start} and T_{goal} respectively [27];

q_{new_start} and q_{new_goal} – represents the robot configuration resulting from expanding the current nearest vertex in the direction of a random configuration

Next, it is necessary to check the connection, that is, whether the new vertex is connected to the trees T_{start} and T_{goal} for this, the following expression will be used:

$$Connected = ChekConnection(T_{start}, T_{goal}, q_{new\_start}, q_{new\_goal}), \quad (16)$$

Where:

$ChekConnection$ – a function that checks for a connection between trees T_{start} and T_{goal}, returns a boolean value: $true$ if a connection between trees is found, $false$ otherwise [28].

This function evaluates whether it can connect tree T_{start} and tree T_{goal} through new vertices q_{new_start} and q_{new_goal} . The result $Connected$ indicates whether the connection was successful or not, which influences the further progress of the pathfinding algorithm.

To form a route and extract the optimal path using the A* algorithm on a combined graph, it is denoted: via G_{start} and G_{goal} the combined graphs from trees T_{start} and T_{goal}, respectively, into a common graph G_{total}. The union occurs by adding an edge between the nearest vertices in the trees T_{start} and T_{goal}. The formation of a state graph G_{total} represents a state space that includes vertices from both trees and the edges between them. For each edge G_{total} in the graph, determine the cost of moving between the corresponding vertices as the distance between the vertices in the robot configuration space. Let us apply the A* algorithm to the graph G_{total} to find the optimal path from the initial configuration q_{start} to the final configuration q_{goal}. The A* algorithm uses a heuristic function to find a path efficiently, taking into account both the cost to the vertex (along the path already traversed) and the heuristic estimate from the current vertex to the target vertex. The formalized representation of the A* algorithm and path extraction may be denoted as follows:

$$f(n) = g(n) - h(n), \quad (17)$$

Where:

$g(n)$ – the cost of the path from the initial vertex to the vertex n,

$h(n)$ – heuristic cost estimation from the vertex n to the goal vertex.

To extract the optimal path, imagine a sequence of vertices P, in the next way:

$$P = \{v_1, v_2, \ldots, v_k\}, \quad (18)$$

Where:

v_i – vertex in the sequence P represents the state of the robot configuration space at a certain stage of movement along the optimal path;

k – the index indicates the optimal path length and the number of stages that must be passed from the initial configuration to the final one.

The path is retrieved in reverse order. This order ensures the correct sequence of movements for the robot from the starting point to the target:

$$q_{start}, q_i, q_{i-1}, \ldots, q_{goal}, \quad (19)$$

Where:

q_{start} – starting point of the path;

q_i and q_{i-1} – represents the robot configuration at a point in time i;

q_{goal} – the final peak of the path.

Thus, the combined graph, route generation and optimal path extraction using the A* algorithm on this graph allows us to find an effective route for the robot to move from the initial configuration to the target.

5. Separate Fragments of Software Implementation in Constructing the Route of Movement of a Mobile Robot and a Description of the Features of Their Application

To check the correctness of the reasoning, a program has been developed to simulate the work of constructing the route of movement of a mobile robot using the object-oriented Python language in the PhCham 2022.2.3 development environment. PhCham's core choice offers

greater integration capabilities with Python and support for object-oriented programming, which is important for research in the robotics field. This framework will also provide efficient processing of large data sets and visualization, suitable for testing algorithms on large-scale maps.

Fragments of software implementations are given below:

First, the basic settings are set: environment parameters, start and end points and parameters of the BRRT and A* algorithm. These values specify the parameters for finding a path between the starting point (start_point) and the ending point (goal_point) on a 5000x5000 pixel area map. The num_iterations parameter determines the number of iterations of the algorithm, which affects the time to find the optimal path:

width = 100,

height = 100.

start_point = (10, 10) # starting point coordinates q_{start},

goal_point = (5000, 5000) # final point coordinates q_{goal},

num_iterations = 500 # – this parameter determines how often the algorithm will repeat the search process, expanding the tree. More iterations can lead to a deeper exploration of the state space and, ideally, more accurate determination of the optimal route. However, too many iterations can lead to excessive execution time of the algorithm,

step_size = 10.0 # – this parameter determines the step length by which the algorithm expands the current tree at each iteration. A larger step size may speed up the search process but may also result in missing the most optimal paths or even the inability to avoid obstacles. Let us describe the generation of obstacles in the form of blocks of size 10x10. This approach helps to understand how the robot can be interfered with by obstacles of a certain size and shape, which is one of the basic restrictions taken into account when planning a path:

```
np.random.seed(42)
num_obstacles = 20
obstacles = [(np.random.randint(0, width - 10), np.random.randint(0, height - 10)) for _ in range(num_obstacles)]
```

Let us create a function to check collisions with obstacles. This function checks if the point is inside any obstacle on the map. It takes the coordinates of a point and checks for their presence in the list of obstacles, represented as pairs of coordinates of the upper left corner of each 10x10 block. If the point is inside an obstacle, the function returns False, indicating a collision with the obstacle. Otherwise, if the point is not inside any obstacles, the function returns True, which means there are no collisions. This function influences the research results by allowing the path planning algorithm to take into account obstacles and avoid them when finding the optimal path for the mobile robot:

```
def is_collision_free(point):
    for obstacle in obstacles:
        if obstacle[0] <= point[0] < obstacle[0] + 10 and obstacle[1] <= point[1] < obstacle[1] + 10:
            return False
    return True
```

let us create a function to build a route using the BRRT algorithm. This function builds a route for a mobile robot using the BRRT (Bidirectional Rapidly exploring Random Tree) algorithm. It starts at a starting point and iteratively adds new points to the tree until it reaches a given number of iterations. Each new point is selected randomly within the map. Then, the closest point in the tree is found for each new point, and a step size (step_size) is applied to it to determine the new point. The collision check for obstacles (is_collision_free) ensures that a new point is not added to the tree if it is inside an obstacle. This process continues until the endpoint is reached or until enough iterations have been explored:

```
def build_rrt(start, goal, num_iterations, step_size):
    tree = [start]
    for _ in range(num_iterations):
        random_point = np.random.rand(2) * np.array([width, height])
        nearest_point_index = np.argmin([np.linalg.norm(np.array(random_point) - np.array(point)) for point in tree])
        nearest_point = tree[nearest_point_index]
        new_point = nearest_point + step_size * (random_point - nearest_point)
        new_point = tuple(new_point)
        if is_collision_free(new_point):
            tree.append(new_point)
    return tree
```

This algorithm influences the results of the study, allowing you to find routes for a mobile robot, taking into account obstacles and optimizing the path based on a random selection of points and expanding the tree in the direction of these points,

let us create a function to optimize the route using A*. This function creates a graph of the BRRT route points and adds edges between successive points with weights equal to the distance between them. Then, to optimize the route, the A* algorithm is used, which finds the shortest path from the starting point to the ending point on the constructed graph:

```
def optimize_path(rrt_tree, start, goal):
    graph = nx.Graph()
    for point in rrt_tree:
        graph.add_node(point)
    for i in range(len(rrt_tree) - 1):
        graph.add_edge(rrt_tree[i], rrt_tree[i + 1],
```

weight=np.linalg.norm(np.array(rrt_tree[i]) - np.array(rrt_tree[i + 1])))

This affects the results of the study, as it allows us to improve the route found by BRRT by exploring alternative paths and choosing the most optimal one, taking into account the cost of moving between points. Using A* allows us to find a shorter and more efficient path, which can be critical for the operation of a mobile robot in real-time

let us add the starting and ending points to the graph. Adding start and end points to the graph allows us to consider these points when finding the optimal path. The starting and ending points become part of the graph, and the A* algorithm will consider them when finding a path from the starting point to the ending point. This guarantees that the found path will start at the starting point and end at the final point, which is important for the problem of path planning for a mobile robot:

```
graph.add_node(start)
graph.add_node(goal)
```

Adding start and end points also affects the exploration results because they become part of the state space the algorithm must explore. This helps ensure the completeness of the state space exploration and find the optimal path, taking into account the specific conditions of the starting and ending points;

Check the presence of connections between the starting and ending points and points of the route constructed by the BRRT algorithm in the graph. If there is no connection, then it is added, taking into account the distance between the points. This influences the study results because it ensures that the start and end points are associated with the route, ensuring that the path is planned correctly. It also considers the specific conditions of the starting and ending points. This is important for the practical implementation of the possible path of a mobile robot:

```
if not nx.has_path(graph, start, rrt_tree[0]):
    graph.add_edge(start, rrt_tree[0], weight=np.linalg.norm(np.array(start) - np.array(rrt_tree[0])))
if not nx.has_path(graph, rrt_tree[-1], goal):
    graph.add_edge(rrt_tree[-1], goal, weight=np.linalg.norm(np.array(rrt_tree[-1]) - np.array(goal)))
```

Let us create a function to evaluate solutions to problems with degenerate cases. This function is intended to evaluate the ability of the BRRT algorithm to cope with degenerate cases, such as the presence of narrow passages or complex obstacle structures that can make path construction difficult. It takes as input the constructed BRRT tree and the optimal path and then analyzes how successfully the algorithm coped with such complexities:

```
def evaluate_handling_degenerate_cases(rrt_tree, optimal_path):
```

This influences the study results, allowing the algorithm's reliability and efficiency to be assessed in various scenarios. The evaluation results can help improve the algorithm for more successful applications in real-world settings where degenerate cases may be common. An example of an assessment is that the fewer turns in the optimal path, the better. This assessment evaluates the optimal path based on the number of turns, considering that fewer turns indicate a simpler and more efficient route. This approach allows us to evaluate the “smoothness” of the path and its suitability for moving a mobile robot. This is important because more difficult paths with many turns may be less efficient and take longer to complete:

```
    optimal_turns = count_turns(optimal_path)
    return optimal_turns
```

Such an assessment can help optimize path planning by favoring easier paths with fewer turns when choosing the optimal route. This can be especially useful in situations where the speed or energy efficiency of the robot's movement is important,

Visualize the starting ending points and obstacles. Visualization allows us to present the initial data and conditions of the path planning problem for a mobile robot. This is important for analyzing and understanding the path search space, as well as for visual monitoring of the correct operation of planning algorithms:

```
plt.scatter(*start_point, color='green', marker='o', label='Start')
plt.scatter(*goal_point, color='red', marker='o', label='Goal')
for obstacle in obstacles:
    plt.Rectangle((obstacle[0], obstacle[1]), 10, 10, color='black', alpha=0.5)
```

Visualization helps to see the location of the start and end points relative to obstacles, which can be important for determining the complexity of a path planning problem. It also helps to track path changes during optimization and evaluate its effectiveness visually. In general, visualization helps better understand the problem and results of the study, making them more accessible and understandable Route construction. This code builds a route using the BRRT algorithm from the starting point to the ending point. Route construction allows us to evaluate the algorithm's ability to find paths in difficult conditions, considering obstacles. The route results can be used to analyze the effectiveness of the algorithm and its ability to avoid obstacles:

```
rrt_tree = build_rrt(start_point, goal_point, num_iterations, step_size)
```

This affects the results of the study because the quality of the constructed route directly reflects the performance and reliability of the algorithm. The research could be more successful if the most optimal and safe route could be constructed. Also, the route results can be used to compare with other path planning algorithms and evaluate their effectiveness, route optimization.

This code optimizes the constructed route using the A* algorithm. Optimization allows us to improve the BRRT algorithm's original route, considering the cost of moving between points and possible alternative paths. The result of optimization is a shorter and more efficient path from the starting point to the final point:

```
optimal_path, execution_time = optimize_path(rrt_tree, start_point, goal_point)
```

This influences the study results because the optimized path can be more efficient and safer for the mobile robot to move. Also, optimization results can be used to compare with the original route and evaluate the effectiveness of path planning algorithms. In addition, optimization execution time can be an important indicator of the algorithm's performance under real-world operating conditions. Visualization of the route. This code visualizes the constructed route of the BRRT algorithm as lines connecting successive route points. Visualizing a route allows us to visually assess its quality, such as its straightness, avoidance of obstacles and overall structure. This is important for analyzing the operation of the algorithm and checking the correctness of the route construction:

```
for i in range(len(rrt_tree) - 1):
    plt.plot([rrt_tree[i][0], rrt_tree[i + 1][0]], [rrt_tree[i][1], rrt_tree[i + 1][1]], color='blue', alpha=0.5)
```

Route visualization can also help to identify potential problems in the constructed route, such as unnecessary loops or unwanted detours. This allows us to improve the path planning algorithm and create more optimal routes for the mobile robot. In addition, visualization can be used to visually present research results and demonstrate the algorithm's operation to other users or specialists,

Let us visualize the optimal route. This code visualizes the optimal route after optimisation using the A* algorithm. Visualization of the optimal route allows us to evaluate the optimisation quality and compare it with the original route built by the BRRT algorithm. This is important for assessing the effectiveness of path planning algorithms and choosing the most suitable one for specific tasks:

```
for i in range(len(optimal_path) - 1):
    plt.plot([optimal_path[i][0], optimal_path[i + 1][0]], [optimal_path[i][1], optimal_path[i + 1][1]], color='green', linewidth=2)
```

The optimal route visualisation also helps you visually compare it with the original route and evaluate the improvements achieved due to optimization. This can be useful for analyzing the performance of an algorithm and identifying its advantages and disadvantages. In addition, visualization helps to visualize the study's results and draw conclusions about its success.

Assessment of such criteria as the complexity of the environment, reliability and stability of the RRT (Rapidly exploring Random Tree) tree, and assessment of the processing of degenerative cases. This code evaluates various criteria that may influence the results of a path planning study for a mobile robot. Environmental complexity assessment allows us to evaluate how complex the conditions the robot must navigate are. This is important for understanding the requirements of a path planning algorithm and its ability to avoid obstacles.

Assessing the reliability and stability of an RRT tree allows us to evaluate how reliable and stable the constructed tree is, which is the basis for path finding. This allows us to evaluate the performance of the BRRT algorithm and its ability to find optimal paths under various conditions. Evaluating the handling of degenerative cases evaluates how well a path planning algorithm can handle complex scenarios, such as situations where the path passes through narrow passages or contains redundant loops. This is important for assessing the versatility and effectiveness of the algorithm in various scenarios:

```
environment_complexity = evaluate_environment_complexity(obstacles)
reliability_and_stability = evaluate_reliability_and_stability(rrt_tree)
degenerate_cases_evaluation = evaluate_handling_degenerate_cases(rrt_tree, optimal_path)
```

These assessments influence the study results because they allow us to evaluate the quality of the path planning algorithms in various conditions and draw conclusions about their applicability and effectiveness.

6. Experimental Studies and Evaluation of the Brrt Algorithm with A* Optimization for Constructing a Route for a Mobile Robot

The research was carried out based on the hardware component with the following parameters: CPU Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz, RAM DDR 16.0 GB, GPU NVideo GeForce GTX1660TI and Intel(R) UHD Graphics 630, HDD SSD NVMe Micron_2200_MTFD and Toshiba MQ04ABF100. OS Windows 10 Pro 64-bit.

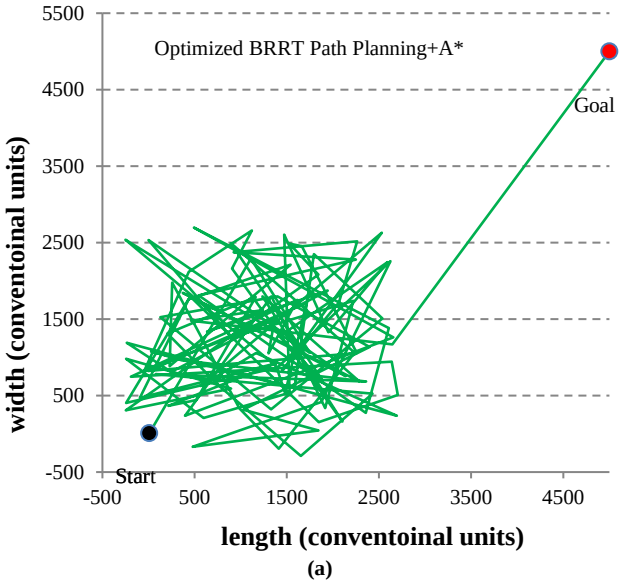
First of all, it was analyzed the influence of changing the basic parameters (number of iterations (N_{iter}) and movement step (M_{step})) of the BRRT and A* algorithm on performance evaluation indicators such as execution time, length of the

resulting route, smoothness of the route (number of turns), complexity of the environment, assessing overall reliability and stability and assessing the resolution of problems with degenerate cases. Execution time – allows us to evaluate the speed of algorithms in real time.

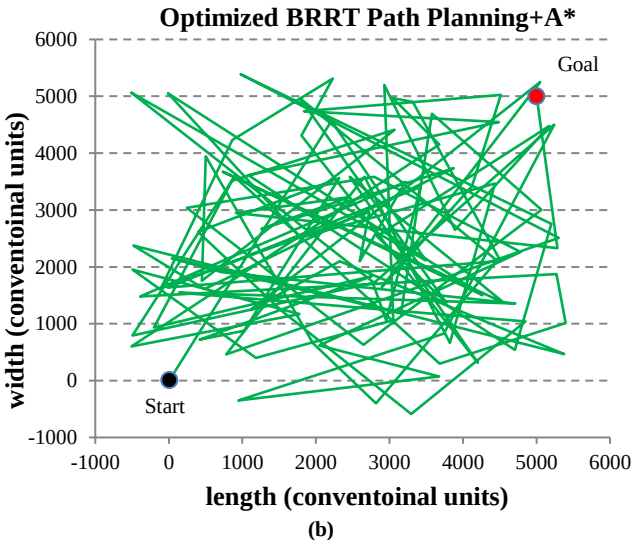
A faster algorithm may be preferable in tasks that require rapid response to environmental changes. The length of the resulting route allows us to evaluate the optimality of the found path. A short path is usually preferred because it requires less time and resources. The smoothness of the route (number of turns) evaluates the convenience and safety of the route for a mobile robot to travel through. A less tortuous route may be preferable for mobile robots, especially at high speeds.

Environmental complexity – assesses the influence of environmental conditions on the passability of the route (multiple and moving obstacles, narrow passages and complex routes, the presence of "traps" and dangerous zones). This is important for algorithms that must avoid obstacles efficiently. Assessing overall reliability and stability allows one to evaluate the robustness of algorithms to various conditions (changes in the environment, changes in the types and structure of obstacles) and ensure their reliable operation in various scenarios.

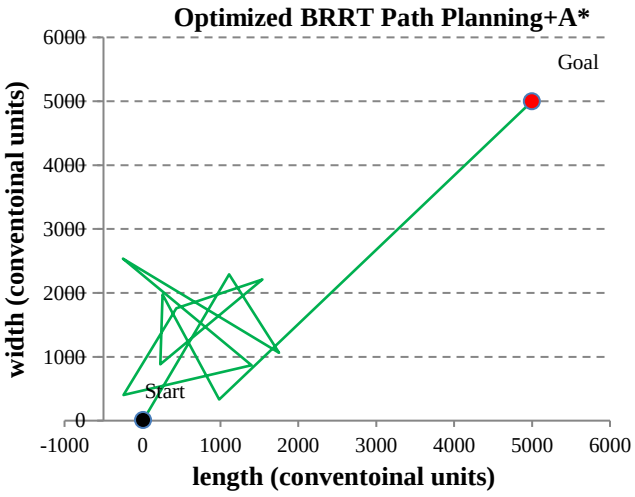
The Degenerate Case Resolution Assessment assesses the ability of algorithms to efficiently handle complex scenarios (with many obstacles, degenerate cases in the path graph, and an unstable positioning signal) and ensure correct operation even under non-standard conditions. Provided that the coordinates of the starting and ending points are the same in all experiments, the results are presented in graphs in Figure 1, and performance assessment indicators are given in Table 1.



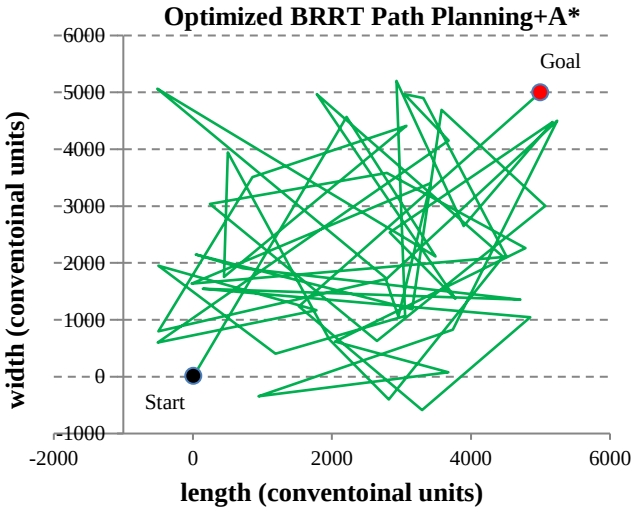
(a)



(b)



(c)



(d)

Fig. 1 Graphs for constructing optimized routes for different basic parameters (number of iterations and movement steps): a) N_{iter} = 100, M_{step}=30; b) N_{iter} = 100, M_{step}=60; c) N_{iter} = 10, M_{step}=30; d) N_{iter} = 50, M_{step}= 60

Table 1. Indicators for assessing the effectiveness of optimized routes for moving a mobile robot for different basic parameters N_{iter} and M_{step}

Performance Evaluation Indicators	Basic parameters (number of iterations and movement step)			
	N_{iter}=100 M_{step}=30	N_{iter}=100 M_{step}=60	N_{iter}=10 M_{step}=30	N_{iter}=50 M_{step}=60
Execution time (s)	0.000999	0.000997	0.000011	0.000997
Received route length (conventional units)	156151.2	312407.5	24077.0	152154.9
Smoothness of the route (number of turns)*	100	100	10	50
Environmental complexity	20	20	20	20
Overall reliability and stability*	101	101	11	51
Assessing the resolution of problems with degenerate cases*	100	100	10	50

* – The lower the value obtained, the higher the score for the result obtained

Based on the obtained results of modeling optimized routes for moving mobile robots with different basic parameters (number of iterations and movement step) for the developed planning method, the following conclusions can be drawn (Figure 1 and Table 1):

- The execution time generally remains relatively low for all parameter options (about 0.001 seconds), which indicates the speed of the proposed route planning method;
- Increasing the number of iterations and the moving step leads to a slight increase in execution time, which is logical since a larger number of iterations and/or a larger step require more computing resources;
- It is noticeable that an increase in the number of iterations and the movement step leads to an increase in the length of the route. This may be because more iterations and/or a larger stride may allow the algorithm to explore more space and thus find longer paths;
- The number of rotations generally increases with the number of iterations and the movement step. This is because more iterations and/or larger strides may result in more complex and "suboptimal" paths that involve more turns;
- The complexity of the environment is measured by the number of obstacles (num_obstacles = 20). In this study, it is constant for various experimental options when changing parameters: the number of iterations (N_{iter}) and the movement step (M_{step});
- Overall reliability and stability generally remain high for all parameter options. This may indicate that the algorithm does a good job of constructing stable routes;
- The score for resolving problems with degenerate cases also remains high for all parameter options. This may indicate that the algorithm can successfully cope with various scenarios and inherent features.

7. Comparative Analysis

Let us analyze and compare the resulting indicators for assessing the effectiveness of optimized routes for different basic parameters N_{iter} and M_{step} in comparison with other algorithms; the comparison results are shown in Table 2.

Table 2. Comparison of individual performance indicators of optimized routes with other algorithms

Algorithm	Planning time [s]	Path length [m]
RRT [29]	17.3	22.6
CW-RRT [29]	18.6	21.4
BRRT [29]	8.7	21.5
TG-BRRT [29]	10.7	20.2
CW-TG-BRRT [29]	4.9	21.0
H-BRRT [29]	6.3	19.0
H-BRRT (developed)*	0.000011	24077.0

* – The map size is 5000x5000 conventional units (pixels). Compared to analogs RRT, CW-RRT, TG-BRRT, CW-TG-BRRT, and H-BRRT the map size is 10x10 meters

When comparing the developed H-BRRT with analogues (Table 2), the following conclusions can be drawn:

- Planning time in H-BRRT (the developed approach) shows a significant superiority in planning time overall analogues. Its planning time is only 0.000011 seconds, while its analogs are much higher, for example, CW-TG-BRRT - 4.9 seconds. This indicates the high efficiency and speed of the developed algorithm;
- The path length of the developed H-BRRT is 24077 meters, which seems abnormally large compared to analogues; it should be taken into account that the developed algorithm works with a map of 5000x5000 pixels, while analogues work with maps of 10x10 meters. Thus, the path length in pixels can be comparable to the path length of analogs in meters, and this value should not be considered as a disadvantage but rather as a feature of working with different map scales.

Thus, the developed H-BRRT demonstrates significant advantages in planning time over its analogues, which makes it a promising choice for route planning problems on large, high-resolution maps. At the same time, the H-BRRT algorithm ensures a balanced result with equal accuracy and efficiency compared to popular algorithms, such as CC-BRRT, S-BRRT* [29]. The research showed that, while equal to the classic A* algorithm, new algorithms can be more effective on large maps due to search optimization and node minimization. The prote CC-BRRT algorithm may be

more effective in smaller and less complex scenarios, and the adaptive A* will optimise 3D environments. Thus, the novelty of the robot lies in the combined BRRT and A* algorithms for effective planning of the route of a mobile robot on large maps, which significantly expands the capabilities of these algorithms in folding environments. The proposed combination of algorithms demonstrated stable results in minds, while other methods reveal improved processing speed and adaptability to various scenarios.

8. Limitations, Future Work and Practical Applications

The study examined the capabilities of the BRRT and A*(H-BRRT) algorithms on large terrain maps. One of the key limitations is the high need for computing resources to work with large map sizes (5000x5000 pixels) and the complexity of the route structure, which affects the performance and speed of calculations. Therefore, an important goal of future research is to reduce the resource intensity of the algorithms without losing the accuracy of route construction. A promising direction is also the adaptation of these algorithms to work in conditions with a variable or dynamic environment. However, the study results have significant potential for practical use in various industries. The BRRT and A*(H-BRRT) algorithms can be applied to planning routes for autonomous robots at industrial enterprises in logistics and security. Such robots can effectively use routes to optimize logistics processes or survey the territory. The research provides a solid foundation for developing autonomous systems capable of adapting to complex and changing working environments, which is especially relevant for large-scale production and work in spacious or difficult terrain.

9. Conclusion

The work considers the symbiosis of the BRRT and A* algorithms for constructing the movement path for a mobile robot. This approach is implemented and tested in Python using the PhCham development environment. Experimental data showed that changing the basic parameters (number of iterations and moving steps) significantly impacts the efficiency of route construction. The execution time of the algorithms remains quite low for all parameter options, which indicates the high speed of the proposed route planning method. However, increasing the number of iterations and the movement step leads to a slight increase in execution time due to greater consumption of computing resources. Increasing the number of iterations and the movement step also leads to an increase in the length of the route and the number of turns. This is because algorithms have more space to explore and can find longer more complex paths. Environmental complexity, measured by the number of obstacles, remains relatively constant across different parameter options. The overall reliability and stability of routes also remain high, and the symbiosis of algorithms successfully copes with degenerate cases and various scenarios.

Thus, the study results allow us to conclude that the BRRT and A* algorithms are highly effective and applicable for constructing optimal routes for a mobile robot in various environmental conditions. However, the research revealed new issues that could become the basis for future work and potential applications. One such issue is the optimization of the BRRT and A* algorithms for work in a dynamic environment where obstacles can change their position. In addition, the task of adapting the proposed approach to work on 3D maps remains open, which will require additional research on computational efficiency. The solution of these issues forms the basis of subsequent research. This will allow the proposed algorithms to be used in more complex conditions, for example, for autonomous work or performing various tasks in multi-level environments.

References

[1] Pietro Bilancia et al., "An Overview of Industrial Robots Control and Programming Approaches," *Applied Sciences*, vol. 13, no. 4, pp. 1-14, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[2] Achim Buerkle et al., "Towards Industrial Robots as a Service (IRaaS): Flexibility, Usability, Safety and Business Models," *Robotics and Computer-Integrated Manufacturing*, vol. 81, pp. 1-20, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[3] Jianjun Ni et al., "An Improved Ssd-Like Deep Network-Based Object Detection Method for Indoor Scenes," *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1-15, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[4] Rodrigo Bernardo, João M.C. Sousa, and Paulo J.S. Gonçalves, "A Novel Framework to Improve Motion Planning of Robotic Systems Through Semantic Knowledge-Based Reasoning," *Computers & Industrial Engineering*, vol. 182, pp. 1-16, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[5] Dingyun Duan et al., "Industrial Robots and Firm Productivity," *Structural Change and Economic Dynamics*, vol. 67, pp. 388-406, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[6] V. Troianskyi et al., "First Reported Observation of Asteroids 2017 AB8, 2017 QX33, and 2017 RV12," *Contributions of the Astronomical Observatory Skalnaté Pleso*, vol. 53, no. 2, pp. 5-15, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[7] Federico Adolfi, Jeffrey S. Bowers, and David Poeppel, "Successes and Critical Failures of Neural Networks in Capturing Human-Like Speech Recognition," *Neural Networks*, vol. 162, pp. 199-211, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[8] Xiangrui Xing et al., "Robot Path Planner Based on Deep Reinforcement Learning and the Seeker Optimization Algorithm," *Mechatronics*, vol. 88, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[9] Xinning Li et al., "Research on an Optimal Path Planning Method Based on A* Algorithm for Multi-View Recognition," *Algorithms*, vol. 15, no. 5, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[10] N. Hung, and et al., "A Review of Path Following Control Strategies for Autonomous Robotic Vehicles: Theory, Simulations, and Experiments," *Journal of Field Robotics*, vol. 40, no. 3, pp. 747-779, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[11] Shiwei Lin et al., "An Intelligence-Based Hybrid PSO-SA for Mobile Robot Path Planning in Warehouse," *Journal of Computational Science*, vol. 67, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[12] Martina Benko Loknar, Gregor Klančar, and Sašo Blažič, "Minimum-Time Trajectory Generation for Wheeled Mobile Systems Using Bézier Curves with Constraints on Velocity, Acceleration and Jerk," *Sensors*, vol. 23, no. 4, pp. 1-16, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[13] Jaafar Ahmed Abdulsaheb, and Dheyaa Jasim Kadhim, "Classical and Heuristic Approaches for Mobile Robot Path Planning: A Survey," *Robotics*, vol. 12, no. 4, pp. 1-35, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[14] Wei Li et al., "CC-BRRT: A Path Planning Algorithm Based on Central Circle Sampling Bidirectional RRT," *Web Information Systems and Applications: 18th International Conference*, Kaifeng, China, pp. 430-441, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[15] Bing-Gang Jhong, and Mei-Yung Chen, "An Enhanced Navigation Algorithm with An Adaptive Controller for Wheeled Mobile Robot Based on Bidirectional RRT," *Actuators*, vol. 11, no. 10, pp. 1-18, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[16] Mehmet Korkmaz, and Akif Durdu, "Comparison of Optimal Path Planning Algorithms," *2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering*, Lviv-Slavske, Ukraine, pp. 255-258, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[17] Yubo Zhang, and Daoxiong Gong, "S-BRRT*: A Spline-based Bidirectional RRT with Strategies under Nonholonomic Constraint," *2021 33rd Chinese Control and Decision Conference*, Kunming, China, pp. 1753-1758, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[18] Xin Shu et al., "Locally Guided Multiple Bi-RRT∗ for Fast Path Planning in Narrow Passages," *2019 IEEE International Conference on Robotics and Biomimetics*, Dali, China, pp. 2085-2091, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[19] Pengkai Wang et al., "ABA*-Adaptive Bidirectional A* Algorithm for Aerial Robot Path Planning," *IEEE Access*, vol. 11, pp. 103521-103529, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[20] Chunlei Wang, Xiao Yang, and He Li, "Improved Q-learning Applied to Dynamic Obstacle Avoidance and Path Planning," *IEEE Access*, vol. 10, pp. 92879-92888, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[21] Karthik Karur et al., "A Survey of Path Planning Algorithms for Mobile Robots," *Vehicles*, vol. 3, no. 3, pp. 448-468, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[22] Liwei Yang et al., "Path Planning Technique for Mobile Robots: A Review," *Machines*, vol. 11, no. 10, pp. 1-46, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[23] Zeynep Gari, Durmuş Karayel, and Murat Erhan Çimen, "A Study on Path Planning Optimization of Mobile Robots Based on Hybrid Algorithm," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 5, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[24] Shang Erke et al., "An Improved A-Star Based Path Planning Algorithm for Autonomous Land Vehicles," *International Journal of Advanced Robotic Systems*, vol. 17, no. 5, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[25] Joshua Buckner et al., "pyCHARMM: Embedding CHARMM Functionality in a Python Framework," *Journal of Chemical Theory and Computation*, vol. 19, no. 12, pp. 3752-3762, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[26] Alex Martelli et al., *Python in a Nutshell*, O'Reilly Media, pp. 1-738, 2023. [Google Scholar] [Publisher Link]

[27] Mustafa Tosun et al., "DAWN-Sim: A Distributed Algorithm Simulator for Wireless Ad-Hoc Networks in Python," *2023 International Conference on Computing, Networking and Communications (ICNC)*, Honolulu, HI, USA, pp. 635-639, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[28] Rainer Otterbach et al., "System Verification Throughout the Development Cycle," *ATZ Worldwide*, vol. 109, pp. 5-8, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[29] Junki Wang et al., "Hybrid Bidirectional Rapidly Exploring Random Tree Path Planning Algorithm with Reinforcement Learning," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 25, no. 1, pp. 121-129, 2021. [CrossRef] [Google Scholar] [Publisher Link]