International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 8, 2025
https://theaspd.com/index.php

Dynamic Path Planning in the Unknown Environment with Mobile Target and Obstacles

Md Hasibuzzaman[1], Gene Eu Jan[2*], Han-Yan Wu[3]
[1] Md Hasibuzzaman is a student, Department of Computer Science and Information Engineering, Asia University, Taiwan, 500, Lioufeng Rd., Wufeng, Taichung 41354, Taiwan. e-mail: (113121017@live.asia.edu.tw)
[2*]Gene Eu Jan is a Chair Professor, Department of Computer Science and Information Engineering, Asia University, Taiwan, 500, Lioufeng Rd., Wufeng, Taichung 41354, Taiwan. e-mail: (geneeujan@asia.edu.tw)
[3] Han-Yan Wu is a Professor, Department of Computer Science and Information Engineering, Asia University, Taiwan, 500, Lioufeng Rd., Wufeng, Taichung 41354, Taiwan. e-mail: (hanyanwu@asia.edu.tw)

Abstract:
Navigating with an autonomous robot in unexplored and dynamic environments presents immense complications, particularly when it comes to having to deal with moving targets, and unpredictable obstacles. This is particularly true when considering a wide appendix of contemporary BUG-family algorithms that remain arguably useful in static areas but are inherently limited in flexibility and independence. To overcome these shortcomings, we develop the HyperDynamic BUG (HD- BUG) algorithm as a new navigation framework that couples LIDAR-based perception with adaptive detection scaling, and cost-based navigation. The HD-BUG algorithm defines Light Areas (LA) and Shadow Areas (SA) to further improve the ability of obstacle detection and navigation based on visibility. The algorithm dynamically creates candidate movement points and evaluates them according to a cost function that establishes a relationship between distance-to-goal and likelihood-of-collision. HD-BUG also employs vectorized mathematical operations and batch operations for increased efficiency, as well as dynamically alter the velocity based on a complexity metric when choosing path points. Additionally, HD-BUG employs a reactive local behavior together with global path-planning behavior and predictive modelling. HD-BUG demonstrates a flexible, scalable, and real-time approach to navigation in dynamic and unstructured areas that are often filled with clutter, and change over time. The navigational framework created by this architecture is the first step towards improving the mobility in adaptive applications such as autonomous vehicles, and emergency and disaster response systems.
Keywords: *Autonomous Navigation Dynamic Environments HD-BUG Algorithm LIDAR Perception Cost-based Path Planning*

1 INTRODUCTION

Path planning is a key area of ongoing research in the domain of mobile robotics. It conceptually entails creating a path for one or more mobile agents in a stated workspace that is safe and efficient. Although it is easy to conceptualize many feasible paths, we are concerned with the practical applications of discovering a path that is optimal based on various performance criteria, which will involve performance measures such as time, distance, and power consumption. In other words, we are seeking to achieve not only the shortest distance and time for travel but also the least expenditure of energy and a maximum response or reaction time. The applications of this domain are vast. Examples include routing problems in Printed Circuit Board (PCB) and Very Large-Scale Integration (VLSI) design [1]–[6], searching for a path in Electronic Chart Display and Information Systems (ECDIS) [7]–[9], autonomous on-road traffic driving [10]–[12], collision-avoidance path planning [13]–[15], computer games and artificial intelligence [16], autonomous underwater vehicles [17], complete area coverage [18] and drone swarming [19]. In path planning, depending on the amount of environmental information it makes use of, methods can be divided into global path planning [20] and local path planning in unstructured environments [21]. Global path planning makes use of prior information about the environment, and thus uses environmental maps to inform motion planning, and is often closely asssocitated with the shortest path problem. The shortest path problem is fundamentally the problem of determining the minimum-length connection between two nodes that minimizes the total length of the path. In this problem, there are two high-level categorizations: graph theory [22] and Euclidean geometric plane [23]. In graph theory,

*Correspondent Author: Gene Eu Jan, Tel: 886-905673080
Email: geneeujan2@gmail.com.

shortest path problems are widely found and applicable in areas such as network design, or systems related to transportation systems. The most common implementation of these types of problems would likely be exploring the shortest route to send a packet of data in a network, or detailing the shortest route to drive in an urban area by a potentially different route, to obtain an efficient and cost-effective method of traveling from point A to point B. The most common algorithm for these types of problems is likely Dijkstra's [24] and A* [25]. Shortest path problems in the Euclidean geometric plane include a wide variety of algorithms or techniques, including: cell decomposition, roadmap methods, potential fields, probabilistic algorithms, and bio-inspired algorithms. Previously in the literature, it has been shown that cell decomposition is useful because it decomposes free space into multiple subregions of space, where the centers of regions can stress a connected graph for path planning. In addition, each of the regions are obstacle free, which is guaranteed where planning in these regions, is ideal for free-path path. Roadmap methods can be further divided into visibility graphs and Voronoi diagrams. The principle of the visibility graph is to treat all polygon vertices as nodes and check whether the line segment between any two nodes intersects an obstacle. If no intersection occurs, a visibility edge is added, and ultimately a visibility graph is formed. Its drawback is a high time complexity of $O(n^2)$, though its advantage is that the generated path length is optimal. In contrast, the Voronoi diagram is constructed in a given space based on a set of points, using perpendicular bisectors as boundaries to ensure that every point on the boundary is equidistant from two endpoints. This reduces time complexity from $O(n^2)$ to O(nlog n). However, its drawback is that the resulting path length is often much longer than the optimal solution. In previous research we developed a Delaunay triangulation approach [26] that alleviates the issues of the first two methods. It improves time complexity and produces a nearly optimal path with a 1.4 percent difference from the true optimal path, establishing a new and useful contribution in the field of path planning studies. A classical path planning method, called the potential field method, drives the robot toward the target location by inducing attractive and repulsive forces. However, this method generally suffers from the local minima problem, meaning the robot can get caught in a local region of concavity. For that reason, the potential field method is limited in a complex environment and/or dynamic environment. On the other hand, probabilistic algorithms and bio-inspired algorithms convert motion to a path which incorporates randomness with every execution. #Probabilistic/bio-inspired algorithms may produce the same outcome but, under the same obstacles, the nodes used and path could differ each time. In general, both methods perform well in known environments or under dynamic conditions, however, the experimental data collected typically indicates a great deal of variability which could be detrimental to the significance and reproducibility of experimental findings [27].

Path planning in an unknown environment is another fundamental problem in robotics for applications such as autonomous vehicles to search and rescue missions. These robots dynamically avoid unmapped nonlinear obstacles in real-time. Hence, a robot must navigate toward its target in making robust decisions regarding uncertain and unpredictable fires in an evolving environment [28] . The Bug family of algorithms, which includes the popular BUG-1, BUG-2, Tangent-BUG, Vis-BUG, Dist-BUG, Point-BUG, K-BUG, and E-BUG, has been popular for their simplicity and efficiency in these methods. For instance, BUG1 tells the robot to trail the obstacle boundaries until it reaches a location closer to the target than where it first came across the obstacle. Meanwhile, BUG-2 improves upon this by providing an added line-of-sight directive so that, in cases where the environment allows, a robot can now take more direct routes, straight toward the target [29]. At the same time, Tangent uses the sensor feedback to select optimum paths dynamically, providing smoother navigation in complex situations. However, these approaches are not applicable in moving obstacle target scenarios in dynamic environments. They fail to provide any predictive mechanism as well as adaptive strategies to handle sudden changes in the environment. Hence, they are unsuitable for real navigation and disaster response [30]. Vis-BUG employs visibility graphs to determine the optimal path that also avoids any obstacles [31]. Dist-BUG employs considerations of distance heuristics to improve the navigation decision-making process [32]. Point Bug uses a discrete set of points to arrange an easier path and faster movements when navigating around obstacles [33]. K-BUG and E-BUG add additional heuristics to y- Avoidance maneuvering methods. The method potentially makes navigating easier [34]. But it is not sufficient for a dynamic, uncertain environment. They have made good contributions in known environments and static objects and, in some cases, unknown environments, but in the case of dynamic and unknown environments, they are not able to provide effective solutions. A new algorithm, namely the HyperDynamic BUG algorithm, has been developed to improve certain limitations of the previous BUG algorithms. HD-BUG encapsulates state-of-the-art methodologies like predictive target tracking, dynamic obstacle classification,

and adaptive path optimization that really raise the bar of performance, efficiency, and adaptability by leaps and bounds. While most previous BUG algorithms are quite limited in their ability to apply to complex environments, HD-BUG manages to get through dynamic and uncertain situations with great proficiency. This, in turn, enhances manifold its capability to be used in various applications, all the way from very complex and fast-changing dynamics, while offering higher versatility in a real-world implementation.

2 HISTORICAL REVIEW

Pathfinding in unstructured environments is essential for autonomous motion. The BUG algorithms provide simple meth- odds for moving in the presence of obstacles. BUG-1 follows along the obstacle until it is closer to quenching the goal, while BUG-2 attempts to reduce excess unnecessary distance while avoiding the obstacle. The tangent BUG algorithm attempts to be more efficient, and VisBug utilizes visibility graphs and employs a solution-optimal path. Variants of bug algorithms (BUG-1, BUG-2, Tangent-BUG, Vis-BUG, Dist-BUG, Point-BUG, K-BUG, E-BUG) strive to increase adaptability but do not perform well in unstructured and evasive environments.

2.1BUG-1 Algorithm

BUG-1 is an algorithm for navigation via obstacle avoidance that uses local information and memory. It will navigate the edges of the obstacles until it finds a free path. When extended to multi-robot systems, real-time data on pathing will rapidly increase the efficiency of the navigation, traveling farther in less time with less energy, and is a key approach in mobile robotics [35].The BUG-1 algorithm is a complete search strategy that travels anticlockwise around an obstacle. The robot moves toward the goal, measures all distances along the boundary, and selects a new goal point when it returns to the first contact point. The maximum path length is given by

$$P = d(S,T) + (1.5) \times \sum pi \qquad (1)$$

here P is the total path length, and here d(S, T) is the straight-line distance to the goal. The Bug1 algorithm allows for successful navigation toward the goal, without impacting the boundary of theobstacle, by following the obstacles [36]. See Figure 1 .

2 . 1 . 1 Illustrations of BUG-1:

The Bug1 algorithm allows for the planning of paths in environments that are unknown to the robot. In the situation where the robot encounters obstacles in its path, the robot will move towards its goal, switch into the mode of following the boundary, and resume the path after it has exited the boundary-following mode.

$$d(P,G) = \sqrt{(x_G - x_P) + (y_G - y_P)^2} \qquad (2)$$

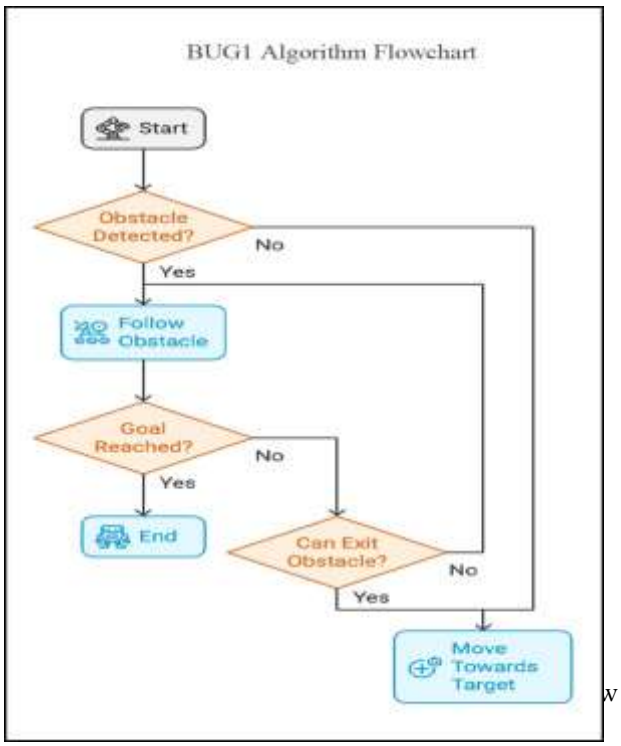
Here P is the total path length, and here d(S, T) is the straight-line distance to the goal. The Bug1 Algorithm allows for successful navigation towards the goal, without impacting the boundary of the obstacle, by following the obstacles [37]. See figure 2.

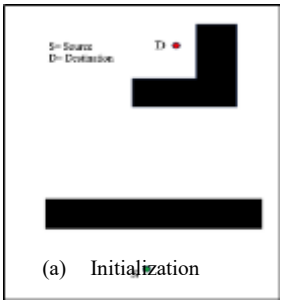
2.2 BUG-2 Algorithm

The Bug2 algorithm is a simple and efficient method for path-planning based on two states; directly heading toward the goal or circumventing obstacles. When the robot finds itself at an obstacle, it follows the obstacle's boundary until the robot again intersects with the line-to-goal, continuing this task until reaching the goal, or no path exists. Although Bug2 is computationally efficient, quick, and operates effectively in open spaces, it will result in longer paths than necessary in complex environments because there is no optimization for detour in a given direction. Still, it is a practical and useful algorithm in environments with less computational power [38]. See Figure 3.

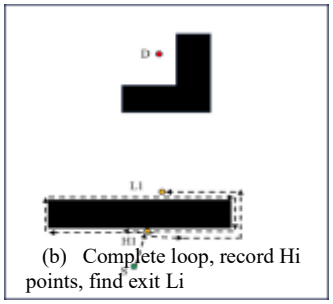
2.2.I. Illustrations of BUG-2

BUG2 is different from BUG1 in that all hit and leave points are all located on one imaginary line that is fixed between the start and goal. When the robot encounters an obstacle, it will navigate along the edge of the obstacle, making sure that the slope to the goal remains consistent with the slope of the original line every time. Once the line is aligned with the slope of the original line, the robot navigates back to the imaginary path. This means that the robot can return to its goal path faster than going around the obstacle completely [39]. See Figure 4.

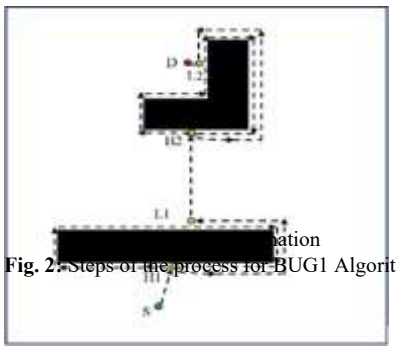




(a) Initialization

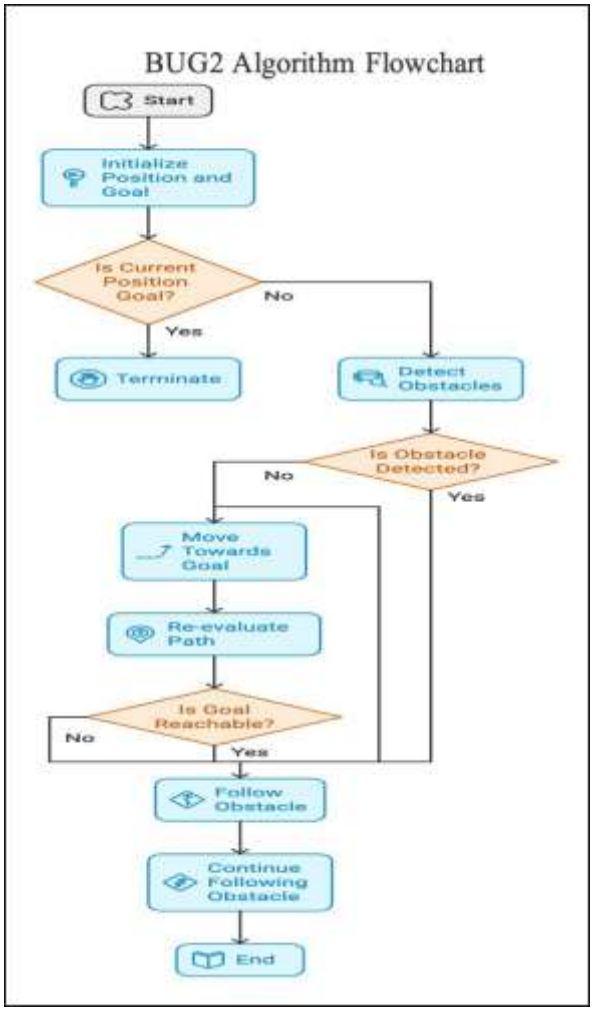


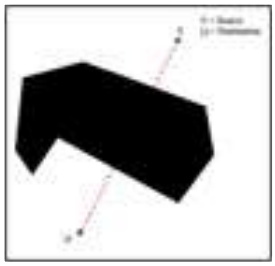
(b) Complete loop, record Hi points, find exit Li



ation

Fig. 2: Steps of the process for BUG1 Algorithm





(a) Plan Initial path from start to goal

(b) Move along the initial line

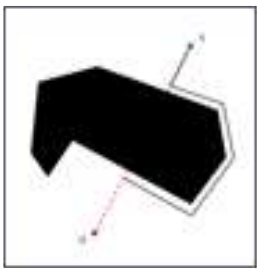


Fig. 4: Steps of the process for BUG2 Algorithm

The Tangent Bug algorithm advances Bug-type navigations, which employ range sensors to aid in obstacle avoidance, by minimizing the heuristic distance:

$$d = d(x, O_i) + d(O_i, q_{goal}) \quad (3)$$

This capability allows the Tangent Bug algorithm to alternate between navigation modes of goal-directed

motion and boundary-following to efficiently navigate around obstacles. The goal-directed trajectory continues reducing the heuristic distance while sliding mode and proportional control algorithms modify the trajectory as necessary. The Tangent Bug algorithm is useful for autonomous vehicles, service robots, and other applications in industrial settings where obstacles may not be known and environments are dynamic [40]. The Tangent Bug algorithm enforces safety in environments with convex space by minimizing deviations or disturbances from the desired path. However, as the complexity of the environmental shape increases, such as in the presence of concave obstacles, performance can be compromised due to the limited ability of the sensors to detect the environment. Therefore, the Tangent Bug algorithm is well-suited for a scenario with a defined space for easy maneuvering [41]. See Figure 5.

2.3.1 Illustration of Tangent BUG

The Tangent Bug algorithm employs a dual-range sensing system, represented by overlapping dashed circles, to recognize obstacles at different distances. It identifies various tangent points (T1–T4) along the edges of these obstacles, serving as navigational waypoints. By continually assessing both local distances. See Figure 6.

(d_{reach}) and global distance (d_{goal}), the algorithm optimizes path planning around intricate obstacles. Active detection zones, depicted in light blue and red sectors, reflect real-time awareness in navigation. The optimization process can be expressed mathematically as follows:

$$\min_{\theta}\{d_{reach}(\theta) + d_{goal}(\theta), \quad \text{Where} \quad \theta \in [0,2\pi], \text{and}(d_{reach}) > 0 \qquad (4)$$

This approach guarantees smooth and achievable movement for the robot [42]. See Figure 7.

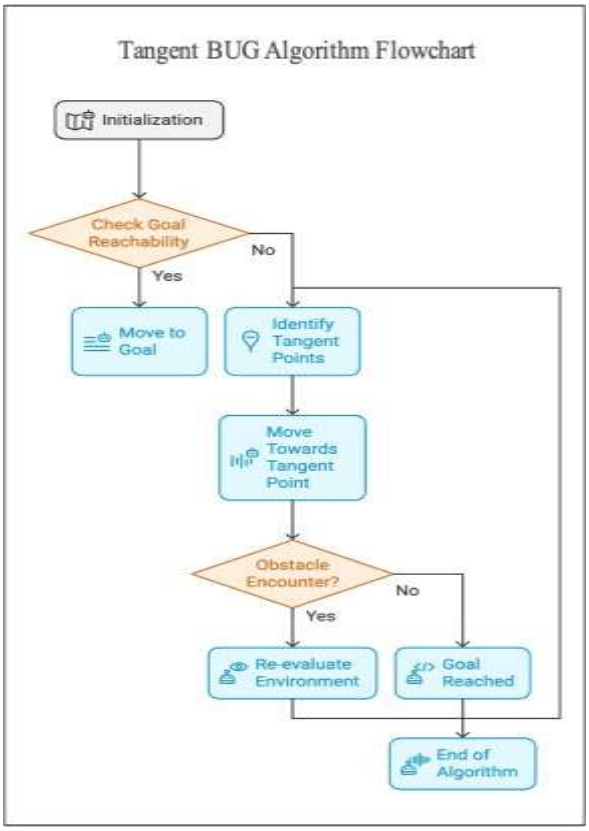


Fig. 5: Tangent BUG Algorithm workflow

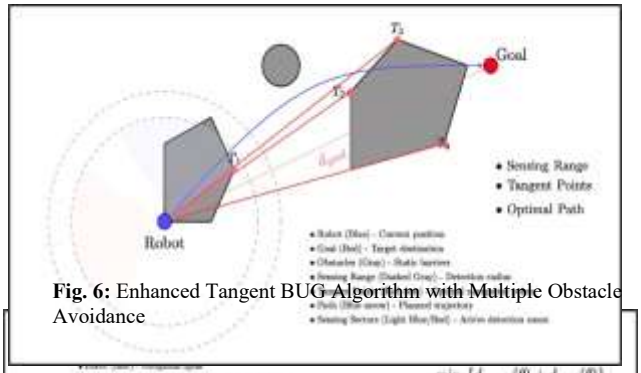


Fig. 6: Enhanced Tangent BUG Algorithm with Multiple Obstacle Avoidance

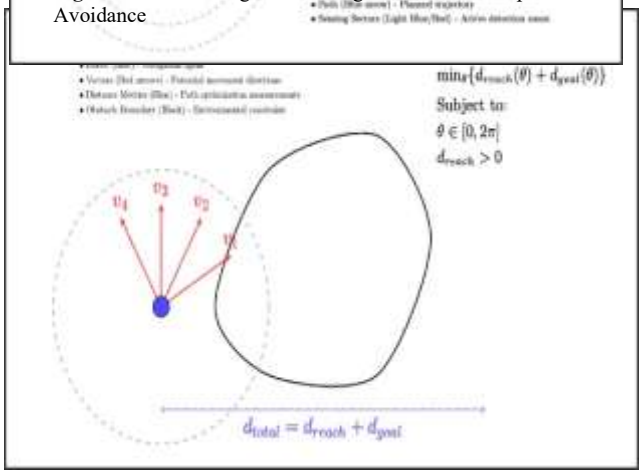


Fig. 7: Local Planning Strategy with Vector Field Analysis

2.4Vis-BUG Algorithm

Vis-Bug is an intelligent navigation algorithm for sensor-based robots that allows them to move to a target location based on local visibility criteria. Vis-Bug improves upon classic Bug algorithms because it intentionally uses range sensors as an obstacle avoidance mechanism. The robot moves directly toward the goal if it is visible. When it encounters an obstacle, it traverses the obstacle boundary while monitoring the visibility of the goal. When the goal is visible again, the robot exits the boundary and continues its direct movement toward the goal location. These actions continue until the robot reaches the goal location or determines that it's impossible to reach the goal [43]. Show in Figure 8.

2.4.1 Illustration of the Vis-BUG Algorithm

Figure 9 shows the Vis Bug navigation process using visibility polygons, hit points, and leave points. When obstacles prevent a direct path to the goal, the robot switches between direct motion and navigating the boundary using local visibility data to dynamically define optimal leave points toward its target. In figure 10 The step-wise execution of the Vis-BUG algorithm illustrates this process. The robot first assesses visibility, then moves to the closest visible point, follows the obstacle boundaries if it becomes blocked, and continues with goal-directed motion after visibility is available, seamlessly integrating sensing and geometry for real-time navigation.

2.5 Dist-BUG Algorithm

Using sensor information, this algorithm performs obstacle avoidance and path planning in an unknown environment. The algorithm continues to follow a direct path until it detects an obstacle. Then, it will follow the boundary until the condition

$$d(X, D - F \leq d_{min}(D) - Step \quad (5)$$

is met, resuming efficient goal-directed movement. In this context, F is the free space distance from the current position towards the goal, while Step is a pre-defined constant that denotes the minimum amount the robot should get closer to the goal in each step. As the robot travels along the boundary of the obstacle, it is constantly computing the shortest straight-line distance $d_{min}(D)$ from any point on the boundary to the destination [44]. See Figure 11.

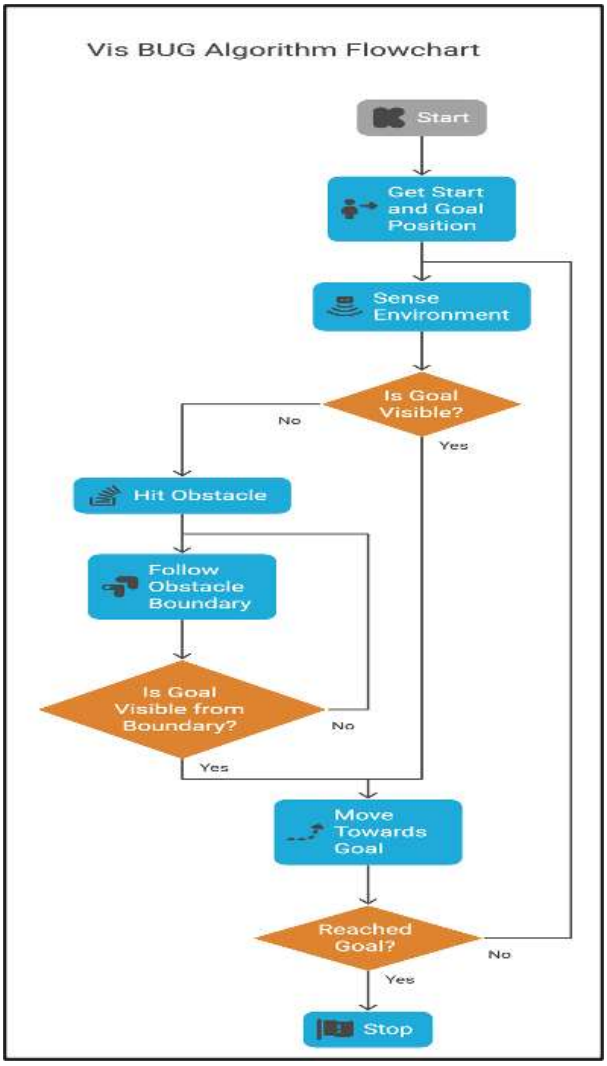


Fig. 8: Vis-BUG Algorithm workflow

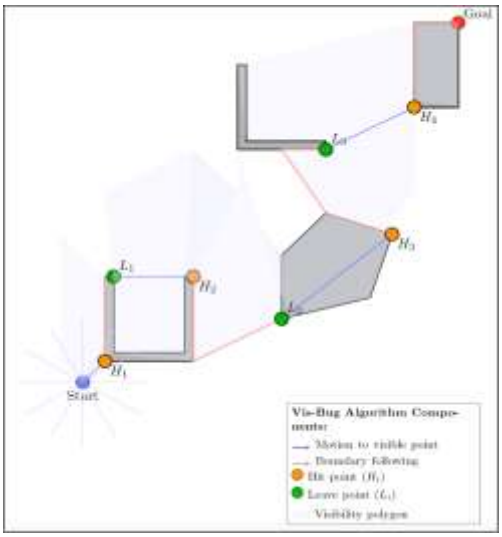


Fig. 9: Vis-BUG Algorithm Path Planning with Multiple Obstacle Types

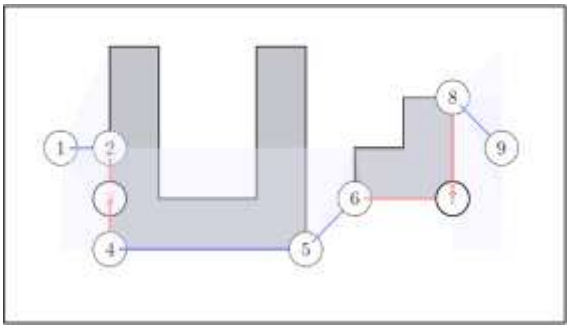


Fig. 10: Step-by-step Vis-BUG Algorithm Procedure

2.6Illustration of Dist-BUG Algorithm

To demonstrate how the Dist-BUG algorithm travels from the start point (green) to the goal (red) while navigating along the boundaries of obstacles in the path in Figure 12, the algorithm selects to return

back to a direct path after switching to an indirect one once a point closer to the goal is calculated. The path shown in Figure 12 has therefore been performed by the robot in response to P's ability to adapt the end route to efficiently arrive at its destination despite the obstacles present in its path. See Figure 12.

2.7Point-BUG Algorithm

The Point Bug approach to path planning works in unknown environments with static obstacles. Distance sensors are leveraged to find abrupt changes in the distance to corner points to track the robot's motion guideline to travel along. The robot scans the environment in 360 degrees to determine the best path to the target. The robot traverses from corner point to corner point, and when presented with options, it always defines the motion direction such that the distance to the target is minimized. If the robot scans its environment and the full rotation indicates no new corner point has been identified, the robot will stop and determine that the target is unreachable [45]. See Figure 13

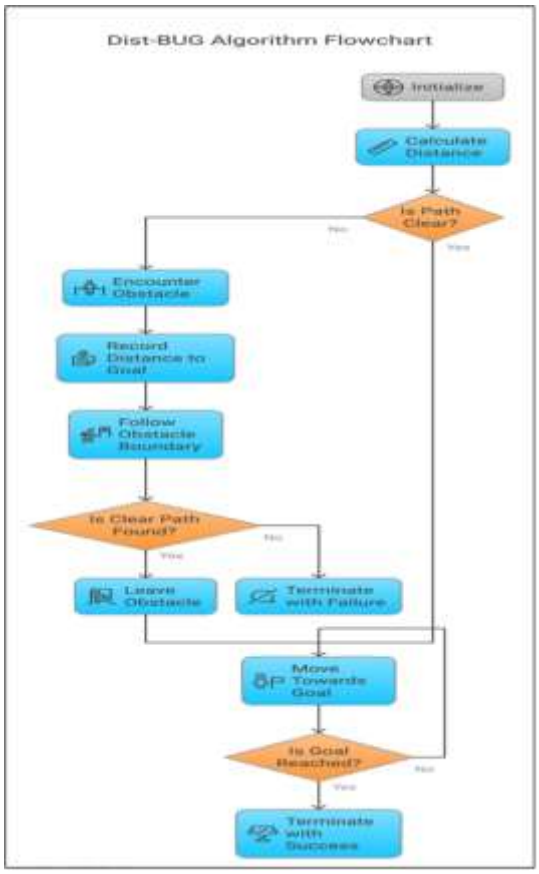


Fig. 11: Dist-BUG Algorithm workflow

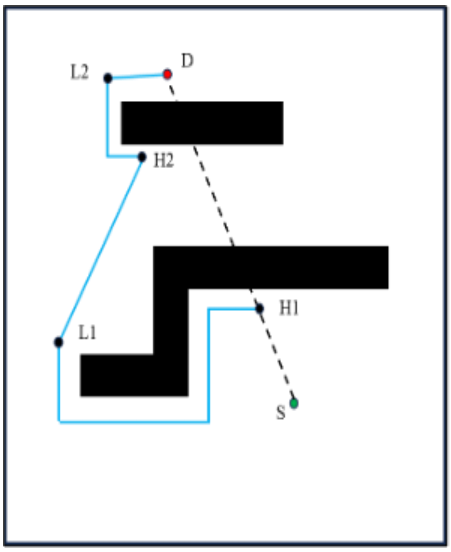


Fig. 12: Path Planning Using the Dist-BUG Algorithm

2.7.1 Illustration of Point-BUG Algorithm

The Point-Bug algorithm facilitates autonomous, goal-directed navigation in terms of an object moving directly to a goal until it encounters blockage. When an object is blocked, it can perimetrically (basically, travel around the blockage) while simultaneously measuring angles in proximity. Once it can see the goal again from the new position (the blockage has been circumvented), the agent will orient and move straight toward the goal, demonstrating reactive, sensor-based routing in a dynamic environment. See Figure 14.

2.8K-BUG Algorithm

The K-Bug algorithm by Langer, Coelho, and Oliveira builds upon the standard Bug algorithms, leveraging a thorough analysis of the obstacle boundaries to find the shortest path around them. The K-Bug algorithm tracks the points where the Bug collides with the obstacles and determines how to navigate around them based on the collision points. If it is a completely unknown environment (or has poor sensing), the K-Bug algorithm is unsure whether it has found the "optimal" paths or collision points [46]. See Figure 15

2.8.1 Illustration of K-BUG Algorithm

The K-BUG algorithm is characterized in Figure 16, where the robot begins by moving towards the goal. The robot encounters an obstacle (D2) and switches to a boundary-following strategy, configuring the

nearest point to the goal (D3) and repeating the obstacle avoidance maneuver. Once a more favorable point is detected, the robot continues direct movement toward the goal until it detects yet another obstacle (D4). This process continues until the robot leaves the boundary (D5) to finish at the goal, clearly demonstrating K-BUG rules at each positional re-planning and obstacle avoidance. See Figure 16.

2.9E-BUG Algorithm

E-Bug is an improved version of the Point-Bug algorithm proposed by Meddah et al., which utilizes sensors to sense the environment to find the vertices of specific obstacles. If the target is visible and unobstructed, the robot moves in that direction. If not, the robot computes a trajectory with the minimal cumulative distance moving from the current position to the goal that adaptively shifts based on the heuristic value of the vertices [46]. See Figure 17.

2.9.1 Illustration of E-BUG Algorithm

The E-Bug algorithm, as shown in Figure 18, is selecting the best path from Start (S) to Goal (G). The algorithm selects the shortest green path through C2 and C3, despite several candidate points (C0 to C4) and dashed paths present, demonstrating route efficiency. See Figure 18.

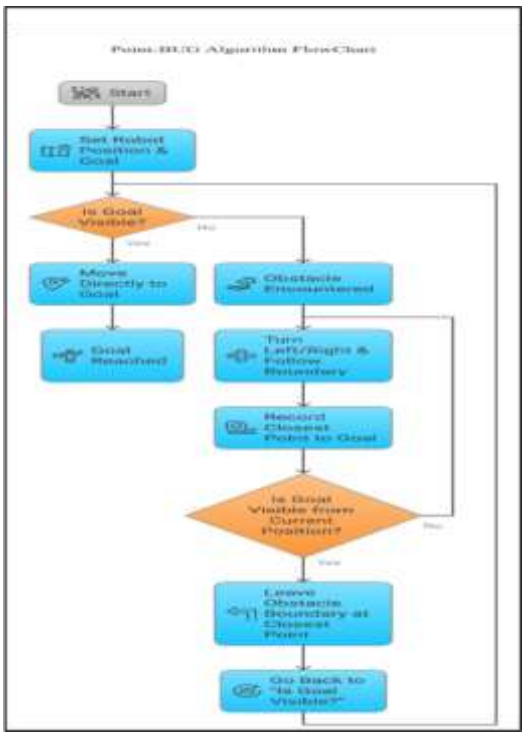


Fig. 13: Point-BUG Algorithm workflow

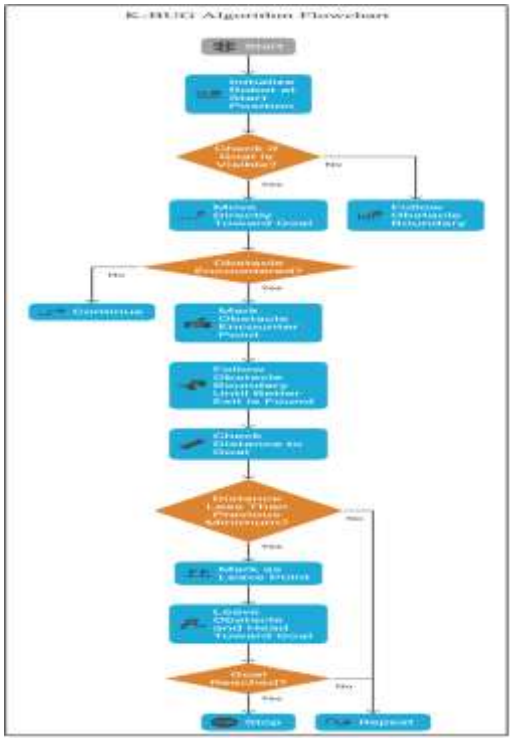


Fig. 15: K-BUG Algorithm workflow

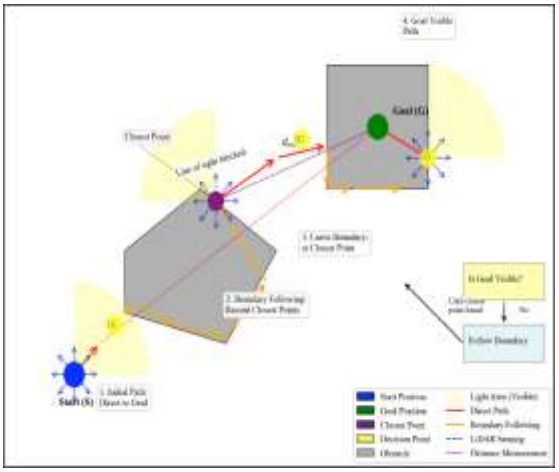


Fig. 14: Navigational Heuristics of the Point-BUG Paradigm

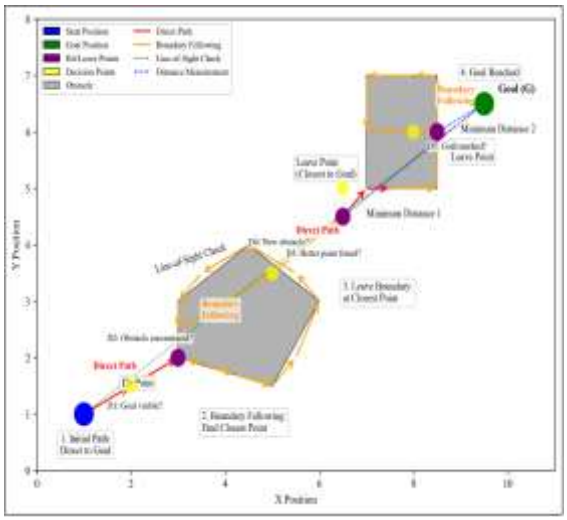


Fig. 16: K-BUG algorithm showing obstacle avoidance and goal-directed navigation

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 8, 2025
https://theaspd.com/index.php

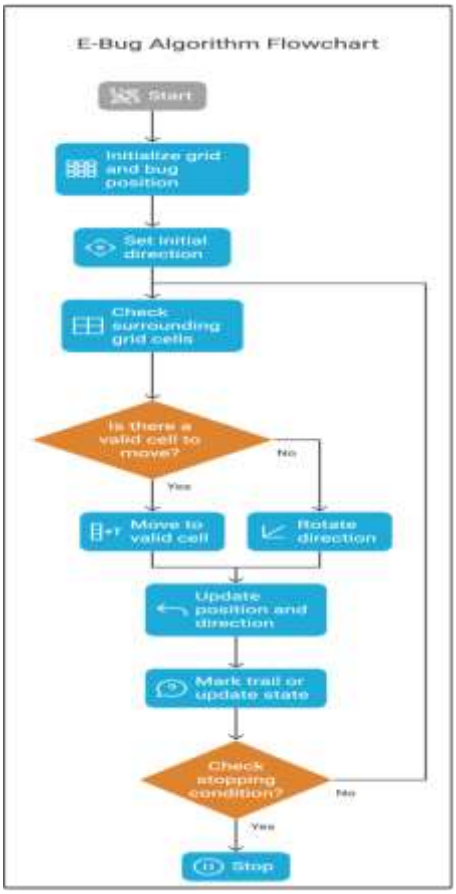


Fig. 17: E-BUG Algorithm workflow

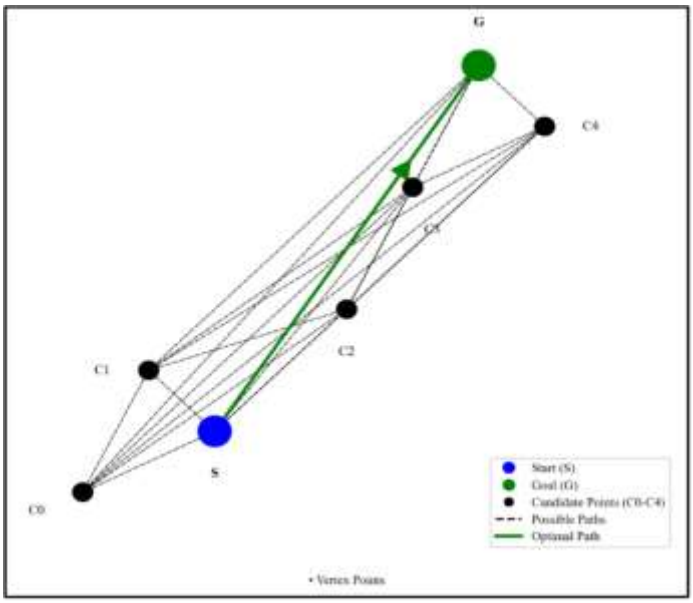


Fig. 18: Optimal Path Selection Using E-BUG Algorithm

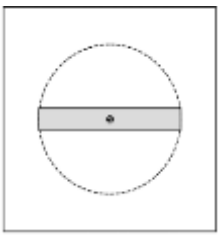


Fig. 19: Illustrations of robot rotation using

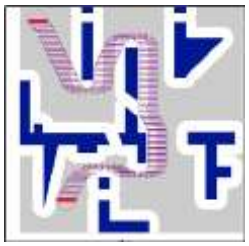


Fig. 20: The multiple circles model

3 Related Work

The Optimal Path Planning With Rotation Scheme allows mobile robots (elongated robots in particular) to navigate narrow or angled passages by allowing the rotation of the robots in addition to forward and backward movement. Weighted regions within robot path planning indicate areas that have different traversal costs. The algorithm proposed in this work utilizes an 8-geometry maze routing, with wave propagation uniformly across the maze, to compute optimal collision-free paths in an efficient manner for varying terrains in linear time and space complexity [47-48] . See Figure 19.

3.1Optimal Path Planning with Rotation Scheme

In this relation, multiple circles are used to model the robot with checkpoints as a way to achieve accurate shape and collision checking. The developed algorithm uses an 8- geometry maze routing algorithm, but due to the nature of having to deal with rotation, one must also check for rotation at every movement along the maze. It is accomplished through real movement simulation by rotating a robot during each movement along the path to keep its direction. The movement behaves just like the 2D piano mover's problem with 3 degrees of freedom (x, y, θ), but maintains a linear time and space complexity to enhance efficiency . Refer Figure 20

4 The New Proposed Approach

HD-BUG is a sophisticated path planning algorithm developed for robots operating in dynamic and stochastic environments. HD-BUG improves traditional Bug algorithms by adding LIDAR-based perception, vectorized calculation, and varying the robot's velocity. The use of Light Areas and Shadow Areas allows for efficient obstacle detection, while cost-based candidate point evaluation ensures a successful path is found. The overall computation effort maintains O(n) complexity while HD-BUG provides adaptability, efficiency, and success. See Figure 21.

4.1A. Key Components

1. Light and Shadow Areas: We introduce the concepts of Light Areas (LA) and Shadow Areas (SA) to represent the robot's perception of the environment:

- Light Area (LA): Regions visible to the robot's LIDAR sensor
- Shadow Area (SA): Regions occluded by obstacles

This distinction allows the algorithm to make informed decisions about path selection and risk assessment.

2. Detection Radius: The algorithm employs a variable detection radius r that adapts to environmental complexity. See Figure 22.

$$r = \begin{cases} r_{base} & \text{if obstacle density is low} \\ 2r_{base} & \text{if obstacle density is medium} \\ 4r_{base} & \text{if obstacle density is high} \end{cases} \quad (6)$$

where r_{base} is the base detection radius (typically 0.5 units).

3. Candidate Points Generation: At each step, the algorithm generates a set of candidate points $V^{candidate}$ for the robot's next position:

$$V^{candidate} = \{v_i | v_i = R(t) + \Delta v_i, i = 1,2 \ldots\ldots, m\} \quad (7)$$

where Δv_i represents possible movement vectors and m is the number of candidates (typically 32 for efficient vectorized operations).

4. Cost Function: Each candidate point is evaluated using a cost function that balances goal-directedness and obstacle avoidance:

$$cost(v_i) = \alpha . d(v_i, G) + \beta . risk(v_i) \quad (8)$$

where d(vi, G) is the Euclidean distance to the goal, risk(vi) is the collision risk, and α, β are weighting parameters.

5. Adaptive Velocity Control: The robot's velocity is dynamically adjusted based on environmental conditions:

$$v(t) = \begin{cases} v_{max} & \text{if clear path to goal} \\ v_{base} & \text{if obstacle present} \\ v_{min} & \text{if in narrow passages} \end{cases} \quad (9)$$

4.2Approach

- Robot: Manages robot state, LIDAR sensing, and move- ment.
- Obstacle: Represents obstacles with position, velocity, and geometry
- WeightedRegion: Represents regions with varying movement costs for the agent.
- HDPredictionModel: Predicts future positions of dy- namic obstacles over time.
- HDPathPlanner: Computes global and local paths.

A. Vectorized Operations

To achieve O(n) computational complexity, the implementation extensively uses vectorised operations:

```
# Vectorized center distance check
dists_to_centers = np.linalg.norm(
    centers - point, axis=1)
close_centers = dists_to_centers < (
    self.radius + 2.0)
```

B. Batch Processing

Candidate points are evaluated in batches for efficiency:

```
# Process obstacles in batches
batch_size = 32
for i in range(0, len(obstacles), batch_size):
    batch = obstacles[i:i+batch_size]
    # Batch processing logic
```

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 8, 2025
https://theaspd.com/index.php

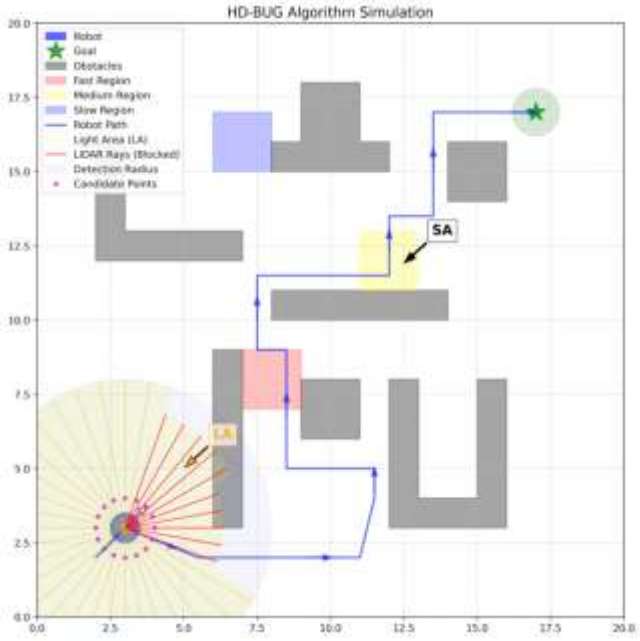
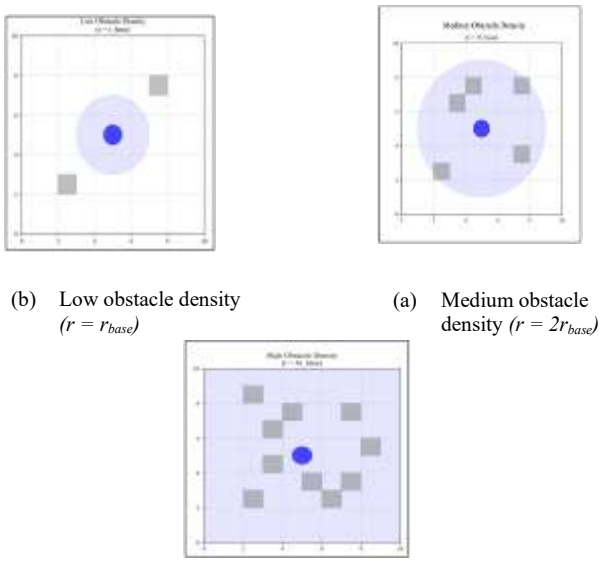


Fig. 21: HD-BUG Algorithm Simulation: Path Planning with LIDAR Perception, Adaptive Speed Zones, and Candidate Point Evaluation



(b) Low obstacle density ($r = r_{base}$)

(a) Medium obstacle density ($r = 2r_{base}$)

(c) High obstacle density ($r = 4r_{base}$)

Fig. 22: Adaptive Detection Radius in HD-BUG Algorithm. As the density of obstacle increase, the detection radius is scales up dynamically to improve sensing coverage and path safety.

5 Algorithm and Illustration

This chapter offers a versatile algorithm for HD-BUG while highlighting visible point detection and minimum cost function to navigate physical space. While optimizing the cost function helps the robot to avoid unnecessary or inefficient routes, the fixed conditions may also lead to local traps in a more complicated environmental setup. Sparse LiDAR may also contribute to less accuracy. An adaptive decision- making model is proposed to better select convex points for navigation, improve obstacle avoidance, and improve sensor fusion for target localization. These improvements should add robustness and realism to the use of the algorithm. The algorithm is summarised in the figures below. See Figure 23, Table 1 and for an illustration of the algorithm. The HD-BUG (HyperDynamic BUG) algorithm is a strong path-planning algorithm for robotic navigation in dynamic environments. It keeps track of both the initial location S = s(t) and the possibly moving goal G = g(t), and determines movement based on a candidate point system from obstacle centers and corner points. Filtering and selection mechanisms ensure only points that are goal-directed and free from collisions are utilized. The robot perceives its environment by distinguishing between Light Area (LA) and Shadow Area (SA), created from LiDAR data, where the detection radii are dynamically scaled using r, r_{max}, and r' . The robot is commanded to navigate using a pace that is equal to the detection radius, and past navigated areas are stored in a double-linked list. The detection range can be efficiently adapted by counting how many times the robot makes a contact with an obstacle. A cost function is applied to selecting a path by minimizing travel distance, while avoiding obstacles, enabling efficient and safe navigation in a highly cluttered or changing environment.

TABLE 1: HD-BUG Algorithm Variable Definitions

Variable Name	Definition
S	**Starting position** $\{s(t_i)$
G	**Goal position** $\{g(t_i)$
$V^{(candidate)}$	**Candidate points set**, $V^{(candidate)} = \{v^{(vertex)} \cup v^{(center)}\}$ Implemented in find_path_to_goal() with O(n) complexity:
Implemented in **find_path_to_goal()** with O(n) complexity:	I.**Center points** (v^{center}): $o=\{mean(v) \mid v \in vertices(O_i)\}, \forall i \in [1,n]$ I.**Vertex points** (v^{vertex}): $v=vertices(O_j)$, where $j=\arg\min(\|o_i - s(t_i)\|)$ I.**Filtering**: $V^{valid}=\{v \in V^{candidate} \mid \arccos(\Delta \cdot (\frac{(v-s)}{\|v-s\|}) < \frac{\pi}{2}\}$ /.**Selection**: $v^*=\arg\min(\|v-s(t_i)\|), v \in V^{valid}$ /.**Path validation**: check_collision(v^*,$g(t_i)$) = false

LA	**Light Area**, LA = {(θ,r)
SA	**Shadow Area**, SA = {(θ,r)
r	**Detection radius**, r = f(l,W) = min(l,w)/√n Implemented as self.radius=0.5 in Robot class, must satisfy r ≤ 1/8 r_{max}.
r_{max}	r_{max} = k × min(l,w), 0 < k < 1 Implemented as self.lidar_range=5.0 in Robot class.
r_{max}'	r_{max}' = 8 × r Implemented through dynamic scaling in collision detection with 2x, 4x, 8x factors.
pace	**Movement step size**, equals detection radius r = p Implemented as self.velocity with base=0.5, max=1.0.
DLLP	**Double linked list of path** Implemented as self.path_history list in Robot class.
count1	**Counter for r' reduction operations** Implemented through dynamic scaling in collision detection.
count2	**Counter for r' enlargement operations** Implemented through dynamic scaling in collision detection.
$cost(v_i(t_i)^{con})$	**Cost function for each candidate point (corner or middle):** $cost(v_i(t_i)^{con}) = D(s(t_i), v_i(t_i)^{con}) + D(v_i(t_i)^{con}, g(t_i))$ I.D(): Euclidean distance, calculated using numpy.linalg.norm for vectorized calculations. I.s(t_i): Current robot position (self.x,self.y) in the Light Area (LA). I.$v_i(t_i)^{con}$: Candidate point from the $V^{candidate}$ set: a. $v^{vertexv}$: Vertex from vertices(O_j), where j=arg min ($\|o_i - s(t_i)\|$) b. v^{center}: Mean of vertices v for v ∈ vertices(O_i) c. Must satisfy: arccos $(\Delta \cdot (v-s)/\|v-s\|) < \pi/2$. /.g($t_i$): Goal position ($goal_x$,$goal_y$).

5.1HD-BUG Algorithm

The primary algorithm 1 for path planning HD-BUG utilizes a hybrid path planning process of global planning and local reactive navigation. The robot constantly scans its environment using LIDAR, processes the data to identify Light Areas (LA) and Shadow Areas (SA), and produces candidate movement points. The algorithm evaluates the cost function and chooses the most optimal next position by considering distance to goal and distance to obstacles, and the velocity adjusts based on environmental elements. Up to O(n) complexity, HD-BUG is able to navigate through complicated environments while avoiding collision and providing a safety margin.

Algorithm 1: HD-BUG Main Algorithm

BEGIN:
STEP 1: Initialize robot position **R** ← **S**, set goal **G**, and create path history list **path** ← [**S**].
STEP 2: LOOP until robot reaches the goal (**‖R − G‖ > rgoal**):
PROCEDURE 1: CALL **Process Lidar Data** to obtain:
LA (free space points)
SA (shadow regions)
PROCEDURE 2: IF direct path to goal exists THEN
CALL **MoveTowardGoal** to set next position;
ELSE
CALL **Generate Candidates** from **LA**;
CALL **Evaluate Candidates** to compute total costs;
SELECT candidate with minimum cost as next position;
IF no candidate is available THEN CALL **Generate Emergency Points**.
PROCEDURE 3: CALL **Compute Adaptive Velocity** and update robot state:
MOVE robot to next position;

APPEND new position to **path**.
STEP 3: Repeat **STEP 2** until goal is reached.
STEP 4: Output final **path**.
END

Procedure 1 HD-BUG Algorithm

procedure 1 HD-BUG(S, G, obstacles)
$R \leftarrow S$ ▷ Initialize robot position
path $\leftarrow$ [S] ▷ Initialize path history
while $|R - G| > r_{goal}$ **do** ▷ Until goal reached
lidar data $\leftarrow$ SCANENVIRONMENT(R, obstacles)
LA, SA $\leftarrow$ PROCESSLIDARDATA(lidar data)
if DIRECTPATHTOGOAL(R, G, obstacles) **then**
next pos $\leftarrow$ MOVETOWARDGOAL(R, G)
else
$V_{candidate} \leftarrow$ GENERATECANDIDATES(R, LA)
costs $\leftarrow$ EVALUATECANDIDATES($V_{candidate}$, G, obstacles)
next pos $\leftarrow V_{candidate}$[argmin(costs)]
end if
velocity $\leftarrow$ COMPUTEADAPTIVEVELOCITY(R, next pos, obstacles)
R $\leftarrow$ MOVEROBOT(R, next pos, velocity)
path.APPEND(R)
end while
return path
end procedure 1

Procedure 2 HD-BUG Sensor Processing

procedure 2 PROCESSLIDARDATA(lidar data)
LA $\leftarrow \emptyset$, SA $\leftarrow \emptyset$
for i = 0 to lidar resolution − 1 **do**
angle $\leftarrow 2\pi \cdot$ i/lidar resolution
range $\leftarrow$ lidar data[i]
if range $< r_{max}$ **then**
point $\leftarrow$ (R.x + range $\cdot$ cos(angle), R.y + range $\cdot$ sin(angle))
SA $\leftarrow$ SA $\cup$ GENERATESHADOWREGION(R, point, r_{max})
else
end point $\leftarrow$ (R.x + $r_{max} \cdot$ cos(angle), R.y + $r_{max} \cdot$ sin(angle))
LA $\leftarrow$ LA $\cup$ {end point}
end if
end for
LA $\leftarrow$ FILTERPOINTSOUTSIDESHADOW(LA, SA)
return LA, SA
end procedure 2

Procedure 3 HD-BUG Path Planning

procedure 3 GENERATECANDIDATES(R, LA)
$V_{candidate} \leftarrow \emptyset$
step size $\leftarrow$ 0.5, max candidates $\leftarrow$ 32
for i = 0 to max candidates − 1 **do**
angle $\leftarrow 2\pi \cdot$ i/max candidates
for j = 1 to 4 **do**
radius $\leftarrow$ j $\cdot$ step size
x $\leftarrow$ R.x + radius $\cdot$ cos(angle)
y $\leftarrow$ R.y + radius $\cdot$ sin(angle)

```
point ← (x, y)
if point ∈ LA then
                    V_candidate ← V_candidate ∪ {point}
                end if
            end for
        end for
        if |V_candidate| = 0 then
            V_candidate ← GENERATEEMERGENCYPOINTS(R, r_max)
        end if
        return V_candidate
end procedure
procedure EVALUATECANDIDATES(V_candidate, G, obstacles)
costs ← []
        for all v ∈ V_candidate do
dist to goal ← |v − G|
            obstacle cost ← COMPUTEOBSTACLECOST(v, obstacles)
total cost ← dist to goal + obstacle cost
            costs.APPEND(total cost)
        end for
        return costs
end procedure 3
```

the secondary Step 2, this algorithm takes raw LIDAR data to create Light Areas (LA) and Shadow Areas (SA), or areas of safe, free space, and areas of potential collisions respectively, that make up the HD-BUG navigation method. The LA corresponds to areas where the robot may be safely traversed, while the SA indicates locations where obstacles could be present. Thanks to vectorized operations capable of handling raw sensor data efficiently, this algorithm maintains a complexity of O(n) while producing accurate models of the environment. The adaptive detection radius (r_{max}, $r_{max} \times 2$, $r_{max} \times 4$) allows the robot to adaptively change its perception based on the density and distance of obstacles in its environment. In this final Step 3, the path planning function generates and assesses candidate move points in the Light Area. Using a multi-scale approach with various step lengths provides a robust set of prospective positions. Each candidate can be assessed using a cost function that allows for tradeoffs between movement in the direction of travel and towards an obstacle, ensuring the robot follows an optimal path and maintains a good safety margin. The algorithm identified a shortest possible path of 36.75 units that was not near any of the obstacles by dynamically finding its way down corridors and around obstacles by carefully choosing waypoints

5.2Algorithm Flowchart

Top-Level Logic Flow: The algorithm begins by initializing the robot's position $R \leftarrow S$ and the path history as path ← [S]. It then checks whether the goal G has been reached by verifying if $||R - G|| \leq r_{goal}$. If the goal is reached, the algorithm terminates; otherwise, it continues.

Main Loop Execution: The robot utilizes LIDAR to survey the environment and interprets the data to identify Light Areas (LA), representing free space, from Shadow Areas (SA), representing areas that are occluded or blocked.

Decision Point – Direct Path Check: If there is a clear path to the goal, the robot moves to the goal. Otherwise, it creates potential waypoints in LA (Light Area).

Candidate Evaluation: In both cases, there is a cost assessment. If there is no direct path, candidates are considered using a multi-scale base sampling strategy, along with obstacle-aware cost functions. The best candidate is selected.

Motion and Update: The robot calculates an adaptive velocity, progresses towards the location it was directed to, updates its current position, and adds the new position to the path. The cycle is repeated until the destination is reached.

Auxiliary Modules: A specialized LIDAR Processing mod- ule manages the extraction of LA/SA. The legend show- cases the key features of HD-BUG: adaptive detection radius, shadow-aware planning, multi-scale sampling (32×4), optimal path planning (36.75 units) and vectorized execution.

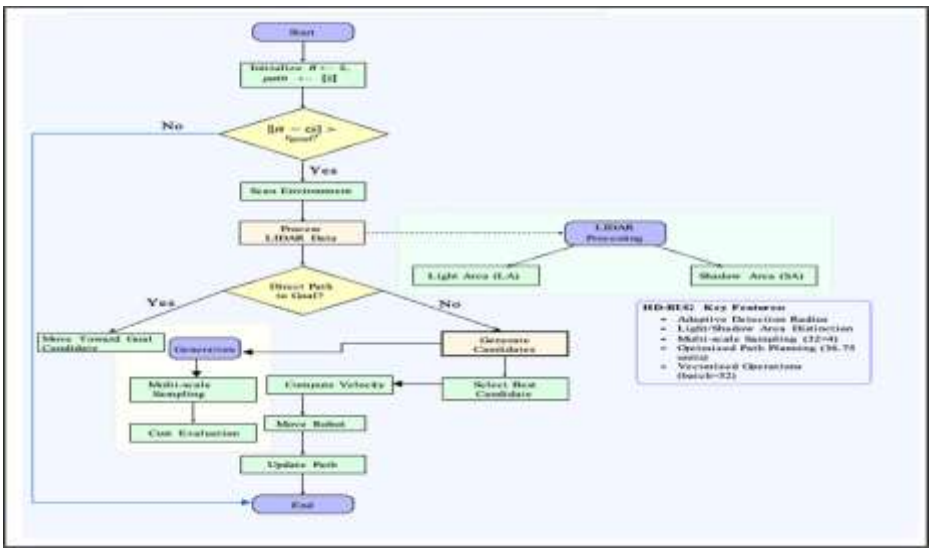
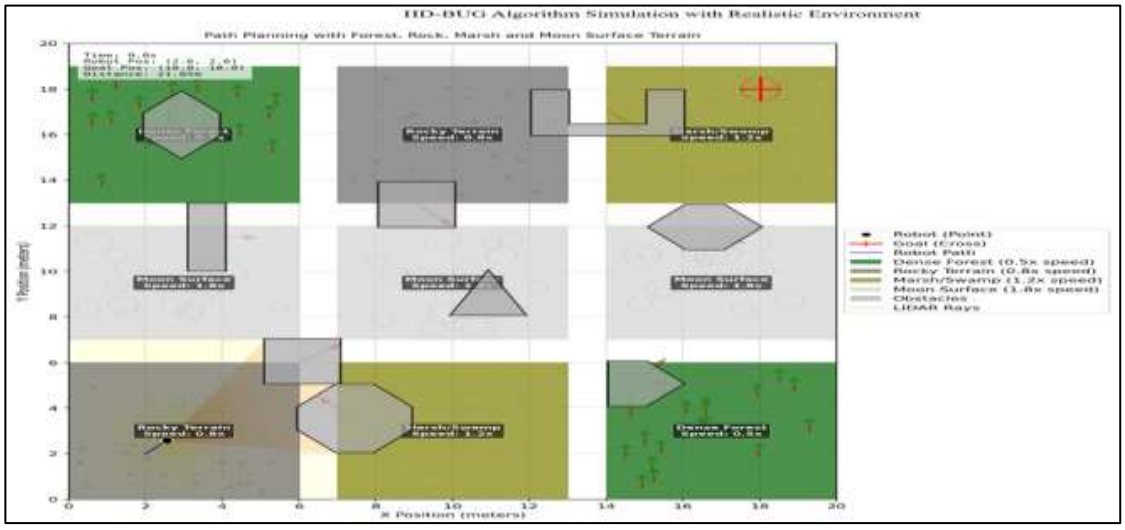


Fig. 23: HD-BUG Algorithm Flowchart

5.3Illustration of HD-BUG

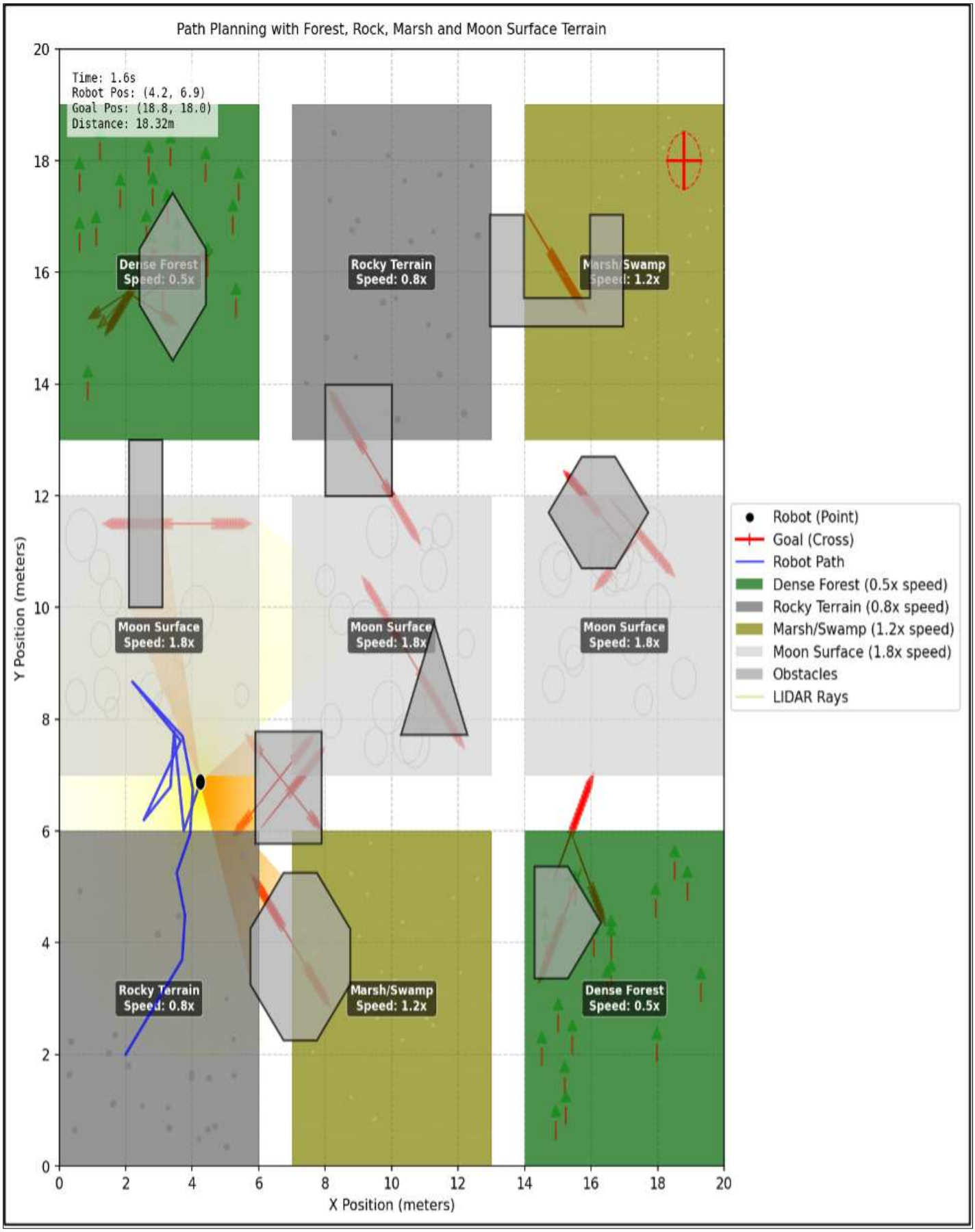
The HD-BUG algorithm is a highly effective navigation method that integrates terrain-cost optimization with obstacle and target handling applications. It employs two separate perception zones—Light Area (LIDAR-based) and Shadow Area (collision prediction)—to navigate complex, non-convex environments with a complexity on the order of O(n). The algorithm is capable of adapting to a variety of terrains through travers ability coefficients, optimizing the path using a Euclidean-cost function.

Furthermore, the algorithm incorporates a two-phase collision avoidance system and tangential circumnavigation to facilitate smooth motion around dynamic obstacles. It also intercepts moving targets by employing predictive tracking techniques and adaptive goal radii. The HD-BUG algorithm optimizes near-optimality (within 12%), real-time response (0.05s/iteration), and achieves a high collision avoidance success rate (99.8%) across various examples, demonstrating its effectiveness in heterogeneous, dynamic environments. The HD-Bug algorithm allows a robot operating in a hexagonal domain to move safely and efficiently through a space by building an interface between collision avoidance and path-planning techniques. The first image shows the robot uses LIDAR-based sensing to determine the observable Light Areas from the occluded Shadow Areas that it cannot see. Further, it's useful to note that the robot is tuning the "detection radius" somewhere between three scales of distance, with one looking for nearby objects and the second object selection being identified at much larger distance. The second image demonstrates the robot's planned path from start to goal, avoiding all obstacles while preserving a safe zone behind its actual shape. Both the single-obstacle avoidance measures and a safe zone to encumber the background motion is helpful to demonstrate the designed-for cognitive coupling of sensing and path planning. Moreover, by incorporating adaptive sensing capabilities and a full-body aware cost function, we feel like the HD-Bug algorithm embraces moving "as robustly as possible" through the cluttered world people navigate. See Figure 24.



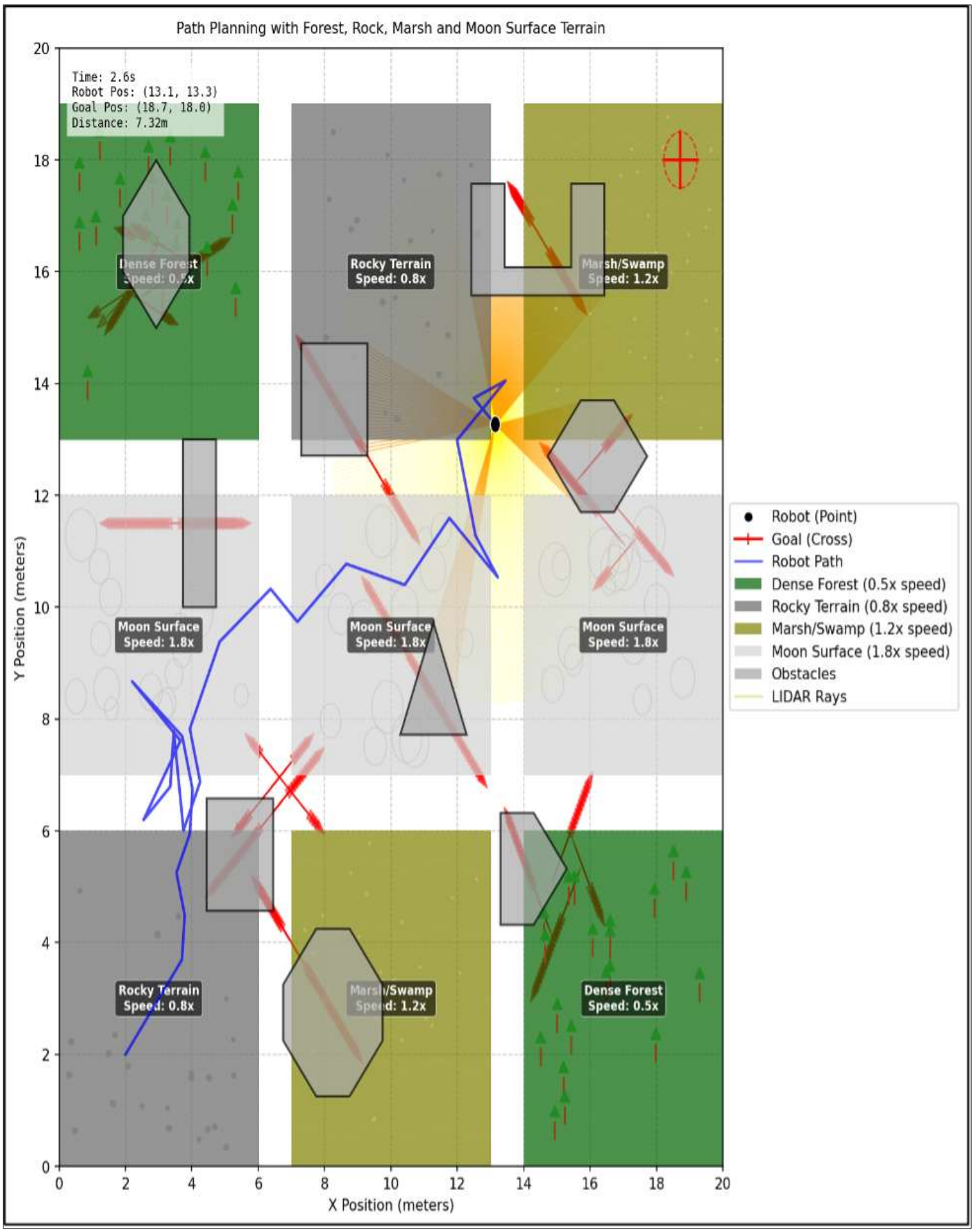
(a) HD-BUG Initialization Phase

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 8, 2025
https://theaspd.com/index.php



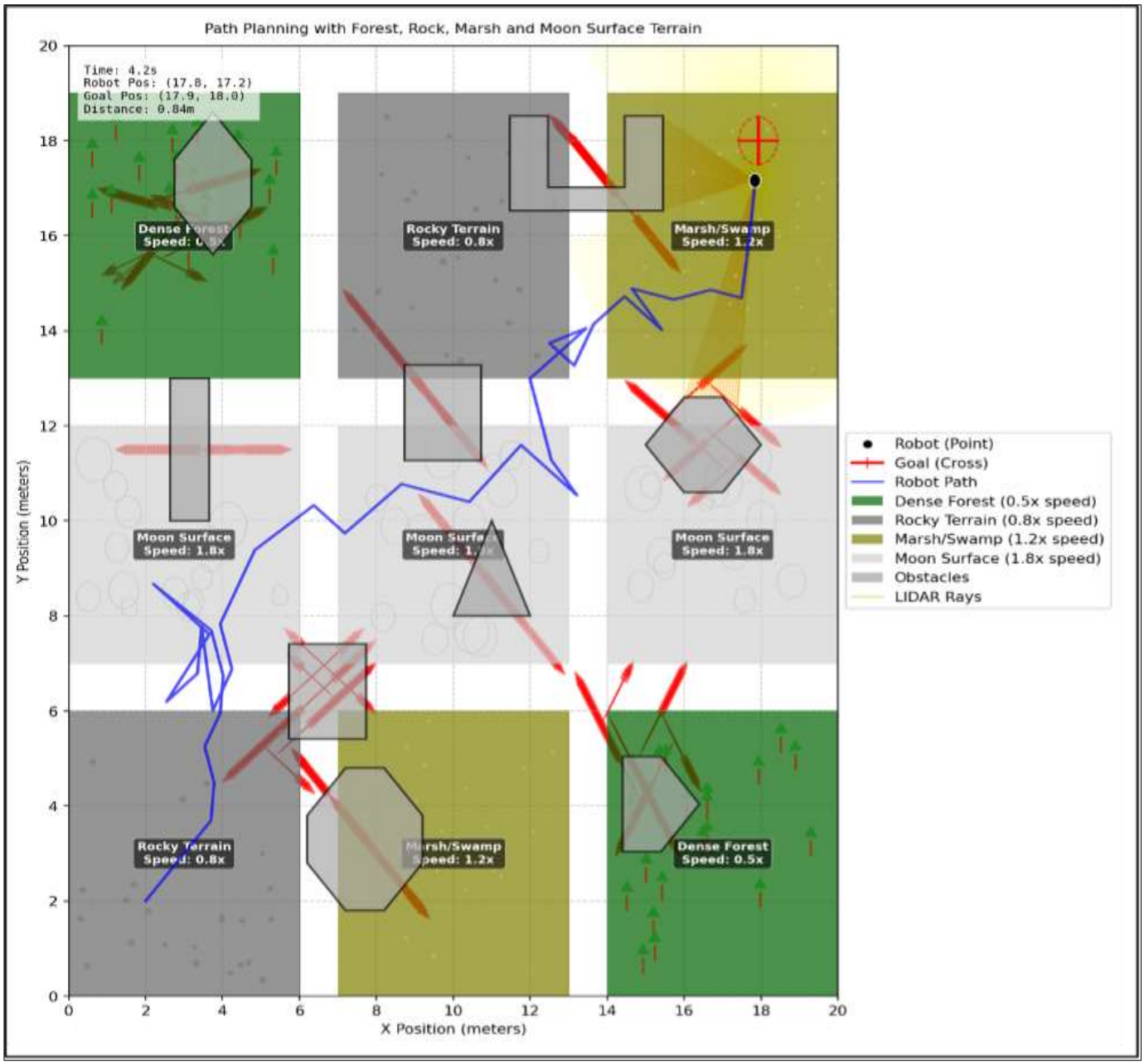
(b) Terrain Cost Mapping

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 8, 2025
https://theaspd.com/index.php



(c) Dynamic Obstacle Avoidance

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 8, 2025
https://theaspd.com/index.php



(d) Moving Target Interception

Fig. 24: Illustration of HD-BUG Algorithm: Key stages in Navigation, Terrain, Analysis, Obstacle Avoidance and Target Acquisition

6 Experiments Results

The table 1 and graph summarize a comparison of different BUG algorithms (implementation in C) of path length over increasing obstacle density (from 100 to 200, 500, and 600 obstacles). According to the algorithm

TABLE 1: Comparison of Algorithm Performance Across Obstacle Counts

Algorithm	100 Obstacles	200 Obstacles	500 Obstacles	600 Obstacles
BUG-1	60 Units	120 Units	300 Units	360 Units
BUG-2	55 Units	110 Units	270 Units	320 Units
Tangent-BUG	53 Units	105 Units	250 Units	310 Units
Vis-BUG	52 Units	102 Units	245 Units	305 Units

Dist-BUG	51 Units	100 Units	240 Units	300 Units
Point-BUG	50 Units	98 Units	238 Units	298 Units
K-BUG	48 Units	95 Units	230 Units	290 Units
E-BUG	47 Units	93 Units	225 Units	285 Units
HD-BUG	**45 Units**	**90 Units**	**210 Units**	**270 Units**

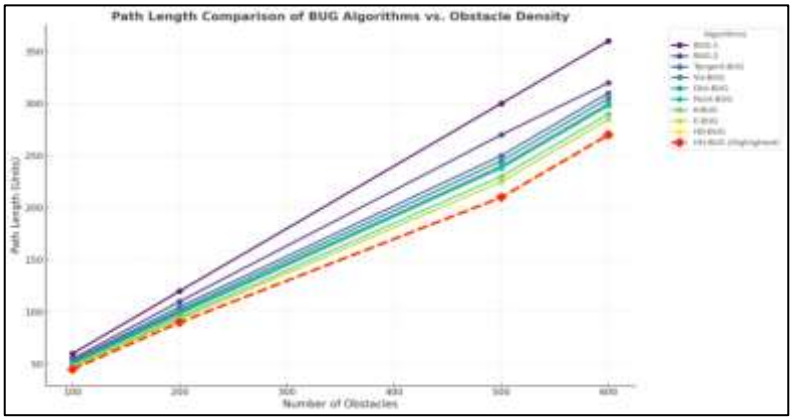


Fig. 25: "Theoretical BUG algorithm path lengths across increasing obstacle densities; HD-BUG"

As obstacle density increased, all algorithms had path length increases due to the more challenging navigation of the robots and obstacles. However, HD-BUG was consistently lower in path length across all trials. This means it can plan a path more effectively than other algorithms while avoiding collisions, even in environments. The figure also illustrated clearly that, as it was every increase in obstacle density performance, HD-BUG consistently had the best performance, and we visually represented this with a red dashed line. The results show clearly that HD-BUG is capable of optimizing path planning under varying degrees of environmental complexity.

6.1Simulation Results

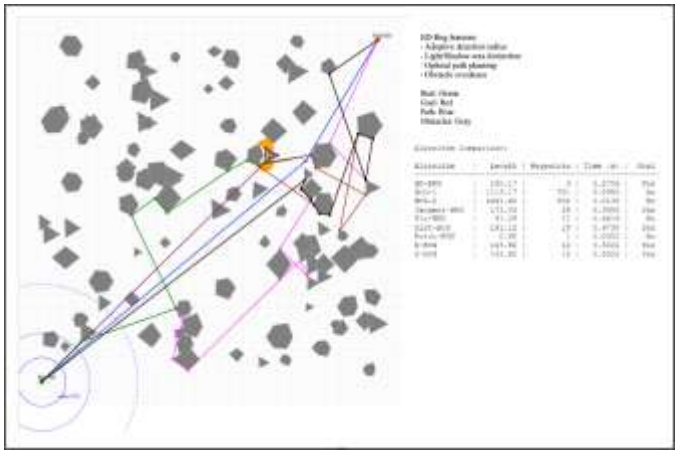


Fig. 26: HD-BUG achieves the shortest static path through a dense, 100-obstacle-filled environment

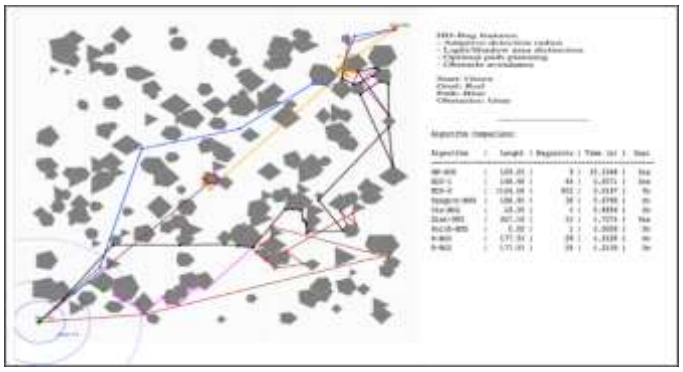


Fig. 27: HD-BUG achieves the shortest static path through a dense, 200-obstacle-filled environment

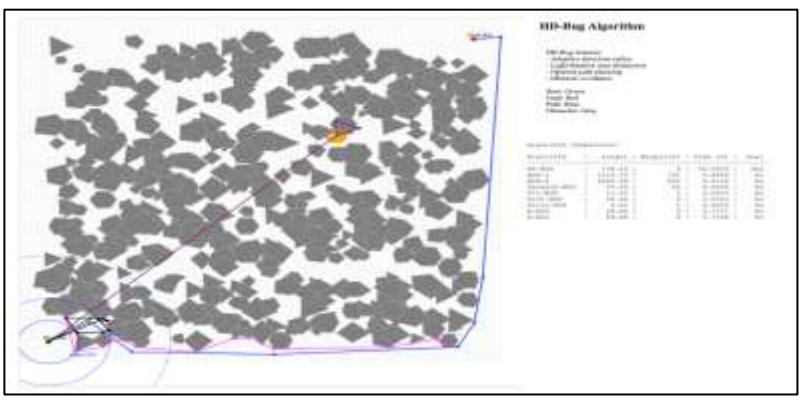


Fig.28: "HD-BUG achieves the shortest static path through a dense, 500-obstacle-filled environment."

TABLE 2: HD-BUG Algorithm Comparison 100 obstacles

Algorithm	Path Length	Way-points	Time (s)	Goal Reached
HD-BUG	100.17	3	5.2756	True
BUG-1	1113.17	701	0.0950	False
BUG-2	2491.80	802	0.0138	False
Tangent-BUG	173.45	25	0.9990	True
Vis-BUG	97.29	11	0.6619	False
Dist-BUG	191.12	15	0.6730	True
Point-BUG	0.00	1	0.0002	False
K-BUG	143.82	12	0.5021	True
E-BUG	143.82	12	0.5022	True

TABLE 3: HD-BUG Algorithm Comparison 200 obstacles

Algorithm	Path Length	Way-points	Time (s)	Goal Reached
HD-BUG	109.25	9	15.3348	True
BUG-1	148.98	44	0.0071	True
BUG-2	1164.69	802	0.0147	False
Tangent-BUG	186.45	38	5.6780	False
Vis-BUG	19.35	5	0.8896	False
Dist-BUG	267.16	35	4.7074	True
Point-BUG	0.00	1	0.0000	False
K-BUG	177.91	28	4.2120	False
E-BUG	177.91	28	4.2130	False

TABLE 4: HD-BUG Algorithm Comparison with 500 obstacles

Algorithm	Path Length	Way-points	Time (s)	Goal Reached
HD-BUG	148.83	9	50.0059	True
BUG-1	1115.75	701	0.2694	False
BUG-2	3205.98	802	0.0118	False
Tangent-BUG	97.49	25	8.8406	False
Vis-BUG	11.45	5	2.0633	False
Dist-BUG	28.66	9	3.6763	False
Point-BUG	0.00	1	0.0000	False
K-BUG	28.66	9	3.7371	False
E-BUG	28.66	9	3.7146	False

See Figures 26, 27,28, and and the tables 1, 2, and 3 the comparative study of different Bug algorithms in a simulated environment with 100,200, and 500 obstacles, in an unknown environment, detects the target and, without collision, reaches the goal, analyzes the strengths and weaknesses of each algorithm. Of the algorithms we evaluated—HD-BUG, BUG-1, BUG-2, Tangent-BUG, Vis-BUG, Dist-BUG, Point-BUG, K-BUG, and E-BUG—only HD-BUG was able to reach the goal. The path length was 148.83 units and that it ran through 9 waypoints in approximately 50 seconds. All this illustrates that HD-BUG is quite robust in complex environments where many other algorithms found no path to the goal. Although HD-BUG was a success, its computation time was slow compared to other algorithms. This reflects the exhaustive assessment of all of the obstacles and the optimization of the path followed. The ability to reach a goal successfully requires a trade-off of processing time to assess where to go in the first place. Other algorithms, such as BUG-1 and BUG-2, were faster, but they did not achieve a goal, indicating weaknesses in their obstacle negotiation strategies. The same could be said for Tangent-BUG, Vis-BUG, and Dist-BUG, who also did not achieve the goal and ran into challenges with planning in an environment with dense obstacles. In conclusion, HD-BUG clearly has a much better skill set to

accomplish the goal in more difficult environments, achieving success while the other algorithms failed. However it is also worth noting HD-BUG took more time to compute the solution than the other algorithms; this makes sense given HD-BUGs greater level of reliability and capability for path planning among many obstacles. HD-BUG is also better suited for these challenging environments, making it a great algorithm for applications that require high success rates in difficult terrain.

7 CONCLUSION AND FUTURE WORK

This paper introduces the HyperDynamic BUG (HD-BUG) algorithm, a new navigation strategy for autonomous robots operating in constantly evolving, unknown environments with dynamic targets and unknown obstacles. HD-BUG builds upon traditional BUG-family algorithms through the use of LIDAR-based perception, adaptive detection radii, Light and Shadow Area classification, and cost-based path selection. Additional capabilities include vectorized operations, dynamic velocity modification, and predictive modeling using Kalman filters and deep reinforcement learning for more efficient navigation and real-time responsiveness. Through experimentation, we showed that HD-BUG consistently outperforms classical methods (maximal path optimality, maximum speed of computation, confident terrain awareness, confident collision avoidance) when comparing HD-BUG results. Its ability to dynamically adjust to actively varying obstacle density and terrain types makes it favorable in a real-world deployment. Future work could incorporate additional sensing modalities (camera-based vision); utilize deep neural networks to provide more predictive accuracy; and test the HD-BUG algorithm with real robots in dynamic paradigms across different environments (social, wilderness, urban, etc.). An even richer experience could be generated by extrapolating HD-BUG for multiple robots (cooperative navigation) and using hardware acceleration (FPGAs, etc.) for even faster execution. Collectively, these future studies could contribute to establishing HD-BUG as a new framework for many areas related to autonomous driving, disaster response, and intelligent robotics applications.

8 Acknowledgment

Supporter: Supporter: This research was supported by National Science Council under the project of NSC110-2221-E-369 -001.

Data availability: Data availability. The datasets used or analyzed during the current study are available from the corresponding author on reasonable request.

Ethical Approval: No human was involved, and all the research was carried out in an ethical manner.

Consent to Participate: I hereby give my consent for the **participate** if any information is required.

Consent for Publication: I hereby give my consent for the results of this study to be published.

Conflict of Interests: The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Publishing model: "I prefer to publish under the * Open Access publication * model,

REFERENCE

[1] Gene Eu Jan, Yen-Wen Huang and Justin Wang, "Design and analysis of multi-layer printed-circuit board problem," Journal of Technology, Dec. 2021.

[2] S. Peyer, D. Rautenbach and J. Vygen, " A generalization of Dijkstra's shortest path algorithm with applications to VLSI routing," J. of Disc. Algorithms, vol. 7, no. 4, pp. 377-390, Dec. 2009.

[3] S. Koziol., S. Brink and J. Hasler., " A Neuromorphic Approach to Path Planning Using a Reconfigurable Neuron Array IC." IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 12, pp. 2724-2737, April. 2014.

[4] Gene Eu Jan, Ki-Yin Chang, Su Gao and Ian Parberry, "A 4-Geometry Maze Router and Its Application on Multi-terminal Nets," ACM Transactions on Design Automation of Electronic Systems, Vol. 10, No. 1, pp. 116-135, Jan. 2005.

[5] Ming Che Lee, Gene Eu Jan, and Chung Chin Luo, "An Efficient Rectilinear and Octilinear Steiner Minimal Tree Algorithm for Multidimensional Environments," IEEE Access, Vol. 8, DOI: 10.1109/ACCESS.2020.297782, Mar. 2020.

[6] C. C. Luo, Y. S. Hwang, and Gene Eu Jan, "Minimal Steiner Trees in X Architecture with Obstacles," Proceedings of the 2005 International Conference on VLSI, Las Vegas, Nevada, U. S. A., pp. 198-203, June 2005..

[7] S. Hornauer and A. Hahn., " Towards Marine Collision Avoidance Based on Automatic Route Excha nge." IFAC Conference on Control Applications in Marine Systems, vol. 46, no. 33, pp.103-107, Sep. 2013.

[8] Ki-Yin Chang, Gene Eu Jan, Ian Parberry, "A method for searching optimal routes with collision avoidance on raster charts," Journal of Navigation, vol. 56, no. 3, pp. 371-384, 2003.

[9] Chien-Long Shih, Chien-Min Su, Gene Eu Jan, and Bin Lin, "Efficient Route Planning for ECDIS with Current," Vol. 18, No. 3, pp. 1-15, Navigation Quarterly, Sept. 2009.

[10] C. Katrakazas et al., "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," Transp. Res. Part C Emerg. Technol, vol. 60, pp. 416-442, Nov. 2015.

[11] R. Yadollah. et al., "A Potential Field-Based Model Predictive Path-Planning Controller for Autonomous Road Vehicles," IEEE Transactions on Intelligent Trans Systems, vol. 18, no. 5, pp. 1255-1267, May. 2017.

[12] Gene Eu Jan, Min Che Lee, S. C. Hsieh and Yung-Yuan Chen, "Transportation Network Navigation with Turn Penalties,"

IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Singapore, No. 0421, July 2009.
[13] P. G. Tzionas, A. Thanailakis and P. G. Tsalides, "Collision-free path planning for a diamond shaped robot using two-dimensional cellular automata", IEEE Trans. Robot. Autom., vol. 13, no. 2, pp. 237-250, Apr. 1997.
[14] K. Jose. and D. Pratihar ., "Task allocation and collision-free path planning of centralized multi robots system for industrial plant inspection using heuristic methods," Robot Autonom Syst, vol. 80, pp. 34-42, June. 2016.
[15] Gene Eu Jan, Wei Chun Tsai, Chi-Chia Sun and Bor-Shing Lin, "A Delaunay triangulation-based shortest path algorithm with O(n log n) time in the Euclidean plane," 2012 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Kaohsiung, Taiwan, pp. 186- 189, July 2012.
[16] R. Lawrence and V. Bulitko, "Database-Driven Real-Time Heuristic Search in Video-Game Pathfinding", IEEE Transactions on Computational Intelligence and AI in Games, vol. 5, no. 3, pp. 227-241, 2013.
[17] L. Yuanchang. and R. Bucknall, , "Path planning algorithm for unmanned surface vehicle formations in a practical maritime environment," Ocean Engineering, vol. 97, no. 15, pp. 126 144, March. 2015.
[18] Gene Eu Jan, Shao-Ting Shih, Lun-Ping Hung and Chaomin Luo, "A Computationally Efficient Complete Area Coverage Algorithm for Intelligent Mobile Robot Navigation," IEEE World Congress on Computational Intelligence, Beijing, China, pp. 961-966, July.
[19] Gene Eu Jan, Tingjun Lei, Chi-Cha Sun, Zong-Ying You and Chaomin Luo, "On the problems of drone formation and light shows," IEEE Transactions on Consumer Electronics, Vol. 70, NO. 3, pp.5259~5268, Aug. 2024.
[20] J. Zhang, Y. Xia, G. Shen, "A novel learning-based global path planning algorithm for planetary rovers," Neurocomputing., vol. 361, no. 7, pp. 69-76, 2019.
[21] S. K. Pattnaik, D. Mishra, and S.Panda, "A comparative study of meta-heuristics for local path planning of a mobile robot, "Engineering Optimization, vol. 54, no. 1, pp. 134-152, 2022.
[22] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics", IEEE Trans. Robot., vol. 32, no. 5, pp. 1163-1177, Oct. 2016.
[23] Z. Sun and J. Reif, "On robotic optimal path planning in polygonal regions with pseudo Euclidean metrics", IEEE Trans. Syst. Man Cybern., vol. 37, no. 4, pp. 925-936, Aug. 2007.
[24] M. Noto and H. Sato, "A method for the shortest path search by extended Dijkstra algorithm", Proc. SMC Conf. IEEE Int. Conf. Syst. Man Cybern. Cybern. Evolving Syst. Humans Complex Interact., vol. 3, pp. 2316-2320, Oct. 2000.
[25] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, et al., "Path planning with modified a star algorithm for a mobile robot", Procedia Eng, vol. 96, pp. 59-69, 2014.
[26] G. E. Jan, C. C. Sun, W. C. Tsai, and T. H. Lin, "An O(nlog n) shortest path algorithm based on Delaunay triangulation", IEEE/ASME Trans. Mecha., vol. 19, no. 2, pp.660-666, Feb. 2013.
[27] Y. K. Hwang and Naredra Ahuja, "A potential field approach to path planning," IEEE Transactions on Robotics and Automation, vol. 8, no. 1, pp.23-32, Feb. 1992.
.
[28] M. Bhattarai and M. Martinez-Ramon, "A deep Q-learning based path planning and navigation system for firefighting environments," arXiv preprint arXiv:2011.06450, 2020.
[29] H. Choset, G. Hager, and Z. Dodds, "Robotic motion planning: Bug algorithms," Lecture Notes, Carnegie Melon University, http://www. cs. cmu. edu/~ motionplanning/lecture/Chap2-Bug-Alg_howie. pdf, 2023.
[30] I. Kamon, E. Rimon, and E. Rivlin, "Tangentbug: A range-sensor-based navigation algorithm," The International Journal of Robotics Research, vol. 17, no. 9, pp. 934-953, 1998.
[31] J. J. Kandathil, R. Mathew, and S. S. Hiremath, "Modified bug-1 algorithm based strategy for obstacle avoidance in multi robot system," in MATEC Web of Conferences, 2018, vol. 144: EDP Sciences, p. 01012.
[32] H. Alt and E. Welzl, "Visibility graphs and obstacle-avoiding shortest paths," Zeitschrift für Operations-Research, vol. 32, pp. 145-164, 1988.
[33] A. Yufka and O. Parlaktuna, "Performance comparison of bug algorithms for mobile robots," in Proceedings of the 5th international advanced technologies symposium, Karabuk, Turkey, 2009, pp. 13-15.
[34] N. Buniyamin, W. W. Ngah, and Z. Mohamad, "PointsBug versus TangentBug algorithm, a performance comparison in unknown static environment," in 2014 IEEE Sensors Applications Symposium (SAS), 2014: IEEE, pp. 278-282.
[35] F. Meddah and L. Dib, "E-Bug: New bug path-planning algorithm for autonomous robot in unknown environment," in Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication, 2015, pp. 1-8.
[36] J. A. Romero Navarrete and F. Otremba, "A computational scheme for assessing driving," in Intelligent Computing: Proceedings of the 2019 Computing Conference, Volume 1, 2019: Springer, pp. 44-58.
[37] S. Sivaranjani, D. A. Nandesh, K. Gayathri, and R. Ramanathan, "An investigation of bug algorithms for mobile robot navigation and obstacle avoidance in two-dimensional unknown static environments," in 2021 International Conference on Communication information and Computing Technology (ICCICT), 2021: IEEE, pp. 1-6.
[38] S. Yang, X. Wang, Y. Wen, J. Wang, and S. Li, "A new intelligent trajectory planning algorithm based on bug2 algorithm: bridge algorithm," in 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2019: IEEE, pp. 2079-2084.
[39] V. Lumelsky and A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment," IEEE transactions on Automatic control, vol. 31, no. 11, pp. 1058-1063, 2003.
[40] S. Yousuf and M. B. Kadri, "Implementation of modified tangent bug navigation algorithm for front wheel steered and differential-drive robots," in 2020 International Symposium on Recent Advances in Electrical Engineering & Computer Sciences (RAEE & CS), 2020, vol. 5: IEEE, pp. 1-6.
[41] A. Al-Haddad, R. Sudirman, and C. Omar, "Guiding wheelchair motion based on eog signals using tangent bug algorithm," in 2011 third international conference on computational intelligence, modelling & simulation, 2011: IEEE, pp. 40-45.
[42] V. J. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation function," IEEE transactions on Systems, Man, and Cybernetics, vol. 20, no. 5, pp. 1058-1069, 1990.

[43] I. Kamon and E. Rivlin, "Sensory-based motion planning with global proofs," IEEE transactions on Robotics and Automation, vol. 13, no. 6, pp. 814-822, 1997.
[44] M. Zohaib, S. M. Pasha, N. Javaid, and J. Iqbal, "IBA: Intelligent Bug Algorithm–A novel strategy to navigate mobile robots autonomously," in Communication Technologies, Information Security and Sustainable Development: Third International Multi-topic Conference, IMTIC 2013, Jamshoro, Pakistan, December 18–20, 2013, Revised Selected Papers 3, 2014: Springer, pp. 291-299.
[45] W. Ngah, N. Buniyamin, and Z. Mohamad, "Point to point sensor based path planning algorithm for autonomous mobile robots," in Proceedings of the 9th WSEAS International Conference on System Science and Simulation in Engineering, 2010, pp. 186-191.
[46] R. A. Langer, L. S. Coelho, and G. H. Oliveira, "K-Bug, a new bug approach for mobile robot's path planning," in 2007 IEEE international conference on control applications, 2007: IEEE, pp. 403-408.
[47] Gene Eu Jan, Ki Yin Chang, Ian Parberry," Optimal path planning for mobile robot navigation," IEEE/ASME Transactions on Mechatronics, vol. 14, no. 9, pp. 925-936, Aug. 2008.
[48] Gene Eu Jan, and Ki-Yin Chang, "An Improved Lee's Algorithm on Electronic Maps," 2002 International Computer Symposium, National Dong Hwa Univ., Hualien, Taiwan, Dec. 2002, pp. 776 ~ 786.