Neighborhood-Search-Based Path Planning Algorithm Applied to a Python Robotic Simulation Environment

Polliana Barelli Trentini
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Santo André, Brazil
p.barelli@aluno.ufabc.edu.br

Mateus Coelho Silva
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Santo André, Brazil
mateuscoelho.ccom@gmail.com

***Abstract*—Path planning has become one of the most important fields for mobile robots. It involves analyzing various parameters, including energy consumption, movement smoothness, travel time, and collision avoidance. However, to test these path planning algorithms, researchers are often limited by the need to use 3D simulators. In this context, our work focuses on developing a path planning algorithm with a neighborhood-search-based approach using a lightweight, user-friendly framework software based on Python. This conventional approach, combined with a simulator, allowed us to test our algorithm on different maps, finding successful paths between our initial and goal points, especially on simpler maps with fewer obstacles. The methodology was evaluated in several experiments using different parameters as intermediate points and noise force for disturbances. Our results indicate the feasibility of the proposed technique, which generated valid paths and optimized them towards better solutions.**

***Index Terms*—Path planning, Neighborhood search, Mobile robots, Simulation**

I. Introduction

Over the last decade, path planning has emerged as one of the most critical fields for mobile robots. In light of ship-drones [1] and underwater vehicles [2], path planning does not limit itself only to ground-based robots, which summarizes its high importance today.

There are two main categories of methodologies in path planning: conventional approaches, such as global and local planning, and learning-based approaches based on Deep Reinforcement Learning (DRL) technology [3]. The path planning success rate is directly related to the robot's energy consumption, movement smoothness, travel time, and collision avoidance [4].

Path planning algorithms are typically categorized into three main groups: classical algorithms, bionic algorithms, and artificial intelligence algorithms. Our focus is on classical algorithms. Classical algorithms are generally used in global path planning, where the environment is previously known by the mobile robot. The primary objective in using this algorithm is not only to identify the path with the highest success rate but also to assess the computational cost and efficiency during processing [5].

In this paper, we focus on a conventional approach using the IR-SIM platform [6], a lightweight Python robot simulator that enables the integration of path planning algorithms. IR-SIM allows the development and testing of robotics algorithms with minimal hardware requirements and is user-friendly, which makes robot navigation intuitive not only for robots but also for users.

II. Related Works

Some related works in the literature have applied similar strategies to ours. Huang et al. [7], for instance, employed a self-adaptive neighborhood search A* algorithm to solve the path planning. Although their approach was efficient, it required a further step in developing an abstract structure of the map. Our approach utilizes iterative testing to validate, resulting in a simpler algorithm that solves the same task.

Li et al. [8] propose a Large Neighborhood Search algorithm for solving multiple-agent path planning. Although they presented a solid return, their algorithm addresses an issue with complexities that are beyond the scope of this work. Therefore, our algorithm offers a more straightforward approach to path planning within the scope of single autonomous robots.

Also, Li et al. [9] discuss the usage of a Variable Neighborhood Search algorithm combined with a bidirectional A* to provide path planning. The algorithm also requires the production of an abstract map structure and employs more complex structures. Our approach offers a more straightforward solution, utilizing a continuous mapping method that eliminates the need to create an explicit occupancy grid.

Literature works often employ more complex algorithmic and data structures than our work does. Nonetheless, our results indicate the feasibility of a simple approach using a simulator and a simpler algorithm.

III. Methodology

Our first step is to set a start point and a goal point. The points are represented by [x, y, θ], in which x and y portray the robot position in the space, and θ, the movement direction of the robot. For simplicity, we chose [0, 0, 0] as the starting

point and [10, 10, 0] as the goal point. For a more coherent path, we will use 'intermediate points' between the starting point and the goal point, with the possibility of increasing or reducing them according to the demand of the environment size.

Instead of starting with initial random points, we use a base trajectory that divides the environment into equal parts. For example, in a 10×10 environment using 3 intermediate points, our initial trajectory will be [[0,0,0], [2.5, 2,5, 0], [5, 5, 0], [7.5, 7.5, 0]]. Based on the base path, we create random trajectories with noise that will be evaluated. To apply this noise, we used discrete uniform distribution through Python's library NumPy, regulating the disturbance by the force of the noise. Then, for each iteration, the trajectory that exhibits the better outcome in terms of distance to the final goal and total number of steps will be the baseline for the next iteration. In this way, we create paths that not only avoid path regression but also obstacles.

Algorithm 1 Iterative Neighborhood Search Trajectory Optimization with Random Disturbances

Require: Number of iterations N_{iter}, number of disturbances N_{dist}, number of points P
Ensure: Optimized trajectory τ^*

```
Initialize results ← ∅
for s = 0 to N_iter − 1 do
    if s = 0 then
        Generate initial trajectory τ
    end if
    obj ← ∅, τ_dist ← ∅
    for t = 1 to N_dist do
        τ' ← DisturbTrajectory(τ)
        (d, n, d_f) ← Simulate(τ')
        J ← d_f + 0.01 · n
        Append J to obj
        Append τ' to τ_dist
    end for
    i* ← arg min(obj)
    τ ← τ_dist[i*]
    Simulate τ to get (d, n, d_f)
end for
return τ
```

The algorithm 1 depicts the pseudo-code of the method where N_{iter} is the number of iterations, N_{dist} is the number of disturbances that the trajectory will receive to create new paths, and P are the intermediate points between the starting point and the destination point. As shown, our output is the optimized trajectory represented by τ^*, which will be the path with the minimum distance and steps.

First, we initialize the results as an empty variable. Then, for each s, which represents the IR-SIM simulations, from 0 to the number of iterations minus one, if it is the first simulation, we generate an initial trajectory represented by τ using our base path. Now, we can initialize the variable obj, which will store the cost function $J = d_f + 0.01 \cdot n$, where d_f is the distance to the final objective and n is the total number of steps, and the variable τ_{dist} that will store our disturbed trajectories.

Subsequently, with our initial trajectory, we can use the function $DisturbTrajectory(\tau)$ to create our disturbed path, represented by τ'. The simulation with the τ' path will return the total distance traveled, d, the number of steps, n, and the final distance to the goal point, d_f. Lastly, we will take the minimum value of J stored in obj with $i^* \leftarrow \arg\min(\text{obj})$, using this trajectory to create more disturbances until we satisfy N_{dist} and N_{iter}.

IV. Results and Discussions

For the environments, IR-SIM uses a 2D grid-based environment. It is a grayscale map in which black pixels are detected as obstacles. Our tests will be divided into two separate maps. Since one is more complex than the other, they will serve as a comparison method.

A. First Map

In Figure 1, we have a simple grid map with obstacles that were previously configured in the YAML configuration obstacle. For our first test, we ran 50 simulations three times with $N_{iter} = 10$, $N_{dist} = 5$ and $P = 3$. The force of the noise is set to 1. The cycle with the most successful path samples will be considered the best.

Fig. 1. IR-SIM grid map environment with configured obstacles

After all cycles, we can notice a few patterns. In the first ten simulations, the robot crashes directly into the obstacle, and in the last ten simulations, the robot rarely crashes. Additionally, due to the grid map structure with an obstacle in the center, the robot tends to move towards a specific region underneath or upward.

The third cycle was the best, with 21 collisions and 29 successful paths. After going through the algorithm, it returned Figure 2 as the best path, which has the following coordinate points: [0,0,0], [1.5,1,0], [7,4,0], [9.5,5.5,0], [10,10,0]. Visually, we can infer that the path is not perfect, but it is also interesting to consider a safety margin for the obstacle, which improves our view of this path.

For our second test, we changed only the force of the noise, now set to 2. After three cycles, the second cycle yielded the best results, with 24 collisions and 26 successful paths. Figure 3 was the best path, with coordinates points [0,0,0],

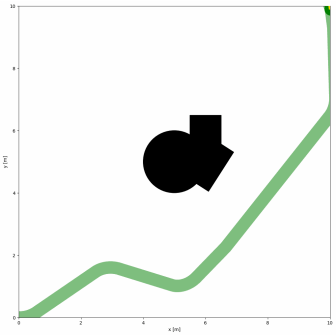

Fig. 2. Path [0,0,0], [1.5,1,0], [7,4,0], [9.5,5.5,0], [10,10,0]

[1,0,0], [5,2,0], [10,10,0], [10,10,0]. The robot's path crosses the obstacle too closely, which is not ideal because it lacks a safety margin. However, it does produce a fast trajectory.

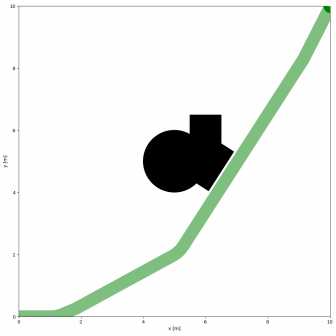

Fig. 3. Path [0,0,0], [1,0,0], [5,2,0], [10,10,0], [10,10,0]

Regarding the force set to 2, we can attest that it is significantly more variable than force 1. It does not exhibit a clear pattern of collisions, and, contrary to our expectations, the difference between the two forces when discussing the robot's exploration is not significant. After both tests, we decided to run an additional 50 simulations for three cycles with $N_{iter} = 10$, $N_{dist} = 5$, and $P = 2$ for both force values used previously.

For the force of the noise set as 1, our best cycle was the first, with 24 collisions and 26 successful paths. It returned Figure 4 as the best path, with coordinates at points [0,0,0], [2.3,0,0], [7.6,4.6,0], and [10,10,0].

For the force of the noise set to 2, all three cycles yielded high results, but the first one stood out, with 14 collisions and 36 successful paths. The best of all the cycles we have had so far. It returned us Figure 5 as the best path, with coordinate points [0,0,0], [5.3,3.3,0], [7.6,3.6,0], [10,10,0].

B. Second Map

In Figure 6, we have one of the IR-SIM preset grid maps, a cave environment. It is a grayscale PNG image. Since it has more obstacles spread across the map, we consider this one significantly more complex than the other, and our objective is to find a successful path under the 50 simulations. For our first test with this map, we performed the same 50 simulations

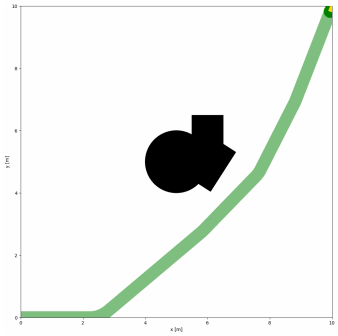

Fig. 4. Path [0,0,0], [2.3,0,0], [7.6,4.6,0], [10,10,0]

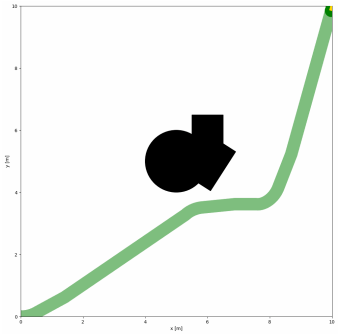

Fig. 5. Path [0,0,0], [5.3,3.3,0], [7.6,3.6,0], [10,10,0]

three times, with $N_{iter} = 10$, $N_{dist} = 5$, and $P = 3$, and the noise force set to 1. Again, we will consider the best cycle, the one with the most successful path samples.

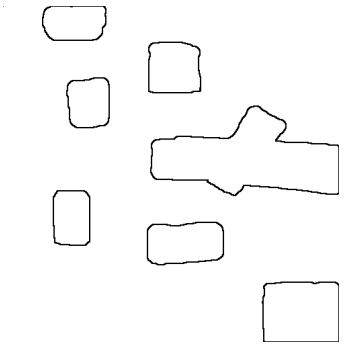

Fig. 6. IR-SIM cave grid map environment

After all cycles, we can only consider one successful result. The first two were complete failures, and our last one only had one successful path. Throughout the simulation, we could see promising paths, but usually the robot crashed after an open curve or the last point was generated directly at an obstacle. Our algorithm returned us Figure 7 as the best path, with coordinates points [0, 0, 0], [3.5, 0.5, 0], [3.0, 5.0, 0], [6.5, 7.5, 0], [10, 10, 0].

For our next test, we set the noise to 2. After three cycles, our robot was unable to find a single successful path, indicating that force 2 is not suitable for this map. While the robot attempted to find a path to the goal throughout all the

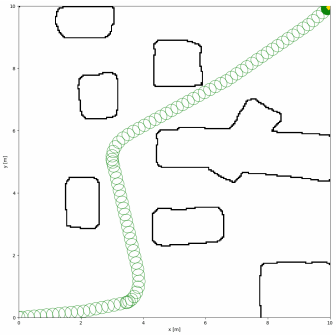

Fig. 7. Path [0, 0, 0], [3.5, 0.5, 0], [3.0, 5.0, 0], [6.5, 7.5, 0], [10, 10, 0]

simulations, we observed that after a few disturbances, the robot became stuck at the bottom of the map, crashing into all the obstacles. As attested by the first map tests, Force 2 is inconsistent and does not help us explore more of the map as expected, failing to meet our needs, especially on maps with greater complexity.

Continuing our tests with $N_{iter} = 10$, $N_{dist} = 5$, and $P = 2$ for both force values used previously, we start with the force set as 1. Our first cycle had two successful paths, whereas the second and third cycles experienced collisions in all simulations. For the best path, our algorithm returned Figure 8 with the following coordinate points: [0,0,0], [2.3333333333333335, 0.3333333333333335, 0], [3.666666666666667, 8.666666666666668, 0], and [10,10,0]. It is a direct and relatively safe path, contradicting our foreknowledge that a map with more obstacles would require more intermediate points to create a path.

To finish our tests, we set the noise to 2. Just like with 3 points, our robot exhibited similar behavior and was unable to find a path during the three cycles. After analyzing all our results, we can infer some things about the behavior of our algorithm. We can attest that, using our second map as an example, our robot was able to find a path only with a specific level of noise. It was clear that when we set our noise level to 2, our robot did not explore the map as expected, but instead tended to make erratic movements.

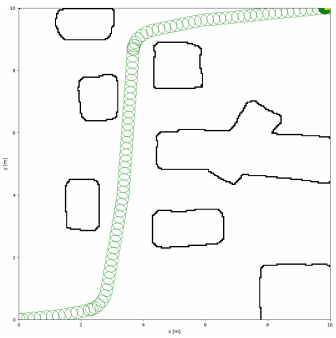

Fig. 8. Path [0,0,0], [2.3333333333333335, 0.3333333333333335, 0], [3.666666666666667, 8.666666666666668, 0], [10,10,0]

This analysis suggests that our base trajectory also needs to be adapted to our map, and not only regarding our initial and goal points. While Figure 3 shows us that a diagonal base trajectory was a good idea since it motivates our robot to explore more of the obstacle sides, Figure 8 makes it clear that this specific base trajectory was getting in the way of the robot to find a successful path. However, since our algorithm was able to find more than one effective path for both maps, achieving its initial objective, we can assert that our path planning algorithm had a satisfactory performance.

V. Conclusion

This research emphasizes the effectiveness not only of a neighborhood-search-based algorithm for path planning, but also of our simulator of choice, IR-SIM. By combining our algorithm with IR-SIM, we were able to observe our robot's behavior in a robotic simulation environment.

Our methodology involved analyzing common path planning challenges to create Python functions in IR-SIM. Since our algorithm is based on a neighborhood-search approach, we prevented possible problems in simple random sampling by aggregating a base trajectory compatible with our intermediate points. Our algorithm proved to be effective, especially on maps with less complexity, demonstrating that using a neighborhood-search base can be beneficial if we limit our parameters to reduce the likelihood of generating anomalies.

Acknowledgment

The authors would like to acknowledge CAPES, CNPq, the Federal University of ABC for supporting this research. This work was developed at the laboratory of artificial intelligence, robotics and algorithms (IARA++), at the Federal University of ABC, Santo André, Brazil.

References

[1] X. Li and H. Zhang, "Cooperative path planning optimization for ship-drone delivery in maritime supply operations," *Complex Intelligent Systems*, vol. 11, pp. 1–24, 05 2025.

[2] H. B. Amundsen, M. Føre, S. J. Ohrem, B. O. A. Haugaløkken, and E. Kelasidi, "Three-dimensional obstacle avoidance and path planning for unmanned underwater vehicles using elastic bands," *IEEE Transactions on Field Robotics*, vol. 1, pp. 70–92, 2024.

[3] Y. Zhang, W. Zhao, J. Wang, and Y. Yuan, "Recent progress, challenges and future prospects of applied deep reinforcement learning : A practical perspective in path planning," *Neurocomputing*, vol. 608, p. 128423, 2024.

[4] M. Gemeinder and M. Gerke, "Ga-based path planning for mobile robot systems employing an active search algorithm," *Applied Soft Computing*, vol. 3, no. 2, pp. 149–158, 2003.

[5] L. Liu, X. Wang, X. Yang, H. Liu, J. Li, and P. Wang, "Path planning techniques for mobile robots: Review and prospect," *Expert Systems with Applications*, vol. 227, p. 120254, 2023.

[6] e. a. Han, Ruihua, "IR-SIM: An open-source lightweight simulator for robot navigation, control, and learning," 2024.

[7] J. Huang, C. Chen, J. Shen, G. Liu, and F. Xu, "A self-adaptive neighborhood search a-star algorithm for mobile robots global path planning," *Computers and Electrical Engineering*, vol. 123, p. 110018, 2025.

[8] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Mapf-lns2: Fast repairing for multi-agent path finding via large neighborhood search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 10256–10265, 2022.

[9] P. Li, Y. Li, and X. Dai, "Vns-ba*: An improved bidirectional a* path planning algorithm based on variable neighborhood search," *Sensors*, vol. 24, no. 21, p. 6929, 2024.