Real-Time Pathfinding in Dynamic Open World Conditions via L-BFGS Optimization

Hansu Kim

Department of Energy Engineering, Korea Institute of Energy Technology (KENTECH), Naju, 58217, Republic of Korea

Abstract

This investigation focuses on the real-time pathfinding in dynamic open worlds based on the L-BFGS optimization with Ajimo line search. The loss function consists of three terms: length loss, smoothness loss, and obstacle loss. The length and smoothness loss adjust the path to minimize the length and prevent sudden changes. Based on the soft constraint, and linearly increasing its weight, the path is not to overlap with obstacles. The terminated condition was set as the standard deviation of the last 10 iterations' loss is lower than a certain threshold. In the update protocol for each timestep, the optimized path is used for the initial path of the next step, which can reduce the computation cost since the obstacle loss can be dismissed in the optimized path at the prior timestep. As a result, the optimal paths were found in the various maps. Since the proposed method considers the overall space as a continuous field, the computation cost is independent of the grid size, compared with the A* algorithm. However, some limitations about the potential formation of complex obstacles and the path reconfiguration method should be more considered for detecting the most optimal path. Since the pathfinding can be applied to various fields like Mobile robots, autonomous drones, and self-driving vehicles. further research should proceed for improved pathfinding and its applications.

1. Introduction

Patfinding is a critical problem in various fields like Mobile robots, autonomous drones, and self-driving vehicles (1). Efficient and safe, meaning to find the smooth, short, and collision-free path, within limited time and resources, can directly affect the performance and reliability of mobile agents, especially in dynamic conditions; the conditions that the agent or target incessantly move. Traditional graph-based algorithms like A* usually detect the path based on the mesh or voxel grid. However, the computational cost increases with the resolution of the grid map, implying the need for optimization-based methods capable of efficiently exploring continuous spaces (2).

In this investigation, the L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) method combined with the Ajimo line search was used for path optimization. L-BFGS is a quasi-Newton optimization algorithm that efficiently updates Hessian and gradient information using limited memory, which is appropriate for large-scale and real-time problems. Based on the direction from L-BFGS, the Ajimo line search determines appropriate step sizes along the descent direction, improving convergence speed and ensuring stability in the path optimization process (3). In this optimization, the termination condition depends on the standard deviation of the last few loss trajectories. In addition, to connect the frames, I proposed that the optimized path resulting from the prior timestep is the initial input for the next timestep. Based on the proposed optimization, the pathfinding was tested for various maps and analyzed based on the update protocol.

2. Methodology

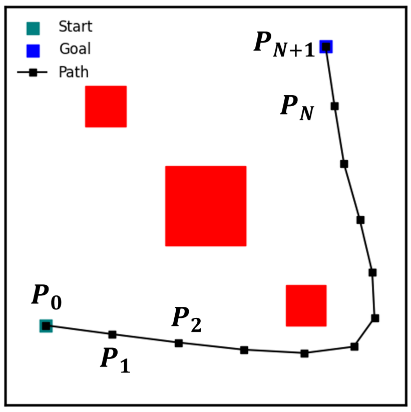


Figure 1: Schematic of the path composed of N intermediate points.

In this research, the full path connecting from the start to the target point was represented as a sequence of N intermediate points, $\{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, ..., \mathbf{P}_N, \mathbf{P}_{N+1}\}$, where $\mathbf{P}_0$ and $\mathbf{P}_{N+1}$ correspond to the start and goal points, respectively, as shown in Figure 1.

2.1. Objective function and constraints

$$\begin{aligned} \min_{\mathbf{P}_1,\dots,\mathbf{P}_N} \quad & \lambda_{\text{length}} L_{\text{length}} + \lambda_{\text{smooth}} L_{\text{smooth}} + \lambda_{\text{obs}} L_{\text{obs}} \\ \text{s.t.} \quad & \mathbf{P}_0 = \text{start}, \quad \mathbf{P}_{N+1} = \text{goal} \end{aligned} \tag{1}$$

The pathfinding optimization problem in open-world conditions can be constructed as shown in Equation 1. The optimization focused on minimizing the loss of path length, path smoothness, and loss from overlapping with obstacles. The constraint was $\mathbf{P}_0$ and $\mathbf{P}_{N+1}$ correspond to the start and goal point, respectively. the position in 2-dimension space, $\mathbf{P}_i = [x_i, y_i]^T \in \mathbb{R}^2, \quad i = 1, \ldots, N$ denotes the i-th control point of the path, and N is the number of inner control points. The complete path is defined as

$$\mathbf{P} = \{\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_N, \mathbf{P}_{N+1}\}.$$

The loss terms consist of L_{length} to contract the overall path, L_{smooth} to promotes smooth transitions between consecutive control points and L_{obs} to prevent the path from overlapping with the obstacle as soft constraint, connected with their weight, λ_{length},λ_{smooth} and λ_{obs}.

2.1.1. Length and Smoothness Loss

The first two components of the total loss function focus on the geometric properties of the path, defined by its total length and smoothness. The path length loss L_{length} operates on the path to make it as short as possible. By minimizing the length-loss term, more direct paths between the start and goal points can be favored, and the distance between the midpoints becomes uniform, as shown in Equation 2.

$$L_{\text{length}} = \sum_{i=0}^{N} \|\mathbf{P}_{i+1} - \mathbf{P}_i\|^2, \tag{2}$$

The path smoothness loss L_{smooth} functions to prevent a sudden change in course as shown in Equation 3. It increases when three neighboring points are not on a straight line to reflect more realistic conditions, and the distance between the midpoints becomes uniform (4). In practice, this helps to avoid sharp turns or oscillations, which are undesirable in real-world pathfinding applications.

$$L_{\text{smooth}} = \sum_{i=1}^{N} \|\mathbf{P}_{i+1} - 2\mathbf{P}_i + \mathbf{P}_{i-1}\|^2. \tag{3}$$

The impact of two loss terms is controlled by their weight, λ_{length} and λ_{smooth}. the values of λ_{length} and λ_{smooth} were set as constant, 1.

2.1.2. Obstacle Loss

The obstacle loss, L_{obs}, operates to penalize overlapped points or lines, preventing the collision between obstacles. Rather than considering the obstacles as a rigid body, they are calculated as soft constraints of a single-maximum potential, allowing the path to adjust smoothly around obstacles.

In this investigation, only rectangular obstacles were considered. The regions of obstacles were defined by their area range, including x_{min}, x_{max}, y_{min}, y_{max} to manifest the rectangular region. In addition, the position of the peak point in potential, C_x, C_y, was applied to improve the obstacle's expressiveness. By applying a margin, m, the effective obstacle region can be defined as shown in Equation 4. The inner points, (x, y), in the effective obstacle region defined by the obstacle and margin can be normalized in the range from -1 to 1 as shown in Equation 5. The potential for each position, $V(x, y)$, is defined by exponential terms and a normalized term in Equation 6, forming the single-maximum potential to prevent the path from falling into a local minimum as shown in Equation 7. In the boundary of potential, by setting the zero value and zero gradient as shown in Figure 2. Zero potential and zero gradient in the obstacle boundary operate to smoothly decrease towards the edges of the obstacle. Furthermore, various modifications can be possible by adjusting the center of the peak point, C_x and C_y ever to the condition with two obstacles connected in Figure 3.

$$\begin{aligned} x_{\text{min}}^{\text{eff}} &= x_{\text{min}} - m, \\ x_{\text{max}}^{\text{eff}} &= x_{\text{max}} + m, \\ y_{\text{min}}^{\text{eff}} &= y_{\text{min}} - m, \\ y_{\text{max}}^{\text{eff}} &= y_{\text{max}} + m. \end{aligned} \tag{4}$$

$$\begin{aligned} n(x) &= \begin{cases} \dfrac{x - C_x}{C_x - x_{\text{min}}^{\text{eff}}}, & x_{\text{min}}^{\text{eff}} \leq x < C_x \text{ (left side)} \\ \dfrac{x - C_x}{x_{\text{max}}^{\text{eff}} - C_x}, & x_{\text{max}}^{\text{eff}} \geq x \geq C_x \text{ (right side)} \end{cases}, \\ n(y) &= \begin{cases} \dfrac{y - C_y}{C_y - y_{\text{min}}^{\text{eff}}}, & y_{\text{min}}^{\text{eff}} \leq y < C_y \text{ (bottom side)} \\ \dfrac{y - C_y}{y_{\text{max}}^{\text{eff}} - C_y}, & y_{\text{max}}^{\text{eff}} \geq y \geq C_y \text{ (top side)} \end{cases}. \end{aligned} \tag{5}$$

$$f(n) = e^{-n^2}(1 - n^2)^2, \quad n \in [-1, 1]. \tag{6}$$

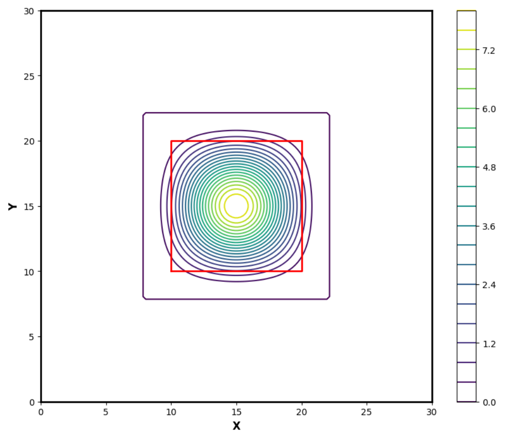
$$V(x, y) = \begin{cases} 8\, f(n(x)) f(n(y)), & x_{\text{min}}^{\text{eff}} \leq x \leq x_{\text{max}}^{\text{eff}},\ y_{\text{min}}^{\text{eff}} \leq y \leq y_{\text{max}}^{\text{eff}} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

The obstacle loss L_{obs} is constructed by integrating the potential function $V(x, y)$ along the path. For each intermediate point pair $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$ in Equation 8, four sample points are generated on the line connected from $\mathbf{P}_i$ to $\mathbf{P}_{i+1}$, and the potential of each point is evaluated and summed to compute the obstacle loss in Equation 9.

$$\mathcal{S} = \{(\mathbf{P}_i, \mathbf{P}_{i+1}) \mid i = 0, \ldots, N\} \tag{8}$$

$$L_{\text{obs}} = \sum_{(\mathbf{P}_i, \mathbf{P}_{i+1}) \in \mathcal{S}} \sum_{j=1}^{4} V(x_{i,j}, y_{i,j}) \tag{9}$$

The impact of the loss due to obstacle overlap is represented by the weight λ_{obs}. Although soft constraints are applied for each obstacle, they define regions that cannot be traversed, making it essential to adjust the weight to find the optimal path while avoiding obstacles. To prevent sudden changes in the path caused by obstacles during the initial stages of optimization, the weight is initialized at 0.1 and increased linearly by 1 at each iteration. In addition, the obstacle weight has an upper limit of 32, guaranteeing stable convergence as shown in Figure 4.


(a) Contour plot of the rectangle potential with margin. The red lines means the boundary of obstacle.

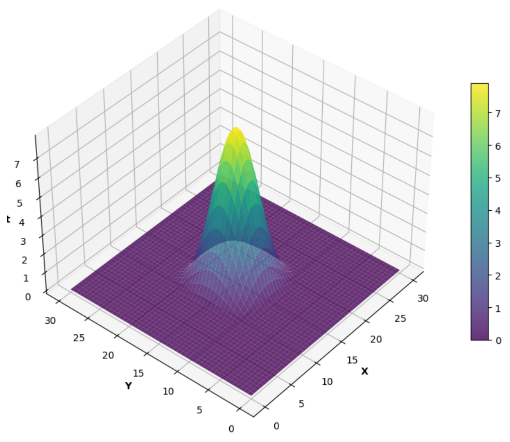
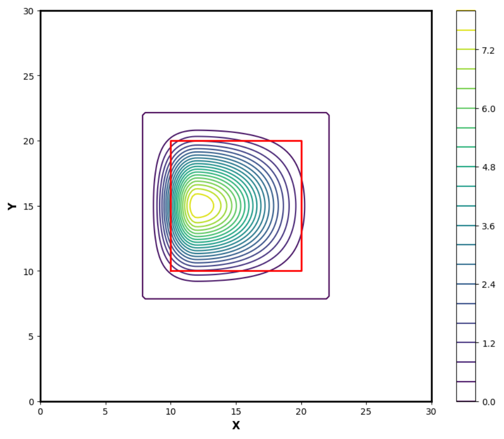

(b) Surface plot of the rectangle potential with margin.

Figure 2: Schematic of 2-dimension and 3-dimension the rectangle potential with margin.


(a) Contour plot of shifted C_x and C_y

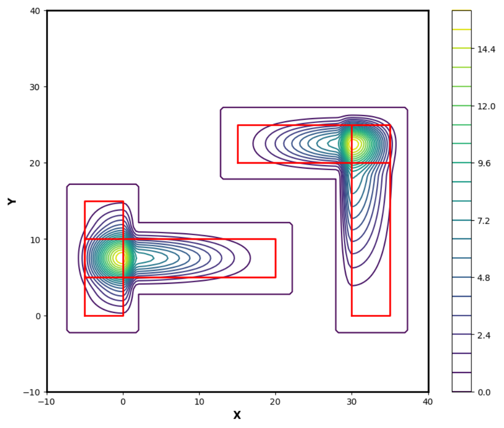

(b) Contour plot of two obstacles connected condition

Figure 3: Contour plot of various modified condition

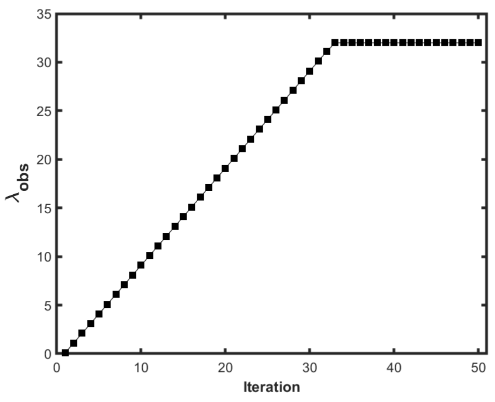


Figure 4: The weight of obstacle, λ_{obs}, during optimization iteration

2.2. L-BFGS optimization

2.2.1. Path initialize

The initial path is constructed by placing N inner control points linearly between the start point $\mathbf{P}_0$ and the goal point $\mathbf{P}_{N+1}$, as shown in Equation 10. A straight-line initialization is chosen for several reasons. It provides a simple and uniform starting distribution of control points, avoiding unnecessary distortions in the initial path. In addition, by starting from the shortest path, the optimized path is more likely to remain short (5). Therefore, this linear initialization serves as a neutral starting point for the optimizer to iteratively refine the path while considering smoothness and obstacle avoidance.

$$\mathbf{P}_i = \mathbf{P}_0 + \frac{i}{N+1}(\mathbf{P}_{N+1} - \mathbf{P}_0), \quad i = 1, \ldots, N. \tag{10}$$

2.2.2. L-BFGS

The path optimization is performed based on the L-BFGS and Armijo line search algorithm. L-BFGS is a quasi-Newton method that approximates the inverse Hessian matrix to efficiently compute search directions to activate the fast convergence by using the second derivative information (6). The Armijo line search determines the step size for a defined direction.

For the optimization process, the initial path $\mathbf{P}$ is reshaped into a vector $\mathbf{x}_0$ of size $2N \times 1$ in Equation 11. Let $s_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ be the displacement vector and $y_k = \nabla L(\mathbf{x}_{k+1}) - \nabla L(\mathbf{x}_k)$ be the change in gradient. At iteration k, the search direction $p_k = -H_k \nabla L(\mathbf{x}_k)$ is computed efficiently using a two-loop recursion algorithm without explicitly forming H_k. Based on the direction from L-BFGS, Armijo line search was operated to calculate the step size 1.

$$\mathbf{x}_0 = [x_1, y_1, x_2, y_2, \ldots, x_N, y_N]^\top \in \mathbb{R}^{2N} \tag{11}$$

2.2.3. Terminated condition

In terminated condition, merely calculating gradient change between two steps may interfere with the optimal path search. In addition, since the λ_{obs} increase until about 30 iteration, comparing gradient change as terminated condition is unstable. So, proposed terminated condition in this investigation is that the

Algorithm 1 Trajectory Optimization via L-BFGS and Amijo Line Search

```
Require: Initial path $\mathbf{x}_0$, Memory size $m$, Max iterations $K$, Decay rate $\tau$, Parameter $c_1$
0: $k \leftarrow 0$
0: $\mathcal{S} \leftarrow \emptyset$,  $\mathcal{Y} \leftarrow \emptyset$,  $\mathcal{R} \leftarrow \emptyset$ {Initialize empty lists}
0: while $k < K$ do
0:   $\mathbf{g}_k \leftarrow \nabla L(\mathbf{x}_k)$
0:   // Phase 1: Compute Descent Direction
0:   $\mathbf{q} \leftarrow \mathbf{g}_k$
0:   if $\mathcal{S}$ is empty then
0:     $\mathbf{p}_k \leftarrow -\mathbf{q}$
0:   else
0:     $M \leftarrow$ number of stored pairs
0:     Initialize vector $\boldsymbol{\alpha}$ of size $M$
0:     // Backward Pass (Recent to Old)
0:     for $i = M$ downto 1 do
0:       $\alpha_i \leftarrow \mathcal{R}_i \cdot (\mathbf{s}_i^\top \mathbf{q})$
0:       $\mathbf{q} \leftarrow \mathbf{q} - \alpha_i \mathbf{y}_i$
0:     end for
0:     // Scaling
0:     $\gamma \leftarrow (\mathbf{s}_M^\top \mathbf{y}_M)/(\mathbf{y}_M^\top \mathbf{y}_M)$
0:     $\mathbf{r} \leftarrow \gamma \mathbf{q}$
0:     // Forward Pass (Old to Recent)
0:     for $i = 1$ to $M$ do
0:       $\beta \leftarrow \mathcal{R}_i \cdot (\mathbf{y}_i^\top \mathbf{r})$
0:       $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{s}_i \cdot (\alpha_i - \beta)$
0:     end for
0:     $\mathbf{p}_k \leftarrow -\mathbf{r}$
0:   end if
0:   // Phase 2: Armijo Line Search
0:   $\eta \leftarrow 1.0$
0:   $L_{curr} \leftarrow L(\mathbf{x}_k)$
0:   while True do
0:     $\mathbf{x}_{new} \leftarrow \mathbf{x}_k + \eta \mathbf{p}_k$
0:     if $L(\mathbf{x}_{new}) \leq L_{curr} + c_1 \eta (\mathbf{g}_k^\top \mathbf{p}_k)$ then
0:       break
0:     end if
0:     $\eta \leftarrow \eta \cdot \tau$
0:   end while
0:   // Phase 3: Update
0:   $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_{new}$
0:   $\mathbf{s}_{new} \leftarrow \mathbf{x}_{k+1} - \mathbf{x}_k$
0:   $\mathbf{y}_{new} \leftarrow \nabla L(\mathbf{x}_{k+1}) - \mathbf{g}_k$
0:   if $\mathbf{s}_{new}^\top \mathbf{y}_{new} > 10^{-10}$ then
0:     Update $\mathcal{S}, \mathcal{Y}, \mathcal{R}$ with $(\mathbf{s}_{new}, \mathbf{y}_{new}, \rho_{new})$
0:     Discard oldest pair if size $> m$
0:   end if
0:   $k \leftarrow k + 1$
0: end while=0
```

iteration is more than 20, the standard deviation of total loss is lower than 0.1. Additionally, an upper limit of 200 iterations is imposed to avoid excessively long optimization cycles in the algorithm 2.

Algorithm 2 Termination Condition for Path Optimization

```
1:  Set minimum iteration threshold $K_{\text{min}} = 20$.
2:  Set maximum iteration limit $K_{\text{max}} = 200$.
3:  Set loss standard deviation threshold $\sigma_{\text{th}} = 0.1$.
4:  for iteration $k = 1$ to $K_{\text{max}}$ do
5:      Update path and compute total loss $L_k$.
6:      Compute the standard deviation of recent losses:
            $\sigma_L = \text{Std}(L_{k-W+1}, \ldots, L_k)$
7:      if $k > K_{\text{min}}$ and $\sigma_L < \sigma_{\text{th}}$ then
8:          Terminate optimization
9:          break
10:     end if
11: end for=0
```

2.3. Update protocol

In the dynamic condition, where that target point changes in real time, restarting and detecting the L-BFGS optimization from a straight path initialization at every time step may lead to unnecessary computational cost. So, in this investigation, the optimized path at the prior timestep is used for the initial path for the next optimization process. Since the optimized path at the prior timestep has already adapted to the obstacle geometry in earlier iterations, most control points are already positioned away from obstacle boundaries, accelerating the optimization process. In other words, since the obstacle loss of the initial path referred to by the already optimized path may be dismissed, the remaining optimization consists mainly of small deformations rather than a complete reconstruction of a collision-free trajectory. Therefore, fewer iterations may be required, improving the optimization process.

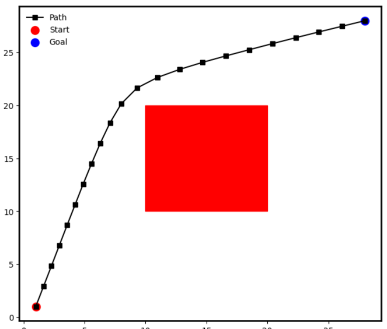
3. Result and Discussion

In the test condition, the number of intermediate point, N, was set as 20, the margin of obstacles, m, was set as 2, the memory size of L-BFGS process was set as 8, the number of sample points at each point pair was set as 4 in Equation 9, the Amijo parameter, $c1$ was set as 0.0001, the weight of length and smoothness loss, λ_{length} and λ_{smooth}, were set as 1, the weight of obstacle loss, λ_{obs}, started from an initial value of 0.1 and was increased by 1 at each iteration, up to a maximum value of 32, as illustrated in Figure 4. The full code and details can be found at `https://github.com/CheongJea/Pathfinding_Optimization_LBFGS/tree/main`.

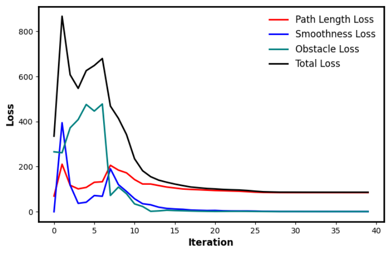
3.1. Performance for various map

3.1.1. Toy map

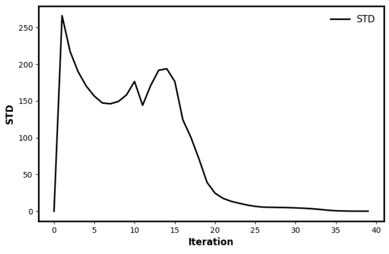
The optimized path and history for the toy map, consisting of a single rectangular obstacle, are shown in Figure 5. During the



(a) Optimized path in toy map



(b) History of optimization process



(c) Standard deviation of last 10 total loss by iterations

Figure 5: Optimization result for toy map

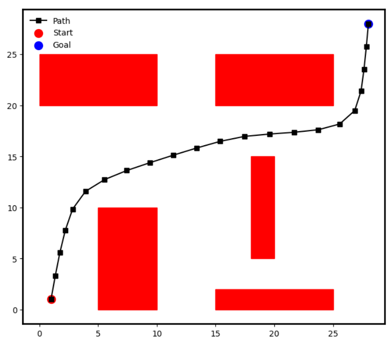
initial optimization process, up to 20 iterations, the overall loss fluctuated due to the increase in the obstacle weight, λ_{obs}. After adjusting the path not to overlap with the obstacle, the path is optimized for satisfying the minimum length and appropriate smoothness. Finally, the path converged to a stable configuration, resulting in the low standard deviation of the last 10 total losses.

3.1.2. Maze and curved map

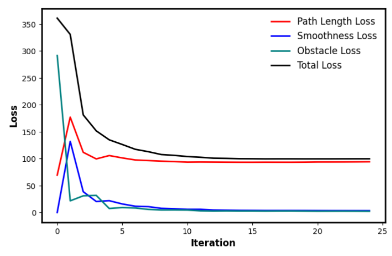
The optimized path and history for the maze and curved map are shown in Figure 6 and Figure 7. The result can be interpreted as the same principles as a toy map. During the initial optimization process, with increased weight of obstacle, λ_{obs}, the path tends to be optimized to avoid the obstacles, increasing the path length and smoothness loss. After adjusting the path not to overlap with obstacles, the path converged to a stable configuration, resulting in a low standard deviation history.

3.2. Update analysis

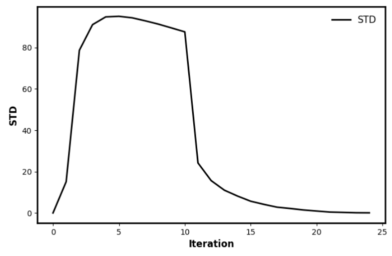
For the curved map, update protocol was additionally applied to adjust the path based on the pre-optimized path when the goal moves to the bottom regions, referred from the Figure 7. In the path update process based on the optimized path at prior



(a) Optimized path in maze map

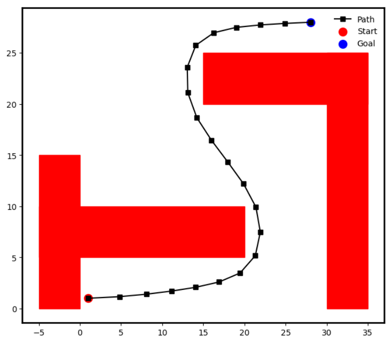


(b) History of optimization process

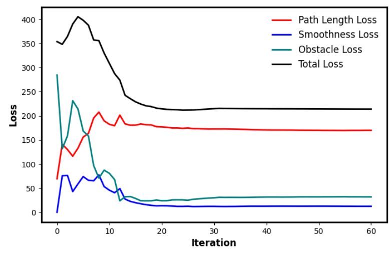


(c) Standard deviation of last 10 total loss by iterations

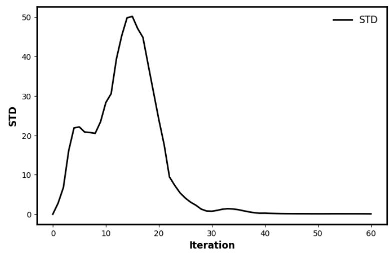
Figure 6: Optimization result for maze map



(a) Optimized path in curved map



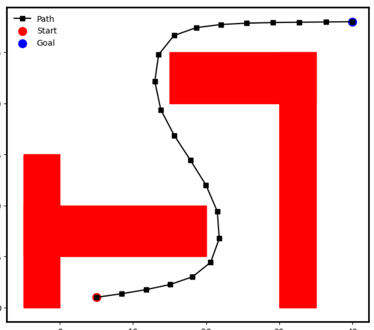
(b) History of optimization process



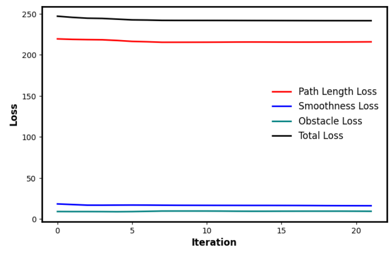
(c) Standard deviation of last 10 total loss by iterations

Figure 7: Optimization result for curved map

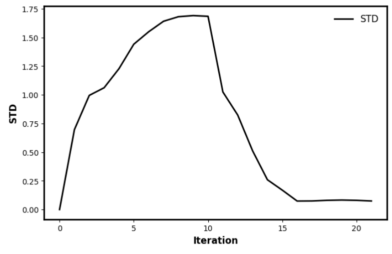
timestep, overall optimization terminated under the 30 iteration, since the optimized path at the prior timestep has already adapted to the obstacle geometry in earlier iterations as shown in Figure 8. and Figure 9. Therefore, the computation time of optimization based on the optimized path is lower than the optimization based on the starting from the straight line between start and goal point as shown in Figure 10.



(a) Optimized path in curved map

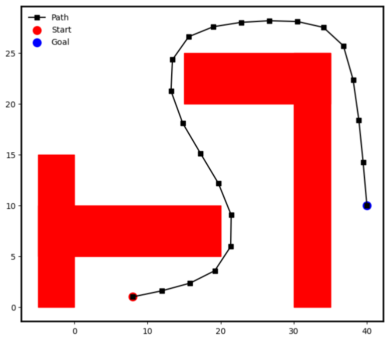


(b) History of optimization process

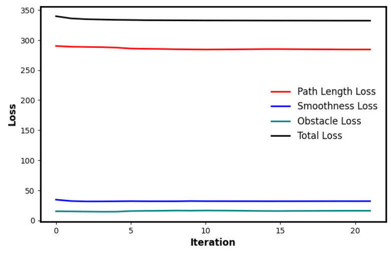


(c) Standard deviation of last 10 total loss by iterations

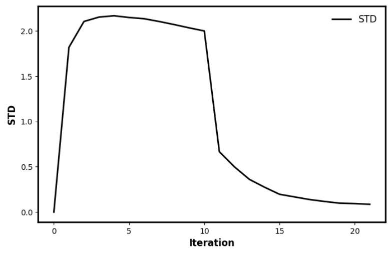
Figure 8: Optimization result of next timestep from Figure 7



(a) Optimized path in curved map



(b) History of optimization process



(c) Standard deviation of last 10 total loss by iterations

Figure 9: Optimization result of next timestep from Figure 8

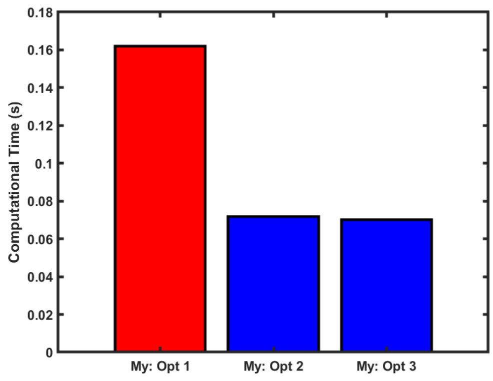
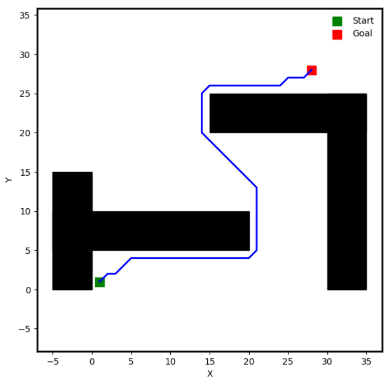


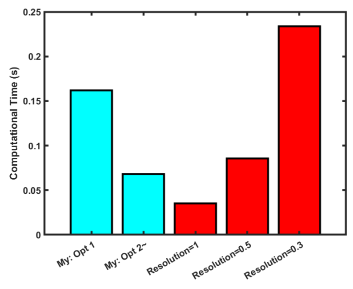
Figure 10: The computation cost comparison between optimization from straight line and optimized path.

3.3. *Comparison with A**

As a famous pathfinding algorithm, A* algorithm detects the optimal path on the grid map. In the pathfinding on the discretized grid map, the accuracy and precision of pathfinding depend on the resolution of the grid, which overall increases the computational cost with grid resolution as shown in Figure 11 (7; 8). However, in the proposed method in this investigation, the space is considered as continuous, and the proposed method may have an advantage in terms of memory efficiency, compared with the high-resolution A* pathfinding algorithm. In addition, although the A* algorithm detects the minimum length path, the proposed method can detect a more realistic and feasible path by applying a smoothness loss term.



(a) Optimized path of A* pathfinding with the resolution 1.



(b) Comparison of computational cost between A* and proposed method

Figure 11: Comparison between A* pathfinding and proposed method

3.4. *Limitation*

3.4.1. *Complex obstacle potential formation*

In this investigation, a soft constraint of obstacles was applied to optimize the path so as not to overlap and collide with the obstacles. To optimize the path to be smooth and stable, the boundary should have a single-peak distribution and satisfy the boundary condition (zero value and zero gradient at the boundary). However, in the complex obstacle environment, constructing a potential satisfying the boundary condition is hard to define. Complex obstacle geometries often include irregular gradients, discontinuities, or multiple local minima in the potential field. These characteristics can distort the optimization landscape, making it difficult for optimization, resulting in unstable pathfinding.

3.4.2. *Path reconfiguration strategy*

Although the proposed path update method based on the optimized path can improve the computational cost, the path determined by the optimized path at the prior timestep can not guarantee that it is the minimum route. As shown in Figure 9, although the more optimal path connecting between the start and goal point (bottom line), the path was optimized as a longer line. Therefore, relying solely on the previously optimized path may lead the algorithm to overlook a more globally optimal route. This limitation highlights the necessity of an additional mechanism to escape suboptimal regions and a method for the optimizer to transfer to the true minimum-length path.

3.5. *Applications*

The concept of optimal pathfinding principles can be broadly applied across various fields that require fast, smooth, and adaptive path planning in dynamic environments. For example, from the robotics and autonomous navigation perspectives, Mobile robots, autonomous drones, and self-driving vehicles should plan the optimal path, which consists of smooth paths while reacting to moving obstacles or changing maps (9). In addition, from the Game Development and Real-Time Simulation perspectives, in open-world games or large-scale simulations, non-player characters (NPCs) often require natural and continuous movement (10). So, the proposed method can provide smoother and more physically feasible motion compared to traditional pathfinding, improving realistic properties. Finally, from the industrial automation and smart manufacturing perspective, they need the optimal path of robotic arms and automated guided vehicles (11). So, this method can provide insight for a smooth and optimal path for them.

4. Conclusion

In this investigation, I focused on the optimal pathfinding based on the L-BFGS optimization in the open world dynamic conditions. Based on the soft constraint and linearly increased weight of obstacles, the optimal path that does not overlap the obstacles could be detected. The terminated condition based on the standard deviation of loss history can prevent the invalid shutdown, contributing to stable optimization. The proposed method functions well in the basic, maze, and curved maps, constructing a more realistic path compared with the A* algorithm since the proposed method considers the spaces as a continuous field. In addition, by referring to the optimized path at the prior timestep, the computational cost of the optimization process may be improved. Furthermore, the consideration of constructing the complex obstacle potential and path reconnection strategy is needed for detecting the most optimal path stably. The findings of this investigation can be applied across various fields like robotics and autonomous navigation, real-time simulation, industrial automation, and so on. Therefore, further research should be focused on the precise and accurate pathfinding and its applications.

Data availability statement

The full code and details can be found at `https://github.com/CheongJea/Pathfinding_Optimization_LBFGS/tree/main`.

References

[1] A. Botea, B. Bouzy, M. Buro, C. Bauckhage, D. Nau, Pathfinding in games (2013).

[2] W. Sun, P. Sun, W. Ding, J. Zhao, Y. Li, Gradient-based autonomous obstacle avoidance trajectory planning for b-spline uavs, Scientific Reports 14 (1) (2024) 14458.

[3] P. Kenneweg, T. Kenneweg, B. Hammer, Improving line search methods for large scale neural network training, in: 2024 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA), IEEE, 2024, pp. 1–6.

[4] M. Hou, Z. Lei, H. Hu, Z. Wang, Y. Wang, L. Xie, H. Wang, Towards efficient path finding and moving stability-focused trajectory planning for mobile robots, in: 2024 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2024, pp. 1111–1116. doi:10.1109/ROBIO64047.2024.10907636.

[5] I. Fdez. Galván, M. J. Field, Improving the efficiency of the NEB reaction path finding algorithm, Journal of Computational Chemistry 29 (1) (2008) 139–143. doi:10.1002/jcc.20780.

[6] M. J. Kochenderfer, T. A. Wheeler, Algorithms for optimization, Mit Press, 2019.

[7] P. Lester, A* Pathfinding for Beginners, http://www.gamedev.net/reference/articles/article2003.asp, accessed: 2009-02-08 (2005).

[8] C. Henkel, M. Toussaint, Optimized directed roadmap graph for multi-agent path finding using stochastic gradient descent, in: Proceedings of the 35th Annual ACM Symposium on Applied Computing, ACM, 2020, pp. 776–783.

[9] D. J. Kim, Optimization-based path planning and control for autonomous driving vehicle, Phd thesis, Hanyang University, Seoul, Korea (2022).

[10] P. Harsadi, S. Asmiatun, A. N. Putri, Dynamic pathfinding for non-player character follower on game, Jurnal Teknik Informatika CIT Medicom 13 (2) (2021) 55–63.

[11] Y. Xu, S. Kong, X. Kong, Z. Ying, Multi-objective trajectory optimization method for industrial robots based on improved td3 algorithm, Scientific Reports 15 (1) (2025) 41970.