

Robot Path Planning with Safety Zones

Evis Plaku¹^a, Arben Çela^{1,2}^b and Erion Plaku³^c

¹*AI Laboratory, Tirana Metropolitan University, Sotir Kolea Street, Tirana, Albania*

²*Laboratory of Images, Signals, and Intelligent Systems, ESIEE Paris, Noisy-le-Grand CEDEX, France*

³*Department of Computer Science, George Mason University, Fairfax, Virginia, U.S.A.*

Keywords: Robot Path Planning, Safety Zones.

Abstract: Path planning is essential for guiding a robot to its destination while avoiding obstacles. In practical scenarios, the robot is often required to remain within predefined safe areas during navigation. This allows the robot to divert from its main path during emergencies and follow alternative routes to a safety center. This paper introduces a novel method to incorporate safety zones into path planning. Each zone is defined by a central point and a radius. Our approach efficiently plans paths to the goal, ensuring that the robot can reach a safety center without having to travel more than the radius of the safety zone. Using sampling, our approach constructs a roadmap with navigation routes and identifies safe locations that satisfy the distance requirements for reaching a safety center. The safe portion of the roadmap is then searched to find a path to the goal. We demonstrate the effectiveness of our approach through simulated experiments in obstacle-rich 2D and 3D environments, utilizing car and blimp robot models.

1 INTRODUCTION


Path planning plays a crucial role in ensuring that a robot can successfully reach its destination while avoiding collisions (Choset et al., 2005). To enhance safety, a practical approach is to require the robot to adhere to designated safe areas so that it can promptly respond to emergencies by diverting from its intended path and following alternative routes to safety centers. This is imperative for reducing potential risks and ensuring the safety of the robot and its surroundings.


Path planning with safety zones holds potential for diverse domains. Robots in industrial settings or search-and-rescue missions could utilize path planning with safety zones to navigate hazardous environments or unstable terrains. Autonomous vehicles could employ safety zones for lane adherence, obstacle response, and handling unpredictable behavior. Healthcare robots could adhere to safety zones to prioritize patient and staff well-being while delivering supplies and performing tasks. Overall, the adoption of path planning with safety zones could enhance reliability, efficiency, and safety across domains.




Figure 1: Example of path planning with safety zones for a car robot model. Each safety zone is defined by its center (blue cylinder) and radius (blue circle). For each intermediate configuration along the main path to the goal, the figure also shows the emergency route (black segments) to reach a safety center within the distance constraints. If an emergency path partially overlaps with the main path, only the non-overlapping portion is visible. The safe locations (from which a safety center can be reached within the distance constraints) are shown in yellow, cyan, magenta, and blue for safety zones 1, 2, 3, and 4, respectively. Videos can be found at <https://tinyurl.com/y4d2urcf>.

Toward this objective, this paper makes it possible to incorporate safety zones into path planning. Each safety zone consists of a center location and a corresponding radius, representing the maximum permissible distance the robot can travel to reach the center. A location is deemed safe if the robot is able

^a <https://orcid.org/0009-0002-4042-2673>

^b <https://orcid.org/0000-0001-5708-1743>

^c <https://orcid.org/0000-0002-6622-386X>

to reach a safety center while avoiding collisions and without traveling more than the radius associated with the safety center. The objective is then to compute a collision-free path that enables the robot to reach its goal while remaining within safe locations. In this way, the robot can always detour to a safety center in response to emergency situations. Fig. 1 shows an example of path planning with safety zones.

Path planning with safety zones presents significant challenges. Even without considering safety zones, planning a collision-free path is challenging, particularly in unstructured and obstacle-rich environments. The robot must navigate around several obstacles and maneuver through narrow passages to reach its goal. When safety zones must be taken into account, the complexity increases further. The path planner must now reason about location safety and generate safety paths from each intermediate location along the main path. Consequently, this gives rise to numerous path-planning problems, demanding novel methods to ensure efficient and safe robot navigation.

To overcome these challenges, our approach combines sampling-based exploration with discrete search, enabling safe navigation in unstructured and obstacle-rich environments. Our approach facilitates navigation by constructing a dense roadmap that connects neighboring locations through collision-free paths. Safe locations are identified by searching over the roadmap from each safety center. Finally, the path to the goal is computed through a shortest-path search restricted to the safe portion of the roadmap. If no such path exists, we add more samples and create new connections to improve the roadmap coverage. This process of enhancing the roadmap, identifying safe locations, and searching the safe portion continues until a solution is found or a runtime limit is reached. We demonstrate the efficacy of our approach by conducting simulated experiments in unstructured 2D and 3D environments with numerous obstacles, employing both car and blimp robot models.

2 RELATED WORK

Sampling-based approaches have proven effective for path planning in complex environments. One popular approach is the Probabilistic RoadMap (PRM) (Kavraki et al., 1996), which constructs a roadmap by sampling and connecting neighboring robot configurations to capture the connectivity of the environment. Various techniques seek to improve sampling near obstacles (Cao et al., 2019), deferring collision checking (Bohlin and Kavraki, 2000), utilizing machine learning (Baldoni et al., 2022), or provide asymptotically

optimal solutions (Karaman and Frazzoli, 2011).

Other approaches utilize Rapidly-exploring Random Tree (RRT) (LaValle and Kuffner, 2001). Enhancements include multi-step connections (Kuffner and LaValle, 2000), regions (Denny et al., 2020), obstacle clearance (Plaku et al., 2017), direct-path superfacets (Plaku et al., 2018), and machine learning (Wang et al., 2020; Zhao et al., 2022; Bui et al., 2022).

Risk-aware methods aim to compute collision-free paths that reduce a risk metric (Feyzabadi and Carpin, 2014; Primatesta et al., 2019). Other methods incorporate uncertainty reasoning to ensure the safe execution of planned paths by the robot (Schouwenars et al., 2004; Pepy and Lambert, 2006).

Our proposed approach leverages from PRM the idea of constructing a roadmap to capture the connectivity of the environment. The roadmap in our case is denser to provide alternative connections between locations. Moreover, we introduce the concept of safety zones and leverage the roadmap to identify safe locations. By searching the safe portion of the roadmap, our approach finds collision-free paths to the goal, ensuring detours to safe centers from any intermediate configuration along the planned path. This adaptive capability is essential to reducing potential risks.

3 PROBLEM FORMULATION

This section defines the environment, safety zones, robot models, and the overall problem.

The environment, denoted as \mathcal{W} , consists of obstacles $O = \{O_1, \dots, O_m\}$ and safety zones $\mathcal{S} = \{S_1, \dots, S_n\}$. Obstacles are often represented as 2D polygons or 3D triangular meshes. Each $S_i \in \mathcal{S}$ is defined by its center and radius, denoted as $S_i.c$ and $S_i.r$.

The robot model is a tuple $\mathcal{R} = \langle \mathcal{P}, \mathcal{C} \rangle$, where \mathcal{P} denotes its geometric shape and \mathcal{C} denotes the configuration space. A configuration $c = \langle p, \theta \rangle \in \mathcal{C}$ defines the position p and orientation θ . The robot models used in the experiments are shown in Fig. 2. For the car model, $c = \langle p = (x, y), \theta \rangle$, where θ denotes the rotation in the xy -plane. The blimp can fly parallel to the xy -plane, so $c = \langle p = (x, y, z), \theta \rangle$.

A local motion from configuration $c_1 \in \mathcal{C}$ to $c_2 \in \mathcal{C}$ is denoted as $c_1 \rightsquigarrow c_2$ and is defined via interpolation, i.e., $\text{interp} : \mathcal{C} \times \mathcal{C} \times [0, 1] \rightarrow \mathcal{C}$, where $\text{interp}(c_1, c_2, t) = (1-t)c_1 + tc_2$.

A path is defined as a sequence of configurations $\sigma : \{1, \dots, \ell\} \rightarrow \mathcal{C}$, where the robot follows the interpolated motion from σ_i to σ_{i+1} . The notation $|\sigma|$ denotes the number of configurations, i.e., $|\sigma| = \ell$. The path length is defined as the distance traveled, i.e., $\text{dist}(\sigma) = \sum_{i=1}^{|\sigma|-1} \text{dist}(\sigma_i, \sigma_{i+1})$.



Figure 2: The robot models used in the experiments.

It is often desirable to ensure that the intermediate configurations in σ are no more than a distance $d \in \mathbb{R}^{>0}$ from each other, i.e., $\text{dist}(\sigma, \sigma_{i+1}) \leq d$.

A configuration $c \in \mathcal{C}$ is collision-free if the robot does not collide with an obstacle when placed according to c . PQP (Larsen et al., 2000) is used to check for collisions. A path is collision-free if every configuration in the sequence and the interpolated motions between intermediate configurations are collision-free.

A configuration $c \in \mathcal{C}$ is safe if there is a collision-free path λ from c to a safety center $S_i.c$ such that $\text{dist}(\lambda) \leq S_i.r$. Note that some locations within the ball defined by $S_i.c$ and $S_i.r$ could be unsafe since, due to obstacles, there may not be a collision-free path to a safety center that satisfies the distance constraints.

Putting it all together, the input to path planning with safety zones is the environment \mathcal{W} , obstacles $O = \{O_1, \dots, O_m\}$, safety zones $\mathcal{S} = \{S_1, \dots, S_n\}$, the robot model $\mathcal{R} = \langle \mathcal{P}, \mathcal{C} \rangle$, a maximum distance d between configurations, and the initial and goal configurations $c_{\text{init}}, c_{\text{goal}} \in \mathcal{C}$. The objective is to compute a collision-free path σ from c_{init} to c_{goal} consisting entirely of safe configurations. In addition, the approach should compute a safety path λ_i for each intermediate configuration $c_i \in \sigma$. Intermediate configurations along σ and each λ_i should be within a maximum distance d from each other. Figs. 1 and 3 provide several illustrations of path planning with safety zones.

4 METHOD

Our approach has three components. The first component constructs a roadmap to facilitate navigation within the safe zones. The second component identifies the safe roadmap nodes, i.e., nodes that can reach a safety center with a path whose length does not exceed the radius of the safety zone. The third component searches over the safe portion of the roadmap to find a path from the initial to the goal configuration. Pseudocode is shown in Alg. 1. Details follow.

4.1 Roadmap Construction Within Safety Zones

The roadmap is represented as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each node $v \in \mathcal{G}.\mathcal{V}$ corresponds to a collision-free configuration, denoted as $v.c$. Each edge $\langle v, u \rangle \in \mathcal{G}.\mathcal{E}$

Algorithm 1: Path planning with safety zones.

Input: \mathcal{W} : environment; O : obstacles;
 $\mathcal{S} = \{S_1, \dots, S_n\}$: safety zones; $\mathcal{R} = \langle \mathcal{P}, \mathcal{C} \rangle$: robot model; d : maximum roadmap edge distance; k : number of roadmap neighbors; nrSamplesToAdd: roadmap batch size; $c_{\text{init}}, c_{\text{goal}} \in \mathcal{C}$: initial and goal configurations; t_{max} : runtime limit.
Output: A collision-free path σ from c_{init} to c_{goal} consisting entirely of safe configurations, and a collision-free safety path λ_i for each $c_i \in \sigma$.

```

1  $\mathcal{G} = (\mathcal{V}, \mathcal{E}) \leftarrow (\emptyset, \emptyset)$ ;
2 for  $c \in \{c_{\text{init}}, c_{\text{goal}}, S_1.c, \dots, S_n.c\}$  do
3    $v.c \leftarrow \langle c, \text{safe}, d_1, \dots, d_n \rangle \leftarrow$ 
    $\text{ADDNODE}(\langle c, \text{false}, \infty, \dots, \infty \rangle)$ ;
4 while  $\text{TIME}() \leq t_{\text{max}}$  do
5   nrNodes  $\leftarrow |\mathcal{G}.\mathcal{V}|$ ;
6   for  $i = 1, \dots, \text{nrSamplesToAdd}$  do
7      $c \leftarrow \text{SAMPLEVALIDCFG}(\mathcal{W}, O, S)$ ;
8      $v.c \leftarrow \langle c, \text{safe}, d_1, \dots, d_n \rangle \leftarrow$ 
      $\text{ADDNODE}(\mathcal{G}, \langle c, \text{false}, \infty, \dots, \infty \rangle)$ ;
9   for  $i = \text{nrNodes}, \dots, |\mathcal{G}.\mathcal{V}| - 1$  do
10     $v \leftarrow \text{GETNODE}(\mathcal{G}, i)$ ;
11    for  $u \in \text{NEIGHBORS}(\mathcal{G}, v.c, k)$  do
12      if  $\neg \text{COLLISION}(\mathcal{W}, O, v.c \rightsquigarrow u.c)$ 
13        then  $\text{ADDEDGE}(\mathcal{G}, \langle v, u \rangle, d)$ ;
14    for  $S_i \in \mathcal{S}$  do
15       $S_i.\text{parents} \leftarrow \text{FINDSAFE}(\mathcal{G}, S_i.c, S_i.r)$ ;
16      for  $\langle v, \text{parent}, \text{cost} \rangle \in S_i.\text{parents}$  do
17         $v.\text{safe} \leftarrow \text{true}$ ;  $v.d_i \leftarrow \text{cost}$ ;
18       $\mathcal{V}_{\text{safe}} \leftarrow \{v : v \in \mathcal{G}.\mathcal{V} \wedge v.\text{safe} = \text{true}\}$ ;
19       $\mathcal{E}_{\text{safe}} \leftarrow \mathcal{G}.\mathcal{E} \cap (\mathcal{V}_{\text{safe}} \times \mathcal{V}_{\text{safe}})$ ;
20       $\mathcal{G}_{\text{safe}} \leftarrow \langle \mathcal{V}_{\text{safe}}, \mathcal{E}_{\text{safe}} \rangle$ ;
21      if  $\text{CONNECTED}(\mathcal{G}_{\text{safe}}, v_{\text{init}}, v_{\text{goal}})$  then
22         $\sigma \leftarrow \text{SHORTESTPATH}(\mathcal{G}_{\text{safe}}, v_{\text{init}}, v_{\text{goal}})$ ;
23        for  $i = 1, \dots, |\sigma|$  do
24           $\lambda_i \leftarrow \text{RETRIEVESAFEPATH}(S, \sigma_i)$ ;
25        return  $\langle \sigma, \lambda_1, \dots, \lambda_{|\sigma|} \rangle$ ;
26 return  $\emptyset$ ;
```

indicates a collision-free path from $v.c$ to $u.c$, which is obtained by interpolation and denoted by $v.c \rightsquigarrow u.c$. Moreover, each node v maintains information about the shortest-path distances to each safety center, represented as $v.d_1, \dots, v.d_n$. The node v is marked as safe (indicated by a Boolean flag $v.\text{safe}$) if one of these shortest paths reaches its safety center within the distance constraints, i.e., $v.d_i \leq S_i.r$. In this way, the roadmap corresponds to a network of collision-free configurations interconnected via collision-free paths, designed to facilitate efficient and safe navigation.

The roadmap construction starts by adding the initial and goal configurations, along with the center of each safety zone, as roadmap nodes (Alg. 1:1–3). The

roadmap is further populated by adding randomly-generated collision-free configurations (Alg. 1:6–8). `SAMPLEVALIDCFG` repeatedly samples a configuration until it is not in collision. The sampling is exclusively conducted within the safety zones, as configurations outside these zones are inherently unsafe.

The next stage seeks to connect each node v to its k -nearest configurations through collision-free paths (Alg. 1:9–12). As mentioned earlier, the path from v to a neighbor u is obtained through interpolation from $v.c$ to $u.c$. If any intermediate configuration is in collision, the edge (v, u) is discarded. To streamline this process, we create a mesh that encompasses all the robot placements along the intermediate configurations. Subsequently, we employ efficient collision checking, e.g., PQP, to assess whether the mesh is in collision. This allows us to efficiently determine the collision-free nature of the entire path, without the need for individual configuration checks.

If the path from $v.c$ to $u.c$ is collision-free, we connect v and u in the roadmap (Alg. 1:12). However, if this edge is long, the function `ADDEGE` divides it into smaller segments, each no longer than a specified distance d . These segments are then added as individual edges to the roadmap. This approach ensures that the roadmap captures the connectivity between nodes accurately, even for longer edges.

4.2 Identifying Safe Roadmap Nodes

After constructing the roadmap, the next task is to determine which roadmap nodes are safe. To qualify as safe, a node v must have a roadmap path λ that reaches the center of a safety zone \mathcal{S}_i such that $\text{dist}(\lambda) \leq \mathcal{S}_i.r$. One possible approach to identify the safe nodes is to perform a shortest-path search from each node in the roadmap. However, this can be computationally expensive, especially when the roadmap is dense.

To address this, we utilize Dijkstra’s shortest-path algorithm (Dijkstra, 1959) and initiate the search from each safety center (Alg. 1:13–16). The search starting from $\mathcal{S}_i.c$ ends when a node is removed from the priority queue with a path length exceeding $\mathcal{S}_i.r$. This is possible because all the remaining nodes in the queue will have a path length greater than $\mathcal{S}_i.r$, and thus, they can be excluded from further consideration. By employing this optimization, we can efficiently identify the safe nodes without the need to run shortest-path searches from every roadmap node.

Upon completion of the search, we are able to identify all the nodes that are accessible from $\mathcal{S}_i.c$ through paths with a length that does not surpass $\mathcal{S}_i.r$. For each of these nodes, we maintain records of both the index of the corresponding safety center and the

path length required to reach that safety center. This information allows us to keep track of the connection between the nodes and their associated safety centers, as well as the corresponding path distances. In addition, each \mathcal{S}_i maintains the parent map generated by Dijkstra’s algorithm. The entries in the parent map are tuples $\langle v, \text{parent}, \text{cost} \rangle$, where v serves as the index key, `parent` denotes the parent node of v , and `cost` represents the path length from $\mathcal{S}_i.c$ to v . This data structure enables efficient retrieval and reconstruction of the paths from the safe nodes to $\mathcal{S}_i.c$.

4.3 Searching over the Safe Portion of the Roadmap

When considering safety zones, the planned path must only traverse safe nodes. To achieve this, we eliminate all nodes (and their edges) that are not marked as safe. This refined subset of the roadmap, consisting exclusively of safe nodes and edges, is referred to as the “safe portion” of the roadmap, and is denoted as $\mathcal{G}_{\text{safe}} = \langle \mathcal{V}_{\text{safe}}, \mathcal{E}_{\text{safe}} \rangle$ (Alg. 1:17–19).

To determine the shortest path, we employ A* (Hart et al., 1968) on $\mathcal{G}_{\text{safe}}$ (Alg. 1:20–21). To save computational time, we run A* only when we are certain that a path exists in $\mathcal{G}_{\text{safe}}$ from c_{init} to c_{goal} . This certainty is achieved by utilizing a disjoint-set data structure (Galler and Fisher, 1964), which helps us keep track of the connected components within $\mathcal{G}_{\text{safe}}$.

Let σ denote the path obtained from A*. In addition to σ , we also require the sub-paths from each configuration in σ to a corresponding safety center (Alg. 1:24–25). To reconstruct these sub-paths, we utilize the parent maps generated by Dijkstra’s algorithm, which are stored by each safety zone \mathcal{S}_i . In cases where σ_i is associated with multiple safety centers, we select the shortest path among them. This ensures that we obtain the most efficient path from each configuration in σ to its corresponding safety center.

If it is determined that there is no path connecting the initial and goal configurations within the $\mathcal{G}_{\text{safe}}$ graph, it could be due to inadequate sampling or insufficient connections within the roadmap. In such cases, we return to the roadmap construction stage and continue populating the roadmap with additional nodes and establishing new connections.

Following the updated roadmap construction, we once again identify the nodes that are considered safe and perform a search exclusively within the safe portion of the roadmap. This iterative process of augmenting the roadmap, determining safe nodes, and conducting searches over the safe portion of the roadmap is repeated until a solution path is discovered or a predefined runtime limit is reached.

5 EXPERIMENTS AND RESULTS

Experiments are conducted in simulation using a car and a blimp robot model operating in unstructured environments with numerous obstacles.

5.1 Tree-Based Path Planner for Baseline Comparison

To the best of our knowledge, there is no existing approach that addresses the specific problem considered in this paper. To establish a baseline, we developed an alternative method comprised of a main and an auxiliary planner, both based on RRT. Before adding a node to the main tree, the auxiliary planner is invoked to compute a safety path from the node to the center of a safety zone, ensuring the path length stays within the safety zone’s radius. The node is added to the tree only if a valid safety path is found.

Pseudocode is shown in Alg. 2. In each iteration, a target configuration is randomly sampled. Small steps are then taken from the nearest tree node v_{near} toward the target along an interpolation path until the target is reached or a collision is detected. If the goal is reached, a solution is found. This process of sampling a target and expanding a branch from the nearest node toward the target is repeated until a solution is found or a runtime limit is reached.

The auxiliary and main planners utilize different strategies for implementing `SELECTTARGET`, `ACCEPTABLE(v)`, and `REACHEDGOAL(v)`. In the auxiliary planner, targets are sampled from the entire configuration space, with occasional guidance toward a random safety center. `ACCEPTABLE(v)` checks if it is still possible to reach any safety center. If the distance from the root of the tree to v and from v to the center of a safety zone \mathcal{S}_i exceeds \mathcal{S}_i ’s radius, v cannot reach the center of \mathcal{S}_i . If none of the centers are reachable, v is deemed unacceptable. `REACHEDGOAL(v)` is satisfied when v reaches any of the safety centers.

In the main planner, targets are generally sampled from the entire configuration space, but occasionally the goal configuration c_{goal} is selected as the target. `REACHEDGOAL(v)` is considered satisfied when v reaches c_{goal} . `ACCEPTABLE(v)` utilizes the auxiliary planner to check if there exists a path from v to a safety center that satisfies the distance constraints.

5.2 Environments

The experiments are conducted using four scene types: maze, random, waves, and 3D rooms, as shown in Figs. 1 and 3. These scenes pose significant challenges, as the robot must avoid numerous obstacles

Algorithm 2: Tree-based path planning with safety zones for baseline comparisons.

Input: \mathcal{W} : environment; \mathcal{O} : obstacles;
 $\mathcal{S} = \{\mathcal{S}_1, \dots, \mathcal{S}_n\}$: safety zones; $\mathcal{R} = \langle \mathcal{P}, \mathcal{C} \rangle$:
robot; b : sampling bias; t_{max} : runtime limit.
Output: A collision-free path σ from c_{init} to c_{goal}
consisting entirely of safe configurations, and a
collision-free safety path λ_i for each $c_i \in \mathcal{S}$.

```

1  $\mathcal{T} \leftarrow \text{INITIALIZETREE}(c_{\text{root}})$ ;
2 while  $\text{TIME}() < t_{\text{max}}$  do
3    $c_{\text{target}} \leftarrow \text{SAMPLETARGET}()$ ;
4    $v \leftarrow v_{\text{near}} \leftarrow \text{NEAREST}(\mathcal{T}, c)$ ;
5    $\ell \leftarrow \text{NRSTEPSPATH}(v.c, c_{\text{target}})$ ;
6   for  $i = 1, \dots, \ell$  do
7      $v_{\text{new}} \leftarrow \text{NEWNODE}()$ ;
8      $v_{\text{new}}.c \leftarrow \text{interp}(v_{\text{near}}.c, c_{\text{target}}, i/\ell)$ ;
9      $v_{\text{new}}.parent \leftarrow v$ ;
10     $v_{\text{new}}.d \leftarrow v.d + \text{dist}(v.c, c)$ ;
11    if  $\neg \text{COLLISION}(v)$  then break;
12    if  $\neg \text{ACCEPTABLE}(v)$  then break;
13     $\text{ADDNODE}(\mathcal{T}, v_{\text{new}})$ ;
14    if  $\text{REACHEDGOAL}(v_{\text{new}})$  then
15      return  $\text{SOLUTION}(\mathcal{T}, v_{\text{new}})$ ;
16   $v \leftarrow v_{\text{new}}$ ;
17 return  $\emptyset$ 
```

Auxiliary Planner

```

SAMPLETARGET()
if  $\text{RAND}() < b$  then
  return select one of the  $\mathcal{S}_i.c$  at random;
else return random cfg from entire space;
ACCEPTABLE(v)
return  $\exists \mathcal{S}_i \in \mathcal{S} : v.d + \text{dist}(v.c, \mathcal{S}_i.c) \leq \mathcal{S}_i.r$ ;
REACHEDGOAL(v)
return  $\exists \mathcal{S}_i \in \mathcal{S} : \text{dist}(v.c, \mathcal{S}_i.c) \leq \epsilon$ 
```

Main Planner

```

SAMPLETARGET()
if  $\text{RAND}() < b$  then return  $c_{\text{goal}}$ ;
else return random cfg from entire space;
ACCEPTABLE(v): return  $\text{auxPlanner.PLAN}(v) \neq \emptyset$ 
REACHEDGOAL(v): return  $\text{dist}(v.c, c_{\text{goal}}) \leq \epsilon$ 
```

and navigate narrow passages to reach its destination.

For the maze, random, and waves scenes, we created three difficulty levels, with L3 being the most challenging. Maze levels L1, L2, and L3 correspond to maze dimensions of 10×10 , 14×14 , and 18×18 , respectively. During maze generation, we used Kruskal’s algorithm to set the grid dimensions and randomly place walls. However, to introduce alternative paths in the maze, approximately 10% of the walls were removed. This modification allows for multiple navigation options, but the maze remains

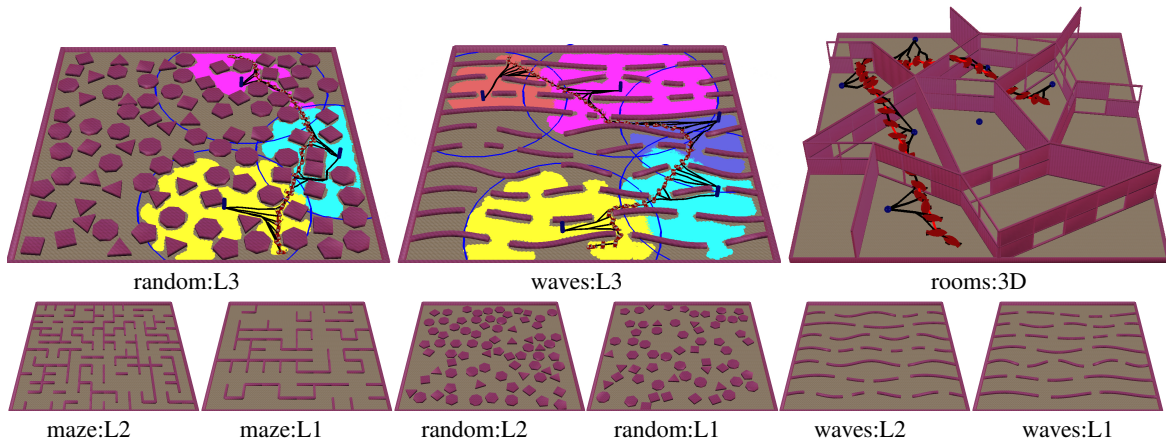


Figure 3: The other scene types (random, waves, rooms) used in the experiments. The maze scene type is shown in Fig. 1. For the maze, random, and waves scene types, we have three levels of difficulty with L3 being the most challenging. In the random and waves scene types at the L3 difficulty level, the figures also illustrate safety zones, along with the safe locations and solutions determined by our approach. Due to the visual complexity of presenting safety zones and safe locations in the 3D rooms scene, we opted to showcase solely the centers of the safety zones and the solution computed by our approach. The car model is used in the maze, random, and waves scene types, while the blimp model is used in the 3D rooms scene.

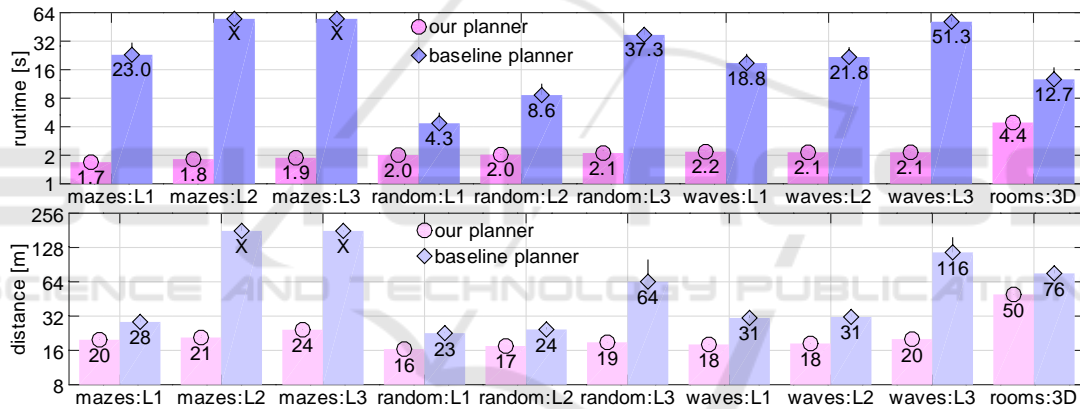


Figure 4: Runtime and solution length results on a logarithmic scale. These results correspond to problem instances where the radii of safety zones are randomly set within the interval $I_2 = [4m, 6m]$. Entries marked with X indicate failure by the baseline planner to solve the problem instances within the 60s time limit per run.

challenging due to its density and zigzag paths.

The random scenes consist of randomly placed obstacles, with varying difficulty levels determined by the percentage of area covered by these obstacles. L1, L2, and L3 correspond to 20%, 25%, and 30% coverage, respectively. These levels create dense environments with limited passages for the robot.

The waves scene type consists of wavy obstacles spaced slightly apart from each other. Each wave contains randomly positioned gaps of varying width. However, these gaps are relatively narrow, requiring the robot to maneuver with precision. Different difficulty levels are achieved by adjusting the number of wave obstacles. L1, L2, and L3 levels contain 5, 7, and 10 wave obstacles, respectively.

The 3D rooms scene comprises multiple rooms,

each containing small openings positioned at various heights that the blimp robot can traverse. This creates a challenging environment, as the blimp robot must navigate with precision, flying from one room to another in order to reach its destination.

5.3 Problem Instances

For the maze, random, and waves scene types, we created 3 scenes for each level of difficulty for a total of 27 scenes. We have one scene for the 3D rooms, so 28 different scenes were used in the experiments.

A problem instance is defined by placing the initial and goal configurations and the safety zones. The initial and goal configurations are placed at collision-free locations near the bottom and top, respectively,

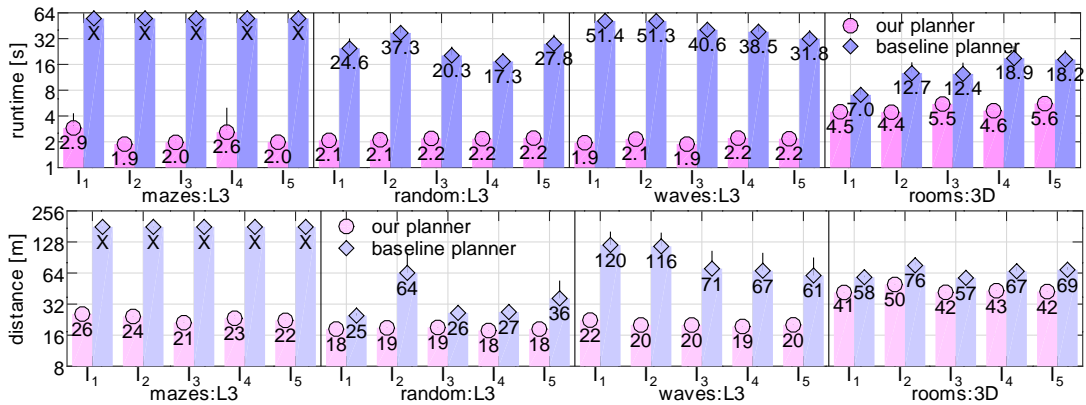


Figure 5: Runtime and solution length results when varying the intervals used to set the radii of the safety zones, where $I_1 = [2m, 4m]$, $I_2 = [4m, 6m]$, $I_3 = [6m, 8m]$, $I_4 = [8m, 10m]$, and $I_5 = [10m, 12m]$.

requiring the robot to navigate a considerable distance. The safety zones are created by randomly positioning their centers and setting their radii at random within specified minimum and maximum values. These safety zones cover both the initial and goal configurations and form a connected component, with overlaps between zones considered as connections.

For the experiments, we generated problem instances with safety zone radii falling into five intervals: $I_1 = [2m, 4m]$, $I_2 = [4m, 6m]$, $I_3 = [6m, 8m]$, $I_4 = [8m, 10m]$, and $I_5 = [10m, 12m]$. For each scene and interval I_j , we created 30 instances following the procedure described above. As a result, we generated a total of $28 \times 5 \times 30 = 4200$ problem instances.

5.4 Measuring Performance

The experiments were conducted on a computing cluster comprising nodes with Dell PowerEdge R640 Intel(R) Xeon(R) Gold 6240R CPU 2.40GHz, each equipped with 48 cores. The experiments did not utilize parallelism or multi-threading; instead, each instance was executed on a single core. The code was implemented in C++ and compiled using g++-9.3.0.

The results provide information on the runtime and solution length for our approach and the baseline planner. The runtime includes the time taken from reading the input until a solution is found or until the runtime limit of 60s per run is reached. To obtain more reliable statistics, the mean and standard deviation are calculated after removing the top and bottom 25% of the results to mitigate the influence of outliers.

5.5 Results

Results over All Scene Types: Fig. 4 presents the runtime and solution length results for all the scene types utilized in the experiments. The findings

demonstrate the computational efficiency of our approach, as it is able to rapidly discover solutions even in highly challenging environments. This efficiency stems from a combination of roadmap construction, identification of safe locations, and search over the safe portion of the roadmap. The roadmap captures the connectivity of the environment, facilitating efficient navigation within and between safety zones. The search-based approaches quickly identify the safe nodes within the roadmap and find a collision-free path to the goal, ensuring that each intermediate configuration along the path can reach a safety center within the distance constraints.

The baseline planner is capable of solving some of the problem types, but requires significant time, and even fails, in the more challenging problems where the robot has to navigate around numerous obstacles and pass through several narrow passages to reach its goal. This outcome is anticipated due to the nature of the baseline planner, which requires invoking the auxiliary planner multiple times to determine paths towards safety centers for each node added to the main tree. Consequently, this process incurs a significant computational cost, making it less efficient and effective in complex problem instances.

When comparing solution lengths, our approach consistently produces significantly shorter solutions than the baseline planner. This is due to the shortest-path searches over the roadmap, enabling our approach to reduce the distance required to reach the safety centers and the goal.

Results When Varying the Area of Safety Zones: Fig. 5 shows the runtime and solution length results obtained by varying the radii of the safety zones across different intervals. These results further highlight the computational efficiency of our approach and its ability to generate short solutions. Our approach consistently performs well across all problem

instances, benefiting from the roadmap construction that facilitates navigation within and between safety zones, even when their areas change. The shortest-path searches conducted over the roadmap enable our approach to identify safe locations and determine a safe path to the goal. In contrast, the baseline planner faces challenges in expanding the tree effectively to satisfy the distance constraints imposed by the safety zones. This further highlights the advantage of our approach in handling varying safety zone radii.

6 DISCUSSION

This paper presented a novel approach to incorporate safety zones into path planning. The approach made it possible to plan the path of a robot to reach its goal while always being able to detour to a safety center within the specified distance constraints in case of emergencies. Experiments in challenging 2D and 3D environments, involving car and blimp robot models, demonstrated the efficacy of the approach.

This work opens up several potential research directions. One direction is leveraging machine learning to predict safe locations. Another is to handle complex tasks, including multiple goals, exploration, and even extend the approach to multiple robots.

REFERENCES

- Baldoni, P., McMahon, J., and Plaku, E. (2022). Leveraging neural networks to guide path planning: Improving dataset generation and planning efficiency. In *IEEE International Conference on Automation Science and Engineering*, pages 667–674.
- Bohlin, R. and Kavraki, L. E. (2000). Path planning using lazy PRM. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 521–528.
- Bui, H.-D., Lu, Y., and Plaku, E. (2022). Improving the efficiency of sampling-based motion planners via runtime predictions for motion-planning problems with dynamics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4486–4491.
- Cao, K., Cheng, Q., Gao, S., Chen, Y., and Chen, C. (2019). Improved PRM for path planning in narrow passages. In *IEEE International Conference on Mechatronics and Automation*, pages 45–50.
- Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press.
- Denny, J., Sandström, R., Bregger, A., and Amato, N. M. (2020). Dynamic region-biased rapidly-exploring random trees. In *Workshop on the Algorithmic Foundations of Robotics*, pages 640–655.
- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.
- Feyzabadi, S. and Carpin, S. (2014). Risk-aware path planning using hierarchical constrained markov decision processes. In *IEEE International Conference on Automation Science and Engineering*, pages 297–303.
- Galler, B. A. and Fisher, M. J. (1964). An improved equivalence algorithm. *Communications of the ACM*, 7(5):301–303.
- Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.
- Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894.
- Kavraki, L. E., Švestka, P., Latombe, J. C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580.
- Kuffner, J. J. and LaValle, S. M. (2000). RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 995–1001.
- Larsen, E., Gottschalk, S., Lin, M., and Manocha, D. (2000). Fast proximity queries with swept sphere volumes. In *IEEE International Conference on Robotics and Automation*, pages 3719–3726.
- LaValle, S. M. and Kuffner, J. J. (2001). Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400.
- Pepy, R. and Lambert, A. (2006). Safe path planning in an uncertain-configuration space using RRT. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5376–5381.
- Plaku, E., Plaku, E., and Simari, P. (2017). Direct path superfacets: An intermediate representation for motion planning. *IEEE Robotics and Automation Letters*, 2(1):350–357.
- Plaku, E., Plaku, E., and Simari, P. (2018). Clearance-driven motion planning for mobile robots with differential constraints. *Robotica*, 36(7):971–993.
- Primatesta, S., Guglieri, G., and Rizzo, A. (2019). A risk-aware path planning strategy for UAVs in urban environments. *Journal of Intelligent & Robotic Systems*, 95:629–643.
- Schouwenaars, T., How, J., and Feron, E. (2004). Receding horizon path planning with implicit safety guarantees. In *American Control Conference*, volume 6, pages 5576–5581.
- Wang, J., Chi, W., Li, C., Wang, C., and Meng, M. Q.-H. (2020). Neural RRT*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, 17(4):1748–1758.
- Zhao, C., Zhu, Y., Du, Y., Liao, F., and Chan, C.-Y. (2022). A novel direct trajectory planning approach based on generative adversarial networks and rapidly-exploring random tree. *IEEE Transactions on Intelligent Transportation Systems*, 23(10):17910–17921.