Robotica (2022), **40**, pp. 136–153
doi:10.1017/S0263574721000436

CAMBRIDGE
UNIVERSITY PRESS

RESEARCH ARTICLE

ADD-RRV for motion planning in complex environments

Peng Cai, Xiaokui Yue*, and Hongwen Zhang

National Key Laboratory of Aerospace Flight Dynamics, Northwestern Polytechnical University, Xi'an 710072, P. R. China
*Corresponding author. Email: xkyue@nwpu.edu.cn

Received: 21 March 2020; **Revised:** 4 February 2021; **Accepted:** 27 March 2021; **First published online:** 14 May 2021

Keywords: Motion planning, Complex environments, Local space identification, Principal component analysis, Bridge test

Abstract

In this paper, we present a novel sampling-based motion planning method in various complex environments, especially with narrow passages. We use online the results of the planner in the ADD-RRT framework to identify the types of the local configuration space based on the principal component analysis (PCA). The identification result is then used to accelerate the expansion similar to RRV around obstacles and through narrow passages. We also propose a modified bridge test to identify the entrance of a narrow passage and boost samples inside it. We have compared our method with known motion planners in several scenarios through simulations. Our method shows the best performance across all the tested planners in the tested scenarios.

1. Introduction

Motion planning problems, requiring to compute a path from start to goal in the configuration space without colliding with obstacles, have been widely studied mainly in robotics, for example, autonomous driving [1], unmanned aerial vehicles [2], and on-orbit operations [3], as well as other research areas, such as computer animation, part assembly [4], and computational biology [5] since the past decades.

Diverse algorithms have been proposed to solve the motion planning problems, for example, A* [6], artificial potential field method [7], and evolutionary algorithms [8]. These algorithms and their variant versions have been successfully applied to scenarios with various constraints. In addition, for motion planning in high-dimensional space, sampling-based planning algorithms have received widespread attention in recent years. Such algorithms conduct a search to explore the configuration space with a sampling scheme, and avoid constructing obstacles into the configuration space explicitly [9]. The most successful sampling-based algorithms include Probabilistic Roadmap Method (PRM) [10] and Rapidly-Exploring Random Tree (RRT) [11].

PRM is used to solve multi-query problems and has two stages. In the offline learning stage, random collision-free samples are generated repeatedly in the configuration space and then connected, if possible, by a local planner with their nearby neighbors to construct a roadmap. In the online query stage, any given start and goal configurations are connected to the roadmap, and then the roadmap is searched for a path. In contrast, RRT is meant to solve single-query problems and incrementally builds a tree from an initial configuration. A random sample is generated iteratively and connected to its nearest node in the tree until the goal configuration is approached close enough by the tree or after a certain number of iterations.

Unfortunately, the performance of sampling-based techniques degrades significantly when handling problems with narrow passages [12, 13]. Narrow passages are collision-free regions that contain parts of the solution, but its volume is relatively small compared with the whole sampling domain. As a result, it is a challenging task for a probabilistic planner to discover such region via generating random

© The Author(s), 2021. Published by Cambridge University Press.

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

samples uniformly and find a path through it. Different strategies have been proposed to improve the basic algorithms, including modified sampling techniques, the retraction-based approaches, utilizing configuration space or workspace information as a guidance, etc.

The representative work that adopts modified sampling techniques is first applied in PRM [14, 15]. They both tend to generate more free samples in difficult regions. The Gaussian sampling strategy [14] obtains samples near obstacles by generating two adjacent configurations each time and retaining the collision-free one only when another one is colliding. The Bridge Test technique [15] shares the similar idea and generates two adjacent configurations to form a line segment as a bridge every time. But it keeps the middle configuration of the line only if the both endpoints are colliding and the midpoint is collision-free. Modified sampling technique is also developed in RRT-based planner. In DD-RRT [16], the effect of obstacle regions on Voronoi bias during the expansion of the tree is taken into consideration and each node of the tree is assigned a sampling radius. The random sample is selected to expand only if it is located within the radius of its nearest node. The radius of new nodes is set to infinity initially and reduces to a designed value once the extension from the nodes is interrupted by obstacles. Therefore, this scheme filters the samples beyond the extension ability of the nodes. Unfortunately, the performance of DD-RRT is sensitive to the radius value and subsequently adaptive tune of this radius was proposed in ref. [17].

The retraction-based approaches try to retract in-colliding configurations along the boundary of the obstacles [18] or toward the medial axis of the free region in the configuration space [19, 20], ensuring the tree expansion through narrow passages more efficient than classical counterparts. However, these methods might run very slowly due to the expensive computational overhead in the less difficult regions (e.g., regions without narrow passages). In Selective Retraction-Based RRT [21], modified bridge line tests are used to concentrate the retraction on narrow passages and avoid expansive expansion in wide open free spaces. But the cost of penetration computation which is necessary in retraction process is still considerable in higher dimension.

The Obstacle-based RRT [22] makes use of the obstacle information in the workspace or configuration space as a hint, and takes different expansion procedures according to obtained information. However, expensive geometric computation of the obstacles is required to guide the extension. In ref. [23], the author used the cell decomposition to distinguish narrow passages from open collision-free regions in workspace, and then adapted the sampling distribution. The ideal of the workspace decomposition is also shared by a new sampling-based algorithm named as XXL [24], where a novel workspace-guided sampling is combined to search through the high dimension configuration space. As proposed in ref. [25], increasing the sampling probability around key configurations close to narrow passages can help to find a solution. But it is not trivial to find such key configuration, since narrow passages are located in the configuration space and usually hard to estimate from the workspace. In refs. [2], [26], multiple trees are generated based on the results of the modified bridge tests, enhancing the local exploration in complex regions. But a large number of samples need to be precomputed to detect such regions. In refs. [27, 28], the approximate solution of a simplified problem is found by scaling down the robot, and then the solution is iteratively used as a guidance to search a solution for the increasing size of the robot in the configuration space, until a solution for the original robot is found. However, the approximate solution might generate an unreachable guidance, which degrades the performance.

Recently, the linear dimensional reduction technique has been introduced to identify the local extension directions of the configuration space based on principal component analysis (PCA) [29]. However, it only leverages the obtained free space information and could lose accuracy due to misidentification if the free space is locally unconstrained or equally constrained in all directions. Moreover, a RRT variant called Rapidly-Exploring Random Vines (RRV) [30] has also been studied based on this technique. In RRV, both free and colliding configurations are used to identify the type of the local space more precisely, and the tree could expand around the obstacles and actively discover the entrance of a narrow passage, behaving like real vines climbing the objects under the support of tendrils. However, the RRV expansion process might be computationally intensive in case it does not use parallel computing for

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

PCA-based collision checking, since the algorithm identifies the shape of the local space by massively generating additional random samples and then checking collision once the basic expansion process is interrupted by obstacles.

In this paper, we propose a novel sampling-based planner that integrates the modified sampling techniques and the use of the configuration information to guide expansion, in order to deal with various complex environments robustly. Our method, named as ADD-RRV, utilizes the ADD-RRT framework [17] to generate the random samples more efficiently. Each time the vine (tree) extension from the nearest node is blocked by obstacles, the colliding configurations are stored in a boundary set. Thereafter, as in RRV [30], we perform PCA on local free and colliding configurations around the nearest node to identify whether the node is located inside a narrow passage, and take different further extension strategies according to the identification results. However, the configurations we use for identification are generated online by the planner, which is different from those used in RRV and dramatically improves the computational efficiency. In order to discover a narrow passage more efficiently and boost the nodes inside the passage, the modified bridge is constructed. And the local free space information is also used to bias the bridge direction, increasing the accuracy in high dimensional spaces. The various demonstrated scenarios show that our approach deals with motion planning problems in complex environments more efficiently in a higher success rate with fewer random samples and less computational overhead.

The rest of the paper is organized as follows. In Section 2, we will present the formulation of the motion planning problem and our proposed method in detail. The corresponding simulation results will be given in Section 3. And the considerations and limitations about our method are discussed in Section 4. The final section is the conclusion and our future work.

2. Proposed method

Let $\mathcal{C}$ denote the configuration space, where the motion planning task is defined. The obstacle region forms the subset $\mathcal{C}_{obst} \subseteq \mathcal{C}$, at which the robot collides with obstacles, and *boundary configurations* are those located close to the surface of $\mathcal{C}_{obst}$. Let $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obst}$ denote the set of the collision-free configurations, at which the robot can move freely. The motion planning task aims to find a collision-free path in configuration space from the start configuration $x_{start} \in \mathcal{C}_{free}$ to the goal configuration $x_{goal} \in \mathcal{C}_{free}$.

Our approach was proposed based on two observations. First, we noticed that collision detection process is extremely time consuming in RRT-based motion planners, and a large number of configurations especially in complex environments, failing to pass the process and be extended to, would be discarded, which is a waste of resources. Second, configurations, whether passing collision detection successfully or not during the random tree expansion, both have rich information to describe the local space. We proposed a method that can reduce the number of the costly collision detection operations by taking advantage of a sample-filtering strategy, and increase the searching efficiency by making full use of the colliding points that would have been dropped in standard RRT.

In this section, we describe a novel motion planning method that actively discovers the entrance of a narrow passage and accelerates the search through the passage, without much extra computational effort. This is achieved by considering ADD-RRT framework to increase sampling efficiency, using modified bridge test to search for configurations in the narrow passages, and utilizing boundary and free configurations obtained online to describe the shape of the local space and guide the search. Details of our approach will be shown in the following section.

2.1. Expanding and gathering boundary configurations

The proposed method adopts ADD-RRT framework considering its ability of dealing with motion planning problems in most complex environments more efficiently by control sampling domain. Since the main technique used to discover and expand through narrow passages is based on RRV and some improvements have been made, we denote our approach as ADD-RRV (Alg. 1).

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

Algorithm 1. ADD-RRV

1: $\mathcal{V}$.AddNode(x_{init})
2: $x_{init}.radius = \infty$
3: $X_{obst} = \emptyset$
4: **while** *IterNum* < *MaxIterNum* **do**
5: **repeat**
6: **if** Random(0, 1) < P_{goal} **then**
7: $x_{rand} \leftarrow x_{goal}$
8: **else**
9: $x_{rand} \leftarrow$ RandomSample($\mathcal{C}$)
10: **end if**
11: $x_{near} \leftarrow$ NearestNeighbor($\mathcal{V}, x_{rand}$)
12: **until** Distance(x_{near}, x_{rand}) < $x_{near}.radius$
13: $x_{new}, x_{obst} \leftarrow$ Extend(x_{near}, x_{rand})
14: **if** $x_{new} \neq \varnothing$ **then**
15: $\mathcal{V}$.AddNode(x_{new})
16: $\mathcal{V}$.AddEdge(x_{near}, x_{new})
17: $x_{new}.radius = \infty$
18: $x_{near}.radius = (1 + \alpha) \times x_{near}.radius$
19: **else**
20: **if** $x_{near}.radius = \infty$ **then**
21: $x_{near}.radius = R_{DD}$
22: **else**
23: $x_{near}.radius = (1 - \alpha) \times x_{near}.radius$
24: **end if**
25: $X_{obst} \leftarrow$ AddBoundPoint(X_{obst}, x_{obst})
26: ObstacleBasedExtend($x_{near}, x_{rand}, x_{obst}, \mathcal{V}, X_{obst}$)
27: **end if**
28: **end while**

Similar to ADD-RRT, in our approach, each node of the vine $\mathcal{V}$ is assigned a radius of dynamic domain. In each iteration, a random configuration x_{rand} is generated with probability P_{goal} using the goal configuration x_{goal} directly and with probability $(1 - P_{goal})$ from the whole configuration space. Then, its nearest node x_{near} in the vine is found in accordance with certain distance metrics. Without loss of generality, Euclidian distance is used in this paper. The random node is selected for extension by a local planner only if the distance between two nodes is within the radius of x_{near} (Lines 5–13 in Alg. 1). ADD-RRV differs from ADD-RRT in two main respects: first, considering that the boundary of obstacles of the configuration space could rarely be explicitly represented, the colliding configurations that would have been discarded in the standard extension process are preserved to estimate the boundary. Second, further extension from x_{near} is tried based on the analysis of the online results of the planner, when extension from x_{near} to x_{rand} is unsuccessful in basic ADD-RRT algorithm.

In the local extension (Alg. 2), for the temporary initial configuration x_{from} and goal configuration x_{to}, a given number of moves with some fixed distance ϵ are made in the direction of $\overrightarrow{x_{from}, x_{to}}$ from x_{from}, successively generating configurations $x_1, \ldots, x_n$. Afterward, collision detection is iteratively taken to determine whether these configurations are free or not. The first colliding configuration, denoted as x_{obst}, is used to represent the boundary configuration. And a series of collision-free configurations generated before the boundary configuration, if exist, are viewed as x_{new}.

In our paper, one move is tried from x_{near} toward x_{rand} (Line 13 in Alg. 1). If this extension is successful, a new configuration x_{new} is generated and added to the vine with its dynamic domain being infinite,

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

Algorithm 2. Extend(x_{from}, x_{to})

1: $x_1, \ldots, x_n \leftarrow$ move given steps from x_{from} in the direction of $\overrightarrow{x_{from}, x_{to}}$
2: **for** $i = 1$ to n **do**
3: **if** IsCollision(x_i) **then**
4: **if** $i > 1$ **then**
5: $x_{new} \leftarrow [x_1, \ldots, x_{i-1}]$
6: **return** (x_{new}, x_i)
7: **else**
8: **return** $(\varnothing, x_1)$
9: **end if**
10: **end if**
11: **end for**
12: $x_{new} \leftarrow [x_1, \ldots, x_n]$
13: **return** $(x_{new}, \varnothing)$

and the radius value of x_{near} is increased just like what ADD-RRT does. Otherwise, the radius value of x_{near} is decreased (Lines 14–24 in Alg. 1). Notice that R_{DD} represents the initial designed radius of dynamic domain and a relatively large value is preferred in our approach, while α is the tune ratio. In basic ADD-RRT, if this extension fails, x_{rand} is rejected and no configuration will be preserved. After the radius of x_{near} is adjusted, the next iteration begins. However, in our approach, the boundary configuration x_{obst} is stored in the *boundary set* X_{obst}. We believe sufficient boundary configurations combined with free configurations in the vine can be used to describe the local shape of the obstacle region, and then identify the directions for further extension from x_{near} based on the type of this region (Lines 25–26 in Alg. 1). Different configurations are generated in our approach to improve the efficiency of the local space identification though, such idea is mainly in accordance to the RRV algorithm.

2.2. *Identifying local space*

When extension to x_{rand} is blocked by obstacles, we try to find other extension directions so that the vine climbs around the obstacles and discovers its way through difficult regions as in RRV [30]. Its pseudocode is shown in Alg. 3. Unlike RRV, in order to classify the local space, we do not generate the tendril local domain set and then the tendril local climbing support set through large quantities of extra uniform random sampling and collision checking operations respectively. Instead, our approach only makes use of the existing information to achieve similar effects, significantly reducing the computational intensiveness. To perform this, free configurations around x_{near} with radius R_a are first collected from nodes in $\mathcal{V}$ and stored in the tendril local domain free set X_{free}. Then, boundary configurations are collected in the same way from the boundary set X_{obst} and stored in the tendril local climbing support set X^t_{obst}, where collision checking operations are no more needed (Lines 1–2 in Alg. 3). Though the radius R_a is an important parameter that is related to the width of narrow passages, we have not focused on delicately selecting it. But a significant large or small value might reduce the efficiency of our approach.

Considering that boundary configurations are located along the surface of the obstacle, we assume that the distribution of obstacle areas in local space could be reflected with sufficient such configurations. Therefore, the principal component analysis (PCA) is performed on the local climbing support (boundary) set X^t_{obst} to compute its covariance matrix and then find the eigenvalues Λ^i_{obst} and eigenvectors U^i_{obst} of the matrix (Line 6 in Alg. 3). Notice that the eigenvectors represent the typical variation directions of the set and the corresponding eigenvalues represent the variance of the set in respect directions. As in RRV, a confidence ellipsoid ε^t_{obst} is constructed and located at the center of X^t_{obst} with its axis directions coinciding with the eigenvectors and the length of each axis related to the corresponding eigenvalue (Fig. 1-left). The ellipsoid reflects the outline of the obstacle surface to some extend, and can be used to

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

Algorithm 3. ObstacleBasedExtend($x_{near}, x_{rand}, x_{obst}, \mathcal{V}, X_{obst}$)

1: $X_{free} \leftarrow$ TendrilDomainFreeSet($\mathcal{V}, x_{near}, R_a$)
2: $X^t_{obst} \leftarrow$ TendrilClimbingSupportSet(X_{obst}, x_{near}, R_a)
3: **if** NotSufficient(X^t_{obst}) **then**
4: $x'_{rand,1} \leftarrow$ ModifiedBridgeTest(x_{obst}, X_{free})
5: **else**
6: $\varepsilon^t_{obst} \leftarrow$ PCABasedEllipsoid(x_{near}, X^t_{obst})
7: $X^t_{free} \leftarrow$ TendrilSetForNarrowPassage($X_{free}, \varepsilon^t_{obst}$)
8: **if** IsEmptySet (X^t_{free}) **then**
9: $x'_{rand,1} \leftarrow$ ConvexObstacleDirection($x_{near}, x_{rand}, \varepsilon^t_{obst}$)
10: $x'_{rand,2} \leftarrow$ ModifiedBridgeTest(x_{obst}, X_{free})
11: **else**
12: $x'_{rand,1} \leftarrow$ NarrowPassageDirection($x_{near}, x_{rand}, X^t_{free}$)
13: **end if**
14: **end if**
15: **for** each $x'_{rand,i}$ **do**
16: $x_{new}, x_{obst} \leftarrow$ Extend($x_{near}, x'_{rand,i}$)
17: $\mathcal{V}$.AddNode(x_{new})
18: $\mathcal{V}$.AddEdge(x_{near}, x_{new})
19: $x_{new}.radius = \infty$
20: $X_{obst} \leftarrow$ AddBoundPoint(X_{obst}, x_{obst})
21: **end for**

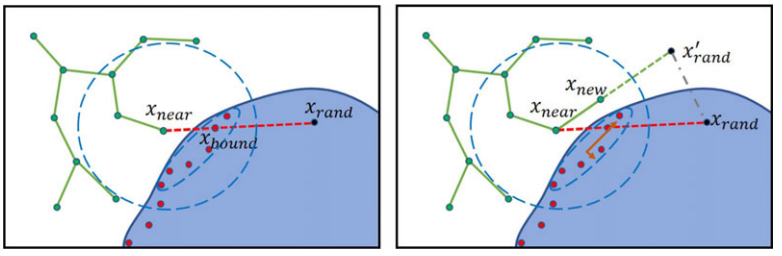


Figure 1. *If a node x_{near} fails to extend to x_{rand}, the neighbor free nodes in the vine (green) and colliding nodes in the boundary set (red) around x_{near} with a radius R_a are collected respectively. x_{near} is believed to be in front of a convex obstacle if no free node is inside the confidence ellipsoid (left). A new extension direction is obtained according to the distribution of the local boundary configurations (right).*

identify the type of the local space. Subsequently, it is checked whether there exists any configuration of X_{free} surrounded by the ellipsoid, and those configurations inside the ellipsoid are stored in the tendril local set X^t_{free} for the narrow passage identification (Line 7 in Alg. 3). In case no such configuration is located inside ε_{obst}, we conclude that x_{near} is in front of a convex obstacle. Otherwise, x_{near} might be inside a narrow passage [30] (Lines 8–13 Alg. 3).

2.2.1. Vine in front of a convex obstacle

In this case, the ellipsoid ε^t_{obst} of the local boundary set has an elongated shape. We assume that the longest axis of the ellipsoid, that is the principal component with the maximum eigenvalue, indicates the local extension direction of the convex obstacle surface. On the contrary, the shortest axis, corresponding to the least variance eigenvector, shows the direction to obstacle from x_{near}. Therefore, the original random extension direction toward x_{rand} needs to be changed according to the results of the PCA, in order to find a preferable direction [30].

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

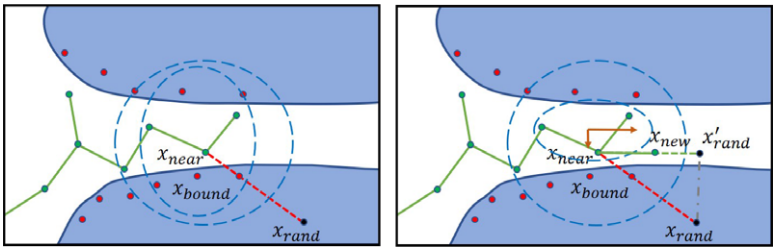


Figure 2. *A node x_{near} that fails to extend to x_{rand} is believed to be inside a narrow passage when there exist free nodes inside confidence ellipsoid (left). A new extension direction is obtained according to the distribution of the local free configurations (right).*

To meet this goal, x_{rand} coordinates are projected to keep high variance directions unchanged while minimizing the extension to low variance direction (Line 9 in Alg. 3). We apply here a similar projection method used in PCA-RRT [29], rather than directly discard the least significant direction as in RRV. This projection method is originally used only inside narrow passages, but we extend it according to this situation. In this way, the vine remains the tendency to climb closer around obstacle, which would facilitate the discovery of narrow passages to some extent. More precisely, placing the origin on x_{near}, the coordinates of the new random direction x'_{rand}, in the eigenbasis, are obtained through

$$x'_{rand} = x_{near} + \sum_{i=1}^{n} \left(\frac{\Lambda^i}{\lambda^1} (x_{rand} - x_{near}) \cdot U^i \right) U^i, \tag{1}$$

where λ^1 is the maximum eigenvalue among Λ^i. Eigenvalues Λ^i_{obst} and eigenvectors U^i_{obst} are used here to generate the new random direction.

The modified random extension direction is illustrated in Fig. 1-right. Growing along this direction, the vine will circumnavigate the convex obstacle much fast, which in turn will help it to efficiently discover a narrow passage as in case of RRV [30].

2.2.2. Vine inside a narrow passage

As in case of the RRV algorithm [30], when there exist some free configurations of the vine located inside the confidence ellipsoid of the local boundary set (Fig. 2-left), we conclude that x_{near} is already inside a narrow passage. In this circumstance, the local free space is extremely constrained and its elongated shape indicates how the vine can extend in this area. Therefore, PCA is performed on the tendril local set X^t_{free} to determine its principal components, and the one with the maximum variance is assumed to align with the longest axis of the local free space with the elongated shape (Fig. 2-right). Thus, the same projection method shown in Eq. (1) is utilized to obtain the modified random extension direction, where the PCA results of the tendril local set X^t_{free}, that is eigenvalues Λ^i_{free} and eigenvectors U^i_{free}, are used in the projection equation (Line 12 in Alg. 3).

However, we might draw the wrong conclusions in this condition as well, since the node x_{near} is also likely to lie in a cluttered area or in front of the non-convex obstacle, as mentioned in [30]. But these wrong conclusions have less effect on the performance of our approach, and the reasons are: first, the time-consuming PCA computation is conducted only after enough local configurations have been collected, which means the vine might have expanded through such less complex area before expensive computation is needed. Second, even in the worst case when x_{near} is in a corner-like region, our expand process will not get stuck and waste time trying further extension, since we use the ADD-RRT framework and with the number of iterations increasing, the probability of this node selected to be extended again gradually decreases.

Notice that compared with the extension process of standard RRT, above process can acquire more efficient extension directions after the expansion of the vine is interrupted by obstacles, especially narrow

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

Algorithm 4. ModifiedBridgeTest(x_{obst}, X_{free})

1: $d_b \leftarrow$ a random line direction from x_{obst}
2: $d'_b \leftarrow$ PCABiasedDirection(d_b, X_{free})
3: $x_{end} \leftarrow$ the endpoint of the line according to the Gaussian distribution
4: **if** IsCollision(x_{end}) **then**
5: $x' \leftarrow$ the midpoint of the line $\overline{x_{obst}, x_{end}}$
6: **if** !IsCollision(x') **then**
7: **return** x'
8: **end if**
9: **end if**
10: **return** $\varnothing$

passages, through identifying the types of the local space around x_{near}. But it still cannot actively discover the entrance of a narrow passage. We do not perform the similar operation like RRV does to positively distinguish whether there exists an entrance (details can be found in ref. [30]), because we do not have enough boundary configurations for further analysis when the vine reaches the entrance in the early stage. For an n-dimensional space, at least $(n+1)$ configurations are needed to perform PCA computation. Instead, we perform our modified bridge test to increase the probability of finding the entrance of a narrow passage and sample more configurations inside the narrow passage (Line 4 in Alg. 3).

2.3. Finding key configurations

When the first extension from x_{near} is failed and the boundary configurations around x_{near} are insufficient for further analysis, we have little knowledge about the types of the local space around x_{near}. Then, the modified bridge test is performed to determine whether there exist configurations located inside a narrow passage. These configurations are key to guide the vine to get into a narrow passage and accelerate the extension through it.

As shown in Alg. 4, a bridge line is generated in a random direction with one end at x_{obst}. Though different methods might exist, we generate this direction d_b in a uniform distribution manner. Then, the other endpoint of the line x_{end} is obtained by computing a random distance from x_{obst} according to a Gaussian distribution. And we set the mean of the Gaussian distribution to be the radius value that we use to collect the neighbor configurations around x_{near} (i.e., R_a), and the standard deviation to be $R_a/2$. Afterward, collision detection is performed on the endpoint x_{end} and the midpoint of the bridge line segment x'. Only if x_{end} is a colliding configuration and x' is collision-free, x' is considered in a narrow passage and used as a new extension direction (see Fig. 3-left). If this test fails, no new direction for the further extension will be generated and the algorithm will return a null value (Line 10 in Alg. 4).

The above bridge test is efficient to find key configurations to guide the extension, especially when x_{near} is already inside a narrow passage. However, its performance degrades with the dimension of the configuration space getting higher, since the bridge test is performed only in one dimension at one time. Once again the information of the local free space distribution can be used to bias the direction of the bridge test line. In a similar manner in the previous section, PCA on the tendril local domain free set X_{free} is performed. Notice that we have not clearly identified the type of the local space at this time, thus we assume that the principal component with the minimum eigenvalue indicates the constrained direction of the space. In order to generate the random line direction to cross the extension direction of the narrow passage, the coordinates of the direction vector is projected based on the results of PCA computation, biased to the constrained direction (see Fig. 3-right). The projection is based on the following equation:

$$d'_b = \sum_{i=1}^{n} \left(\frac{\lambda^1}{\Lambda^i} d_b \cdot U^i \right) U^i. \tag{2}$$

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

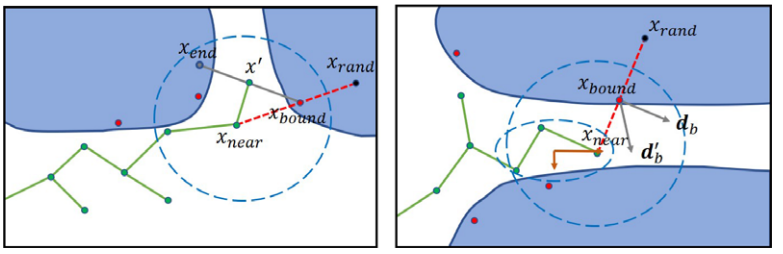


Figure 3. *This figure shows two occasions involved in the modified bridge test: if configurations around x_{near} are insufficient for local space identification, the bridge test is used to find the entrance of a narrow passage (left). If x_{near} is already inside a narrow passage, the bridge test can boost configurations inside it. And the distribution of the local free configurations are used to bias the bridge direction to improve the test (right).*

In addition, in order to overcome the negative effect of the inaccurate assumption, every time Alg. 4 is called, the original line direction d_b is used to perform the bridge test with probability 0.5 instead of the PCA-biased direction d'_b.

The modified bridge test is utilized mainly to boost the configuration density in narrow passages when the local boundary set is insufficient to identify the local space. However, we might still get wrong identification results without rich enough configurations, for example, we might mistake an entrance of a narrow passage as a convex obstacle because the vine has not expanded into the entrance. Therefore, we need to perform the bridge test again to look for the entrance configuration, though x_{near} has been believed in front on a convex obstacle (Line 9 in Alg. 3).

Although the above modification has been made to improve the probability of finding a new extension direction, the performance of the bridge test would still degrade inevitably with the dimension increasing. Therefore, every time Alg. 4 is needed, the modified bridge test could be constructed multiple times before a new direction is found.

2.4. Trying further extension

After the new directions have been constructed, different extension strategies are taken in accordance with the types of the local space around x_{near} (Lines 15–16 in Alg. 3). First, as in [30], if x_{near} is in front of a convex obstacle, one step extension to the new direction x'_{rand} obtained through Line 9 in Alg. 3 is tried in order to make the vine grow around the obstacle instead of rejecting the original x_{rand} like the classic RRT algorithm does. And at most one new configuration x_{new} would be generated. Second, if x_{near} is already located inside a narrow passage, a greedy extension strategy is necessary for the vine to quickly expand through the passage. As in [30], in our paper, the vine continuously extends along the modified random direction obtained through Line 12 in Alg. 3 with the given number of steps before the process is interrupted by obstacles, generating a series of successive new configurations. Although other extension strategy that takes into account the length of the narrow passage might exist, our strategy is general and efficient enough to handle various problems. Lastly, in the case that a key configuration is discovered through the modified bridge test, we try to extend the vine to this configuration directly, and multiple intermediate new configurations might be generated until the key configuration is reached or the extension process is interrupted by obstacles. Considering the significant value of the key configuration, if the key configuration fails to be reached, this configuration could be stored to bias the sampling process (in Alg. 1) in future iterations.

After the above extension process, all newly generated free nodes x_{new} are added into the vine in succession and their initial radius of dynamic domain are set as infinity. The boundary configurations x_{obst}, if exist, are stored in the boundary set X_{obst}. Notice that whether any new configuration is generated successfully or not, the dynamic domain radius of x_{near} would not be tuned again here. In addition, the

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

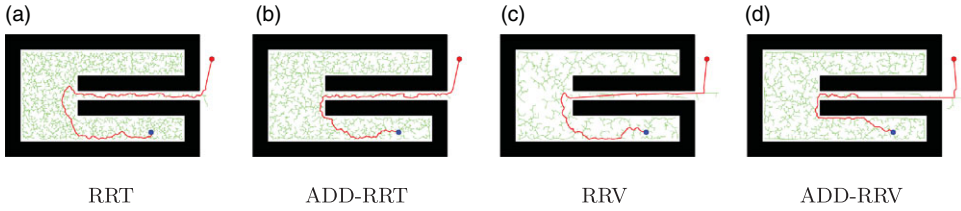


Figure 4. *Realization solution of the bug trap problem by four algorithms.*

case mainly arising from Line 4 in Alg. 3 may also occur that no new extension direction is obtained, when Alg. 4 is called but the null value is returned. Then, this attempt for the further extension is over, and we return to the random sampling process in Alg. 1 directly for the next iteration.

3. Simulation results

In this section, we compare the proposed ADD-RRV motion planning algorithm with classic RRT, basic ADD-RRT, and RRV. In order to demonstrate the advantages of dealing various complex environments of our approach, we present results in three representative scenarios, including moving rigid robots with different shapes in the two-dimensional workspace and operating a 6-degree of freedom (DOF) planar manipulator in a tight workspace.[1] We count the number of iterations, number of generated nodes, number of collision detection (CD) calls, number of PCA calls, and total computation time to compare these motion planning algorithms. Note that the local space analysis in RRV needs additional collision detection calls, which are performed by parallel computing and not counted in the number of classic collision detection, thus we count the number separately.

All of the simulations were performed using MATLAB on a 2.60 GHz Intel i7-9750 laptop with 16 GB RAM and six cores. The simulation models were represented with the convex polygons or circles, and the collision detection was realized using the GJK algorithm [31]. Unless otherwise stated, the Euclidian distance metric with the same weight was used for sampling, searching neighbor configurations and PCA computation. All algorithms have been run 100 times for each problem and we stopped the motion planner after 100×10^3 iterations if it failed to find the solution for each trial.

We set the biased probability P_{goal} for directly sampling the goal configuration to 0.1. The expansion step for all planners was set to $\epsilon = 0.1$ map units. For ADD-RRT framework, we set the dynamic domain radius $R_{DD} = 30\epsilon$ with the adaptive rate $\alpha = 0.05$. We used a fixed radius $R_a = 5\epsilon$ to collect the neighbor configurations and identify the type of the local space, which is adopted from the ref. [30] and shows good adaptability in our simulations. In addition, for each further analysis in RRV, N configurations are required to be sampled and perform extra collision checking parallelly. In our simulations, we set $N = 100n$ and n is the dimension of the configuration space. In this section, every time Alg. 4 was called by the proposed method during simulations, the modified bridge test was constructed only once. And the effects of multiple bridge tests will be discussed in Section 4.

3.1. Modified bug trap scenario

The modified bug trap scenario is adopted from the ref. [30]. The size of the tested environment is 10×10 map units and the task was to move the circular robot with the radius $r = 0.1$ map units out of the trap. The configuration space consists of two-dimensional (2D) translations. Figure 4 illustrates one realization of all tested methods in this scenario. Unlike RRV which is aware of a narrow passage once arriving, ADD-RRV needs more try and operations before identifying the narrow passage.

As shown in Table 1, the average value of the number of iterations in ADD-RRV is about 6 times better than in classic RRT, 2 times better than in ADD-RRT and 1.8 times better than in RRV. Correspondingly, in the terms of the number of classic CD calls, our method shows the similar but slightly reduced

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

Table 1. *Comparison of planners in modified bug trap problem.*

	RRT	ADD-RRT	RRV	ADD-RRV
Iterations	23,499.26	7823.56	6798.7	3882.64
Nodes	3027.68	2996.6	1647.33	1892.97
Classic CD calls	23,499.26	7823.56	7889.66	4823.49
Parallel CD calls	0	0	453.04	0
PCA computation	0	0	584.89	405.68
Running time (s)	4.93	2.99	14.73 (Ideally 2.21)	1.71

Note: All statistics are the average of 100 runs.

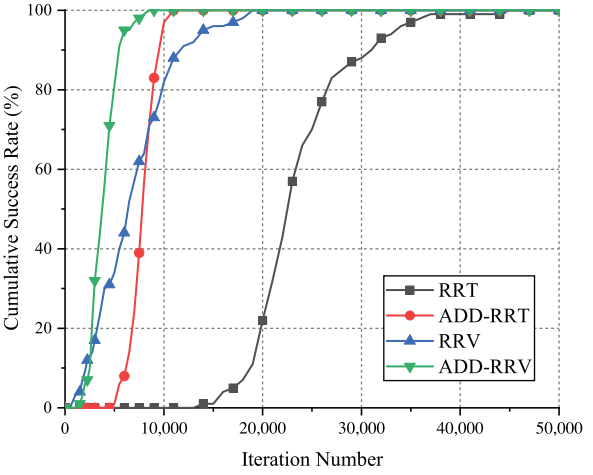


Figure 5. *Cumulative success rate of finding a solution with the given number of iterations in the bug trap scenario.*

improvements compared with classic RRT, ADD-RRT, and RRV, respectively, since several additional CD calls in ADD-RRV are required to perform the bridge tests and further attempts to extend. Moreover, the ratio of the number of nodes added into the tree to the number of collision detections in ADD-RRV is higher over the other tested methods, which reflects the utilization efficiency of randomly sampled configurations to some extent. However, it is notable that a minimal number of nodes are generated in RRV, which means the least memory is required.

Both our method and RRV need to conduct PCA on the local configuration set to identify the narrow passage and then extend greedily through the passage. In this scenario, the number of PCA computation in our method is slightly fewer than in RRV. However, RRV requires additional N one configuration CD calls to classify the random configuration set before PCA operation, that is each parallel CD call includes N operations of collision checking. Although such collision detection could be performed parallelly, its execution time varies depending on the simulation platform and computer performance. In our simulation environment, parallel computing only provides a slight improvement in execution time. As a result, the average running time of RRV is much more than the other three methods, 84.98% of which is the execution time of parallel CD calls. But it should be admitted that the total execution time of RRV could be reduced further with sufficient computational resources, and the theoretically ideal time is about 2.21 s (calculated by subtracting the execution time of parallel CD calls from the total time). In contrast, ADD-RRV makes about 1.9 times improvement over classic RRT and 75% over ADD-RRT in terms of running time, and is still more efficient than theoretically ideal RRV.

According to the cumulative success rate (shown in Fig. 5), the performance of ADD-RRV is better than other tested methods in most cases. Overall, ADD-RRV achieves a higher success rate to find a

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

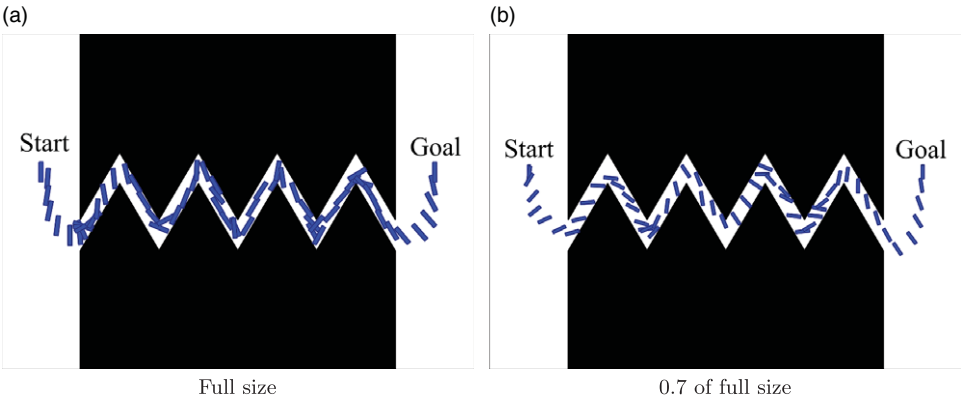


Figure 6. *This figure shows typical planning results of ADD-RRV in the the long and narrow jaggy passage with two different sizes of the robot.*

solution with fewer iterations and even the worst case of ADD-RRV outperforms the best one of RRT. Consequently, fewer CD calls are required in ADD-RRV, which explains its less running time than the other three methods.

3.2. Long and narrow jaggy passage scenario

In the second scenario, a rigid segment robot moves through a long and narrow jaggy passage via translating and rotating. The size of this environment is 12×8 map units, and the width of the passage is about 0.46 map units. The full size of the robot is 0.5×0.1 map units. When inside the passage, the robot has to translate along the direction of the passage. Translation in the orthogonal directions and rotation are highly restricted. It can only reorient at corners. In addition, another robot with the same shape, but 0.7 of the full size is also considered as a simple situation for comparison, which can rotate in the passage more freely. Thus, the motion planning problems are both constructed in the $\mathbb{R}^2 \times \mathbb{S}^1$ space, that is 2D translations and rotations (given in radians). The typical planning results of the two robots realized by ADD-RRV are illustrated in Fig. 6.

The statistics of planning results of the two robots are summarized in Table 2. In both situations, ADD-RRV shows more efficiency over classic RRT and ADD-RRT in terms of the number of iterations, number of classic CD calls, and running time. Moreover, the possibility of finding a path with fewer iterations by ADD-RRV is also higher than that by classic RRT and ADD-RRT (see Fig. 7). Notice that even though after 100×10^3 iterations, the possibility that a solution could be found by RRT in the full-size scenario is about 92%. For a given problem and planner, our underestimation of the average running time to find a solution increases with the success rate decreasing. It is notable that RRV generates the minimum number of nodes in both situations.

In the full-size situation, ADD-RRV and RRV share the comparable performance in terms of the number of iterations and classic CD calls. However, less PCA computation is required by ADD-RRV due to the combined utilization of the ADD-RRT framework and the modified bridge test, which boosts the number of the free configurations lying inside the narrow passage. As a result, ADD-RRV spends less time on average finding a solution than theoretically ideal RRV.

Ideally, the situation of the robot with small size should be easier for all planners. In fact, when using the Euclidian distance metric with the same weights relative to translations and rotations, although RRV does make improvements in some aspects compared with classic RRT, the performance of RRV with small size is worse than that with full size. However, after using the carefully selected weights (1 and 0.5 relative to translations and rotations, respectively), all planners perform better while RRV makes the most improvement. This phenomenon shows the significance of the proper metric weights and that

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

Table 2. *Comparison of planners in a long and narrow jaggy passage scenario.*

	RRT	ADD-RRT	RRV	ADD-RRV
Full size (same weights)				
Iterations	68,862.98	22,481.15	9252.54	8102.57
Nodes	8993.93	10,241.01	2125.85	6776.46
Classic CD calls	68,862.98	22,481.15	17,746.81	18,449.56
Parallel CD calls	0	0	7929.92	0
PCA computation	0	0	14,914.50	8529.10
Running time (s)	34.24	32.51	362.75 (Ideally 29.46)	18.88
Success rate	92%	100%	100%	100%
0.7 of full size (same weights)				
Iterations	24,867.28	12,010.61	12,565.13	4014.76
Nodes	3864.78	5628.25	3265.33	4171.43
Classic CD calls	24,867.28	12,010.61	22,015.80	10,152.91
Parallel CD calls	0	0	8363.03	0
PCA computation	0	0	15,609.83	4316.16
Running time (s)	8.37	10.56	392.85 (Ideally 38.89)	7.69
0.7 of full size (selected weights)				
Iterations	21,018.43	8278.15	7425.94	2851.38
Nodes	3035.30	3956.11	2003.08	2916.79
Classic CD calls	21,018.43	8278.15	14,537.27	6400.66
Parallel CD calls	0	0	6333.75	0
PCA computation	0	0	12,331.00	2711.83
Running time (s)	7.48	6.16	287.15 (Ideally 18.41)	4.54

Note: All statistics are the average of 100 runs.

our integrated method, which makes comprehensive use of the modified sampling techniques and the informative configurations obtained online by the planner to identify the extension directions, is more efficient and robust in some degree.

3.3. 6-DOF manipulator

The last scenario contains a 6-DOF articulated planar manipulator with a fixed base and the size of each link being 1×0.1 map units. The manipulator starts at a unconstrained configuration where all its joints can move freely. Then the manipulator has to move along the obstacle and insert itself into a narrow opening to reach the goal configuration. The search trees (vines) are constructed in the 6D joint space, where both start and goal configurations are given by the angular positions of joints. And every time the collision detection procedure was called, the forward kinematics problem needed to be solved first to determine the position and orientation of the center point of each link, based on which the geometric model boundary of each link was calculated for collision detection. Then collision detection was performed between links and obstacles as well as between links themselves. One typical realization of ADD-RRV is illustrated in Fig. 8-left.

The statistics of the performance of the tested algorithms are shown in the Table 3. In this scenario, the task has been challenging for classic RRT and ADD-RRT with only 81% and 87% of possibility, respectively, to find a solution after 100×10^3 iterations. In contrast, both ADD-RRV and RRV could solve the problem in less than 70×10^3 iterations (Fig. 8-right). However, ADD-RRV is still more efficient than RRV in all aspects.

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

Table 3. *Comparison of planners in the manipulator operation problem.*

	RRT	ADD-RRT	RRV	ADD-RRV
Iterations	66,643.58	59,844.55	25,382.44	13,543.16
Nodes	43,572.49	45,822.69	17,626.73	15,392.65
Classic CD calls	66,643.58	59,844.55	28,864.85	22,212.22
Parallel CD calls	0	0	2570.05	0
PCA computation	0	0	5133.29	5149.52
Running time (s)	234.30	175.41	334.06 (Ideally 65.71)	43.22
Success rate	81%	87%	100%	100%

Note: All statistics are the average of 100 runs.

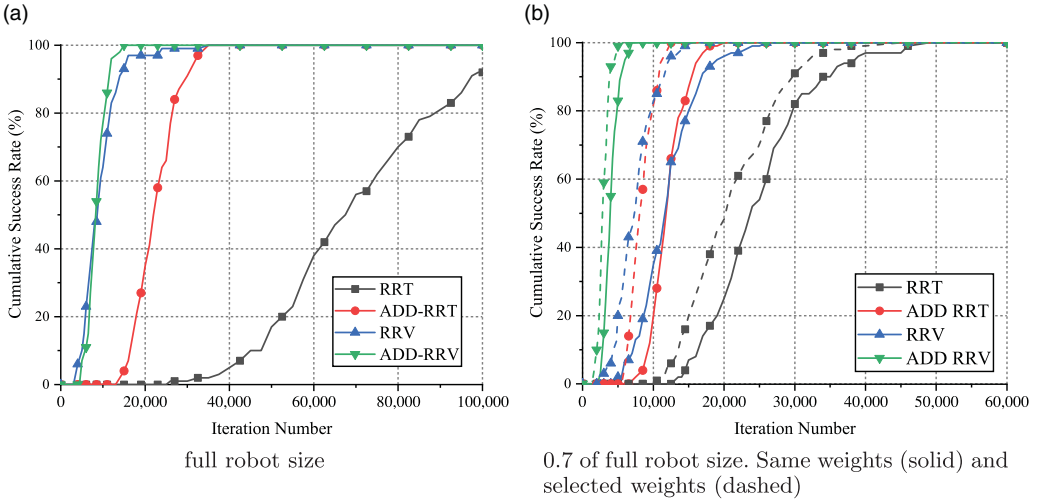


Figure 7. *Cumulative success rate of finding a solution with the number of iterations in the long narrow passage scenario.*

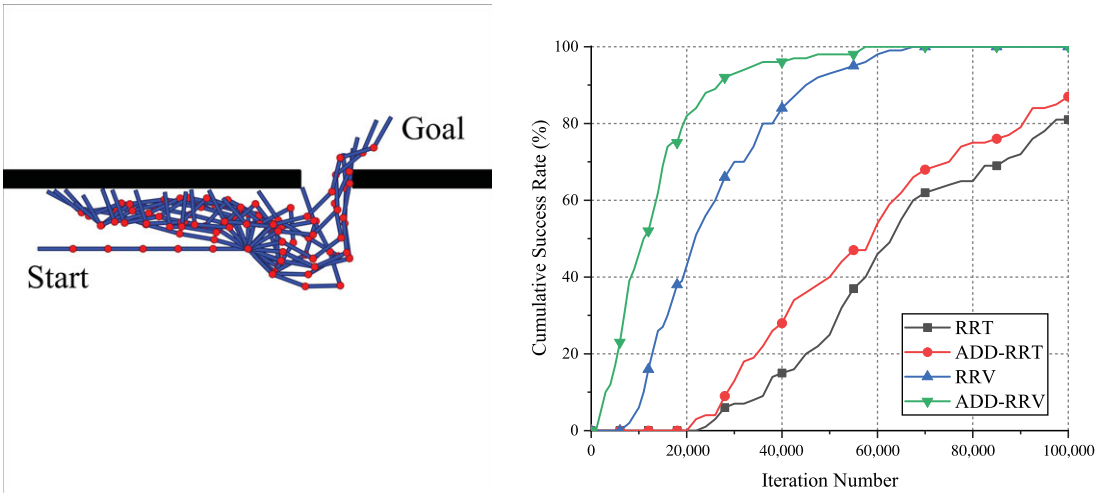


Figure 8. *This figure shows the typical planning result of ADD-RRV (left) and Cumulative success rate of finding a solution with the number of iterations (right) in the manipulator operation scenario.*

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

Table 4. *Breakdown of the running time of the ADD-RRV.*

	CD (%)	PCA (%)	NN (%)	Others (%)
Bug trap	49	9	22	20
Long Passage – 1*	37	18	23	22
Long Passage – 0.7*	47	20	14	19
6-DOF manipulator	66	6	11	17

*1 denotes the full robot size, while 0.7 denotes the smaller size and only the situation with the same metric weights is considered here.

4. Discussion

In this section, we will discuss some important issues of our approach.

4.1. Analysis of efficiency

Our approach shows higher efficiency overall in various complex environments, but it generates a larger proportion of nodes relative to the number of randomly sampled configurations compared with other tested methods. Two factors account for this situation. First, the sample filtering feature of the adopted ADD-RRT framework improves the utilization efficiency of the randomly sampled configurations, that is the probability of the random samples to be successfully extended to becomes higher. Besides, even if the random samples are rejected by the basic ADD-RRT extension process, further extension is tried based on the identification of the local space. This process generates more nodes again, especially during the greedy extension through the identified narrow passages.

The above factors ensure our approach to find a solution of the motion planning problems with fewer iterations, increasing the success rate of our approach. However, a portion of the generated nodes are located at the wide open free spaces. These nodes make few contributions to find a solution in the complex environment but require considerable memory. This situation worsens when the dimension of the configuration space gets higher and the constrained space is relatively small compared with the sampling domain, since we have not taken the techniques to filter the random samples in such open regions.

Moreover, dense nodes might decrease the efficiency of finding the nearest neighbor of the random nodes and searching the neighbor configurations around the blocked nodes, where minimun function and matrix operations were applied in our simulation. As shown in Table 4, neighbor node (NN) searches take about 11–23% of the overall running time in the tested scenarios using ADD-RRV. Collision detection occupies most of the running time. In addition, PCA computation takes a greater percentage of running time in the long narrow passage compared with other scenes.

In addition, our approach needs extra memory to store the boundary configurations. The number of boundary configurations is less than the total number of CD calls minus the number of generated nodes, since we did not store the colliding configurations from the bridge tests. But this number is still considerable. In order to reduce the storage memory and improve the efficiency of the neighbor node search, we set a distance threshold. If the minimum distance between the new boundary configuration and the existing ones is less than the threshold, this configuration is no longer be saved repeatedly.

4.2. Implementation issues

As stated in Section 2.3, in order to alleviate the negative impact of the inevitable decline in algorithm efficiency with the space dimension increasing, multiple bridge tests could be performed before a key configuration is found if Alg. 4 is called. The effects of the number of bridge tests on the number of iterations and execution time have been shown in Fig. 9. All results were obtained form the average of

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

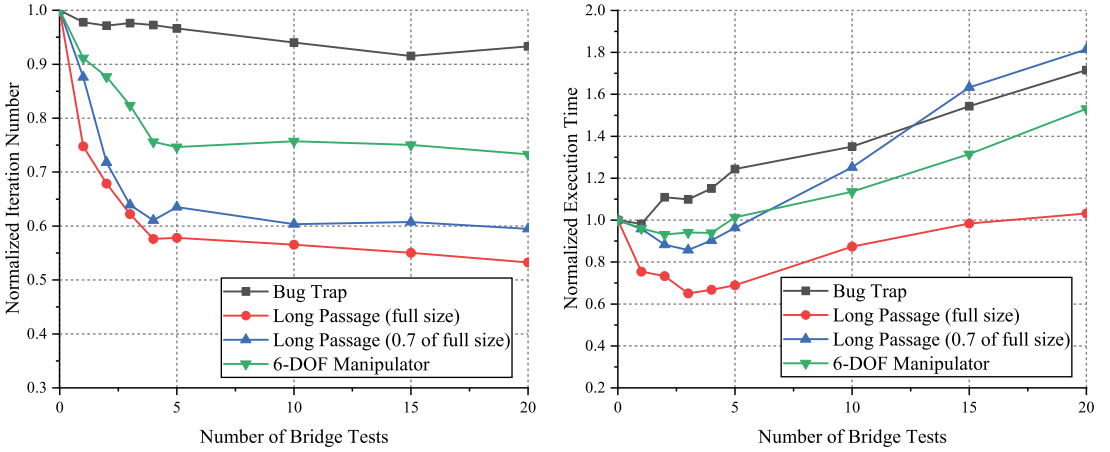


Figure 9. *Effects of multiple bridge tests on the number of iterations (left) and execution time (right). Note that the iteration number and the execution time in each scenario have been divided by those without bridge tests, respectively, and the number of bridge tests have been divided by the dimension of space.*

100 runs and only the situation with the same metric weights was considered. To facilitate comparison, the statistics of the iteration number and the execution time in each scenario have been divided by those without bridge tests respectively, and the number of bridge tests have been divided by the dimension of space n. Obviously, the iteration number shows the decreasing and gradually stabilizing trend overall, while the execution time gradually increases after a temporary dropping with the number of bridge tests increasing, which results from the rising number of collision detections. Thus, considering the performance of both parts, about $3n$ bridge tests would be a preferable choice for the practical implementation.

The metric plays a significant role in sampling-based motion planners, as it indicates the cost to go between two configurations and which configurations could be connected easier. It is commonly believed that a proper distance metric between two configurations should reflect the workspace region swept by the robot when traveling between them, and the appropriate metric will promote the performance of a planner. Therefore, it has to be admitted that the Euclidian distance with carefully chosen different weights (e.g., different weights relative to translations and rotations) might outperform that with same weights. However, in this paper we have not been devoted to choose a metric. In addition, as mentioned in [29], the PCA method is scale dependent and therefore the normalization of the different DOFs (i.e., the Euclidian distance metric with different weights) has been used there. Above effect has also been illustrated in our simulation of the long and narrow jaggy passage scenario, where RRV performs worse in case of the scaling down robot when selecting the improper metric weights. In contrast, ADD-RRV shows less sensitivity to the metric weights.

5. Conclusion and future works

We have presented a novel integrated RRT algorithm, that is ADD-RRV, which addresses the motion planning problem in the complex environment more efficiently. We adopted ADD-RRT framework to filter the random samples that exceed the exploration ability of the vine. Moreover, the colliding and free configurations generated online by the planner are used to retain the advantages of RRV, that is efficiently identifying and extending through the complex regions, but require no additional collision detections, thus significantly reducing the computational intensiveness. In order to discover the entrance of a narrow passage and increase nodes sampled inside the passage, we proposed the modified bridge test and used

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

the local space information to bias the bridge direction. As is demonstrated in simulations, our approach always shows the best performance among all the tested approaches in all the tested scenarios. This result shows that our approach is more robust and general in the complex environment.

There are several issues we are interested in to improve the current work in the future. We would like to design more efficient filtering methods that more samples are generated in the complex areas with fewer in wide open free regions to avoid generating too many nodes. In addition, better neighbor node search methods could be adopted to improve the performance of our approach. Lastly, we would also like to extend our research to the motion planning of the redundant manipulator, where the tasks described in Cartesian space will be taken as inputs and various target configurations corresponding to the same task will be considered.

Acknowledgments. This work is supported by National Natural Science Foundation of China (11972026).

Note

1 https://github.com/HarryCP/ADD-RRV.git.

References

[1] Y. Rasekhipour, A. Khajepour, S. K. Chen and B. Litkouhi, "A potential field-based model predictive path-planning controller for autonomous road vehicles," *IEEE Trans. Intell. Transp. Syst.* **18**(5), 1255–1267 (2016).
[2] D. Li, Q. Li, N. Cheng and J. Song, "Extended RRT-based Path Planning for Flying Robots in Complex 3D Environments with Narrow Passages," *IEEE International Conference on Automation Science and Engineering (CASE)* (2012) pp. 1173–1178.
[3] T. Rybus, "Point-to-point motion planning of a free-floating space manipulator using the rapidly-exploring random trees (RRT) method," *Robotica*, **38**(6), 957–982 (2020).
[4] M. McEvoy, E. Komendera and N. Correll, "Assembly Path Planning for Stable Robotic Construction," *IEEE International Conference on Technologies for Practical Robot Applications (TePRA)* (2014) pp. 1–6.
[5] V. Vonásek and B. Kozlíková, "Tunnel Detection in Protein Structures Using Sampling-based Motion Planning," *11th International Workshop on Robot Motion and Control (RoMoCo)* (2017) pp. 185–192.
[6] P. E. Hart, N. J. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.* **4**(2), 100–107 (1968).
[7] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *Proceedings of the IEEE International Conference on Robotics and Automation* (1985) pp. 500–505.
[8] S. Hosseininejad and C. Dadkhah, "Mobile robot path planning in dynamic environment based on cuckoo optimization algorithm," *Int. J. Adv. Robot. Syst.* **16**(2), 1–13 (2019).
[9] S. M. LaValle, *Planning Algorithms* (Cambridge University Press, Cambridge, England, United Kingdom, 2006).
[10] L. E. Kavraki, P. Svestka, J.-C. Latombe and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.* **12**(4), 566–580 (1996).
[11] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," TR 98-11 (Computer Science Department, Iowa State University, 1998).
[12] D. Hsu, J.-C. Latombe and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *Int. J. Robot. Res.* **25**(7), 627–643 (2006).
[13] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani and S. Sorkin, "On Finding Narrow Passages with Probabilistic Roadmap Planners," *Robotics: the Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics* (1998) pp. 141–154.
[14] V. Boor, M. H. Overmars and A. F. Van Der Stappen, "The Gaussian Sampling Strategy for Probabilistic Roadmap Planners," *Proceedings of the IEEE International Conference on Robotics and Automation* (1999) pp. 1018–1023.
[15] D. Hsu, T. Jiang, J. Reif and Z. Sun, "The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners," *Proceedings of the IEEE International Conference on Robotics and Automation* (2003) pp. 4420–4426.
[16] A. Yershova, L. Jaillet, T. Siméon and S. M. LaValle, "Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling Domain," *Proceedings of the IEEE International Conference on Robotics and Automation* (2005) pp. 3856–3861.
[17] L. Jaillet, A. Yershova, S. M. La Valle and T. Siméon, "Adaptive Tuning of the Sampling Domain for Dynamic-Domain RRTs," *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2005) pp. 2851–2856.
[18] L. Zhang and D. Manocha, "An Efficient Retraction-based RRT Planner," *Proceedings of the IEEE International Conference on Robotics and Automation* (2008) pp. 3743–3750.

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press

[19] S. A. Wilmarth, N. M. Amato and P. F. Stiller, "MAPRM: A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space," *Proceedings of the IEEE International Conference on Robotics and Automation* (1999) pp. 1024–1031.
[20] J. Denny, E. Greco, S. Thomas and N. M. Amato, "MARRT: Medial Axis Biased Rapidly-Exploring Random Trees," *Proceedings of the IEEE International Conference on Robotics and Automation* (2014) pp. 90–97.
[21] J. Lee, O. Kwon, L. Zhang and S.-e. Yoon, "SR-RRT: Selective Retraction-based RRT Planner," *Proceedings of the IEEE International Conference on Robotics and Automation* (IEEE, 2012) pp. 2543–2550.
[22] S. Rodriguez, X. Tang, J.-M. Lien and N. M. Amato, "An Obstacle-based Rapidly-Exploring Random Tree," *Proceedings of the IEEE International Conference on Robotics and Automation* (2006) pp. 895–900.
[23] J. P. Van den Berg and M. H. Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners," *Int. J. Robot. Res.* **24**(12), 1055–1071 (2005).
[24] R. Luna, M. Moll, J. Badger and L. E. Kavraki, "A scalable motion planner for high-dimensional kinematic systems," *Int. J. Robot. Res.* **39**(4), 361–388 (2020).
[25] E. Szádeczky-Kardoss and B. Kiss, "Extension of the Rapidly Exploring Random Tree Algorithm with Key Configurations for Nonholonomic Motion Planning," *IEEE International Conference on Mechatronics* (2006) pp. 363–368.
[26] W. Wang, L. Zuo and X. Xu, "A learning-based multi-RRT approach for robot path planning in narrow passages," *J. Intell. Robot. Syst.* **90**(1–2), 81–100 (2018).
[27] V. Vonásek, "Motion Planning of 3D Objects Using Rapidly Exploring Random Tree Guided by Approximate Solutions," *IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)* (2018) pp. 713–720.
[28] V. Vonásek and R. Pěnička, "Computation of Approximate Solutions for Guided Sampling-Based Motion Planning of 3D Objects," *12th International Workshop on Robot Motion and Control (RoMoCo)* (2019) pp. 231–238.
[29] S. Dalibard and J.-P. Laumond, "Linear dimensionality reduction in random motion planning," *Int. J. Robot. Res.* **30**(12), 1461–1476 (2011).
[30] A. Tahirovic and M. Ferizbegovic, "Rapidly-Exploring Random Vines (RRV) for Motion Planning in Configuration Spaces with Narrow Passages," *Proceedings of the IEEE International Conference on Robotics and Automation* (2018) pp. 7055–7062.
[31] G. v. d. Bergen, "A fast and robust GJK implementation for collision detection of convex objects," *J. Graphics Tools* **4**(2), 7–25 (1999).

Cite this article: P. Cai, X. Yue and H. Zhang (2022). "ADD-RRV for Motion Planning in Complex Environments", Robotica **40**, 136–153. https://doi.org/10.1017/S0263574721000436

https://doi.org/10.1017/S0263574721000436 Published online by Cambridge University Press