2018 IEEE International Conference on Robotics and Automation (ICRA)
May 21-25, 2018, Brisbane, Australia

Rapidly-exploring Random Vines (RRV) for Motion Planning in Configuration Spaces with Narrow Passages

Adnan Tahirovic[1] and Mina Ferizbegovic[1]

***Abstract*— Classical RRT algorithm is blind to efficiently explore configuration space for expanding the tree through a narrow passage when solving a motion planning (MP) problem. Although there have been several attempts to deal with narrow passages which are based on a wide spectrum of assumptions and configuration setups, we solve this problem in rather general way. We use dominant eigenvectors of the configuration sets formed by properly sampling the space around the nearest node, to efficiently expand the tree around the obstacles and through narrow passages. Unlike classical RRT, our algorithm is aware of having the tree nodes in front of a narrow passage and in a narrow passage, which enables a proper tree expansion in a vine-like manner. A thorough comparison with RRT, RRT-connect, and DDRRT algorithm is provided by solving three different difficult MP problems. The results suggest a significant superiority the proposed Rapidly-exploring Random Vines (RRV) algorithm might have in configuration spaces with narrow passages.**

I. Introduction

The appearance of sampling based motion planners, such as the Probabilistic Road Map (PRM) [1] and the Rapidly-exploring Random Tree (RRT) [2], [3] has open a new perspectives in the MP community. The MP problem has been set in the configuration space in such a way to avoid the necessity of having obstacles and robot geometry explicitly included within the problem. This was possible because a sample-based planner explores the configuration space by means of samples spread across this space and by means of collision-free algorithm that is now needed to check whether two samples can be connected or not. In particular, the RRT algorithm incrementally grows the tree of connections starting from an initial configurations towards a goal configuration. Such an approach turned out to be an efficient way in comparison to all other MP algorithms in solving MP problems, even for highly dimensional spaces. However, the main bottleneck of the RRT planning algorithm, is the presence of a narrow passage in the configuration space in case when the part of any feasible solution of a MP problem must be within this passage.

There exist at least three questions which are useful and important to be answered when one tries to address planning problem in difficult areas by using sample-based planners, 'How could a narrow passage be efficiently found?', 'Once a narrow passage is found, could the planner be aware of it, and would it be possible to expand the tree to enter easily into the passage?', and 'Once a tree node is in the passage, how to expand the tree to pass quickly through it?'. The conventional RRT algorithm is ignorant to these questions since it blindly explores difficult areas in exactly the same way that free areas are explored. However, there is a variety of prior works, some of which will be described in Section II, that address the problem of difficult areas ensuring only partial answers to these questions.

[1] Adnan Tahirovic and Mina Ferizbegovic are with the Faculty of Electrical Engineering, Department for Automatic Control and Electronics, University of Sarajevo, 71000 Sarajevo, Bosnia and Herzegovina `atahirovic@etf.unsa.ba`, `mferizbegovic@etf.unsa.ba`

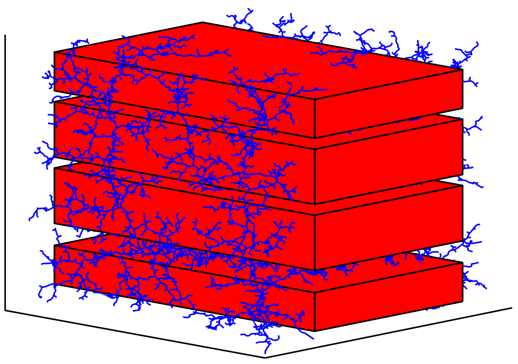

Fig. 1. An illustration in 3D space of how the RRV might circumnavigate an obstacle and expand the vine trough passages in a real-vine manner

In this paper, we propose a sample-based algorithm (Rapidly-exploring Random Vines - RRV), which is continuously capable of providing the answers to the aforementioned questions during the vine (tree) expansion. By being aware of having the vine nodes in front of a narrow passage and inside a narrow passage, enables an efficient vine expansion to solve the MP problem rather quickly, in a real vine-like manner. First, the RRV algorithm is capable of expanding the vine along the obstacles in configuration space. As seen from Fig. 1, the proposed approach tends to circumnavigate obstacles in a similar manner the real vine climbs the objects using the tendrils for support. Being able to circumnavigate the obstacles is an important feature for an efficient discovery of narrow passages. Second, the RRV is capable of understanding when it has a node that is close to a narrow passage, which makes it much easier to expand the vine into that passage. Third, our algorithm is aware of having a node inside a narrow passage, which significantly helps the vine to grow along the passage, much quicker than it would be possible with a classical RRT algorithm.

In Section II, we provide a relevant work to this specific

MP problem. In Section III, we define the terminology used within the paper, then we include some different configuration setups which vine might face, and we explain how the RRV deals with all of them. Pseudo-code is also provided in Section III. Section IV explains simulation setup and presents the results. Section V additionally discusses the choice of parameters and some other specifics, while Section VI concludes the paper.

II. Relevant work for dealing with narrow passages

There exist plenty of sample-based algorithms that are able to provide some of the answers to the questions mentioned in Section I. We separate those algorithms into four groups, those that use workspace information to guide the sampling, those that are based on volumes in configuration space, those that separate regions in configuration space, and the retraction-based algorithms that are capable of extending the trees along the obstacles in configuration space.

The representative work of the planners that use workspace information to guide the sampling can be found in [4], [5], [6]. One of the first planners appeared in this group [4] constructs the medial axis of the workspace to move the robot close to this axis by appropriate sampling in configuration space. In [5], the authors have tried to detect large collision-free regions in the workspace, to assign the space among these regions as narrow passages, and to adapt the sampling distribution in configuration space. The method works well for short narrow passages located between two large obstacle-free regions. Finding an appropriate sampling distribution of probabilistic MP planners to a particular problem class by using selected features, reflected from the underlying workspace, has been proposed in [6].

All these planners use some of the features present in the workspace to learn when and how to pay attention on difficult areas in configuration space. Which feature to use, how informative they are in configuration space, and what they are able to guarantee in terms of adapted sampling distribution used in configuration space, is not easy to understand and predict. Moreover, the obtained underlying non-uniform sampling distribution in configuration space might efficiently guide the tree in certain class of problems, but in some other setups they would not. However, these kind of planners, in general, are not aware of their current expansion state in terms of the three questions posed in Section I.

In [7], the authors have proposed another algorithm from the first group of planners to efficiently explore difficult regions. The algorithm steers the direction along which the tree should grow by using the workspace shape of the obstacles. However, by using directions of the obstacle in the workspace requires geometric computation that is expensive, and still gives a blind heuristic to grow the tree in configuration space. This makes it hard to understand how the planner manages to deal with difficult areas in configuration space and to claim that it will be consistent with different MP setups.

The idea of the planners that belong to the second group is based on building collision-free volumes through configuration space or workspace, which are then used to easily find a collision-free path from the initial configuration to the final one. The *bubble-tree* used in [8], [9], builds volumetric tunnels in the workspace by growing a tree of overlapping spheres based on the shortest collision distance. Once a tunnel is found, it is used to guide the sampling in configuration space to find the final path. Since the sampling is based on the tunnels constructed in workspace, this approach does not possesses Voronoi bias which is a basic advantage of classical RRT algorithms, hence they work well when planning in the workspace. On the other hand, *bubbles of free configuration space* [10], which represent collision-free volumes around the given configurations, have been also used to construct volumetric tunnels in the configuration space [11]. Once such *bubbles* are found, they can be used with the RRT algorithm to efficiently explore configuration space with adapted edges used to expand the tree. A recent attempt to reach beyond the bubble boundary in burs-like manner has been given in [12]. However, all bubble-like approaches provide a carefully constructed collision-free space within which the solution is found, but they still blindly explore this space. In [13], the authors form volumetric forbidden regions around each RRT node based on the closest distance w.r.t. to obstacles. These volumes are then used to reject all unuseful samples towards the tree could possibly grow. Although the resulting tree is sparse even in difficult areas, unlike classical RRT, and the number of collision checking calls are significantly reduced (a large number of non-informative samples are being rejected), this planner is not biased toward narrow passages and it sill depends on the probability to take new samples appropriately within these difficult areas.

In Dynamic Domain RRT (DDRRT) [14] each node holds the information about its radius used as a threshold for potential expansion. Namely, the DDDRT finds the nearest neighbor only among those nodes for which the distance from a random sample is less than the node threshold. In case the connection between the random and the nearest node fails, then the radius is set to a designer-based and smaller value of the radius. Such an approach decreases the volume around the node within which the expansion is permitted, and ensures that the frontier nodes, which are close to obstacles, are not frequently selected for the expansion. Although the algorithm is efficient to address the problems with narrow passages, it is highly sensitive to the threshold value.

The representative work of the third group of planners can be taken from [15] and [16]. These planners partition configuration space into regions based on the knowledge obtained from this space. In [15], the authors classify the regions of configuration space based on the entropy of their initial samples in configuration space. The classification is used to further refine the sampling, i.e. encouraging more samples in narrow regions and decreasing number of samples in large-free regions. These regions are then appropriately connected to form a region-based graph to solve multi-query planning problem. In [16], the authors have used machine learning approach to separate the regions in configuration space (e.g., cluttered vs. free, rough vs. smooth). Each region is then assigned with a motion planner which is considered as a suitable choice for that kind of region. Although these kind of planners might be aware of being inside difficult areas (the third question), they require ad-hoc tuning to select how to best define regions. These planners also requires extensive sampling to tune the chosen classification technique and to classify narrow passages and other difficult areas. The proposed RRV algorithm also distinguishes some different configuration regions, but it does not require any prior classification tuning and extensive initial sampling.

Among all other planners, the fourth group of planners have the most in common with the proposed RRV approach. Namely, these algorithms retract all random samples [17], towards which the tree should grow, including those that would be rejected by classical RRT approach. The samples are retracted to guide the tree growing around obstacles in configuration space. Such a feature ensures efficient tree expansion through narrow passages with respect to classical counterparts, which makes these group of planners capable of efficiently finding narrow passages (first question). Unlike the proposed RRV approach, these planners are not aware of having a node in a narrow passage and efficacy of passing through it strongly depends on shape of obstacles in configuration space and random sampling. However, the main disadvantage of the retraction-based planners is in computation of the closest boundary point which requires solving highly complex globally optimum penetration depth problem [18],[19].

The work presented in [20] refers only to the third question from Section I, to solve a narrow passage problem once the tree has a node inside this passage. Although the approach has been based on principal component analysis (PCA) that is also used in RRV, it does not take care of finding and entering a narrow passage. PCA used in [20] is based on the set formed by the sequence of latest added nodes, which is different than the set we use as explained in Section III.

In this paper, we show that the proposed RRV algorithm is capable of providing the answers to the questions from Section I, which in turn helps the planner to efficiently find, enter and quickly pass a narrow passage in a vine-like manner.

III. The proposed RRV motion planner

Although the proposed RRV approach can be considered as a variant of RRT algorithm, we use a bit different name to underline its inherent capability of being able to expand the tree (vine) around configuration obstacles (see Fig. 1). This feature helps the RRV to discover narrow passages more efficiently than the classical RRT algorithm does, and provides quick vine growth through a narrow passage once it is found. Such maneuverability irresistibly reminds on how the real vine supported by the tendrils climbs the standstill objects and finds in-between spaces. In this section, we introduce the terminology used within the paper and explain the underlying RRV concept through different stages at which the classical RRT algorithm struggles to grow the tree in the configurations occupied with narrow passages.

A. Terminology

The configuration space ($\mathcal{C}$) is a space domain of the system configurations q, in which the MP task is defined.

The obstacle region ($\mathcal{O}$) in the configuration space $\mathcal{C}$ is the set of all system configurations q at which the system collides with obstacles.

The collision-free space ($\mathcal{C}_{free}$) is the set of all system configurations q at which the system does not collide with obstacles, $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{O}$.

The Vine ($\mathcal{V} = (V, E)$) is a directed graph used to incrementally find the path between the initial q_{init} and the final configuration q_{goal}, which consists of vine nodes $q_{node} \in V$ and edges (tendrils) $t \in E$ that connect parent nodes with child nodes.

The tendril local domain set ($Q^t(q_{node}, N, R)$) which consists of N uniformly distributed random configurations q_i^t ($i = 1, .., N$) taken from a hypersphere of radius R and centered at the node q_{node}. This set will be used for understanding the configuration space in vicinity of q_{node} once the algorithm discovers that this node is close to an obstacle, and as a local domain in which a new configuration and its corresponding tendril is found for the vine expansion.

The tendril local climbing support set ($Q^{t,obst}$), where $Q^{t,obst} = \{\forall q_i^t \mid q_i^t \in \mathcal{O}\}$, that is the set made of all configurations $q_i^t \in Q^t$ which are not collision free, and will be here denoted as $q_i^{t,obst}$. This set will be used to distinguish the three different cases of the vine expansion from the node q_{node}, the node being in front of an obstacle, the node being in front of a narrow passage, and the node being in a narrow passage. However, in the former case, this set additionally supports the expansion of the vine around the obstacles in a climbing manner, which in turn relieves finding a narrow passage.

The PCA-based ellipsoid ($\mathcal{E}_{eig}$) for the given sample set Q of the n^{th} dimensional configuration space is defined by the $(n+1)$-tuple $\{\mu, p_1, p_2, .., p_i, .., p_n\}$, where p_i is the i^{th} principal component (eigenvector) of the sample set Q representing the i^{th} axis of $\mathcal{E}_{eig}$, and μ the mean value of the set Q representing the center of $\mathcal{E}_{eig}$. $\mathcal{E}_{eig}^{t,obst}$ denotes an ellipsoid made from the set $Q^{t,obst}$.

The tendril local set for narrow passages ($Q^{t,free}$), were $Q^{t,free} = \{\forall q_i^t \mid q_i^t \in \mathcal{C}_{free} \cap \mathcal{E}_{eig}^{t,o}\}$, that is the set made of all configurations $q_i^t \in Q^t$ which are collision free and being in the ellipsoid $\mathcal{E}_{eig}^{t,obst}$. We denote these configurations as $q^{t,free}$. $Q^{t,free}$ is not an empty set when there is a node q_i^t in-between two obstacles, so by constructing $\mathcal{E}_{eig}^{t,obst}$ there could be a number of collision-free configurations inside. The set $Q^{t,free}$ will be used for finding the directions in which the vine tendrils could possibly expand from q_{node} once the node is inside a narrow passage.

B. Vine in free space

Similarly to the RRT algorithm, after a random configuration $q_{rand} \in \mathcal{C}$ is generated, the RRV also finds the nearest

Algorithm 1 EXTEND(q_{near}, q_{rand})

```
1:  if EXPANSIONOK($q_{near}$, $q_{rand}$) then
2:      $q_{new} \leftarrow$ RRTEXT($q_{near}$, $q_{rand}$)
3:      return $q_{new}$
4:  else
5:      $Q^t \leftarrow$TENDRILDOMAINSET($q_{near}$,$N$,$R$)
6:      $Q^{t,obst} \leftarrow$TENDRILCLIMBINGSUPPORTSET($Q^t$)
7:      $\mathcal{E}_{eig}^{t,obst} \leftarrow$PCABASEDELLIPSOID($q_{near}$, $Q^{t,obst}$)
8:      $Q^{t,free} \leftarrow$TENDRILSETFORNRWPSG($Q^t$,$\mathcal{E}_{eig}^{t,obst}$)
9:      if QFREEEMPTYSET($Q^{t,free}$) then
10:         $q_{new} \leftarrow$ CNVXOBSTEXT($q_{near}$, $q_{rand}$, $\mathcal{E}_{eig}^{t,obst}$)
11:         return $q_{new}$
12:     else
13:         if QNEARNOTINELLIPSOID($q_{near}$, $\mathcal{E}_{eig}^{t,obst}$) then
14:             $\mathbf{q}_{new} \leftarrow$ENTREXT($q_{near}$,$q_{rand}$,$Q^{t,free}$,$\mathcal{E}_{eig}^{t,obst}$)
15:             return $\mathbf{q}_{new} = [q_{new}^1, q_{new}^2]$
16:         else
17:             $\mathbf{q}_{new} \leftarrow$NRWPSGEXT($q_{near}$)
18:             return $\mathbf{q}_{new} = [q_{new}^1, q_{new}^2, ..., q_{new}^M]$
19:         end if
20:     end if
21: end if
```

node $q_{near} \in \mathcal{V}$ with respect to selected distance metrics. Without loss of generality, we use the Euclidian distance in our paper, although other distance metrics can be used as well. Then, the RRV tries to connect q_{near} with q_{rand} for a fixed expansion step. For the sake of brevity, we only show EXTEND() function in the paper. In case the expansion is successful, which means the obtained tendril is collision-free, the terminal tendril configuration state q_{new} is added to $\mathcal{V}$ (Line 3 in Algorithm 1).

Since the ultimate goal of this paper was to develop a motion planner capable of dealing with the configuration spaces with narrow passages, it would be ideal that the RRV algorithm runs equally as the RRT in the configuration spaces without narrow passages. However, we have noticed that the RRV algorithm behaved a bit poorer than the RRT in the configuration spaces with many obstacles but without narrow passages, where the RRT does not usually struggle to find the final path to the goal configuration.

For this reason, we suggest using the RRV algorithm once the system designer has a priori knowledge that the configuration space includes narrow passages, or using the RRV algorithm to support the RRT expansion immediately after it becomes aware that a solution would be difficult to find. The latter case requires an indicator showing that the classical RRT algorithm struggles to find the solution. This indicator can be triggered either by approaching a maximum threshold number of total nodes in the tree, or by measuring the distance between the newly added nodes and the previous ones. Smaller the average of such successively computed distances, the likely the RRT struggles to find the solution.

C. Vine in front of a convex obstacle

In case the expansion of the vine from q_{near} towards q_{random} was not successful (Fig.2-right), the RRV does not reject q_{random} as in classical RRT algorithm, but performs the PCA computation to find *the PCA-based ellipsoid* $\mathcal{E}_{eig}^{t,obst}$. To do so, the RRV algorithm (Line 5) firstly generates random configurations to get $Q^t(q_{near}, N, R)$ (green and red dots in Fig.2-left), picks only non-collision-free configurations $q_i^{t,obst}$ (red dots) to form *the tendril local climbing support set* $Q^{t,obst}$ (Line 6), and then conducts the PCA over this set to find its eigenvectors to construct *the PCA-based ellipsoid* $\mathcal{E}_{eig}^{t,obst}$ (Line 7), which is shown in Fig. 2-right over the obstacle.

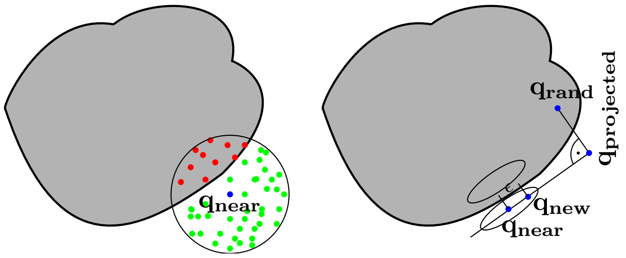

Fig. 2. A vine node q_{near} is in front of a convex obstacle and is not able to expand towards q_{rand}. Random configurations taken around q_{near} are depicted with green and red dots over which a PCA is performed to get the hyperplane (line, in this case) onto which q_{rand} is projected to $q_{projected}$. q_{near} expands then towards $q_{projected}$ to add q_{new} in $\mathcal{V}$.

Secondly, the RRV algorithms checks whether there exist any configurations $q^{t,free}$ (no green dots in $\mathcal{E}_{eig}^{t,obst}$), that is, whether *the tendril local set for narrow passages* $Q^{t,free}$ (Line 8) is an empty set (Line 9). In case $Q^{t,free}$ is an empty set, then the RRV algorithm concludes that q_{near} is in front of a convex obstacle and executes function in Line 10. Otherwise, the algorithm further checks whether q_{near} is in front of or inside a narrow passage which will be explained in separate subsections.

Thirdly, once the RRV has detected the case of being in front of a convex obstacle (function in Line 10), q_{rand} is projected onto a hyperplane which is constructed by the $(n-1)$ most significant eigenvectors. This hyperplane passes through q_{near} and it is orthogonal on the least significant eigenvector (which actually shows the direction to the obstacle from q_{near}) in the n^{th} configuration space $\mathcal{C}$. This means that by using this hyperplane for a new expansion, the RRV is constrained to not expand towards the obstacle and free to grow in all other $(n-1)$ directions along the hyperplane. Finally, the RRV expands towards the configuration $q_{projected}$ which is the projection of q_{rand} onto the hyperplane. The terminal expansion node q_{new} is then added to the vine $\mathcal{V}$.

This expansion is illustrated in Fig.(2-right) for the two-dimensional configuration space, and where the hyperplane degenerates into the line (there is only one most significant eigenvector to form the hyperplane after the least significant eigenvector has been discarded) on which q_{rand} is projected to $q_{projected}$. The terminal tendril configuration is q_{new}

which is added to $\mathcal{V}$. For a three-dimensional case, q_{rand} will be projected onto a plane (there are two most significant eigenvectors to form the hyperplane after the least significant eigenvector has been discarded). Note that for the three-dimensional example, the vine expansion from q_{near} is much less restrictive than in case of the two-dimensional case since we now have two possible directions (along the plane) for the vine expansion.

Such an RRV expansion ensures that q_{rand}, which would be rejected by the classical RRT algorithm, is exploited such that the vine expansion is conducted to circumnavigate a convex obstacle for a fixed step. In case of having a difficult configuration space, where the classical RRT would reject a significant number of random configurations, the RRV expands (climbs) around the obstacles, which in turn helps it to find a narrow passage much efficiently (see Fig. 1). Unlike real trees which usually might get stuck in front of obstacles, real vines efficiently climb the objects with the support of the tendrils.

D. Vine in front of a narrow passage

When q_{near} is not able to expand towards q_{random}, the RRV finds *the PCA-based ellipsoid* $\mathcal{E}_{eig}^{t,obst}$ (Line 7). Then, the algorithm checks whether *the tendril local set for narrow passages* $Q^{t,free}$ (Line 8) is an empty set (Line 9) (whether there are green dots inside $\mathcal{E}_{eig}^{t,obst}$). If now this set is not empty, the algorithm concludes the node q_{node} is not in front of a convex obstacle, and executes Lines 13-20. Figs. 3-left and Fig.4-left depict the two different cases when this would be possible to happen, when q_{near} is in front of a narrow passage, and q_{near} is inside a narrow passage. In these figures, the green dots from $\mathcal{E}_{eig}^{t,obst}$ are now marked with black circles to distinguish them from other green dots.

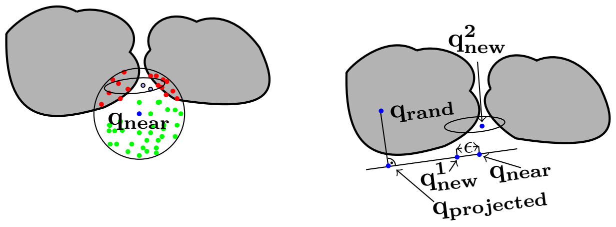

Fig. 3. A vine node q_{near} is in front of a narrow passage and is not able to expand towards q_{rand}. After PCA has been conducted, the vine grows in two directions, q^1_{new} and q^2_{new}. The first one is the same as for convex obstacles, while the second one is for entering the passage.

To further understand whether q_{near} is in front of a narrow passage or already inside the passage, it is important to check whether q_{near} is inside *the PCA-based ellipsoid* $\mathcal{E}_{eig}^{t,obst}$, $q_{near} \in \mathcal{E}_{eig}^{t,o}$ (Line 13). If not, we have an indication that q_{near} might be in front of a narrow passage (Fig.3) (execution of Line 14), otherwise it might be already inside this passage (Fig.4) (execution of Line 17).

We note here that it might also happen that $q_{near} \in \mathcal{E}_{eig}^{t,obst}$ (e.g., when q_{near} is in front of a nonconvex obstacle). Since the case when q_{near} is in front of a narrow passage is the one we deal with, we illustrate the expansion of the RRV algorithm in Fig. 3-right for a two-dimensional problem. However, the RRV algorithm is set to expand with two tendrils, q^1_{new} and q^2_{new}. Namely, the first tendril expands from q_{near} to q^1_{new} in the same way the vine is in front of a convex obstacle by projecting q_{rand} onto the hyperplane as explained in Subsection III-C (Fig.3-right). This expansion preserves the previous behavior of the RRV in case we have wrongly concluded the vine was in front of a narrow passage. The second tendril expands towards q^2_{new} to ensure the RRV enters the narrow passage if it exists, where q^2_{new} is the mean value of *the tendril local set for narrow passages*, $\mu^{free} = \mu(Q^{t,free})$.

Although there would be many ways to connect q_{near} and q^2_{new}, we first try to connect them directly, and if this would not be possible, we then use the classical RRT algorithm (small RRT) for a fixed number of iterations, based on a much smaller expansion step than the one initially selected for RRV. However, if the small RRT also fails, then the vine would wait either the next small RRT-like attempt, or the appropriate random samples to grow in the same way the classical RRT-based tree does.

E. Vine inside a narrow passage

When the RRV is aware that q_{near} is not possible to expand towards q_{random}, $Q^{t,free}$ is not an empty set (Line 9), and $q_{near} \in \mathcal{E}_{eig}^{t,obst}$ as well (Line 13), it concludes that the node q_{near} might be in a narrow passage (Fig. 4) and executes the function from Line 17.

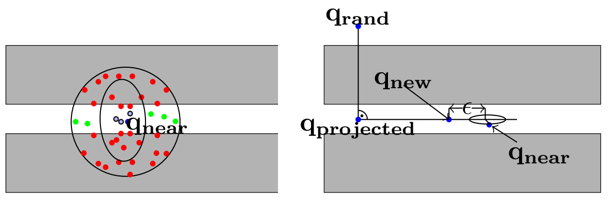

Fig. 4. A vine node q_{near} is in a narrow passage and is not able to expand towards q_{rand}. After PCA has been conducted, the vine grows along the passage.

We note here that it might also happen that q_{near} is actually in a cluttered area. Although we agree that there would be possible to distinguish this case from being in a narrow passage by using the information the configuration samples Q^t might provide, we have not put the focus on the cluttered areas, so the RRV could find its way through this area the same way the RRT algorithm does, that is by waiting appropriate sample configurations towards which the vine would be successfully expanded.

Once q_{near} is inside a narrow passage (execution of Line 17), then it performs the PCA over *the tendril local set for narrow passages* $Q^{t,free}$ to find its principal components. This can be explained as finding possible directions of the free space inside a narrow passage along which the vine would be possible to grow. By using the ordered eigenvalues, the RRV designer can set the threshold value under which the contribution of a particular eigenvector can be considered insignificant, hence it can be discarded.

In case we have only one dominant component (e.g., the total variance of the samples along this component is much larger than the accumulated variance along all other components), this represents the most difficult 'narrow passage' case when we have only one direction left to expand the vine. For a 3D space, this can be understood as a tube-like passage. However, a narrow passage might have more than one dominant principal component. The configuration setup shown in Fig. 1 can be used to illustrated such a case. When q_{near} is inside a narrow passage (between two blocks) the vine can grow in a plane defined with the two coordinate axis (two major principal components), which actually define the plane in-between two obstacle blocks.

The last part of the algorithm covers the designer decision on whether, in case there is only one possible direction to grow the vine within a narrow passage, to force growing the tendrils (by several successive iterations) and for how long (the number of successive iterations).

The vine expansion inside a narrow passage is based on the successive straight expansions, where each of those expansions are constructed by connecting the two lastly added nodes. In case the current expansion is no longer collision free (this might frequently happen in highly curved narrow passages), then the new PCA is conducted around the lastly added node, to find the most significant eigenvector which shows the next free direction along the narrow passage. In the proposed algorithm, we force the vine to grow inside a narrow passage until a certain number of nodes are added to the vine $\mathcal{V}$. However, we argue here that it would be possible to find a way under which the vine would be aware of reaching the end of the passage. In that case, the RRV would be capable of passing a narrow passage in only one set of successive iterations, once the entrance of passage is found. For instance, we could check the distance to the closest obstacle for each newly added vine node in the passage. Getting the distance value larger than a threshold value can be used as an indicator of a successful pass.

IV. Simulation results

In this section, we compare the proposed RRV motion planner with the RRT, RRT-Connect and Dynamic Domain RRT (DDRRT) algorithms. The comparison has been conducted by the three different planning setups, (a) moving a point robot through a *bug trap* (Fig. 5), (b) moving a 3-link planar arm in a tight workspace, and (c) moving a 3D rigid body through a long narrow passage. In all these setups, one section of any feasible path must be inside a narrow passage to solve the problem.

To compare these MP algorithms, we count the *total number of nodes*, *number of rejected points*, *number of collision checking calls*, and *execution time* needed to solve a query. For the RRV algorithm, we also include the *total number of PCA computations*, and stress that each PCA computation requires N *one-configuration* collision detection calls, which are not counted in classical collision checking measure presented for the RRV. However, to get an idea how much all collision checking calls influence the execution time, one must accumulate these two measures (classical collision checking and N one-configuration checking calls, for each PCA run).

All algorithms have been written in C++, the nearest neighbor search has been coded by the support of FLANN library [21], while the collision detection procedure was based on the FCL library [22]. All simulations are conducted on a PC with Intel Core i7-4500U 1.8 GHz and 8 GB of RAM. All algorithms have been run 30 times for each of the three MP problems to collect the relevant statistics. The expansion step for all algorithms and problems has been set to $\epsilon = 0.1$. To get the set of random configurations $Q^t(q_{node}, N, R)$, we have used a fixed $R = 5\epsilon$ (ϵ is the expansion step), while the number of configurations N was different for each scenario, and will be given later.

A modified *bug trap* MP problem (Fig. 5) is solved by a circle-like robot with a radius, $r = 0.1$. In the RRV algorithm, we set $N = 200$. One realization of all four planners has been shown in Fig. 5 to get an idea how the RRV algorithm might generate a superior vine sparsity (Fig. 5 d) w.r.t. the trees generated by all other algorithms (Figs. 5 a-c). In terms of the statistics related to *total number of nodes* shown in Table I, the mean value in RRV is about 50 times better than in classical RRT, 22 times better than in RRT-connect, and 9 times better than in DDRRT. Moreover, even the best scenario in RRT-connect was not better than the worse one in RRV, which is rather superior. However, the best DDRRT case was somewhat comparable to the RRV results, albeit the worse case shows how much DDR results can be inconsistent by means of variance of this statistics.

Number of rejected points accumulated with *total number of nodes* shows how much random points in total were needed to solve the problem. In terms of mean value statistics, RRV is now around 53 times more efficient than RRT, 43 times more efficient than RRT-connect and 27 times efficient than DDRRT algorithm.

		RRT	**RRV**	**RRT-Connect**	**DDRRT**
Nodes number	Mean	144497.92	2817.40	62548.43	26561.13
	Median	89632.00	2457.50	59778.00	7953.00
	Min	18835.00	535.00	11073.00	2292.00
	Max	287745.00	6830.00	154602.00	160860.00
Number of rejected points	Mean	364434.50	5168.43	281276.43	112163.40
	Median	251784.00	3951.50	268701.00	32915.00
	Min	49930.00	350.00	48693.00	9808.00
	Max	705187.00	15709.00	695362.00	684646.00
Collision Checking number	Mean	508932.42	7985.83	343824.86	138724.53
	Median	341416.00	6384.50	328479.00	40868.00
	Min	68765.00	885.00	59766.00	12100.00
	Max	990117.00	22539.00	849964.00	845506.00
PCA computation number	Mean	0	76.07	0	0
	Median	0	76.50	0	0
	Min	0	25.00	0	0
	Max	0	105.00	0	0
Execution time [s]	Mean	16.04	0.38	6.13	2.49
	Median	8.52	0.35	5.61	1.63
	Min	1.57	0.07	0.93	0.19
	Max	37.15	0.80	15.90	16.69

TABLE I

Comparison of simple metrics to solve *bug trap* problem

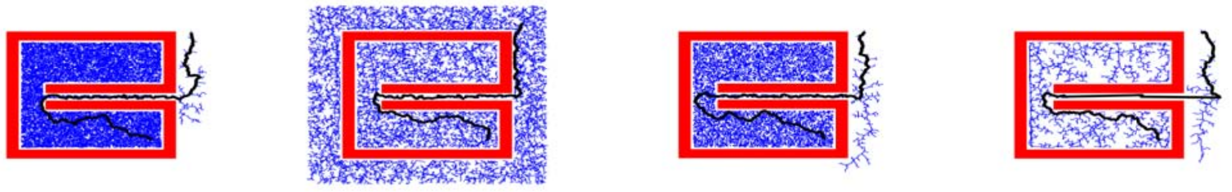

Fig. 5. One realization of MP solution, obtained by all four algorithms for bug trap problem: (a) RRT and (b) RRT - Connect (c) DDRRT (d) RRV

		RRT	**RRV**	**RRT-Connect**	**DDRRT**
Nodes number	Mean	6866.83	1029.77	746.93	1865.80
	Median	6850.00	613.50	687.00	1491.00
	Min	4458.00	295.00	285.00	929.00
	Max	11633.00	10757.00	1473.00	4479.00
Number of rejected points	Mean	56236.50	9252.53	107667.27	17654.40
	Median	56744.50	7204.50	80348.50	12493.50
	Min	31542.00	592.00	11353.00	7976.00
	Max	105967.00	26343.00	270100.00	52541.00
Collision Checking number	Mean	63103.33	11229.30	108414.20	19520.20
	Median	63750.00	8899.00	81051.00	13989.00
	Min	36000.00	1027.00	11638.00	8927.00
	Max	117600.00	30610.00	271242.00	56838.00
PCA computation number	Mean	0	73.67	0	0
	Median	0	81.00	0	0
	Min	0	35.00	0	0
	Max	0	103.00	0	0
Execution time [s]	Mean	1.55	0.53	1.06	0.87
	Median	1.56	0.44	0.85	0.59
	Min	0.86	0.07	0.13	0.41
	Max	2.84	1.30	2.63	2.70

TABLE II

COMPARISON OF SIMPLE METRICS FOR 3 LINK PLANAR ARM

Statistics derived from total number of classical collision checking calls is also superior for RRV algorithm. However, to get a complete understanding of total number of collision checking calls, one must add the total number of PCA computation used in RRV. Each PCA computation additionally required N *one-configuration* checking calls. Since the collision checking run is the bottle neck of all algorithms, the statistics on *execution time* required to solve the problem is the most informative in terms of the overall efficiency of each algorithm. In terms of mean value of execution time, RRV was around 42 more superior than RRT, 16 times more superior than RRT-connect and 6 times more superior than DDRRT. Here we stress an additional important fact reflecting the nature of the used PCA computation. Namely, computation of N *one-configuration* checking algorithm can be parallelized in terms of each of N configurations. This means that, by using more computer cores, one may reduce (or totally eliminate) *the bottle neck* of the RRV algorithm, and make it even more efficient.

The second MP problem settled for comparison, includes a 3-link planar arm tasked to wrap through workspace with a narrow passage to reach the goal position, where we select $N = 200$. It might be seen from the statistics presented in Table II that the results obtained for RRV, RRT-connect and DDRRT are quite comparable in terms of total *number of nodes* and sparsity. To improve superiority of RRV by means of this statistics, designer should select a larger N. However, even for the current case, the statistics on *number of rejected points* and *number of collision checking* calls show again that RRV was more efficient in terms of exploration of state space. We can say that RRV is significantly *less blind* in exploring the space than other algorithms. This fact is reflected in statistics derived for *execution time* which is required to solve the problem. Again, the RRV can use a larger N and be parallelized with up-to N cores, to improve superiority even further.

Table III shows the results obtained while solving the MP problem for a 3D rigid body inside a workspace with a narrow passage. In order to properly explore local space, we set now N to be much larger $N = 2000$. It can be seen from Table III that a larger number of N helps in more efficient explorations which is directly reflected by means of *number of nodes*, *number of rejecting points* and *collision checking* runs. However, this in turns affects the PCA computation time, so the statistics on execution time is not significantly better than in case of RRT-connect and DDRRT algorithms. However, the more computer cores, the more significant improvement we can achieve.

		RRT	**RRV**	**RRT-Connect**	**DDRRT**
Nodes number	Mean	28080.00	280.10	17193.54	93661.06
	Median	15438.50	134.00	11522.00	58335.50
	Min	275.00	83.00	650.00	12361.00
	Max	157596.00	2755.00	67584.00	266574.00
Number of rejected points	Mean	566591.80	6397.27	1602109.54	27170.88
	Median	279486.50	150.50	1155355.50	13326.50
	Min	748.00	49.00	20247.00	1991.00
	Max	3312054.00	164197.00	7235592.00	87342.00
Collision Checking number	Mean	594671.80	6677.37	1619303.08	120831.94
	Median	294925.00	308.00	1170524.00	71662.00
	Min	20686.00	150.00	20976.00	14352.00
	Max	3469650.00	164280.00	7287624.00	353916.00
PCA computation number	Mean	0	155.30	0	0
	Median	0	146.00	0	0
	Min	0	15.00	0	0
	Max	0	468.00	0	0
Execution time [s]	Mean	26.58	7.92	12.37	18.01
	Median	12.32	2.60	8.65	10.12
	Min	1.13	0.21	0.19	1.77
	Max	160.29	101.31	54.00	58.69

TABLE III

COMPARISON OF SIMPLE METRICS FOR 3D RIGID BODY

V. DISCUSSION

The RRV preserves *probabilistic completeness* from the RRT algorithm. When the vine is in *free space*, it behaves as RRT. When the vine is *in front of an obstacle*, it exploits all random samples and adds more informative nodes than RRT would be able to, so this modification also does not affect completeness. When the vine is *in front of a narrow passage*, it uses classical RRT algorithm with a smaller expansion step to enter the passage, performing only a limited number of iterations. If such limitation would not provide the solution, the vine would penetrate the passage in the same manner as the RRT, by waiting for an appropriate random sample. When vine is *in a narrow passage*, it grows through the passage by a limited number of successive iterations, which cannot degrade its completeness either. In all these vine-special cases, we actually help the RRT algorithm to expand more efficiently by adding some additional and more informative nodes which the RRT would not be able to find so quickly.

R and N are important *design parameters* in RRV algorithm. Although we have not put focus on understanding the selection of R yet, we understand that a larger value of R would in turn provide less information on local space around q_{rand}, which is also true even for a significant small value of R. Moreover, it is obvious that the value must be selected to deal with geometry of a narrow passage, to ensure the PCA used within RRV algorithm does help in dealing with such spaces. N should be set to sufficiently cover the sphere around q_{rand} defined by radius R, so its proper choice depends on N. Larger the value, the more information from the local space around q_{rand} would be possible to extract, and the more efficient space exploration secured. On the other side, the more intensive computations would be encountered. However, we stress that the possibility *to parallelized N one-configuration collision checking* called within the PCA-based computations is a rather important inherent feature of RRV algorithm. It practically enables the use of any arbitrarily large N aiming to increase efficiency of space explorations, whose influence on execution time would be then possible to reduce, or even eliminate.

VI. Conclusion

This paper addresses the MP problem in configuration spaces with narrow passages. To deal with such spaces, the proposed algorithm samples the configuration space around the nearest node each time the vine is not possible to grow in RRT manner. The dominant eigenvectors computed over the set formed by such samples helps in expanding the vine both around obstacles and through narrow passages.

The paper explains the RRV algorithm and provides relevant pseudo-codes sufficient to reproduce the results presented in the paper. The algorithm covers all different cases the vine might encounter during the expansion, including vine in free space, vine in front of a convex obstacle, vine in front of a narrow passage, and vine inside narrow passage. We also explain why the RRV deals well in all these cases.

We compare our algorithm with RRT, RRT-connect and DDRRT algorithms by solving three MP problems for which a section of any feasible solution lies inside a narrow passage. We show how the RRV is significantly efficient in exploring the space and solving the problem. We also provide a proper discussion to cover the choice of some design parameters and to provoke some future work.

References

[1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[2] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep. 98-11, Oct. 1998.

[3] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[4] C. Holleman and L. E. Kavraki, "A framework for using the workspace medial axis in prm planners," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 1408–1413.

[5] J. P. Van den Berg and M. H. Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners," *The International Journal of Robotics Research*, vol. 24, no. 12, pp. 1055–1071, 2005.

[6] M. Zucker, J. Kuffner, and J. A. Bagnell, "Adaptive workspace biasing for sampling-based planners," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 3757–3762.

[7] X. Tang, J.-M. Lien, N. Amato *et al.*, "An obstacle-based rapidly-exploring random tree," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 895–900.

[8] O. Brock and L. E. Kavraki, "Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 2. IEEE, 2001, pp. 1469–1474.

[9] M. Rickert, O. Brock, and A. Knoll, "Balancing exploration and exploitation in motion planning," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 2812–2817.

[10] S. Quinlan, "Real-time modification of collision-free paths," Ph.D. dissertation, Stanford University, 1994.

[11] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. IEEE, 1993, pp. 802–807.

[12] B. Lacevic, D. Osmankovic, and A. Ademovic, "Burs of free c-space: a novel structure for path planning," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 70–76.

[13] A. Shkolnik and R. Tedrake, "Sample-based planning with volumes in configuration space," *arXiv preprint arXiv:1109.3145*, 2011.

[14] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle, "Dynamic-domain rrts: Efficient exploration by controlling the sampling domain," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005, April 18-22, 2005, Barcelona, Spain*, 2005, pp. 3856–3861.

[15] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato, "Resampl: A region-sensitive adaptive motion planner," in *Algorithmic Foundation of Robotics VII*. Springer, 2008, pp. 285–300.

[16] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato, "A machine learning approach for feature-sensitive motion planning," in *Algorithmic Foundations of Robotics VI*. Springer, 2004, pp. 361–376.

[17] L. Zhang and D. Manocha, "An efficient retraction-based rrt planner," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 3743–3750.

[18] L. Zhang, Y. J. Kim, and D. Manocha, "A fast and practical algorithm for generalized penetration depth computation." in *Robotics: science and systems*, 2007.

[19] S. Redon and M. C. Lin, "Practical local planning in the contact space," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 4200–4205.

[20] S. Dalibard and J.-P. Laumond, "Linear dimensionality reduction in random motion planning," *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1461–1476, 2011.

[21] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application VISSAPP'09)*. INSTICC Press, 2009, pp. 331–340.

[22] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," 1994.