Robotics and Autonomous Systems 68 (2015) 1–11

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments

Ahmed Hussain Qureshi *, Yasar Ayaz

Robotics And Intelligent Systems Engineering (RISE) Lab, Department of Robotics and Artificial Intelligence, School of Mechanical And Manufacturing Engineering (SMME), National University of Sciences And Technology (NUST), H-12 Campus, Islamabad, 44000, Pakistan

HIGHLIGHTS

- Intelligent sample insertion.
- Non-greedy trees connection.
- Improves the convergence rate.

ARTICLE INFO

Article history:
Received 8 September 2014
Received in revised form
6 January 2015
Accepted 16 February 2015
Available online 23 February 2015

Keywords:
Motion planning
Sampling-based algorithms
RRT
RRT*
Optimal path planning
Bidirectional trees

ABSTRACT

The sampling-based motion planning algorithm known as Rapidly-exploring Random Trees (RRT) has gained the attention of many researchers due to their computational efficiency and effectiveness. Recently, a variant of RRT called RRT* has been proposed that ensures asymptotic optimality. Subsequently its bidirectional version has also been introduced in the literature known as Bidirectional-RRT* (B-RRT*). We introduce a new variant called Intelligent Bidirectional-RRT* (IB-RRT*) which is an improved variant of the optimal RRT* and bidirectional version of RRT* (B-RRT*) algorithms and is specially designed for complex cluttered environments. IB-RRT* utilizes the bidirectional trees approach and introduces intelligent sample insertion heuristic for fast convergence to the optimal path solution using uniform sampling heuristics. The proposed algorithm is evaluated theoretically and experimental results are presented that compares IB-RRT* with RRT* and B-RRT*. Moreover, experimental results demonstrate the superior efficiency of IB-RRT* in comparison with RRT* and B-RRT in complex cluttered environments.

1. Introduction

Motion planning is a well-known problem in robotics [1]. It can be defined as the process of finding a collision-free path for a robot from its initial to goal point while avoiding collisions with any static obstacles or other agents present in its environment. Although motion planning is not the only fundamental problem of robotics, perhaps it has gained popularity among researchers due to widespread applications such as in robotics [2], assembly maintenance [3], computer animation [4], computer-aided surgery [5], manufacturing [6], and many other aspects of daily life.

The journey of finding solution to motion planning problems started with complete planning algorithms that comprised of deterministic path planning approach. Complete motion planning algorithms [7,8] are those algorithms that converges to a path solution, if one exists, in finite time. These algorithms are proven to be computationally inefficient [9] in most of the practical motion planning problems [10]. Resolution complete algorithms were then introduced that require fine tuning of resolution parameters for providing the motion planning solution, if one exists, in a finite time period. Artificial Potential Fields (APF) [11] is a well-known resolution complete algorithm. However, APF suffers from the problem of local minima [12] and does not perform well in the environment with narrow passages. Hence, the search for an efficient solution to the problem continued and the idea of exact roadmaps was introduced in the literature which relies on the discretization of the given search space. This discretization of search space makes the algorithm computationally expensive for higher dimensional spaces, that is why the application of such algorithms like Cell Decomposition methods [13,14], Delaunay Triangulations [15] and Dynamic Graph Search methods [16,17] are limited to low dimensional spaces only [18]. Moreover the algorithms that combine the set of allowed motions with the graph search methods thus generating state lattices, such as in [19–21], also suffered from the undesirable effects of discretization.

* Corresponding author.
E-mail address: ahqureshi@smme.nust.edu.pk (A.H. Qureshi).

http://dx.doi.org/10.1016/j.robot.2015.02.007
0921-8890/

Hence to solve the higher dimensional planning problems, the sampling-based algorithms were introduced [18]; the main advantage of sampling-based algorithms as compared to other state-of-the-art algorithms is avoidance of explicit construction of obstacle configuration space. These algorithms ensure *probabilistic completeness* which implies that as the number of iterations increases to infinity, the probability of finding a solution, if one exists, approaches one. The sampling-based algorithms have proven to be computationally efficient [9] solution to motion planning problems. Arguably, the most well-known sampling-based algorithms include Probabilistic Road Maps (PRM) [22,23] and Rapidly exploring Random Trees (RRT) [24]. However, PRMs tend to be inefficient when obstacle geometry is not known beforehand [10]. Therefore, in order to derive efficient solutions for motion planning in the practical world, the Rapidly-exploring Random Trees (RRT) algorithms [24] have been extensively explored. Various algorithms enhancing original RRT algorithm have been proposed [25–27,10]. These algorithms present a solution regardless of whether specific geometry of the obstacles is known beforehand or not. One of the most remarkable variant of RRT algorithm is RRT*, an algorithm which guarantees eventual convergence to an optimal path solution [10], unlike the original RRT algorithm. Just like the RRT algorithm, RRT* is able to generate an initial path towards the goal very quickly. It then continues to refine this initial path in successive iterations, eventually returning an optimal or near optimal path towards the goal as the number of iterations approach infinity [28]. This additional guarantee of optimality makes the RRT* algorithm very useful for real -time applications [29]. However, some major constraints still exist in this RRT variant which are presented in this paper. For example:

(i) its slow convergence rate in achieving the optimal solution;
(ii) its significantly large memory requirements due to the large number of iterations utilized to calculate the optimal path; and
(iii) its rejection of samples which may not be directly connectable with the existing nodes in the tree, but may lie closer to the goal region and hence could aid the algorithm in determining an optimal path much faster.

Various heuristics have been introduced, such as [30–33], which perform guided search of the given space instead of pure uniform search (as by RRT and RRT*). Although these biased sampling heuristics make the original RRT* algorithm fast but there is a drawback of computational overload caused by biased sampling. This computational overload limits their application to a limited number of fields [34]. Moreover another disadvantage of deterministic sampling heuristics is that they may interfere with the algorithm characteristics. For example assume a simple case of using goal-biased sampling [31] with bidirectional RRT that alternatively grows two trees. The use of this biased sampling will cause the two trees to always remain in one half of the search space, which is quite undesirable. Hence, to cover the whole search space, a separate sample generator is required for both trees which will cost a significant computational load. Hence there is a need of some better approach that enhances the convergence rate of RRT* for achieving the optimal path solution without affecting the randomization of its sampling heuristic. More recent proposition is the bidirectional version of RRT* known as B-RRT* [35]. B-RRT* presented in [35] is a simple bidirectional implementation of RRT*. B-RRT* uses a slight variation of greedy RRT-Connect heuristic [27] for the connection of two trees. Two directional trees employing greedy connect heuristic for the connection of trees does not ensure *asymptotic optimality* [35]. The B-RRT* uses slight variation of greedy heuristic i.e., the tree under process first searches for the neighbor vertices before making an attempt to connect the trees using RRT-Connect heuristic [27]. This hybrid greedy connection heuristic of B-RRT* slows down its ability to converge to the optimal solution and also makes it computationally expensive. More detailed discussion is provided in the analysis section. This paper introduces a bidirectional variation to the RRT* algorithm, with unique sample insertion and tree connection heuristics that allows fast convergence to the optimal path solution. The proposed Intelligent Bidirectional-RRT* (IB-RRT*) algorithm has been tested for its robustness in both 2-D and 3-D environments and has also been compared with other state-of-art algorithms such as Bidirectional-RRT* [35] and RRT* itself [28]. The rest of the paper is organized as follows. Section 2 addresses the problem definition, Section 3 explains the RRT* algorithm while Section 4 describes the B-RRT* motion planning algorithm in detail. Section 5 presents the proposed Intelligent Bidirectional-RRT* (IB-RRT*). Section 6 presents analysis of the three algorithms under investigation in terms of probabilistic completeness, asymptotic optimality, convergence to the optimal solution and computational complexity. Section 7 provides experimental evidence in support of theoretical results presented in the previous section, whereas Section 8 concludes the paper, also suggesting some future areas of research in this particular domain.

2. Problem definition

Let the given state space be denoted by a set $X \subset \mathbb{R}^n$, where n represents the dimension of the given space i.e., $n \in \mathbb{N}, n \geq 2$. The configuration space is further classified into obstacle and obstacle-free regions denoted by $X_{\text{obs}} \subset X$ and $X_{\text{free}} = X \setminus X_{\text{obs}}$, respectively. $X_{\text{goal}} \subset X_{\text{free}}$ is the goal region. Let $T_{\text{a}} = (V_{\text{a}}, E_{\text{a}}) \subset X_{\text{free}}$ and $T_{\text{b}} = (V_{\text{b}}, E_{\text{b}}) \subset X_{\text{free}}$ represent two growing random trees, where V denotes the nodes and E denotes the edges connecting these nodes. $x^{\text{a}}_{\text{init}} \in X_{\text{free}}$ and $x^{\text{b}}_{\text{init}} \in X_{\text{goal}}$ represent the starting states for T_{a} and T_{b}. The function $\mu()$ computes the Lebesgue measure of any given state space e.g. $\mu(X)$ denotes the Lebesgue measure of the whole state space X. It is also called the n-dimensional volume of any given configuration. This paper only considers Euclidean space and positive Euclidean distance between any two states e.g., $x_1 \in X$ and $x_2 \in X$ is denoted by $\text{d}(x_1, x_2)$. The closed ball region of radius $r \in \mathbb{R}, r > 0$ centered at x is denoted as $\mathfrak{B}_{x,r} := \{x_2 \in X : d(x, x_2) \leq r\}$, where $x \in X$ can be any given configuration state. Let the path connecting any two states $x_1 \in X_{\text{free}}$ and $x_2 \in X_{\text{free}}$ be denoted by $\sigma : [0, s']$, such that $\sigma(0) = x_1$ and $\sigma(s') = x_2$, whereas s' is the positive scalar length of the path. The set of all collision-free paths σ is denoted as $\sum_{\text{free}}$. Given any random state $x \in X_{\text{free}}$, the path function connecting initial state x_{init} and random state x is denoted as $\sigma'_{\text{a}}[0, s_{\text{a}}] \subset X_{\text{free}} | \{\sigma'_{\text{a}}(0) = x_{\text{init}}$ and $\sigma'_{\text{a}}(s_{\text{a}}) = x\}$, while the path function connecting random state x and goal region X_{goal} is denoted as $\sigma'_{\text{b}}[0, s_{\text{b}}] \subset X_{\text{free}} | \{\sigma'_{\text{b}}(0) = x$ and $\sigma'_{\text{b}}(s_{\text{b}}) \in X_{\text{goal}}\}$. The complete, end-to-end path function i.e., the path function from root to the goal is denoted by $\sigma'_{\text{f}}(s) = \sigma'_{\text{a}} | \sigma'_{\text{b}} : [0, s] \in X$, where s represents the scalar length of the end-to-end path. The expression $\sigma'_{\text{a}} | \sigma'_{\text{b}} \in X$ describes the concatenation of the two path functions, σ'_{a} and σ'_{b}. The path function σ_{f} is the end-to-end feasible path in obstacle-free configuration space, i.e., $\sigma_{\text{f}} \in X_{\text{free}}$. The set of all end-to-end collision-free paths is denoted as $\sum_{\text{f}}$ i.e., $\sigma_{\text{f}} \in \sum_{\text{f}}$. The cost function $c(\cdot)$ computes the cost in terms of Euclidean distance.

The following motion planning problems will be considered in the proposed algorithm:

Problem 1 (*Feasible Path Solution*)**.** Find a path $\sigma_{\text{f}} : [0, s]$, if one exists, in obstacle-free space $X_{\text{free}} \subset X$ such that $\sigma_{\text{f}}(0) = x_{\text{init}} \in X_{\text{free}}$ and $\sigma_{\text{f}}(s) \in X_{\text{goal}}$. If no such path exists, report failure.

Problem 2 (*Optimal Path Solution*)**.** Find an optimal path $\sigma^*_{\text{f}} : [0, s]$ connecting x_{init} and X_{goal} in obstacle-free space $X_{\text{free}} \subset X$, such that the cost of the path σ^*_{f} is minimum, i.e., $c(\sigma^*_{\text{f}}) = \{\min_{\sigma_{\text{s}}} c(\sigma_{\text{f}}) : \sigma_{\text{f}} \in \sum_{\text{f}}\}$.

Problem 3 (*Convergence to Optimal Solution*)**.** Find the optimal path $\sigma_f^* : [0, s]$ in obstacle-free space $X_{free} \subset X$ in the least possible time $t \in \mathbb{R}$.

3. RRT* algorithm

This section describes the RRT* algorithm [28]. Algorithm 1 is a slightly modified implementation of RRT*. In this version, improvements were made to the original algorithm in order to enhance the computational efficiency of RRT* by reducing the number of calls to the obstacle-free procedure [29]. Following are some of the processes employed by RRT*:

Algorithm 1: RRT*(x_{init})

1 $V \leftarrow \{x_{init}\}$; $E \leftarrow \emptyset$; $T \leftarrow (V, E)$;
2 **for** $i \leftarrow 0$ ***to*** N **do**
3 $x_{rand} \leftarrow$ Sample(i)
4 $X_{near} \leftarrow$ NearVertices(x_{rand}, T)
5 **if** $X_{near} = \emptyset$ **then**
6 $X_{near} \leftarrow$ NearestVertex(x_{rand}, T)
7 $L_s \leftarrow$ GetSortedList(x_{rand}, X_{near})
8 $x_{min} \leftarrow$ ChooseBestParent(L_s)
9 **if** $x_{min} \neq \emptyset$ **then**
10 $T \leftarrow$ InsertVertex(x_{rand}, x_{min}, T)
11 $T \leftarrow$ RewireVertices(x_{rand}, L_s, E)
12 **return** $T = (V, E)$

Algorithm 2: GetSortedList(x_{rand}, X_{near})

1 $L_s \leftarrow \emptyset$
2 **for** $x' \in X_{near}$ **do**
3 $\sigma' \leftarrow$ Steer(x', x_{rand})
4 $C' \leftarrow c(x_{init}, x') + c(x', x_{rand})$
5 $L_s \leftarrow (x', C', \sigma')$
6 $L_s \leftarrow$ SortList(L_s)
7 **return** L_s

Random sampling: the sample procedure returns an independent and uniformly distributed random sample from the obstacle-free space, i.e., $x_{rand} \in X_{free}$.

Collision check: the procedure ObstracleFree(σ) checks whether the given path $\sigma : [0, 1]$ belongs to X_{free} or not. A true value is reported if $\sigma(s) \in X_{free} \forall s[0, 1]$.

Near vertices: given a sample $x \in X$, the tree $T = (V, E)$ and the ball region $\mathfrak{B}_{x,r}$ of radius r centered at x, the set of near vertices is defined as:

Near$(x, T, r) := \{v \in V : v \in \mathfrak{B}_{x,r}\} \mapsto X_{near} \subseteq V$. More specifically, $X_{near} = \{v \in V : d(x, v) \leq \gamma(logi/i)^{1/n}\}$ where i is the number of vertices, n represents the dimensions and γ is a constant.

Nearest vertex: as its name suggests, this procedure returns the nearest vertex in the tree from any given state $x \in X$. Given the tree $T = (V, E)$, the nearest vertex procedure can be defined as:

$$\text{Nearest}(T, x) := \text{argmin}_{v \in V} d(x, v) \mapsto x_{min}.$$

Getting sorted list: the procedure GetSortedList in Algorithm 2 constructs a tuple and returns it as the list L_s. Each element of this list is a triplet of form $(x', c(\sigma), \sigma') \in L_s$ where $x' \in X_{near}$. The list L_s is sorted in the ascending value of the cost function.

Steering: the steering function utilized in this modified version of RRT* takes two states as an input and returns the straight trajectory connecting those two states. For example, for two given states $x_1 \in X$ and $x_2 \in X$, the path $\sigma : [0, 1]$ will be the path connecting these two states, i.e., $\sigma(0) = x_1$ and $\sigma(1) = x_2$. The steering is done from x_1 to x_2 in small, discrete steps and can be summarized as $\sigma(s') = (1 - s')x_1 + s'x_2; \forall s'[0, 1]$.

Choosing best parent: this procedure is used to search the list L_s for a state, $x_{min} \in L_s$ which provides the shortest, collision-free path σ' from the initial state x_{init} to the random sample x_{rand}. Alternatively, σ' is the shortest collision-free path connecting the initial state x_{init} and the random sample x_{rand} via $x_{min} \in L_s$. Algorithm 3 outlines the implementation of this procedure.

Algorithm 3: ChooseBestParent(L_s)

1 **for** $(x', C', \sigma') \in L_s$ **do**
2 **if** ObstacleFree(σ') **then**
3 **return** x'
4 **return** $\emptyset$

Algorithm 4: RewireVertices(x_{rand}, L_s, E)

1 **for** $(x', C', \sigma') \in L_s$ **do**
2 **if** $\left(c(x_{init}, x_{rand}) + c(x_{rand}, x')\right) < c(x_{init}, x')$ **then**
3 **if** ObstacleFree(σ_{new}) **then**
4 $x_{parent} \leftarrow$ Parent(E, x')
5 $E \leftarrow (E \backslash \{(x_{parent}, x')\}) \cup (\{x_{rand}, x'\})$
6 **return** E

The RRT* algorithm provides asymptotic optimality. In reference to Algorithm 1, the RRT* algorithm after preliminary initialization process starts its iterative process by sampling the random sample x_{rand} from the given configuration space X_{free} (Line 3). After this, the RRT* finds the set of near vertices X_{near} from the tree lying inside the ball region centered at x_{rand}. If the set of near vertices X_{near} computed by NearVertices procedure is empty, then the set X_{near} is filled by the NearestVertex procedure (Line 4–6). The populated set X_{near} is then sorted by the GetSortedList procedure to form a list of form $(x', c(\sigma), \sigma')$, arranged in The ascending order of cost function $c(\sigma)$ (Line 7). The procedure ChooseBestParent iterates over the sorted list L_s (Line 8), returning the best parent vertex $x_{min} \in X_{near}$ through which x_{init} and x_{rand} can be connected in obstacle-free space. Once such a state is located, i.e., the best parent vertex x_{min} is no longer empty, x_{min} is inserted into the tree by making x_{rand} its child and then the rewiring step is executed (Line 9–11). Algorithm 4 presents the pseudocode of the rewiring process. Here, the algorithm examines each vertex $x' \in X_{near}$ lying inside the ball region centered at x_{rand}. If the cost of the path connecting x_{init} and x' through x_{rand} is less than the existing cost of reaching x' and if this path lies in obstacle-free space X_{free} (Algorithm 4 Line 1–3), then x_{rand} is made the parent of x' (Algorithm 4 Line 4–5). If these conditions do not hold true, no change is made to the tree and the algorithm moves on to check the next vertex. This process is iteratively performed for every vertex x' present in the sorted list L_s.

4. B-RRT* algorithm

This section explains the implementation of Bidirectional-RRT*(B-RRT*) [35]. Algorithm 5 outlines the implementation of B-RRT*; extra procedures employed by B-RRT* are explained below while the rest are exactly the same as they were for RRT*.

Extend: given two nodes $x_1, x_2 \in X$, the Extend(x_1, x_2) procedure returns a new node $x_{new} \in \mathbb{R}^n$ such that x_{new} is more closer to x_2 than x_1 in the direction from x_1 to x_2.

Connect: connect heuristic employed by B-RRT* is slight variation of greedy RRT-Connect heuristic [27]. Algorithm 6 outlines the

Algorithm 5: B-RRT*(x_{init},x_{goal})

1 $V \leftarrow \{x_{init}, x_{goal}\}$; $E \leftarrow \emptyset$; $T_a \leftarrow (x_{init}, E)$; $T_b \leftarrow (x_{goal}, E)$;
2 $\sigma_{best} \leftarrow \infty$
3 **for** $i \leftarrow 0$ ***to*** N **do**
4 $x_{rand} \leftarrow$ Sample(i)
5 $x_{nearest} \leftarrow$ NearestVertex(x_{rand}, T_a)
6 $x_{new} \leftarrow$ Extend($x_{nearest}$, x_{rand})
7 $X_{near} \leftarrow$ NearVertices(x_{new}, T_a)
8 $L_s \leftarrow$ GetSortedList(x_{new}, X_{near})
9 $x_{min} \leftarrow$ ChooseBestParent(L_s)
10 **if** $x_{min} \neq \emptyset$ **then**
11 $T \leftarrow$ InsertVertex(x_{new}, x_{min}, T_a)
12 $T \leftarrow$ RewireVertices(x_{new}, L_s, E)
13 $x_{conn} \leftarrow$ NearestVertex(x_{new}, T_b)
14 $\sigma_{new} \leftarrow$ Connect(x_{new}, x_{conn}, T_b)
15 **if** $\sigma_{new} \neq \emptyset$&&$c(\sigma_{new}) < c(\sigma_{best})$ **then**
16 $\sigma_{best} \leftarrow \sigma_{new}$
17 SwapTrees(T_a, T_b)
18 **return** $T_a, T_b = (V, E)$

implementation of Connect heuristic of B-RRT*. Typical RRT* iteration is performed on the input nodes x_1, x_2, where x_1 plays the role of x_{rand} while the set of near vertices is computed from the other tree (Line 1–2). After the computing set of near vertices from tree b, the procedure GetSortedList (explained in the previous section) is executed, and the best vertex is selected from the sorted list such that it provides collision-free low-cost connection between the trees T_a, T_b. Finally, this procedure ends by generating and returning the end-to-end feasible path solution, connecting x_{init} and X_{goal}.

In reference to Algorithm 5, the B-RRT* works in exactly the same manner as the original RRT* algorithm in its initial phases i.e., it starts with sampling of the configuration space X_{free}, then various operations (just like RRT*) are performed on this random sample x_{rand} (Line 4–12). After successful insertion of random sample into the tree under operation (Line 11–12), the algorithm computes nearest vertex x_{conn} from x_{new} in the tree T_b, and then executes the connect procedure for the connection of two trees (Line 13–14). If the attempt to make connection is successful, the collision-free path σ_{new} connecting x_{init} and X_{goal} is returned by the connect function. The cost of this σ_{new} is then compared with the previously computed path σ_{best}. If the cost of σ_{new} is less than the cost of σ_{best}, then σ_{best} is overwritten by σ_{new} (Line 15–16). Finally, the iteration ends by swapping the trees (Line 17) and in the next iteration the same procedure is executed on the other tree.

Algorithm 6: Connect(x_1,x_2,T_b)

1 $x_{new} \leftarrow$ Extend(x_2, x_1)
2 $X_{near} \leftarrow$ NearVertices(x_{new}, T_b)
3 $L_s \leftarrow$ GetSortedList(x_1, X_{near})
4 $x_{min} \leftarrow$ ChooseBestParent(L_s)
5 **if** $x_{min} \neq \emptyset$ **then**
6 $E \leftarrow E \cup ((x_{min}, x_1))$
7 $\sigma_f \leftarrow$ GeneratePath(x_{min}, x_1)
8 **return** σ_f
9 **return** NULL

5. IB-RRT* algorithm

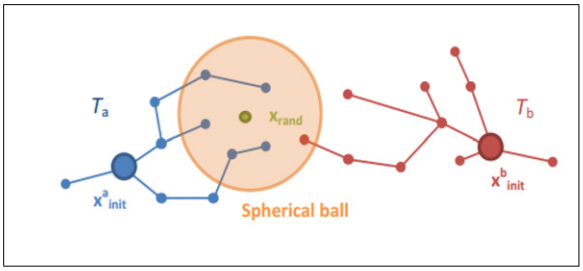
This section presents our proposed IB-RRT* algorithm. IB-RRT* is specifically designed for motion planning in complex cluttered environments where exploration of configuration space is difficult. Let the sets of near vertices from tree T_a and T_b be denoted by X^a_{near} and X^b_{near}, respectively. The path connecting x^a_{init} and x_{rand} is denoted by $\sigma'_a : [0, s_a]$, while the path connecting x^b_{init} and x_{rand} is denoted by $\sigma'_b : [0, s_b]$. Algorithm 7 outlines the implementation of IB-RRT*. In Algorithm 7, the Boolean variable Connection represents the feasibility of connecting the two trees while the Boolean variable flag indicates the best selected tree. IB-RRT* builds the bidirectional trees incrementally. It starts by picking a random sample x_{rand} from the obstacle-free configuration space X_{free} i.e., $x_{rand} \in X_{free}$ (Line 6). It then populates the set of near vertices X^a_{near}, X^b_{near} for both trees using the NearVertices procedure (Line 7). It should be noted that a ball region centered at x_{rand} of radius r is formed and the sets of the near vertices from both trees are computed i.e., $X^a_{near} := \{v \in V_a : v \in \mathfrak{B}_{x_{rand},r}\}$ and $X^b_{near} := \{v \in V_b : v \in \mathfrak{B}_{x_{rand},r}\}$, as shown in Fig. 1(a). X^a_{near} and X^b_{near} now contain near vertices of x_{rand} from trees T_a and T_b, respectively. In the case of both sets of near vertices being found empty, these sets are filled with the closest vertex from their respective trees instead, i.e., the vertex on their respective tree which lies closest to the random sample (Line 8–9). Both sets are then sorted by the GetSortedList procedure (Line 11–12) outlined in Algorithm 2. Once the list is in the ascending order, the random vertex x_{rand} is inserted in the best selected tree (Line 13–19). The procedure BestSelectedTree (explained later in this section) returns the nearest vertex on the best selected tree which is eligible to become parent of the random sample.

Algorithm 7: IB-RRT* (x^a_{init}, x^b_{init})

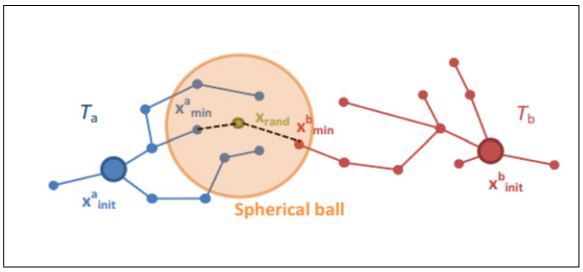
1 $V_a \leftarrow \{x^a_{init}\}$; $E_a \leftarrow \emptyset$; $T_a \leftarrow (V_a, E_a)$
2 $V_b \leftarrow \{x^b_{init}\}$; $E_b \leftarrow \emptyset$; $T_b \leftarrow (V_b, E_b)$
3 $\sigma_f \leftarrow \infty$; $E \leftarrow \emptyset$
4 Connection $\leftarrow$ True
5 **for** $i \leftarrow 0$ ***to*** N **do**
6 $x_{rand} \leftarrow$ Sample(i)
7 $\{X^a_{near}, X^b_{near}\} \leftarrow$ NearVertices(x_{rand}, T_a, T_b)
8 **if** $X^a_{near} = \emptyset$ & & $X^b_{near} = \emptyset$ **then**
9 $\{X^a_{near}, X^b_{near}\} \leftarrow$ NearestVertex(x_{rand}, T_a, T_b)
10 Connection $\leftarrow$ False
11 $L^a_s \leftarrow$ GetSortedList(x_{rand}, X^a_{near})
12 $L^b_s \leftarrow$ GetSortedList(x_{rand}, X^b_{near})
13 $\{x_{min}$, flag, $\sigma_f\} \leftarrow$ GetBestTreeParent(L^a_s, L^b_s, Connection)
14 **if** (flag) **then**
15 $T_a \leftarrow$ InsertVertex(x_{rand}, x_{min}, T_a)
16 $T_a \leftarrow$ RewireVertices(x_{rand}, L_s, E_a)
17 **else**
18 $T_b \leftarrow$ InsertVertex(x_{rand}, x_{min}, T_b)
19 $T_b \leftarrow$ RewireVertices(x_{rand}, L_s, E_b)
20 $E \leftarrow E_a \cup E_b$
21 $V \leftarrow V_a \cup V_b$
22 **return** $(\{T_a, T_b\} = V, E)$

Additional features included in IB-RRT* are explained below. The rest of the procedures employed by our algorithm are the same as those outlined in the previous section.

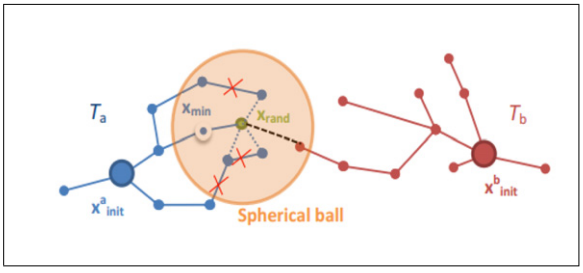
Selecting best tree parent: the operation GetBestTreeParent replaces the ChooseBestParent procedure of the RRT* algorithm. The implementation of this procedure is outlined in Algorithm 8. Initially the best parent vertex from both the trees is calculated, as shown in Fig. 1(b). This process (Line 2–9) is similar to ChooseBestParent procedure outlined in Algorithm 3. The best selected triplets from each tree T_a and T_b are assigned to $\{x^a_{min}, C^a_{min}, \sigma_a\} \in L^a_s$ and $\{x^b_{min}, C^b_{min}, \sigma_b\} \in L^b_s$ respectively. Following this, the GetBestTreeParent procedure selects the best tree from amongst T_a and T_b



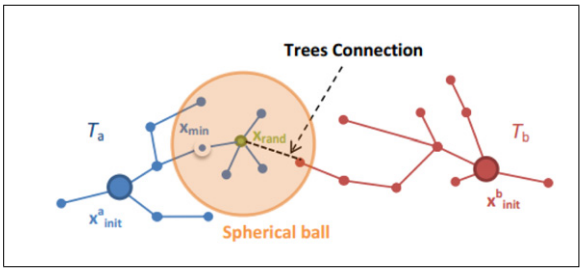
(a) Computing set of near nodes from both trees.



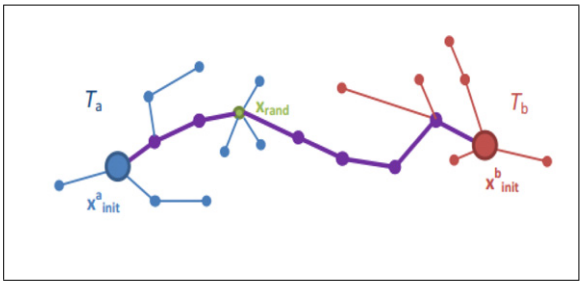
(b) Choosing best parent.



(c) Choosing best parent vertex and rewiring.



(d) Connecting trees.



(e) End-to-end path.

Fig. 1. Intelligent bidirectional trees.

on the basis of costs C^a_{min} and C^b_{min} associated with each best selected triplet. The best selected vertex of the best selected tree, i.e. either x^a_{min} or x^b_{min}, is then assigned to x_{min} for the insertion process. For the scenario depicted in Fig. 1, tree T_a is selected as the best

Algorithm 8: GetBestTreeParent(L^a_s, L^b_s, Connection)

1 flag ← True
2 **for** $(x'_a, C'_a, \sigma'_a) \in L^a_s$ **do**
3 **if** ObstacleFree(σ'_a) **then**
4 $\{x^a_{min}, C^a_{min}, \sigma_a\} \leftarrow \{x'_a, C'_a, \sigma'_a\}$
5 Break
6 **for** $(x'_b, C'_b, \sigma'_b) \in L^b_s$ **do**
7 **if** ObstacleFree(σ'_b) **then**
8 $\{x^b_{min}, C^b_{min}, \sigma_b\} \leftarrow \{x'_b, C'_b, \sigma'_b\}$
9 Break
10 **if** $(C^a_{min} \leq C^b_{min})$ **then**
11 $x_{min} \leftarrow x^a_{min}$
12 **else if** $(C^b_{min} < C^a_{min})$ **then**
13 $x_{min} \leftarrow x^b_{min}$
14 flag ← False
15 **if** Connection **then**
16 $\sigma_f \leftarrow$ ConnectTrees(σ_a, σ_b)
17 **return** $\{x_{min}$, flag, $\sigma_f\}$

Algorithm 9: ConnectTrees(σ_a, σ_b)

1 **if** $c(\sigma_f) < c(\sigma_a|\sigma_b)$ **then**
2 $\sigma_f \leftarrow \sigma_a|\sigma_b$
3 **return** σ_f

tree and therefore x^a_{min} is assigned to x_{min} as shown in Fig. 1(c). The Boolean variable flag indicates which tree has been selected as the best tree for any single iteration (Line 10–14). The algorithm then attempts to connect the bidirectional trees (Line 15) on the basis of the Boolean variable connection. This is explained further on in the paper. The GetBestTreeParent procedure concludes by returning the best vertex x_{min}, the Boolean flag and, if it exists, the final path σ_f connecting the initial state to the goal region.

Bidirectional trees connection: Algorithm 9 gives the pseudocode of the procedure ConnectTrees. Given collision-free paths $\sigma_a : [0, s_a]$ and $\sigma_b : [0, b]$, where $\sigma_a(0) = x^a_{init}$, $\sigma_b(0) = x^b_{init}$ and $\sigma_a(s_a) = \sigma_b(s_b) = x_{rand}$. This procedure updates the end-to-end collision-free path $\sigma_f : [0, s]$ connecting $\sigma_f(0) = x^a_{init}$ and $\sigma_f(s) = x^b_{init}$ if the cost of concatenated paths, $c(\sigma_a|\sigma_b)$, is found to be less than the cost of the existing end-to-end path $c(\sigma_f)$ (Line 1–2). Connection between the trees is only successful if the Boolean variable connection is true. As mentioned in the previous explanation, the occurrence of empty sets for both X^a_{near} and X^b_{near} causes the procedure NearestVertex to be called (Algorithm 7, Line 7–8). The NearestVertex changes the Boolean variable connection to false. Therefore, the Boolean connection is true only when the procedure NearVertices populates both sets. This implies that the two trees are connected if ball of region centered at x_{rand} contains near vertices from both trees T_a and T_b. Hence, unlike the connect heuristic [27], the IB-RRT* is not greedy since the connection is only made inside the ball region as shown in Fig. 1(d). Finally the tree connection generates end-to-end global path, as shown in Fig. 1(e).

6. Analysis

6.1. Probabilistic completeness

In any configuration space, an algorithm is said to be *Probabilistically Complete* if the probability of finding a path solution, if ones exist, approaches one as the number of samples taken from the configuration space reaches infinity. It is known that RRT is a probabilistically complete algorithm, as its optimal variant RRT* [10].

Since our proposed IB-RRT* algorithm performs the random sampling function exactly like the aforementioned algorithms and is merely a bidirectional version of RRT* with intelligent sample insertion, it can be reasonably proffered that it also inherits the probabilistic completeness property of RRT*.

6.2. Asymptotic optimality

It is known that RRT and its variant RRT-Connect do not ensure optimality even if the number of iterations are increased to infinity [10]. However, RRT* is an optimal variant of RRT, ensuring almost-sure convergence to an optimal solution [28]. As explained earlier, IB-RRT* attempts to connect both trees, $T_a = (V_a, E_a)$ and $T_b = (V_b, E_b)$, in every iteration. A random sample x_{rand} is used as a point of connection between the two trees (shown in Fig. 1(d)) if the ball region centered at x_{rand} is found to contain near vertices from both the trees, i.e., $v_a \in V_a : v_a \in \mathfrak{B}_{x_{rand},r}$ and $v_b \in V_b : v_b \in \mathfrak{B}_{x_{rand},r}$. A similar procedure is employed by the RRT* algorithm [28] to connect the random sample with its chosen parent. Since there is no extra connection heuristic required for connection of the two trees and the two trees are generated exactly as the tree generated in the original RRT* algorithm, it can be reasonably proposed that the IB-RRT* algorithm inherits the asymptotic optimality property of RRT*.

6.3. Rapid convergence to optimal path

This section provides proof of IB-RRT*'s rapid convergence to the optimal solution and that this algorithm provides faster convergence rates as compared to state of the art algorithms RRT* and B-RRT*. For simplicity, the following assumptions are made:

Assumption 1 (*Uniform Sampling*)**.** The sampling operation take samples from a configuration space X such that the samples are continuously distributed.

Assumption 2 (*Cluttered Configuration Space*)**.** The configuration space X is cluttered such that the tree initially grows near its initial state x_{init} and then incrementally grows towards the unsearched configuration space.

Assumption 3 (*Uniformity and Additivity of Cost Function*)**.** For a given set of path functions, the cost function must satisfy: $c(\sigma_1) \leq c(\sigma_1|\sigma_2) : c(\sigma_1|\sigma_2) = c(\sigma_1) + c(\sigma_2) \forall \sigma_1, \sigma_2 \in \sum_{free}$.

Assumption 1 ensures that the sampling operation is not biased or goal directed. Biasing the samples for rapid convergence to the optimal solution is computationally inefficient, specifically in higher dimension configuration spaces [10]. Assumption 2 states that the environment contains obstacles that hinder the expansion of the two trees in the configuration space. Finally, Assumption 3 simply asserts that the longer path has a higher cost than the shorter one.

As mentioned in Section 2, $\sum_{free}$ denote the set of all collision-free paths in the tree $T = (V, E)$. Let $\sigma_i, \sigma_i' \in \sum_{free}$ such that σ_i' is closest to σ_i in terms of Euclidean distance function. The following lemma states that any sampling-based algorithm can provide almost-sure convergence to the optimal path solution if distance variation $\|\sigma_i' - \sigma_i\|$ approaches zero as the number of iterations approaches infinity.

Lemma 1 (*[10]*)**.** *A sampling-based algorithm ensures asymptotic optimality, such that*

$$\mathbb{P}\left(\lim_{i\to\infty} \|\sigma_i' - \sigma_i\| = 0;\ \forall \sigma_i, \sigma_i' \in \sum_{free}\right) = 1.$$

With Lemma 1 following corollary immediate.

Corollary 1. *By increasing the number of path variations minimized per iteration, the algorithm can greatly improve its rate of convergence to an optimal solution.*

Given a tree $T = (V, E)$, random configuration state $x \in X_{free}$ and a set of near vertices X_{near} inside a ball region $\mathfrak{B}_{x,r}$ centered at x, the intensity of near vertices around x, denoted by J_x, can be defined as:

$$J_x := \{\text{card}(X_{near})/\mu(\mathfrak{B}_{x,r}) : X_{near} | x = \mathfrak{B}_{x,r} \cap V\}.$$

Regarding the intensity of near vertices Lemma 2 is stated as follows:

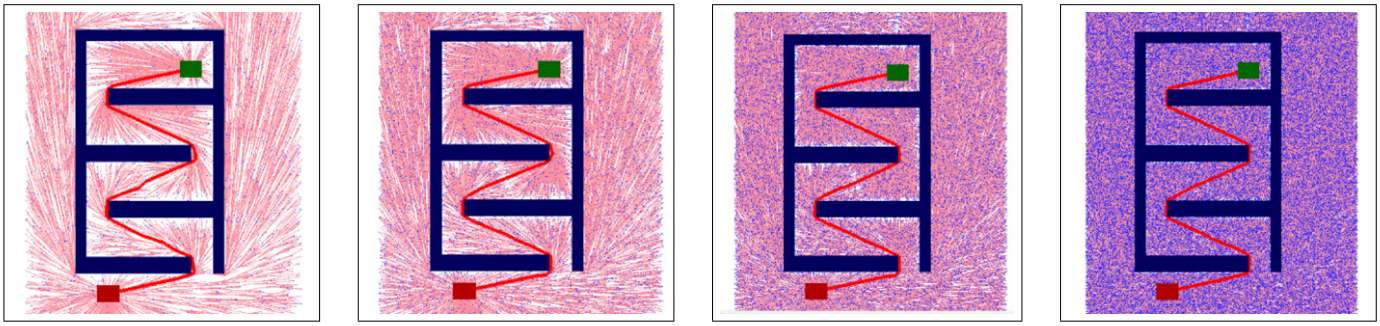
Lemma 2. *If Assumptions 1, 2, 3 hold, then Intensity J_x is higher in the regions closer to the point of generation of the tree.*

Sketch of proof. Let $\epsilon \in \mathbb{R}_+$. X_{free} is obstacle-free configuration space. X_{free} is searched for the set of near neighbors X_{near} that lie inside a ball region $\mathfrak{B}_{x,r}$ of radius $r > 0$ centered at the random state $x \in X_{free}$. Any state $x' \in X_{near}$ can become the parent of x_{rand}, if it provides a lower cost path connecting x_{rand} to x_{init} than the one provided by all other vertices in X_{near}. This implies that $\|x - x'\| = \epsilon$, where $\epsilon < r = \gamma(logi/i)^{1/n}$. This ensures that the growth of the algorithm presented in this paper is incremental as the tree grows in small incremental distances ϵ. For the incremental or wavefront expansion of the trees it is proven that the regions near the point of generation of trees are more dense [24]. Therefore, there is a high probability of having high cardinality of set X_{near}, if the random state lies closer to the point of generation of tree.

Hence, with Corollary 1 and Lemma 2 holds the following theorem stating effectiveness of IB-RRT* is given below.

Theorem 1. *The IB-RRT* algorithm converges to the optimal solution more quickly as compared to RRT* and B-RRT*.*

Sketch of proof. Given a random configuration x_{rand} and minimum cost path functions $\sigma_a[0, s_a] := \{\sigma_a(0) = x^a_{init}, \sigma_a(s_a) = x_{rand}\}$ and $\sigma_b[0, s_b] := \{\sigma_b(0) = x^b_{init}, \sigma_b(s_b) = x_{rand}\}$, then the insertion process of IB-RRT* can be summarized as $\{x_{rand} \in V_a : c(\sigma_a) \leq c(\sigma_b)$ otherwise $x_{rand} \in V_b : c(\sigma_b) < c(\sigma_a)\}$ (Algorithm 8). As the random sample x_{rand} is always inserted into the tree whose initial state is closer to x_{rand}, this ensures that the sample is inserted into a region in the configuration space where the intensity of near vertices J_x is high. Since the rewiring process explained earlier tries to minimize the distance variation $\|\sigma_i' - \sigma_i\|$ between any two closest paths in each tree. This is done by checking viability of the random sample x_{rand} as the parent of each vertex in the set X_{near}. If the cost to reach a particular vertex x' in the near set X_{near} through random sample x_{rand} is lower then the existing cost, then x_{rand} becomes the parent of that particular vertex $x' \in X_{near}$. Hence, IB-RRT* inserts the sample into high intensity regions J_x, maximizing the rewiring process per iteration. This step allows rapid convergence to optimal solution and serves as evidence that IB-RRT* provides better convergence rates than both RRT* and B-RRT* algorithms. Furthermore, trees connection heuristic employed by B-RRT* [35] is partially greedy, similar to the connect heuristic [27]. It has already been proved that if the bidirectional version of RRT* uses purely RRT-Connect heuristic [27] for the connection of two trees, it is no longer asymptotically optimal [35]. This happens because when only the connect heuristic [27] is used, an edge originating from T_a for example, tries to reach the closest vertex on T_b. This implies that near vertices present inside the ball region are never considered for best parent selection. B-RRT* does eventually converge to an optimal solution but the convergence process is slowed down due to its partially greedy characteristic. Nevertheless, compared to RRT*, the B-RRT* has faster convergence rate due to its generation of two trees. However, in comparison to IB-RRT*, it has significantly less convergence rate. This is later on evident from the experimental results as well.



(a) $i = 7988$, $t = 1.8$ s, $C = 95$. (b) $i = 22\,164$, $t = 4.9$ s, $C = 93.9$. (c) $i = 79\,187$, $t = 18.8$ s, $C = 93.4$. (d) $i = 161\,437$, $t = 40$ s, $C = 93.1$.

Fig. 2. IB-RRT* performance in 2-D environment (A).

6.4. Computational complexity

This section compares computational complexities of IB-RRT* with complexities of RRT* and the bidirectional version of RRT. Let $S_i^{\text{RRT}^*}$ and S_i^{BiRRT} denotes the number of processes executed per iteration by RRT* and bidirectional-RRT (BiRRT), respectively. Let S_i^{Ours} denote the number of processes executed by IB-RRT*. Theorems 2 and 3 propose that the running time of all processes executed per iteration by IB-RRT* is a constant times higher than both RRT* and BiRRT.

Theorem 2. *The computational ratio of IB-RRT* and RRT* is such that there exists a constant ϕ_1 i.e.,*

$$\lim_{i\to\infty} \mathbb{E}\left[\frac{S_i^{\text{Ours}}}{S_i^{\text{RRT}^*}}\right] \leq \phi_1.$$

Theorem 3. *The computational ratio of IB-RRT* and BiRRT is such that there exists a constant ϕ_2 i.e.,*

$$\lim_{i\to\infty} \mathbb{E}\left[\frac{S_i^{\text{Ours}}}{S_i^{\text{BiRRT}}}\right] \leq \phi_2.$$

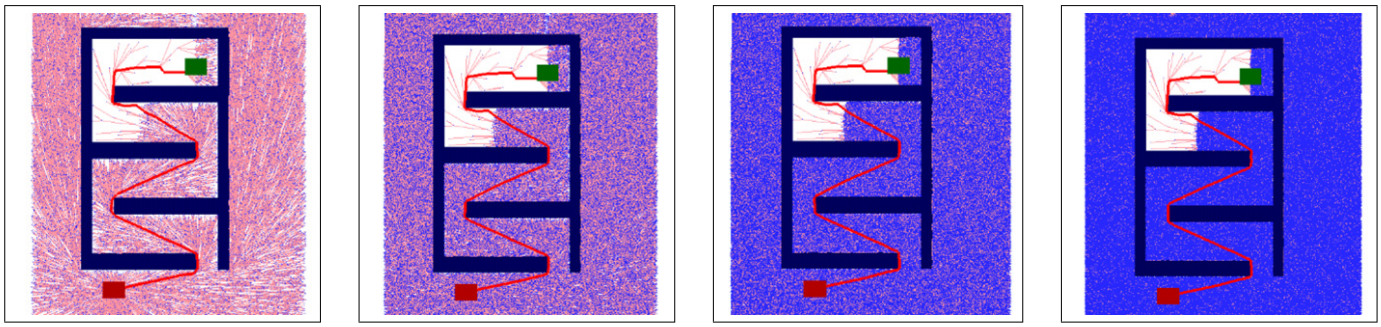
Similar to RRT*, our proposed algorithm calls the procedures Sample and RewireVertices exactly once. The procedure of choosing best parent in RRT* is replaced by FindBestTree in the IB-RRT* algorithm, which also includes the ConnectTrees procedure. As explained earlier, the ConnectTrees function has negligible computational overhead since it merely concatenates two paths. In Intelligent Bidirectional-RRT* (IB-RRT*), for every iteration the procedures NearestVertex and NearVertices are executed for both trees T_a and T_b. It has already been proved that both procedures have to run in *logi* expected time [28]. Furthermore, while IB-RRT* makes its best effort to increase the cardinality of the near vertices, the number of near vertices per tree returned by the procedure NearVertices cannot exceed a constant number [28]. Hence, it can be concluded that the execution of NearestVertex and NearVertices procedures on both trees per iteration adds up a constant computational complexity overhead as compared to RRT*. Hence, it can be concluded that IB-RRT* has the same computational complexity as RRT*. The proof for Theorem 3 is exactly the same to one provided for the computational ratio of RRT* and RRT in [28].

7. Experimental results

This section presents simulations performed on a 2.4 GHz Intel corei5 processor with 4 GB RAM. Here, performance results of our IB-RRT* algorithm are compared with RRT* and B-RRT*. Since exploration of the configuration space by B-RRT* after a large number of iterations is similar to that of IB-RRT*, snap shots presented here only depict the IB-RRT* and RRT* algorithms. This is to better demonstrate the difference between the expansion of trees of the two types of algorithms. For proper comparison, experimental conditions and size of the configuration space were kept constant for all algorithms. Since randomized sampling-based algorithms exhibit large variations in results, the algorithms were run up to 50 times with different seed values for each type of environment. Maximum, minimum and average number of iterations i as well as time t utilized by each algorithm to reach the optimal path solution is presented in Table 1. To restrain the computational time within reasonable limits, the maximum limit for the number of tree nodes was kept at 5 million. The column fail in the table denotes the number of runs for which the corresponding algorithm failed to find an optimal path solution within node limits when executed with different seed values for the random function. Although, algorithms were able to determine feasible path solution, this is still considered as a failure, since the table provides comparison for the determination of an optimal path solution only.

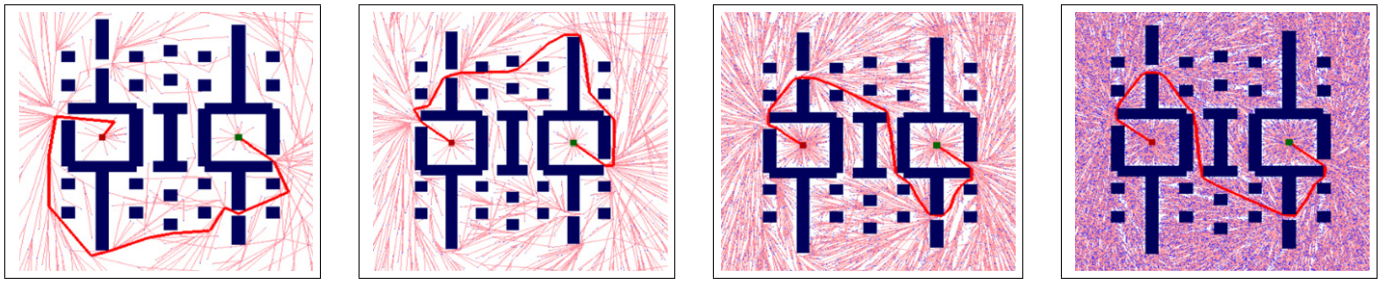
Figs. 2 and 3 illustrate the trees maintained by IB-RRT* and RRT* respectively at different numbers of iterations. The cost C of the path in terms of Euclidean distance is also indicated at each iterations. Table 1 summarizes the number of iterations and time consumed by IB-RRT*, B-RRT* and RRT* to reach an optimal path in this problem. It should be noted that the RRT* algorithm is unable to fully sample the given configuration space and thus fails to converge to the optimal solution within the limit of 5 million iterations. Although both IB-RRT* and B-RRT* were successful in finding the optimal solution, B-RRT* took an extremely large number of iterations to converge in comparison with IB-RRT*. B-RRT* utilizes the partial greedy heuristic approach as discussed earlier (algorithm line), this significantly reduces its ability of convergence to the optimal path solution. Figs. 4 and 5 represent particularly challenging maze type of cluttered 2D test environment. The environment has been set up in such a way that the starting and goal regions, while placed close together, are separated by the maze. All algorithms were tested, Figs. 4(a)–(d) and 5(a)–(d) show the convergence from the initial path solution to the optimal path solution by IB-RRT* and RRT*, respectively. For determination of the optimal path, the IB-RRT* algorithm takes the least number of average iterations ($i_{\text{avg}} = 95\,192$) as compared to B-RRT*($i_{\text{avg}} = 286\,721$) and the extraordinarily large number of iterations taken up by RRT* ($i_{\text{avg}} = 1\,059\,268$) as shown in Table 1.

Fig. 7 shows the 3-D environment containing a multiple of barriers which separate the initial state and the goal region. IB-RRT* determines an optimal path most quickly ($i = 204\,321$) as compared to B-RRT* ($i = 838\,692$) and RRT* ($i = 1\,961\,825$). Although all algorithms utilize uniform sampling heuristic, however, IB-RRT* maximizes the rewiring process per iteration due to intelligent sample insertion heuristic and hence quickly converges to the optimal path solution as compared to B-RRT* and RRT*. Figs. 6 and 8 depict different scenarios in three -dimensional space. Their



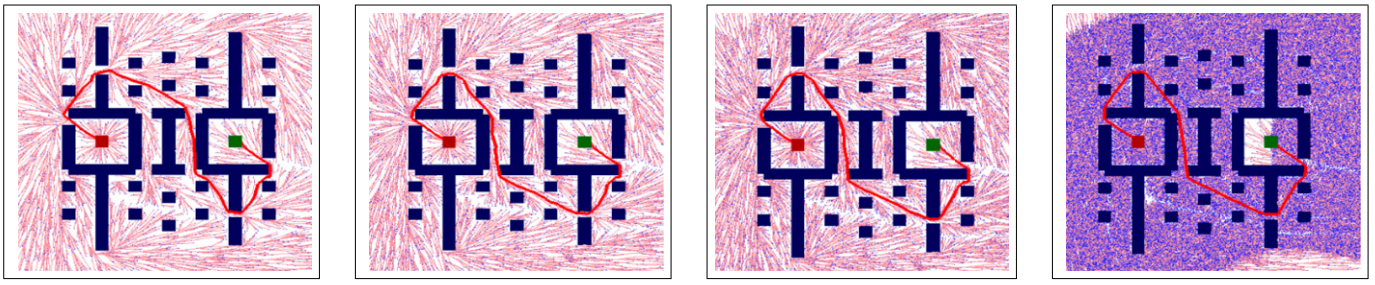
(a) $i = 102\,016$, $t = 17$ s, $C = 97.0$. (b) $i = 510\,982$, $t = 84$ s, $C = 96.5$. (c) $i = 1\,532\,191$, $t = 255$ s, $C = 96.3$. (d) $i = 3\,192\,640$, $t = 510$ s, $C = 96.0$.

Fig. 3. RRT* performance in 2-D environment (A).



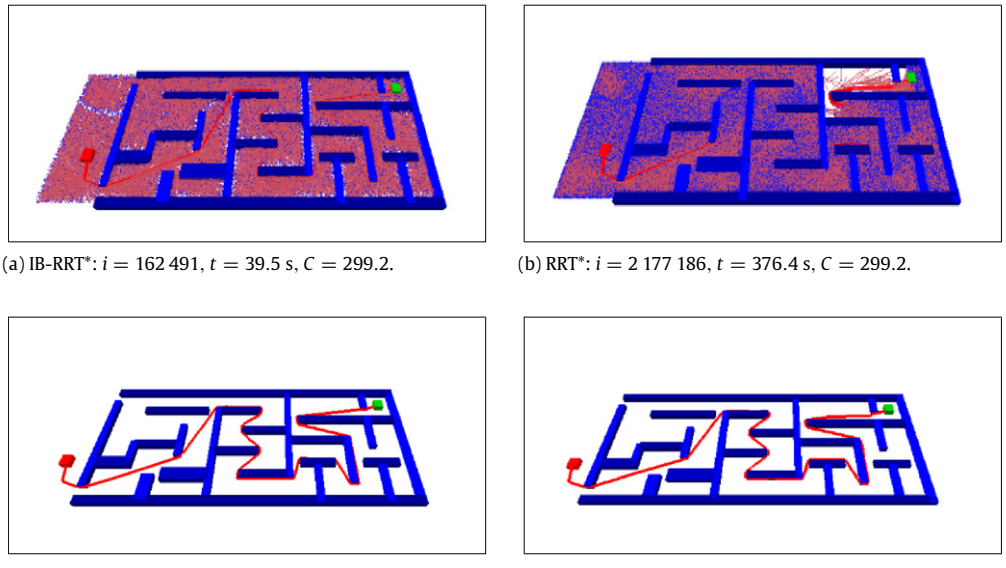
(a) $i = 1812$, $t = 0.42$ s, $C = 83.4$. (b) $i = 5026$, $t = 1.22$ s, $C = 75.4$. (c) $i = 30\,641$, $t = 7.8$ s, $C = 70.8$. (d) $i = 94\,865$, $t = 21.6$ s, $C = 69.4$.

Fig. 4. IB-RRT* performance in 2-D environment (B).



(a) $i = 31\,971$, $t = 5.61$ s, $C = 72.4$. (b) $i = 74\,119$, $t = 12.81$ s, $C = 71.4$. (c) $i = 317\,951$, $t = 55.6$ s, $C = 70.6$. (d) $i = 1\,071\,861$, $t = 184.2$ s, $C = 69.4$.

Fig. 5. RRT* performance in 2-D environment (B).



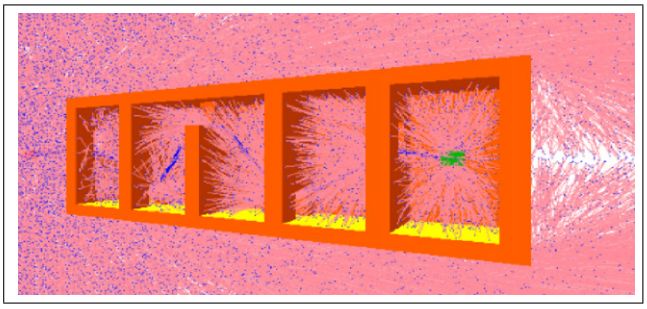
(a) IB-RRT*: $i = 162\,491$, $t = 39.5$ s, $C = 299.2$. (b) RRT*: $i = 2\,177\,186$, $t = 376.4$ s, $C = 299.2$.

(c) IB-RRT*: Optimal path solution. (d) RRT*: Optimal path solution.

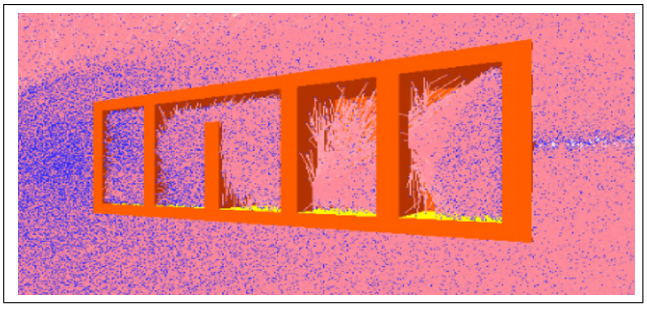
Fig. 6. Performance of IB-RRT* and RRT* in complex maze environment.

Table 1
Experimental results for computing optimal path solution.

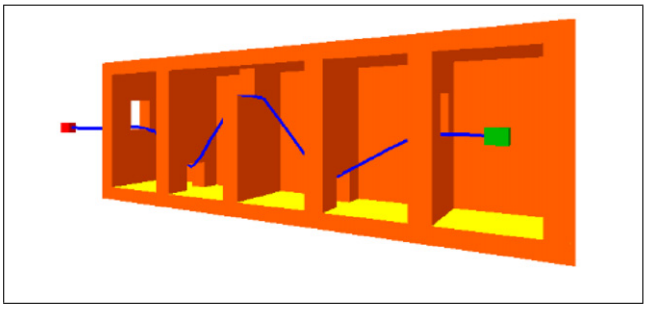
Environment	Algorithm	i_{min}	i_{max}	i_{avg}	t_{min} (s)	t_{max} (s)	t_{avg} (s)	C	Fail
2D-Cluttered (A) (Figs. 2 and 3)	IB-RRT*	159 861	181 521	162 809	37.9	42.3	39.6	93.1	0
	B-RRT*	438 785	463 526	458 328	90.8	97.8	95.3	93.1	3
	RRT*	–	–	–	–	–	–	–	50
2D-Cluttered (B) (Figs. 4 and 5)	IB-RRT*	78 652	141 936	95 192	18.9	33.4	23.3	69.4	0
	B-RRT*	261 716	375 341	286 721	55.2	78.2	60.3	69.4	0
	RRT*	981 431	1 219 628	1 059 268	168.9	210.9	183.4	69.4	7
3D-Multiple barriers (Fig. 7)	IB-RRT*	193 593	218 586	204 321	45.8	53.6	47.8	81.9	0
	B-RRT*	810 581	853 248	838 692	167.3	178.1	176.3	81.9	4
	RRT*	1 941 613	1 978 581	1 961 825	329.7	337.4	332.2	81.9	11
3D-Narrow passages (Fig. 8)	IB-RRT*	29 651	34 239	32 361	6.8	8.2	7.8	69.8	1
	B-RRT*	46 971	57 891	54 916	9.8	12.4	11.3	69.8	5
	RRT*	163 872	168 494	165 627	27.2	29.4	28.6	69.8	8
3D-Maze (Fig. 6)	IB-RRT*	148 786	171 543	168 932	33.9	41.5	39.8	299.2	3
	B-RRT*	753 861	764 926	758 438	155.3	161.2	159.7	299.2	7
	RRT*	2 174 180	2 189 742	2 184 761	368	372	371	299.2	16



(a) IB-RRT*: $i = 186\,731$, $t = 45.4$ s, $C = 81.9$.



(b) RRT*: $i = 2\,106\,391$, $t = 364$ s, $C = 81.9$.



(c) Optimal path solution.

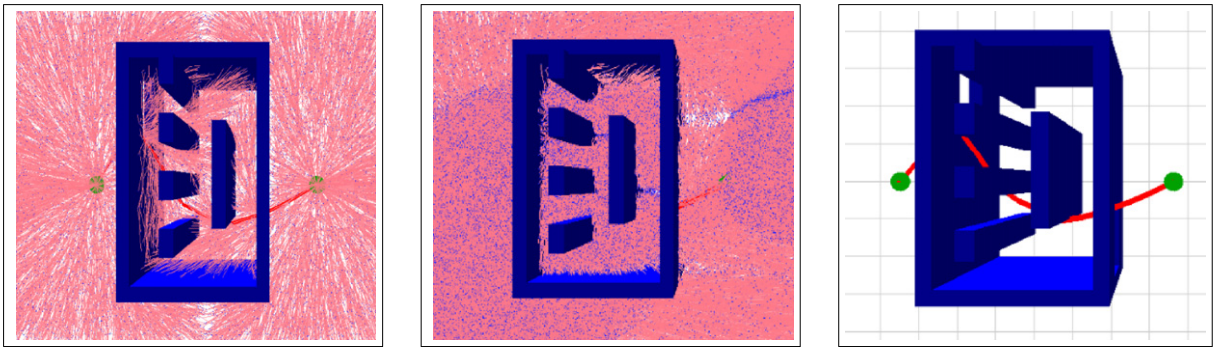
Fig. 7. Sequence of complex barriers.

results are summarized in Table 1. It can be seen that a similar trend is followed by the algorithms in all environments, i.e., IB-RRT* rapidly converges to the optimal solution followed by B-RRT* and then RRT*. Moreover, in the maze problem depicted in Fig. 6, RRT* was unable to sample the area close to the goal region even after an extremely large number of iterations while IB-RRT* was able to fully explore the space in a few thousand iterations.

Fig. 9 summarizes the experimental test results performed in 10 different complex cluttered 2D and 3D environments for the comparison of IB-RRT*, B-RRT* and RRT*. The comparison is done in terms of: (a) iterations and time consumed to determine initial path as well as optimal path solution; (b) memory consumed in terms of bytes for the determination of optimal path solution; (c) convergence rate. From Fig. 9(a)–(e), it can be seen that IB-RRT* consumes lesser iterations, time and memory as compared to B-RRT* and RRT* for the determination of feasible path solution. Fig. 9(f) provides another type of comparison using boxplot. In this the convergence rate of IB-RRT*, B-RRT* and RRT* are compared in these 10 different complex cluttered environments. Let the initial feasible path, denoted by σ_{init}, is computed in t_{init} time while the optimal path solution, denoted as $\sigma*$, is computed in $t*$ time. Then the convergence rate is defined as $\frac{c(\sigma_{\text{init}})-c(\sigma^*)}{t*-t_{\text{init}}}$. Since the process of convergence to the optimal path solution begins after finding the initial feasible path solution, the convergence rate is calculated after initial path computation. It is clear from the box plot that convergence rates of IB-RRT* are highest, followed by B-RRT* and RRT*. There also exists a sizable difference between the convergence rates of IB-RRT* and B-RRT*. Fig. 10 shows the running time ratio of (a) IB-RRT* over BiRRT and (b) IB-RRT* over RRT* after the execution of each iteration. The running time ratio of Algorithm A (AL-A) over Algorithm B (AL-B) is defined as the ratio of time consumed by AL-A over the time consumed by AL-B. It can be seen that as the number of iterations increases, the running time ratio reaches a constant value in both cases. Hence, large numbers of iterations imply that the random samples are fully and uniformly distributed in the obstacle-free space. However, before that time, computational complexity of IB-RRT* remains fairly lower than BiRRT and almost equal to RRT*. As a matter of fact, in this specific environment, the average amount of time taken by our proposed IB-RRT* algorithm to determine a viable path to the goal was seen to be barely four times that of BiRRT and 1.4 times that of RRT*.

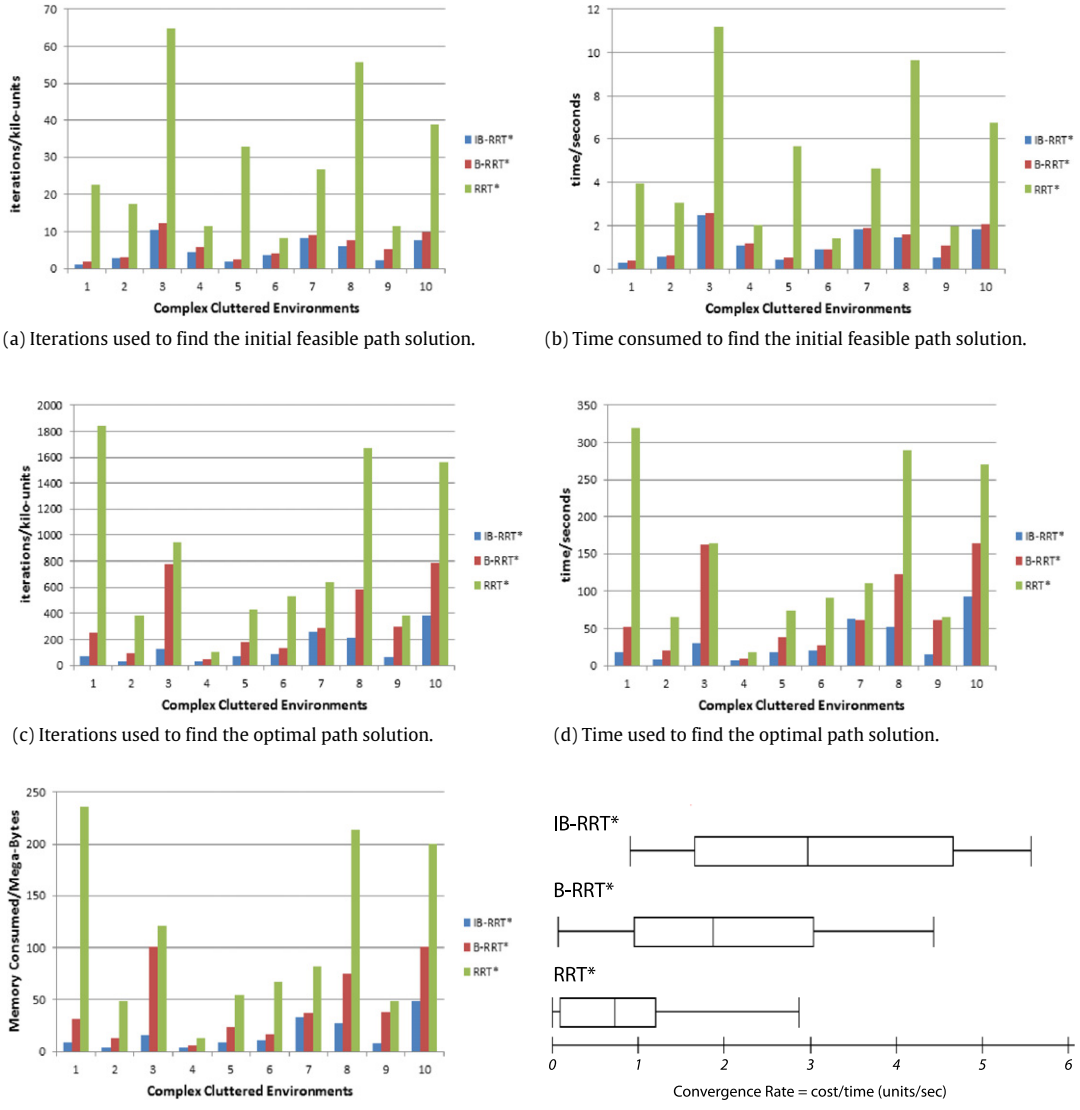
8. Conclusions and future work

This paper presents a detailed comparative analysis of performance of our proposed IB-RRT* algorithm with the existing algorithms RRT* and B-RRT*. It is proven both analytically and experimentally that our proposed algorithm (i) has almost similar computational complexity as RRT* and BiRRT; (ii) provides almost-sure convergence to the optimal path solution; (iii) has the higher convergence rate meaning that it rapidly converges to the optimal solution as compared to both state of the art algorithms RRT*



(a) IB-RRT*: $i = 79\,483$, $t = 18.9$ s, $C = 69.8$. (b) RRT*: $i = 1\,364\,129$, $t = 235.9$ s, $C = 69.8$. (c) Optimal path solution.

Fig. 8. Sequence of narrow passages.



(a) Iterations used to find the initial feasible path solution.

(b) Time consumed to find the initial feasible path solution.

(c) Iterations used to find the optimal path solution.

(d) Time used to find the optimal path solution.

(e) Memory consumed to find the optimal path solution.

(f) Convergence rate comparison.

Fig. 9. Comparison of IB-RRT*, B-RRT* and RRT* in 10 complex cluttered environments.

and B-RRT*; (iv) consumes lesser memory to converge to the optimal solution, as it utilizes lesser iterations and each iteration consumes memory. This paper also presents path planning problems in which the original RRT* algorithm fails to reach the optimal path solution within reasonable limit of iterations. Experimental results supporting theoretical analysis are also presented in this paper. The proposed algorithm IB-RRT* allows rapid convergence to the optimal solution without tuning of the sampling operation for optimal paths. Therefore, the proposed planner is of importance in the field of real-time motion planning. Hence, we anticipate employing IB-RRT* for online motion planning of animated characters in complex 3-D environments.

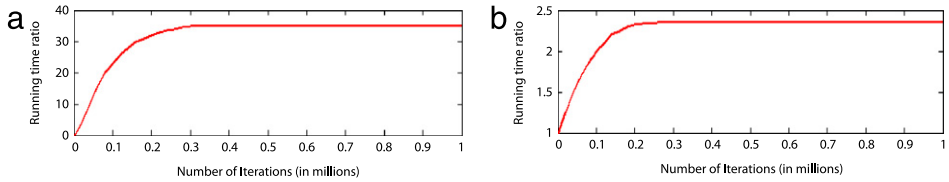


Fig. 10. Running time ratio of (a) IB-RRT* over BiRRT (b) IB-RRT* over RRT*.

Acknowledgments

The authors are grateful to Dr. Sertac Karaman of MIT for sharing implementation RRT* algorithm. Moreover, authors are also thankful to Alejandro Perez of MIT CSAIL for insightful discussion on sampling-based optimal motion planning algorithms.

References

[1] S.M. LaValle, Planning Algorithms, Cambridge University Press, 2006.
[2] J.-C. Latombe, ROBOT MOTION PLANNING.: Edition en Anglais, Springer, 1990.
[3] H. Chang, T.-Y. Li, Assembly maintainability study with motion planning, in: Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on, Vol. 1, IEEE, 1995, pp. 1012–1019.
[4] M. Girard, A.A. Maciejewski, Computational modeling for the computer animation of legged figures, in: ACM SIGGRAPH Computer Graphics, Vol. 19, ACM, 1985, pp. 263–270.
[5] R.D. Howe, Y. Matsuoka, Robotics for surgery, Annu. Rev. Biomed. Eng. 1 (1) (1999) 211–240.
[6] J.-C. Latombe, Motion planning: a journey of robots, molecules, digital actors, and other artifacts, Int. J. Robot. Res. 18 (11) (1999) 1119–1128.
[7] J.T. Schwartz, M. Sharir, On the "piano movers problem". II. General techniques for computing topological properties of real algebraic manifolds, Adv. in Appl. Math. 4 (3) (1983) 298–351.
[8] T. Lozano-Pérez, M.A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, Commun. ACM 22 (10) (1979) 560–570.
[9] J. Canny, The Complexity of Robot Motion Planning, The MIT press, 1988.
[10] S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning, Int. J. Robot. Res. 30 (7) (2011) 846–894.
[11] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, Int. J. Robot. Res. 5 (1) (1986) 90–98.
[12] Y. Koren, J. Borenstein, Potential field methods and their inherent limitations for mobile robot navigation, in: Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on, IEEE, 1991, pp. 1398–1404.
[13] D. Kuan, J. Zamiska, R.A. Brooks, Natural decomposition of free space for path planning, in: Robotics and Automation. Proceedings. 1985 IEEE International Conference on, Vol. 2, IEEE, 1985, pp. 168–173.
[14] R.A. Brooks, T. Lozano-Perez, A subdivision algorithm in configuration space for findpath with rotation.
[15] G.E. Jan, C.-C. Sun, W.C. Tsai, T.-H. Lin, An shortest path algorithm based on delaunay triangulation, IEEE/ASME Trans. Mechatronics 19 (2) (2014) 660–666.
[16] P.C. Chen, Y.K. Hwang, Sandros: a dynamic graph search algorithm for motion planning, IEEE Trans. Robot. Autom. 14 (3) (1998) 390–403.
[17] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, S. Thrun, Anytime search in dynamic graphs, Artif. Intell. 172 (14) (2008) 1613–1643. http://dx.doi.org/10.1016/j.artint.2007.11.009. URL: http://dx.doi.org/10.1016/j.artint.2007.11.009.
[18] M. Elbanhawi, M. Simic, Sampling-based robot motion planning: a review, IEEE Access 2 (2014) 56–77.
[19] T.M. Howard, C.J. Green, A. Kelly, State space sampling of feasible motions for high performance mobile robot navigation in highly constrained environments, in: Field and Service Robotics, Springer, 2008, pp. 585–593.
[20] M. Pivtoraiko, R.A. Knepper, A. Kelly, Differentially constrained mobile robot motion planning in state lattices, J. Field Robot. 26 (3) (2009) 308–333.
[21] M. Pivtoraiko, A. Kelly, Kinodynamic motion planning with state lattice motion primitives, in: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, IEEE, 2011, pp. 2172–2179.
[22] L. Kavraki, J.-C. Latombe, Randomized preprocessing of configuration for fast path planning, in: Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on, IEEE, 1994, pp. 2138–2145.
[23] L.E. Kavraki, P. Svestka, J.-C. Latombe, M.H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Trans. Robot. Autom. 12 (4) (1996) 566–580.
[24] S.M. LaValle, Rapidly-exploring random trees a new tool for path planning.
[25] S.R. Lindemann, S.M. LaValle, Incrementally reducing dispersion by increasing voronoi bias in RRTs, in: Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on, Vol. 4, IEEE, 2004, pp. 3251–3257.
[26] I. Garcia, J.P. How, Improving the efficiency of rapidly-exploring random trees using a potential function planner, in: Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on, IEEE, 2005, pp. 7965–7970.
[27] J.J. Kuffner Jr., 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.
[28] S. Karaman, E. Frazzoli, Incremental sampling-based algorithms for optimal motion planning. ArXiv Preprint arXiv:1005.0416.
[29] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, M.R. Walter, Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms, in: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, IEEE, 2011, pp. 4307–4313.
[30] A.H. Qureshi, K.F. Iqbal, S.M. Qamar, F. Islam, Y. Ayaz, N. Muhammad, Potential guided directional-RRT* for accelerated motion planning in cluttered environments, in: Mechatronics and Automation (ICMA), 2013 IEEE International Conference on, IEEE, 2013, pp. 519–524.
[31] A.H. Qureshi, S. Mumtaz, K.F. Iqbal, B. Ali, Y. Ayaz, F. Ahmed, M.S. Muhammad, O. Hasan, W.Y. Kim, M. Ra, Adaptive potential guided directional-RRT, in: Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on, IEEE, 2013, pp. 1887–1892.
[32] A.H. Qureshi, S. Mumtaz, K.F. Iqbal, Y. Ayaz, M.S. Muhammad, O. Hasan, W.Y. Kim, M. Ra, Triangular geometry based optimal motion planning using RRT*-motion planner, in: Advanced Motion Control (AMC), 2014 IEEE 13th International Workshop on, IEEE, 2014, pp. 380–385.
[33] D. Kim, J. Lee, S.-E. Yoon, Cloud RRT: sampling cloud based RRT, in: Proc. IEEE Int. Conf. Robot. Autom, 2014.
[34] S.R. Lindemann, S.M. LaValle, Current issues in sampling-based motion planning, in: Robotics Research, Springer, 2005, pp. 36–54.
[35] M. Jordan, A. Perez, Optimal bidirectional rapidly-exploring random trees, Tech. Rep. MIT-CSAIL-TR-2013-021, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 2013, August.

Ahmed Hussain Qureshi received his B. Eng. degree in Electrical Engineering from National University of Sciences and Technology (NUST), Pakistan, in 2014. He was a research assistant in Robotics and Intelligent Systems Engineering (RISE) lab at NUST from 2012–2014. Currently, he is working as a research associate in a joint research project with RISE lab (NUST) and Sakura Wheelchair Project, Japan. His research interests include Motion Planning and Control, Human–Robot Interaction and Computer Graphics.

Yasar Ayaz received his B. Eng. degree in Mechatronics Engineering and M. Eng. degree in Electrical Engineering from the National University of Sciences and Technology (NUST), Pakistan, in 2003 & 2005, respectively. He received his Ph.D. degree specializing in Robotics and Machine Intelligence from Tohoku University, Japan, in 2009. He is currently working as the Head of the Department of Robotics and Artificial Intelligence at the School of Mechanical and Manufacturing Engineering (SMME), NUST, Pakistan. His research interests include motion planning, navigation and control of humanoids and mobile robots.