IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 22, 2019, accepted January 8, 2020, date of publication January 21, 2020, date of current version January 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2968471

Hybrid RRT: A Semi-Dual-Tree RRT-Based Motion Planner

REZA MASHAYEKHI[1], MOHD YAMANI IDNA IDRIS[1], MOHAMMAD HOSSEIN ANISI[2], (Senior Member, IEEE), AND ISMAIL AHMEDY[1]

[1]Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia
[2]School of Computer Science and Electronic Engineering, University of Essex–Colchester, Colchester CO4 3SQ, U.K.

Corresponding authors: Reza Mashayekhi (rz.mashayekhi@gmail.com) and Mohd Yamani Idna Idris (yamani@um.edu.my)

This work was supported by the University of Malaya under Grant FP055-2019A and Grant GPF003D-2018.

ABSTRACT Rapidly-exploring Random Trees (RRTs) have been widely used for motion planning problems due to their ability to efficiently find solutions. Informed RRT* is an optimized version of RRT, which not only implements the rewiring process to optimize the tree but also limits the search area to a subset of the state space to return near-optimal solutions faster. However, limiting the state space is a function of the obtained shortest path so that before a solution is found, the planner cannot limit the state space to a subset. Moreover, unidirectional RRTs such as Informed RRT* take more time to find initial solutions in comparison to the bidirectional RRTs. In this paper, we propose Hybrid RRT, which divides the planning process into three parts: finding initial solutions by a dual-tree search, combining two trees into one, and optimizing the solution. In order to obtain an initial solution, Hybrid RRT implements a dual-tree search, which helps it find solutions faster than unidirectional searches. Then, it combines the start tree and the goal tree of the dual-tree search into one so as to implement informed sampling for a single tree to optimize the current solution. The simulation carried out in Open Motion Planning Library (OMPL), which shows that Hybrid RRT achieved outstanding improvement over RRT* and Informed RRT*.

INDEX TERMS Motion planning, path planning, RRT, informed sampling, path optimization.

I. INTRODUCTION

MOTION planning is involved in various applications such as Unmanned Aerial Vehicles (UAVs), Autonomous Underwater Vehicles (AUVs), driver-less cars, virtual prototyping, biology, and computer graphics [1]–[8]. Motion planners need to find collision-free paths for movable agents from one point to another in the state spaces. Several parameters define the differences between the performances of motion planners, such as planning time and path cost. In other words, a motion planner needs to return low-cost paths in a short period.

Motion planning is mostly about finding paths in continuous spaces, which is considered as an Np-hard problem to be solved. In order to avoid this complexity, planners discretize continuous spaces into discrete spaces to limit the number of states that the planners need to check to release paths. There are two types of motion planners: graph-based planners and sampling-based planners.

A. GRAPH-BASED PLANNING

Graph-based methods create maps from continuous state spaces in order to have a finite number of states so that they have a limited number of states to check. There are several graph-based methods, such as Dijkstra [9] and A* [10]. These methods are mostly able to return optimal solutions if at least one solution exists, and they will return failure when there is no solution. Therefore, they are resolution optimal and resolution complete. However, graph-based algorithms have problems with scaling with the problem dimension and size [11].

B. SAMPLING-BASED PLANNING

Sampling-based planning methods do not make a grid map from the state space at the beginning of the planning process. They take random samples from the state space and then check the visibility of the sample to accept or reject it. Thus, sampling-based methods can be scale well with the problem dimension and size. These properties make sampling-based methods successful in robotic motion planning and animated characters [4]. They are probabilistically complete, which is a weaker notion of complete [12]. It means that they are able to return solutions with a sufficient number of samples if a solution exists. However, they are not able to return failure if no solution exists.

There are two types of sampling-based methods: Multi-query and single-query methods. Multi-query planners

The associate editor coordinating the review of this manuscript and approving it for publication was Luigi Biagiotti.

such as Probabilistic Road Map (PRM) [13] can solve several problems with different start locations and goal locations in the state space. They first create a roadmap by taking random samples, then connect different locations of the map through the created roadmap.

single-query planners such as Rapidly-exploring Random Tree (RRT) [14] do not make a roadmap like multi-query planners. Instead, they construct a tree rooted at the start location and explore the state space by growing the tree toward random samples. Once the goal location is sampled, the exploring process will be stopped.

However, it may take time to spot a sample in the goal area due to the random sampling process. Therefore, RRT-Connect [15] has been proposed to address this problem. It is a bidirectional version of RRT, which grow two trees simultaneously, one from the start location and another one from the goal location. Exploring the state space with two trees makes RRT-Connect a faster planner in comparison to RRT, especially when the goal location is challenging to be sampled by using unidirectional searches.

Although RRT-based methods can solve the motion planning problems efficiently, they provide non-optimal solutions [16]. It is due to the fact that they explore the state space with the random walk so that their outputs would be a sequence of random samples, and they do not have any procedure for optimizing their trees.

RRT* [16] implements a rewiring operation, which leads to near-optimal solutions. These types of planners called asymptotically optimal, which means that they will return near-optimal solutions by increasing the number of samples. RRT* does not stop exploring the state spaces after a solution is found. It continues exploring the state space with the aim of returning better solutions than the current one. RRT* keeps sampling all over the state space to optimize the current solution so that it is an inefficient way due to its single-query nature [11].

Informed RRT* [11], [17] solves this problem of RRT* by limiting the search area to a subset of the state space so as to return near-optimal solutions faster than the standard version of RRT*. The subset is a function of the current solution, which means that Informed RRT* cannot limit the state space to a subset before a solution is found. In other words, Informed RRT* acts similarly to RRT* before a solution is found. Therefore, Informed RRT* only expedites the optimization process. It still has the problems of other unidirectional methods, which is spotting a sample in the goal area, especially when the goal area is hidden beyond the narrow passages.

There are some other RRT-based methods, which try to find initial solutions faster than the standard version of RRT* [18]–[20]. *Wang et al.* [18], [19] modified the sampling process to find solutions faster than standard RRT*. However, these methods resulted in nonuniform sample distributions. Batch Informed Trees (BIT*) [20] limits the state space to a subset of it, which including the start location and the goal location in order to return initial solutions faster. Although BIT* could return first solutions faster than RRT* in many scenarios, it requires more time to find the first solutions in Bug-Trap-like scenarios in comparison to methods that do not limit the exploring area.

Almost all RRT-based methods can be divided into two categorizes: non-optimized versions and optimized versions. Non-optimized versions such as RRT and RRT-Connect are used to find initial solutions. On the other hand, optimized versions like RRT* and Informed RRT* have been designed to return near-optimal solutions so that they optimizing their trees from the start moment of planning until the end. However, it makes them slower than non-optimized versions of RRT in terms of finding initial solutions. In Addition to these categorizes, most RRT-based methods are also categorized into two groups: unidirectional and bidirectional methods.

In this paper, we introduce a single-query semi-bidirectional planning method for optimal motion planning problems called Hybrid RRT, which divides the planning time into three phases. Phase one is to find an initial solution, the second phase is to combine two trees of phase one into one tree, and phase three is to optimize the solution. Hybrid RRT implements a dual-tree search to achieve the first solutions faster than unidirectional methods. After finding the first solution, Hybrid RRT needs to merge its two trees into one. Then, it optimizes the tree to find near-optimal solutions.

In order to achieve fast results out from the optimization process, Hybrid RRT limits the state space into a subset of the state space like Informed RRT*. Therefore, it needs to combine two trees of phase one into one tree to be able to implement Informed sampling on a single tree.

Hybrid RRT is neither an entirely unidirectional method nor a fully bidirectional one. It is a combination of both groups. Moreover, it is neither a non-optimized version nor an optimized version. It uses a non-optimized search for finding initial solutions, which make it faster than optimized versions of RRT. Moreover, it implements optimization process to be able to return near-optimal solutions. Hybrid RRT can find first solutions as fast as RRT-Connect and returns the near-optimal solutions as quickly as Informed RRT*.

The remainder of the paper is organized as follows. Section II presents the necessary background for the paper, including motion planning definition and related works. The proposed method, Hybrid RRT, is introduced in Section III. Section IV presents the simulation and Section IV evaluates the simulation results. Section VI concludes the paper.

II. BACKGROUND

This section presents the paper background, including the definitions of the problem, informed set, and related literature.

A. PROBLEM DEFINITION

This paper defines the path planning problem similarly to [11], [16]. Let X be the state space, the configurations that

cannot be selected due to the presence of obstacles shown as $X_{obs} \subset X$, the configurations that can be sampled without having any collisions with obstacles is $X_{free} = cl\,(X_{obs})$. Let show the start position as $x_{start} \in X_{free}$ and the goal area as $X_{goal} \subset X_{free}$. Therefore, a path between the start position and the goal area will be defined as a set $\sigma : [0, 1] \rightarrow X_{free}$ such that $\sigma\,(0) = x_{start}$ and $\sigma\,(1) \in X_{goal}$.

Another important definition is path cost, which will be shown as $c : \Sigma_{X_{free}} \rightarrow \mathbb{R}_{\geq 0}$. This function defines a cost to each path, which is used as the quality criterion so that lower path cost means more optimality. As a result, searching for paths with lower path cost is the definition of the optimal path planning. This definition is represented in (1).

$$\sigma^* = arg_{\sigma\in\Sigma} min\{c\,(\sigma) \mid \sigma\,(0) = x_{start},\ \sigma\,(1) \in X_{goal},\ \forall s \in [0, 1], \sigma(s) \in X_{free}\} \quad (1)$$

A subset of the state space that potentially can provide shorter paths than the current one shown as $X_f \subseteq X$. Let c_{best} be the cost of the current shortest path,

$$X_f = \{x \in X \mid f(x) < c_{best}\}. \quad (2)$$

If a planner limits the search area to X_f, which has fewer states than the whole state space, it then can return near-optimal solutions faster.

Although searching within X_f can help find near-optimal solutions faster, finding the exact subset would be computationally expensive. Therefore, an estimation of this subset would be better to be considered for optimization purposes. In order to have an estimated subset, a heuristic function is required, $\hat{f}(\cdot)$. The definition of $\hat{f}(\cdot)$ is similar to (2).

B. INFORMED SET

Gammell *et al.* [11] defined this subset of the state space as an n-dimensional prolate hyperspheroid. They divided the path cost,$f(x)$, into two different costs $g(x)$ and $h(x)$. The cost-to-come, $g(x)$, is the path cost from the start location to x. Similarly, the cost-to-go, $h(x)$, is the path cost from x to the goal location. In order to estimate $f(x)$, they estimated $g(x)$ and $h(x)$. The estimation of $f(x)$, $\hat{f}(x)$, is equal to the sum of the estimation of $g(x)$, $\hat{g}(x)$, and the estimation of $h(x)$, $\hat{h}(x)$.

$\hat{g}(x)$ is the Euclidean distance between the start location and x. Similarly, $\hat{h}(x)$ is the Euclidean distance between x and the goal location. Therefore, $\hat{f}(x)$ will be defined as

$$X_{\hat{f}} = \{x \in X \mid \| x_{start} - x \|_2 + \| x - x_{goal} \|_2 \leq c_{best}\}.$$

A 2D example of this subset is shown in Fig. 1, in which the start location, x_{start}, and the goal location, x_{goal}, are the focal points of the ellipse. The transverse diameter of the ellipse is the path cost of the shortest path, c_{best}, and its conjugate diameter is obtained via $\sqrt{c_{best}^2 - c_{min}^2}$, where the Euclidean distance between x_{start} and x_{goal} shown as c_{min}.

C. DIRECT SAMPLING OF AN ELLIPSOIDAL SUBSET

In order to limit the sampling area to an ellipsoidal subset, we need to have a sampler function, which returns samples

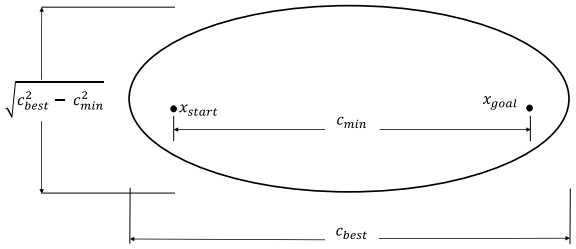


FIGURE 1. The informed set is an ellipse that the start location and the goal location are the focal points. c_{min} is the Euclidean distance between the start and the goal locations, and c_{best} is the cost of the current shortest path.

from the subset instead of the whole state space. Direct sampling from the ellipsoidal subset has introduced in [11].

This sampling process need to distribute the samples uniformly within the subset $X_{ellipse} \sim \mathcal{U}(X_{ellipse})$. It can be achieved by distribute the sample uniformly over a unit n-ball and then transfer them to the ellipsoidal subset, $X_{ball} \sim \mathcal{U}(X_{ball})$,

$$x_{ellipse} = Lx_{ball} + x_{center},$$

x_{center} will be obtained from $(x_{f1} + x_{f2})/2$, where x_{f1} and x_{f2} are the focal points of the ellipsoidal subset. X_{ball} is the samples inside a unit n-ball, $X_{ball} = \{x \in X \mid \| x \|_2 \leq 1\}$ [21].

The transformation is obtained by using Cholesky decomposition of the hyperellipsoid matrix, $S \in \mathbb{R}^{n\times n}$,

$$LL^T \equiv S,$$
$$(x - x_{center})^T S(x - x_{center}) = 1,$$

S including eigenvectors corresponding to the axes of the hyperellipsoid, $\{a_i\}$, and eigenvalues corresponding to the squares of its radii, $\{r_i^2\}$. The transformation, L, maintains the uniform distribution in $X_{ellipse}$ [22].

As a result, the estimation of subset, $X_{\hat{f}}$, will be obtained by transverse the radii and axis. The diagonal matrix of transverse axis is

$$S = diag\{\frac{c_{best}^2}{4}, \frac{c_{best}^2 - c_{\min}^2}{4}, \ldots, \frac{c_{best}^2 - c_{\min}^2}{4}\}$$

and decomposition

$$L = diag\{\frac{c_{best}}{2}, \frac{\sqrt{c_{best}^2 - c_{\min}^2}}{2}, \ldots, \frac{\sqrt{c_{best}^2 - c_{\min}^2}}{2}\}$$

where $diag\{\cdot\}$ is the diagonal matrix.

After transforming samples from a unit n-ball to a n-dimensional ellipsoid, they must be rotated to the world frame. Wahba problem [23] is used to solve it.

The rotation matrix is

$$C = U\ diag\{1, \ldots, 1, det(U)det(V)\}V_T,$$

where $det(\cdot)$ is matrix determinant, $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{n\times n}$ are unitary matrices of $U\Sigma V^T \equiv M$ through singular value decomposition. The matrix M is calculated via the outer product of the first column of the identity matrix, 1_1, and the

transverse axis on the world frame, a_1,

$$M = a_1 1_1^T,$$

where

$$a_1 = (x_{goal} - x_{start})/ \parallel x_{goal} - x_{start} \parallel_2 .$$

Therefore, the informed subset will be obtained through

$$x_{\hat{f}} = CLx_{ball} + x_{center},$$

The procedure of sampling from the informed set is presented in Alg. 7.

D. GRAPH PRUNING

Graph pruning is a process that eradicates unnecessary vertices from the tree so as to keep the tree as small as possible, which helps the planner explore the state space faster. Several works implement graph pruning, such as [24]. In [24], each vertex, x, has an estimated solution cost, $\hat{f}(x)$, which is the sum of the actual cost-to-come, $g(x)$, and the estimated cost-to-go, $\hat{h}(x)$. Thus, $\hat{f}(x) = g(x) + \hat{h}(x)$.

$g(x)$ is the actual cost of the vertex from the start location to the vertex via passing through the tree. The estimated Cost-to-go, $\hat{h}(x)$, is the Euclidean distance between the vertex and the goal location. If $\hat{f}(x)$ is higher than the length of the current shortest path, then the vertex will be removed from the tree. Although this method keeps the tree smaller, the cost-to-come of a vertex could get smaller due to the rewiring process, which means that this process may remove some vertices that could potentially provide better solutions.

Gammell *et al.* [17] implemented the pruning process similar to [24], but they replaced the actual cost-to-come with the estimated one to avoid removing vertices, which could possibly provide better solutions. Therefore, the estimated solution cost will be changed to $\hat{f}(x) = \hat{g}(x) + \hat{h}(x)$. In other words, they considered the lowest cost of the path by taking the shortest distance of x from the start location and the goal location. As a result, the only vertices that could not provide better solutions than the current one will be removed from the tree. Alg. 9 presents the pruning tree process.

E. RELATED WORK

The related works of this paper are presented in this section. Rapidly-exploring Random Tree (RRT) [14] is a tree-based motion planning method that solves single-query problems by growing a tree from the start location toward random samples. RRT stops exploring once it adds one sample located inside the goal area to its tree.

RRT is an efficient and straightforward motion planner that explores the collision-free part of the state spaces rapidly. However, it may take time to sample the goal region due to its random sampling process.

Kuffner and LaValle [15] introduced a dual-tree version of RRT, RRT-Connect, which explores the state space by implementing two trees, one tree is growing from the start location, T_{start}, and another tree is growing from the goal location, T_{goal}. RRT-Connect grow T_{start} and T_{goal} simultaneously and try to find a connection between them. The planner stops exploring the state space once one connection between its two trees is found.

RRT and RRT-Connect are able to solve high-dimensional single-query path planning problems. However, their outputs remain non-optimal due to the lack of the optimization process.

Karaman and Frazzoli [16] proposed RRT*, which has an optimization method for RRT. The optimization process rewires the tree around the newly added vertices by considering them as potential parents for their nearby vertices. This method returns optimal solutions in which the path length is defined as the optimality criterion.

Unlike RRT, RRT* does not stop exploring the state space after an initial solution is found. It keeps sampling the state space with the aim of optimizing the tree.

Although RRT* can return near-optimal solutions, it is exploring all over the state space to optimize the tree, which is not an efficient method due to its single-query nature [11].

It is better to search through the states that could possibly provide better solutions than the current one instead of all over the state space. Gammel *et al.* [11], [17] proposed another version of RRT*, Informed RRT*, which limits the search area to a subset of the state space based on the current solution length. It helps the planner returns near-optimal solutions faster than the standard version of RRT*.

Although RRT*, and Informed RRT* could return near-optimal solutions, they are unidirectional searches, which means that they may take more time to find an initial solution than bidirectional methods.

III. THE PROPOSED METHOD

In this section, we present the proposed method, Hybrid RRT, which is a semi-dual-tree RRT-based method.

Hybrid RRT divides the planning process into three sub-processes: finding-an-initial-solution, combining-two-trees, and optimizing-the-current-solution.

For finding an initial solution, Hybrid RRT implements a dual-tree search to be able to find the first solutions faster than the unidirectional searches. For the optimization process, it applies informed sampling on a single-tree, which helps it return near-optimal solutions more quickly than other methods that do not limit their search area for the optimization process. In other words, it uses a bidirectional search in finding-an-initial-solution sub-process, and uses a unidirectional search for optimizing-the-current-solution sub-process. Therefore, it needs to convert the two trees of the first sub-process into one to be able to pass it the third sub-process. Thus, the second sub-process duty is to transform the bidirectional trees into a unidirectional tree.

A. HYBRID RRT ALGORITHM

Alg. 1 outlines the steps of the Hybrid RRT method. It first initializes the parameters of the algorithm from line 1 to line 5. The two trees are initialized by having x_{start} and x_{goal} as their roots. There are two other parameters, X_{soln}

and c_{best}, which need to be initialized. X_{soln} keeps all the vertices located within the goal region, and c_{best} keeps the cost of the shortest path. Therefore, at the beginning of the planning process, X_{soln} is an empty collection, and c_{best} is equal to infinity, which means that no solution is found and the cost of going from the start location to the goal location is infinity. Then, Hybrid RRT calls *FindFirstSolution* function to find a solution. If this function could find an initial solution, the obtained solution needs to be optimized. After finding the first solution, the two trees need to be merged so as to become ready for the optimization process. Therefore, Hybrid RRT calls *CombineTwoTrees* function to combine the two trees into one. Then, the obtained tree will be passed to *OptimizeTree* function for the optimization process.

Algorithm 1 Hybrid RRT Algorithm

$V_a \leftarrow \{x_{start}\}$; $E_a \leftarrow \emptyset$;
$V_b \leftarrow \{x_{goal}\}$; $E_b \leftarrow \emptyset$;
$G_a \leftarrow (V_a, E_a)$; $G_b \leftarrow (V_b, E_b)$;
$X_{soln} \leftarrow \emptyset$;
$c_{best} \leftarrow \infty$;
$[status, x_{cxn}] \leftarrow$ *FindFirstSolution*(G_a, G_b);
if $status \neq$ *Failure* **then**
  $[G_a, c_{best}, X_{soln}] \leftarrow$ *CombineTwoTrees*$(G_a, G_b, c_{best}, X_{soln}, x_{cxn})$;
  $G_a \leftarrow$ *OptimizeTree*$(G_a, x_{start}, x_{goal}, c_{best}, X_{soln})$;
end if
return G_a;

1) FINDFIRSTSOLUTION FUNCTION

FindFirstSolution gets two trees (G_a and G_b) as its input arguments. Then, it starts exploring the state space. It, first, gets a random sample then expands G_a toward the sampled point by *Extend* function. It then expands G_b toward the newly added vertex of G_a by *Connect* function. Afterward, G_a and G_b will be swapped for the next iteration. Once a link between these two trees is found, the function returns *Success* as the status, as well as the connection point of the trees, x_{cxn}. If the number of iterations reaches its maximum and no link between trees is found, the function returns *Failure* as the status, and *null* as the connection point, x_{cxn}, so that the planning will be stopped.

Algorithm 2 *FindFirstSolution* Function

function *FindFirstSolution*(G_a, G_b)
  for $i = 1$ *to* n **do**
    $x_{rand} \leftarrow$ *Sample*();
    if *Extend*$(G_a, x_{rand}) \neq$ *Trapped* **then**
      if *Connect*$(G_b, x_{new}) =$ *Reached* **then**
        $x_{cxn} \leftarrow x_{new}$;
        $status \leftarrow$ *Success*;
        return $[status, x_{cxn}]$;
      end if
    end if
    Swap(G_a, G_b);
  end for
  $x_{cxn} \leftarrow$ *null*;
  $status \leftarrow$ *Failure*;
  return $[status, x_{cxn}]$;
end function

Extend function gets the tree and the random sample, it then finds the nearest vertex of the tree to the sample. Afterward, it implements the required constraints via *Steer* function. Finally, it adds the sampled point to the tree if this connection is collision-free.

Extend function provides three different status, *Reached*, *Trapped*, and *Advanced*. *Reached* is when the sample is added to the tree, and *Trapped* is when the sample cannot be added to the tree due to the presence of an obstacle. Finally, if the sample is far from the tree reach and then another vertex in the direction of the sample but nearer to the tree is added to the tree, the function will return *Advanced* Alg. 3 outlines the *Extend* function.

Algorithm 3 *Extend* Function

function *Extend*$(G = (V, E), x)$
  $x_{nearest} \leftarrow$ *Nearest*(G, x);
  $x_{new} \leftarrow$ *Steer*$(x_{nearest}, x)$;
  if *isCollisionFree*$(x_{nearest}, x_{new})$ **then**
    $V \leftarrow V \bigcup \{x_{new}\}$;
    $E \leftarrow E \bigcup \{x_{nearest}, x_{new}\}$;
    if $(x_{new} = x)$ **then**
      return *Reached*;
    else
      return *Advanced*;
    end if
  end if
  return *Trapped*;
end function

Connect function duty is to connect two trees. It gets the newly added vertex of G_a, x_{new}, and G_b, and then try to add x_{new} to G_b by calling *Extend* function repeatedly. It stops calling *Extend* function when *Extend* function returns either *Reached* or *Trapped*. When *Connect* function receives *Reached* from *Extend* function, it means that the connection between the two trees is found. Alg. 4 presents this procedure.

Algorithm 4 *Connect* Function

function *Connect*(G, x)
  repeat
    $S \leftarrow$ *Extend*(G, x);
  until $S \neq$ *Advanced*;
  return S;
end function

2) COMBINETWOTREES FUNCTION

After *FindFirstSolution* function finds a path, it is time to merge the two trees into one. Therefore, all vertices and edges of the goal tree must be added to the start tree.

$$E_{startTree} = E_{startTree} \cup E_{goalTree}$$
$$V_{startTree} = V_{startTree} \cup V_{goalTree}$$

where $E_{startTree}$ and $E_{goalTree}$ stand for edges of the start tree and the goal tree, respectively. Similarly, $V_{startTree}$ and $V_{goalTree}$ are the vertices of the start tree and the goal tree, respectively. For simplicity $E_{startTree}$ and $V_{startTree}$ will be shown by E and V.

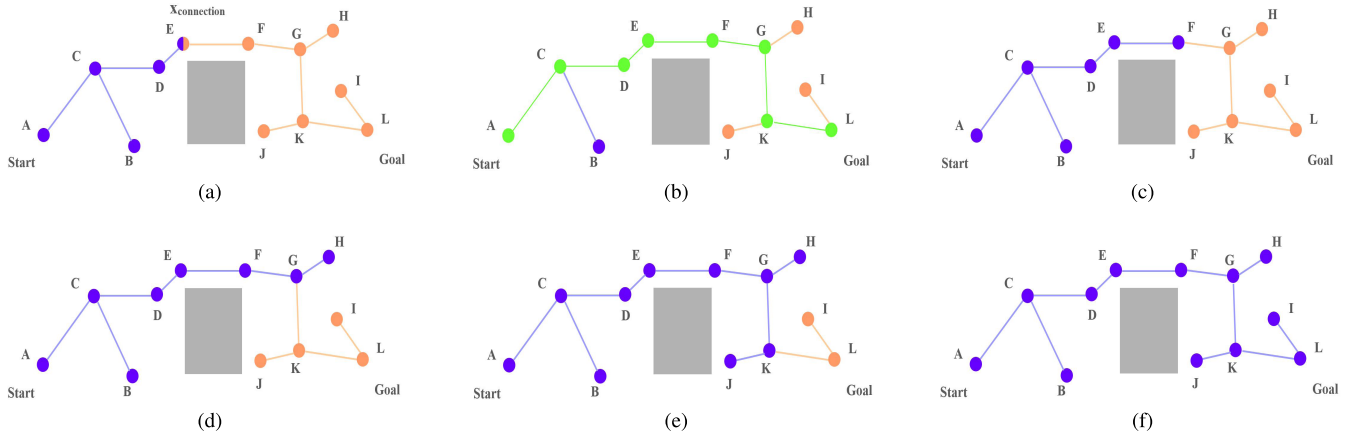


FIGURE 2. An example of combining two trees together.

At this stage, all vertices and edges of the goal tree are added to the start tree. However, some modifications are needed to connect these two trees correctly. All vertices of the goal tree must be directed to the x_{start} instead of x_{goal}. In other words, the root of all the goal tree vertices must be changed from x_{goal} to x_{start}.

The path is the only connection between these two trees so that the path is the starting stage of merging these two trees.

In problem definition section, II-A, path defined as $\sigma[0,\ 1] \rightarrow X_{free}$ such that $\sigma\ (0) = x_{start}$ and $\sigma\ (1) \in\ X_{goal}$. *FindFirstSolution* function implements bidirectional search to find solutions. Therefore, the path has two different parts; one part is from the start tree, while another is from the goal tree. Let x_{cxn} be the connection vertex that is in both trees. Therefore, the path defined as $\sigma\ =\ \sigma_{start} \cup \sigma_{goal}$ such that

$$\sigma_{start}[0,\ 1] \rightarrow X_{free} \mid \sigma_{start}(0) = x_{start},\ \sigma_{start}(1) = x_{cxn},$$

and

$$\sigma_{goal}[0,\ 1] \rightarrow X_{free} \mid \sigma_{goal}(0) = x_{goal},\ \sigma_{goal}(1) = x_{cxn}.$$

Let $n+1$ be the number of vertices in σ_{start}, and $m+1$ be the number of vertices in σ_{goal}. So, σ_{start} can be defined such

$$\begin{aligned}\sigma_{start}\ &= \sigma_{start}(0) \cup \sigma_{start}(\frac{1}{n}) \cup \sigma_{start}(\frac{2}{n}) \\ &\quad \cup \ldots \cup \sigma_{start}(\frac{n-1}{n}) \\ \cup\sigma_{start}(1) &= \bigcup_{i=0}^{n} \sigma_{start}(\frac{i}{n})\end{aligned}$$

Similarly, σ_{goal} can be defined such

$$\begin{aligned}\sigma_{goal}\ &= \sigma_{goal}(0) \cup \sigma_{goal}(\frac{1}{m}) \cup \sigma_{goal}(\frac{2}{m}) \\ &\quad \cup \ldots \cup \sigma_{goal}(\frac{m-1}{m}) \\ \cup\sigma_{goal}(1) &= \bigcup_{i=0}^{m} \sigma_{start}(\frac{i}{m})\end{aligned}$$

σ_{start} is already part of the start tree, while σ_{goal} needs to be modified to be part of the start tree. The first step is to remove the edge of x_{cxn} and its parent in the goal tree and add it to the start tree. In other words, the rule of child and parent must be exchanged, x_{cxn} must be parent of its parent in the goal tree, $\sigma_{goal}(\frac{m-1}{m})$.

Therefore, the edge in which $\sigma_{goal}(\frac{m-1}{m})$ was the parent and x_{cxn} was the child must be removed from E. Let show an edge with its two vertices as $(x_{parent},\ x_{child})$. So,

$$E \leftarrow E \backslash (\sigma_{goal}(\frac{m-1}{m}),\ x_{cxn})$$

Then, the new edge in which x_{cxn} is the parent of $\sigma_{goal}(\frac{m-1}{m})$ must be added to E.

$$E \leftarrow E \cup (x_{cxn},\ \sigma_{goal}(\frac{m-1}{m}))$$

As a result, the two trees connection is no longer x_{cxn}. They are now connected via the previous parent of x_{cxn}, $\sigma_{goal}(\frac{m-1}{m})$. In other words, x_{cxn} gets one edge closer to x_{goal}. This process must be continued until x_{goal} added to the start tree. It means that all the edges of σ_{goal} must be removed from E.

$$E \leftarrow E \backslash (\bigcup_{i=m}^{0} (\sigma_{goal}(\frac{i-1}{m}),\ \sigma_{goal}(\frac{i}{m})))$$

Instead of the removed edges, the reversed version of them must be added to E.

$$E \leftarrow E \bigcup_{i=m}^{0} (\sigma_{goal}(\frac{i}{m}),\ \sigma_{goal}(\frac{i-1}{m}))$$

After changing the directions of all edges of σ_{goal}, the vertices of the path have their way back to x_{start}, which means that they are connected to the start tree correctly.

All vertices of the goal tree can be categorized into three groups from the viewpoint of σ_{goal}: the first group is the vertices located on the path, the second group is the vertices that pass from at least one of the path vertices to reach their root, x_{goal}, and finally, the third group is the vertices that in their way to x_{goal}, they do not pass from any of the path vertices. The third group is connected to x_{goal} via other branches of the goal tree than the path.

By reversing the path edges in the goal tree part, the vertices of the first group are now connected to the start tree correctly. The second group, which are connected to their roots via at least one of the path vertices, they are now connected to x_{start} instead of x_{goal}. It is due to the fact that when they are going back to their root, they need to pass via at least one of the path vertices; once they reached the path vertices, they will be directed to x_{start}. The third group, which are connected to x_{goal} without passing from any of the path vertices, they are also connected to the x_{start}. When they start going back to their root, they will reach x_{goal}, which is now a part of the start tree. Therefore, all the vertices of the goal tree are connected to the start tree correctly. This procedure is presented algorithmically in Alg. 5.

Fig. 2 shows an example of merging two tree by implementing the presented methodology. Fig. 2a shows the two trees before start merging them. The start tree highlighted by blue, and the goal tree highlighted by orange. The vertex E is connection vertex, x_{cxn}. The path between vertex A, as x_{start}, and L as x_{goal}, is shown in Fig. 2b, in which the vertices and the edges of the path are highlighted by green color.

The first step is to remove the edge of x_{cxn}, E, and its parent, F, in the goal tree, and add another edge in which x_{cxn}, E, will be the parent of F, Fig. 2c.

Similar to the first step, all other path edges that belong to the goal tree must be removed from the edge matrix, and their new versions in which the rule of child and parent are swapped must be added. Therefore, the next change is to remove G from its parent, K, and connect it as a child to F, Fig. 2d. By replacing the parent of G, vertex H is now connected to the start tree correctly, because, in its way back to the root, it comes to vertex G so that it will be directed to x_{start}, vertex A, instead of going to x_{goal}, vertex L.

Similarly, K must be disconnected from L and then connected to G as one of its children. By doing so, J will be connected to the start tree, too, Fig. 2e.

The finall step is to add L, x_{goal}, as a child to K, Fig. 2f. As a result, all other vertices that are connected to x_{goal}, such as I, have their path back to x_{start}, vertex A.

Algorithm 5 *CombineTwoTrees* Function

function *CombineTwoTrees*(G_a, G_b, c_{best}, X_{soln}, x_{cxn})
  $E \leftarrow E_a \bigcup E_b$;
  $V \leftarrow V_a \bigcup V_b$;
  $G = (V, E)$;
  $child = x_{cxn}.ParentInGoalTree$;
  $newParent = x_{cxn}$;
  while *isNotNull*($child$) **do**
    $oldParent = child.parent$;
    $E \leftarrow E \setminus \{(oldParent, child)\}$;
    $E \leftarrow E \bigcup \{(newParent, child)\}$;
    $newParent = child$;
    $child = oldParent$;
  end while
  $X_{soln} \leftarrow \{v \in V \mid v \in X_{goal}\}$;
  $c_{best} \leftarrow min_{x_{soln}} \in X_{soln}\{Cost(x_{soln})\}$;
  return $[G, c_{best}, X_{soln}]$;
end function

3) OPTIMIZETREE FUNCTION

In this stage, the planner has found a solution by using the dual-tree search and then merge the two trees into one. Therefore, it is time to optimize the solution. There are several methods for optimizing the current solution of RRT-based methods. Among them, informed sampling has shown a significant impact by limiting the state space to one of its subsets to make exploring area smaller, which helps the planner return near-optimal solutions faster than other methods.

Algorithm 6 *OptimizeTree* Function

function *OptimizeTree*(G, x_{start}, x_{goal}, c_{best}, X_{soln})
  for $i = 1$ *to* n **do**
    $x_{rand} \leftarrow InformedSample(x_{start}, x_{goal}, c_{best})$;
    if $Extend^*(G, x_{rand}) \neq Trapped$ **then**
      if $x_{new} \in X_{goal}$ **then**
        $X_{soln} \leftarrow X_{soln} \bigcup \{x_{new}\}$
      end if
      $previous\_c_{best} \leftarrow c_{best}$;
      $c_{best} \leftarrow min_{x_{soln}} \in X_{soln}\{Cost(x_{soln})\}$;
      if $c_{best} < previous\_c_{best}$ **then**
        $PruneTree(V, E, c_{best})$;
      end if
    end if
  end for
  return G;
end function

Alg. 6 outlines the steps of *OptimizeTree* function, which uses Informed sampling to optimize the tree. *InformedSample* is called to return a sample within the subset, and then the x_{rand} returned from the *InformedSample* will be passed to the *Extend** to expand the tree toward the x_{rand}. If *Extend** function was not successful in adding x_{new} to the tree, then the algorithm goes for the next iteration and takes another sample. Otherwise, the newly added vertex, x_{new}, will be checked to find whether it is located within the goal area, X_{goal}. If so, x_{new} will be added to X_{soln}. Then, the shortest paths will be calculated, and it will be compared with its previous value. If a better value is found, the tree must be pruned based on the new value of c_{best}.

InformedSample gets x_{start}, x_{goal}, and c_{best} and then returns a sample within the informed set. This sampling process is outlined in Alg. 7.

Algorithm 7 *InformedSample* Function

function *InformedSample*(x_{start}, x_{goal}, c_{best})
  $c_{min} \leftarrow \| x_{goal} - x_{start} \|_2$;
  $x_{center} \leftarrow (x_{start} + x_{goal})/2$;
  $\mathbf{C} \leftarrow RotationToWorldFrame(x_{start}, x_{goal})$;
  $r_1 \leftarrow c_{best}/2$;
  $\{r_i\}_{i=2,\ldots,n} \leftarrow (\sqrt{c_{max}^2 - c_{min}^2})/2$;
  $\mathbf{L} \leftarrow diag\{r_1, r_2, \ldots, r_n\}$;
  $x_{ball} \leftarrow SampleUnitBall$;
  $x_{rand} \leftarrow (\mathbf{C}\mathbf{L}x_{ball} + x_{center}) \bigcap X$;
  return x_{rand};
end function

Extend* function is the optimized version of *Extend*. It includes the rewiring process, which considers x_{new} as a

potential parent for its near vertices. Alg. 8 outlines *Extend** function.

Algorithm 8 Extend* Function

```
function Extend*(G = (V, E), x)
    x_nearest ← Nearest(G, x);
    x_new ← Steer(x_nearest, x);
    if isCollisionFree(x_nearest, x_new) then
        V ← V ⋃ {x_new};
        x_min ← x_nearest;
        X_near ← Near(G, x_new, r_RRT*);
        c_min ← Cost(x_nearest, G) + Cost(Line(x_nearest, x_new));
        for each x_near ∈ X_near \ x_nearest do
            if isCollisionFree(x_near, x_new) & (Cost(x_near, G) +
Cost(Line(x_near, x_new)) < c_min) then
                x_min ← x_near;
                c_min ← Cost(x_near, G) + Cost(Line(x_near, x_new));
            end if
        end for
        E ← E ⋃ {x_min, x_new};
        for each x_near ∈ X_near \ x_min do
            if isCollisionFree(x_near, x_new) & (Cost(x_new, G) +
Cost(Line(x_new, x_near)) < Cost(x_near, G)) then
                x_parent ← Parent(x_near, G);
                E ← E \ {(x_parent, x_near)};
                E ← E ⋃ {(x_new, x_near)};
            end if
        end for
        if (x_new = x) then
            return Reached;
        else
            return Advanced;
        end if
    end if
    return Trapped;
end function
```

PruneTree is removing the vertices of the tree which are not able to provide better solutions than the current one. This function removes the leaves of the tree that have the estimated cost, $\hat{f}(v) = \hat{g}(v) + \hat{h}(v)$, more than the current shortest path cost, c_{best}. This procedure is presented in Alg. 9.

IV. SIMULATION

Hybrid RRT was compared to other RRT-based methods on simulated problems in $\mathbb{R}^3$ and $\mathbb{R}^6$ using Open Motion Planning Library (OMPL) [25]. The experiments have been carried out on a laptop with the Intel Core i7-4600M processor and 16 GB of RAM. The experiments were run in Ubuntu 18.04.3 LTS installed on VMware (8 GB of RAM were assigned to the VMware).

Four different state spaces are selected for the simulation, which are the built-in state spaces of OMPL App. These four state spaces are shown in Fig. 3 and Fig. 4. The planners were compared together in these state spaces based on their ability to find initial solutions, and their ability to return near-optimal solutions. The planners were run 100 times in each scenario.

OMPL is a motion planning library written in C++, which is integrated with the Robot Operating System (ROS) [26]. OMPL App is the front-end for OMPL, which has several rigid bodies and state spaces. Simulation carried out on four different OMPL App scenarios, BugTrap_planar, Maze_planar, Home, and Twistycool. BugTrap_planar (Fig. 3), and Maze_planar (Fig. 4a) are 3D state spaces, while Home (Fig. 4b), and Twistycool (Fig. 4c) are 6D state spaces.

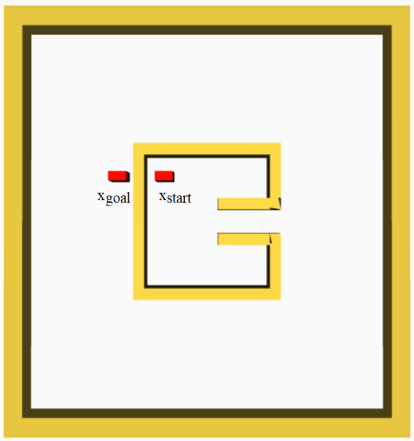


FIGURE 3. The OMPL App BugTrap_planar state space. It includes two rigid bodies representing the start location, x_{start}, and the goal location, x_{goal}.

Algorithm 9 *PruneTree* Function

```
function PruneTree(V ⊆ X, E ⊆ V × V, c_best ∈ ℝ_⩾0)
    do
        V_prune ← {v ∈ V | f̂(v) > c_best, and ∀w ∈ V, (v, w) ∉ E};
        E ← {(u, v) ∈ E | v ∈ V_prune};
        V ← V_prune;
    while V_prune ≠ ∅;
end function
```

Simulations are divided into two categories: the ability to find initial solutions and the ability to return near-optimal solutions.

A. FIND INITIAL SOLUTIONS

In this test, Hybrid RRT has been compared to RRT, RRT-Connect, RRT*, Informed RRT*, and BIT*. In order to test the ability of planners in terms of finding initial solutions, each planner has unlimited time to find an initial solution. All planners found first solutions for 100 times.

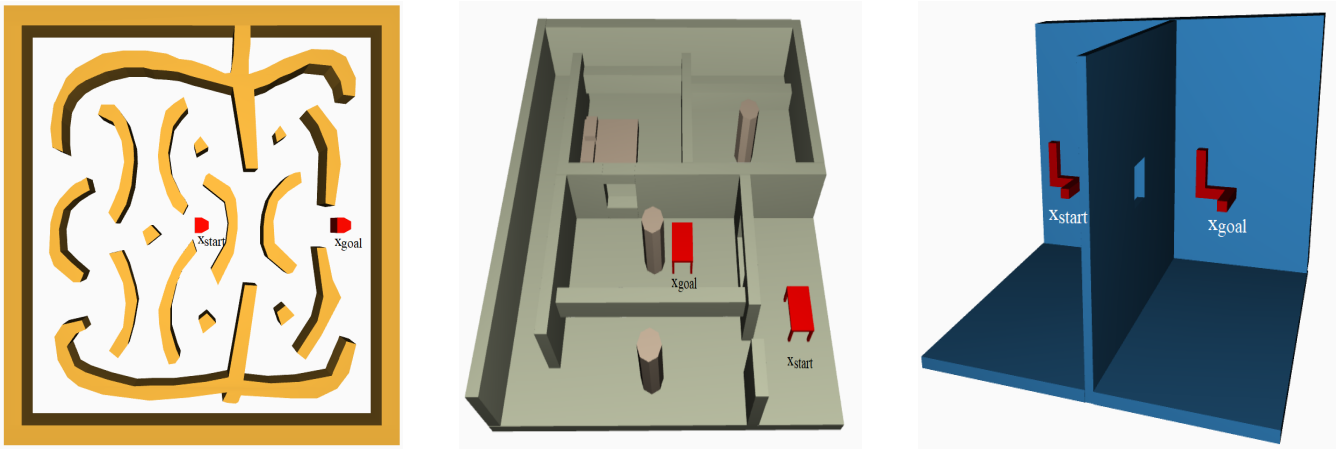
1) BUGTRAP_PLANAR

BugTrap_planar, Fig. 3, is an OMPL App state space with 3 Degree of Freedoms (DoFs), including one rotation and two real vectors (x-axis and y-axis). This simulation is designed to compare the ability of different planners in terms of finding initial solutions.

As can be seen in table 1, Hybrid RRT could return initial solutions with nearly the same amount of time as iRRT-Connect, and faster than other planners.

B. FIND NEAR-OPTIMAL SOLUTIONS

In this simulation, Hybrid RRT has been compared to RRT* and Informed RRT* in terms of finding near-optimal solu-



(a) Maze_planar (b) Home (c) Twistycool

FIGURE 4. The OMPL state spaces used for the simulation. The rigid bodies are highlighted by red color. Each state space has two rigid bodies, which are representing the start location, x_{start}, and the goal location, x_{goal}.

TABLE 1. The obtained results of the planners in BugTrap_planar scenario. Each planner was run for 100 times, and the average time needed to find the initial solutions are presented.

Planner	Average Time (Second)
RRT	0.411341305
RRT-Connect	0.404057516
Hybrid RRT	0.393124017
RRT*	0.994565494
Informed RRT*	0.640909516
BIT*	0.528384085

tions in a limited time. In each scenario, the planners had limited time for finding solutions and optimizing them.

1) MAZE_PLANAR

Maze_planar, Fig. 4a, is an OMPL App state space with 3 Degree of Freedoms (DoFs), including one rotation and two real vectors (x-axis and y-axis). The planners had 5 seconds to solve this problem in each run.

2) HOME

Home is 6DoFs problems (3 coordinate planes (x, y, z) and their rotations (roll, pitch, yaw)) Fig. 4b. In order to return near-optimal solutions, the planners must pass through the window located between x_{start} and x_{goal}. Ten seconds had been given to planners to solve this problem in each run.

3) TWISTYCOOL

Twistycool is a 6DoFs problem, Fig. 4c, which is difficult to be solved due to offering only a small passage to connect x_{start} to x_{goal}. There is a wall in the middle of the map, and it has only a small window, which is the only passage through the wall so that the planners need to find it to solve the problem. Due to the difficulty of this scenario, each planner had 100 seconds to solve the problem in each run.

V. EVALUATION

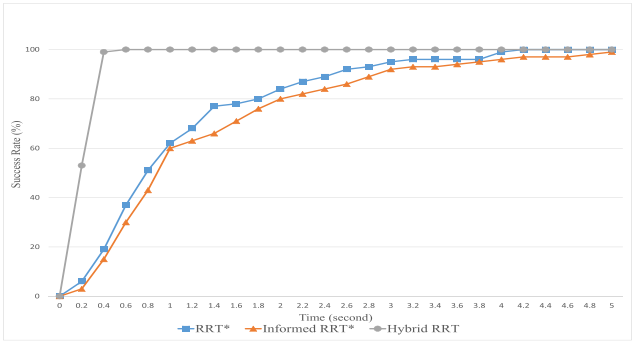
The evaluation of the simulation is represented in this section. In the first simulation, finding initial solutions, it can be seen that the optimized methods (RRT*, Informed RRT*, and BIT*) are slower than non-optimized versions. It is due to the fact that the optimized versions try to rewire their trees from the early stage of planning so that this process takes time and does not let them expand their trees as fast as non-optimized versions. Hybrid RRT postpones the rewiring trees until it finds an initial solution. Then, it starts optimizing its solutions. This ability makes Hybrid RRT a fast planner in terms of finding initial solutions.

The next three simulations were carried out to compare the ability of RRT*, Informed RRT*, and Hybrid RRT in terms of returning near-optimal solutions. The success rates over time of planners in three scenarios are shown in Fig. 5, and the median of path lengths are shown Fig. 6.

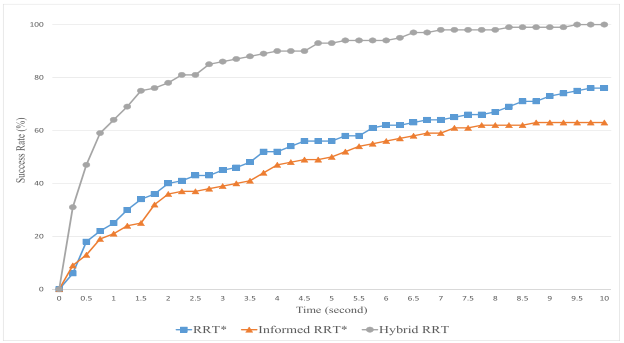
In all the scenarios, Hybrid RRT could achieve 100% success rate, while RRT* and Informed RRT* could only achieve complete success in Maze_planar, which is a 3DoF problem.

Although all the planners could achieve 100% success rate in Maze_planar, they reached it by consuming different amounts of time. Hybrid RRT reached 100% after only 0.4s, while RRT* and Informed RRT* achieved it after approximately 4s. Therefore, Hybrid RRT acted ten times faster than other planners in Maze_planar scenario. In terms of path length, Hybrid RRT could achieve near-optimal solutions faster and RRT* and Informed RRT*.

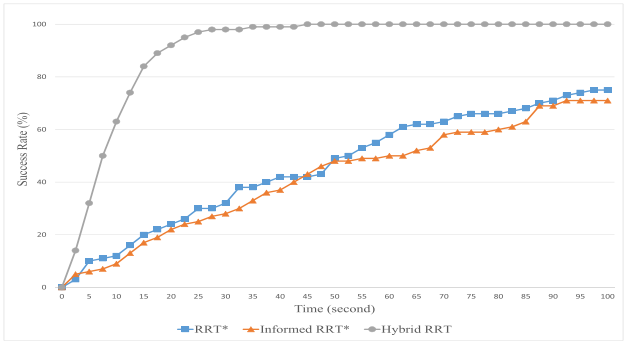
In the Home scenario, Hybrid RRT could reach 100% after about 8s, while RRT* and Informed RRT* could only achieve approximately 75% and 65% after 10s, respectively. Moreover, Hybrid RRT was the fastest planner in terms of optimizing the solution. Hybrid RRT obtained solution cost of 310 before 3s, while Informed RRT* achieved it after 6s, and RRT* could not reach this level at the end of the planning time frame.



(a) Maze_planar

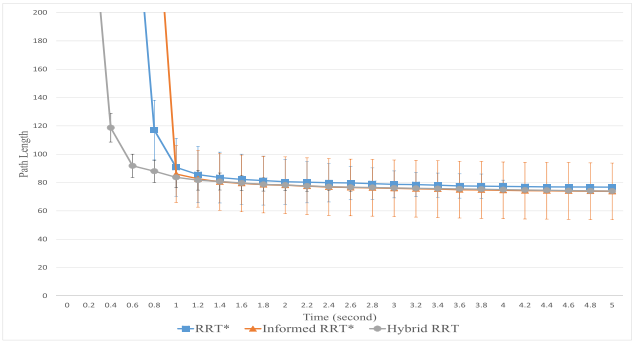


(b) Home

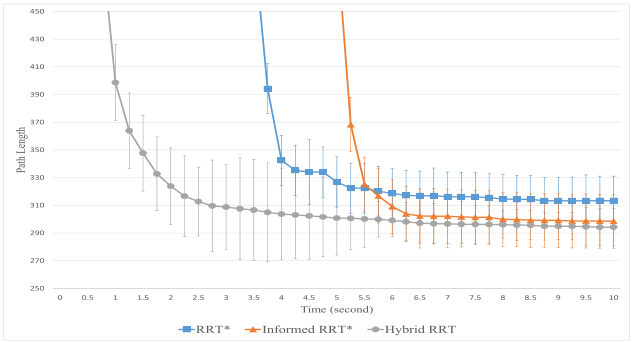


(c) Twistycool

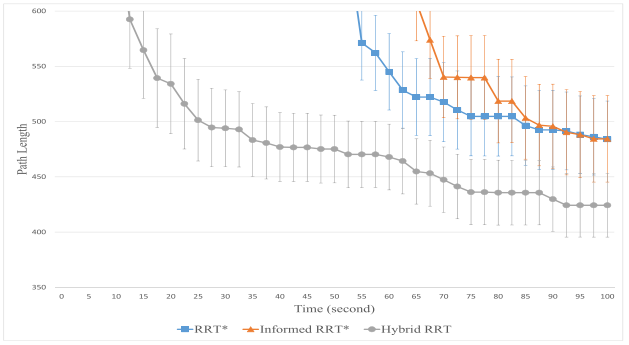
FIGURE 5. The rate of success of the three planners versus time on all the scenarios.



(a) Maze_planar



(b) Home



(c) Twistycool

FIGURE 6. The solution cost versus time of the three planners on all the scenarios. Error bars represent a nonparametric 95% confidence interval for median solution cost.

Twistycool is the most challenging problem to be solved among all the simulated scenarios. Hybrid RRT was able to reach total success after around 45s, while RRT* and Informed RRT* were not able to reach 100%. They could achieve nearly 80% after 100s. RRT* and Informed RRT* could optimize their solution to approximately 470 after 100s, which has been obtained by Hybrid RRT after only 35s.

VI. CONCLUSION

We presented a new path planner, Hybrid RRT, which combines the abilities of bidirectional and unidirectional searches to be able to surpass them. Hybrid RRT implements a bidirectional search to find an initial solution faster than unidirectional methods. Then, it merges its two trees into one so as to optimize it via implementing informed sampling for a single tree.

The simulations show that Hybrid RRT outperforms RRT* and Informed RRT* in terms of the success rate as well as optimization time. It has a higher success rate, and it could return near-optimal solutions faster. These abilities make Hybrid RRT suitable for various motion planning problems.

REFERENCES

[1] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *IEEE Trans. Control Syst. Technol.*, vol. 17, no. 5, pp. 1105–1118, Sep. 2009.

[2] J.-C. Latombe, "Motion planning: A journey of robots, molecules, digital actors, and other artifacts," *Int. J. Robot. Res.*, vol. 18, no. 11, pp. 1119–1128, Nov. 1999.

[3] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 33–55, Mar. 2016.

[4] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[5] D. Gonzalez, J. Perez, V. Milanes, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1135–1145, Apr. 2016.
[6] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier, "Probabilistic navigation in dynamic environment using Rapidly-exploring Random Trees and Gaussian processes," in *Proc. 2008 IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2008, pp. 1056–1062.
[7] X. Lan and S. Di Cairano, "Continuous curvature path planning for semi-autonomous vehicle maneuvers using RRT," in *Proc. Eur.Control Conf. (ECC)*, Jul. 2015.
[8] C. Zammit and E.-J. Van Kampen, "Comparison between a* and RRT algorithms for UAV path planning," in *Proc. 2018 AIAA Guid., Navigat., Control Conf.*, Jan. 2018, p. 1846.
[9] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.
[10] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.
[11] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2014, pp. 2997–3004.
[12] S. M. Lavalle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, May 2001.
[13] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.
[14] S. M. LaValle and J. J. Kuffner, Jr., "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.
[15] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, vol. 2, Apr. 2000, pp. 995–1001.
[16] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.
[17] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Informed sampling for asymptotically optimal path planning," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 966–984, Aug. 2018.
[18] J. Wang, W. Chi, M. Shao, and M. Q.-H. Meng, "Finding a high-quality initial solution for the RRTs algorithms in 2D environments," *Robotica*, vol. 37, no. 10, pp. 1677–1694, Oct. 2019.
[19] J. Wang, C. X.-T. Li, W. Chi, and M. Q.-H. Meng, "Tropistic RRT*: An efficient planning algorithm via adaptive restricted sampling space," in *Proc. IEEE Int. Conf. Inf. Automat. (ICIA)*, Aug. 2018, pp. 1639–1646.
[20] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2015, pp. 3067–3074.
[21] H. Sun and M. Farooq, "Note on the generation of random points uniformly distributed in hyper-ellipsoids," in *Proc. 5th Int. Conf. Inf. Fusion*, Jun. 2003, pp. 489-496.
[22] J. D. Gammell and T. D. Barfoot, "The probability density function of a transformation-based hyperellipsoid sampling technique," 2014, *arXiv:1404.1347*. [Online]. Available: https://arxiv.org/abs/1404.1347
[23] G. Wahba, "A least squares estimate of satellite attitude," *SIAM Rev.*, vol. 7, no. 3, pp. 409–409, Jul. 1965.
[24] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 1478–1483.
[25] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Automat. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.
[26] Quigley, Morgan, "ROS: An open-source Robot Operating System," in *ICRA Workshop Open Source Softw.*, 2009, vol. 3, nos. 3–2, p. 5.

REZA MASHAYEKHI received a bachelor's degree in electronics and a master's in electrical engineering (Mechatronics and Automatic Control). He is currently pursuing a Ph.D. degree in Computer Science at the University of Malaya. His research interests are Robotics, Motion Planning, Automatic Control, and Embedded Systems.

MOHD YAMANI IDNA IDRIS received the Ph.D. degree in electrical engineering and has vast experience in research. He is currently an Associate Professor with the Faculty of Computer Science and Information Technology, University of Malaya. His research interest are in the areas of robotics, embedded systems, sensor networks, and signal/image processing.

MOHAMMAD HOSSEIN ANISI (Senior Member, IEEE) is currently an Assistant Professor with the School of Computer Science and Electronic Engineering, University of Essex, and head of the Internet of Everything Laboratory. Prior to that, he worked as a Senior Research Associate with the University of East Anglia, U.K., and a Senior Lecturer with the University of Malaya, Malaysia, where he received Excellent Service Award for his achievements. As a computer scientist, he has designed and developed novel architectures and routing protocols for the Internet of Things (IoT) enabling technologies including wireless sensor and actuator networks, vehicular networks, heterogeneous networks, body area networks, and his research results have directly contributed to the technology industry. He has a strong collaboration with industry and working with several companies in the U.K., with the focus on monitoring and automation systems based on IoT concept capable of reliable and seamless generation, transmission, processing, and demonstration of data. He has published more than 80 articles in high-quality journals and several conference papers and won two medals for his innovations from PECIPTA 2015 and IIDEX 2016 expositions. His research has focused specifically on real world application domains such as energy management, transportation, healthcare and other potential life domains. He has received several international and national funding awards for his fundamental and practical research as PI and Co-I.

He is a Senior Member of the Institute of Research Engineers and Doctors (the IRED), a Member of ACM, IEEE Council on RFID, the IEEE Sensors Council, the IEEE Systems Council and International Association of Engineers (IAENG). He is a Fellow of Higher Education Academy. He is also a Technical Committee Member of Finnish-Russian University Cooperation in Telecommunications (FRUCT). He has been also serving as an executive/technical committee member of several conferences. He is an Associate Editor of a number of journals including IEEE Access, the *Ad Hoc & Sensor Wireless Networks*, the *IET Wireless Sensor Systems*, *International Journal of Distributed Sensor Networks*. the *KSII Transactions on Internet and Information Systems* journals and *Journal of Sensor and Actuator Networks*. He has been a guest editor of special issues of the journals and Lead organizer of special sessions and workshops at the IEEE conferences such as the IEEE CAMAD, the IEEE PIMRC, and the IEEE VTC.

ISMAIL AHMEDY received the Ph.D. degree in computer science. He is currently a Senior Lecturer with the Department of Computer Systems and Technology, Faculty Science Computer and Information Technology, University of Malaya. His areas of expertise are underwater acoustic sensor networks, embedded systems, and wireless sensor networks.