HySST: An Asymptotically Near-Optimal Motion Planning Algorithm for Hybrid Systems*

Nan Wang and Ricardo G. Sanfelice

***Abstract*— This paper proposes a stable sparse rapidly-exploring random trees (SST) algorithm to solve the optimal motion planning problem for hybrid systems. At each iteration, the proposed algorithm, called HySST, selects a vertex with minimal cost among all the vertices within the neighborhood of a random sample, subsequently extending the search tree through flow or jump, which is also chosen randomly when both regimes are possible. In addition, HySST maintains a static set of witness points where all vertices within each witness's neighborhood are pruned, except for the ones with lowest cost. We show that HySST is asymptotically near-optimal, namely, the probability of failing to find a motion plan with cost close to the optimal approaches zero as the number of iterations of the algorithm increases to infinity. The proposed algorithm is applied to a collision-resilient tensegrity multicopter system so as to highlight its generality and computational features.**

I. Introduction

Motion planning consists of finding a state trajectory and associated inputs that connect the initial and final state sets while satisfying the dynamics of the systems and given safety requirements. Motion planning for purely continuous-time systems and purely discrete-time systems has been well studied in the literature. In recent years, several feasible motion planning algorithms have been developed, including graph search algorithms [1], artificial potential [2] and fluid [3] field methods, and sampling-based algorithms. The sampling-based algorithms have drawn much attention in recent years because of their fast exploration speed for high dimensional problems and theoretical guarantees; specially, probabilistic completeness. Two popular sampling-based algorithms are the probabilistic roadmap (PRM) algorithm [4] and the rapidly-exploring random tree (RRT) algorithm [5].

A feasible solution is not sufficient in most applications as the quality of the solution returned by the motion planning algorithms is critical. It has been shown in [6] that the solution returned by RRT converges to a sub-optimal solution. Therefore, variants of PRM and RRT, such as PRM* and RRT* [7], have been developed to solve optimal motion planning problems with guaranteed asymptotic optimality. However, both PRM* and RRT* require a steering function returning the solution of a two-point boundary value problem (TPBVP). Unfortunately, solutions to TPBVPs are difficult to generate for most dynamical systems, which prevents them from being widely applied. On the other hand, the stable sparse RRT (SST) algorithm [8] does not require a steering function and is guaranteed to be asymptotically near optimal, which means that the probability of finding a solution that has a cost close to the minimal cost converges to one as the number of iterations approaches infinity.

The aforementioned motion planning algorithms have been widely applied to purely continuous-time and purely discrete-time systems. However, much fewer efforts have been devoted to motion planning for systems with combined continuous and discrete behaviors, which we refer to as *hybrid systems*, such as walking robots [9], quadrupeds [10], unmanned aerial underwater vehicles [11], and collision resilient aerial vehicles [12]. In our previous work [13], a feasible motion planning problem is formulated for hybrid system given in terms of hybrid equations as in [14], which is a general framework that captures a broad class of hybrid systems. In [13], a probabilistically complete RRT algorithm for hybrid systems is designed to solve the feasible motion planning problems for such systems.

In this paper, we formulate the optimal motion planning problem for hybrid systems and design an SST-type algorithm with the goal of assuring asymptotic optimality of the solution. The proposed algorithm, called HySST, incrementally constructs a search tree rooted in the initial state set toward the random samples. At first, HySST draws samples from the state space. Then, it selects a vertex such that the state associated with this vertex is within a ball centered at the random sample and has minimal cost. Next, HySST propagates the state trajectory from the selected vertex, and adds a new vertex and edge from the propagated trajectory. In addition, HySST maintains a static set of state points, called *witnesses*, to represent the explored regions, and prunes all the vertices within each witness's neighborhood except for those with lowest cost. We show that, under mild assumptions, HySST is asymptotically near-optimal. To the authors' best knowledge, HySST is the first optimal RRT-type algorithm for hybrid systems. The proposed algorithm is illustrated in a collision-resilient tensegrity multicopter system.

The remainder of the paper is organized as follows. Section II presents notation and preliminaries. Section III presents the problem statement and introduces an example. Section IV presents the HySST algorithm. Section V presents the asymptotically near optimal result for HySST. Section VI illustrates HySST in the said example. Due to space constraints, proofs of the results will be published elsewhere.

*Research by R. G. Sanfelice partially supported by NSF Grants no. CNS-2039054 and CNS-2111688, by AFOSR Grants nos. FA9550-19-1-0169, FA9550-20-1-0238, FA9550-23-1-0145, and FA9550-23-1-0313, by AFRL Grant nos. FA8651-22-1-0017 and FA8651-23-1-0004, by ARO Grant no. W911NF-20-1-0253, and by DoD Grant no. W911NF-23-1-0158.

Nan Wang and Ricardo G. Sanfelice are with the Department of Electrical and Computer Engineering, University of California, Santa Cruz, CA 95064, USA; nanwang@ucsc.edu, ricardo@ucsc.edu

II. Notation and Preliminaries

A. Notation

The real numbers are denoted as $\mathbb{R}$ and its nonnegative subset is denoted as $\mathbb{R}_{\geq 0}$. The set of nonnegative integers is denoted as $\mathbb{N}$. The notation $\operatorname{int} S$ denotes the interior of the set S. The notation $\overline{S}$ denotes the closure of the set S. The notation ∂S denotes the boundary of the set S. The notation $\mathbb{B}$ denotes the closed unit ball in the Euclidean norm. Given vectors u and v, $[u^\top, v^\top]^\top$ is equivalent to (u, v).

B. Preliminaries

A hybrid system $\mathcal{H}$ with inputs is modeled as [14]

$$\mathcal{H}: \begin{cases} \dot{x} = f(x,u) & (x,u) \in C \\ x^+ = g(x,u) & (x,u) \in D \end{cases} \tag{1}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^m$ is the input, $C \subset \mathbb{R}^n \times \mathbb{R}^m$ represents the flow set, $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ represents the flow map, $D \subset \mathbb{R}^n \times \mathbb{R}^m$ represents the jump set, and $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ represents the jump map, respectively. The continuous evolution of x is captured by the flow map f. The discrete evolution of x is captured by the jump map g. The flow set C collects the points where the state may evolve continuously. The jump set D collects the points where jumps may occur. Given a flow set C, the set $U_C := \{u \in \mathbb{R}^m : \exists x \in \mathbb{R}^n \text{ s.t. } (x,u) \in C\}$ includes all possible input values that can be applied during flows. Similarly, given a jump set D, the set $U_D := \{u \in \mathbb{R}^m : \exists x \in \mathbb{R}^n \text{ s.t. } (x,u) \in D\}$ includes all possible input values that can be applied at jumps. These sets satisfy $C \subset \mathbb{R}^n \times U_C$ and $D \subset \mathbb{R}^n \times U_D$. Given a set $K \subset \mathbb{R}^n \times U_\star$, where $\star$ is either C or D, we define $\Pi_\star(K) := \{x : \exists u \in U_\star \text{ s.t. } (x,u) \in K\}$ as the projection of K onto $\mathbb{R}^n$, and define $C' := \Pi_C(C)$ and $D' := \Pi_D(D)$.

In addition to ordinary time $t \in \mathbb{R}_{\geq 0}$, we employ $j \in \mathbb{N}$ to denote the number of jumps of the evolution of x and u for $\mathcal{H}$ in (1), leading to hybrid time (t, j) for the parameterization of its solutions and inputs. The domain of a solution to $\mathcal{H}$ is given by a hybrid time domain. A hybrid time domain is defined as a subset E of $\mathbb{R}_{\geq 0} \times \mathbb{N}$ that, for each $(T, J) \in E$, $E \cap ([0, T] \times \{0, 1, ..., J\})$ can be written as $\cup_{j=0}^{J}([t_j, t_{j+1}], j)$ for some finite sequence of times $0 = t_0 \leq t_1 \leq t_2 \leq ... \leq t_{J+1} = T$. A hybrid arc $\phi : \operatorname{dom}\phi \to \mathbb{R}^n$ is a function on a hybrid time domain that, for each $j \in \mathbb{N}$, $t \mapsto \phi(t, j)$ is locally absolutely continuous on each interval $I^j := \{t : (t, j) \in \operatorname{dom}\phi\}$ with nonempty interior. A solution pair to a hybrid system is defined as follows.

Definition 2.1 (Solution pair to a hybrid system [14]): Given a pair of functions $\phi : \operatorname{dom}\phi \to \mathbb{R}^n$ and $u : \operatorname{dom} u \to \mathbb{R}^m$, (ϕ, u) is a solution pair to (1) if $\operatorname{dom}(\phi, u) := \operatorname{dom}\phi = \operatorname{dom} u$ is a hybrid time domain, $(\phi(0,0), u(0,0)) \in \overline{C} \cup D$, and the following hold:

1) For each $j \in \mathbb{N}$ such that I^j has nonempty interior,
   a) $t \mapsto \phi(t, j)$ is locally absolutely continuous,
   b) $(\phi(t, j), u(t, j)) \in C$ for all $t \in \operatorname{int} I^j$,
   c) the function $t \mapsto u(t, j)$ is Lebesgue measurable and locally bounded,
   d) for almost all $t \in I^j$, $\frac{\mathrm{d}\phi(t,j)}{\mathrm{d}t} = f(\phi(t, j), u(t, j))$.
2) For each $(t, j) \in \operatorname{dom}(\phi, u)$ such that $(t, j+1) \in \operatorname{dom}(\phi, u)$, $(\phi(t, j), u(t, j)) \in D$, $\phi(t, j+1) = g(\phi(t, j), u(t, j))$.

III. Problem Statement

The formulation of the feasible motion planning problem for hybrid systems can be found in [13, Problem 1] and is denoted as $\mathcal{P} = (X_0, X_f, X_u, (C, f, D, g))$, where the initial state set is denoted as $X_0 \subset \mathbb{R}^n$, the final state set is denoted as $X_f \subset \mathbb{R}^n$, and the unsafe set is denoted as $X_u \subset \mathbb{R}^n \times \mathbb{R}^m$. Let $\hat{\mathcal{S}}_{\mathcal{H}}$ denote the set of all solution pairs to $\mathcal{H}$. Let $\hat{\mathcal{S}}^{\phi}_{\mathcal{H}}$ denote the set of state trajectories of all the solution pairs in $\hat{\mathcal{S}}_{\mathcal{H}}$. The optimal motion planning problem for hybrid systems consists of finding a feasible motion plan with minimum cost [7, Problem 3].

Problem 1: (Optimal motion planning) Given a motion planning problem $\mathcal{P} = (X_0, X_f, X_u, (C, f, D, g))$ and a cost functional $c : \hat{\mathcal{S}}^{\phi}_{\mathcal{H}} \to \mathbb{R}_{\geq 0}$, find a feasible motion plan (ϕ^*, u^*) to $\mathcal{P}$ such that $(\phi^*, u^*) = \arg\min_{(\phi,u)\in\hat{\mathcal{S}}_{\mathcal{H}}} c(\phi)$. Given sets X_0, X_f, and X_u, a hybrid system $\mathcal{H}$ with data (C, f, D, g), and a cost functional c, an optimal motion planning problem $\mathcal{P}^*$ is formulated as $\mathcal{P}^* = (X_0, X_f, X_u, (C, f, D, g), c)$.

Problem 1 is illustrated in the following example.

Example 3.1: (Collision-resilient tensegrity multicopter system [15]) Consider a planar collision-resilient tensegrity multicopter that is resilient to collisions with a wall. The state of the multicopter involves the position vector $p := (p_x, p_y) \in \mathbb{R}^2$, the velocity vector $v := (v_x, v_y) \in \mathbb{R}^2$, and the acceleration vector $a := (a_x, a_y) \in \mathbb{R}^2$, where, respectively, p_x and p_y denote the position, v_x and v_y denote the velocity, and a_x and a_y denote the acceleration along the x-axis and y-axis. The state of the system is $x := (p, v, a) \in \mathbb{R}^6$ and its input is $u := (u_x, u_y) \in \mathbb{R}^2$ which represents the effect of the torque. The environment is assumed to be known. Define the walls as the region $\mathcal{W} \subset \mathbb{R}^2$, which is a closed set represented by the blue rectangles in Figure 1. Flow is allowed when the multicopter is in $C := \overline{(\mathbb{R}^2 \backslash \mathcal{W})} \times \mathbb{R}^4 \times \mathbb{R}^2$, which defines the flow set. The dynamics of the multicopter when no collision occurs is captured as $\dot{x} = (v, a, u) =: f(x, u)$ $(x, u) \in C$.

At collisions, the position is assumed to remain constant. To model the change of v, denote the velocity component of $v = (v_x, v_y)$ that is normal to the wall as v_N and the velocity component that is tangential to the wall as v_T. Then, the velocity component v_N after the jump is modeled as $v_N^+ = -\lambda v_N =: \tilde{g}_N(v)$ where $\lambda \in (0, 1)$ is the coefficient of restitution. The velocity component v_T after the jump is modeled as $v_T^+ = v_T + \kappa(-\lambda - 1)\arctan\frac{v_T}{v_N} v_N =: \tilde{g}_T(v)$, where $\kappa \in \mathbb{R}$ is a constant; see [15]. Denoting the projection of the updated vector (v_N^+, v_T^+) onto the x-axis as $\Pi_x(v_N^+, v_T^+)$ and the projection of the updated vector (v_N^+, v_T^+) onto the y-axis as $\Pi_y(v_N^+, v_T^+)$, we

have $v^+ = (\Pi_x(\tilde{g}_N(v), \tilde{g}_T(v)), \Pi_y(\tilde{g}_N(v), \tilde{g}_T(v))) =: \tilde{g}(v)$. We assume that $a^+ = 0$, which, through a post-impact hovering maneuver, can be mitigated in the control layer. The discrete dynamics capturing the collision process is modeled as $x^+ = (p, \tilde{g}(v), 0) =: g(x,u)$ $(x,u) \in D$. Jumps are allowed when the multicopter is on the wall surface with positive velocity towards the wall. Hence, the jump set is $D := \{((p,v,a),u) \in \mathbb{R}^6 \times \mathbb{R}^2 : p \in \partial\mathcal{W}, v_N \leq 0\}$.

Given the initial state set as $X_0 = \{(1,2,0,0,0,0)\}$, the final state set as $X_f = \{(5,4)\} \times \mathbb{R}^4$, and the unsafe set as $X_u = \{(x,u) \in \mathbb{R}^6 \times \mathbb{R}^2 : \sqrt{(p_x-5)^2 + (p_y-3)^2}| \leq 0.3\}$ which represents the green ball in Figure 1 that is forbidden to fly into or collide with, an instance of the optimal motion planning problem for the collision-resilient tensegrity multicopter system is to find the motion plan with minimal hybrid time. To capture the hybrid time domain information, an auxiliary state $\tau \in \mathbb{R}_{\geq 0}$ representing the ordinary time and an auxiliary state $k \in \mathbb{N}$ representing the number of jumps associated to collisions are included. The resulting hybrid system $\overline{\mathcal{H}} := (\overline{C}, \overline{f}, \overline{D}, \overline{g})$ with state $\overline{x} := (x, \tau, k) \in \mathbb{R}^2 \times \mathbb{R}_{\geq 0} \times \mathbb{N}$, input $u \in \mathbb{R}^2$, and data $\overline{C} := \big\{(\overline{x}, u) \in \mathbb{R}^2 \times \mathbb{R}_{\geq 0} \times \mathbb{N} \times \mathbb{R} : (x,u) \in C\big\}$; $\overline{f}(\overline{x}, u) := (f(x,u), 1, 0)$ for each $(\overline{x}, u) \in \overline{C}$; $\overline{D} := \big\{(\overline{x}, u) \in \mathbb{R}^2 \times \mathbb{R}_{\geq 0} \times \mathbb{N} \times \mathbb{R} : (x,u) \in D\big\}$; $\overline{g}(\overline{x}, u) := (g(x,u), \tau, k+1)$ for each $(\overline{x}, u) \in \overline{D}$ with the X_0, X_f, and X_u extended as $\overline{X}_0 := X_0 \times \{0\} \times \{0\}$, $\overline{X}_f := X_f \times \mathbb{R}_{\geq 0} \times \mathbb{N}$, $\overline{X}_u := X_u \times \mathbb{R}_{\geq 0} \times \mathbb{N}$. Then, with $\overline{\phi} = (\phi, \tau, k)$ being a state trajectory of the solution pair to $\overline{\mathcal{H}}$, the cost functional c can be defined as $c(\overline{\phi}) := \tau(T,J) + k(T,J)$, where $(T,J) = \max \operatorname{dom} \overline{\phi}$. The resulting optimal motion planning problem is defined as $\mathcal{P}^* = (\overline{X}_0, \overline{X}_f, \overline{X}_u, (\overline{C}, \overline{f}, \overline{D}, \overline{g}), c)$.

In the forthcoming Example 6.1, we employ HySST to solve this motion planning problem.

IV. HySST: An Asymptotically Near-Optimal Motion Planning Algorithm for Hybrid Systems

A. Overview

HySST searches for the optimal motion plan by incrementally constructing a search tree. The search tree is a pair $\mathcal{T} = (V, E)$, where V is a set whose elements are called vertices, denoted v, and E is a set of paired vertices whose elements are called edges, denoted e. A path in $\mathcal{T}$ is a sequence of vertices $p = (v_1, v_2, ..., v_k)$ such that $(v_i, v_{i+1}) \in E$ for all $i \in \{1, 2, ..., k-1\}$. For details about the search tree, see [13, Section 4.A].

Each vertex $v \in V$ in the search tree $\mathcal{T} = (V, E)$ is associated with a state value of $\mathcal{H}$, denoted $\overline{x}_v$, and a cost value that, via addition, compounds the cost from the root vertex up to the vertex v, denoted $\overline{c}_v$. Each edge $e \in E$ in the search tree $\mathcal{T} = (V, E)$ is associated with a solution pair to $\mathcal{H}$, denoted $\overline{\psi}_e$. The solution pair that the path $p = (v_1, v_2, ..., v_k)$ represents is the concatenation of all those solution pairs associated with the edges therein, namely, $\widetilde{\psi}_p := \overline{\psi}_{(v_1,v_2)} | \overline{\psi}_{(v_2,v_3)}| \ ... \ | \overline{\psi}_{(v_{k-1},v_k)}$ where $\widetilde{\psi}_p$ denotes the solution pair associated with the path p. For details on such concatenation, see [13, Definition 2.2].

HySST requires a library of possible inputs. The input library $(\mathcal{U}_C, \mathcal{U}_D)$ includes the input signals that can be applied during flows (collected in $\mathcal{U}_C$) and the input values that can be applied at jumps (collected in $\mathcal{U}_D$).

HySST selects the vertex associated with the lowest cost within the vicinity of a randomly selected state. This vicinity is referred to as *random state neighborhood* and defined by a ball of radius $\delta_{BN} \in \mathbb{R}_{>0}$. Then, HySST employs a pruning process to decrease the number of vertices in the search tree. This pruning operation is implemented by maintaining a *witness state set*, denoted S, such that all the vertices within the vicinity of the witnesses are deleted except the ones with lowest cost. This vicinity is referred to as *closest witness neighborhood* and defined by a ball of radius $\delta_s \in \mathbb{R}_{>0}$, For every witness s kept in S, a single vertex in the tree represents that witness. Such a vertex is stored in $s.rep$ for each witness $s \in S$. Note that a vertex, say, v_a, may be associated with a higher cost than other vertices within the same witness's neighborhood, but has a child vertex, say, v_b, associated with the lowest cost compared with other vertices in the same witness's neighborhood. In this case, v_a should not be removed from the search tree because, if it is removed, then all of its child vertices, including v_b with the lowest cost, are consequently removed. However, even v_a is not removed, it will not be selected, and, therefore, will be kept in a separate set called *inactive vertex set*, denoted $V_{inactive}$. On the other hand, the vertices that are not pruned are stored in a set called the *active vertex set*, denoted V_{active}.

Next, we introduce the main steps executed by HySST. Given the optimal motion planning problem $\mathcal{P}^* = (X_0, X_f, X_u, (C, f, D, g), c)$ and the input library $(\mathcal{U}_C, \mathcal{U}_D)$, HySST performs the following steps:

- **Step 1**: Initialize a search tree $\mathcal{T} = (V, E)$ by sampling a finite number of points from X_0. For each sampling point x_0, add a vertex v_0 and assign $\overline{x}_{v_0} \leftarrow x_0$. Initialize E by $E \leftarrow \emptyset$. Initialize the witness state set $S \subset \mathbb{R}^n$ by $S \leftarrow \emptyset$. For each $v \in V$ such that $|\overline{x}_v - \overline{x}_{v'}| > \delta_s$ for all $v' \in V \backslash v$, add the witness state $s = \overline{x}_v$ to S and set the representative of s as $s.rep \leftarrow v$. Initialize the active vertices set V_{active} by $V_{active} \leftarrow \{s.rep \in V : s \in S\}$. Initialize the inactive vertices set $V_{inactive}$ by $V_{inactive} \leftarrow \emptyset$.
- **Step 2**: Randomly select flow regime or jump regime for the evolution of $\mathcal{H}$.
- **Step 3**: Randomly select a point x_{rand} from C' (D') if the flow (respectively, jump) regime is selected in **Step 2**.
- **Step 4**: Find all the vertices in V_{active} associated with the state values that are within δ_{BN} to x_{rand} and collect them in the set V_{BN}. Then, find vertex in V_{BN} that has minimal cost, denoted v_{cur}. If no vertex is collected in V_{BN}, then find vertex in the search tree that has minimal distance to x_{rand} and assign it to v_{cur}.
- **Step 5**: Randomly select an input signal (respectively, value) from $\mathcal{U}_C$ (respectively, $\mathcal{U}_D$) if $\overline{x}_{v_{cur}} \in$

$C' \backslash D'$ (respectively, $\overline{x}_{v_{cur}} \in D' \backslash C'$). Then, compute a solution pair denoted $\psi_{new} = (\phi_{new}, u_{new})$ starting from $\overline{x}_{v_{cur}}$ with the selected input applied via flow (respectively, jump). If $\overline{x}_{v_{cur}} \in D' \cap C'$, a random process is employed to decide whether to proceed the computation with flow or jump. Denote the final state of ϕ_{new} as x_{new}. Compute the cost at x_{new}, denoted c_{new}, by $c_{new} \leftarrow \overline{c}_{v_{cur}} + c(\phi_{new})$. If ψ_{new} intersects with X_u, then go to **Step 2**.

Step 6: Find the witness in S that is closest to x_{new}, denoted s_{near}, and proceed as follows:

- If x_{new} is not in the closest witness neighborhood of s_{near}, namely, $|s_{near} - x_{new}| > \delta_s$, then add a vertex v_{new} associated with x_{new} to V_{active} and an edge (v_{cur}, v_{new}) associated with ψ_{new} to E. Add a new witness to S and set its representative as v_{new}. Then, go to **Step 2**.
- If $|s_{near} - x_{new}| \leq \delta_s$,
  - if $\overline{c}_{s_{near}.rep} > c_{new}$, add a vertex v_{new} associated with x_{new} to V_{active} and an edge (v_{cur}, v_{new}) associated with ψ_{new} to E. Then, update the representative of s_{near} with v_{new} and prune the vertex, say, v_{pre_near} which is previously witnessed by s_{near}. If v_{pre_near} is an active vertex, then add v_{pre_near} to $V_{inactive}$. Otherwise, remove v_{pre_near} and all its child vertices from the search tree. Then, go to **Step 2**.
  - if $\overline{c}_{s_{near}.rep} \leq c_{new}$, go to **Step 2** directly.

B. HySST Algorithm

Following the overview above, the proposed algorithm is given in Algorithm 1. The inputs of Algorithm 1 are the problem $\mathcal{P}^* = (X_0, X_f, X_u, (C, f, D, g), c)$, the input library $(\mathcal{U}_C, \mathcal{U}_D)$, a parameter $p_n \in (0, 1)$, which tunes the probability of evolving with the flow regime or the jump regime, an upper bound $K \in \mathbb{N}_{>0}$ for the number of iterations to execute, and two tunable sets $X_c \supset \overline{C'}$ and $X_d \supset D'$, which act as constraints in finding a closest vertex to x_{rand}. In addition, HySST requires parameters δ_{BN} and δ_s to tune the radius of random state neighborhood and closest witness neighborhood, respectively. Each function in Algorithm 1 is defined next.

1) $\mathcal{T}$.init(X_0): The function call $\mathcal{T}$.init is used to initialize a search tree $\mathcal{T} = (V, E)$. It randomly selects a finite number of points from X_0. For each sampling point x_0, a vertex v_0 associated with x_0 is added to V. At this step, no edge is added to E.

2) return $\leftarrow$ is_vertex_locally_the_best$(x, cost, S, \delta_s)$: The function call is_vertex_locally_the_best describes the conditions under which the state x is considered for addition to the search tree as is shown in Algorithm 2. First, this function searches for the closest witness s_{new} to x from the witness set S (line 1). If the closest witness distance to x is larger than δ_s, a new witness is added to S (lines 2 - 6). If s_{new} is just added as a witness or *cost* is less than the cost of the closest witness's representatives (line 7), then the state x with the cost *cost* is locally optimal and a true signal is returned (line 8). Otherwise, a false signal is returned.

Algorithm 1 HySST algorithm

Input: $X_0, X_f, X_u, c, \mathcal{H} = (C, f, D, g), (\mathcal{U}_C, \mathcal{U}_D), p_n \in (0, 1)$, $K \in \mathbb{N}$, X_c, X_d, δ_{BN} and δ_s

1: $\mathcal{T}$.init(X_0);
2: $V_{active} \leftarrow V$, $V_{inactive} \leftarrow \emptyset$, $S \leftarrow \emptyset$;
3: **for all** $v_0 \in V$ **do**
4: **if** is_vertex_locally_the_best$(\overline{x}_{v_0}, 0, S, \delta_s)$ **then**
5: $(S, V_{active}, V_{inactive}, E) \leftarrow$ prune_dominated_vertices$(v_0, S, V_{active}, V_{inactive}, E)$
6: **end if**
7: **end for**
8: **for** $k = 1$ to K **do**
9: randomly select a real number r from $[0, 1]$;
10: **if** $r \leq p_n$ **then**
11: $x_{rand} \leftarrow$ random_state$(\overline{C'})$;
12: $v_{cur} \leftarrow$ best_near_selection$(x_{rand}, V_{active}, \delta_{BN}, X_c)$;
13: **else**
14: $x_{rand} \leftarrow$ random_state(D');
15: $v_{cur} \leftarrow$ best_near_selection$(x_{rand}, V_{active}, \delta_{BN}, X_d)$;
16: **end if**
17: $(\text{is\_a\_new\_vertex\_generated}, x_{new}, \psi_{new}, cost_{new}) \leftarrow$ new_state$(v_{cur}, (\mathcal{U}_C, \mathcal{U}_D), \mathcal{H}, X_u)$
18: **if** is_a_new_vertex_generated & is_vertex_locally_the_best$(x_{new}, cost_{new}, S, \delta_s)$ **then**
19: $v_{new} \leftarrow V_{active}$.add_vertex$(x_{new}, cost_{new})$;
20: E.add_edge$(v_{cur}, v_{new}, \psi_{new})$;
21: $(S, V_{active}, V_{inactive}, E) \leftarrow$ prune_dominated_vertices$(v_{new}, S, V_{active}, V_{inactive}, E)$;
22: **end if**
23: **end for**
24: **return** $\mathcal{T}$;

Algorithm 2 is_vertex_locally_the_best$(x, cost, S, \delta_s)$

1: $s_{new} \leftarrow$ nearest(S, x);
2: **if** $|x - s_{new}| > \delta_s$ **then**
3: $s_{new} \leftarrow x$
4: $s_{new}.rep \leftarrow NULL$
5: $S \leftarrow S \cup \{s_{new}\}$;
6: **end if**
7: **if** $s_{new}.rep == NULL$ or $cost < \overline{c}_{s_{new}.rep}$ **then**
8: **return** true;
9: **end if**
10: **return** false;

3) $(S, V_{active}, V_{inactive}, E) \leftarrow$ prune_dominated_vertices$(v, S, V_{active}, V_{inactive}, E)$: The function call prune_dominated_vertices describes the pruning process as in Algorithm 3. First, this function searches for the witnesses s_{new} that are closest to $\overline{x}_v$ and their representatives v_{peer} (lines 1 - 2). Then, v_{peer} is moved from V_{active} to $V_{inactive}$ (lines 4 - 5) and, consequently, v replaces v_{peer} as the representative of s_{new} (line 7). Then, v_{peer} is removed from $V_{inactive}$, along with all its parent vertices that are in $V_{inactive}$ and that have no child vertices after the removal of v_{peer} (lines 8 - 13).

4) $x_{rand} \leftarrow$ random_state(S): The function call random_state randomly selects a point from $S \subset \mathbb{R}^n$.

5) $v_{cur} \leftarrow$ best_near_selection$(x_{rand}, V_{active}, \delta_{BN}, X_\star)$: The function call best_near_selection searches

for a vertex v_{cur} in the active vertex set V_{active} such that its associated state value is in the intersection between the set $X_\star$ and $x_{rand} + \delta_{BN}\mathbb{B}$, and has minimal cost, where $\star$ is either c or d. This function is implemented by solving

Algorithm 3 $(S, V_{active}, V_{inactive}, E) \leftarrow$ `prune_dominated_vertices`$(v, S, V_{active}, V_{inactive}, E)$

$s_{new} \leftarrow$ `nearest`$(S, \overline{x}_v)$;
$v_{peer} \leftarrow s_{new}.rep$;
if $v_{peer}! = NULL$ **then**
  $V_{active} \leftarrow V_{active} \backslash \{v_{peer}\}$;
  $V_{inactive} \leftarrow V_{inactive} \cup \{v_{peer}\}$;
end if
$s_{new}.rep \leftarrow v$;
while `isleaf`(v_{peer}) and $v_{peer} \in V_{inactive}$ **do**
  $v_{parent} \leftarrow$ `parent`(v_{peer});
  $E \leftarrow E \backslash \{(v_{parent}, v_{peer})\}$;
  $V_{inactive} \leftarrow V_{inactive} \backslash v_{peer}$;
  $v_{peer} \leftarrow v_{parent}$;
end while

the following optimization problem.

Problem 2: Given $x_{rand} \in \mathbb{R}^n$, a radius $\delta_{BN} > 0$ of the random state neighborhood, a tunable state constraint set $X_\star$, and an active vertex set V_{active}, solve

$$\begin{aligned} \underset{v \in V_{active}}{\arg\min} \quad & \overline{c}_v \\ \text{s.t.} \quad & |\overline{x}_v - x_{rand}| \leq \delta_{BN} \\ & \overline{x}_v \in X_\star. \end{aligned}$$

Data of Problem 2 comes from the arguments of `best_near_selection` function call. This optimization problem is solved by traversing all the vertices in V_{active}.

6) $(is\_a\_new\_vertex\_generated, x_{new}, \psi_{new}, cost_{new}) \leftarrow new\_state(v_{cur}, (\mathcal{U}_C, \mathcal{U}_D), \mathcal{H}, X_u)$*:* If $\overline{x}_{v_{cur}} \in \overline{C'} \backslash D'$ (respectively, $\overline{x}_{v_{cur}} \in D' \backslash \overline{C'}$), the function call *new_state* generates a new solution pair ψ_{new} to the hybrid system $\mathcal{H}$ starting from $\overline{x}_{v_{cur}}$ by applying an input signal $\tilde{u}$ (respectively, an input value u_D) randomly selected from $\mathcal{U}_C$ (respectively, $\mathcal{U}_D$). If $\overline{x}_{v_{cur}} \in \overline{C'} \cap D'$, then this function generates ψ_{new} by randomly selecting flow or jump. The final state of $\psi_{new} = (\phi_{new}, u_{new})$ is denoted as x_{new}. The cost $cost_{new}$ at x_{new} is computed by $cost_{new} \leftarrow \overline{c}_{v_{cur}} + c(\phi_{new})$.

After ψ_{new} and x_{new} are generated, the function `new_state` checks if there exists $(t, j) \in \operatorname{dom} \psi_{new}$ such that $\psi_{new}(t, j) \in X_u$. If so, we have `is_a_new_vertex_generated` $\leftarrow$ `false`. Otherwise, we have `is_a_new_vertex_generated` $\leftarrow$ `true`.

7) $v_{new} \leftarrow V_{active}.add\_vertex(x_{new}, cost_{new})$ *and* $E.add\_edge(v_{cur}, v_{new}, \psi_{new})$*:* The function call V_{active}.`add_vertex` adds a new vertex v_{new} to V_{active} such that $\overline{x}_{v_{new}} \leftarrow x_{new}$ and $\overline{c}_{v_{new}} \leftarrow cost_{new}$ and, consequently, returns v_{new}. The function call E.`add_edge` adds a new edge $e_{new} = (v_{cur}, v_{new})$ associated with ψ_{new} to E.

C. Solution Checking during HySST Construction

At each iteration, when a new vertex and a new edge are added to the search tree, i.e., `is_a_new_vertex_generated` = `true`, a solution checking function is employed to check if a path in $\mathcal{T}$ can be used to construct a motion plan to the given motion planning problem. If this function finds a path $p = ((v_0, v_1), (v_1, v_2), ..., (v_{n-1}, v_n)) =: (e_0, e_1, ..., e_{n-1})$ in $\mathcal{T}$ such that 1) $\overline{x}_{v_0} \in X_0$ and 2) $\overline{x}_{v_n} \in X_f$, then the solution pair $\widetilde{\psi}_p$ is a motion plan to the given motion planning problem.

V. Asymptotic Near-optimality Analysis

This section analyzes the asymptotic optimality property of HySST algorithm. The following assumption assumes that the cost functional is Lipchitz continuous along the purely continuous solution pairs, locally bounded at jumps, and satisfies additivity, monotonicity, and non-degeneracy.

Assumption 5.1: The cost functional $c : \hat{\mathcal{S}}^\phi_{\mathcal{H}} \to \mathbb{R}_{\geq 0}$ satisfies the following:

1) It is Lipschitz continuous for all continuous solution pairs (ϕ_0, u_0) and (ϕ_1, u_1) to $\mathcal{H}$ such that $\phi_0(0,0) = \phi_1(0,0)$; specifically, there exists $K_c > 0$ such that $|c(\phi_0) - c(\phi_1)| \leq K_c \sup_{(t,0) \in \operatorname{dom}\phi_0 \cap \operatorname{dom}\phi_1} \{|\phi_0(t,0) - \phi_1(t,0)|\}$.
2) For each pair of purely discrete solution pairs (ϕ_0, u_0) and (ϕ_1, u_1) to $\mathcal{H}$ such that $\operatorname{dom}\phi_0 = \operatorname{dom}\phi_1 = \{0\} \times \{0, 1\}$ and $\phi_0(0,0) = \phi_1(0,0)$, there exists $K_d > 0$ such that $|c(\phi_0) - c(\phi_1)| \leq K_d \sup_{j \in \{0,1\}} \{|\phi_0(0,j) - \phi_1(0,j)|\}$.
3) Consider two solution pairs $\psi_0 = (\phi_0, u_0)$ and $\psi_1 = (\phi_1, u_1)$, and let their concatenation be $\psi_0 | \psi_1$. The following hold:
   a) $c(\phi_0 | \phi_1) = c(\phi_0) + c(\phi_1)$ (additivity);
   b) $c(\phi_1) \leq c(\phi_0 | \phi_1)$ (monotonicity);
   c) For each $t_2 > t_1 \geq 0$ such that $(t_1, j) \in \operatorname{dom}\psi_0$ and $(t_2, j) \in \operatorname{dom}\psi_0$ for some $j \in \mathbb{N}$, there exists $M_c > 0$ such that $t_2 - t_1 \leq M_c |c(\phi_0(t_2, j)) - c(\phi_0(t_1, j))|$ (non-degeneracy during flows).
   d) For each $j_1, j_2 \in \mathbb{N}$ such that $j_2 > j_1$, $(t, j_1) \in \operatorname{dom}\psi_0$ and $(t, j_2) \in \operatorname{dom}\psi_0$ for some $t \in \mathbb{R}_{\geq 0}$, there exists $M_d > 0$ such that $j_2 - j_1 \leq M_d |c(\phi_0(t, j_2)) - c(\psi_0(t, j_1))|$ (non-degeneracy at jumps).

Next we define the clearance of the potential motion plans, which is heavily used in the literature; see, e.g., [16].

Definition 5.2: (Safety clearance of a motion plan) Given a motion plan $\psi = (\phi, u)$ to the motion planning problem $\mathcal{P} = (X_0, X_f, X_u, (C, f, D, g))$, the safety clearance of $\psi = (\phi, u)$, denoted δ_s, is such that for each $\delta' \in [0, \delta_s]$, the following conditions are satisfied:

1) $\phi(0,0) + \delta'\mathbb{B} \subset X_0$;
2) $\phi(T, J) + \delta'\mathbb{B} \subset X_f$, where $(T, J) = \max \operatorname{dom}\psi$;
3) For all $(t, j) \in \operatorname{dom}\psi$, $(\phi(t, j) + \delta'\mathbb{B}, u(t, j) + \delta'\mathbb{B}) \cap X_u = \emptyset$.

Assumption 5.3: The optimal motion plan to the optimal motion planning problem has positive safety clearance.

Assuming that the optimal motion plan is away from the boundary of the flow set and jump set is restrictive for hybrid systems [13]. To overcome this issue, the δ_f-inflation of

hybrid systems, denoted $\mathcal{H}_{\delta_f} := (C_{\delta_f}, f_{\delta_f}, D_{\delta_f}, g_{\delta_f})$ for some $\delta_f > 0$, is employed to create a positive dynamics clearance in our previous work [13].

The following assumption relating the safety clearance δ_s of the optimal motion plan and the inflation parameter δ_f with the algorithm parameters δ_{BN} and δ_s guarantees that the pruning process maintains at least one vertex close to the optimal motion planning if such vertex has been generated; see [8, Lemma 27] for details.

Assumption 5.4: The parameters δ_{BN} and δ_s need to satisfy $\delta_{BN} + 2\delta_s < \min\{\delta_s, \delta_f\}$.

The conditions in [13, Assumptions 5.3 - 5.6] regarding the random process, input library, and the continuous and discrete dynamics are also assumed in this paper. These assumptions are expected, based on what is known for the continuous-time and discrete-time cases. We are ready to provide our main result, which states that, by feeding the inflation $\mathcal{H}_{\delta_f}$, HySST returns a motion plan with cost that is close to the minimal cost.

Theorem 5.5: Given an optimal motion planning problem $\mathcal{P}^* = (X_0, X_f, X_u, (C, f, D, g), c)$, suppose Assumptions 5.1, 5.4, and assumptions in [13, Assumptions 5.3 - 5.6] are satisfied and that there exists an optimal motion plan $\psi^* = (\phi^*, u^*)$ to $\mathcal{P}^*$ satisfying Assumption 5.3 for some $\delta_s > 0$. When HySST is used to solve the motion planning problem $\mathcal{P}^*_{\delta_f} = (X_0, X_f, X_u, (C_{\delta_f}, f_{\delta_f}, D_{\delta_f}, g_{\delta_f}), c)$ where, for some $\delta_f > 0$, $(C_{\delta_f}, f_{\delta_f}, D_{\delta_f}, g_{\delta_f})$ denotes δ_f-inflation of (C, f, D, g), the probability that HySST finds a motion plan $\psi = (\phi, u)$ such that $c(\phi) \leq (1 + \alpha\delta)c(\phi^*)$ converges to one as the number of iterations k approaches infinity, where $\alpha \geq 0$ and $\delta = \min\{\delta_s, \delta_f\}$.

VI. HySST Software Tool for Optimal Motion Planning Problems for Hybrid Systems

Algorithm 1 has been implemented in a software tool[1] to solve the optimal motion planning problems for hybrid systems. This software only requires the inputs listed in Algorithm 1. Next, the HySST algorithm and this tool are illustrated in Example 3.1.

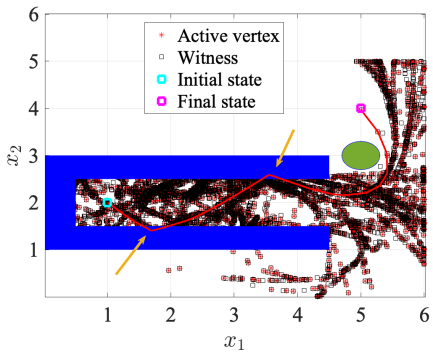

Fig. 1. The motion plan generated by HySST for the collision-resilient tensegrity multicopter in Example 3.1. The blue rectangles denote the walls where collisions potentially occur. The green circle denotes the forbidden zone. The yellow arrows point to the location where collisions occur.

[1]Code at https://github.com/HybridSystemsLab/hybridSST.

Example 6.1: (Collision-resilient tensegrity multicopter in Example 3.1, revisited) The simulation result in Figure 1 shows that HySST is able to ultilize the collision with the wall to decrease the hybrid time of the motion plan for multicopter. The simulation for this problem takes 54.7 seconds and creates 2094 active vertices on average.

VII. Conclusion

In this paper, a HySST algorithm is proposed to solve optimal motion planning problems for hybrid systems. The proposed algorithm is illustrated in the multicopter example and the results show its capacity to solve the problem. In addition, this paper provides a result showing HySST algorithm is asymptotically near optimal under mild assumptions.

References

[1] G. T. Wilfong, "Motion planning for an autonomous vehicle," in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*. IEEE, 1988, pp. 529–533.

[2] Y. Huang, H. Ding, Y. Zhang, H. Wang, D. Cao, N. Xu, and C. Hu, "A motion planning and tracking framework for autonomous vehicles based on artificial potential field elaborated resistance network approach," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 2, pp. 1376–1386, 2019.

[3] N. Wang, M. Song, J. Wang, and T. Gordon, "A flow-field guided method of path planning for unmanned ground vehicles," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 2762–2767.

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

[5] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[6] O. Nechushtan, B. Raveh, and D. Halperin, "Sampling-diagram automata: A tool for analyzing path quality in tree planners." in *WAFR*. Springer, 2010, pp. 285–301.

[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[8] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.

[9] J. W. Grizzle, G. Abba, and F. Plestan, "Asymptotically stable walking for biped robots: Analysis via systems with impulse effects," *IEEE Transactions on Automatic Control*, vol. 46, no. 1, pp. 51–64, 2001.

[10] Z. Song, L. Yue, G. Sun, Y. Ling, H. Wei, L. Gui, and Y.-H. Liu, "An optimal motion planning framework for quadruped jumping," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 11 366–11 373.

[11] P. M. Pinheiro, A. A. Neto, R. B. Grando, C. B. d. Silva, V. M. Aoki, D. S. Cardoso, A. C. Horn, and P. L. Drews Jr, "Trajectory planning for hybrid unmanned aerial underwater vehicles with smooth media transition," *Journal of Intelligent & Robotic Systems*, vol. 104, no. 3, p. 46, 2022.

[12] P. De Petris, S. J. Carlson, C. Papachristos, and K. Alexis, "Collision-tolerant aerial robots: A survey," *arXiv:2212.03196*, 2022.

[13] N. Wang and R. G. Sanfelice, "A rapidly-exploring random trees motion planning algorithm for hybrid dynamical systems," in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 2626–2631.

[14] R. G. Sanfelice, *Hybrid Feedback Control*. Princeton University Press, 2021.

[15] J. Zha and M. W. Mueller, "Exploiting collisions for sampling-based multicopter motion planning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 7943–7949.

[16] M. Kleinbort, K. Solovey, Z. Littlefield, K. E. Bekris, and D. Halperin, "Probabilistic completeness of rrt for geometric and kinodynamic planning with forward propagation," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. x–xvi, 2018.