

cHyRRT and cHySST: Motion Planning Tools for Hybrid Dynamical Systems in OMPL

Beverly Xu, Nan Wang, and Ricardo G. Sanfelice¹

Abstract—This paper presents two implementations of the recently developed motion planning algorithms HyRRT [1] and HySST [2]. Specifically, cHyRRT, an implementation of the HyRRT algorithm, generates solutions to motion planning problems for hybrid systems with a probabilistic completeness guarantee, while cHySST, an implementation of the asymptotically near-optimal HySST algorithm, finds near-optimal trajectories based on a user-defined cost function. The implementations align with the theoretical foundations of hybrid system theory and are designed based on OMPL, ensuring compatibility with ROS while prioritizing computational efficiency. The structure, components, and usage of both tools are detailed. A modified pinball game example is provided to illustrate the tools' key capabilities.

I. INTRODUCTION

Motion planning is becoming an increasingly important tool for researchers and practitioners as emerging robotic systems such as quadrupled robots and drones enter operation in complex environments. Among various implementations, the Open Motion Planning Library (OMPL) [3] is widely used due to its compatibility with the Robot Operating System (ROS) and its comprehensive collection of state-of-the-art planners for systems with purely continuous-time or purely discrete-time dynamics. However, motion planning for hybrid systems, in which the states can evolve continuously and, at times, exhibit jumps, along with implementations of such algorithms, are not currently available.

We consider the hybrid equation framework in [4] to model hybrid dynamical systems,

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

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, $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ represents the flow map, $D \subset \mathbb{R}^n \times \mathbb{R}^m$ represents the jump set, and $g : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ represents the jump map. Here, flow map f and jump map g capture continuous and discrete evolution of x , respectively. The flow set C collects the points where the state can evolve continuously. The jump set D collects the points where jumps can occur.

*Research by B. Xu, N. Wang and R. G. Sanfelice is 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.

¹Beverly Xu, Nan Wang, and Ricardo G. Sanfelice are with the Department of Computer Engineering, University of California, Santa Cruz, Santa Cruz 95064, CA xu21beve@gmail.com, nanwang@ucsc.edu, ricardo@ucsc.edu

¹Beverly Xu was a research intern at HSL during this work.

This general framework for hybrid systems captures a broad class of hybrid systems, including the class of hybrid systems considered in [5] and those that incorporate timers, impulses, constraints, and environmental contacts.

Within this framework, the feasible and the optimal motion planning problems for hybrid dynamical systems are formulated and addressed in [1] and [2], respectively. In practice, Rapidly-exploring Random Trees (RRT)-type algorithms efficiently compute trajectories for high-dimensional problems by incrementally constructing a search tree through random state-space sampling. Following the RRT algorithm scheme, HyRRT in [1] solves the feasible motion planning problem for hybrid systems, inheriting its probabilistically complete guarantee, meaning that the probability of failing to find a motion plan converges to zero as the number of samples approaches infinity [6].

In many motion planning applications, an optimal solution is preferred over merely a feasible but suboptimal one [7]. Unfortunately, solutions generated by RRT may converge to a suboptimal plan [8], while optimal variants such as PRM* and RRT* [9] rely on steering functions, restricting applicability. In contrast, the stable sparse RRT (SST) algorithm [10] eliminates the need for a steering function while guaranteeing asymptotic near-optimality, meaning that as the number of samples approaches infinity, the probability of finding a solution with a cost close to the minimum converges to one. Therefore, HySST is particularly useful when a user-defined cost functional is provided to evaluate the quality of solutions. To date, no motion planners for hybrid dynamical systems have been implemented within OMPL. To address this gap, we introduce cHyRRT and cHySST¹, two planners integrated into OMPL, allowing for compatibility with ROS [11].

II. NOTATION AND PRELIMINARIES

A. Notation

The set of real numbers is denoted as \mathbb{R} and its non-negative subset is denoted as $\mathbb{R}_{\geq 0}$. The set of nonnegative integers is denoted as \mathbb{N} . The notation $\text{int } I$ denotes the interior of the interval I . Given sets $P \subseteq \mathbb{R}^n$ and $Q \subseteq \mathbb{R}^n$, the Minkowski sum of P and Q , denoted as $P + Q$, is the set $\{p + q : p \in P, q \in Q\}$. The notation $\text{rge } f$ denotes the range of the function f . The notation $|\cdot|$ denotes a norm. Given a vector $h \in \mathbb{R}^2$, h_x is its first component, and h_y is its second component.

¹Code for both cHyRRT and cHySST is available at <https://github.com/xu21beve/ompl>

B. Preliminaries

Given a flow set C , the set $U_C := \{u \in \mathbb{R}^m : \exists x \in \mathbb{R}^n \text{ such that } (x, u) \in C\}$ includes all possible input values that can be applied during flows. Similarly, given a jump set D , the set $U_D := \{u \in \mathbb{R}^m : \exists x \in \mathbb{R}^n \text{ such that } (x, u) \in D\}$ includes all possible input values that can be applied at jumps. These sets satisfy $C \subseteq \mathbb{R}^n \times U_C$ and $D \subseteq \mathbb{R}^n \times U_D$. Given a set $K \subseteq \mathbb{R}^n \times U_*$, where $*$ is either C or D , we define $\pi_*(K) := \{x : \exists u \in U_* \text{ such that } (x, u) \in K\}$ 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 $(T, J) \in E$, $E \cap ([0, T] \times \{0, 1, \dots, J\})$ can be written as $\bigcup_{j=0}^J ([t_j, t_{j+1}], j)$ for some finite sequence of times $0 = t_0 \leq t_1 \leq t_2 \leq \dots \leq t_{J+1} = T$. A hybrid arc $\phi : \text{dom } \phi \rightarrow \mathbb{R}^n$ 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 $I^j := \{t : (t, j) \in \text{dom } \phi\}$ with nonempty interior.

The definition of a solution pair to a hybrid system is given as follows.

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

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

$$(\phi(t, j), u(t, j)) \in D \quad \phi(t, j + 1) = g(\phi(t, j), u(t, j))$$

Our motion planning algorithms require concatenating solution pairs. The concatenation operation of solution pairs is defined next.

Definition 2.2 (Concatenation operation): Given two functions $\phi_1 : \text{dom } \phi_1 \rightarrow \mathbb{R}^n$ and $\phi_2 : \text{dom } \phi_2 \rightarrow \mathbb{R}^n$, where $\text{dom } \phi_1$ and $\text{dom } \phi_2$ are hybrid time domains, ϕ_2 can be concatenated to ϕ_1 if ϕ_1 is compact and $\phi : \text{dom } \phi \rightarrow \mathbb{R}^n$ is the concatenation of ϕ_2 to ϕ_1 , denoted $\phi = \phi_1 | \phi_2$, namely:

- 1) $\text{dom } \phi = \text{dom } \phi_1 \cup (\text{dom } \phi_2 + \{(T, J)\})$, where $(T, J) = \max \text{dom } \phi_1$ and the plus sign denotes Minkowski addition;

- 2) $\phi(t, j) = \phi_1(t, j)$ for all $(t, j) \in \text{dom } \phi_1 \setminus \{(T, J)\}$ and $\phi(t, j) = \phi_2(t - T, j - J)$ for all $(t, j) \in \text{dom } \phi_2 + \{(T, J)\}$.

C. C++ Notation

In this paper, the following C++ notation is used. Namespace is a declarative region providing scope to one or more identifiers. In this paper, we also assume use of namespace `ob = ompl::base` and namespace `oc = ompl::control`. A lambda function is an anonymous function object that can be passed in as an argument. A pointer is a variable storage of the memory address of another variable. An asterisk `*`, when preceding a variable name at the time of declaration, is used to declare a pointer. The sequence of symbols `:` is the scope resolution operator used to traverse scopes such as namespaces and classes, to access identifiers. A double-valued scalar is a real, floating-point number with a maximum size of 64 bits.

III. MOTION PLANNING FOR HYBRID DYNAMICAL SYSTEMS

This section formulates the motion planning problem for hybrid dynamical systems following [2] and presents the data structure that implements its solution.

A. Problem Formulation

This paper solves a motion planning problem for hybrid dynamical systems, defined by flow and jump sets C and D , flow and jump maps f and g , along with the conditions to i) start from a given initial state, ii) end within a given goal state set, and iii) avoid reaching the unsafe set. This problem is mathematically formulated as follows.

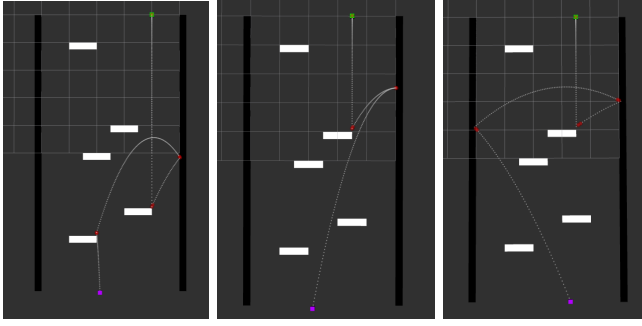
Problem 1: Given a hybrid system \mathcal{H} with input $u \in \mathbb{R}^m$ and state $x \in \mathbb{R}^n$, the initial state set $X_0 \subset \mathbb{R}^n$, the final state set $X_f \subset \mathbb{R}^n$, and the unsafe set $X_u \subset \mathbb{R}^n \times \mathbb{R}^m$, find a pair $(\phi, u) : \text{dom}(\phi, u) \rightarrow \mathbb{R}^n \times \mathbb{R}^m$, namely a *motion plan*, such that for some $(T, J) \in \text{dom}(\phi, u)$, the following hold:

- 1) $\phi(0, 0) \in X_0$, namely, the initial state of the solution belongs to the given initial state set X_0 ;
- 2) (ϕ, u) is a solution pair to \mathcal{H} as defined in Definition 2.1;
- 3) (T, J) is such that $\phi(T, J) \in X_f$, namely, the solution belongs to the final state set at hybrid time (T, J) ;
- 4) $(\phi(t, j), u(t, j)) \notin X_u$ for each $(t, j) \in \text{dom}(\phi, u)$ such that $t + j \leq T + J$, namely, the solution pair does not intersect with the unsafe set before its state trajectory reaches the final state set.

Therefore, given sets X_0 , X_f , and X_u , and a hybrid system \mathcal{H} with data (C, f, D, g) , a motion planning problem P is formulated as $P = (X_0, X_f, X_u, (C, f, D, g))$.

This problem is illustrated in the following example.

Example 3.1 (Modified Pinball Game): The state of the pinball is composed of 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 $a := (a_x, a_y) \in \mathbb{R}^2$. The state of the system is $x := (p, v, a) \in \mathbb{R}^6$ and its input is $u := u_x \in \mathbb{R} : u_x \in [-4, 4]$. The pinball machine environment is composed



(a) Simulation result solved by cHyRRT with initial position $p_0 = (4, 0)$. (b) Simulation result solved by cHyRRT with initial position $p_0 = (3.5, 0)$. (c) Simulation result solved by cHySST with initial position $p_0 = (3.5, 0)$.

Fig. 1. Simulated solution to the pinball example, graphed as p_x vs p_y . The start and goal vertices are marked by green and purple squares, respectively, and vertices in the jump regime are marked by red circles.

of actuated paddles, as represented by the white rectangles in Figure 1 and the two unactuated walls, represented by the black rectangles in Figure 1. We define the upper right corner of the left wall to be $p_{origin} = (0, 0)$, the downward direction as the negative y , and the rightward direction as the positive x . The union of the paddles and the walls defines the region $M \subset \mathbb{R}^2$. Hence, the flow set is $C := \{(p, v, a), u \in \mathbb{R}^6 \times \mathbb{R}^2 : p \in \mathbb{R}^2 \setminus M\}$. When the state of the pinball is in the flow set, its dynamics are governed by the following flow map:

$$\dot{x} = \begin{bmatrix} v \\ a \\ u \end{bmatrix} =: f(x, u) \quad (x, u) \in C. \text{ The jump set is defined}$$

as $D := \{(p, v, a), u \in \mathbb{R}^6 \times \mathbb{R}^2 : p \in \partial M, v \cdot n(p) \geq 0\}$, where $n(p)$ is the inward-pointing normal to the boundary ∂M of the closest paddle or wall, when the pinball is at position p . When the pinball experiences collision with the jump set D , $p^+ = p$. 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 $v_n^+ = -ev_n + u_{paddle} =: g_{v_n}(v)$ where $e \in (0, 1)$ is the coefficient of restitution. Note that the input u_{paddle} for v_n is only nonzero during collisions with the vertical paddle sides. During such collisions, the vectors u_{paddle} and $-ev_n$ point in the same direction. The velocity component v_t after the jump is modeled as $v_t^+ = v_t + u =: g_{v_t}(v)$ where the input u for v_t is only nonzero during collisions with the top surface of a paddle. Denote the projection of the updated vector (v_n^+, v_t^+) onto the x -axis as $\bar{x}(v_n^+, v_t^+)$ and the projection of the updated vector (v_n^+, v_t^+) onto the y -axis as $\bar{y}(v_n^+, v_t^+)$. Therefore, $v^+ = \begin{bmatrix} \bar{x}(g_{v_n}(v), g_{v_t}(v)) \\ \bar{y}(g_{v_n}(v), g_{v_t}(v)) \end{bmatrix} =: g_v(v)$. In both flow and jump sets, the dynamics of the pinball assume $a^+ = 0$. The discrete dynamics capturing the collision process is modeled

$$\text{as } x^+ = \begin{bmatrix} p \\ g_v(v) \\ 0 \end{bmatrix} =: g(x, u) \quad (x, u) \in D. \text{ Given the initial state set as } X_0 := \{0.5, 1, 2, 3.5, 4, 4.5\} \times \{0\}^4 \times \{-9.81\}, \text{ the final state set as } X_f = [1, 4] \times \{-10\} \times \mathbb{R}^4, \text{ and the}$$

unsafe set as

$$X_u = \{(x, u) \in \mathbb{R}^6 \times \mathbb{R}^2 : p_x \in [0, 1] \cup (4, 5] \cup p_y \in (-\infty, -10), (p_x, p_y) \in \text{int } M\},$$

The selection radius $\epsilon_{BN} := 0.8$ and the pruning radius $\epsilon_S := 0.2$. The example motion planning problem is set as $P = (X_0, X_f, X_u, (C, f, D, g))$. As the paddles obstruct a no-collision solution, the pinball machine example demonstrates a key strength of HyRRT over RRT; the capability to solve motion planning problems requiring collisions. Solutions to the pinball system are presented later in this paper.

B. Data structures

Both HyRRT and HySST algorithms search for motion plans by incrementally constructing search trees [1]. The search tree is modeled by a directed tree. A directed tree \mathcal{T} is a pair $\mathcal{T} = (V, E)$, where V is a set whose elements are called vertices and E is a set of paired vertices whose elements are called edges. Each vertex in the search tree \mathcal{T} is

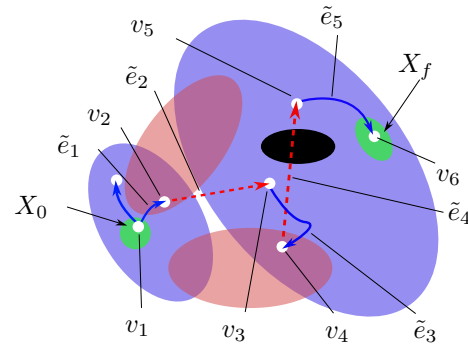


Fig. 2. Illustration of the search tree constructed by HyRRT/HySST. The path $p = (v_1, v_2, \dots, v_6)$ and the solution pair $\tilde{e}_p = \tilde{e}_1 | \tilde{e}_2 | \dots | \tilde{e}_5$.

associated with a state value of \mathcal{H} . Each edge in the search tree is associated with a solution pair to \mathcal{H} that connects the state values associated with their endpoint vertices. The state value associated with vertex $v \in V$ is denoted as x_v and the solution pair associated with edge $e \in E$ is denoted as $\tilde{e} = (\phi, u)$, where $\phi : \text{dom } \phi \rightarrow \mathbb{R}^n, u : \text{dom } u \rightarrow \mathbb{R}^m$. The solution pair that the path $p = (v_1, v_2, \dots, v_k)$ represents is the concatenation of all those solutions associated with the edges therein, namely, $\tilde{p} := \tilde{e}_{(v_1, v_2)} | \tilde{e}_{(v_2, v_3)} | \dots | \tilde{e}_{(v_{k-1}, v_k)}$ where \tilde{p} denotes the solution pair associated with the path p .

1) *State Space*: As we assume the sets X_0, C, D, U_C , and U_D to have finite and positive Lebesgue measure [12, Assumption 6.15], we consider motion planning problems in a finite state space, which both algorithms make random selection from. We implement a state space using any derived class of the abstract OMPL class `ob::StateSpace2`.

2) *Solution Pair*: Concatenation of solution pairs, as defined in Definition 2.2, is required by our motion planning algorithms. Therefore, we augment the OMPL data structure `Motion3` to store an edge, which links the associated

²See: https://ompl.kavrakilab.org/classompl_1_1base_1_1StateSpace.html

³See: https://ompl.kavrakilab.org/classompl_1_1geometric_1_1RRT_1_1Motion.html

solution pair and inputs to other objects of `Motion` which share either endpoint of edge e . The augmented `Motion` class is implemented as follows:

```

1 class Motion {
2     ob::State *state;
3     Motion *parent;
4     std::vector<ob::State *> *solutionPair;
5     oc::Control input;
6     unsigned numChildren; // Only used in HySST:
7     bool inactive; // Only used in HySST:
8     ob::Cost accCost; // Only used in HySST:
9 };

```

In the data structure `Motion`, the discretized solution pair \tilde{e} is implemented as a vector of states `solutionPair`, following Definition 2.1. The input associated with each discretized state, sampled from the input sets (U_C, U_D) , is stored in `Motion` as the control input `input`. The complete implementation details for hybrid time, inputs, and edge associated with each `Motion` are introduced in the forthcoming subsections.

3) *Hybrid Time*: In the `Motion` datastructure, each state x is parameterized by a hybrid time (t, j) , where t is a double-valued scalar and j is an integer-valued scalar. As this implementation depends on OMPL, which does not presently contain a state space class with the capability to capture hybrid time, we designed such a class, `ob::HybridStateSpace`, inheriting from the OMPL class `ob::CompoundStateSpace` and storing the state space in the first `ob::StateSpace` element, and the hybrid time space in the second `ob::HybridTimeStateSpace` element, where the double-valued attribute `position` and integer-valued attribute `jumps`, representing the flow time and number of jumps, respectively.

4) *Inputs*: In the `Motion` datastructure, the input is stored as an object of the OMPL class `oc::Control`. Input sets U_C and U_D have minimum and maximum values for each state, implemented as 2-by- m arrays, containing double-valued scalars, where m denotes the dimension of input in (1).

5) *Edge*: In the `Motion` datastructure, the edge e is implemented as a C++ pointer to the left endpoint of the edge, or the `parent`. The right endpoint of the edge is represented by the vertex x_v , or its attribute `state`.

IV. IMPLEMENTATION OF THE HYRRT ALGORITHM

Next, we introduce the main steps executed by HyRRT. Given the motion planning problem $P = (X_0, X_f, X_u, (C, f, D, g))$ and the input library (U_C, U_D) , HyRRT performs the following steps, as in [1]:

Step 1: Sample a finite number of points from X_0 and initialize a search tree $\mathcal{T} = (V, E)$ by adding vertices associated with each sampling point.

Step 2: Randomly select a point x_{rand} from C or D by randomly sampling the state space and set checking using flow and jump sets C and D , respectively. The definite planning space is defined in Section III-B.1.

Step 3: Find the vertex v_{cur} associated with the state value that has minimal distance to x_{rand} .

Step 4: Randomly select an input signal (value) from U_C (U_D) if the flow (jump, respectively) regime is selected. Then, compute a solution pair using the flow map f or jump map g , starting from $x_{v_{\text{cur}}}$ with the selected input applied, denoted $\tilde{e}_{\text{new}} = (\phi_{\text{new}}, u_{\text{new}})$. If, during a simulation starting from the flow regime, ϕ_{new} intersects with the jump regime, compute an additional solution pair using the jump map from the collision vertex. Denote the final state of ϕ_{new} as x_{new} . If \tilde{e}_{new} does not intersect with X_u , add a vertex v_{new} associated with x_{new} to V and an edge $(v_{\text{cur}}, v_{\text{new}})$ associated with \tilde{e}_{new} to E . Then, go to Step 2.

Following the above overview of HyRRT, the pseudocode for the proposed algorithm can be found in [12, Algorithm 2]. Revisiting Example 3.1, we next introduce the implementation of selected steps in HyRRT.

A. Step 2 (Random State Selection):

HyRRT randomly selects a point from C or D separately, rather than selecting from $C \cup D$. The reason is that if C (or D) has zero measure while D (or C , respectively) does not, the probability that the point selected from $C \cup D$ lies in C (or D , respectively) is zero, which would prevent establishing probabilistic completeness. The flow and jump sets C and D are defined as functions `flowSet_`⁴ and `jumpSet_`, respectively, and the random selection is implemented by the static function `randomSample`.

- 1) Jump set D is implemented as the lambda function `jumpSet_`. It takes in an arbitrary state as an input and outputs `true` if the state belongs to jump set D , and `false` if not.
- 2) Flow set C is implemented as the lambda function `flowSet_`. It takes in an arbitrary state as an input and outputs `true` if the state belongs to flow set C , and `false` if not.
- 3) The OMPL `SpaceInformation` class's built-in state space sampler function `randomSample` is utilized to select a random state.

B. Step 3 (Find Nearest Vertex):

HyRRT searches for a vertex v_{cur} in the search tree $\mathcal{T} = (V, E)$ such that its associated state value has minimal distance to x_{rand} . This process is implemented as follows.

- When $x_{\text{rand}} \in C$, the following optimization problem is solved over X_c :

$$\text{Problem 2: } \min_{v \in V} \|x_v - x_{\text{rand}}\| \quad \text{s.t. } x_v \in X_c.$$
- When $x_{\text{rand}} \in D$, the following optimization problem is solved over X_d :

$$\text{Problem 3: } \min_{v \in V} \|x_v - x_{\text{rand}}\| \quad \text{s.t. } x_v \in X_d.$$

⁴Following OMPL style for class member variables, we include an underscore after `flowSet`. We treat `flowSet_` as a member variable rather than a function because it is specific to instances of the class.

This optimization problem can be solved by traversing all the vertices in $\mathcal{T} = (V, E)$. The unsafe set is defined as the function `unsafeSet_`. Below, we present the static optimization function used to solve Problems 2 and 3.

- 1) `setDistanceFunction`: Set the function `distanceFunc_` that computes distance between states.
- 2) `NearestNeighbors`: A built-in OMPL class is used to store and search within the search tree of `Motion` objects `nn_`, which is an object of `NearestNeighbors`. The function call `nearest` solves both Problem 2 and 3, returning the `Motion` in `nn_` with the minimal distance to `randomMotion`, measured by `distanceFunc_`.
- 3) X_u : The unsafe set X_u is implemented as the lambda function `unsafeSet_`. It takes in an arbitrary state as an input and outputs `true` if the state belongs to unsafe set X_u and `false` if not.

C. Step 4 (Propagation of the Search Tree):

If $x_{v_{cur}} \in C \setminus D$ (or $x_{v_{cur}} \in D \setminus C$), a new solution pair \tilde{e}_{new} to the hybrid system H , starting from $x_{v_{cur}}$, is generated by applying an input signal \tilde{u} (or an input value u_D) randomly selected from U_C (or U_D , respectively). If $x_{v_{cur}} \in C \cap D$, then this function generates \tilde{e}_{new} by randomly selecting flows or jumps. The final state of \tilde{e}_{new} is denoted as x_{new} . Note that the choices of inputs are random. After \tilde{e}_{new} and x_{new} are generated, HyRRT checks if there exists $(t, j) \in \text{dom}(\tilde{e}_{new})$ such that $\tilde{e}_{new}(t, j) \in X_u$. If so, then \tilde{e}_{new} intersects with the unsafe set, no x_{new} is returned. Otherwise, x_{new} is successfully returned. When $x_{v_{cur}} \in C$, this step is implemented by the function `continuousSimulator_`, which uses numerical integration to propagate the state, given the randomly generated flow time input, and with respect to the dynamics of flow map f . Then, the solution pair is checked for collision with the jump regime using the function `collisionChecker_`. When $x_{v_{cur}} \in D$ or $x_{v_{cur}}$ experiences a collision with the jump regime during continuous propagation, a jump is simulated by the discrete simulator `discreteSimulator_`.

- 1) Flow map f an object of the OMPL class `oc::StatePropagator`⁵. Each discretized integration step spans a duration specified by the user as class member variable `flowStepDuration_`. A full example implementation of flow map f can be found in [13].
- 2) The discrete simulator is implemented within the lambda function `discreteSimulator_`. The function `discreteSimulator_` returns a state propagated once.
- 3) The collision checker is implemented as the lambda function `collisionChecker_`. It takes in an input of a `Motion` object and, if a collision occurs, outputs `true` and updates the edge's right endpoint to reflect

⁵See OMPL class reference: https://ompl.kavrakilab.org/classompl_1_1_control_1_1_StatePropagator.html

the collision state. If no collision occurs, then the function outputs `false` and makes no modification to the edge.

D. Solution Checking Process:

A solution checking function is employed to check if a path in \mathcal{T} can be used to construct a motion plan for the given motion planning problem. If this function finds a path $p = ((v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_n)) =: (e_0, e_1, \dots, e_{n-1})$ in \mathcal{T} such that 1) $x_{v_0} \in X_0$ and 2) $x_{v_n} \in X_f$, then the solution pair \tilde{p} is a motion plan for the given motion planning problem. This function is constructed by implementing the abstract OMPL class `ob::Goal`⁶.

V. IMPLEMENTATION OF THE HYSST ALGORITHM

HySST generates asymptotically near-optimal solutions, with only two notable deviations from the main steps of HyRRT in Section IV:

- Step 3: Find the vertex v_{cur} associated with the state value that has the minimal cost functional, within the neighborhood defined by a ball of selection radius ϵ_{BN} of x_{rand} . If no vertex exists within the neighborhood, the nearest vertex in V is selected. The larger the magnitude of selection radius ϵ_{BN} , the stronger the preference is for lower cost vertices, and thus, the lower the average cost for generated solutions. Typically, however, the greater the preference for lower cost vertices, the greater the computational burden. The selection radius ϵ_{BN} is set using `setSelectionRadius(double selectionRadius)` function.
- Step 4: Once a solution pair is computed, if \tilde{e}_{new} does not intersect with X_u and v_{new} has a minimal cost within the neighborhood defined by a ball of pruning radius ϵ_S , add the vertex v_{new} associated with x_{new} to V and an edge (v_{cur}, v_{new}) associated with \tilde{e}_{new} to E . Then, the pruning process removes from the search tree the vertices without child vertices and with higher cost, in the neighborhood of the new vertex v_{active} defined by a ball of radius ϵ_S . This additional step maintains S , a static set of witnesses to sparsify the vertices. The larger the magnitude of pruning radius ϵ_S , the sparser the tree, and the worse the cost, compared to that of the real optimal solution.

The HySST algorithm pseudocode can be found in [2].

The number of solutions allowed in a single instance of HySST is set using the `setBatchSize(int batchSize)` function. Once the maximum number of solutions is reached, the tool's `solve` function, which generates the solution path, will return the solution with the lowest cost. Batch size is default-initialized to be 1 and must be a positive integer if customized. The principle and effect of

⁶See OMPL class reference: https://ompl.kavrakilab.org/classompl_1_1_base_1_1_Goal.html

batch size will be further discussed in Section VII through Example 3.1.

VI. SIMULATION RESULTS

We illustrate both cHyRRT and cHySST in Example 3.1.

Example 6.1 (Example 3.1, revisited): The following procedure is used to generate a solution to the motion planning problem for the specific instance of the pinball system presented in Example 3.1 using cHySST:

- 1) Inside the C++ file `PinballPlanning.cpp`, the cost function, initial conditions, and planning and simulation parameters are set; the cost of a state x of the pinball is the negative value of its total distance traveled along the x -axis, from its starting state x_0 to its current state x .
- 2) The motion planner is run using the `solve` function within cHySST to return a trajectory.

On average, the pinball example experienced a reduction in computation time, from an average of 6.425 seconds per run when using HyRRT in MATLAB, to an average of 0.7173 seconds per run when using cHyRRT (C++).

VII. DISCUSSION: EVOLUTION OF SOLUTION COSTS OVER RANGE OF HYSST SOLUTION BATCH SIZES AND LIMITATIONS

A. Evolution of Solution Costs Over Range of HySST Solution Batch Sizes

The data collected from 15 runs of cHySST solving the pinball motion planning problem shows an inverse correlation between solution batch size and the cost of the lowest-cost solution. As the batch size increases, both the minimum and mean costs of the 15 lowest-cost solutions decrease. However, while increasing batch size generally reduces the cost of the generated solution with the best cost, it also increases computational consumption. As more of the planning space is explored and vertices are sparsified, the rate at which new vertices v_{new} are added to V declines, increasing the difficulty of finding a unique, lower-cost solution.

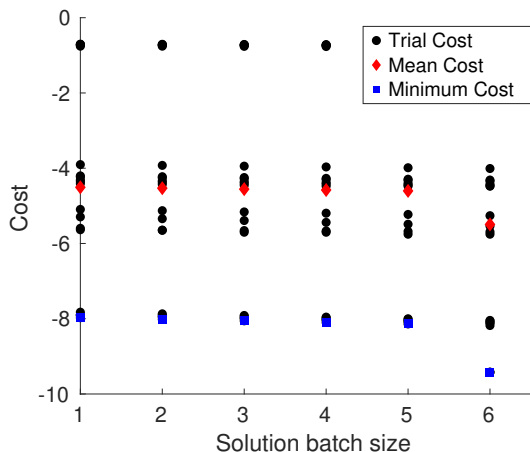


Fig. 3. Cost of lowest-cost solution given a solution batch size.

B. Limitations

Both HySST and HyRRT face two major limitations, which pose opportunities for promising future work: i) dependence on a successfully formulated hybrid model and ii) dimensional explosion. Expertise in hybrid systems is required to develop an accurate hybrid model. Both algorithms also face exponential growth of vertices during search. Let n be the number of input signals in inputs sets U_C and U_D . While a solution is yet to be found, n input signals can be applied to m vertices, with the number of vertices m increasing as the search tree grows, resulting in dimensional explosion. A more detailed time complexity analysis for HyRRT and HySST can be found in [14, Section 3.4].

VIII. CONCLUSION

The two tools cHyRRT and cHySST, for planning of hybrid systems, were described and illustrated in examples. Leveraging the computational efficiency of C++, the applicability to high-dimensional hybrid systems of the RRT-type and SST-type tools, and the generalizability of OMPL and ROS to robotics applications, we present two motion planning tools. In future work, we will implement HyRRT-Connect, a bidirectional RRT algorithm, in C++/OMPL.

REFERENCES

- [1] 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)*, pp. 2626–2631, IEEE, 2022.
- [2] N. Wang and R. G. Sanfelice, "HySST: An asymptotically near-optimal motion planning algorithm for hybrid systems," in *2023 62nd IEEE Conference on Decision and Control (CDC)*, pp. 2865–2870, IEEE, 2023.
- [3] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, 12 2012. <https://ompl.kavrakilab.org>.
- [4] R. Goebel, R. G. Sanfelice, and A. R. Teel, "Hybrid dynamical systems," *IEEE Control Systems Magazine*, vol. 29, no. 2, pp. 28–93, 2009.
- [5] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan, "Sampling-based planning and control," in *Proceedings of the 12th Yale Workshop on Adaptive and Learning Systems*, (New Haven, CT), Citeseer, 2003.
- [6] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.
- [7] Y. Yang, "Survey of optimal motion planning," *IET Cyber-Systems and Robotics*, vol. 1, pp. 13–19(6), 6 2019.
- [8] O. Nechushtan, B. Raveh, and D. Halperin, "Sampling-diagram automata: A tool for analyzing path quality in tree planners," vol. 68, pp. 285–301, 01 2010.
- [9] 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.
- [10] 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.
- [11] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.
- [12] N. Wang and R. G. Sanfelice, "Motion planning for hybrid dynamical systems: Framework, algorithm template, and a sampling-based approach," *arXiv preprint arXiv:2406.01802*, 2024.
- [13] B. Xu, N. Wang, and R. Sanfelice, "chyrtr and chysst: Two motion planning tools for hybrid dynamical systems," 2024.
- [14] N. Wang, "Provably-correct and efficient motion planning for hybrid dynamical systems," Jan 2025.