Assisted Mobile Robot Teleoperation with Intent-aligned Trajectories via Biased Incremental Action Sampling

Xuning Yang and Nathan Michael

***Abstract*— We present a method to assist the operator in teleoperation of mobile robots by generating trajectories such that the vehicle completes the desired task with ease in unstructured environments. Traditional assisted teleoperation methods have focused on reactive methods to avoid collisions, but neglect the operator's intention in doing so. Instead, we generate long horizon, smooth trajectories that follow the operator's intended direction while circumventing obstacles for a seamless teleoperation experience. For mobile robot teleoperation, an explicit goal in the state space is often unclear in cases such as exploration or navigation. Therefore, we model the intent as a direction and encode it as a cost function. As trajectories of various lengths can satisfy the same directional objective, we iteratively construct a tree of sequential actions that form multiple trajectories along the intended direction. We show our algorithm on a real-time teleoperation task of a simulated hexarotor vehicle in a dense random forest environment. By doing so, our approach allows operator to achieve the navigation task while requiring less effort than reactive methods.**

I. Introduction

Teleoperation of mobile robots is often used in unstructured environments to achieve a task such as navigation, exploration, or search and rescue. To assist operators, current works mitigate collisions by reactively steering the vehicle away from obstacles using haptic feedback [1] or augmenting control inputs [2]. These approaches do not take into consideration the operator's intentions, and require operators to react fast in environments such as those shown in Fig. 1. Instead of pushing the vehicle away from an obstacle that may be along the operator's intended path, we propose generating long horizon smooth trajectories that circumvent obstacles while following the operator's intention. The autonomously generated trajectory reduces the need to engage the operator at a high frequency in dense environments, which has been shown to cause fatigue [3].

Generating trajectories that align with the operator's intentions is a difficult task, as a fixed goal may not be known, or the operator has a specific trajectory in mind that they would like to take. In this case, it is prudent to represent operator's intent as a direction instead of an explicit goal in the state space [4]. An example of this is directional intent objective that corresponds to a "go straight" motion, as shown in Fig. 2. In this navigation scenario, it is more likely that the operator want to move forward while avoiding the obstacle than to crash into the obstacle, even though the provided input would lead to a crash.

There are a few challenges for this trajectory generation problem with the intent objective as a cost function: 1) The intent objective may be optimized with multiple trajectories with varying lengths; 2) the trajectories needs to remain in a collision-free space while adhering to the intent objective; and 3) the trajectories needs to be generated without specifying a specific state-space goal.

The Authors are with the Robotics Institute at Carnegie Mellon University, Pittsburgh PA, 15213, USA {xuning, nmichael}@cmu.edu

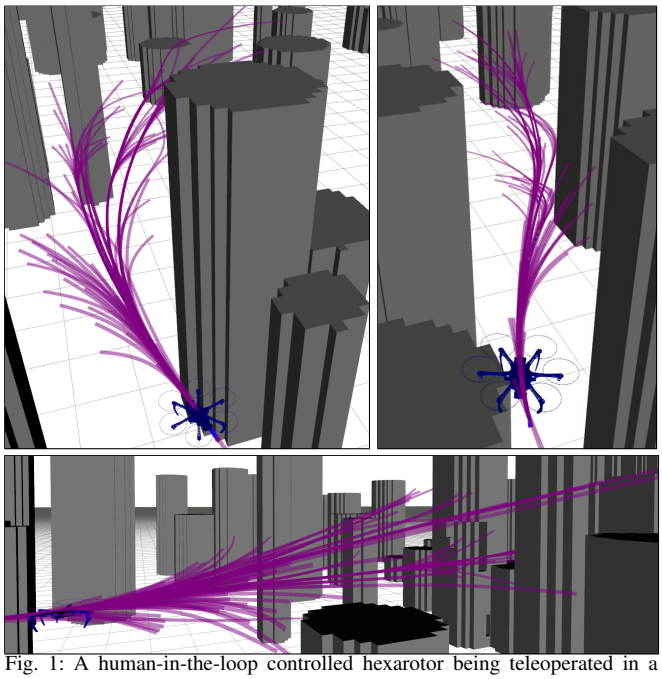

Fig. 1: A human-in-the-loop controlled hexarotor being teleoperated in a densely cluttered environment, with trajectories that follow the directional intent of the operator while bypassing obstacles in the environment. The proposed method generates long-horizon, smooth trajectories that enables operator navigation of mobile vehicles in unstructured environments with reduced effort, especially in dense environments that require operators to be vigilant to avoid collisions while focusing on the navigation task.

Therefore, we propose a trajectory generation method, Biased Incremental Action Sampling (BIAS), that minimize length-agnostic intent objectives while remaining inside a nonconvex collision-free space. Instead of sampling states in the larger state space, the proposed approach iteratively constructs trajectories by sampling and adaptively increasing the feasible action space as needed, with the cost function acting as a *bias* towards the minimum cost action sequences. The resulting action sequences are translated into sequential motion primitives which form a snap-continuous trajectory.

We showcase this approach in a navigation task of a UAV in a dense random forest as shown in Fig. 1. *The contribution of this paper is a teleoperation framework that, instead of taking reactive measures to assist the operator from collisions, assists the operator by generating predictive long horizon trajectories that align with the operator's directional intent while circumventing obstacles.* The resulting long-horizon trajectories enable navigation tasks to be completed according to operator intent with reduced effort required

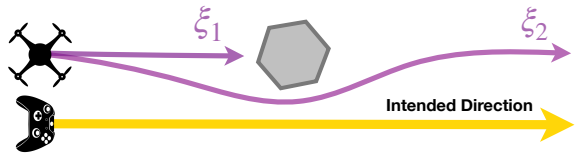

Fig. 2: This illustration shows two trajectories, ξ_1 and ξ_2, that both characterize a "go straight" motion. In this scenario, ξ_1 can be achieved by taking one action (forward), and ξ_2 requires at least three actions in order to successfully avoid the obstacle (forward, right, left). However, the vehicle will likely end up in a collision state after taking ξ_1 with the obstacle. It is more likely that ξ_2 is the intended trajectory that the operator prefers.

from the operator. The proposed approach can run online up to 10Hz on a CPU in high-density environments.

II. Related Works

1) Teleoperation: Recent works on assisted teleoperation of UAVs have focused on methods targeting low level operator control to ensure safety. Haptic feedback informs and prevents collisions at the interface level [1, 5, 6]. Low-level control inputs from the operator are augmented in order to avoid the immediate obstacles [2, 7]. One-step action selection allows operator to select a parameterized trajectory and prunes infeasible actions[8]. These methods are *reactive* methods to prevent collisions, but do not consider the task at hand or the intention of the operator.

2) Motion planning: Given an objective and an environment, motion planners generate a trajectory using either a gradient-based optimizer or using sample-based motion planning. Recent work extends sample-based motion planning from state lattices to incorporate considerations of dynamics [9, 10]. Bi-level kinodynamic methods [11] search directly in the space of discretized higher order derivatives, then refine the trajectory via optimization [10, 12]. Alternatively, a solution is to directly sample from the space of trajectories [9] and output a distribution of parameters that represent the minimum cost trajectories. All of these methods assume that a goal or objective is specified in the state-space.

Instead of geometric paths, one can compute motion primitives on discretized lattice grids for ground [13, 14] and aerial [15] robots. By concatenating motion primitives, one can build a tree of trajectories [16], although a naive tree construction could face combinatorial explosion in the number of resulting nodes, which can quickly become intractable to evaluate in terms of the number of trajectories. Monte Carlo Tree Search (MCTS) has been applied in single and multi-robot motion planning for mobile robots [17, 18].

III. Problem Formulation

A motion primitive $\gamma(t)$ is a parameterized function defined over a time interval $t \in [0, T]$ which generates a unique sequence of states given an initial state $\mathbf{x}_0 \in \mathcal{X}$:

$$\gamma_{\mathbf{a},T} : [0, T] \to \mathcal{X} \qquad \mathbf{a} \in \mathcal{A},\ T \in [0, \infty)$$

where $\mathcal{A} = \mathbb{R}^m$ is the action or parameter space. A choice of the motion primitive function is presented in Sect. V-.1.

The parameter space $\mathcal{A}$ could be constrained due to safety or feasibility. For example, the set of parameters that are dynamically feasible for a mobile robot can be defined as follows:

$$\mathcal{A}_{\text{feas}} = \{\mathbf{a} \in \mathcal{A} \mid \gamma^{(i)}_{\mathbf{a},T}(t) \leq \mathbf{x}^{(i)}_{\max}\} \ \forall\, t \in [0, T] \qquad (1)$$

where the relevant i-th order derivative (e.g., acceleration) along the trajectory is upper bounded by some known limit of the vehicle $\mathbf{x}^{(i)}_{\max}$. The set of parameters that are safe for a mobile robot can be defined as follows:

$$\mathcal{A}_{\text{safe}} = \{\mathbf{a} \in \mathcal{A} \mid \gamma_{\mathbf{a},T}(t) \in \mathcal{X}_{\text{safe}}\} \ \forall\, t \in [0, T] \qquad (2)$$

where $\mathcal{X}_{\text{safe}} = \mathcal{X} \setminus \mathcal{O}$ and $\mathcal{O}$ is the set of obstacles. Note that $\mathcal{A}_{\text{feas}}$ and $\mathcal{A}_{\text{safe}}$ may not be convex spaces.

Define a trajectory to be a sequence of N motion primitives:

$$\xi = (\gamma_1, \ldots, \gamma_N) = (\gamma_i)_{i=1}^N \qquad (3)$$

The duration of the trajectory is given by $\mathbf{T} = \sum_{i=1}^N T_i$, where T_i is the duration of the i-th primitive. The trajectory function is defined as:

$$\xi(t) = \gamma_i(t - \tau_{i-1}) \qquad t \in [\tau_{i-1}, \tau_i)$$

where τ_i is the cumulative duration up to primitive i. The parameters of a trajectory are given by

$$\alpha = [\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_N] \in \mathbb{R}^{mn}.$$

For a trajectory to be feasible and safe, the parameters of each motion primitive must be in the feasible parameter set $\mathbf{a}_i \in \mathcal{A}_{\text{feas}} \cap \mathcal{A}_{\text{safe}},\ i = 1, ..., N$.

A cost function that evaluates a trajectory with respect to the operator's directional intent is given as, $C_{\mathbf{a}^*}(\xi)$, where $\mathbf{a}^*$ is the operator's desired input. The problem statement is then as follows: Given $\mathbf{a}^*$, find a trajectory ξ that minimizes the cost function with respect to the parameters of the trajectory with n-segments in the feasible space:

$$\min_{\alpha \in \mathbf{\Lambda}_{\text{feas}} \cap \mathbf{\Lambda}_{\text{safe}}, n} C_{\mathbf{a}^*}(\xi) \qquad n \in \mathbb{N}^+, \qquad (4)$$

where $\mathbf{\Lambda} = \prod_{i=1}^n \mathcal{A}_i$ represents the action space of a trajectory. The action space scales with respect to the number of segments. The space of all of the number of possible segments in each trajectory sequence, n, forms a discrete set; therefore Eq. (4) is a mixed integer program unless n is fixed by choice.

IV. Biased Incremental Action Sampling (BIAS)

We iteratively build a tree of sequential actions that minimizes an objective as described in Eq. 4. The algorithm is detailed in Algorithm 1, and a simplified illustration of one iteration of the process is shown in Fig. 3.

Starting at the root node, the tree is constructed by selecting actions via weighted sampling from the Sample Set $\mathcal{S}$ (the set of possible nodes for expansion). For each node to be expanded, all known feasible actions (successors) are evaluated, and added to the Sample Set. The set of possible actions at each step is known a priori: they are generated by uniformly discretizing[1] the continuous action space $\mathcal{A}$ along each parameter dimension. Let the discretized action set of $\mathcal{A}$ be represented by $\mathcal{B} = \{\mathbf{a}_i\}_{i=1}^K$; therefore the maximum branching factor of the tree is K. At each iteration, the feasibility of an action $\mathbf{a}_i$ is evaluated by checking Eq. (1)

[1]Instead of sampling from a uniformly discretized action set, one can sample from a non-uniform prior over the continuous action space based on the initial state of the system. Such formulation would allow the samples to be chosen via an informative prior, and possibly reduce the number of nodes to be evaluated. We leave this to future work.

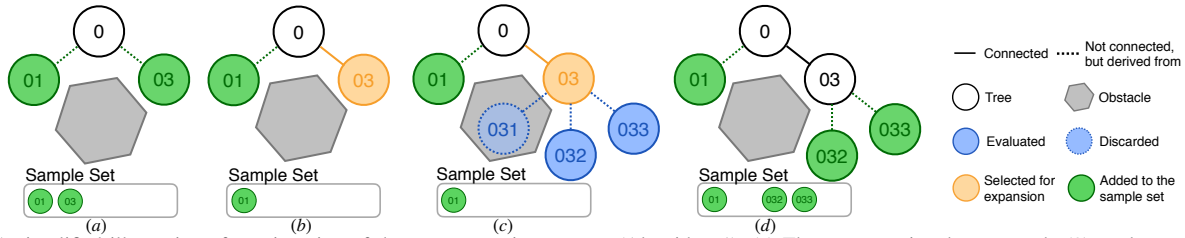

Fig. 3: A simplified illustration of one iteration of the tree generation process (Algorithm 1). (a) The tree contains the root node (0), and two nodes (01, 03) in the sample set. (b) From the sample set, node 03 is selected and added to the tree. (c) The set of possible successors to 03 (031, 032, 033) are evaluated. (d) From the set of successors, the nodes leading to a collision (031) are discarded, and the others added to the sample set. This process is repeated until a terminating condition is met.

and (2). In our implementation, safety of each action is checked with respect to a given map representation, which is discussed in Sect. VII.

Each node η at depth D represents the sequence of parameters starting from the root node, i.e. $\eta^D = \left(\mathbf{a}^d\right)_{d=0}^{d=D}$. Therefore, the cost function $C\left(\eta^D\right)$ of a node at depth D evaluates the sequence of parameters up to η^D. The cost of each node is used to generate a weighting of the node such that lower cost nodes maintain a higher likelihood to be sampled; e.g., $w(\eta) = \frac{1}{C(\eta)}$.

The Sample Set $\mathcal{S}$ is a set of tuples that contains the node parameters and its associated cost (η, w). At each iteration, J nodes are sampled from $\mathcal{S}$ according to their weights (Line 4). All feasible and safe children of the sampled nodes are then evaluated and added to the Sample Set (Line 9). As elements are added, the Sample Set increases exponentially in size thus causing this set to contain a large number of lower weighted elements. In order to highlight the higher weighted elements, the sampling is limited to an elite set containing the top ρ samples within the Sample Set, and those weights are passed through a softmax function:

$$\sigma(w_i) = \frac{e^{\beta w_i}}{\sum_{j=1}^{N} e^{\beta w_j}} \qquad \beta > 0$$

where β is a tuning parameter. This amplifies the likelihood of the higher weighted elements being sampled. We introduce a cost bound $\bar{C}$ which is updated after every iteration (Line 18). By doing so, this effectively bounds our search to the first local minima encountered.

V. Motion Primitive Trees

A motion primitive tree is generated once a new input is received from the operator via the joystick, and the lowest cost trajectory is executed. The resulting tree of actions generated using BIAS is transformed into a motion primitive tree by computing the appropriate motion primitives parameterized by the actions and an initial state. Each node η in the tree represents a motion primitive that takes the robot from one state to another via a time-parameterized trajectory function γ, generated by our choice of $\mathbf{f}$ with action $\mathbf{a}$. Each successor contains parameters that guarantee a snap-continuous trajectory from the parent node by construction. The root node is seeded with the initial state of the system. Thus, each sequence of nodes η with depth D is an equivalent representation of trajectory ξ with N segments given γ, with $D = N$:

$$\eta^D = \left(\mathbf{a}^d\right)_{d=0}^{d=D} \quad \Longleftrightarrow \quad \xi = (\gamma_i(t))_{i=1}^{i=N}$$

The continuous action space $\mathcal{A}$ is the action space of the motion primitive parameters. A *motion primitive library* is a discrete set of motion primitives created using the discretized action set $\mathcal{B}$: $\Gamma = \{\gamma(\mathbf{a}_i)\}_{i=1}^{K},\ \mathbf{a}_i \in \mathcal{B}$. The successors to each node form a motion primitive library with the initial state as determined by the end state of the current primitive. An example of the forward-arc motion primitive library is shown in Fig. 6.

The motion primitive tree becomes a set of trajectories given an initial condition. The application of this algorithm to an intent function (described in Sect. VI) enables trajectory generation for a right turning maneuver as shown in Fig. 4. Each multi-step primitive trajectory, by construction, naturally becomes an extension of our prior work with one-step reactive teleoperation [8]: In the worst case where

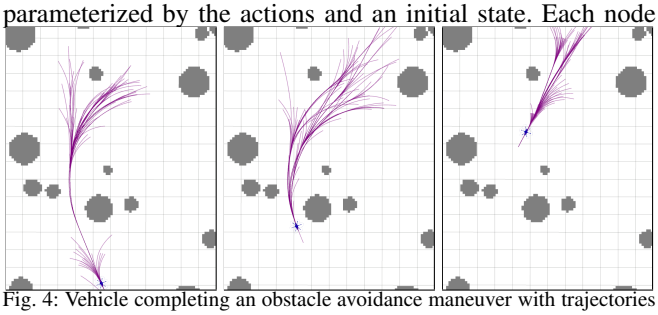

Fig. 4: Vehicle completing an obstacle avoidance maneuver with trajectories generated using motion primitive trees, shown in the x-y plane. The three sets of trajectories successfully enable the vehicle to complete the maneuver according to the intended direction of motion.

Algorithm 1: Biased Incremental Action Sampling

Input: Given a cost function $C(\xi)$, a batch sampling parameter $J \in \mathbb{N}^+$, and an initial state $\mathbf{x}_0$

1 Initialize the Sample Set with the root node: $\mathcal{S} = \{(\eta^0, 0)\}$
2 Initialize cost bound $\bar{C} = \infty$
3 **while** *terminating condition not met* **do**
4 Sample $\underline{J} = \min(|\mathcal{S}|, J)$ nodes from $\mathcal{S}$ according to their weights without replacement
5 **for** *each sampled node* η_j *with depth* d, $j = 1, ..., \underline{J}$ **do**
6 Add the sampled node η_j to the tree
7 Remove the sampled node from the Sample Set: $\mathcal{S} \leftarrow \mathcal{S} \setminus \{(\eta_j, w_j)\}$
8 **for** *all* $\mathbf{a} \in \mathcal{B}$ **do**
9 **if** $\mathbf{a} \in \mathcal{A}_{feas} \cap \mathcal{A}_{safe}$ **then**
10 Generate new node $\eta(\mathbf{a}) = (\eta_j^d, \mathbf{a})$
11 Evaluate cost of node $C(\eta)$
12 **if** $C(\eta) < \bar{C}$ **then**
13 Compute weight of node $w(\eta)$
14 Add node and its weight to the Sample Set: $\mathcal{S} \leftarrow (\eta, w)$
15 Optional: Update cost bound: $\bar{C} = \max_{s \in \mathcal{S}} C(s)$

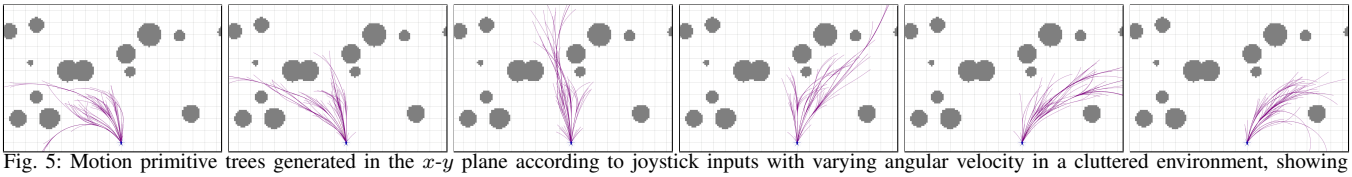

Fig. 5: Motion primitive trees generated in the x-y plane according to joystick inputs with varying angular velocity in a cluttered environment, showing effective obstacle avoidance while adhering to the directional intent specified by the operator. All trajectories are smooth and continuous up to snap.

every action downstream from the one-step action set incurs a higher cost, the algorithm simply returns the one-step motion primitive library, defaulting to the behavior of one-step teleoperation where the operator's selected primitive is carried out until the next iteration.

1) Forward Arc Motion Primitives: Multirotors are differentially flat systems. Exact inputs can be computed such that the vehicle follows a specified trajectory in the flat outputs x, y, z, and yaw θ. Define the state to be $\mathbf{x}{=}[x, y, z, \theta]$. We use *forward-arc motion primitives*, which generates a polynomial trajectory in $\mathbf{x}$ given an action $\mathbf{a}$. Let $\mathbf{a} = [v_x, \omega, v_z, T]^\top$, where $v_x{\in}\mathcal{V}_x$ is the linear velocity, $\omega{\in}\Omega$ is the angular velocity, $v_z{\in}\mathcal{V}_z$ is the z-velocity, and $T{\in}\mathcal{T}$ is the duration of the primitive. The action space is therefore $\mathcal{A} = \mathcal{V}_x \times \Omega \times \mathcal{V}_z \times \mathcal{T}$. Note that, instead of selecting a fixed duration T, we treat it as a parameter. The forward-arc motion primitives evolves the state following the unicycle model [14]:

$$\dot{\mathbf{f}}_{\mathbf{a}}(t) = [v_x \cos(\omega t) \quad v_x \sin(\omega t) \quad v_z \quad \omega]^\top, t \in [0, T] \quad (5)$$

Given an initial state $\mathbf{x}_0$ and its higher order derivatives $\dot{\mathbf{x}}_0, \ddot{\mathbf{x}}_0, \mathbf{x}_0^{(4)}$, the motion primitive function γ produces an eighth order parameterized polynomial with velocity endpoints constrained according to Eq. (5):

$$\begin{aligned} \gamma_{\mathbf{a},T}(t) &= \sum_{i=0}^{8} \mathbf{c}_i t^i \qquad (6) \\ \text{s.t. } \gamma^{(j)}(0) &= \mathbf{x}_0^{(j)}(t_k) \ \text{ for } j = 0, 1, 2, 3, 4 \\ \dot{\gamma}(T) &= \dot{\mathbf{f}}_{\mathbf{a}}(T) \\ \gamma^{(j)}(T) &= 0 \ \text{ for } j = 2, 3, 4 \end{aligned}$$

where $\{\cdot\}^{(j)}$ specifies the j^{th} time derivative.

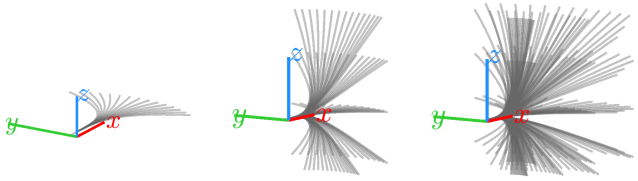

Fig. 6: Motion Primitive Library constructed with forward arc primitives. The variations in angular velocity, z velocity, and linear velocity are added incrementally for maximum clarity.

2) Trajectory selection policy: The existence of many local minimas will necessarily require us to terminate the algorithm once a desirable set of trajectories have been generated. Therefore, the termination criterion can be reaching a max tree size P; or until the Sample Set becomes empty. The algorithm returns a set of trajectories and the vehicle executes the minimum cost trajectory.

VI. Intent Cost Formulation

We present an intent cost function that encodes the desired direction of motion, which are used to generate motion primitive trees as shown in Fig. 5. Given a prediction of the most likely input, $\mathbf{a}^*$, the intent cost function can be described as follows. The cost function compares an approximate directional vector between the trajectory $\xi(t)$ and a one-step motion primitive, generated using the most optimal action as given by the intent model. Specifically, the trajectory is evaluated at its end points $\mathbf{x_0} = \xi(0)$, and $\mathbf{x_T} = \xi(\mathbf{T})$. A one-step motion primitive γ^* is generated using $\mathbf{a}^*$ with duration $\mathbf{T}$ according to Eq. (6), such that $\gamma^*(t) = \gamma_{\mathbf{a}^*,\mathbf{T}}(t)$. The endpoints generated by the desired motion primitive would be given by $\mathbf{x}_0^* = \gamma(0)$, and $\mathbf{x}_\mathbf{T}^* = \gamma(\mathbf{T})$. The cost function is given as:

$$C_{\text{input}_{\mathbf{a}^*}}(\xi) = \|1 - \mathbf{p} \cdot \mathbf{p}^*\| \qquad (7)$$

$$\mathbf{p} = \frac{\mathbf{x_T} - \mathbf{x_0}}{|\mathbf{x_T} - \mathbf{x_0}|} \qquad \mathbf{p}^* = \frac{\mathbf{x}_\mathbf{T}^* - \mathbf{x}_0^*}{|\mathbf{x}_\mathbf{T}^* - \mathbf{x}_0^*|} \qquad (8)$$

which shifts the dot product such that $C_{\text{input}} \geq 0$ and remains within $[0, 2]$. The most likely input $\mathbf{a}^*$ is generated in real-time by utilizing the prediction framework outlined in [19].

1) User preference and behavior heuristics: We augment the intent objective with descriptors of trajectory behavior in order to capture user preference in the qualitative shape of the trajectory. Trajectory behavior describes *how* a particular trajectory is executed. These are adjectives such as *slow* vs. *fast* or *smooth* vs. *aggressive*, which are translated to a value-function that penalizes higher order derivatives such as velocity, acceleration and jerk. For an $(n{+}1)$-segment trajectory comprised of motion primitives each parameterized by actions $\mathbf{a}_0, \mathbf{a}_1, ...$, we define and utilize the following behavior functions:

$$C_{\text{Straight}} = \sum_{i=1}^{n} |\omega_i| + \sum_{i=1}^{n} |v_{zi}| \quad C_{\text{Speed}} = \sum_{i=1}^{n} \frac{1}{\|\mathbf{v}_i\|_2}$$

penalizes nonzero curvature — penalizes trajectory with slow speeds

$$C_{\text{Smooth}} = \sum_{i=1}^{n} |\mathbf{a}_i - \mathbf{a}_{i-1}| \quad C_{\text{Duration}} = \sum_{i=1}^{n} \frac{1}{T_i}$$

penalizes changes in curvature — penalizes short duration trajectories

The cost functions are linearly combined with a set of user selected weights. A visualization of the effect of these cost functions is shown in Fig. 7.

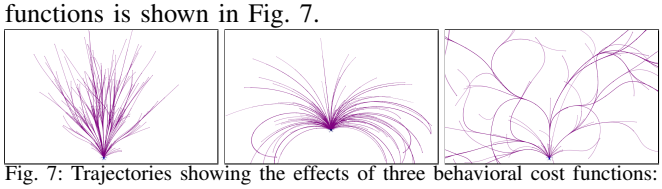

Fig. 7: Trajectories showing the effects of three behavioral cost functions: C_{straight} (left), C_{smooth} (middle), and C_{duration} (right). These show the effect of each cost function on the shapes of the resulting trajectories. The effect of C_{speed} is omitted as it is not intuitive to visualize in a path.

VII. Implementation

The proposed method is implemented in C++ on a CPU (Intel Core 2.20GHz i7-8750H CPU), with 16GB of RAM.

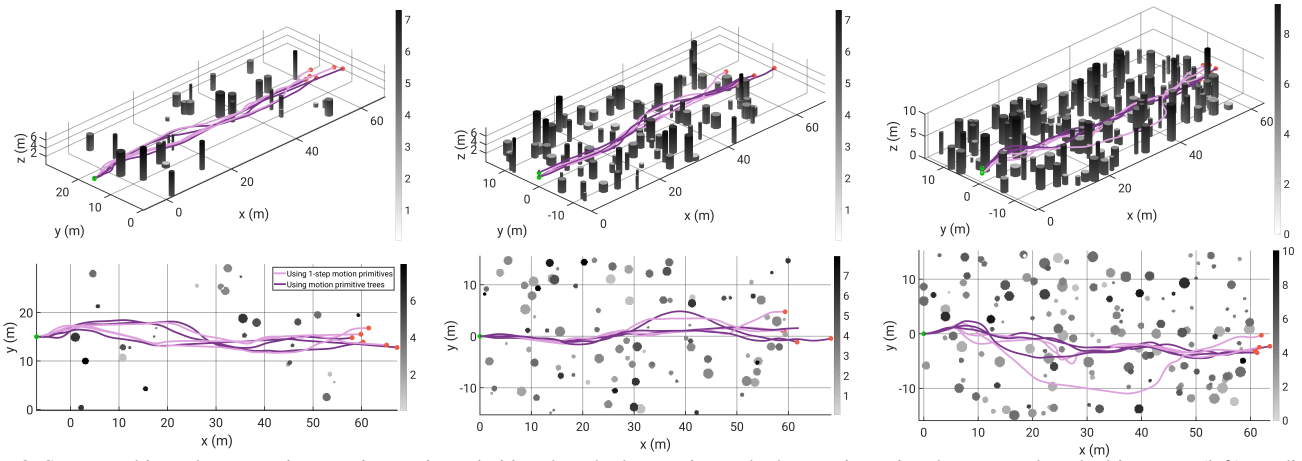

Fig. 8: Some resulting odometry using reactive motion primitives based teleoperation and teleoperation using the proposed method in sparse (left), medium (middle), and dense (right) enviroments for three trials each. The colorbar to the right indicate the pillar heights in meters. Both methods complete sparse environments equally, however the proposed method results in qualitatively smoother trajectories in the dense scenario. Data from these trials are summarized in Table V.

12 threads are allocated in order to support parallel evaluation of the discretized actions (Lines 8 - 17 in Algorithm 1).

We use a global map representation using both a KD-Tree based voxel representation, as well as a signed distance field (SDF) representation. KD-Tree is efficient for a local map representation [8]; However, SDF provide faster queries for global map since it can be processed offline. [2]

VIII. Experiment and Results

The proposed method is tested in simulation using a high-fidelity hexarotor model in various density random forest environments using a combination of the intent and behavior cost functions as described in Sect. VI.

Three random forest environments are used in this experiment with varying sparsity, each with size $60\text{m} \times 30\text{m} \times 10\text{m}$. The sparse, medium, and dense random tree forest contains approximately 30, 70, and 120 pillars respectively. The task is as follows: Navigate the simulated hexarotor vehicle from one end of the environment to the other. The operator can choose to take any path they wish to complete the task. The operator is given a third-person follower view of the vehicle which is ensured to be occlusion free such that the operator remains within line-of-sight of the vehicle.

The motion primitive parameters are fixed for all trials (Table I). The linear velocity is directly controlled by the operator using one of the axis of the joystick during operation with the maximum velocity capped at $2\text{m}/\text{s}$. The algorithm-associated parameters are given in Table II. The simulated vehicle has a radius of 0.6m with a collision radius set constant at 0.1m. The discretization size for both KDTree and SDF is fixed at 0.1m.

1) Timing results: The per-node timing evaluation is provided in Table III. The cost evaluation per node for all cost functions averages 0.0168ms. The collision check time in the densest environment for both SDF and KD-Tree representations are on the order of $10^{-3}\sim10^{-2}\text{ms}$. The collision check time should reduce significantly if using a local map.

[2]The method can be readily adapted to local maps instead of global maps. Local maps generated using limited range sensors may require limiting the maximum tree depth, such that the trajectories are within the known map.

TABLE I: Trajectory parameters: The 2D action space includes ω and T, and the 3D action space includes ω, T, and v_z.

Parameter	Min	Max	Num. discretizations
Duration T	0.2s	1.5s	5
Angular vel. ω	0.75rad/s	0.75rad/s	15
Z vel. v_z	-0.75m/s	0.75m/s	3
Total size of the motion primitive library			75 (2D), 225 (3D)

TABLE II: Cost parameters for random forest navigation

Cost function weights		Sampling Parameters	
w_{smooth}	0.3	Softmax parameter β	0.5
w_{straight}	0.1	Num. nodes sampled J	2
w_{duration}	0.6	Tree size (Num. nodes P)	100
w_{speed}	0.3	Elite set size ρ	500
w_{intent}	1.8		

The timing results for the trajectory generation process are shown in Table IV. Trajectory generation for a 2D tree takes approximately 113.12ms for an average depth of 5, and 300ms for a 3D tree with an average depth of 4. Approximately 9785 and 28906 nodes are evaluated to generate the 2D and 3D tree respectively. To contexualize our result, note that naively generating a fixed size 2D tree with depth 5 and 3D tree with depth 4 would result in $75^5 = 2.373\text{T}$ and $225^4 = 2.562\text{B}$ nodes respectively.

2) Task completion results: We compare the proposed method with respect to the previous reactive, one-step teleop-

TABLE III: Timing results per node (data from 2.4M node evaluations with the densest random forest environment)

Per node	Time
Cost Evaluation	0.0168 ± 0.0664 ms
Collision Check with SDF	0.00269 ± 0.00421 ms
Collision Check with KDTree	0.01351 ± 0.06001 ms
Total with SDF	0.0230 ± 0.0804 ms
Total with KDTree	0.0320 ± 0.0694 ms

TABLE IV: Timing and size data per tree (data from 100 trees generated in the densest random forest environment)

Time	2D (75 children)	3D (225 children)
Generation	113.12 ± 91.04 ms	299.78 ± 88.46 ms
Selection	1.14 ± 0.95 ms	1.73 ± 7.01 ms
Generation + Selection	116.44 ± 91.40 ms	315.57 ± 61.9 ms
Num. nodes processed	9785 ± 1499	28906 ± 4584
Num. traj per tree	56 ± 42	93 ± 21
Num. iter. per tree	42 ± 25	61 ± 16
Average depth of tree	5 ± 1	4 ± 1

2D trees refer to motion primitive trees generated in the x-y plane.
3D trees refer to those generated in the euclidean space.

eration proposed in [8]. The reactive method parameterizes the operator's control input and generates a single motion primitive. Both methods are evaluated in the sparse, medium and dense random forest environments with five trials each.

The key metric we evaluate is the number of joystick inputs received to complete each task, which represents the *effort* of the operator. This metric is critical to evaluating the operator engagement with the system during teleoperation: If the number of joystick inputs is high, it indicates that the operator has to frequently engage the system to control the vehicle. If the operator does not need to engage as much with the system as much over time, then this implies that the vehicle is following the operator's intended trajectory and therefore the operator does not feel the need to control the vehicle. We observe the task completion time as well as the smoothness of the trajectory via the jerk integral. The results are reported in Table V. A subset of the resulting trajectories are shown in Fig. 8.

The number of joystick inputs of each run ranges from ~ 44 control inputs for the sparse environment to ~ 58 control inputs for the densest environment. Contrast this to the reactive method, which utilized ~ 192 control inputs for the sparse case, and up to ~ 421 control inputs for the densest environment. In the sparse enviroment, the operator has to engage less due to the lack of obstacles. But in the dense environment, the number of inputs rise significantly for the reactive method due to the frequent encountering of obstacles and difficult structures that causes the operator to be highly engaged. The frequency of engagement is significantly reduced using the proposed multi-step method, indicating a reduction in effort required to achieve the same navigation task. Only 25% control inputs are required for the sparse environment, and the number of inputs only increased by around 15-20 for the densest environment. This represents a significant reduction operator's effort when navigating through difficult scenarios such as narrow gaps shown in the opening figure of the paper (Fig. 1).

Lastly, we observe that the completion time for each of the tasks are similar. We also observe that the generated trajectories exhibit a lower jerk profile throughout the task, which indicates that the proposed method generates smoother trajectories than the reactive teleoperation method.

TABLE V: Comparison of the different density enviroments

Approach	Sparse	Medium	Dense
Time to completion (s)			
Proposed (multi-step)	39.45 ± 1.43	36.40 ± 1.99	37.55 ± 2.34
Reactive (one-step)	38.84 ± 1.08	37.06 ± 0.50	37.95 ± 1.91
Jerk Integral (m^2/s^3)			
Proposed (multi-step)	18.96 ± 5.05	25.32 ± 4.32	33.75 ± 2.77
Reactive (one-step)	28.41 ± 2.88	26.72 ± 4.30	50.14 ± 7.38
Number of joystick inputs received to complete the trial			
Proposed (multi-step)	44 ± 12	38 ± 9	58 ± 10
Reactive (one-step)	192 ± 27	147 ± 45	421 ± 67

IX. Conclusion and Future Work

We present a method for assisted teleoperation by generating smooth trajectories that optimize directional intent objectives by iteratively constructing a tree of sequential actions. The intent objective is a cost function that describes the intended direction of the operator and behavioral descriptors that customize the shape of the trajectories. The proposed method is applied to navigation tasks in sparse, medium and dense random forest environments. We show a significant reduction in required operator effort to complete the task using the proposed method as compared to one-step reactive teleoperation. We plan to conduct a user study to evaluate the implications of this approach on user experience as part of our future work.

The proposed method can be readily extended in a few ways. First, the method can be extended to dynamic obstacles, since the sequential nature of the trajectory generation would allow collision checking of each primitive with respect to time-stamped map representations. Further, other choices of motion primitive can be used depending on the application and the vehicle dynamics. Finally, the trajectory generation process is applicable to other domains where the objective may not include a state-space goal; e.g., exploration and search and rescue. We leave these extensions as future work.

References

[1] Stefano Stramigioli, Robert Mahony, and Peter Corke. A novel approach to haptic tele-operation of aerial robot vehicles. In *IEEE Int. Conf. on Robot. and Autom. (ICRA)*. IEEE, 2010.

[2] Marcin Odelga, Paolo Stegagno, and Heinrich H Bülthoff. Obstacle detection, tracking and avoidance for a teleoperated uav. In *IEEE Int. Conf. on Robot. and Autom. (ICRA)*. IEEE, 2016.

[3] Maarten AS Boksem, Theo F Meijman, and Monicque M Lorist. Effects of mental fatigue on attention: an erp study. *Cognitive brain research*, 2005.

[4] Xuning Yang, Ayush Agrawal, Koushil Sreenath, and Nathan Michael. Online adaptive teleoperation via motion primitives for mobile robots. *Special Issue on Learning for Human-Robot Collaboration, Autonomous Robots*, 2018.

[5] Xiaolei Hou and Robert Mahony. Dynamic kinesthetic boundary for haptic teleoperation of aerial robotic vehicles. In *IEEE Int. Conf. on Intell. Robots and Syst. (IROS)*. IEEE, 2013.

[6] Thanh Mung Lam, Harmen Wigert Boschloo, Max Mulder, and Marinus M Van Paassen. Artificial force field for haptic feedback in uav teleoperation. *IEEE Trans. on Systems, Man, and Cybernetics*, 2009.

[7] Matthias Nieuwenhuisen, David Droeschel, Johannes Schneider, Dirk Holz, Thomas Läbe, and Sven Behnke. Multimodal obstacle detection and collision avoidance for micro aerial vehicles. In *2013 European Conference on Mobile Robots*. IEEE, 2013.

[8] Alex Spitzer, Xuning Yang, John Yao, Aditya Dhawale, Kshitij Goel, Mosam Dabhi, Matt Collins, Curt Boirum, and Nathan Michael. Fast and agile vision-based flight with teleoperation and collision avoidance on a multirotor. In *Int. Sym. on Exp. Robot. (ISER)*. Springer, 2018.

[9] Marin Kobilarov. Cross-entropy randomized motion planning. *Robotics: Science and Systems VII*, 2012.

[10] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *IEEE Int. Conf. on Intell. Robots and Syst. (IROS)*. IEEE, 2017.

[11] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *Int. J. Robot. Res.*, 2010.

[12] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 2019.

[13] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *Journal of guidance, control, and dynamics*, 2002.

[14] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *J. Field Robot.*, 2009.

[15] Mihail Pivtoraiko, Daniel Mellinger, and Vijay Kumar. Incremental micro-uav motion replanning for exploring unknown environments. In *IEEE Int. Conf. on Robot. and Autom. (ICRA)*. IEEE, 2013.

[16] Wennie Tabib, Micah Corah, Nathan Michael, and Red Whittaker. Computationally efficient information-theoretic exploration of pits and caves. In *IEEE Int. Conf. on Intell. Robots and Syst. (IROS)*. IEEE, 2016.

[17] Mikko Lauri and Risto Ritala. Planning for robotic exploration based on forward simulation. *Robotics and Autonomous Systems*, 2016.

[18] Micah Corah and Nathan Michael. Distributed matroid-constrained submodular maximization for multi-robot exploration: Theory and practice. *Autonomous Robots*, 2019.

[19] Xuning Yang, Koushil Sreenath, and Nathan Michael. A framework for efficient teleoperation via online adaptation. In *IEEE Int. Conf. on Robot. and Autom. (ICRA)*, 2017.