PiP-X: Funnel-based Online Feedback Motion Planning/Replanning in Dynamic Environments

Mohamed Khalid M Jaffar and Michael Otte

University of Maryland, College Park, MD 20742, USA
{khalid26, otte}@umd.edu

Abstract. We propose an *online* single-query sampling-based feedback motion *re-planning* algorithm using finite-time invariant sets, *"funnels"*. We combine concepts from nonlinear systems analysis, sampling-based motion planning, and graph-search methods to create a single framework that enables *feedback motion planning/replanning* for general nonlinear dynamical systems in a dynamic workspace. We introduce a novel graph data structure to represent a network of *volumetric* funnels, enabling the use of quick graph-replanning techniques. The use of incremental search techniques and a pre-computed library of motion-primitives ensure that our method can be used for quick *on-the-fly* rewiring of controllable motion plans in response to changes in the environment. We validate our approach on a simulated 6DOF quadrotor platform operating in a maze, and random forest environment.

Keywords: feedback motion planning, online replanning, nonlinear systems analysis, sampling-based algorithms, incremental graph-search

1 Introduction

The ability to replan is essential whenever a robot must explore an unknown or changing environment while using a limited sensor radius. In scenarios where the operating workspace has fast-moving obstacles or is densely cluttered with obstacles, a motion-plan must be updated quickly and *on-the-fly*. Hence, there is a need for fast replanning algorithms. As the obstacle-space changes, the valid state-space also changes, and it is computationally expensive to rebuild motion plans from scratch in such non-convex spaces. Sampling-based planning methods, along with theoretical control techniques are often used in dynamic and high-dimensional spaces to provide a motion-plan that respects the kinematics and dynamics of the vehicle. *Feedback motion planning* considers the two sub-blocks of a robot-autonomy stack—motion planner and controller—in *tandem.*

For robots that must react quickly to changes in the environment, the computational complexity of brute-force replanning—planning from scratch whenever the environment changes—is often impractical. Quick replanning methods derive efficiency by repairing an old plan to reflect the updated state of the environment (in practice, the invalid portion is often a small fraction of the whole plan; in such cases it can be orders of magnitude quicker to *repair*, than to replan from

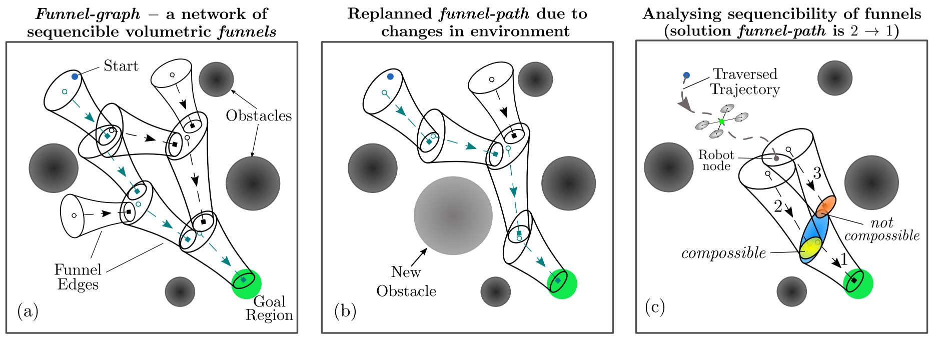

Funnel-graph – a network of sequencible volumetric *funnels*
Start
Obstacles
Funnel Edges
Goal Region
(a)
Replanned *funnel-path* due to changes in environment
New Obstacle
(b)
Analysing sequencibility of funnels (solution *funnel-path* is 2 → 1)
Traversed Trajectory
Robot node
3
2
1
not compossible
compossible
(c)

Fig. 1. Online funnel-based re-planning algorithm: PiP-X

scratch). However, the use of such methods has been limited to discrete grids and graphs in which edges represent one-dimensional trajectories through space-time. In this paper, we extend the use of such incremental search methods to the case of *volumetric funnels* by using a novel graph data structure representation.

The algorithm that we present is called PiP-X (Planning/replanning in Pipes in dynamic or initially unknown environments). Using systems analysis and invariant set theory, our approach computes dynamically feasible and verified trajectories with formal stability guarantees. The use of sampling techniques enables our algorithm to be computationally tractable for higher dimensional systems and configuration spaces. Additionally, our method is capable of using trajectory libraries to speedup online computation. Hence, our *kinodynamically-feasible*, *online re-planning* method finds relevance in practical scenarios such as safely navigating through dynamic environments. The **novelties** of our algorithm are summarized below.

- Feedback motion re-planning with funnels, using sampling-based techniques and incremental graph-search.
- A novel approach to compute motion plans with formal invariance guarantees for any nonlinear robot-system, that can be feasibly tracked by the robot.
- Representing the network of funnels and its inter-sequencibility using a *bipartite graph data structure*, enabling the use of fast graph-replanning methods to update kinodynamically-feasible safe motion-plans on-the-fly.

We believe this is the first work to propose techniques for *funnel-based online motion re-planning* using graph-based quick rewiring methods. A full exposition of our work—with detailed explanations, additional discussions and experimental results—appears in a *technical report* [1], included as supplementary material.

2 Related Work

A variety of kinodynamic planners [2,3] extend geometric sampling-based methods, such as RRT [4] and PRM [5], to address motion planning for differentially constrained robots. Such algorithms "steer" the vehicle by randomly sampling control inputs and simulating the trajectory based on the dynamics [6]. Some methods formulate an optimal control problem with the trajectory given by the planner, and solve it using shooting methods [7] or closed-form solutions [8]. Others smooth the path using splines or trajectory optimisation [9], and track it using feedback controllers such as PID or receding-horizon controllers [10].

Historically, robot-autonomy stacks have a hierarchical structure: the high-level path planner computes an open-loop trajectory, and a low-level controller tracks the trajectory. This decoupled approach can be disadvantageous because controller tracking errors, and uncertainties or actuator saturations might render the planned path infeasible to track. Tracking errors between the planned and actual trajectories can lead to critical failures such as collisions with obstacles. These shortcomings are addressed by *feedback motion planning*, in which the planner explicitly considers the stabilising feedback controller to generate motion-plans for dynamical continuous systems.

Tedrake et al. [11] popularised the notion of LQR-trees − an algorithm that covers the state-space using a tree of trajectories, locally stabilised by an LQR controller and verified by Lyapunov level-set theory. Majumdar et al. [12] compute the funnels offline, and use them to plan online for flying a glider through a dense setting. Funnel-based motion planning for a robotic arm using adaptive feedback control is presented in [13]. In parallel to such Lyapunov analysis, the controls community have proposed reachability set-based trajectory design [14] [15]. FaSTrack [16] proposes an adversarial game-theoretical approach to generate worst-case tracking error bounds around trajectories using Hamilton-Jacobi reachability analysis. Singh et al. [17] use contraction theory and convex optimisation to compute invariant-tubes around trajectories, and plan using them.

Earlier work on re-planning, such as D* [18] and D*Lite [19], are based on incremental, heuristic-guided shortest-path repairs on a *discrete grid* embedded in the robot's workspace. Such discretization assumes a constant resolution, and requires post-processing to achieve kinodynamic feasibility and controllability. RRT^X [20], an asymptotically optimal *sampling-based re-planning* algorithm, rewires the shortest-path subtree to exclude nodes and edges that are in collision.

Contributions: Our research differs from previous work in that it is the first *online global* feedback motion *re-planning* algorithm using funnels. Through the use of a graph to represent funnels and its sequencibility, we are able to recast the challenging problem of feedback motion planning for non-trivial systems into a geometric one, making it tractable for online re-planning. We implicitly address the *two-point boundary value problem* during graph-rewiring by using system stability analysis and trajectory sequencibility.

3 Preliminaries

3.1 Invariant Set Theory

The notion of region of attraction of asymptotically stable fixed points can be extended to certifying time-varying trajectories. Such regions of finite-time invariance around a trajectory are referred to as "*funnels*" [21]. Consider a closed-loop nonlinear system, $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t))$ with state $\boldsymbol{x} \in \mathbb{R}^n$, and $\boldsymbol{f}$ being Lipschitz continuous in $\boldsymbol{x}$ and piecewise continuous in t. For a finite time interval $[t_0, t_f]$, a *funnel* is formally defined as follows,

Definition 1. Funnel – *A set, $\mathcal{F} \subseteq [t_0, t_f] \times \mathbb{R}^n$, such that for each $(\tau, \boldsymbol{x}_\tau) \in \mathcal{F}$, $\tau \in [t_0, t_f]$, the solution to the system's equations, $\boldsymbol{x}(t)$, with initial condition $\boldsymbol{x}(\tau) = x_\tau$, lies entirely within $\mathcal{F}$ till final time, i.e. $(t, \boldsymbol{x}(t)) \in \mathcal{F}\ \forall\ t \in [\tau, t_f]$.*

Intuitively, if the closed-loop system starts within the funnel, then the system states evolving due to its dynamics remain within the funnel during the entire finite-time horizon. We leverage tools from Lyapunov theory to compute bounded, inner-approximations of the funnel. The compact level-sets $\mathcal{B}(t)$ of a candidate function, $V(t, \boldsymbol{x})$, satisfying the Lyapunov conditions ($V \geq 0$, $\dot{V} \leq 0$), is positively invariant. Then, the set $\mathcal{F}$ defined as in Eq. (1) is a *funnel*.

$$\mathcal{F} = \{(t, \mathcal{B}(t)) \mid t \in [t_0, t_f]\} \qquad \mathcal{B}(t) = \{\boldsymbol{x} \mid 0 \leq V(t, \boldsymbol{x}) \leq \rho(t)\} \tag{1}$$

As noted in [21], under certain mild assumptions, it is sufficient to analyse the boundary of the level sets $\partial\mathcal{B}(t)$, and the invariance conditions can be reformulated in terms of boundary function, $\rho(t)$ as: $V(t, \boldsymbol{x}) = \rho(t) \implies \dot{V}(t, \boldsymbol{x}) \leq \dot{\rho}(t)$. In other words, with respect to time, the Lyapunov function at the boundary, $\partial\mathcal{B}(t)$ should decrease faster than $\rho(t)$ for invariance. In our case, we are interested in computing backward-reachable sets to a compact region of state-space. Given a bounded space of desired final-states, referred to as *sub-goal region* $\mathcal{X}_f \subseteq \mathbb{R}^n$, we compute funnels that end within the region, i.e. $\mathcal{B}(t_f) \subseteq \mathcal{X}_f$.

3.2 Verified Trajectory Libraries

Maneuver Automaton, introduced in [22], discusses the relevant properties required for sequencing trajectories from a library of maneuvers. To analyse the sequencibility of funnels during online motion planning, we decompose the state vector into *cyclic* and *non-cyclic* states, $\boldsymbol{x} = \left[\boldsymbol{x}_c^T\ \boldsymbol{x}_{nc}^T\right]^T$. Cyclic states are defined as the coordinates to which the open-loop dynamics of a Lagrangian system are invariant, or alternatively, the dynamics depend only on the non-cyclic states, $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}'(t, \boldsymbol{x}_{nc}(t), \boldsymbol{u}(t))$.

For example, in a 2D disc robot, position is the cyclic coordinates, whereas velocity would be the non-cyclic counterpart. It is sufficient to verify whether the regions-of-invariance projected onto a subspace formed by the non-cyclic state coordinates are sequentially compossible [23]. One can shift the funnel along the cyclic coordinates so as to contain the outlet of the previous funnel.

Definition 2. *A funnel-pair, $(\mathcal{F}_i, \mathcal{F}_j)$ is said to be* **motion-plan compossible** *if and only if $\mathcal{P}^{\mathcal{S}}_{nc}(\mathcal{B}_i(t_{f_i})) \subseteq \mathcal{P}^{\mathcal{S}}_{nc}(\mathcal{B}_j(t_{0_j}))$, where $\mathcal{P}^{\mathcal{S}}_{nc}(.)$ is the projection operator from the state-space onto the subspace formed by non-cyclic coordinates.*

In addition to posing a less strict condition, the notion of *motion-plan compossibility* plays a significant role in designing the funnel library — one can use a finite number of motion primitives to cover an infinite subspace of cyclic coordinates by "shifting" the trajectories appropriately.

3.3 Discrete graph-based replanning using incremental search

Incremental search techniques reuse current valid information to recalculate the solution path in the next iteration, often achieving orders of magnitude speedup versus methods that replan from scratch. One way in which quick replanning algorithms, such as D*Lite [19], achieve efficiency is by maintaining a sorted queue of nodes that need to have their cost values repaired. We adopt a *rewiring-cascade* similar to D*Lite for repairing the shortest-path-to-goal subtree. For each node, the algorithm maintains an estimate of *cost-to-goal* value $g(v)$, defined as the sum of cost of all edges along the path from node v to the goal. Additionally, it computes an *lmc* value (one-step *lookahead minimum cost*) for all nodes, defined as, $lmc(v) = \min_{v' \in N^+(v)}\{c(v, v') + g(v')\}$, where $N^+(v)$ is the set of out-neighbors of v. The key idea is to compare the two values, g and *lmc*, to determine whether changes have occurred in the shortest path to the goal, explained as follows.

1. $g(v) = lmc(v) \implies v$ is *consistent* $\rightarrow$ no changes to the shortest path from v to goal.
2. $g(v) < lmc(v) \implies v$ is *under-consistent* $\rightarrow$ cost of the path to goal has increased, and we have to repair the entire (reverse) subtree rooted at v.
3. $g(v) > lmc(v) \implies v$ is *over-consistent* $\rightarrow$ a shorter path exists: update the parent and cost-to-goal of v, and propagate this cost-change information to the in-neighbors of v.

The algorithm does not repair all the nodes after every edge-cost change; rather, it repairs only *promising nodes* that have the "potential" to lie in the robot's shortest path to goal, determined using *heuristics*, g and *lmc* values. In typical graph-based motion planning algorithms, graph-vertices and edges represent configurations and trajectories, respectively. In contrast, in this paper, we use graphs to represent a network of *funnels* in a way that respects their volumetric nature and compossibility constraints (Definition 2).

4 Problem Formulation

Consider a Lagrangian robot-system with state, $\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{v}) \in \mathcal{S} \subseteq \mathbb{R}^{2d}$, where $\boldsymbol{q} \in \mathcal{C} \subseteq \mathbb{R}^d$ is the pose, $\boldsymbol{v} \in \mathbb{R}^d$ represents the velocities, and d is the dimension of $\mathcal{C}$-space. The robot operates in a workspace $\mathcal{W} \subseteq \mathcal{C}$, with finite number of obstacles occupying a subspace, $\mathcal{O} \subset \mathcal{W}$. Let $\mathcal{C}_{obs}$ be the open subset of configurations in which the robot is in-collision. $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$ is the closed subset of $\mathcal{C}$-space, in which the robot can "safely" operate without colliding with obstacles.

Definition 3. Funnel-edge – *Given a compact set $\mathcal{X}_w \subseteq \mathcal{C}$ centered around $w \in \mathcal{C}$, an initial configuration $v \in \mathcal{C}$, and finite time interval $[t_0, t_f]$,* ***funnel-edge*** *$\phi(v, w) \subseteq \mathcal{C}$ is the projection of* maximum-volume *funnel $\mathcal{F}$ satisfying (1), such that $v \in \mathcal{P}^{\mathcal{S}}_{\mathcal{C}}(\mathcal{B}(t_0))$ and $\mathcal{P}^{\mathcal{S}}_{\mathcal{C}}(\mathcal{B}(t_f)) \subseteq \mathcal{X}_w$. A funnel-edge is said to be* **valid**, *if and only if $\mathcal{P}^{\mathcal{C}}_{\mathcal{W}}(\phi(v, w)) \cap \mathcal{O} = \emptyset$. The cost of the funnel-edge is given by the length of the nominal trajectory $\boldsymbol{x}_0(t)$ projected down to $\mathcal{C}$-space, $\boldsymbol{q}_0(t)$, $c_\phi(v, w) = \int_{t_0}^{t_f} ds(t)$, where $ds = \|\boldsymbol{dq_0}\|_2$ (Euclidean norm), and $\mathcal{P}^A_B(.)$ is the projection operator from a space A to a lower dimensional subspace B.*

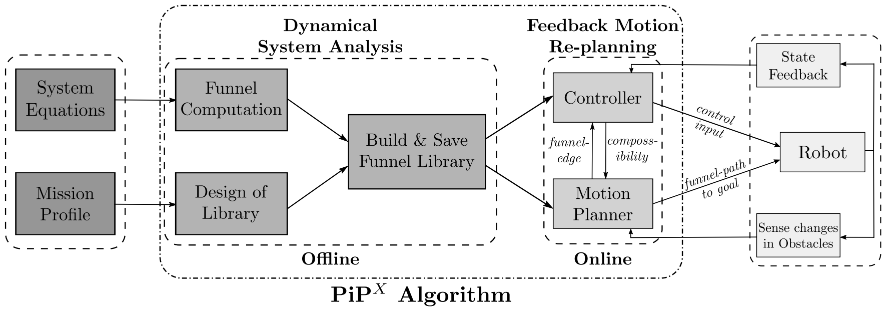


Fig. 2. Overview of PiP-X algorithm: our method consists of an offline stage of dynamical system analysis, and an online phase of sampling-based funnel-graph construction and incremental re-planning in dynamic environments

Definition 4. Funnel-path – *For a configuration $q_1 \in \mathcal{C}_{free}$, and a compact set $\mathcal{X}_2 \subseteq \mathcal{C}_{free}$, funnel-path $\pi(q_1, \mathcal{X}_2)$ is a finite sequence of* valid *funnel-edges with underlying* motion-plan compossibility, *i.e.* $\pi(q_1, \mathcal{X}_2) = \{\phi_1, \phi_2, \ldots \phi_n\}$, *such that* $q_1 \in \mathcal{P}^{\mathcal{S}}_{\mathcal{C}}(\mathcal{B}_{\phi_1}(t_{0_1}))$ *and* $\mathcal{P}^{\mathcal{S}}_{\mathcal{C}}(\mathcal{B}_{\phi_n}(t_{f_n})) \subseteq \mathcal{X}_2$. *The cost of the funnel-path is defined as* $c_\pi(q_1, \mathcal{X}_2) = \sum_{i=1}^{n} c_{\phi_i}$

Problem 1. **Online feedback motion planning** – Given $\mathcal{C}_{free}$, obstacle space $\mathcal{O}(t)$, and a goal region $\mathcal{X}_{goal}$, for a robot starting at a configuration, $q_{robot}(0) = q_{start} \in \mathcal{C}_{free}$, calculate the *optimal funnel-path*, $\pi^*(q_{robot}(t), \mathcal{X}_{goal})$, move the robot by applying a feedback control policy, $u : (\boldsymbol{x}, \boldsymbol{x}_0, t) \rightarrow \mathcal{U}$ and keep updating π^* until $q_{robot}(t) \in \mathcal{X}_{goal}$, with $\pi^*(q_{robot}(t), \mathcal{X}_{goal}) = \arg\min_\pi c_\pi(q_{robot}(t), \mathcal{X}_{goal})$

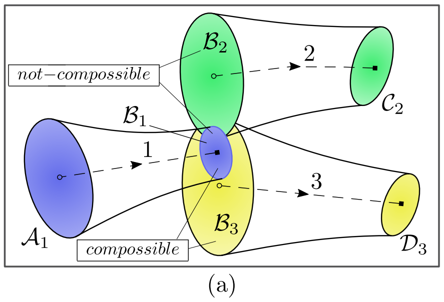
Problem 2. **Feedback motion replanning** – Assuming that the robot has an ability to sense obstacle changes $\mathit{\Delta}\mathcal{O}(t)$, continually recompute $\pi^*(q_{robot}(t), \mathcal{X}_{goal})$ until $q_{robot}(t) \in \mathcal{X}_{goal}$.

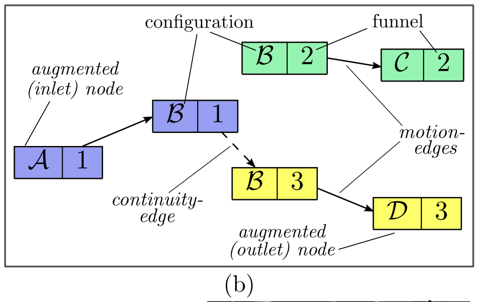
We propose techniques and algorithms to plan/replan *minimum-cost funnel-paths* on-the-fly. The funnel-path is a sequence of maneuvers with formal guarantees of invariance, associated with state-feedback tracking control policies. The continuously updated motion plan with invariant sets and control policies ensures that the robot trajectory always lies within the funnel path, avoids dynamic obstacles, and ultimately reaches the desired goal region.

5 Approach

We begin this section with a brief high-level outline of our approach, which involves: pre-computing a library of funnels, and online motion re-planner that keeps updating the optimal funnel-path. Each part will be discussed in greater detail in its own subsection, Sections 5.1–5.3. As illustrated in Fig. 2, our method has an offline stage of nonlinear system analysis using invariant set theory, and an online phase of sampling-based motion re-planning using incremental search.

We compute backward-reachable invariant sets, detailed in Section 5.3, using Lyapunov theory. Given an initial state, a compact set of desired final-states and





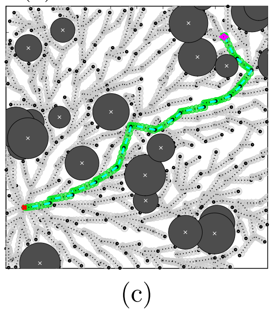


Fig. 3. (a) Funnels in $\mathbb{R}^+ \times \mathbb{R}^n$ (b) Corresponding *graph data structure* with *augmented* vertices representing the funnel-network. *Motion-edges* (solid) representing traversability are finite-cost, whereas dashed-edges are zero-cost *continuity-edges* encoding compossibility information — whether or not trajectories "flow" into the subsequent funnel. Note that going from $\mathcal{A}_1$ to $\mathcal{C}_2$ is infeasible, whereas $\mathcal{A}_1 - \mathcal{B}_1 - \mathcal{B}_3 - \mathcal{D}_3$ is a feasible *funnel-path.* (c) An example in a dynamic forest. Funnel-edges are light gray, dynamic obstacles are dark gray, the funnel-path is green, the start is magenta, and goal is red.

a finite-time horizon, we calculate the certified region of invariance characterised by Lyapunov level-sets centered around the nominal-trajectory. Due to the computational complexity of nonlinear system analysis, we *pre-construct* a library of verified trajectories with various combinations of initial and final states, to be used during the online-phase of feedback motion re-planning.

A network of funnels is incrementally built *online* using sampling methods (RRG), and motion plans/replans are computed through it. The *funnel-graph* embedded in the $\mathcal{C}$-space is constructed based on the aforementioned system analysis in the higher dimensional state-space. We analyse the *motion-plan compossibility* (Definition 2) among funnel-pairs, and additionally include that information in the form of an *augmented graph data structure* (see Fig. 3 and Section 5.1). With this graph, we compute a shortest-path subgraph (tree) of funnels, rooted at the goal using incremental search, introduced in Section 3.3.

Motion plans are recomputed *on-the-fly* in the event of changes in obstacle-space, $\Delta\mathcal{O}$, either due to robot sensing new obstacles or the obstacles being dynamic themselves. The funnel-path to the goal region and the corresponding sequence of control-inputs are input to the robot, with state-observer and obstacle-sensors closing the feedback-loop. Section 5.2 describes our feedback motion planning/replanning algorithm in-depth.

5.1 Graph data structure to represent a *volumetric funnel network*

Quick replanning using volumetric funnels requires that funnels and its inter-compossibility information be stored in a graph data structure that, both, (i) reflects the volumetric nature of funnels in the $\mathcal{C}$-space, and (ii) is compatible with existing fast graph-based replanning techniques. In this section, we present a data structure to represent the network of funnels, enabling the use of incremental graph-replanning techniques to quickly rewire funnel-paths (Definition 4). Our method essentially constructs "links" between regions of state-space with funnels that have an implicit notion of time. Traversability of the robot system and sequencibility of trajectories is represented through *motion-edges* and *continuity edges*, respectively, in the *augmented graph* $\mathcal{G}$.

The edge-set of this graph consists of motion-edges and continuity-edges, $E = E_m \cup E_c$. A graph-vertex, v is a tuple consisting of a configuration q, and the respective funnel f: $v - \boxed{q|f}$. As observed in Fig. 3-b, there are two types of graph vertices $-$ *inlet-nodes* and *outlet-nodes*, $V = \{V_I, V_O\}$. The authors would like to point out that the *augmented graph* is indeed *bipartite* with disjoint sets of inlet-nodes and outlet-nodes [1]. (Dashed) continuity-edges, e_c are *zero-cost*, and have no bearing on the cost of the solution funnel-path to goal, or the optimality guaranteed by the graph-search algorithm.

5.2 Online motion planning-replanning algorithm $-$ PiP-X

We incrementally build a reverse funnel-graph, abstract it into aforementioned graph data structure, and compute the optimal *funnel-path* using the routine$-$ `plan()` (Algorithm 3). The pre-planning process on a higher level is as follows.

1. Sample a configuration q_{rand}, and extend ϵ-distance from the nearest configuration in the existing funnel-graph to determine a new configuration q_{new}.
2. Determine the set of nearest neighbors (line 4) in the shrinking r-ball [24], $r = \min\{r_0(log|V|/|V|)^{1/d}, \epsilon\}$, where $|V|$ is cardinality of the set of configurations, d is the dimensionality of $\mathcal{C}$, and r_0 is a user-specified parameter.
3. From the funnel library $\mathfrak{L}$, choose the trajectory that would steer the robot from n to a δ-ball near q_{new} as well as the return trajectory from q_{new} to a δ-ball near n, for all n in the set of nearest neighbors.
4. Amongst the *funnel-edges* ϕ, check sequencibility, and "overlap" with obstacles. Denote the inlets $\mathcal{X}_i$ and outlets $\mathcal{X}_o$ as *nodes*; ϕ_i as a *motion-edge*, and zero-cost edges signifying compossibility as *continuity-edges* (Algorithm 4).
5. An incremental search (Algorithm 2) on the constructed sampling-based graph keeps updating the shortest-path tree of funnels rooted at goal.

After each update of the shortest-path subgraph (tree) (line 6), all consistent nodes in the graph know their best parent, enabling the planner to backtrack the solution funnel-path using parent pointers. The search is focused towards the robot using an admissible heuristic $h(v)$, thereby enabling quick rewiring of the optimal path, similar to the A* algorithm. If a nontrivial admissible heuristic cannot be found, using the trivially admissible $h(v) = 0$ will work.

The pseudocode of our *online re-planner*, PiP-X is presented in Algorithm 1. The inputs to the algorithm are start configuration q_{start}, goal region $\mathcal{X}_{goal}$, the pre-computed funnel library $\mathfrak{L}$, and the initial environment $-$ characterised through $\mathcal{W}$, and obstacles known *a priori*, $\mathcal{O}$. The obstacle-space will be updated when any changes, $\Delta\mathcal{O}(t)$ are discovered on-the-fly. The various subroutines of the algorithm, providing low-level implementation details, are now explained.

Algorithm 1 PiP-X

```
Input: $q_{start}$, $\mathcal{X}_{goal}$, $\mathcal{C}_{free}$, $\mathcal{O}$, $\mathfrak{L}$
Output: $\mathcal{F}, \mathcal{G}$                    ▷ Funnel-edges set, funnel-Graph
 1: Parameters: $\epsilon$, $r_0$, $T_P$, $I_M$
 2: Initialisation: $t \leftarrow 0$, $startFound \leftarrow 0$,
 3:             $\mathcal{G}$.add($\mathcal{X}_{goal}$), $\mathcal{F} \leftarrow \emptyset$
 4: while $t < T_P \lor \neg startFound$ do              ▷ Pre-planning
 5:     $(\mathcal{F}, \mathcal{G}) \leftarrow$ plan()
 6:     if inFunnel($q_{start}, \mathcal{F}$) then
 7:         $startFound \leftarrow 1$
 8: $j \leftarrow 0$, $q_{robot} \leftarrow q_{start}$, $q_{prev} \leftarrow q_{start}$
 9: while $j < I_M \land q_{robot} \notin \mathcal{X}_{goal}$ do          ▷ Online phase
10:     at sensingFrequency do ▷ Sensing obstacle-changes
11:         $\Delta\mathcal{O} \leftarrow$ senseObstacles()
12:         modifyEdgeCosts($\Delta\mathcal{O}$)
13:     $(\mathcal{F}, \mathcal{G}) \leftarrow$ plan()        ▷ Repairing the motion-plan
14:     at robotMotionFrequency do     ▷ Robot movement
15:         $q_{robot} \leftarrow$ robotMove($q_{robot}, q_{goal}$)
16:         if $g(q_{robot}) \neq \infty$ then          ▷ a funnel-path exists
17:             $k_m \leftarrow k_m +$ computeHeuristic($q_{prev}, q_{robot}$)
18:             $q_{prev} \leftarrow q_{robot}$; $j \leftarrow 0$   ▷ reset idleness count
19:         else
20:             $q_{robot} \leftarrow q_{prev}$           ▷ stay at current location
21:             $j \leftarrow j + 1$                ▷ update idleness count
22:     if $q_{robot} \in \mathcal{X}_{goal}$ then
23:         return SUCCESS                  ▷ Algorithm success
24: return NULL                             ▷ Algorithm failure
```

Algorithm 2 computeShortestPathTree()

```
 1: $k_{start} \leftarrow$ computeKey($q_{start}$)
 2: while $\mathcal{Q}$.topKey() $< k_{start} \lor lmc(q_{start}) \neq g(q_{start})$ do
 3:     $v \leftarrow \mathcal{Q}$.pop(); $k_{old} \leftarrow key(v)$
 4:     $k_{new} \leftarrow$ computeKey($v$)
 5:     if $k_{new} > k_{old}$ then              ▷ check & update key
 6:         $\mathcal{Q}$.push($v, k_{new}$)
 7:     else if $g(v) > lmc(v)$ then             ▷ over-consistent
 8:         $g(v) \leftarrow lmc(v)$
 9:         for all $u \in Pred(v)$ do updateVertex($u$)
10:     else                                   ▷ under-consistent
11:         $g(v) \leftarrow \infty$
12:         updateVertex($v$)
13:         for all $u \in Pred(v)$ do updateVertex($u$)
```

Algorithm 3 $(\mathcal{F}, \mathcal{G}) \leftarrow$ plan()

```
1: $q_{rand} \leftarrow$ sampleFree()
2: $q_{new} \leftarrow$ extend($\mathcal{G}, q_{rand}, \epsilon$)
3: $r \leftarrow$ rBall()
4: $\mathcal{N} \leftarrow$ findNearestNeighbors($q_{new}, r, \mathcal{G}$)
5: $(\mathcal{F}, \mathcal{G}) \leftarrow$ constructFunnelGraph($q_{new}, \mathcal{N}$)
6: computeShortestPathTree()
7: return $(\mathcal{F}, \mathcal{G})$
```

Algorithm 4 $(\mathcal{F}, \mathcal{G}) \leftarrow$ constructFunnelGraph($q_{new}, \mathcal{N}$)

```
 1: for all $n \in \mathcal{N}$ do
 2:     $\mathcal{F}_n^- \leftarrow$ steer($q_{new}, n$)              ▷ funnels out of $q_{new}$
 3:     $\mathcal{F}_n^+ \leftarrow$ steer($n, q_{new}$)              ▷ funnels into $q_{new}$
 4:     if $\mathcal{F}_n^- \neq \emptyset$ then
 5:         $\{\mathcal{X}_i, \mathcal{X}_o\} \leftarrow$ getNode($\mathcal{F}_n^-$)          ▷ inlet-outlet node
 6:         for all $\mathcal{F}_o \in outFunnels(n) \cup \{\mathcal{F}_n^+\}$ do
 7:             if compossible($\mathcal{F}_n^-, \mathcal{F}_o$) then
 8:                 $\mathcal{N}_i \leftarrow inletNode(\mathcal{F}_o)$
 9:                 $E_c \leftarrow E_c \cup (\mathcal{X}_o, \mathcal{N}_i)$           ▷ continuity-edge
10:         $V \leftarrow V \cup \{\mathcal{X}_i, \mathcal{X}_o\}$; $E_m \leftarrow E_m \cup (\mathcal{X}_i, \mathcal{X}_o)$
11:         updateVertex($\mathcal{X}_o$)
12:     if $\mathcal{F}_n^+ \neq \emptyset$ then
13:         $\{\mathcal{X}_i, \mathcal{X}_o\} \leftarrow$ getNode($\mathcal{F}_n^+$)          ▷ inlet-outlet node
14:         for all $\mathcal{F}_i \in inFunnels(n) \cup \{\mathcal{F}_n^-\}$ do
15:             if compossible($\mathcal{F}_i, \mathcal{F}_n^+$) then
16:                 $\mathcal{N}_o \leftarrow outletNode(\mathcal{F}_i)$
17:                 $E_c \leftarrow E_c \cup (\mathcal{N}_o, \mathcal{X}_i)$           ▷ continuity-edge
18:         $V \leftarrow V \cup \{\mathcal{X}_i, \mathcal{X}_o\}$; $E_m \leftarrow E_m \cup (\mathcal{X}_i, \mathcal{X}_o)$
19:     $\mathcal{F} \leftarrow \mathcal{F} \cup \{\mathcal{F}_n^-, \mathcal{F}_n^+\}$          ▷ adding to funnel-edges set
20: for all $\mathcal{F}_i \in inFunnels(q_{new})$ do
21:     for all $\mathcal{F}_o \in outFunnels(q_{new})$ do
22:         if compossible($\mathcal{F}_i, \mathcal{F}_o$) then
23:             $\mathcal{X}_o \leftarrow outletNode(\mathcal{F}_i)$
24:             $\mathcal{X}_i \leftarrow inletNode(\mathcal{F}_o)$
25:             $E_c \leftarrow E_c \cup (\mathcal{X}_o, \mathcal{X}_i)$               ▷ continuity-edge
26: return $\mathcal{F}$, $\mathcal{G} = (V, E_m, E_c)$
```

Algorithm 5 updateVertex(v)

```
1: $lmc(v) \leftarrow$ computeLMC($v$)
2: $parent(v) \leftarrow$ findParent($v$)
3: if $v \in \mathcal{Q}$ then
4:     $\mathcal{Q}$.remove($v$)
5: if $g(v) \neq lmc(v)$ then
6:     $key(v) \leftarrow$ computeKey($v$)
7:     $\mathcal{Q}$.push($v, key(v)$)
```

Algorithm 6 modifyEdgeCosts($\Delta\mathcal{O}$)

```
1: for all $e = (v, w) \in E_m$ do
2:     if ¬collisionFree($e, \Delta\mathcal{O}$) then
3:         $c(v, w) \leftarrow \infty$
4:     else
5:         $c(v, w) \leftarrow c_{prev}$
6:     updateVertex($v$)
```

Sampling and Graph extension: The $\mathcal{C}$-space is explored using sampleFree() and extend() routines (Algorithm 3, lines 1–2). The configurations q_{rand} are independent and identically (i.i.d.) drawn from the free-space, $\mathcal{C}_{free}$ at random. We determine the nearest configuration in the existing funnel-graph, based on geodesic distance, and extend by at most an ϵ-distance to obtain a new configuration q_{new}. If this configuration is already in one of the funnel-inlets, we discard and continue with the next sampling, because we are guaranteed to find a set of maneuvers which can drive the robot from this configuration to the goal.

Constructing the search *funnel-graph* (Algorithm 4): We construct funnels, whenever possible, between the new configuration q_{new} and all of the nodes in the neighbor-set, $n \in \mathcal{N}$, by using the subroutine steer(). The inlet-nodes $\mathcal{X}_i$, and outlet nodes $\mathcal{X}_o$ are added to the set of graph-vertices V, and the directed edge, $(\mathcal{X}_i, \mathcal{X}_o)$, is added to the set of *motion-edges*, E_m. Sequencibility amongst the newly constructed funnels is represented through zero-cost *continuity-edges* between the corresponding outlets and inlets. The subroutine compossible() checks whether a funnel-pair is motion-plan compossible as defined in (2). The ellipsoid-in-ellipsoid check is by approximating the outlet-ellipsoid into a convex hull by sampling points on the boundary of the ellipsoid, $\partial\mathcal{E}_o$. The extreme-points are chosen based on singular-value decomposition of the ellipsoid matrix M_o, and checked whether it lies in the interior of $\mathcal{E}_i$.

Invoking `updateVertex`() in line 11 ensures propagation of cost-changes and possible rewiring of the shortest-path subgraph (tree) of funnels due to the new sample. The cost-to-goal value of all the new nodes, $g(v)$ is initialised to be infinite by default. By the virtue of the nodes being inconsistent (specifically over-consistent), they are pushed into the priority queue $\mathcal{Q}$, and will be repaired if they have the "potential" to lie in the solution funnel-path to goal.

Computing the shortest-path tree (Algorithm 2): Inconsistent nodes are popped out of the priority queue $\mathcal{Q}$ and repaired, i.e. made consistent until the robot's start-node becomes consistent, or the queue becomes empty (usually encountered during the pre-planning phase). So, in effect, the shortest-path to goal from each "promising" node in the search-graph, based on cost-to-goal and heuristic (Euclidean distance), is quickly determined. `updateVertex`(v) (Algorithm 5) computes *lmc* value of vertex v, and determines the best parent of the node by analysing its outNeighbors. The node v is removed from the priority queue $\mathcal{Q}$, its key value is updated, and added to the queue only if inconsistent.

Robot motion: The various modules in Algorithm 1: sensing (lines 10−12), planning/replanning (line 13) and robot motion (lines 14−21) have different operating frequencies. `robotMove`(q_1, q_2) in line 15 determines and applies the corresponding control policy to move from q_1 to q_2. The changes to the obstacle set, $\Delta\mathcal{O}$ are estimated using sensors on the robot, and the cost of affected edges are updated using `modifyEdgeCosts`($\Delta\mathcal{O}$) (Algorithm 6) in line 12.

The *pre-planning* phase in Algorithm 1 (lines 4−7) continues until the start configuration lies within one of the funnel-inlets, and the funnel-graph is dense enough to have covered a sufficient volume of the $\mathcal{C}$-space (ensured using a pre-planning timeout, T_p). A solution funnel-path exists if the robot configuration lies in one of the inlet-nodes and has a finite cost-to-goal value. A funnel $\mathcal{F}$ is said to be *in-collision* with obstacle-set $\mathcal{O}$, if $\mathcal{P}^{\mathcal{S}}_{\mathcal{W}}(\mathcal{F}) \cap \mathcal{O} \neq \emptyset$, where $\mathcal{P}^{\mathcal{S}}_{\mathcal{W}}(.)$ projects down to the workspace. Assuming obstacles with locally Lipschitz continuous boundaries, we use convex-hull approximations to check for collision with funnels.

5.3 Precomputing regions of finite-time invariance

Determining a closed-form solution to Eq. (1) for a general class of Lyapunov functions is not guaranteed, and is computationally intractable. Under certain assumptions such as polynomial closed-loop dynamics, and quadratic Lyapunov functions, the problem of computing the funnels can be reformulated into a Sum-of-Squares (SoS) program [11]. Consider a *quadratic Lyapunov candidate function*, defined using a positive definite matrix $P(t)$ centred around the nominal trajectory $\boldsymbol{x}_0(t)$: $V(t, \boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}_0(t))^T P(t)(\boldsymbol{x} - \boldsymbol{x}_0(t))$. For the class of piecewise polynomials $P(t)$, we solve the SoS program using polynomial $\mathcal{S}$-procedure.

The convex optimisation problem of maximising the funnel-volume while satisfying Lyapunov conditions is solved using bilinear alternation — improving $\rho(t)$

and finding Lagrange multipliers to satisfy negativity of $(\dot{V}(t,\boldsymbol{x}) - \dot{\rho}(t))$ in the semi-algebraic sets [21]. For $M \in S_{+}^{n}$, set of $n \times n$ symmetric, positive definite matrices and $\boldsymbol{c} \in \mathbb{R}^n$, $(\boldsymbol{x}-\boldsymbol{c})^T M(\boldsymbol{x}-\boldsymbol{c}) = 1$ represents an ellipsoid centred around $\boldsymbol{c}$. The level sets $\mathcal{B}(t)$ in Eq. (1), corresponding to the quadratic Lyapunov function, are the closed set—interior and boundary—of ellipsoids centered around the nominal trajectory. The corresponding *ellipsoidal matrix* is: $\mathcal{E}(t) = P(t)/\rho(t)$.

The region of desired final-states, referred to as *sub-goal*, $\mathcal{X}_f$ is assumed to be an ellipsoid, defined by $\mathcal{E}_f$ centred at the final state. Computing the *maximal inner-approximation* of the backward-reachable invariant set to the sub-goal region is formulated as an SoS program, and the resulting semi-definite program (SDP) is numerically solved. As noted in [21], we observe that time-sampled relaxations in the semi-definite program improve computational efficiency while closely resembling the actual level-sets.

We *pre-construct* a library of funnels for quick online implementation due to the computational infeasibility of computing such regions of invariance on-the-fly. The *funnel library* $\mathfrak{L}$, consists of a finite number of verified trajectories, encapsulating the information of the certified regions of invariance in finite-time interval. Each funnel $\mathcal{F}_i \in \mathfrak{L}$, is parametrized by the nominal trajectory $\boldsymbol{x}_{0_i}(t)$, the ellipsoidal level-sets $\mathcal{E}_i(t)$, and the finite time horizon $[0, T_{f_i}]$.

The funnel library acts as a bridge between the offline and online phases—invariant set analysis and motion re-planning. Therefore, certain algorithm parameters such as extend-distance ϵ, resolution of the planner, range of obstacle sizes, etc. is considered while constructing the library. Meanwhile, the library provides the vital information of compossibility required during the online phase of motion planning. It is worth mentioning that the initial conditions of the finite number of projected trajectories in the library, with the appropriate shift operator, $\Psi_c(.)$ along the cyclic coordinates should be able to span the entire $\mathcal{C}$-space. This ensures *probabilistic coverage* [11] of the sampling-based motion re-planning algorithm and hence, probabilistic completeness of our algorithm.

6 Experimental Validation

We validate our algorithm on a *quadrotor* UAV in simulations, flying through a space with dynamic obstacles. We demonstrate the relevance of invariant sets and empirically verify the completeness and correctness of our motion re-planning algorithm. The equations of motion of a quadrotor UAV are derived using Newton-Euler formulation. The system-states, $\boldsymbol{x}$ consists of position, orientation, linear velocities, and body rates, with 4 rotor speeds as inputs.

The *controller architecture* has a cascaded structure, with a fast inner loop stabilising the attitude and a outer loop tracking the position [25]. We implement a nested P-PID loop for attitude-tracking. Based on the desired angles, the proportional controller computes the desired angular body rates which are then tracked using a PID controller. The outer loop tracks the desired position setpoints, and is achieved using an LQR controller. Equivalently, the inputs to the quadrotor position controller are the desired setpoints $-$ $[x_d, y_d, z_d, \psi_d = 0]^T$.

For a quadrotor in 3D environment, the configuration space $\mathcal{C} = \mathbb{R}^3$, whereas the state-space of the quadrotor translational subsystem is $\mathcal{S} = \mathbb{R}^3 \times \mathbb{R}^3$, comprising of the position and linear velocities. The mission profile is to fly at a set altitude, $z_d = h$ with a zero heading-angle, $\psi_d = 0$. Owing to the reduced operation-space, the workspace and the sampling is in $\mathbb{R}^2$. The equivalent *closed-loop position* dynamics of the quadrotor is derived from repeated trials with various position setpoints, $\boldsymbol{\xi}_d$ given as inputs to the system: $\dot{\boldsymbol{\xi}} = \boldsymbol{v}, \;\; \dot{\boldsymbol{v}} = \boldsymbol{f}(\boldsymbol{\xi}, \boldsymbol{v}, \boldsymbol{u}), \;\; \boldsymbol{u} = \boldsymbol{\xi}_d$

An estimate of the required finite-time horizon, $[0, T]$ is obtained based on the time taken by the system to reach the defined goal region of $0.3m$ around a desired setpoint. Subsequently, the invariant sets centered around the nominal trajectory are constructed using the methods described in Section 5.3. The SoS optimisation is converted to an SDP by Systems Polynomial Optimisation Toolbox in MATLAB, and solved using SeDuMi [26]. In order to verify the motion plans, we develop a higher fidelity model, additionally incorporating rotor dynamics, actuator saturations, and process noise. We believe these additions will enable the simulation model to more closely resemble the physical system.

7 Results and Discussions

Our algorithm is tested through *repeated trials* in two scenarios — *initially unknown maze* and *dynamically changing random forest* (Figures 4 and 3-c). The performance metrics we evaluate are success rate and traversed trajectory length. Any trial in which the robot collides with a (dynamic) obstacle, or does not find a valid plan before a timeout is defined as a failure, i.e., not a success.

Initially-unknown Maze with finite-sensing: We design a maze with rectangular walls in a workspace of dimensions $50m \times 50m$. The robot is assumed to have a limited sensing radius of $12m$. 25 trials are run for each of 10 different scenarios of start/goal configurations, resulting in 250 different trials. We only use start and goal locations that are initially obstacle free. Under these conditions, the robot was successful in all of our experimental trials, resulting in a 100% success rate across 10 different scenarios. Traversed-trajectory length had high variability due to the fact that the presence or absence of obstacle-walls in key locations could increase or decrease path length by many multiples. A sample run of PiP-X in a user-specified maze environment is shown in Fig. 4.

The solution funnel-path to goal is input to the higher-fidelity simulation-system in real-time, and the system's actual trajectory is analysed. It is observed that the system-trajectory lies within the solution funnel-path throughout the course of the mission profile, verifying set-invariance. In another scenario with different start/goal location, we examine the normalised Lyapunov function value of the system. From Fig. 5-a, we notice that the trajectory stays within the level-set boundary of $V = 1$ till the quadrotor-system reaches the goal region, empirically proving invariance. The peaks in the Lyapunov function value mostly occur in the outlet/inlet region between two funnels.

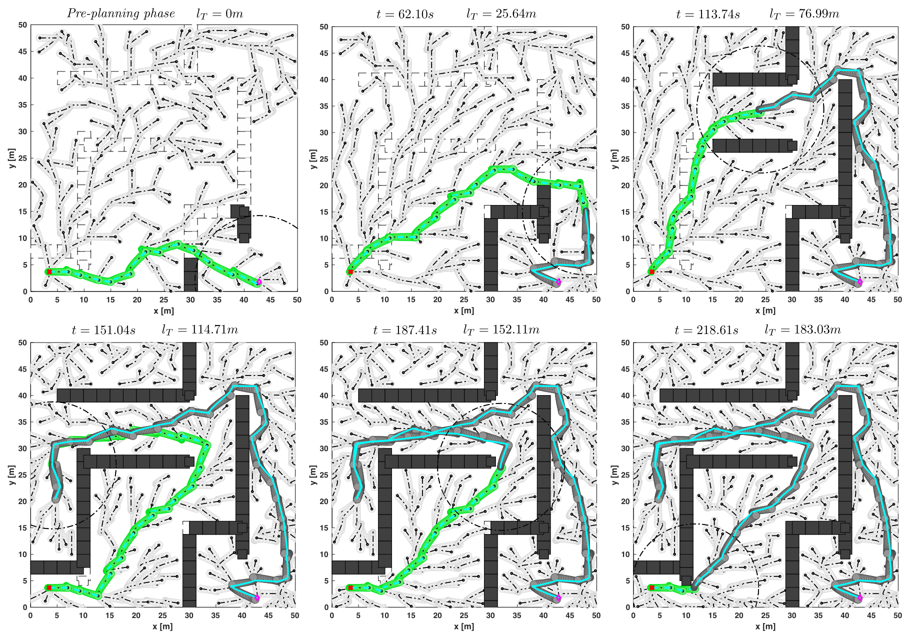


Fig. 4. Time instances of motion plan executed by PiP-X on a quadrotor flying with altitude-hold. The start configuration is in the lower-right corner, with the goal location at lower-left corner. The quadrotor senses obstacle-walls (solid rectangles) within sensor-radius (dashed circle), and recomputes motion plans (green *funnel-path*) accordingly. The traversed funnel-path and funnel-tree are denoted by dark and light gray, respectively. l_T denotes the traversed-path length, and t denotes time elapsed.

Random Forest with dynamic obstacles: We consider a workspace of dimensions $50m \times 50m$ with circular obstacles of random sizes in the range of $[2\ 4]m$ (see Figure 3-c). Dynamic tree-obstacles are deleted and added at random. The changes occur anywhere in the workspace and the robot is capable of sensing all such changes. A scenario is described by number of trees N_t, and change-percentage C. For e.g., in a workspace with $N_t = 45$, $C = 60\%$ implies 27 pre-existing tree-obstacles are removed, and 27 new obstacles are added − changing location and size. We run 25 trials (different starts and goals) for various combination of N_t and C. Values of N_t are varied over the range [5, 135] in increments of 10, while C is varied from 0 to 100 in steps of 10, resulting in 3850 total trials. Mean results are depicted in the form of a contour plot, Fig. 5. We observe that the algorithm failure and mean trajectory-length increases with either increasing number of tree-obstacles, $N_t \geq 45$, or higher level of changes, $C \geq 50$. It completely fails when the environment is densely cluttered with obstacles or highly dynamic (upper right triangle of the contour in Fig. 5-b). Note that trajectory-length in static environments ($C = 0\%$) with $N_t \geq 95$ is not visualised in Fig. 5-c as they are isolated instances of algorithm successes.

Most of the failures we observe are due to (idleness) time-outs by which our algorithm is not able to compute a solution. Success rate is 100% for easy

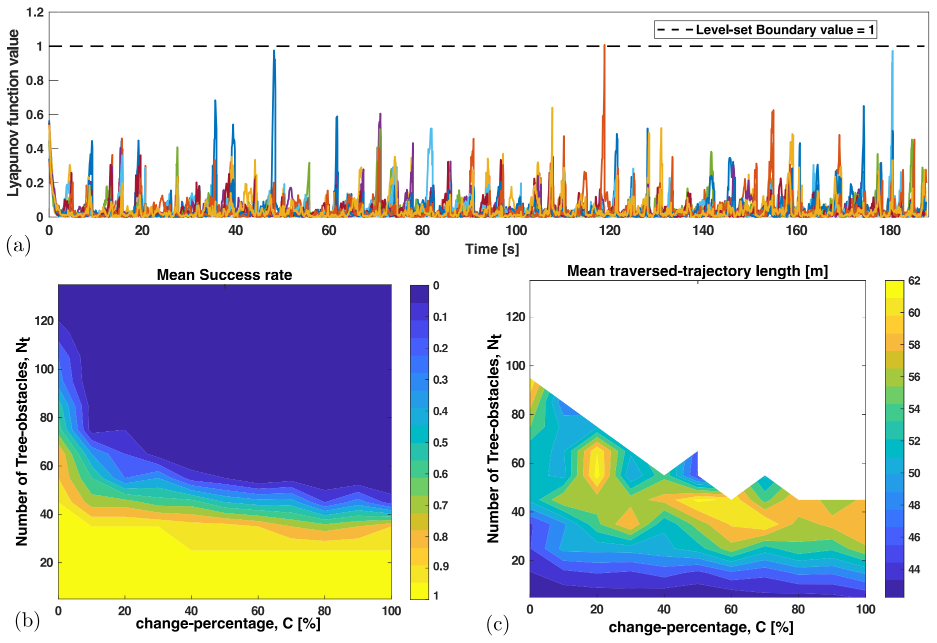


Fig. 5. (a) *Normalised* Lyapunov function value of system's state simulated using the higher-fidelity model, with solution funnel-path given by the algorithm across 10 scenarios (denoted by different line-colors) (b)-(c) Forest environment with dynamic obstacles: mean of performance metrics from 25 trials for each grid-point in the contour

scenarios, and decreases with problem difficulty (Fig. 5-b,c). Indeed, this is a common trait of sampling-based motion planning algorithms, in general. Another common reason for failure is the algorithm's inability to fit a volumetric region of space in narrow gaps, especially in dense-cluttered environments. In scenarios with highly-dynamic obstacles, an obstacle is more probable to appear on the traversing funnel-edge, inevitably leading to a collision with the obstacle.

In most scenarios, our algorithm is able to compute motion plans and repair them on-the-fly, ensuring a sequence of safe trajectories that are dynamically feasible. The theoretical guarantee of set-invariance enables our algorithm to rewire controllable motion plans, implicitly addressing the two-point boundary value problem that is typically encountered during search-tree rewiring. Computing a shortest-path tree of funnels rooted at the goal results in an optimal funnel-path with respect to the iteratively-constructed underlying funnel-graph.

8 Conclusions

We present a novel *online funnel-based* motion *re-planning* algorithm, PiP-X, which uses sampling-based techniques to iteratively construct a search-graph that considers funnel-edges. The information of robot-traversability and funnel-sequencibility is represented together in the form of an *augmented directed-graph*, that can be quickly rewired using incremental graph-replanning techniques. This combination of systems analysis and control, sampling-based methods, and incremental graph-search enables *feedback motion re-planning* for nonlinear robot-systems in dynamic workspaces. Our method can be used to compute safe, controllable motion-plans — and also to quickly and efficiently recompute them *on-the-fly* as obstacles appear and disappear.

PiP-X ensures kinodynamic feasibility of the solution-paths by analysing and formally quantifying stability of trajectories using Lyapunov level-set theory. Verifying the compossibility of a funnel-pair is also a "relaxed" alternative to the two-point boundary value problem, encountered in most single-query sampling-based motion planners that require rewiring. Our technique is validated on a simulated quadrotor platform in a variety of scenarios, including an initially unknown maze and a dynamic forest environment.

Acknowledgements

The authors wish to thank the Robot Locomotion Group at MIT for providing an open-source software distribution for computing funnels. This work is supported by the grant N00421-21-1-0001 from Naval Air Systems Command (NAVAIR).

References

1. Jaffar, M.K.M., Otte, M.: PiP-X: Online feedback motion planning/replanning in dynamic environments using invariant funnels. arXiv preprint arXiv:2202.00772 (2022)
2. Hsu, D., Kindel, R., Latombe, J.C., Rock, S.: Randomized kinodynamic motion planning with moving obstacles. The International Journal of Robotics Research **21**(3), 233–255 (2002)
3. Karaman, S., Frazzoli, E.: Optimal kinodynamic motion planning using incremental sampling-based methods. In: 49th IEEE conference on decision and control (CDC), pp. 7681–7687. IEEE (2010)
4. Kuffner, J.J., LaValle, S.M.: RRT-connect: An efficient approach to single-query path planning. In: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), vol. 2, pp. 995–1001. IEEE (2000)
5. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE transactions on Robotics and Automation **12**(4), 566–580 (1996)
6. LaValle, S.M., Kuffner Jr, J.J.: Randomized kinodynamic planning. The international journal of robotics research **20**(5), 378–400 (2001)
7. hwan Jeon, J., Karaman, S., Frazzoli, E.: Anytime computation of time-optimal off-road vehicle maneuvers using the RRT. In: 2011 50th IEEE Conference on Decision and Control and European Control Conference, pp. 3276–3282. IEEE (2011)
8. Webb, D.J., Van Den Berg, J.: Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In: 2013 IEEE International Conference on Robotics and Automation, pp. 5054–5061. IEEE (2013)

9. Ravankar, A., Ravankar, A.A., Kobayashi, Y., Hoshino, Y., Peng, C.C.: Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges. Sensors **18**(9), 3170 (2018)
10. Basescu, M., Moore, J.: Direct NMPC for post-stall motion planning with fixed-wing UAVs. In: 2020 IEEE International Conference on Robotics and Automation (ICRA), pp. 9592–9598. IEEE (2020)
11. Tedrake, R., Manchester, I.R., Tobenkin, M., Roberts, J.W.: LQR-trees: Feedback motion planning via sums-of-squares verification. The International Journal of Robotics Research **29**(8), 1038–1052 (2010)
12. Majumdar, A., Tedrake, R.: Funnel libraries for real-time robust feedback motion planning. The International Journal of Robotics Research **36**(8), 947–982 (2017)
13. Verginis, C.K., Dimarogonas, D.V., Kavraki, L.E.: KDF: Kinodynamic motion planning via geometric sampling-based algorithms and funnel control. arXiv preprint arXiv:2104.11917 (2021)
14. Bajcsy, A., Bansal, S., Bronstein, E., Tolani, V., Tomlin, C.J.: An efficient reachability-based framework for provably safe autonomous navigation in unknown environments. In: 2019 IEEE 58th Conference on Decision and Control (CDC), pp. 1758–1765. IEEE (2019)
15. Kousik, S., Vaskov, S., Bu, F., Johnson-Roberson, M., Vasudevan, R.: Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots. The International Journal of Robotics Research **39**(12), 1419–1469 (2020)
16. Herbert, S.L., Chen, M., Han, S., Bansal, S., Fisac, J.F., Tomlin, C.J.: FaSTrack: A modular framework for fast and guaranteed safe motion planning. In: 2017 IEEE 56th Annual Conference on Decision and Control (CDC), pp. 1517–1522. IEEE (2017)
17. Singh, S., Landry, B., Majumdar, A., Slotine, J.J., Pavone, M.: Robust feedback motion planning via contraction theory. The International Journal of Robotics Research (2019)
18. Stentz, A., et al.: The focussed D* algorithm for real-time replanning. In: IJCAI, vol. 95, pp. 1652–1659 (1995)
19. Koenig, S., Likhachev, M.: D*Lite. Aaai/iaai **15** (2002)
20. Otte, M., Frazzoli, E.: RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning. The International Journal of Robotics Research **35**(7), 797–822 (2016)
21. Tobenkin, M.M., Manchester, I.R., Tedrake, R.: Invariant funnels around trajectories using sum-of-squares programming. IFAC Proceedings Volumes **44**(1), 9218–9223 (2011)
22. Frazzoli, E., Dahleh, M.A., Feron, E.: Maneuver-based motion planning for nonlinear systems with symmetries. IEEE transactions on robotics **21**(6), 1077–1091 (2005)
23. Majumdar, A., Tedrake, R.: Robust online motion planning with regions of finite time invariance. In: Algorithmic foundations of robotics X, pp. 543–558. Springer (2013)
24. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. The international journal of robotics research **30**(7), 846–894 (2011)
25. Jaffar, M.K.M., Velmurugan, M., Mohan, R.: A novel guidance algorithm and comparison of nonlinear control strategies applied to an indoor quadrotor. In: 2019 Fifth Indian Control Conference (ICC), pp. 466–471. IEEE (2019)
26. Sturm, J.F.: Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Optimization methods and software **11**(1-4), 625–653 (1999)