

# PiP-X: Online feedback motion planning/replanning in dynamic environments using invariant funnels

Mohamed Khalid M Jaffar and Michael Otte

**Abstract**—Computing kinodynamically feasible motion plans and repairing them on-the-fly as the environment changes is a challenging, yet relevant problem in robot-navigation. We propose a novel *online single-query sampling-based motion re-planning* algorithm – PiP-X, using finite-time invariant sets – “*funnels*”. We combine concepts from sampling-based methods, nonlinear systems analysis and control theory to create a single framework that enables feedback motion re-planning for any general nonlinear dynamical system in dynamic workspaces.

A volumetric *funnel-graph* is constructed using sampling-based methods, and an optimal *funnel-path* from robot configuration to a desired goal region is then determined by computing the shortest-path subtree in it. Analysing and formally quantifying the stability of trajectories using Lyapunov level-set theory ensures kinodynamic feasibility and guaranteed set-invariance of the solution-paths. The use of incremental search techniques and a pre-computed library of motion-primitives ensure that our method can be used for quick online rewiring of controllable motion plans in densely cluttered and dynamic environments.

We represent traversability and sequencibility of trajectories together in the form of an *augmented directed-graph*, helping us leverage discrete graph-based replanning algorithms to efficiently recompute feasible and controllable motion plans that are volumetric in nature. We validate our approach on a simulated 6DOF quadrotor platform in a variety of scenarios within a maze and random forest environment. From repeated experiments, we analyse the performance in terms of algorithm-success and length of traversed-trajectory.

**Index Terms**—Feedback motion planning, online replanning, sampling-based algorithms, incremental graph-search, nonlinear systems, invariant set theory, motion primitives

## I. 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 the motion plans in such non-convex spaces. Hence, numerical methods such as sampling-based planning (and re-planning) techniques are often used in dynamic and high-dimensional spaces. However, we still require theoretical control techniques to ensure kinodynamic feasibility and trackability of generated motion plans. This dual requirement of motion planning and

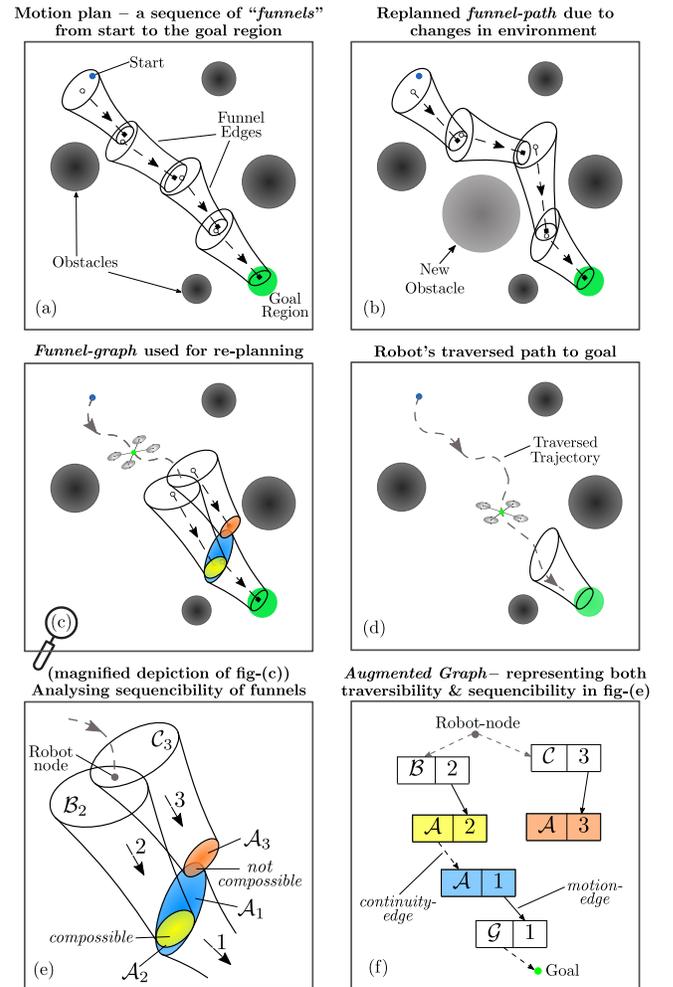


Fig. 1: Online funnel-based re-planning algorithm, PiP-X– (a) Motion plan, a sequence of finite-time invariant sets – *funnels*, each “dropping” into the subsequent funnel, finally into the goal region. (b) Rewiring of the *funnel-path* when changes in the environment are sensed (c) Underlying search-graph with funnel-edges (d) Trajectory of the robot lies completely inside the traversed funnel-path (e) Analysing sequencibility of funnels (f) Encoding the information of traversability (solid *motion-edges*) and compossibility (dashed *continuity-edges*) in an *augmented graph* to enable quick and efficient graph-based re-planning. The solution path is  $B_2-A_2-A_1-G_1$

This work is supported by the Naval Air Systems Command (NAVAIR) under the grant N00421-21-1-0001.

The authors are affiliated with the Department of Aerospace Engineering, University of Maryland, College Park, MD 20742, USA. Email: {khalid26, otte}@umd.edu

robot control is considered in tandem by recent literature in *feedback motion planning*.

While brute-force replanning—planning from scratch when-

ever the environment changes—may work for simple systems, the computational complexity of doing so is often impractical, especially for robots that must react quickly to new information about the changes in the environment. It is much more efficient to reuse the valid portions of previous plans, repairing only the invalid parts to respect the new changes. Incremental search methods which utilise information up to the current iteration to plan/replan in the future can be adapted to achieve quick replanning. This paper presents a novel sampling-based online motion re-planning algorithm using verified trajectories a.k.a. *funnels*; our work utilises a novel graph representation for a network of funnels that allows the use of incremental graph search for replanning on-the-fly.

Our work extends sampling-based replanners to robust kinodynamic settings and dynamic environments. Using system 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, the method is capable of using trajectory libraries to speedup online computation. Hence, our *kinodynamically-feasible, on-line re-planning* method finds relevance in practical scenarios such as safely navigating through dynamic environments. The **novelties** of our algorithm are summarized as follows,

- 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 non-holonomic robot-system, that can be feasibly tracked by the robot
- Representing robot traversability and compossibility of funnels as an *augmented bipartite graph*, enabling the use of discrete graph-search methods to quickly compute kinodynamically-feasible safe motion-plans on-the-fly
- Ability to use a pre-computed library of motion primitives with guaranteed regions-of-invariance for quick online replanning

We believe sampling-based planning using funnels is elegant due to its set-invariance, and use of a limited number of volumetric verified-trajectories to probabilistically cover the configuration space. It can also serve as a bridge between discrete numerical methods (path-planning algorithms) and theoretical analysis (system analysis and control design). We can reconcile the two sub-blocks in a robot-autonomy stack—motion planner and controller—using funnels, thus addressing *feedback motion re-planning*. In essence, we get a planning algorithm that respects the closed-loop dynamics of the robot, computes the minimum-cost path to the goal region, and also efficiently replans around dynamic obstacles.

The authors believe this is the first work to propose techniques for *funnel-based motion re-planning* using incremental graph-search. It sets the foundations for feedback motion replanning on-the-fly by providing a new form of augmented *volumetric* search-graph that is compatible with sampling-based techniques and invariant set analysis. It can additionally provide computation speed-ups using a funnel library. The authors stress that we are not suggesting an improved way to

compute funnels, but rather a new technique of using existing efficient methods of funnel computation to motion plan/replan for any general nonlinear robot-system in dynamic spaces.

### A. Statement of Contributions

The **technical contributions** of our work are three-fold,

- 1) PiP-X<sup>1</sup> – Online single-query sampling-based *feedback motion re-planning* algorithm using *funnels*.
- 2) A novel technique to represent funnel-compossibility using a directed graph embedded within the planning graph, resulting in an *augmented bipartite search-graph*.
- 3) Implicitly addressing the *two-point boundary value problem* (TP-BVP) during graph-rewiring by using the system’s stability analysis and trajectory sequencibility.

### B. Outline

The remainder of this paper is structured as follows: Section II outlines the related work, and Section III introduces the notation while providing the necessary theoretical background. Section IV formally states our problem definition. Our approach is described in Section V and validated in Section VI. Finally, we state our conclusions in Section VII.

## II. RELATED WORK

PiP-X builds on existing literature in the fields of feedback motion planning, sampling-based techniques, and online replanning in dynamic environments. It differs from previous work in that it is the first *online feedback motion re-planning* algorithm using *funnels*.

### A. Sampling-based kinodynamic motion planning

Geometric sampling-based motion planners such as probabilistic roadmaps (PRM) [1] and rapidly exploring random trees (RRT) [2] are relevant for planning in high-dimensional dynamical systems. Karaman and Frazzoli [3] propose RRT\*, and provide theoretical proofs of asymptotic optimality. RRT# algorithm presented in [4] improves the convergence rate, making it suitable for online implementation.

In order to address motion planning for differentially constrained robots, numerous researchers have extended such geometric planners to kinodynamic systems [5] [6]. Such algorithms “steer” the vehicle by randomly sampling control inputs and forward simulating the trajectory based on the dynamics [7] [8]. Some researchers formulate an optimal control problem with the trajectory given by the geometric planner, solved using shooting methods [9] or closed-form analytical solutions [10]. Another popular approach is to smooth the path given by geometric planners through splines or trajectory optimisation [11] and track it using feedback controllers such as PID or receding-horizon controller [12].

<sup>1</sup>PiP-X stands for Planning/replanning in Pipes in dynamic or initially unknown environments

## B. Motion replanning in dynamic environments

Earlier work on re-planning – D\* [13], LPA\* [14], D\*Lite [15] 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, requires additional pre-processing, or post-processing to achieve kinodynamic feasibility and/or controllability, and uses data structures that tend to scale exponentially with the dimensions of the system. Nevertheless, they provide a strong algorithmic foundation for developing quick, efficient re-planning algorithms that are useful in cases such as *geometric* path planning, where robot’s kinematics, dynamics, or control can be ignored.

Previous work on sampling based replanning focused on the feasibility problem – ERRT [16], DRRT [17] and multi-partite RRT [18], completely pruning the edges in collision and attempting to rejoin the disconnected branches to the rooted tree. RRT<sup>X</sup> [19] was the first asymptotically optimal sampling-based *re-planning* algorithm. It rewires the shortest-path subtree from goal to exclude tree nodes and edges that are in collision, similar to D\*Lite. The underlying search graph—built iteratively through sampling and *rewiring-cascade* step—ensures quick replanning and is well-suited for re-planning on-the-fly. Another technique is to resample configurations based on heuristics [20] and leverage the rewiring step from RRT\* to locally repair the solution branch around newly-sensed obstacles [21] [22].

Completeness and optimality guarantees have been achieved for geometric path re-planning, but incorporating robot dynamics without violating these guarantees remains an active area of research. Specifically, most of the optimal sampling-based and re-planning algorithms require solving the two-point boundary-value problem (TPBVP) which is generally difficult for non-holonomic robots—limiting their practical applicability. Some techniques have been proposed to solve without using two-point BVP [23], achieving near-optimality. However, most of the works consider simple or linearised dynamics without considering disturbances and unmodelled effects. Solving the intrinsic TPBVP for an arbitrary nonlinear system remains challenging and computationally intractable for online implementation.

## C. Funnel-based motion planning

Historically, robot-planning stacks have a hierarchical structure: the high-level path planner computes an open-loop trajectory and a low-level controller stabilises and tracks the trajectory. This decoupled approach is limited in practice because controller tracking errors and actuator saturations or uncertainties 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 through *feedback motion planning*, in which the motion planner explicitly considers the stabilising feedback controller to optimise planning for dynamical continuous systems. Mason et al. [24] introduced a metaphor, “*funnel*”, for locally stabilised and verified trajectories. An illustration of sequentially composed funnels reaching

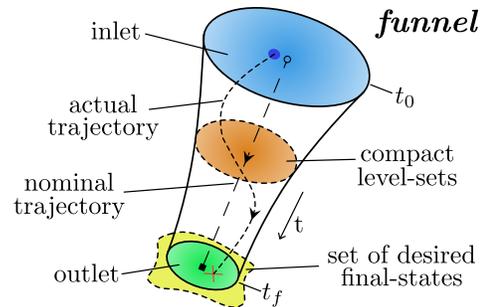


Fig. 2: A sample funnel – finite-time backward-reachable invariant set to a compact region of desired final states,  $\mathcal{X}_f$

a goal region, similar to Fig. 1-a, presented in [25] sparked the motivation to use such funnels for feedback motion planning.

Tedrake et al. [26] popularised the notion of LQR-trees—an algorithm that covers the state-space using a tree of time-varying trajectories, locally stabilised by an LQR controller and verified by Lyapunov level-set theory. [27] presents a detailed approach on how to compute these regions of finite-time invariance using Sum-of-Squares (SoS) programming and bilinear alternations. However, these methods are computationally intensive and are not suitable for scenarios in which the obstacles are not known a priori. Majumdar et al. [28] compute the funnels offline and use them to plan online for flying a glider through a dense setting. Similar work by the same research group leverages these concepts to develop control algorithms for UAV-perching [29], double pendulum [30], and a cart-pole [31]. Funnel-based motion planning for a robotic arm using adaptive feedback control is presented in [32].

## D. Other related work

In parallel to such Lyapunov analysis, the controls community has proposed reachability set-based trajectory design [33] [34]. FaSTrack [35] proposes an adversarial game-theoretical approach to generate worst-case tracking error bounds around trajectories using Hamilton-Jacobi reachability analysis. [36] uses contraction theory and convex optimisation to compute invariant-tubes around trajectories and plan using them. Other researchers also provide ways to compute stabilised and verified trajectories to be used with any motion-planner - using direct transcription [37], and funnel generator functions [38].

Our research extends previous work in *funnel-based motion planning* to *on-the-fly re-planning*. We propose a novel approach to represent funnels and compossibility using graphs, helping us leverage graph-based replanning algorithms to quickly rewire the search tree. Through this, we are able to recast the challenging problem of feedback motion replanning for non-trivial systems into a geometric one, making it tractable for online planning. Specifically, our work focuses on generating kinodynamically feasible motion plans/replans for higher-order nonlinear systems using backward reachable sets, implicitly solving the two-point boundary value problem.

### III. PRELIMINARIES

Our approach combines concepts from invariant set analysis for dynamical systems, design of a library of verified motion primitives, and graph-based replanning techniques. This section briefly provides the requisite background while introducing the notation that would be used in the rest of the paper.

#### A. Invariant Set Theory

The notion of region of attraction of asymptotically stable fixed points is extended to certifying time-varying trajectories. Such regions of finite-time invariance around a trajectory are referred to as “*funnels*” [27]. Considering a closed-loop nonlinear system,

$$\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t)) \quad (1)$$

Where state,  $\mathbf{x} \in \mathbb{R}^n$  and  $\mathbf{f}$  is Lipschitz continuous in  $\mathbf{x}$  and piecewise continuous in  $t$ . This guarantees global existence and uniqueness of a solution [39]. Considering a finite time interval,  $[t_0, t_f]$ , a *funnel* is formally defined as,

**Definition 1. Funnel** - A set,  $\mathcal{F} \subseteq [t_0, t_f] \times \mathbb{R}^n$ , such that for each  $(\tau, \mathbf{x}_\tau) \in \mathcal{F}$ ,  $\tau \in [t_0, t_f]$ , the solution to Eq. (1),  $\mathbf{x}(t)$ , with initial condition  $\mathbf{x}(\tau) = \mathbf{x}_\tau$ , lies entirely within  $\mathcal{F}$  till final time, i.e.  $(t, \mathbf{x}(t)) \in \mathcal{F} \forall t \in [\tau, t_f]$

Intuitively, if the closed-loop system starts within the funnel, then the system states evolving due to Eq. (1) remain within the funnel at all time instances until the final time. We leverage tools from Lyapunov theory to compute bounded, inner-approximations of the funnel. The compact level-sets of a Lyapunov function,  $V(t, \mathbf{x})$ , satisfy the conditions of positive invariance [40],

$$\mathcal{B}(t) = \{\mathbf{x} \mid 0 \leq V(t, \mathbf{x}) \leq \rho(t)\} \quad (2)$$

As noted in [27], 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  $\rho(t)$  such as,

$$\begin{aligned} V(t, \mathbf{x}) = \rho(t) &\implies \dot{V}(t, \mathbf{x}) \leq \dot{\rho}(t) \\ \dot{V}(t, \mathbf{x}) &= \frac{\partial V(t, \mathbf{x})}{\partial t} + \frac{\partial V(t, \mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(t, \mathbf{x}(t)) \end{aligned} \quad (3)$$

In other words, with respect to time, the Lyapunov function at the boundary,  $\partial\mathcal{B}(t)$  should decrease faster than the level set,  $\rho(t)$  for invariance. Then the set defined as follows, satisfying Eq. (3) is a *funnel* [29],

$$\mathcal{F} = \{(t, \mathcal{B}(t)) \mid t \in [t_0, t_f]\} \quad (4)$$

In our case, we are interested in computing backward-reachable sets about a finite-time trajectory, to a compact region of state-space. Given a bounded space of desired final-states,  $\mathcal{X}_f \subseteq \mathbb{R}^n$ , we consider funnels that end within the region, i.e.  $(t_f, \mathbf{x}(t_f)) \in \mathcal{F} \implies \mathbf{x}(t_f) \in \mathcal{X}_f$ , or alternatively,  $\mathcal{B}(t_f) \subseteq \mathcal{X}_f$ . A sample funnel with its parts labelled can be found in Fig. 2.

We wish to maximise the volume of the funnel that flows into the sub-goal region,  $\mathcal{X}_f$  using tools from Lyapunov

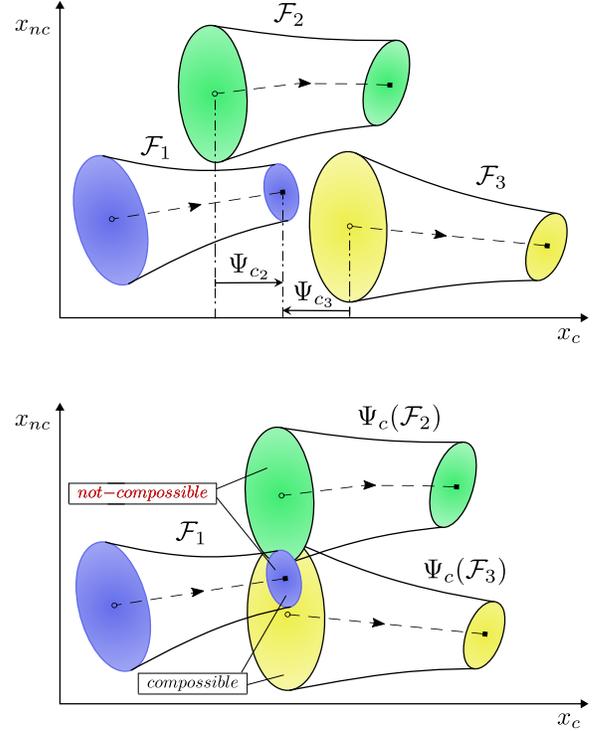


Fig. 3: Compossibility of funnels from the library illustrating shifts,  $\Psi_c(\cdot)$  along cyclic coordinates (invariant dynamics).  $(\mathcal{F}_1, \mathcal{F}_3)$  is *motion-plan compossible*, whereas  $(\mathcal{F}_1, \mathcal{F}_2)$  is not – i.e. outlet of  $\mathcal{F}_1$  is completely contained within the inlet of  $\mathcal{F}_3$  after an appropriate shift operation,  $\Psi_{c_3}$

analysis and convex optimisation. The funnel-volume is as defined in [27]. Section V-D talks about how to compute such *maximum volume* funnels for systems with piecewise polynomial dynamics, using *Sum-of-Squares* (SoS) relaxation to solve for a class of quadratic Lyapunov functions.

#### B. Verified Trajectory Libraries

The idea of saving a pre-computed library of motion primitives for online planning has seen considerable research in the past. An optimisation-based approach to design the library, abstracting the information and sequence of trajectories is provided in [41]. Maneuver Automaton [42] discusses the relevant properties of trajectory libraries required for sequencing, providing a theoretical foundation.

The condition for sequencing trajectories presented by [25] can be extended to funnels by analysing the compact invariant sets satisfying Eq. (3),  $\mathcal{B}(t)$  at the final and initial time. An ordered pair of funnels,  $(\mathcal{F}_i, \mathcal{F}_j)$  is *sequentially compossible* if  $\mathcal{B}_i(t_{f_i}) \subseteq \mathcal{B}_j(t_{0_j})$ . However, this is often a strict condition, and is not necessary for composing motion plans. In order to analyse the sequencibility of trajectories for motion planning, we decompose the state vector into cyclic and non-cyclic states,  $\mathbf{x} = [\mathbf{x}_c^T \ \mathbf{x}_{nc}^T]^T$ . Cyclic states are defined as the coordinates to which the open-loop dynamics of a Lagrangian

system,  $\dot{\mathbf{x}} = \mathbf{f}'(t, \mathbf{x}, \mathbf{u})$  are invariant, or alternatively, the dynamics depend only on the non-cyclic states,

$$\dot{\mathbf{x}}(t) = \mathbf{f}'(t, \mathbf{x}_{nc}(t), \mathbf{u}(t)) \quad (5)$$

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 [43]. One can shift the funnel along the cyclic coordinates so as to contain the outlet of the previous funnel, as illustrated in Fig. 3.

**Definition 2.** A funnel-pair,  $(\mathcal{F}_i, \mathcal{F}_j)$  is said to be *motion-plan compossible* if and only if

$$\mathcal{P}_{nc}^{\mathcal{S}}(\mathcal{B}_i(t_{f_i})) \subseteq \mathcal{P}_{nc}^{\mathcal{S}}(\mathcal{B}_j(t_{0_j})) \quad (6)$$

where  $\mathcal{P}_{nc}^{\mathcal{S}}(\cdot)$  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 vector space of cyclic coordinates by shifting the trajectories appropriately.

For example, during online planning in a UAV, it suffices to check whether the linear and angular velocities at the start of a trajectory match with the current velocities. The position and attitude (cyclic coordinates) of the funnel can be shifted to the current pose of the UAV. The various notions of compossibility is discussed in detail in [28], providing methods to check the condition in Eq. (6) using semi-definite programming.

### C. Discrete graph replanning

Traditionally, path planners make extensive use of graphs to represent the configuration space  $\mathcal{C}$ , and deploy various search techniques to search for a feasible or an optimal path through the graph. So, graphs serve a dual purpose of modelling the topology and traversability of  $\mathcal{C}$ -space, and searching for a solution path through it.

Graphs, in general, are a mathematical tool to model relations between a group of objects. A graph is defined by a set of vertices and edges, formally denoted as  $G = (V, E)$ , where  $V$  is the set of nodes and set of edges,  $E \subseteq \{(v, w) \mid v, w \in V, v \neq w\}$ . In typical graph-based and sampling-based motion planning algorithms, 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, Eq. (6).

Most search algorithms build a connected acyclic subgraph (tree) within the graph, such that each vertex has one, and only one, parent. A path from given start to the goal is readily found by backtracking parent pointers starting from the goal or start depending on the search-direction. Discrete-planners like Dijkstra or A\* find an optimal path with respect to a user-defined cost function. The edge costs satisfy the property of a distance metric,

**Definition 3. Distance metric** in a set,  $\mathcal{V}$ , is a function  $d_M : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}^+$ , to the set of non-negative real numbers. For  $v_1, v_2, v_3 \in \mathcal{V}$ , the distance metric satisfies the following properties,

- 1) Non-negativity:  $d_M(v_1, v_2) > 0 \forall v_1 \neq v_2$
- 2) Identity of indiscernibles:  $d_M(v_1, v_2) = 0$  iff  $v_1 = v_2$
- 3) Triangle inequality:  $d_M(v_1, v_2) + d_M(v_2, v_3) \geq d_M(v_1, v_3)$

Given an edge,  $e = (v, w) \in E$ , a suitable distance metric  $c(v, w)$  represents the cost to move from vertex  $v$  to vertex  $w$ . D\* and LPA\* are discrete graph-based *replanning* algorithms that repair the shortest path-to-goal as edge-costs change. Such incremental search techniques reuse all valid current information to improve the solution path in the next iteration, resulting in a faster replanning speed than algorithms that replan from scratch.

D\*Lite algorithm continually and efficiently repairs the minimum-cost path from robot to goal, despite changing edge-weights while the robot traverses the path. Our algorithm uses a similar idea to maintain a shortest-path reverse-tree of *funnels* rooted at the goal region. 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, through the graph. In our work, the cost of a *funnel-edge* is given by the length of the nominal trajectory within the funnel. Additionally, the algorithm computes an *lmc* value (one-step *lookahead minimum cost*) for all nodes, defined as,

$$lmc(v) = \min_{v' \in N^+(v)} \{c(v, v') + g(v')\} \quad (7)$$

Where  $N^+(v)$  is the set of out-neighbors of vertex,  $v$ . For  $e = (v, w) \in E$ ,  $w$  and  $e$  are said to be the out-neighbor and out-edge of  $v$ , respectively. Similarly,  $v$  and  $e$  are referred to as the in-neighbor and in-edge of  $w$ , respectively. Based on the two values,  $g$  and  $lmc$ , we determine whether changes have occurred in the shortest path to the goal:  $lmc$  is better informed because it gets updated because of changes in out-neighbors' cost-to-goal. The key idea of D\*Lite can be explained as,

- 1)  $g(v) = lmc(v) \implies v$  is *consistent*  $\rightarrow$  no changes to shortest path from  $v$  to goal
- 2)  $g(v) < lmc(v) \implies v$  is *under-consistent*  $\rightarrow$  cost 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 its in-neighbors

In addition to the incremental search, the algorithm uses *heuristics* to focus the reverse-search to the robot-node.

**Definition 4. Heuristic value**,  $h : \mathcal{V} \rightarrow \mathbb{R}^+$  is a non-negative estimate of the cost from start to a node satisfying

- 1)  $h(v) \geq 0 \forall v \neq v_{start}, h(v_{start}) = 0$
- 2) Triangle inequality:  $h(v) \leq h(v') + c(v, v') \forall v, v' \in \mathcal{V}$
- 3) Admissibility:  $h(v) \leq h^*(v) \forall v \in \mathcal{V}$ , where  $h^*(v)$  is the optimal cost-from-start value

We do not repair all the nodes after every edge-cost change, instead only repair *promising nodes* that have the ‘‘potential’’

to lie in the robot's shortest path to goal, determined using  $g$ ,  $lmc$  and  $h$  values as in Eq. (8). A priority queue is utilised to maintain an order in which nodes need to be repaired. The inconsistent nodes, i.e.  $g(v) \neq lmc(v)$ , are pushed into the minimum priority queue based on the following key,

$$key(v) = [\min\{g(v), lmc(v)\} + h_{start}(v), \min\{g(v), lmc(v)\}] \quad (8)$$

In our work, graph-vertices represent regions in state-space— inlet/outlet regions, and edges represent funnels “flowing” from inlets to outlets. We represent both, traversability and compossibility, using a directed graph and implement incremental search methods to calculate the shortest path from robot to the goal, and recalculate it as the environment changes. This ensures there exists a safe, controllable trajectory starting from the initial state in an *inlet-node* to the final state in an *outlet-node*.

#### IV. PROBLEM FORMULATION

For a Lagrangian robot system with state,  $\mathbf{x} = (\mathbf{q}, \mathbf{v}) \in \mathcal{S} \subseteq \mathbb{R}^{2d}$ , where  $\mathbf{q} \in \mathcal{C} \subseteq \mathbb{R}^d$  is position,  $\mathbf{v} \in \mathbb{R}^d$  represents the velocities and  $d \in \mathbb{N}$  is the dimension of the configuration space,  $\mathcal{C}$ , the dynamics are given by,

$$\dot{\mathbf{q}} = \mathbf{v} \quad \dot{\mathbf{v}} = \mathbf{f}(\mathbf{q}, \mathbf{v}, \mathbf{u}) \quad (9)$$

where  $\mathbf{u} \in \mathcal{U} \subseteq \mathbb{R}^m$  is the control-input to the robot system, and  $\mathbf{f}$  representing the system dynamics is locally Lipschitz continuous.

**Assumption 1.** *For a compact set of desired configurations,  $\mathcal{X}_{des} \subseteq \mathcal{C}$ , there exists a state-feedback control policy,  $\mathbf{u} : [t_0, t_f] \rightarrow \mathcal{U}$ , which when input to the system (9), starting at  $\mathbf{x}(t_0) = \mathbf{x}_0$ , ensures  $\mathbf{q}(t_f) \in \mathcal{X}_{des}$ , for some finite time  $t_f \geq 0$ .*

We formally quantify the tracking or stabilising performance of this assumed controller by computing inner-approximations of backward-reachable invariant sets around the nominal trajectory, as described in Section V-D.

Consider a robot that operates in workspace,  $\mathcal{W} \subseteq \mathcal{C} \subseteq \mathbb{R}^d$ , with finite number of obstacles having locally Lipschitz continuous boundaries, occupying a subspace,  $\mathcal{O} \subset \mathcal{W}$ . Correspondingly, 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 5. 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 (2)-(4), such that  $v \in \mathcal{P}_{\mathcal{C}}^S(\mathcal{B}(t_0))$  and  $\mathcal{P}_{\mathcal{C}}^S(\mathcal{B}(t_f)) \subseteq \mathcal{X}_w$ . A funnel-edge is said to be *valid*, if and only if  $\mathcal{P}_{\mathcal{W}}^C(\phi(v, w)) \cap \mathcal{O} = \emptyset$ . The cost of the funnel-edge is given by the length of the nominal trajectory,  $\mathbf{x}_0(t)$  projected down to  $\mathcal{C}$ -space,  $\mathbf{q}_0(t)$

$$c_{\phi}(v, w) = \int_{t_0}^{t_f} ds(t) \quad (10)$$

where  $ds = \|\mathbf{d}\mathbf{q}_0\|$ ,  $\mathbf{q}_0$  satisfies (9), and  $\|\cdot\|$  represents the Euclidean norm.  $\mathcal{P}_B^A(\cdot)$  is the projection operator from a space  $A$  to a lower dimensional subspace,  $B$ .

It is worth mentioning that the defined cost function (10) satisfies the properties in (3), and hence is a distance metric. Since a valid funnel-edge does not intersect with the obstacle-set, the trajectory contained within it due to set-invariance, will not be in collision with obstacles. Thus,  $\mathbf{q}(t) \in \phi_{valid}(\cdot) \Leftrightarrow \mathbf{q}(t) \in \mathcal{C}_{free}$  for  $t_0 \leq t \leq t_f$ .

**Definition 6. 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.  $\pi(q_1, \mathcal{X}_2) = \{\phi_1, \phi_2, \dots, \phi_n\}$ , such that  $q_1 \in \mathcal{P}_{\mathcal{C}}^S(\mathcal{B}_{\phi_1}(t_{0_1}))$  and  $\mathcal{P}_{\mathcal{C}}^S(\mathcal{B}_{\phi_n}(t_{f_n})) \subseteq \mathcal{X}_2$ . The cost of the funnel path is defined,

$$c_{\pi}(q_1, \mathcal{X}_2) = \sum_{i=1}^n c_{\phi_i} \quad (11)$$

**Problem 1. Online motion planning** - Given  $\mathcal{C}_{free}$ , obstacle space,  $\mathcal{O}(t)$ , and a goal region,  $\mathcal{X}_{goal} \subseteq \mathcal{C}_{free}$  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,  $u : (\mathbf{x}, \mathbf{x}_0, t) \rightarrow \mathcal{U}$  and keep updating  $\pi^*$  until  $q_{robot}(t) \in \mathcal{X}_{goal}$

$$\pi^*(q_{robot}(t), \mathcal{X}_{goal}) = \arg \min_{\pi(q_{robot}(t), \mathcal{X}_{goal})} c_{\pi}(q_{robot}(t), \mathcal{X}_{goal})$$

A dynamic environment has its obstacle set changing randomly with time and/or with robot position,  $\Delta\mathcal{O}(t) = s(t, q_{robot}(t))$ . An environment can be modelled as static if  $h$  is known *a priori* or can be deterministically computed. A trivial case of static environment is  $\Delta\mathcal{O}(t) \equiv \emptyset$ . We consider replanning in dynamic environments, where  $\Delta\mathcal{O}(t)$  is neither known *a priori* nor possible to predict.

**Problem 2. Motion replanning** - Assuming that a robot has the ability to sense obstacle changes,  $\Delta\mathcal{O}(t)$ , 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 in real-time. The funnel-path is a sequence of maneuvers with formal guarantees of invariance, associated with state-feedback tracking control policies. The motion plan/replan with invariant sets and control policies ensures that the robot trajectory lies within the funnel path, avoiding dynamic obstacles and ultimately reaching the goal region.

#### V. APPROACH

This section details the various components of our method: the pre-processing stage of computing backward-reachable invariant sets and designing the funnel library, and the online phase of the motion re-planning algorithm, PiP-X.

##### A. Outline

This subsection contains a brief high-level outline of our entire approach, which involves: invariant set analysis, designing

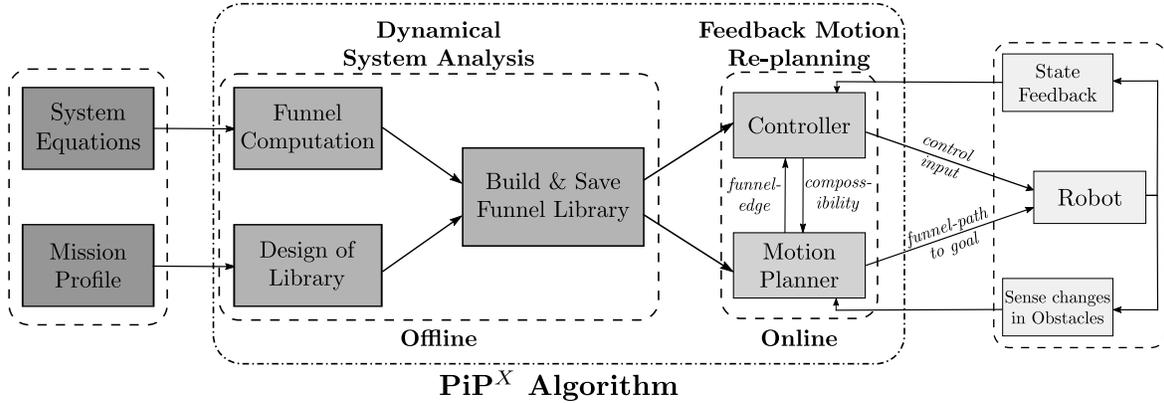


Fig. 4: Overview of PiP-X algorithm: consists of an offline stage of dynamical system analysis, and an online phase of sampling-based graph construction and incremental re-planning in dynamic environments

a funnel library, and the online feedback motion re-planning using an augmented directed graph. Each part is discussed in greater detail in its own subsection. As illustrated in Fig. 4, our method has an offline stage of nonlinear system analysis using Lyapunov theory, and an online phase of sampling-based motion re-planning using incremental graph-search. Given the mission profile, we design the funnel library as described in Section V-E. Design considerations include description of cyclic/non-cyclic coordinates of the state-space for *funnel composability*,  $\mathcal{C}$ -space topology to ensure *probabilistic coverage*, and desired resolution of motion-plan.

We compute backward-reachable invariant sets, detailed in Section V-D, using Lyapunov theory. The system’s equations of motion along with the state-feedback control law are approximated to polynomial dynamics about the nominal trajectory using Taylor-series expansion of order greater than 1. We find that such a bounded polynomial approximation offers a conservative estimate, i.e. it always under-estimates the inner-approximations of backward-reachable sets—a condition sufficient for our re-planning algorithm. Given an initial state, a compact set of desired final-states and 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 graph representing a network (or roadmap) of *funnel-connectivity* is incrementally built using sampling methods (RRG), and motion plans/replans are computed through it. The graph embedded in the  $\mathcal{C}$ -space is constructed based on the aforementioned system analysis in the higher dimensional state-space. The graph-edges represent funnels, and vertices represent inlet or outlet regions in state-space, both projected down to the  $\mathcal{C}$ -space. We analyse the *motion-plan composability* (Definition 2) of funnel-pairs and additionally include that information in the form of an *augmented search graph* (see Fig. 5 and Section V-B). With this graph we compute a shortest-path subtree of funnels, rooted at the goal region using an approach similar to one introduced in D\*Lite [15].

Using graph-based replanning methods, motion plans are

quickly recomputed 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 goal region and the corresponding sequence of control-inputs are input to the robot, with state-observer and obstacle-sensor closing the feedback-loop. Section V-C describes our feedback motion planning/replanning algorithm in-depth.

From Fig. 6, it is worth noticing that the funnels computed based on Lyapunov theory offer a sufficient but not a necessary condition for invariance. Trajectories starting inside the funnel will remain in the funnel for the entire finite time-horizon. However, trajectories starting outside the funnel may or may not terminate within the defined goal region. Nevertheless, this analysis provides formal guarantees about robustness to set of initial conditions and system perturbations, pertinent in sampling-based motion planning/replanning of kinodynamic systems.

### B. Notes on augmented search-graph

We represent the network of funnels using a directed graph *augmented* with the additional information of funnel-composability (see Fig. 5). Such an augmented graph representation enables the use of incremental graph-replanning techniques to quickly rewire funnel-paths (Definition 6) to the goal region. 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,  $E = E_m \cup E_c$ . The graph-vertex is a tuple consisting of the configuration and the funnel-edge,  $v = \begin{bmatrix} q & f \end{bmatrix}$ . The *composite nodes* exhibit certain relations amongst the node-set  $V$ , summarised as,

- $\begin{bmatrix} q & f_1 \end{bmatrix} \& \begin{bmatrix} q & f_2 \end{bmatrix}$  – Sharing the same first element (configuration) implies there will be a directed zero-cost *continuity edge*,  $e_c$  between the two composite nodes, if and only if  $(f_1, f_2)$  is *motion-plan composable*.
- $\begin{bmatrix} q_1 & f \end{bmatrix} \& \begin{bmatrix} q_2 & f \end{bmatrix}$  – Represents the case of two vertices sharing a funnel-edge,  $f$ . Hence, by default

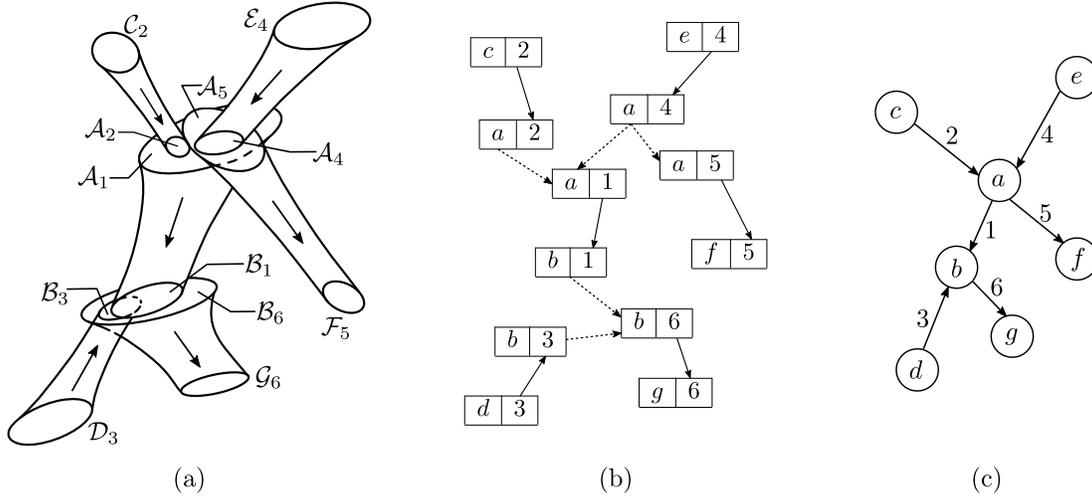


Fig. 5: (a) Search funnels existing in the  $\mathbb{R}^+ \times \mathbb{R}^n$  (b) *Augmented search graph* representing planning in  $\mathcal{C}$ -space with compossibility information. Solid edges are finite-cost *motion-edges* representing traversability, dashed-edges are zero-cost *continuity-edges* encoding compossibility information – whether trajectories “flow” into the subsequent funnel (c) Conventional search graph used by typical tree/graph-based motion-planners

a *motion-edge*,  $e_m$  exists, having a cost as defined in Eq. (10)

- $[q_1 | f_1]$  &  $[q_2 | f_2]$  – These nodes do not have a common entry. In such cases, there will be *no* edge between the 2 composite-nodes.

It is worth mentioning that (dashed) continuity-edges are zero-cost and have no bearing on the cost of the solution path to goal or the optimality guaranteed by the graph-search algorithm. As illustrated in Fig. 5-c, when nodes with the same configuration (first element of the tuple) are grouped together into one *super-node* result in a search-graph with just (solid) motion-edges, usually used by typical path-planners.

As observed in Fig. 5-b, there are two types of graph vertices – *inlet-nodes* and *outlet-nodes*,  $V = \{V_I, V_O\}$ . Inlet-nodes like  $[a | 1]$  have one, and only one, solid out-edge – might have one or more dashed in-edges. Outlet nodes have one, and only one, solid in-edge, and one or more dashed out-edges, e.g.  $[a | 4]$ . For any node, the list of in-edges will have either dashed edges or one solid edge. The same is true for out-edges.

With these properties and observations, the authors would like to point out that the *augmented graph* is indeed *bipartite* – disjoint sets of inlet-nodes and outlet-nodes. Motion-edges go from  $V_I$  to  $V_O$ , and continuity-edges from  $V_O$  to  $V_I$ . Hence, any path from a configuration to the goal will have alternating (solid) motion-edges and (dashed) continuity-edges.

### C. Online motion planning-replanning algorithm – PiP-X

A reverse-search graph is more effective in scenarios that require online replanning, such as a robot navigating through an unknown environment perceiving obstacles within a limited sensor radius. It is efficient because it suffices to alter the motion plans only locally near the robot location, saving us the cost of rewiring the bulk of the search-tree. We incrementally build the *funnel-graph* and compute the optimal *funnel-path*

through the graph using the routine – `plan()` (Algorithm 1). The pre-planning process on a higher level is as follows,

- 1) Sample a configuration  $q_{rand}$  (line 1) and extend an  $\epsilon$ -distance (line 2) from the nearest node in the existing search graph to determine new configuration  $q_{new}$
- 2) Determine the set of nearest neighbors (line 4) in the shrinking  $r$ -ball [3],  $r = \min\{r_0(\log|V|/|V|)^{1/d}, \epsilon\}$ , where  $|V|$  is cardinality of the vertex set,  $d$  is the dimensionality of  $\mathcal{C}$  and  $r_0$  is a user-specified parameter
- 3) From the funnel library  $\mathcal{L}$ , choose the trajectory that would steer (Algorithm 3) the robot from  $n$  to a  $\delta$ -ball near  $q_{new}$  as well as the return trajectory from  $q_{new}$  to a  $\delta$ -ball near  $n$ , for all  $n$  in the set of nearest neighbors
- 4) Once we get the *funnel-edges*,  $\phi$ , check for “overlap” with obstacles, and compossibility among the funnels. Represent the projections of inlets,  $\mathcal{X}_i$  and outlets,  $\mathcal{X}_o$  as nodes,  $\phi$  as a *motion-edge*, and zero-cost *continuity-edges* signifying compossibility – hence is the search-graph iteratively constructed (Algorithm 4)
- 5) An incremental search (Algorithm 6) on the constructed sampling-based graph keeps updating the shortest-path subtree rooted at the goal

---

#### Algorithm 1 $(\mathcal{F}, \mathcal{G}) \leftarrow \text{plan}()$

---

- 1:  $q_{rand} \leftarrow \text{sampleFree}()$
  - 2:  $q_{new} \leftarrow \text{extend}(\mathcal{G}, q_{rand}, \epsilon)$  ▷ geodesic-distance
  - 3:  $r \leftarrow \text{rBall}()$
  - 4:  $\mathcal{N} \leftarrow \text{findNearestNeighbors}(q_{new}, r, \mathcal{G})$
  - 5:  $(\mathcal{F}, \mathcal{G}) \leftarrow \text{constructSearchGraph}(q_{new}, \mathcal{N})$
  - 6:  $\text{computeShortestPathTree}()$
  - 7: **return**  $(\mathcal{F}, \mathcal{G})$
- 

After each update (line 6), all the consistent nodes in the graph know their best parent according to Eq. (12), enabling the planner to backtrack the solution-path using parent point-

---

**Algorithm 2** PiP-X
 

---

**Input:**  $q_{start}, \mathcal{X}_{goal}, \mathcal{C}_{free}, \mathcal{O}, \mathcal{L} \triangleright$  Start, Goal region, Free-space, obstacle-space, Funnel library

**Output:**  $\mathcal{G}, \mathcal{F} \triangleright$  search-Graph and search-Funnel

- 1: **Parameters:**  $\epsilon, r_0, T_P, I_M \triangleright$  extend distance,  $r$ -ball, pre-planning time, *idleness* limit
- 2: **Initialisation:**  $t \leftarrow 0, startFound \leftarrow 0,$
- 3:  $\mathcal{G}.add(\mathcal{X}_{goal}), \mathcal{F} \leftarrow \emptyset$
- 4: **while**  $t < T_P \vee \neg startFound$  **do**
- 5:  $(\mathcal{F}, \mathcal{G}) \leftarrow \text{plan}()$
- 6: **if**  $\text{inFunnel}(q_{start}, \mathcal{F})$  **then**
- 7:  $startFound \leftarrow 1$
- 8: **end if**
- 9: **end while**  $\triangleright$  end of **Pre-planning** phase
- 10:  $j \leftarrow 0, q_{robot} \leftarrow q_{start}, q_{prev} \leftarrow q_{start}$
- 11: **while**  $j < I_M \wedge q_{robot} \notin \mathcal{X}_{goal}$  **do**  $\triangleright$  **Online** phase
- 12: **at sensingFrequency** **do**  $\triangleright$  Sensing obstacle-changes
- 13:  $\Delta\mathcal{O} \leftarrow \text{senseObstacles}()$
- 14:  $\text{modifyEdgeCosts}(\Delta\mathcal{O})$
- 15: **end**
- 16:  $(\mathcal{F}, \mathcal{G}) \leftarrow \text{plan}()$   $\triangleright$  *Repairing* the motion-plan
- 17: **at robotMotionFrequency** **do**  $\triangleright$  Robot movement
- 18:  $q_{robot} \leftarrow \text{robotMove}(q_{robot}, q_{goal})$
- 19: **if**  $g(q_{robot}) \neq \infty$  **then**  $\triangleright$  a *funnel-path* exists
- 20:  $k_m \leftarrow k_m + \text{computeHeuristic}(q_{prev}, q_{robot})$
- 21:  $q_{prev} \leftarrow q_{robot}; j \leftarrow 0 \triangleright$  reset *idleness* count
- 22: **else**
- 23:  $q_{robot} \leftarrow q_{prev} \triangleright$  stay at current location
- 24:  $j \leftarrow j + 1 \triangleright$  update *idleness* count
- 25: **end if**
- 26: **end**
- 27: **if**  $q_{robot} \in \mathcal{X}_{goal}$  **then**
- 28: **return** SUCCESS  $\triangleright$  Algorithm success
- 29: **end if**
- 30: **end while**
- 31: **return** NULL  $\triangleright$  Algorithm failure

---

ers. The search is focused towards the robot location using an admissible heuristic,  $h(v)$  as defined in (4), thereby enabling quick rewiring of the optimal path whenever the heuristic provides useful information.

The pre-planning phase in Algorithm 2 (lines 4–9) continues until the start configuration lies within one of the funnel-inlets and the search-graph is dense enough to have covered a sufficient volume of the  $\mathcal{C}$ -space. A solution funnel path exists if the robot configuration lies within one of the inlet-nodes and has a finite cost-to-goal value. Consequently, we have a sequence of closed-loop control policies to transition from a region of state-space to another, ultimately terminating at the goal region. The entire trajectory is guaranteed to lie within the solution funnel branch by the virtue of set-invariance, provided the actual model sufficiently resembles the nominal model.

The various routines of our *online re-planner*, PiP-X (Algorithm 2) providing low-level implementation details, are explained as follows: We first specify the algorithm parameters and inputs, and initialise the required data structures – graph, kdTree, priority queue (lines 1–3). The planning parameters

are minimum path-resolution,  $\epsilon$ , shrinking  $r$ -ball parameters,  $r_0$  and  $d$ , pre-planning time,  $T_P$  and idleness limit,  $I_M$ . The inputs to the algorithm are start configuration,  $q_{start}$ , goal region,  $\mathcal{X}_{goal}$ , the pre-computed funnel library,  $\mathcal{L}$ , and the initial environment – characterised through  $\mathcal{C}_{free}$  and list of obstacles known *a priori*,  $\mathcal{O}$ . The obstacle-space will be updated when any changes,  $\Delta\mathcal{O}(t)$  are discovered on-the-fly.

1) **Sampling configurations:** `sampleFree()` – The configurations  $q_{rand}$  are independent and identically (i.i.d.) drawn from the free-space,  $\mathcal{C}_{free}$  at random. When the robot starts moving and senses obstacles, the sampling is directed towards the sensed region where changes are certain to have occurred. This helps to rewire the parent-edges near the robot, ensuring the robot has a choice of safe alternate plans around the new-found obstacles. However, configurations are continued to be drawn uniformly random from  $\mathcal{C}_{free}$  at regular frequency even after the robot starts moving, for probabilistic coverage and completeness.

2) **Graph extension:** The  $\mathcal{C}$ -space is explored using the `extend( $\mathcal{G}, q_{rand}, \epsilon$ )` routine. It determines the nearest configuration in the existing search graph, based on geodesic distance, and aims to extend to  $q_{rand}$  by at most an  $\epsilon$ -distance to obtain the new configuration,  $q_{new}$ . If this configuration is already in the search-funnel,  $\mathcal{F}$ , we discard it and continue with the next sampling, because we are guaranteed to find a set of maneuvers which would drive the robot-system from this configuration to the goal region.

`findNearestNeighbors( $q_{new}, r, \mathcal{G}$ )` determines the neighbors within a  $r$ -ball around the new configuration,  $q_{new}$ . It is implemented through a k-D tree built using configurations. The radius of the ball decreases at a “shrinking rate” derived using percolation theory [44].

---

**Algorithm 3**  $\mathcal{F} \leftarrow \text{steer}(q_1, q_2)$ 


---

- 1:  $\mathcal{F}' \leftarrow \text{findFunnel}(q_1, q_2, \mathcal{L})$
- 2:  $\mathcal{F} \leftarrow \text{shiftFunnel}(q_2, \mathcal{F}')$   $\triangleright$  shifts & truncates the funnel
- 3: **if**  $q_1 \notin \mathcal{F}.ellipsoid(\text{start}) \vee \neg \text{collisionFree}(\mathcal{F}, \mathcal{O})$  **then**
- 4: **return**  $\emptyset$
- 5: **end if**
- 6: **return**  $\mathcal{F}$

---

3) **Steering (Algorithm 3):** From the motion-primitives library, `findFunnel()` determines the appropriate funnel that closely drives the system from configuration,  $q_1$  to  $q_2$ . We use `shiftFunnel()` subroutine to shift the funnel along the cyclic coordinates and time, using appropriate shift-operators  $\Psi_c(\cdot)$  and  $\Psi_t(\cdot)$ , respectively. The fact that the funnel is a backward-reachable set, enables us to truncate the funnel at any time,  $t_f \in [0, T)$ .

The funnel is projected down to the workspace for checking any overlaps with the obstacle-set,  $\mathcal{O}$ . If the funnel is *in-collision* or the target-configuration does not lie in the inlet of the funnel projected down to  $\mathcal{C}$  (line 3), the subroutine `steer()` returns a null set. Otherwise, we return the funnel,  $\mathcal{F}$  along with its cost, computed as the length of the nominal trajectory within the funnel according to Eq. (10).

4) **Constructing the search funnel-graph (Algorithm 4):** We attempt to construct funnels (lines 2–3) between the new

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


---

```

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

```

---

configuration,  $q_{new}$  and all of the nodes in the neighbor-set,  $n \in \mathcal{N}$ . The valid funnels flowing into a  $\delta_o$ -ball around  $q_{new}$  are referred to as its *inFunnels*,  $\mathcal{F}_q^- \equiv \mathcal{F}_n^+$  and funnels flowing out of a  $\delta_i$ -ball around  $q_{new}$  as its *outFunnels*,  $\mathcal{F}_q^+ \equiv \mathcal{F}_n^-$ ,  $\forall n \in \mathcal{N}$ .

$\text{getNode}(\mathcal{F})$  in lines 5 and 16 determines the nodes,  $\mathcal{X}_i$  and  $\mathcal{X}_o$ , corresponding to the inlet and outlet of the funnel projected down to  $\mathcal{C}$ -space. These nodes 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*,  $E_m$  (lines 12 and 23) The new constructed funnels are added to the search-funnel,  $\mathcal{F}$  (line 25).

All the pairs of inFunnels and outFunnels at  $q_{new}$  are checked for sequencibility (lines 6–11 and lines 17–22) using  $\text{compossible}()$  (Algorithm 5). If  $\text{compossible}()$ , a zero-cost directed edge from outlet-node to inlet-node is added to the set of *continuity-edges*,  $E_c$  (lines 9 and 20). Additionally, the existing inFunnels and outFunnels at neighbor-nodes,  $n$  are checked for compossibility with the new constructed funnels

to/from  $n$  (lines 27–35). If compossible, zero-cost continuity-edges between outlet-nodes and inlet-nodes at  $n$  are added to  $E_c$  (line 32).

Invoking  $\text{updateVertex}()$  (Algorithm 7) in line 13 ensures propagation of cost-changes and possible rewiring of the shortest-path subtree 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 overconsistent), they are pushed into the priority queue,  $\mathcal{Q}$  with key computed as in Eq. (8), and will be repaired if they have the “potential” to lie in the solution path to goal.

**Algorithm 5**  $\text{compossible}(\mathcal{F}_1, \mathcal{F}_2)$ 


---

```

1:  $\mathcal{E}_i \leftarrow \mathcal{F}_2(\text{start}).\text{ellipsoid}$   $\triangleright$  inlet of  $\mathcal{F}_2$ 
2:  $\mathcal{E}_o \leftarrow \mathcal{F}_1(\text{end}).\text{ellipsoid}$   $\triangleright$  outlet of  $\mathcal{F}_1$ 
3: if  $\mathcal{E}_i \supseteq \mathcal{E}_o$  then  $\triangleright$  inlet completely contains outlet
4:   return TRUE
5: end if
6: return FALSE

```

---

5) *Funnel-related subroutines*: The re-planning algorithm makes use of minor subroutines specific to *funnels*.  $\text{inFunnel}(q, \mathcal{F})$  returns a boolean value, based on whether the configuration  $q$  lies in any of the inlets of the funnel-edges in  $\mathcal{F}$ . This is useful while checking whether a path exists from start to goal region (Algorithm 2 [line 6]) and during sampling too. The check is performed based on Eq. (14), with ellipsoidal inlet regions of funnels projected down to  $\mathcal{C}$ .

$\text{compossible}()$  (Algorithm 5) checks whether the funnel-pair is motion-plan compossible as defined in (2). The ellipsoid-in-ellipsoid check in line 3 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  $\mathcal{E}_i$  using (14).

**Algorithm 6**  $\text{computeShortestPathTree}()$ 


---

```

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

```

---

6) *Building the shortest-path subtree* (Algorithm 6): Given the search graph, a tree rooted at the goal with minimum cost-to-goal is calculated using techniques outlined in Section III-C. Invoking  $\text{computeShortestPathTree}()$  ensures that the robot/start node becomes consistent and also all the

nodes with lesser key value than the start node (line 2). So, in effect the shortest path from each “promising” node in the search graph to goal, based on cost-to-goal and heuristic values, is quickly determined.

Inconsistent nodes are popped out of the priority queue,  $\mathcal{Q}$  and repaired (lines 3–14), i.e. made consistent until the robot or start node becomes consistent or the queue becomes empty (usually encountered during the pre-planning phase). The key-comparisons in lines 2 and 5 are based on lexicographic comparison – the second entry becomes relevant only during tie-breaker among first entries [15].

---

**Algorithm 7** updateVertex( $v$ )

---

```

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

```

---

updateVertex( $v$ ) (Algorithm 7) computes  $lmc$  of vertex  $v$  (line 1) based on Eq. (7). The node,  $v$  is removed from the priority queue (lines 3–5) and added to the priority queue with the updated key value only if it is inconsistent (lines 6–9). The priority value given by computeKey() is computed using Eq. (8). computeHeuristic( $v$ ) calculates the admissible heuristic value – Euclidean distance from  $v$  to  $q_{start}$ . findParent( $v$ ) (line 2) determines the best parent of the node,  $v$  by analysing its outNeighbors,  $N^+(v)$ .

$$parent(v) \leftarrow \arg \min_{v' \in N^+} \{c(v, v') + g(v')\} \quad (12)$$

The priority queue is implemented using a binary heap. The queue operations are briefly described as follows— $\mathcal{Q}.\text{push}(v, key)$  inserts the element,  $v$  into the queue at the appropriate place based on key-value.  $\mathcal{Q}.\text{pop}()$  removes the top element of the queue and returns it.  $\mathcal{Q}.\text{remove}(v)$  removes the entry,  $v$  and rebalances the heap. Lastly,  $\mathcal{Q}.\text{topKey}()$  returns the key-value of the top-most element in the queue.

7) **Robot motion:** The various modules in Algorithm 2–sensing (lines 12–15), planning (line 16) and robot motion (lines 17–26) have different operating frequencies. This is implemented by running the methods on separate threads with different frequencies. robotMove( $q_1, q_2$ ) in line 18 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 8). The “head” of the modified edges, i.e.  $v$  in  $e = (v, w)$  are checked for inconsistencies, and added to the priority queue if inconsistent using updateVertex() (lines 4 and 7).

8) **Collision checking:** We exploit the geometric properties of the funnel and environment to come up with computationally efficient subroutine – collisionFree( $\mathcal{F}, \mathcal{O}$ ) for checking

---

**Algorithm 8** modifyEdgeCosts( $\Delta\mathcal{O}$ )

---

```

1: for all  $e = (v, w) \in E_m$  do ▷ “motion-edges” set
2:   if  $\neg \text{collisionFree}(e, \Delta\mathcal{O})$  then
3:      $c(v, w) \leftarrow \infty$ 
4:     updateVertex( $v$ )
5:   else ▷ if edges become free
6:      $c(v, w) \leftarrow c_{prev}$ 
7:     updateVertex( $v$ )
8:   end if
9: end for

```

---

overlaps with obstacle set,  $\mathcal{O}$ . A funnel  $\mathcal{F}$  is said to be *in-collision* if  $\mathcal{P}_{\mathcal{W}}^{\mathcal{S}}(\mathcal{F}) \cap \mathcal{O} \neq \emptyset$ , where  $\mathcal{P}_{\mathcal{W}}^{\mathcal{S}}(\cdot)$  is the projection of funnel down to the workspace.

Assuming obstacles with locally Lipschitz continuous boundaries, we perform collision-checks between obstacles and the projected level-sets of a funnel. A bounding-volume check constitutes as the first-pass in collision detection. If it fails, the individual ellipsoids of the funnel are checked for collision, in the order given by a Van der Corput sequence [45]. We implement a similar method of forming a convex hull around the obstacle and checking whether the extreme points lie within the ellipsoid using Eq. (14). For a general class of obstacles, one can resort to off-the-shelf software such as RoboDK [46], MPK [47] for collision-detection.

Space	Features – elements, routines, operations
$\mathbb{R}^+ \times \mathbb{R}^n$	funnels, steering, <i>compossibility-check</i>
$\mathcal{C}$ -space	configurations, sampling, <i>re-planning</i>
Workspace	robot, obstacles, <i>collision-checking</i>

#### D. Computing regions of finite-time invariance

Determining a closed-form solution to Eq. (3) from 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 candidate functions, the problem of computing the funnels can be reformulated into a Sum-of-Squares (SoS) program [26]. Consider a quadratic Lyapunov candidate function centred around the nominal trajectory,  $\mathbf{x}_0(t)$  defined using a positive definite matrix,  $P(t)$

$$V(t, \mathbf{x}) = (\mathbf{x} - \mathbf{x}_0(t))^T P(t) (\mathbf{x} - \mathbf{x}_0(t)) \quad (13)$$

For the class of piecewise polynomials  $P(t)$ , we solve the SoS program using polynomial  $\mathcal{S}$ -procedure [48]. The convex optimisation problem of maximising the funnel volume while satisfying constraints Eq. (3) is solved using bilinear alternation – improving  $\rho(t)$  and finding Lagrange multipliers to satisfy negativity of  $(\dot{V}(t, \mathbf{x}) - \dot{\rho}(t))$  in the semi-algebraic sets [27]. This maximises the inner-approximation of the verified regions of invariance around the nominal trajectory [30].

As noted in [27], we observe that time-sampled relaxations in the semi-definite program improve computational efficiency while closely resembling the actual level-sets. Therefore, we leverage this result to carry out optimisations only at discrete time instances between the knot points along the finite time

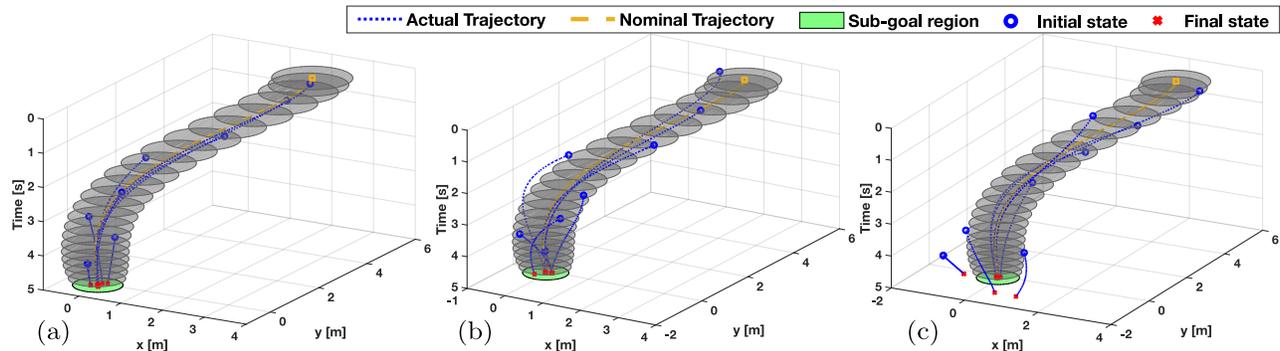


Fig. 6: A funnel computed based on Lyapunov level-set theory offers a sufficient but not a necessary condition for invariance. (a) Trajectories starting inside the funnel stay within the funnel; one can not conclusively comment about trajectories starting outside the funnel - it may (b) or may not (c) terminate within the desired goal set (green region)

interval. For  $M \in S_+^n$ , set of  $n \times n$  symmetric, positive definite matrices and  $c \in \mathbb{R}^n$ , Eq. (14) represents an ellipsoid centred around  $c$ .

$$(\mathbf{x} - c)^T M (\mathbf{x} - c) = 1 \quad (14)$$

The invariant sets,  $\mathcal{B}(t)$  in Eq. (2) corresponding to the quadratic Lyapunov function defined in Eq. (13) are the closed set, i.e. interior and boundary of ellipsoids centered around the nominal trajectory,  $\mathbf{x}_0(t)$ .

$$\mathcal{E}(t) = P(t)/\rho(t) \quad (15)$$

The closed-loop dynamics is derived using the feedback control policy,  $\mathbf{u}(t, \mathbf{x})$  and the system equations. Given a nominal trajectory in finite time interval, the dynamics are approximated to polynomial equations about it using Taylor-series expansion. 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.

Sample funnels computed for quadrotor dynamics with nominal control (presented in Section VI-A) have been illustrated in Fig. 6. Note that the funnels have been projected from  $\{t\} \times \mathbb{R}^n$  down to lower dimensional subspace,  $\{t\} \times \mathbb{R}^2$  for visualisation. Funnels calculated using above-mentioned methods have formal guarantees of invariance, ensuring that robot-trajectory stays within the backward-reachable set, if it starts within the funnel (see Fig. 6-a).

### E. Designing the funnel library

We deal with Lagrangian systems with time-invariant dynamics a.k.a autonomous systems. Considering the state-feedback controller, the closed-loop dynamics reduces down to  $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$ . Hence, for ease of usage we shift the initial time to origin,  $t_0 = 0$ , and the time interval becomes  $[0, T_f]$ , where  $T_f \in \mathbb{R}^+$  is a finite non-negative real number.

The funnel library,  $\mathcal{L}$ , consists of a finite number of verified trajectories, encapsulating the information of the certified regions of invariance in finite-time interval. Each funnel,

$\mathcal{F}_i \in \mathcal{L}$ , is parametrized by the nominal trajectory,  $\mathbf{x}_{0_i}(t)$ , the ellipsoidal level-sets,  $\mathcal{E}_i(t)$  and the final time,  $T_f$ . The trajectories and the ellipsoids are projected from the state-space onto  $\mathcal{C}$ -space using an appropriate projection operator,  $\mathcal{P}_{\mathcal{C}}^{\mathcal{E}}(\cdot) : \mathbb{R}^n \rightarrow \mathcal{C}$ . The resultant projection of invariant sets take the form of ellipsoids when projected onto the  $d$ -dimensional Euclidean subspace of robot-configurations [49],

$$S(t) = (B^T \mathcal{E}(t)^{-1} B)^{-1} \quad (16)$$

where  $B$  is a  $n \times d$  matrix, consisting of the basis vectors

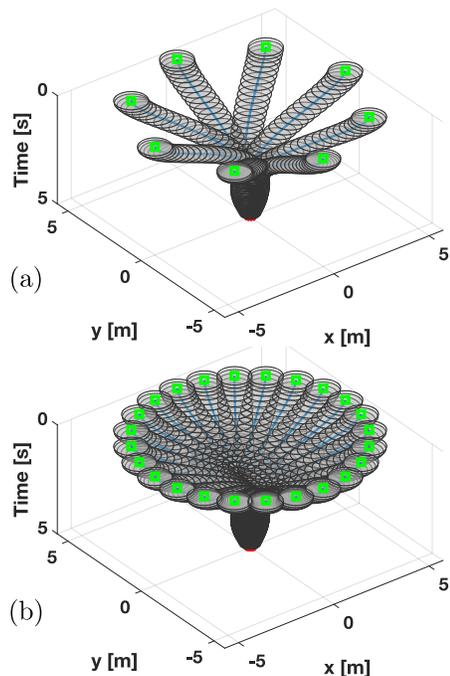


Fig. 7: Funnel library,  $\mathcal{L}$ . The initial configurations at  $t = 0$  lie at an  $\epsilon$ -distance from  $\mathbf{0}$  ( $\epsilon = 5m$ ). Desired final-states is a compact set centered around origin with radius  $0.3m$ . Using  $\mathcal{L}_b$  (bottom) will result in a finer resolution of motion-plan than  $\mathcal{L}_a$  (top)

of the coordinates of the  $\mathcal{C}$ -space in state space. Additionally, the funnels in the library are projected down to the robot-workspace in the pre-processing phase to speedup collision-checking with obstacles during runtime.

The funnel library acts like a bridge between the offline and online phases – invariant set analysis and motion planning. Therefore, certain algorithm parameters such as extend-distance, resolution of the planner, range of obstacle sizes, etc. is considered while constructing the library. Meanwhile, the funnel library provides the vital information of compossibility required during the online phase of motion planning.

Fig. 7 illustrates examples of funnel libraries, with nominal trajectories starting at an  $\epsilon$ -distance from origin, at various translational positions and terminating in a sub-goal region centered around origin at final time,  $T_f$ . These invariant sets can be shifted along the cyclic coordinates, and time to ensure the final pose of the system state aligns with the sampled configurations. Fig. 7-a depicts a sparser library which would result in lesser resolution of the motion plan.

It is worth mentioning that the initial conditions of the finite number of projected trajectories in the library, along with the appropriate shift operator,  $\Psi_c(\cdot)$  along the cyclic coordinates should be able to span the entire  $\mathcal{C}$ -space. This ensures *probabilistic coverage* [26] of the sampling-based motion planning algorithm and hence, probabilistic completeness.

## VI. EXPERIMENTAL VALIDATION

We validate our algorithm on a quadrotor UAV in simulations, flying through an indoor space with dynamic obstacles. This section briefly describes the dynamics of a quadrotor with a nominal controller, and discusses our experiments with implementation details. We demonstrate the relevance of invariant sets and empirically guarantee completeness and correctness of our motion re-planning algorithm.

### A. System Dynamics & Mission Profile

The equations of motion of a quadrotor UAV are derived using Newton-Euler formulation [50]. Considering position,  $\xi = [x, y, z]^T$  and the attitude,  $\eta = [\phi, \theta, \psi]^T$  of the quadcopter defined in an inertial frame, the equations can be written as,

$$\begin{aligned} \dot{\xi} &= v \\ m\dot{v} &= -ge_3 + Re_3T \\ \dot{\eta} &= W_\eta\omega \\ J\dot{\omega} &= -\omega \times J\omega - J_r(\omega \times e_3)\Omega + M \end{aligned} \quad (17)$$

Where,  $m$  is the mass,  $J$  the inertia matrix,  $v$  the linear velocity and  $\omega$  is body angular rates.  $e_3$  is  $[0 \ 0 \ 1]^T$ ,  $J_r$  is inertia of the rotor,  $\Omega = \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4$  is the net rotor speed.  $\Omega_i$  denotes the rotational speed of the individual rotors.

$R \in SO(3)$  is the rotation matrix from the body frame to the inertial frame.  $W_\eta$  is the transformation matrix for angular velocities in the body frame to inertial frame. For the specific

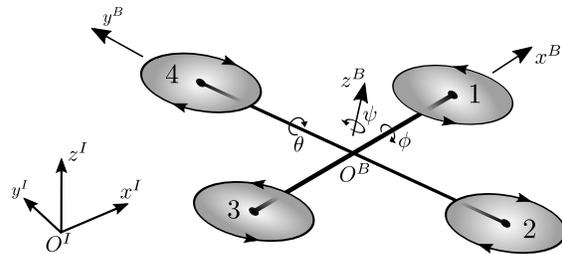


Fig. 8: Schematic of a quadrotor with 6DOF (position and attitude) in inertial,  $I$  and body-fixed,  $B$  frames of reference

configuration of rotors as in Fig. 8, Thrust,  $T$  and Moment,  $M$  are defined as,

$$\begin{aligned} T &= k \sum_{i=1}^4 \Omega_i^2 \\ M &= \begin{bmatrix} M_\phi \\ M_\theta \\ M_\psi \end{bmatrix} = \begin{bmatrix} kl(\Omega_4^2 - \Omega_2^2) \\ kl(\Omega_3^2 - \Omega_1^2) \\ d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{bmatrix} \end{aligned} \quad (18)$$

where  $k$  is the thrust coefficient,  $d$  the counter-moment drag coefficient and  $l$  is the arm length.

The state of the system,  $x = [\xi \ v \ \eta \ \omega]^T$  with 4 rotor speeds as inputs,  $u = [\Omega_1 \ \Omega_2 \ \Omega_3 \ \Omega_4]^T$ . Typically, the controller architecture has a cascaded structure, with a fast inner loop stabilising the attitude and a outer loop tracking the position or velocities [51]. 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. This has been found to be effective in maneuvers which don't require large deviations from nominal hover conditions [52]. The outer loop tracks the desired position setpoints and is achieved using an LQR controller [53]. Equivalently, the inputs to the quadrotor position controller are the desired setpoints  $-[x_d, y_d, z_d, \psi_d = 0]^T$ .

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 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  $\mathbb{R}^2$ , and it suffices to check for possible collisions in the 2D plane between ellipses and obstacles. The start configuration and goal region are defined in the  $xy$ -plane. The library is appropriately constructed, see Fig. 7.

### B. Experimental setup

The equivalent *closed-loop position* dynamics of the quadrotor is derived from repeated trials with various position setpoints,  $\xi_d$  given as inputs to the system.

$$\dot{\xi} = v \quad \dot{v} = f(\xi, v, u) \quad u = \xi_d \quad (19)$$

The system identification of the translational subsystem,  $x = [\xi \ v]$ , is carried out using SysId toolbox in MATLAB. The identified equations are then approximated to polynomial dynamics using a third-order Taylor-series expansion about the

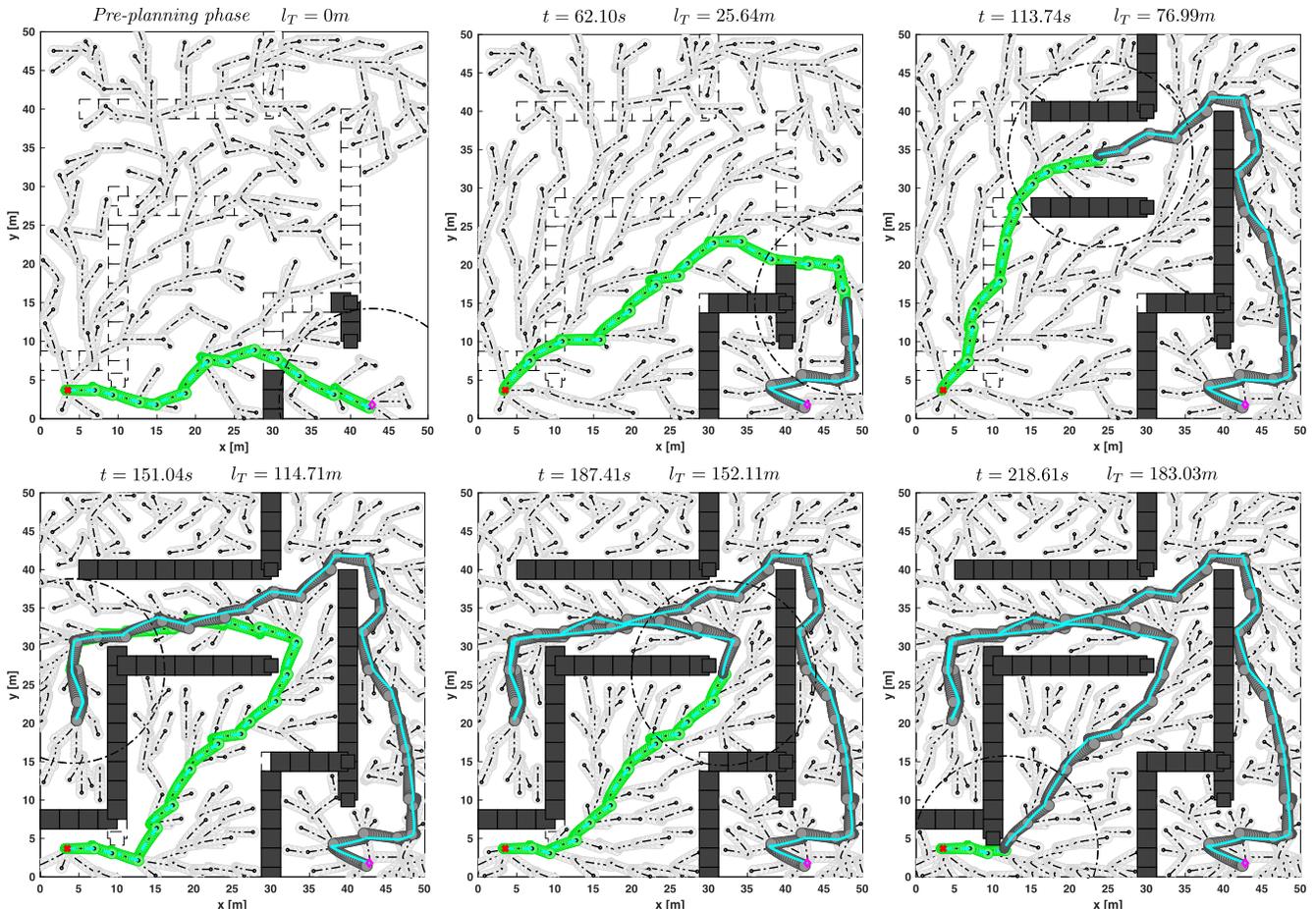


Fig. 9: Time instances of motion plan executed by PiP-X on a quadrotor flying with altitude-hold. The start configuration is in the lower-right corner, with the goal location at lower-left corner. The quadrotor senses obstacle-walls (solid rectangles) within sensor-radius (dashed circle), and recomputes motion plans (green *funnel-path*) accordingly. The traversed funnels and funnel search-tree are denoted by dark and light gray, respectively.  $l_T$  denotes the traversed-path length, and  $t$  denotes time elapsed. Note that the robot-trajectory (cyan) with the funnels and obstacles are projected down to  $(x, y)$  subspace

nominal trajectory. An estimate of the required final time of finite-interval,  $[0, T]$  is obtained based on the time taken by the system to reach within 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 V-D. The Sum-of-Squares optimisation is converted to an SDP by Systems Polynomial Optimisation Toolbox (SPOT) in MATLAB and solved using SeDuMi [54].

Our reverse-search algorithm requires backward-reachable set – starting from different initial conditions, the desired setpoint is given as the origin,  $\xi_d = \mathbf{0}$ . The initial  $(x, y)$  conditions lie on an  $\epsilon$ -circle centered at origin, with  $z(0) = h$  and  $\psi(0) = 0$ , as shown in Fig. 7. All the funnels computed are stored in a dictionary with a key-identifier based on the starting location. Each funnel in the library is characterised by its key, the nominal trajectory, ellipsoidal invariant sets and nominal control inputs - rotor speeds,  $\Omega_i$  at discrete time-steps within the finite-time horizon.

In order to verify the motion plan, we develop a higher

fidelity model in Simulink. The dynamics are based on Eq. (17) - (18). In addition to that, we incorporate actuator saturations, rotor dynamics and process noise. We believe that these additions will enable the simulation model to closely resemble the physical system. Using the funnel library as in Fig. 7-b, our algorithms are tested in two different 2D environments, *random forest* and *maze*, with various types of obstacle-changes, described in the next section.

### C. Results

A sample run of PiP-X in a user-specified maze environment is shown in Fig. 9. The robot perceives obstacle-walls of the maze within a finite sensing radius and updates its motion plans accordingly. We notice that a considerable amount of the  $\mathcal{C}$ -space is explored with fewer samples. Hence, using volumetric verified trajectories potentially speeds up the rate of probabilistic coverage. The position setpoints in the funnel-path to goal, output by the algorithm are given to the simulation system in real-time, and the system's actual trajectory is analysed. It is observed that the trajectory of the system lies

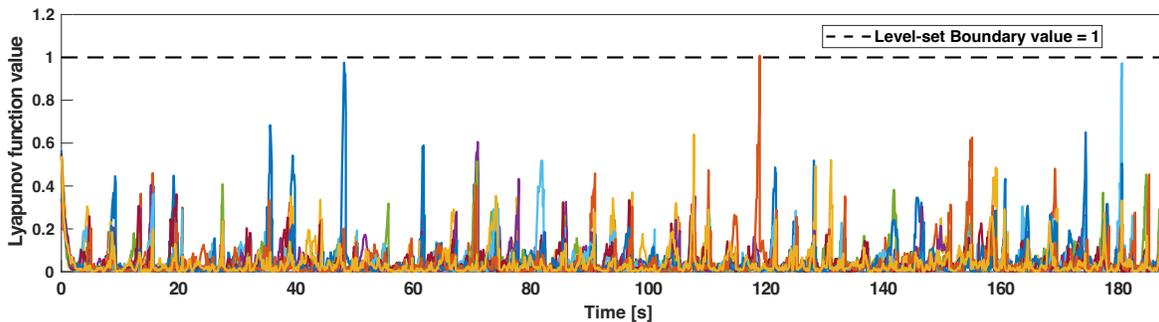


Fig. 10: *Normalised* Lyapunov function value of the system’s state simulated using the higher-fidelity quadrotor model, with solution funnel-path given by the algorithm across 10 different trials (denoted by different line-colors)

within the solution funnel-path throughout the course of the mission profile, verifying set-invariance.

In another scenario within the same maze environment with different start/goal location, we analyse the normalised Lyapunov function value of the system, Eq. (13) - Eq. (15). From Fig. 10, 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 2 funnels.

Our algorithm, based on quick graph-based replanning, is able to repair motion plans on-the-fly, ensuring a sequence of safe trajectories that are dynamically feasible. The theoretical guarantee of set-invariance enables our algorithm to rewire motion plans, implicitly addressing the two-point BVP encountered during search-tree rewiring in most sampling-based motion planners. Computing a shortest-path tree rooted at the goal results in an optimal path with respect to the iteratively-constructed underlying search-graph.

We extensively test our algorithm in two different kinds of environments – initially unknown, and dynamically changing – random forest and maze, across various scenarios and conditions; considering algorithm success and length of traversed trajectory as performance metrics. Algorithm failure is defined to be the robot’s inability to compute a feasible motion plan within a user-defined timeout or its collision with an obstacle.

1) **Initially unknown Random Forest with robot-sensing:** We consider a 2D workspace of dimensions  $50m \times 50m$  with circular obstacles of random sizes within the range of  $[2\ 4]m$ , and at random locations. The sensing radius of the quadrotor is  $12m$ . Each scenario is characterised by the number of tree-obstacles,  $N_t$  present in the environment. In each scenario, we vary the start and goal configurations, and obstacle locations. The start and goal locations are spaced out  $40m$  diametrically apart, for uniformity while comparing performance. The goal region is defined to be a  $\delta$ -ball of radius  $0.3m$  centered at the goal configuration.

The number of tree-obstacles are varied from 0–150 in increments of 1. 30 different trials are run in each scenario, reporting mean/standard deviation of success rate, and mean/range of traversed-trajectory length (see Fig. 11). The nominal path length, i.e if there are no obstacles and the robot is holonomic, is  $40m$ .

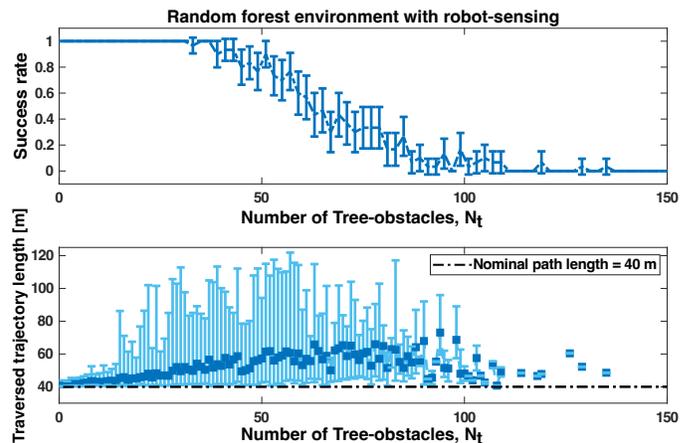


Fig. 11: Performance metrics in random forest environment with finite-sensing (a) mean and standard deviation of success rate (b) mean and range of traversed-trajectory length, across 30 trials

We observe that the algorithm has 100% success rate until  $N_t = 32$  (14.4% of workspace being obstacle space). The success rate drops from 1 to 0 as  $N_t$  increases. The algorithm fails after 134 trees in the forest, which approximately translates to 61.04% of workspace covered with obstacles. As expected, path length increases with number of obstacles, until we start encountering algorithm failure; wherein a few successful trials skews the average trajectory-length.

2) **Random Forest with dynamic obstacles:** Similar to the previous workspace, this environment is such that tree-obstacles are deleted and added at random, emulating a dynamic setup. The changes occur anywhere in the workspace and the robot is capable of sensing all those changes. A scenario is described by number of trees,  $N_t$  and change-percentage,  $C$ . For e.g., a change,  $C = 60\%$  in a workspace with  $N_t = 45$  implies 27 pre-existing trees are removed and 27 new obstacles are added – changing location and size.  $C = 0\%$  trivially refers to a static environment.

Taking input from previous experimental analysis, we consider the range of  $N_t$  to be  $[5, 135]$  in increments of 10. The change-percentage is varied from 0 to 100 in steps of 10. We run 25 trials (different start/goal configurations) and report mean of success rate and traversed-trajectory length in

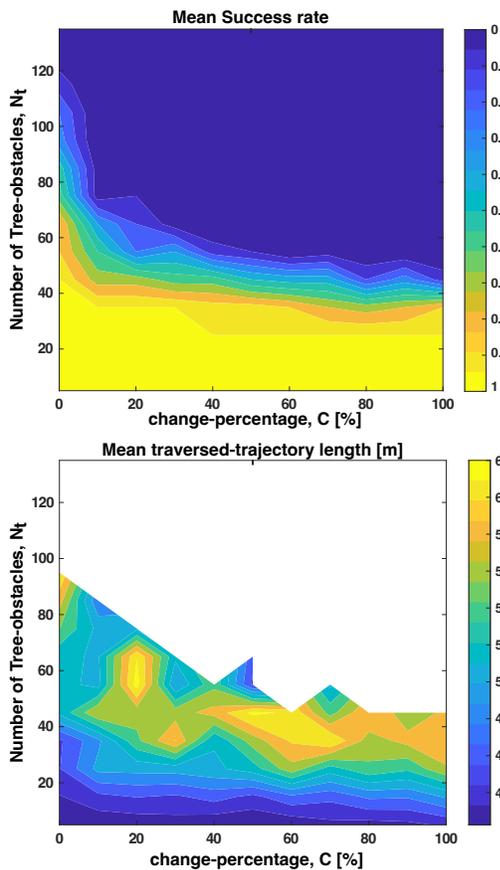


Fig. 12: Forest environment with dynamic obstacles – mean of success-rate and traversed-trajectory length from 25 trials

the form of a contour plot– Fig. 12.

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. 12-a). Note that trajectory-length in static environments ( $C = 0\%$ ) with  $N_t \geq 95$  is not visualised in Fig. 12-b as they are isolated instances of algorithm successes.

3) **Initially-unknown Maze with finite robot-sensing:** A maze-like environment with rectangular walls is designed in a two-dimensional  $50m \times 50m$  workspace, similar to the one in Fig. 9. The robot senses the obstacle-walls within a limited radius of  $12m$ .

25 trials are run for 10 different scenarios of start/goal configurations, and mean/standard deviation of success rate is presented in Fig. 13-a. Since the start and goal locations are such that a solution path exists, our algorithm is always capable of computing an initial motion plan and accordingly replan as new obstacle-walls are perceived. This results in a 100% success rate across 10 different scenarios.

4) **Maze with dynamic changes:** Similar to the previous environment, we additionally incorporate dynamic obstacles in the form of “windows” that can open (obstacle-deletion) or close (obstacle-addition) at random, and the robot is capable of sensing such obstacle-changes.

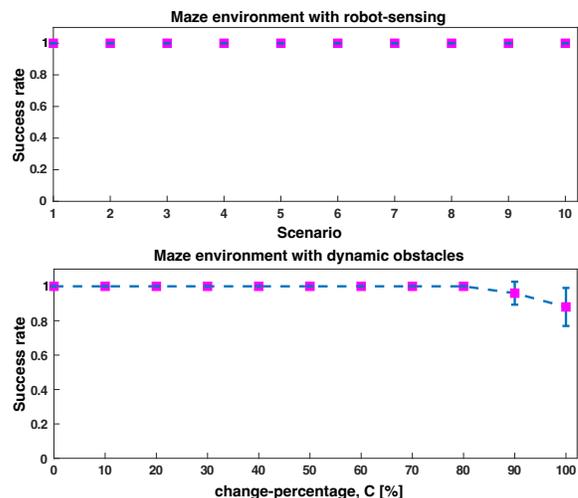


Fig. 13: Mean and standard deviation of algorithm success rate from 25 trials in a maze-environment with (a) finite robot-sensing, and (b) dynamic obstacles

The maze has 20 walls and 10 windows – 33.8% of wall-length (assuming uniform obstacle-width). Change-percentage,  $C$  refers to how many of the windows open and close at sensing-frequency – it is varied from 0% (static) to 100% (dynamic) in increments of 10. From Fig. 13, we observe a drop in success at  $C = 90\%$  and  $C = 100\%$  (highly dynamic). Note that we consider only success rate because the path length depends on the contour of the maze and start/goal configurations – there is no nominal path length for reference.

As a general observation from all experiments and scenarios, most failures are due to “idleness” time-outs,  $I_M$  implying the algorithm’s inability to identify and report that a solution doesn’t exist, as is with the case of all sampling-based motion planning techniques. 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.

## VII. CONCLUSIONS

A novel sampling-based online feedback motion re-planning algorithm using funnels, PiP-X is presented. The search graph with funnel-edges is iteratively constructed using sampling-techniques, and concurrent calculation of the shortest-path subtree of funnels rooted at the goal ensures optimal funnel-path from robot configuration to the goal region. The use of incremental graph-replanning techniques and a pre-computed library of motion primitives ensure that our method can repair paths *on-the-fly* in dynamic environments.

The information of robot-traversability and funnel-sequencibility is represented together in the form of an *augmented directed-graph*, helping us leverage numerical graph-search methods to compute safe, controllable motion-plans. Analysing and formally quantifying stability of trajectories using Lyapunov level-set theory ensures kinodynamic feasibility of the solution-paths. Additionally,

verifying the compossibility of a funnel-pair proves to be 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 environments and scenarios. Performance of the algorithm in terms of success and trajectory-length is examined. Our approach combines concepts from systems analysis, sampling-based methods, and incremental graph-search to address *feedback motion re-planning* for any general nonlinear robot-system in dynamic workspaces.

## ACKNOWLEDGMENTS

The authors are grateful to the Robot Locomotion Group at MIT for providing an open-source software distribution for computing funnels. We like to thank Sharan Nayak for the valuable discussions on this work. We also thank the members of Motion and Teaming Lab, UMD – Loy McGuire, Alex Mendelsohn, Alkesh K. Srivastava and Dalan C. Loudermilk for their feedback during the preparation of this manuscript.

## REFERENCES

- [1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
- [2] J. J. Kuffner and S. M. LaValle, “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. IEEE, 2000, pp. 995–1001.
- [3] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.
- [4] O. Arslan and P. Tsiotras, “Use of relaxation methods in sampling-based algorithms for optimal motion planning,” in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 2421–2428.
- [5] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, “Randomized kinodynamic motion planning with moving obstacles,” *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.
- [6] S. Karaman and E. Frazzoli, “Optimal kinodynamic motion planning using incremental sampling-based methods,” in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 7681–7687.
- [7] S. M. LaValle and J. J. Kuffner Jr, “Randomized kinodynamic planning,” *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.
- [8] M. Kleinbort, K. Solovey, Z. Littlefield, K. E. Bekris, and D. Halperin, “Probabilistic completeness of rrt for geometric and kinodynamic planning with forward propagation,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. x–xvi, 2018.
- [9] J. hwan Jeon, S. Karaman, and E. Frazzoli, “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*. IEEE, 2011, pp. 3276–3282.
- [10] D. J. Webb and J. Van Den Berg, “Kinodynamic RRT\*: Asymptotically optimal motion planning for robots with linear dynamics,” in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 5054–5061.
- [11] A. Ravankar, A. A. Ravankar, Y. Kobayashi, Y. Hoshino, and C.-C. Peng, “Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges,” *Sensors*, vol. 18, no. 9, p. 3170, 2018.
- [12] M. Basescu and J. Moore, “Direct NMPC for post-stall motion planning with fixed-wing UAVs,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9592–9598.
- [13] A. Stentz *et al.*, “The focussed d\* algorithm for real-time replanning,” in *IJCAI*, vol. 95, 1995, pp. 1652–1659.
- [14] S. Koenig, M. Likhachev, and D. Furcy, “Lifelong planning A\*,” *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.
- [15] S. Koenig and M. Likhachev, “D\*lite,” *Aaai/iaai*, vol. 15, 2002.
- [16] J. Bruce and M. M. Veloso, “Real-time randomized path planning for robot navigation,” in *Robot soccer world cup*. Springer, 2002, pp. 288–295.
- [17] D. Ferguson, N. Kalra, and A. Stentz, “Replanning with RRTs,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, ICRA 2006*. IEEE, 2006, pp. 1243–1248.
- [18] M. Zucker, J. Kuffner, and M. Branicky, “Multipartite rrts for rapid replanning in dynamic environments,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 1603–1609.
- [19] M. Otte and E. Frazzoli, “RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning,” *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 797–822, 2016.
- [20] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, “Batch informed trees (BIT\*): Informed asymptotically optimal anytime search,” *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020.
- [21] D. Connell and H. M. La, “Dynamic path planning and replanning for mobile robots using RRT,” in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2017, pp. 1429–1434.
- [22] O. Adiyatov and H. A. Varol, “A novel RRT\*-based algorithm for motion planning in dynamic environments,” in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE, 2017, pp. 1416–1421.
- [23] Y. Li, Z. Littlefield, and K. E. Bekris, “Asymptotically optimal sampling-based kinodynamic planning,” *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.
- [24] M. Mason, “The mechanics of manipulation,” in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 1985, pp. 544–548.
- [25] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, “Sequential composition of dynamically dexterous robot behaviors,” *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999.
- [26] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, “LQR-trees: Feedback motion planning via sums-of-squares verification,” *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.
- [27] M. M. Tobenkin, I. R. Manchester, and R. Tedrake, “Invariant funnels around trajectories using sum-of-squares programming,” *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 9218–9223, 2011.
- [28] A. Majumdar and R. Tedrake, “Funnel libraries for real-time robust feedback motion planning,” *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.
- [29] J. Moore, R. Cory, and R. Tedrake, “Robust post-stall perching with a simple fixed-wing glider using LQR-trees,” *Bioinspiration & biomimetics*, vol. 9, no. 2, p. 025013, 2014.
- [30] A. Majumdar, A. A. Ahmadi, and R. Tedrake, “Control design along trajectories with sums of squares programming,” in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 4054–4061.
- [31] P. Reist, P. Preiswerk, and R. Tedrake, “Feedback-motion-planning with simulation-based lqr-trees,” *The International Journal of Robotics Research*, vol. 35, no. 11, pp. 1393–1416, 2016.
- [32] C. K. Verginis, D. V. Dimarogonas, and L. E. Kavraki, “KDF: Kinodynamic motion planning via geometric sampling-based algorithms and funnel control,” *arXiv preprint arXiv:2104.11917*, 2021.
- [33] A. Bajcsy, S. Bansal, E. Bronstein, V. Tolani, and C. J. Tomlin, “An efficient reachability-based framework for provably safe autonomous navigation in unknown environments,” in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 1758–1765.
- [34] S. Kousik, S. Vaskov, F. Bu, M. Johnson-Roberson, and R. Vasudevan, “Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots,” *The International Journal of Robotics Research*, vol. 39, no. 12, pp. 1419–1469, 2020.
- [35] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, “FaSTrack: A modular framework for fast and guaranteed safe motion planning,” in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 1517–1522.
- [36] S. Singh, B. Landry, A. Majumdar, J.-J. Slotine, and M. Pavone, “Robust feedback motion planning via contraction theory,” *The International Journal of Robotics Research*, 2019.
- [37] Z. Manchester and S. Kuindersma, “Robust direct trajectory optimization using approximate invariant funnels,” *Autonomous Robots*, vol. 43, no. 2, pp. 375–387, 2019.
- [38] H. Ravanbakhsh, F. Laine, and S. A. Seshia, “Real-time funnel generation for restricted motion planning,” *arXiv preprint arXiv:1911.01532*, 2019.

- [39] J.-J. E. Slotine, W. Li, *et al.*, *Applied nonlinear control*. Prentice hall Englewood Cliffs, NJ, 1991, vol. 199, no. 1.
- [40] H. K. Khalil and J. W. Grizzle, *Nonlinear systems*. Prentice hall Upper Saddle River, NJ, 2002, vol. 3.
- [41] D. Dey, T. Liu, B. Sofman, and J. Bagnell, "Efficient optimization of control libraries," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, 2012.
- [42] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE transactions on robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.
- [43] A. Majumdar and R. Tedrake, "Robust online motion planning with regions of finite time invariance," in *Algorithmic foundations of robotics X*. Springer, 2013, pp. 543–558.
- [44] M. Penrose *et al.*, *Random geometric graphs*. Oxford university press, 2003, vol. 5.
- [45] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
- [46] RoboDK, "Simulation and OLP for robots." [Online]. Available: <https://robdk.com/>
- [47] I. Gipson, K. Gupta, and M. Greenspan, "Mpk: An open extensible motion planning kernel," *Journal of Robotic Systems*, vol. 18, no. 8, pp. 433–443, 2001.
- [48] P. A. Parrilo, "Semidefinite programming relaxations for semialgebraic problems," *Mathematical programming*, vol. 96, no. 2, pp. 293–320, 2003.
- [49] W. C. Karl, G. C. Verghese, and A. S. Willsky, "Reconstructing ellipsoids from projections," *CVGIP: Graphical Models and Image Processing*, vol. 56, no. 2, pp. 124–139, 1994.
- [50] P. C. Garcia, R. Lozano, and A. E. Dzul, *Modelling and control of mini-flying machines*. Springer Science & Business Media, 2006.
- [51] M. K. M. Jaffar, M. Velmurugan, and R. Mohan, "A novel guidance algorithm and comparison of nonlinear control strategies applied to an indoor quadrotor," in *2019 Fifth Indian Control Conference (ICC)*. IEEE, 2019, pp. 466–471.
- [52] T. Luukkonen, "Modelling and control of quadcopter," *Independent research project in applied mathematics, Espoo*, vol. 22, p. 22, 2011.
- [53] S. Bouabdallah, A. Noth, and R. Siegwart, "PID vs LQ control techniques applied to an indoor micro quadrotor," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2451–2456.
- [54] J. F. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization methods and software*, vol. 11, no. 1–4, pp. 625–653, 1999.



planning of aerial robots.

**Mohamed Khalid M. Jaffar** received his Dual Degree, Bachelor of Technology (B.Tech.) and Master of Technology (M.Tech.) in Aerospace Engineering from the Indian Institute of Technology Madras, Chennai, India in 2018. He is currently pursuing a Ph.D. degree in Aerospace Engineering, under the supervision of Dr. Michael Otte. He is currently a Research Assistant with the department of Aerospace Engineering, University of Maryland, College Park, MD, USA. His research focuses on the use of control and systems theory for motion



Associate at the U.S. Naval Research Lab. He has been with the Department of Aerospace Engineering, at the University of Maryland, College Park, MD, USA, since 2018. He is the author of over 30 articles, and his research interests include autonomous robotics, motion planning, and multi-agent systems.

**Michael Otte** (M'07) received the B.S. degrees in aeronautical engineering and computer science from Clarkson University, Potsdam, New York, USA, in 2005, and the M.S. and Ph.D. degrees in computer science from the University of Colorado Boulder, Boulder, CO, USA, in 2007 and 2011, respectively. From 2011 to 2014, he was a Postdoctoral Associate at the Massachusetts Institute of Technology. From 2014 to 2015, he was a Visiting Scholar at the U.S. Air Force Research Lab. From 2016 to 2018, he was a National Research Council RAP Postdoctoral