Journal Title
XX(X):1–22

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

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 an *online* single-query sampling-based motion *re-planning* algorithm using finite-time invariant sets, commonly referred to as *'funnels'*. We combine concepts from nonlinear systems analysis, sampling-based techniques, and graph-search methods to create a single framework that enables feedback motion re-planning for any general nonlinear dynamical system in dynamic workspaces.

A *volumetric* network of funnels is constructed in the configuration space using sampling-based methods and invariant set theory; and an *optimal* sequencing of funnels from robot configuration to a desired goal region is then determined by computing the shortest-path subgraph (tree) in the network. Analysing and formally quantifying the stability of trajectories using Lyapunov level-sets ensures kinodynamic feasibility and guaranteed set-invariance of the solution-paths. Though not required, our method is capable of using a pre-computed library of motion-primitives to speedup online computation of controllable motion plans that are volumetric in nature.

We introduce a novel *directed-graph data structure* to represent the funnel-network and its inter-sequencibility; helping us leverage discrete graph-based incremental search to quickly *rewire* feasible and controllable motion plans *on-the-fly* in response to changes in the environment. We validate our approach on a simulated cart-pole, car-like robot, and 6DOF quadrotor platform in a variety of scenarios within a maze and a random forest environment. Using Monte Carlo methods, we evaluate the performance in terms of algorithm-success, length of traversed-trajectory, and runtime.

Keywords

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

1 Introduction

The ability to replan is essential whenever a robot must explore an unknown or changing environment while using a limited sensor radius. In scenarios where the environment contains fast-moving obstacles or is densely cluttered with obstacles, a new motion-plan must be quickly calculated whenever newly gathered information about obstacles invalidates the current plan. Likewise, it is also essential that any global motion plan (or replan) respects the kinodynamic constraints and controllability of the robot-system. *Feedback motion planning* algorithms achieve such compatibility by considering this dual requirement of motion planning and robot control in tandem.

For robots that must react quickly to changes in the obstacle-space, the computational complexity of brute-force replanning—planning from scratch whenever the environment changes—is often impractical. In the case of feedback motion planning, the need to calculate *controllable* motion plans further increases the computational burden of replanning (especially when the *valid* state space is non-convex due to obstacles and/or other system constraints). 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 time-instance to plan/replan in the future can be adapted to achieve such quick replanning. However, the use of such methods in the literature so far has been limited to discrete grids and graphs in which edges represent one-dimensional trajectories through space-time.

This paper presents a sampling-based *online* motion planning/replanning algorithm using trajectories with certified regions of stability, commonly referred to as *funnels*. We extend sampling-based motion planners to nonlinear robot-systems in dynamic workspaces. Using systems analysis and invariant set theory, we compute 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. We extend the use of aforementioned incremental search algorithms to the case of *volumetric funnel-edges* by using a novel graph data structure to model a network of funnels. Additionally, our method is capable of using a library of funnels to speedup online computation.

A preliminary version of this work was presented at the 15th International Workshop on the Algorithmic Foundations of Robotics (WAFR 2022).
The authors are affiliated with the Department of Aerospace Engineering, University of Maryland, College Park, MD, USA

Corresponding author:
Mohamed Khalid M Jaffar, Motion and Teaming Laboratory, University of Maryland, 8228 Paint Branch Dr, College Park, MD 20742, USA.
Email: khalid26@umd.edu

The algorithm that we present is called **PiP-X** (**P**lanning/replanning **i**n **P**ipes in dynamic or initially unknown environments), a novel *kinodynamically-feasible*, *online re-planning** algorithm that is relevant in practical scenarios such as safely navigating through dynamic workspaces. Our method quickly computes, and continuously updates *controllable* motion plans with *formal invariance guarantees*, that can be safely tracked by nonlinear robot-systems despite changes in obstacle space. We recast the challenging problem of online *kinodynamically-feasible* motion re-planning into a discrete graph-search by mapping the network of *volumetric* funnels in the higher dimensional state space to a *directed-graph data structure* representation.

The use of funnels *in lieu* of one-dimensional trajectories through space-time for sampling-based re-planning is advantageous because it serves as a bridge between discrete computational methods (path planning algorithms) and theoretical analysis (system analysis and control design). We 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 obtain a re-planning algorithm that respects the closed-loop dynamics of the robot, computes the minimum-cost path (with formal invariance guarantees) to the goal, and also efficiently replans around dynamic obstacles.

To the best of the authors' knowledge, this is the first work to propose techniques for *funnel-based motion re-planning* using graph-based quick rewiring. It sets the foundations for *global* feedback motion planning/replanning *on-the-fly* by introducing a new form of *volumetric funnel-networks*, leveraging the rich literature in sampling-based planning, invariant set analysis and graph search. 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 a general nonlinear robot-system in dynamic spaces.

1.1 Statement of Contributions

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

1. PiP-X: *Global* feedback motion *re-planning* with *funnels*, using sampling-based techniques and incremental graph-search.
2. A novel technique to represent a network of funnels and its inter-sequencibility using a *bipartite graph data structure* with augmented vertices and edge-sets.
3. Implicitly addressing the *two-point boundary value problem* (TP-BVP) during graph rewiring by using the system's stability analysis and funnel sequencibility.

A preliminary version of this paper appeared in the conference proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR) (Jaffar and Otte 2023). Major extensions in this journal version include: a more detailed technical background with visualisations in Section 3, an in-depth description of our method and algorithm in Section 5, validation in two additional platforms: a cart-pole and a car-like robot (Section 6), and added performance and comparison analysis (Section 7.3–7.4) of our algorithm on a simulated quadrotor in two new environments: initially unknown forest with limited robot-sensing and maze with dynamic changes.

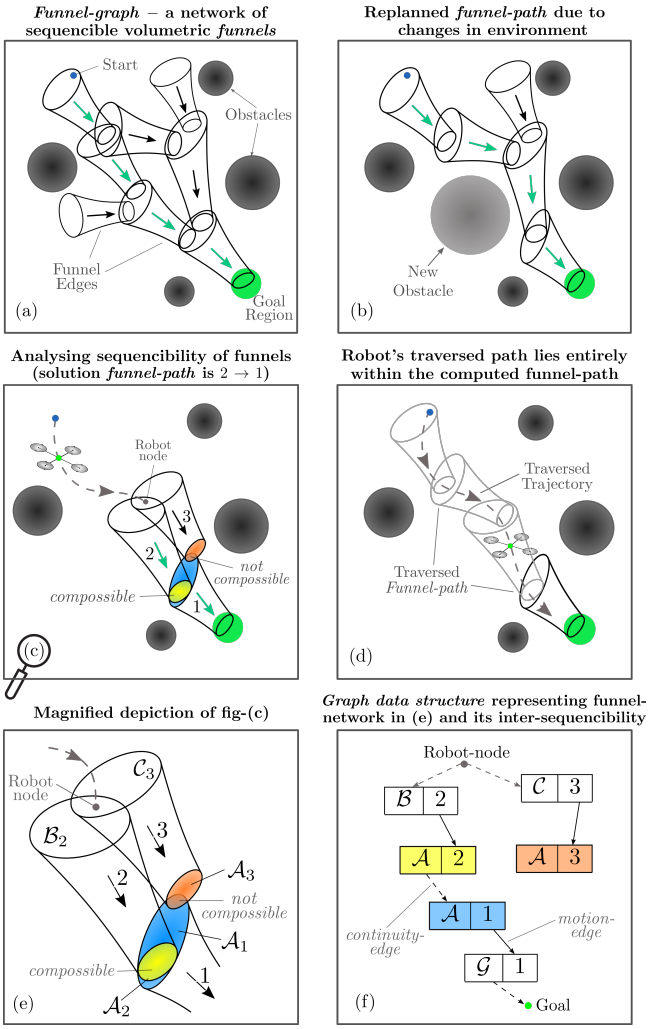

Figure 1. Online funnel-based re-planning algorithm, PiP-X: (a) Funnel-graph, a network of finite-time invariant sets – *funnels*, each "dropping" into the subsequent funnel, finally into the goal region. (b) Rewiring of the *funnel-path* when changes in the environment are sensed (c) Analysing compossibility of funnels (d) Trajectory of the robot lies completely inside the traversed funnel-path (e) Labelled funnel-edges, inlets and outlets (f) Representing funnel-connectivity (solid *motion-edges*) and inter-sequencibility (dashed *continuity-edges*) as a *graph data structure* to enable quick and efficient graph-based re-planning. The solution path is $\mathcal{B}_2 - \mathcal{A}_2 - \mathcal{A}_1 - \mathcal{G}_1$.

1.2 Outline

The remainder of this paper is structured as follows: Section 2 outlines the related work, and Section 3 provides the necessary theoretical background while introducing the notation used in the paper. Section 4 formally states our problem definition. Our approach is described in Section 5 and validated in Sections 6–7. Finally, we state our conclusions in Section 8.

**Re-plan*: plan *and* replan (to reflect the updated state of the environment)

2 Related Work

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

2.1 Sampling-based kinodynamic motion planning

Geometric sampling-based motion planners such as probabilistic roadmaps (PRM) (Kavraki et al. 1996) and rapidly exploring random trees (RRT) (Kuffner and LaValle 2000) have proven to be effective and practical in high-dimensional configuration spaces. Karaman and Frazzoli (2011) propose RRT* with theoretical proofs of asymptotic optimality. RRT$^\#$ algorithm presented by Arslan and Tsiotras (2013) 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 (Hsu et al. 2002; Karaman and Frazzoli 2010; Arteaga et al. 2021; Becerra et al. 2021).

Sampling-based kinodynamic variants *steer* the vehicle by randomly sampling control inputs and forward simulating the trajectory based on the dynamics (LaValle and Kuffner Jr 2001; Kleinbort et al. 2018). SST and (asymptotically optimal) SST* (Li et al. 2016) are kinodynamic planners that do not rely on a steering function, and instead use selective propagation/pruning to maintain a sparse tree. For complex systems with unstable dynamics, Kuwata et al. (2009) propose CL-RRT that uses closed-loop prediction for trajectory generation, growing a tree in the reference-trajectory space; Arslan et al. (2017) present an asymptotically optimal extension to it.

Some researchers formulate an optimal control problem with the trajectory given by the geometric planner, solved using shooting methods (hwan Jeon et al. 2011) or closed-form analytical solutions (Webb and Van Den Berg 2013). Another popular approach is to smooth the path given by geometric planners through splines or trajectory optimisation (Ravankar et al. 2018) and track it using feedback controllers such as PID or receding-horizon controller (Basescu and Moore 2020).

2.2 Graph-based motion replanning in dynamic environments

Earlier work on re-planning – D* (Stentz et al. 1995), LPA* (Koenig et al. 2004), D*Lite (Koenig and Likhachev 2002) 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 *feasible* motion re-planning problem – ERRT (Bruce and Veloso 2002), DRRT (Ferguson et al. 2006) and multipartite RRT (Zucker et al. 2007) completely prune the edges in collision, and attempt to rejoin the disconnected branches to the rooted tree. RRT^X proposed by Otte and Frazzoli (2016) was the first *asymptotically optimal* sampling-based re-planning algorithm. It rewires the shortest-path tree 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 a *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 (Gammell et al. 2020) and leverage the rewiring step from RRT* to locally repair the solution branch around newly-sensed obstacles (Connell and La 2017; Adiyatov and Varol 2017).

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 the kinodynamic re-planning problem without using a two-point BVP solver (Li et al. 2016), achieving near-optimality. However, most previous 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.

2.3 Feedback motion planning using funnels

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 (1985) introduced a metaphor – *funnel*, for locally stabilised and verified trajectories. An illustration of sequentially composed funnels reaching a goal region, similar to Fig. 1-a, presented by Burridge et al. (1999) sparked the motivation to use such funnels for feedback motion planning.

Tedrake et al. (2010) 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. Tobenkin et al. (2011) present 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 and Tedrake (2017) compute the funnels offline

and use them to plan online for flying a glider through a dense setting. Our approach shares much in common with this work, but differs in that it uses a formal *graph-framework* to represent an RRG of funnel-edges, as well as to *online replan* funnel-paths using *incremental* graph-repairs. Similar work by the same research group leverages the concepts of funnels to develop planning algorithms for UAV-perching (Moore et al. 2014), double pendulum (Majumdar et al. 2013), and a cart-pole (Reist et al. 2016). Funnel-based motion planning for a robotic arm using adaptive feedback control is presented by Verginis et al. (2021, 2023).

2.4 Other related work

In parallel to Lyapunov theory-based analysis, researchers have proposed different techniques for reachability set-based trajectory design (Bajcsy et al. 2019; Kousik et al. 2020). FaSTrack (Herbert et al. 2017) is an adversarial game-theoretical approach to generate worst-case tracking error bounds around trajectories using Hamilton-Jacobi reachability analysis. Singh et al. (2019) use contraction theory 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 (Manchester and Kuindersma 2019), and funnel generator functions (Ravanbakhsh et al. 2019).

Optimisation-based approaches such as CHOMP (Ratliff et al. 2009; Zucker et al. 2013) and TrajOpt (Schulman et al. 2014) are also popular in the context of motion re-planning in dynamic spaces. Such algorithms generally use a trajectory optimization framework with a receding horizon controller to avoid collisions while simultaneously satisfying smoothness and dynamics constraints (Borrelli et al. 2004; Park et al. 2012). They abstract the problem completely into a formal mathematical objective and solve the constrained optimisation problem numerically (Richards and How 2002; Blackmore et al. 2011). An application of using an optimisation-based collision avoidance framework for autonomous parking is presented by Zhang et al. (2020).

The ***scientific merit*** of our research is extending previous work in *funnel-based* motion planning to *global on-the-fly replanning* by using graph-based quick rewiring methods. Our primary contribution is that we present a novel graph data structure to map the volumetric funnel-network in the higher dimensional state space, helping us leverage graph-search algorithms to rewire controllable motion plans. Through this, we are able to recast the challenging problem of *feedback motion re-planning* for non-trivial systems into a geometric one, making it tractable for online replanning. Our algorithm generates kinodynamically feasible motion plans/replans for nonlinear robot-systems using backward reachable sets, while implicitly solving the two-point boundary value problem.

3 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 technical background while introducing the notation that will be used in the rest of the paper.

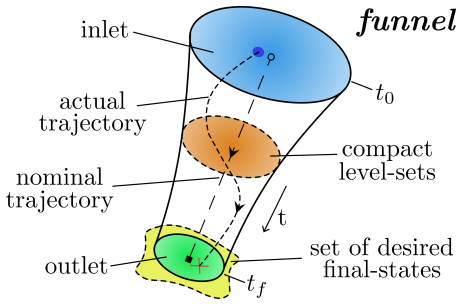

Figure 2. A sample funnel – finite-time backward-reachable invariant set to a compact region of desired final states, $\mathcal{X}_f$

3.1 Invariant Set Theory

The notion of region of attraction of asymptotically stable fixed points can be extended to certifying time-varying trajectories. Such regions of finite-time invariance around a trajectory are referred to as *funnels* (Tobenkin et al. 2011). Considering a closed-loop nonlinear system,

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t)) \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{1}$$

where state, $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{f}$ is Lipschitz continuous in $\boldsymbol{x}$ and piecewise continuous in t, guaranteeing global existence and uniqueness of a solution (Slotine et al. 1991). 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 $(\tau, \boldsymbol{x}_\tau) \in \mathcal{F}$, $\tau \in [t_0, t_f]$, the solution to Eq. (1), $\boldsymbol{x}(t)$, with initial condition $\boldsymbol{x}(\tau) = x_\tau$, lies entirely within $\mathcal{F}$ till final time, i.e. $(t, \boldsymbol{x}(t)) \in \mathcal{F}\ \forall\ t \in [\tau, t_f]$.

Intuitively, if the closed-loop system starts within the funnel, then the system states evolving due to 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, $V(t, \boldsymbol{x})$, satisfy the conditions of positive invariance (Khalil and Grizzle 2002),

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

where $\rho(t)$ is a non-negative real-valued function, describing the time-varying boundary value of level sets. As noted by Tobenkin et al. (2011), 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,

$$\begin{aligned} V(t, \boldsymbol{x}) = \rho(t) \implies \dot{V}(t, \boldsymbol{x}) \leq \dot{\rho}(t) \\ \dot{V}(t, \boldsymbol{x}) = \frac{\partial V(t, \boldsymbol{x})}{\partial t} + \frac{\partial V(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{f}(t, \boldsymbol{x}(t)). \end{aligned} \tag{3}$$

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 in Eq. (4), satisfying the conditions in Eq. (3) is a *funnel* (Moore et al. 2014).

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

In our case, we are interested in computing backward reachable sets about a finite-time, nominal trajectory to a compact region in 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. $(t_f, \boldsymbol{x}(t_f)) \in \mathcal{F} \implies \boldsymbol{x}(t_f) \in \mathcal{X}_f$, 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 analysis and convex optimisation. The funnel-volume is as defined by Tobenkin et al. (2011). Section 5.4 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.

3.2 Verified Trajectory Libraries

The idea of saving a pre-computed library of motion primitives for online planning has seen considerable research in the past. Dey et al. (2012) provide an optimisation-based approach to design the library, abstracting the information and sequence of trajectories. Maneuver Automaton by Frazzoli et al. (2005) discusses the relevant properties of trajectory libraries required for sequencing, providing a theoretical foundation.

The condition for sequencing trajectories presented by Burridge et al. (1999) 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, $\boldsymbol{x} = [\boldsymbol{x}_c^T \ \boldsymbol{x}_{nc}^T]^T$. Cyclic states are defined as (generalised) coordinates to which the open-loop dynamics of a Lagrangian system, $\dot{\boldsymbol{x}} = \boldsymbol{f}'(t, \boldsymbol{x}, \boldsymbol{u})$ are invariant. Or alternatively, the dynamics depend only on the non-cyclic states.

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

For example, in a holonomic ground robot with second-order dynamics, pose 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 (Majumdar and Tedrake 2013). One can shift the funnel along the cyclic coordinates so as to contain the outlet of the previous funnel, as illustrated in Fig. 3. Shift function, $\Psi_c(.)$ is a mapping from the space of funnels to itself that shifts (translates/rotates) the funnel along the cyclic coordinates while not transforming it about the non-cyclic coordinates.

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

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

where $\mathcal{P}^{\mathcal{S}}_{nc}(.)$ is the appropriate projection operator from the state space onto the subspace formed by non-cyclic coordinates. In addition to posing a less strict condition,

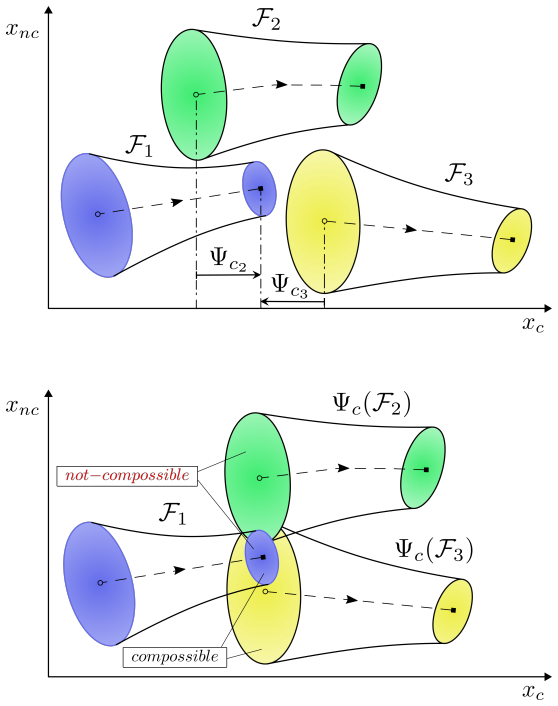

Figure 3. Compossibility of funnels from the library illustrating shifts, $\Psi_c(.)$ along cyclic coordinates (invariant dynamics). $(\mathcal{F}_1, \mathcal{F}_3)$ is *motion-plan compossible*, whereas $(\mathcal{F}_1, \mathcal{F}_2)$ is not; i.e. outlet of $\mathcal{F}_1$ is completely contained within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

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. Section 5.5 talks about projection operators and shift functions in detail for our specific case of funnels characterised by ellipsoidal level sets.

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 heading (cyclic coordinates) of the funnel can be shifted to the current pose of the UAV. Majumdar and Tedrake (2017) discuss the various notions of compossibility in detail, providing methods to check the condition in Eq. (6) using semi-definite programming.

3.3 Discrete graph replanning

An important theme throughout our work is the use of graphs to map a *network* of volumetric funnels in a way that respects their compossibility constraints, Eq. (6). To this effect, we present concepts from graph theory, and how we leverage the notions of *discrete search* to calculate optimal sequencing of funnels to a goal region. In the context of path planning, graphs serve a dual purpose of modelling the topology and traversability of configuration space $\mathcal{C}$, and also to search for a feasible or an optimal path through it. A graph is defined by a set of vertices V, and edges E, formally denoted as $\mathcal{G} = (V, E)$, where $E \subseteq \{(v, w) \mid v, w \in V, v \neq w\}$.

Most search algorithms compute a connected acyclic subgraph (tree) within the graph, such that each vertex has one, and only one, parent. A path from the start to a goal is readily found by backtracking parent pointers starting

from the goal or start, respectively, depending on the search-direction. Discrete *path planners* like Dijsktra or A* find an optimal path with respect to a user-defined cost function. Given an edge, $e = (v, w) \in E$, the edge cost function, $c(v, w)$ represents the cost to move from vertex v to vertex w. The edge costs satisfy the property of a *distance metric*: non-negativity, identity of indiscernibles and triangle inequality.

D*Lite (Koenig and Likhachev 2002) is a graph-based *incremental search* method that reuses all valid current information to improve the solution path in the next iteration, resulting in faster replanning speeds than algorithms that replan from scratch. It continually and efficiently *repairs* the shortest path-to-goal as edge costs change while robot traverses the solution path. Our algorithm adapts this idea to similarly maintain a shortest-path tree of *funnels* rooted at the goal region.

For each node v, 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,

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

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 cost 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 based on changes in the 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 of the path to goal has increased, and we have to repair the entire (reverse) subtree rooted at v
3. $g(v) > lmc(v) \implies v$ is *over-consistent* $\rightarrow$ a shorter path exists: update the parent and cost-to-goal of v, and propagate this cost-change to the in-neighbors of v

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

Definition 3. ***Heuristic value***, $h : V \rightarrow \mathbb{R}^+$ is a non-negative estimate of the cost from start to a node satisfying

1. Non-negativity: $h(v) \geq 0 \; \forall \; v \in V, h(v_{start}) = 0$
2. Triangle inequality: $h(v) \leq h(v') + c(v, v') \quad \forall \; v, v' \in V$
3. Admissibility: $h(v) \leq h^*(v) \; \forall \; v \in V$, 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 minimum 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 priority queue and sorted by ascending value of its key value,

$$\begin{aligned} key(v) = [&min\{g(v), lmc(v)\} + h_{start}(v), \\ &min\{g(v), lmc(v)\}] \end{aligned} \tag{8}$$

where $h_{start}(v)$ is an admissible heuristic for the estimate of cost to go from the start (robot-node) to node v. The key comparisons for priority queue-related operations are based on lexicographic order – the second entry is considered only in case of a tie-breaker among the first entries.

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 sequencibility, using a directed graph constructed backwards from the goal region. The aforementioned incremental search technique is used to calculate the shortest *funnel-path* from robot to the goal, and recalculate it as the environment changes. This ensures there exists a safe, controllable trajectory starting from the robot's current state in an *inlet-region* to an *outlet-region* that is completely contained within the desired goal region.

4 Problem Formulation

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

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{v} \qquad \dot{\boldsymbol{v}}(t) = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{v}, \boldsymbol{u}) \tag{9}$$

where $\boldsymbol{u} \in \mathcal{U} \subseteq \mathbb{R}^m$ is the control-input to the robot system, and $\boldsymbol{f}$ 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, $\boldsymbol{u} : [t_0, t_f] \times \mathcal{S} \rightarrow \mathcal{U}$, which when input to the system (9) starting at $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$, ensures $\boldsymbol{q}(t_f) \in \mathcal{X}_{des}$ 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 5.4. Note that the assumed controller may not be able to handle (dynamic) obstacle spaces, and thus we require a motion planner/replanner (considering this feedback-controller) to safely navigate, avoiding collisions with the obstacles.

Consider a robot that operates in workspace $\mathcal{W}$, 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 4. ***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 $v \in \mathcal{P}^{\mathcal{S}}_{\mathcal{C}}(\mathcal{B}(t_0))$ and $\mathcal{P}^{\mathcal{S}}_{\mathcal{C}}(\mathcal{B}(t_f)) \subseteq \mathcal{X}_w$.

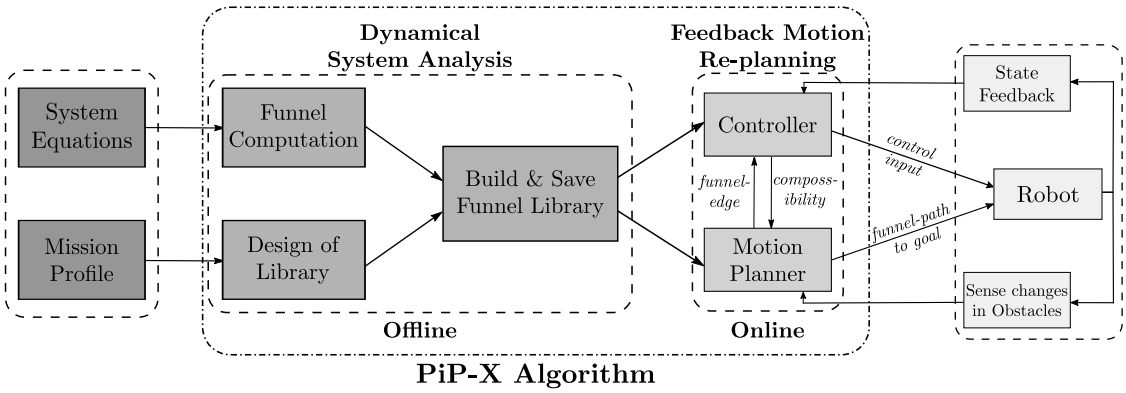

Figure 4. Overview of PiP-X. The algorithm consists of an offline stage of dynamical system analysis, and an online phase of sampling-based graph construction and incremental re-planning in dynamic environments

A funnel-edge is said to be ***valid***, if and only if $\mathcal{P}^{\mathcal{C}}_{\mathcal{W}}(\phi(v,w)) \cap \mathcal{O} = \emptyset$, i.e. it does not overlap with the obstacle set when projected down to the workspace. The cost of a funnel-edge is based on the length of the nominal trajectory within it, $\bar{\boldsymbol{x}}(t)$ projected down to $\mathcal{C}$-space, $\bar{\boldsymbol{q}}(t)$

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

where differential, $ds(t) = \|d\bar{\boldsymbol{q}}(t)\|_W$, $\bar{\boldsymbol{q}}(t)$ satisfies Eq. (9), and $\|\cdot\|_W$ represents a weighted Euclidean norm, with appropriate weights based on the description of $\mathcal{C}$-space. $\mathcal{P}^A_B(.)$ is a projection operator from space A to a lower dimensional space B.

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

Definition 5. ***Funnel-path*** – For a configuration, $q_1 \in \mathcal{C}_{free}$ and a compact set, $\mathcal{X}_2 \subseteq \mathcal{C}_{free}$, *funnel-path*, $\pi(q_1, \mathcal{X}_2)$ is a finite sequence of *valid* funnel-edges with underlying *motion-plan compossibility*, i.e. $\pi(q_1, \mathcal{X}_2) = \{\phi_1, \phi_2, \ldots \phi_n\}$, such that $q_1 \in \mathcal{P}^{\mathcal{S}}_{\mathcal{C}}(\mathcal{B}_{\phi_1}(t_{0_1}))$ and $\mathcal{P}^{\mathcal{S}}_{\mathcal{C}}(\mathcal{B}_{\phi_n}(t_{f_n})) \subseteq \mathcal{X}_2$. The cost of a funnel path is defined in Eq. (11).

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

Problem 1. Online feedback 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, $\boldsymbol{u}(t, \boldsymbol{x}) \in \mathcal{U}$ and keep updating π^* until $q_{robot}(t) \in \mathcal{X}_{goal}$.

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

A dynamic environment has its obstacle space changing randomly with time and/or with robot configuration: $\mathcal{O}(t) = \sigma(t, q_{robot}(t))$, described using a *sensing* function, $\sigma : \{t\} \times \mathbb{R}^d \to \mathcal{O}$. An environment can be modelled as static if σ 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. Feedback motion replanning: Assuming that the robot has the ability to instantaneously sense changes in obstacle space, $\Delta\mathcal{O}(t)$ using $\sigma(t, q_{robot}(t))$; continually recompute $\pi^*(q_{robot}(t), \mathcal{X}_{goal})$ until $q_{robot}(t) \in \mathcal{X}_{goal}$.

To tackle both these above-mentioned problems, we propose **PiP-X**, a feedback motion planning/replanning algorithm that computes *optimal funnel-paths* and updates them *on-the-fly*. The funnel-path is a sequence of maneuvers with formal guarantees of invariance, associated with state-feedback control policies. The continuously updated motion plan with *valid* funnel-edges ensures that the robot trajectory always lies within the funnel-path, avoids dynamic obstacle-spaces, and ultimately reaches the defined goal region.

5 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 feedback motion re-planning algorithm, PiP-X.

5.1 Outline

Our method (see Fig. 4) 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. Though not required by our re-planning algorithm, we *pre-construct* a library of funnels to speedup online computation of optimal *funnel-paths*. Given the mission profile, we design the funnel library considering various combinations of initial and final states, as described in Section 5.5. Design considerations include description of cyclic/non-cyclic coordinates of the state space for funnel compossibility, $\mathcal{C}$-space topology to ensure *probabilistic coverage*, and desired resolution of the motion plans/replans.

We compute funnels (characterised by ellipsoidal level sets) for a given initial state, a compact set of desired final states and a finite-time horizon, using Lyapunov theory

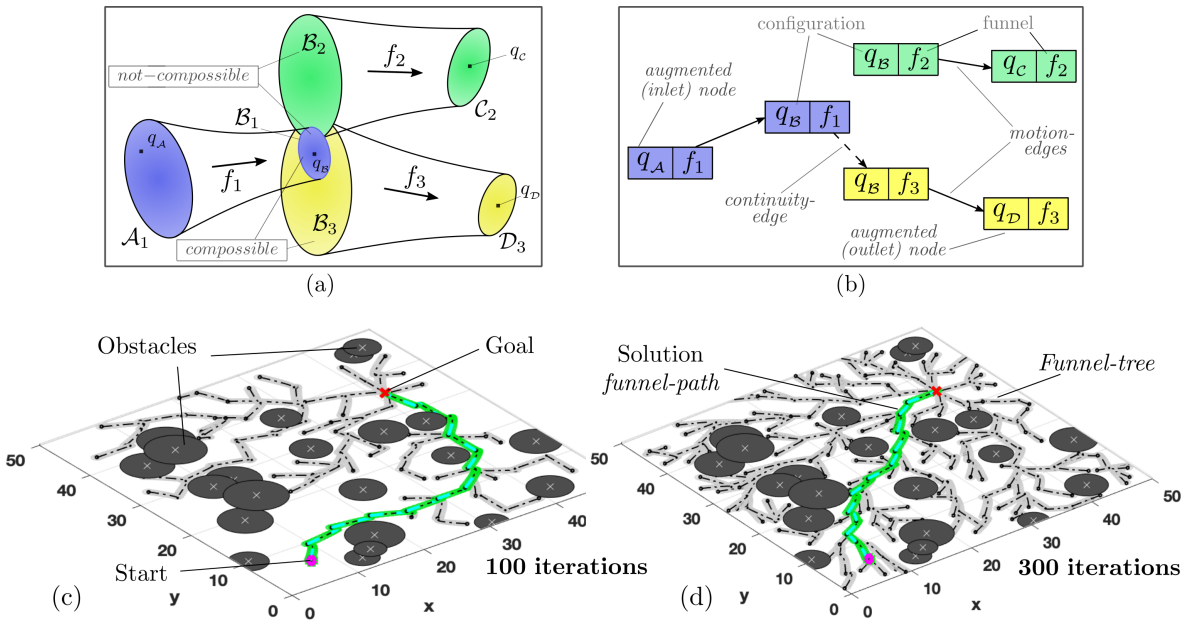

Figure 5. (a) Part of a network of volumetric funnels (b) Corresponding *directed graph data structure* with *augmented* vertices representing the funnel-network. *Motion-edges* (solid) representing traversability are finite-cost, whereas dashed-edges are zero-cost *continuity-edges* encoding compossibility information — whether or not trajectories "flow" into the subsequent funnel. Note that outlet-regions are centered at the configurations. Going from $\mathcal{A}_1$ to $\mathcal{C}_2$ is infeasible, whereas $\mathcal{A}_1 - \mathcal{B}_1 - \mathcal{B}_3 - \mathcal{D}_3$ is a feasible *funnel-path*. (c)-(d) An example in a dynamic forest environment. Funnel-edges are light gray, dynamic obstacles are dark gray, the funnel-path is green, the start is magenta, and goal is red.

(detailed in Section 5.4). The system's equations of motion along with the state-feedback control law are approximated to polynomial dynamics about the nominal trajectory using a Taylor-series expansion of order greater than one. We observe that such a bounded polynomial approximation offers a conservative estimate, i.e. it always underestimates the inner-approximations of backward reachable sets – a condition sufficient for our re-planning algorithm.

During the online phase, we iteratively build a network (or roadmap) of funnels using sampling-based methods: an RRG with *volumetric* funnel-edges. We represent this funnel-connectivity existing in the higher dimensional state space in the form of a *directed graph data structure* (see Fig. 5 and Section 5.2). Graph vertices represent inlet/outlet regions in state space and graph edges denote funnels. Additionally, the edge-set of the graph is *augmented* to include the information of motion-plan compossibility (Definition 2) amongst funnel pairs. On this graph, we use *incremental search* (outlined in Section 3.3) to keep updating the shortest-path subgraph (tree) rooted at goal — translating to optimal funnel-paths (Definition 5) from any configuration in an inlet-region (graph vertex) to the goal region.

Continual *repairs* to the subgraph (tree) ensure that controllable 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(s) closing the feedback loop. Section 5.3 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.

5.2 Notes on graph data structure representing the volumetric funnel-network

We represent the network of volumetric funnels using a directed-graph *augmented* with the additional information of funnel-compossibility (see Fig. 5). Such a graph data structure representation enables the use of incremental graph-replanning methods to quickly rewire funnel-paths (Definition 5) 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 these motion-edges and continuity-edges, $E = E_m \cup E_c$. Vertices of the graph are tuples consisting of a configuration and the respective funnel-edge, $v - \boxed{q \mid f}$. The graph nodes exhibit certain relations amongst the vertex set V, summarised as,

- $\boxed{q \mid f_1}$ & $\boxed{q \mid f_2}$ — Having the same first element (configuration) implies there will be a directed 'zero-cost' *continuity edge*, e_c between the two nodes, if and only if the funnel-pair (f_1, f_2) is *motion-plan compossible* (Definition 2).

- $\boxed{q_a \mid f}$ & $\boxed{q_b \mid f}$ — Represents the case of two vertices sharing the funnel-edge f. By construction, a directed *motion-edge*, e_m exists, having a cost as defined in Eq. (10).
- $\boxed{q_a \mid f_1}$ & $\boxed{q_b \mid f_2}$ — These nodes do not have a common entry. In such cases, there will be *no* edge between the two graph nodes.

It is worth mentioning that continuity-edges (dashed arrows in Fig. 5-b) 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-b, there are two types of graph-vertices: *inlet-nodes* and *outlet-nodes*, $V = \{V_I, V_O\}$. Inlet-nodes like $\boxed{q_{\mathcal{B}} \mid f_3}$ have one, and only one, solid (motion) *out-edge* — might have zero or more dashed (continuity) in-edges. Outlet nodes have only one solid (motion) *in-edge*, and any number of dashed (continuity) out-edges, e.g. $\boxed{q_{\mathcal{B}} \mid f_1}$ and $\boxed{q_{\mathcal{C}} \mid f_2}$. For any node, the set of in-edges consists of either *only* continuity-edges, or *one* motion-edge. The same is true for out-edge set of each node.

With these properties and observations, the authors would like to point out that the *augmented graph* is indeed *bipartite* with (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 region will have *alternating* (solid) motion-edges and (dashed) continuity-edges, similar to the one in Fig. 1-f.

5.3 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 finite sensor horizon. It is efficient because it suffices to alter the motion plans locally near the robot location, saving us the cost of rewiring the bulk of the search tree. The 'inconsistent' nodes are locally repaired in such a way that preserves the global optimality of the path-length defined in Eq. (11).

We incrementally build a *network of funnels* and keep updating the optimal *funnel-path* through this network using the routine: `plan()` (Algorithm 1), outlined on a high-level below. $\mathcal{F}$ denotes the funnel-network, and the graph data structure representing it is denoted by $\mathcal{G}$.

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

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

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

After each update in line 6, all the *promising* nodes in the graph know their best parent that would minimise their one-step lookahead cost (lmc), enabling the planner to backtrack the solution-path using parent pointers. The search is focused towards the robot location using an admissible heuristic, h (Definition 3), thereby enabling quick rewiring of the optimal path whenever the heuristic provides useful information.

Algorithm 2 outlines the pesudo-code of our *online re-planner*, PiP-X. The pre-planning phase (lines 4−7) of the algorithm, continues until the start configuration lies within one of the funnel-inlets and the funnel network 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 one 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 virtue of set-invariance, provided the actual system dynamics closely resembles the nominal model.

The various routines of 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: search graph $\mathcal{G}$, funnel network $\mathcal{F}$, kdTree $\mathcal{T}_k$, priority queue $\mathcal{Q}$ (lines 1−3). The planning parameters are minimum path-resolution ϵ, 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 $\mathfrak{L}$, 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.

5.3.1 *Sampling configurations:* `sampleFree()`

— The configurations q_{rand} are independently and identically (i.i.d.)

[†] Although the shrinking radius of the r-ball given by Karaman and Frazzoli (2011) is found to have practical success historically, recent work by Solovey et al. (2020) states that there is a logical gap in the optimality proof. They propose an amendment by increasing the r-ball to incorporate the additional dimension of time that dictates the samples' ordering: $r' = r_0(log|V|/|V|)^{1/(d+1)}$. Our work uses the tighter bound of $1/d$ due to its empirical success and widespread adoption in literature. The authors wish to state that the (correct) relaxed bound of $1/(d+1)$ can also be used, with not much change to the rest of our approach.

Algorithm 2 PiP-X

Input: $q_{start}, \mathcal{X}_{goal}, \mathcal{C}_{free}, \mathcal{O}, \mathfrak{L}$ ▷ Start, Goal region, Free-space, obstacle-space, Funnel library
Output: $\mathcal{G}, \mathcal{F}$ ▷ search-Graph and Funnel-network

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

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 — to improve coverage in case a replan is eventually required.

5.3.2 ***Exploration:*** The $\mathcal{C}$-space is explored using the extend($\mathcal{G}$, q_{rand}, ϵ) 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 ϵ-distance to obtain the new configuration, q_{new}. If this configuration is already in the funnel-network $\mathcal{F}$, 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, $\mathcal{T}_k$ built using configurations. The radius of the ball decreases at a 'shrinking rate' derived using percolation theory (Penrose et al. 2003).

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

1: $\mathcal{F}' \leftarrow$ findFunnel($q_1, q_2, \mathfrak{L}$)
2: $\mathcal{F} \leftarrow$ shiftFunnel($q_2, \mathcal{F}'$) ▷ shifts & truncates funnel
3: **if** $q_1 \notin \mathcal{F}.ellipsoid$(start) **or** $\neg$collisionFree($\mathcal{F}, \mathcal{O}$) **then**
4: **return** $\emptyset$
5: **return** $\mathcal{F}$

5.3.3 ***Steering (Algorithm 3):*** From the funnel library $\mathfrak{L}$, findFunnel() determines the appropriate funnel that closely drives the system from configuration, q_1 to q_2. We use shiftFunnel() subroutine to shift/rotate the funnel along the cyclic coordinates and time, using appropriate shift-operators $\Psi_c(.)$ and $\Psi_t(.)$, 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 – length of the nominal trajectory within the funnel, computed according to Eq. (10).

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

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

5.3.4 ***Constructing the funnel-network (Algorithm 4):*** We attempt to construct funnels (lines 2–3) between the new configuration, q_{new} and all of the neighbor nodes, n in the neighbor-set, $\mathcal{N}$. Valid (not *in-collision*) funnels flowing into a δ_o-ball around q_{new} are referred to as its $inFunnels$,

$\mathcal{F}_q^- \equiv \mathcal{F}_n^+$ and funnels flowing out of a δ_i-ball around q_{new} as its $outFunnels$, $\mathcal{F}_q^+ \equiv \mathcal{F}_n^-, \forall\, n \in \mathcal{N}$.

getNode($\mathcal{F}$) in lines 5 and 13 determines the node sets: $\mathcal{X}_i$ and $\mathcal{X}_o$, corresponding to the inlet and outlet of the funnel $\mathcal{F}$, projected down to $\mathcal{C}$-space. These nodes are added to the graph vertex-set V, and the directed edge – ($\mathcal{X}_i$, $\mathcal{X}_o$) is added to the set of *motion-edges*, E_m (lines 10 and 18) The newly constructed funnels are added to the funnel-network, $\mathcal{F}$ (line 19).

All the pairs of $inFunnels$ and $outFunnels$ at q_{new} are checked for sequencibility (lines 6–9 and lines 14–17) using compossible() (Algorithm 5). If 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 17). Additionally, the existing $inFunnels$ and $outFunnels$ at neighbor nodes, n are checked for compossibility with the newly constructed funnels to/from n (lines 20–25). If compossible, continuity-edges between outlet-nodes and inlet-nodes at n are also added to E_c (line 25).

Invoking updateVertex() (Algorithm 7) in line 11 ensures propagation of cost-changes and possible rewiring of the shortest-path subgraph (tree) 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 all new 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 compossible($\mathcal{F}_1, \mathcal{F}_2$)

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

5.3.5 ***Funnel-related subroutines:*** The re-planning algorithm makes use of minor subroutines specific to *funnels*. 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 funnel-network, $\mathcal{F}$. 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. (12), with ellipsoidal inlet regions of funnels projected down to $\mathcal{C}$-space. For $M \in S_+^d$, set of $d \times d$ symmetric, positive definite matrices and $q_c \in \mathcal{C}$, Eq. (12) represents the interior of an ellipsoid centred at q_c.

$$(q - q_c)^T M (q - q_c) < 1 \tag{12}$$

compossible() (Algorithm 5) checks whether a pair of funnels is motion-plan compossible as defined in Defintion 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 Eq. (12).

5.3.6 ***Building the shortest-path subgraph (tree) (Algorithm 6):*** Given the search graph, a tree rooted at the goal with minimum cost-to-goal is calculated using techniques outlined in Section 3.3. Invoking computeShortestPathTree() ensures that the robot/start node becomes consistent, and also all the nodes with key value lower than that of the start node (line 2). So in effect, the shortest path to goal from each *promising* node in the search graph, based on cost-to-goal and heuristic values, is continually updated.

Algorithm 6 computeShortestPathTree()

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

Inconsistent nodes are popped out of the priority queue, $\mathcal{Q}$ and repaired (lines 3–13), 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 (Koenig and Likhachev 2002).

Algorithm 7 updateVertex(v)

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

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 3–4) and added to the priority queue with the updated key value only if it is inconsistent (lines 5–7). The priority key 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, $N^+(v)$.

$$parent(v) \leftarrow \underset{v' \in N^+(v)}{\arg\min} \{c(v, v') + g(v')\} \tag{13}$$

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

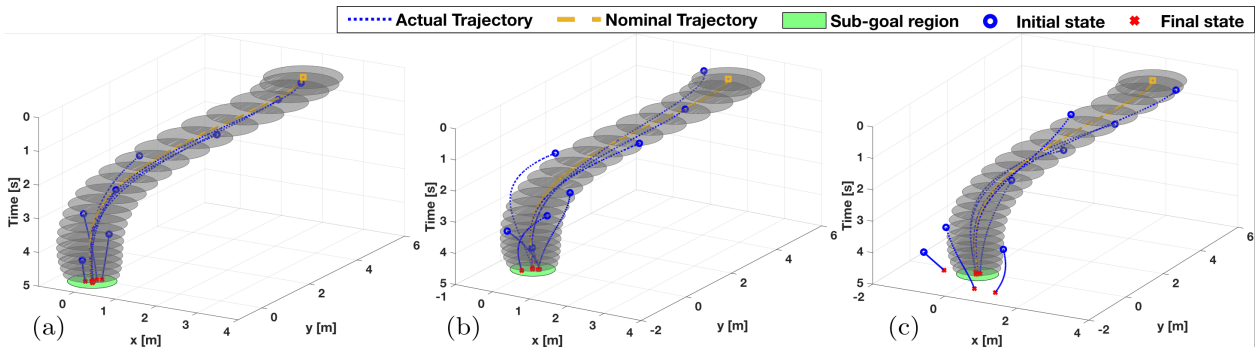

Figure 6. A funnel computed based on Lyapunov level-set theory offers a sufficient but not a necessary condition for invariance. (a) Trajectories starting inside the funnel stay within the funnel; one can not conclusively comment about trajectories starting outside the funnel - it may (b) or may not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

5.3.7 ***Robot motion:*** The various modules of the online phase (lines 9−25) in Algorithm 2 – sensing (lines 10−13), planning (line 14) and robot motion (lines 15−23) have different operating frequencies. This is implemented by running the methods on separate threads at different frequencies. `robotMove`(q_1, q_2) in line 16 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`() (in lines 4 and 7).

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

1: **for all** $e = (v, w) \in E_m$ **do** ▷ ***motion-edges*** set
2: **if** $\neg$`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)

5.3.8 ***Collision checking:*** We exploit the geometric properties of the funnel and environment to come up with a computationally efficient subroutine − `collisionFree`$(\mathcal{F}, \mathcal{O})$ for checking overlaps with obstacle set, $\mathcal{O}$. A funnel $\mathcal{F}$ is said to be *in-collision* if $\mathcal{P}^{\mathcal{S}}_{\mathcal{W}}(\mathcal{F}) \cap \mathcal{O} \neq \emptyset$, where $\mathcal{P}^{\mathcal{S}}_{\mathcal{W}}(.)$ 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 (LaValle 2006). 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. (12). For a general class of obstacles, one can resort to off-the-shelf software such as RoboDK (2017), MPK (Gipson et al. 2001) for collision detection.

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

5.4 Computing regions of finite-time invariance, funnels

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 (Tedrake et al. 2010). Consider a quadratic Lyapunov candidate function centred around a nominal trajectory, $\bar{\boldsymbol{x}}(t)$ defined using a positive definite matrix, $P(t)$.

$$V(t, \boldsymbol{x}) = (\boldsymbol{x} - \bar{\boldsymbol{x}}(t))^T P(t)(\boldsymbol{x} - \bar{\boldsymbol{x}}(t)) \tag{14}$$

For the class of piecewise polynomials $P(t)$, we solve the SoS program using polynomial $\mathcal{S}$-procedure (Parrilo 2003). 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 $(\dot{V}(t, \boldsymbol{x}) - \dot{\rho}(t))$ in the semi-algebraic sets. This maximises the inner-approximation of the verified regions of invariance around the nominal trajectory (Moore et al. 2014).

As noted by Tobenkin et al. (2011), 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 interval. For $M \in S^n_+$, set of $n \times n$ symmetric, positive definite matrices and $\boldsymbol{c} \in \mathbb{R}^n$, Eq. (15) represents an ellipsoid centred around $\boldsymbol{c}$.

$$(\boldsymbol{x} - \boldsymbol{c})^T M(\boldsymbol{x} - \boldsymbol{c}) = 1 \tag{15}$$

The invariant sets, $\mathcal{B}(t)$ in Eq. (2) corresponding to the quadratic Lyapunov function defined in Eq. (14) are the closed set, i.e. interior and boundary of ellipsoids, $\mathcal{E}(t)$ centered around the nominal trajectory, $\bar{\boldsymbol{x}}(t)$.

$$\mathcal{E}(t) = P(t)/\rho(t) \tag{16}$$

The closed-loop system dynamics, as in Eq. (1), is derived using the feedback control policy, $\boldsymbol{u}(t, \boldsymbol{x}, \boldsymbol{x}_d)$ and the system's equations of motion. For a desired final state $\boldsymbol{x}_d$, and finite-time horizon $[t_0, t_f]$, we calculate the system's nominal trajectory, $\bar{\boldsymbol{x}}(t)$ in that time interval by forward integrating the closed-loop dynamics starting from initial state, $\boldsymbol{x}_0$. The dynamics are then approximated to polynomial equations about the finite-time, nominal trajectory using a Taylor-series expansion of order greater than one.

The region of desired final states, referred to as *sub-goal*, $\mathcal{X}_f$ is considered to be an ellipsoid, defined by $\mathcal{E}_f$, centred at the final state, $\boldsymbol{x}_f = \bar{\boldsymbol{x}}(t_f)$. We verify that the *desired* final state, $\boldsymbol{x}_d$ lies within the sub-goal region, $\mathcal{X}_f$ by tuning the feedback-controller, and also the finite-time horizon. 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 using optimisation toolkits (Tobenkin et al. 2011). We ensure the computed funnels can be sequenced together during the online re-planning phase by specifying the sub-goal region $\mathcal{X}_f$, in such a way that *volume* of the inlet, $\mathcal{E}(t_o)$ is larger than that of the outlet, $\mathcal{E}(t_f)$. This can be interpreted as specifying a tighter bound for the set of desired final states, giving lesser margin for steady-state and tracking errors. For a more thorough discussion on this, we refer the readers to Majumdar and Tedrake (2017).

Sample funnels computed for quadrotor dynamics with nominal control (presented in Section 7.1) 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's trajectory stays within the backward reachable set, if it starts within the funnel (see Fig. 6-a).

5.5 Designing the funnel library

Though not required by our algorithm, we make use of a *pre-computed* library of funnels to speedup online computation of optimal funnel-paths. The authors remark that this exercise is necessitated because Lyapunov-based methods to compute funnels are computationally intensive, and impractical for real-time funnel-based replanning. Pre-constructing such a library is optional, and instead, any method capable of calculating backward-reachable invariant sets in real-time can be used.

We consider Lagrangian systems with time-invariant dynamics a.k.a autonomous systems. Considering the state-feedback controller, the closed-loop dynamics in Eq. (1) reduces to $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t))$. Hence, for ease of usage we shift the initial time of funnels to origin, $t_0 = 0$, and the time horizon of a funnel becomes $[0, t_f]$, where $t_f \in R^+$ is finite.

The funnel library $\mathfrak{L}$, consists of a finite number of verified trajectories, encapsulating the information of the certified regions of invariance in the finite-time horizon. Each funnel, $\mathcal{F}_i \in \mathfrak{L}$, is parametrized by the nominal trajectory, $\bar{\boldsymbol{x}}_i(t)$, the ellipsoidal level-sets, $\mathcal{E}_i(t)$ and the final time, t_{f_i}. The trajectories and the ellipsoids are projected from the state space onto $\mathcal{C}$-space using an appropriate projection operator, $\mathcal{P}^{\mathcal{S}}_{\mathcal{C}}(.) : \mathbb{R}^n \to \mathcal{C}$. The resultant projections of ellipsoidal invariant sets also take the form of ellipsoids

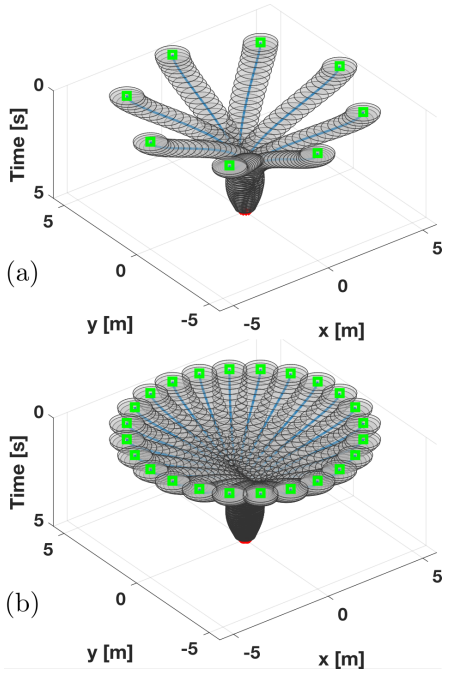

Figure 7. Funnel library (pre-computed) for quadrotor navigation, $\mathfrak{L}$. The initial configurations (green squares) at $t = 0$ lie at an ϵ-distance from $\mathbf{0}$ ($\epsilon = 5$ m). Desired final-states is a compact set centered around origin with radius 0.3 m. Using $\mathfrak{L}_b$ (bottom) will result in a finer *resolution* of motion-plan than $\mathfrak{L}_a$ (top)

when projected onto the d-dimensional Euclidean subspace of robot-configurations (Karl et al. 1994),

$$S(t) = (B^T \mathcal{E}(t)^{-1} B)^{-1} \tag{17}$$

where B is a $n \times d$ matrix, consisting of the basis vectors 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 reduces the amount of computation that is required online, and acts as a bridge between the offline stage of invariant set analysis and the online phase of motion re-planning. Therefore, certain algorithm parameters such as extend-distance, resolution of the planner, range of obstacle sizes, etc. are considered while constructing the library. Meanwhile, the funnel library provides the vital information of compossibility required during the online phase of motion planning/replanning.

Fig. 7 illustrates examples of funnel libraries, with nominal trajectories starting at an ϵ-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/rotated along the cyclic coordinates, and time to ensure motion-plan compossibility (Definition 2). Fig. 7-a depicts a sparser library which would result in a lesser resolution of the motion plan.

It is worth mentioning that the initial states (inlets) of the finite number of funnels in the library projected onto the $\mathcal{C}$-space, along with the appropriate shift operator, $\Psi_c(.)$ about the cyclic coordinates should be able to span the entire $\mathcal{C}$-space. This ensures *probabilistic coverage* (Tedrake et al. 2010) of the sampling-based feedback motion re-planning algorithm.

6 Examples

In this section, we validate our algorithm, PiP-X on two simulation examples: a cart-pole system and a ground car-robot. We empirically evaluate the success rate through repeated Monte Carlo trials in both the example systems using different workspace scenarios. In the next section (Section 7), we conduct more experiments on a simulated 6DOF quadrotor system – discussing performance, execution time and comparative studies of our approach.

6.1 Cart-pole system

The cart-pole system (illustrated in Fig. 8) is a canonical controls example problem because of its highly nonlinear dynamics and underactuation. We consider the scenario of a swing-up maneuver while avoiding moving obstacles (see Fig. 8). The state space is 4D, with system states being cart's position, pole's angular position and their velocities, $\boldsymbol{x} := [x\ v\ \theta\ \omega]^T \in \mathbb{R}^3 \times \mathbb{S}^1$. The control input is force on the cart along the x-direction, $u := F$. Under simplifying assumptions such as the absence of friction, the pole and cart are point masses, and the pendulum-rod is massless; the nonlinear equations of motion are,

$$\begin{bmatrix} \dot{x} \\ \dot{v} \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v \\ \dfrac{F + ml\omega^2 \sin\theta + mg \sin\theta \cos\theta}{M + m\sin^2\theta} \\ \omega \\ \dfrac{-F\cos\theta - ml\omega^2 \sin\theta\cos\theta - (M+m)g\sin\theta}{l(M + m\sin^2\theta)} \end{bmatrix} \tag{18}$$

where M is the cart-mass, m and l are the pole's mass and length, respectively, and g is acceleration due to gravity. We consider the following state constraints: $|x| \leq 10$ m, $\theta \in (-\pi,\ \pi]$ rad, $|v| \leq 1.5$ m/s and $|\omega| \leq \sqrt{4g/l}$ (the maximum possible angular velocity if cart is fixed). The force input, $|F| \leq 10$ N. The system parameters are $M = 1$ kg, $m = 0.1$ kg, $l = 0.5$ m, $g = 9.81$ m/s^2.

In the **offline stage** of our approach, we design a state-feedback controller and pre-compute a library of funnels. The control task is to stabilise the (unstable) 'upright' fixed point ($\boldsymbol{x}_{eq} = [0\ 0\ \pi\ 0]^T$) and create a dampened attractor at the 'hanging' equilibrium, $\boldsymbol{x}_{eq} = [0\ 0\ 0\ 0]^T$. This is achieved through an LQR control law by linearising the system dynamics about the corresponding equilibrium point. Additionally, we make use of an energy-pumping strategy for swing-up control of the pole (Åström and Furuta 2000).

$$u = -k_s \omega \cos\theta \tag{19}$$

Using the closed-loop dynamics, we construct a library of funnels (as outlined in Section 5.4–5.5) starting from initial angular positions θ_0, discretised in steps of $10°$. For θ_0 within $\pm 150°$ (from the vertically down position), we use the LQR controller to regulate to the hanging position, $\theta_f = 0$. Otherwise, the controller stabilises the pole to the upright position, $\theta_f = \pi$. Whenever the pendulum starts at the hanging position (i.e. $\theta_0 = 0$), we use the swing-up controller in Eq. (19) to impart energy and increase the amplitude of oscillations. For computing funnels, we consider a finite-time horizon of $[0, 4]$ s – sufficient for the tuned swing-up controller to reach $\pm 150°$ starting from $0°$, and also for LQR-based regulation to upright/hanging states.

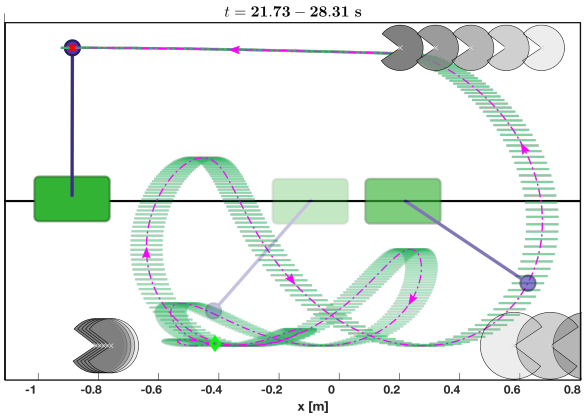

Figure 8. Part of motion plan executed by PiP-X in a *cart-pole* system in a workspace with *moving obstacles*. The funnel-path (green level-sets) is projected down to x-subspace and visualised at the pendulum positions for illustrative purposes. Trajectory of the pole (magenta) starts from hanging position (green) and ends in the upright position (red). The time-history of (circular) obstacles and cart-pole are represented using a color gradient: darker the color, more recent the time instance.

From the system dynamics as in Eq. (18), it is worth noting that the non-cyclic coordinates are v, θ and ω. This implies that for checking inter-compossibility amongst funnels, we can translate the funnels along the cart's position (cyclic coordinate) using a suitable shift operator $\Psi_c(.)$, and analyse the sequencibility along the non-cyclic coordinates, $[v\ \theta\ \omega]$.

We use the pre-computed library of funnels in the **online re-planning stage** (Section 5.3) of optimally sequencing the funnels, and recalculating the funnel-path whenever the obstacle space changes (Algorithm 2). The workspace is boxed by $x \in [-10,\ 10]$ m, and we consider dynamic obstacles that move linearly at either top or bottom levels as illustrated in Fig. 8. The mission objective is to reach a goal region around the upright configuration ($x = 0, \theta = \pi$) at $T_{max} \in [45,\ 50]$ s, starting from hanging configuration ($x = 0, \theta = 0$), while avoiding the moving obstacles.

The obstacles are considered to be circular with radius within $[0.1, 0.2]$ m. The obstacles are capable of changing their velocities repeatedly at random from a range of $[-0.1, 0.1]$ m/s. We instantiate the workspace with 3 obstacles that are randomly at top/bottom levels. It is further assumed that the re-planner is able to sense and estimate the obstacles' position and velocity. We ran **50 trials** with different obstacle space configurations, and found the algorithm to succeed in 46 of the attempts (**92% success rate**). Fig. 8 shows the time-instances of motion plan executed by the cart-pole in one such successful trial.

Implementation details: The motion planning task specifies a final-time constraint T_{max}, so as to require the system to avoid the dynamic obstacles for a considerable amount of time, instead of doing one trivial swing-up to the goal region (upright configuration). This motivates us to motion plan/replan along the time dimension as well, including time in the $\mathcal{C}$-space – configurations $q := [x, \theta, t]$, whereas the workspace comprises of $[x, \theta]$. The 'closeness' in neighbor nodes (for iteratively constructing the funnel network) considers time too, and is determined based on geodesic distance – a weighted metric considering linear/angular position and time (LaValle 2006). It is worth

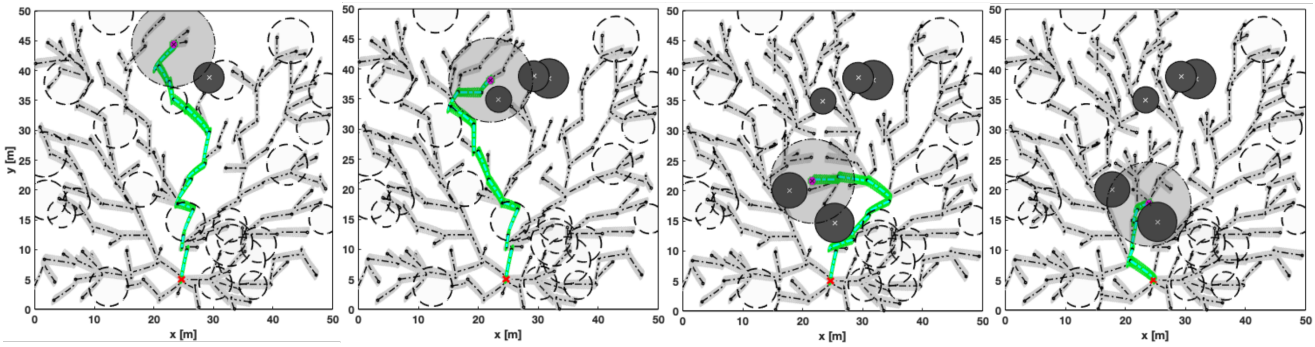

Figure 9. Time instances (progressing left to right) of a *car-robot* with *finite sensing horizon* (shaded circle) running PiP-X in a random forest environment. Perceived obstacles are in dark gray, whereas dashed circles represent yet-to-be sensed obstacles. The minimum-cost tree of funnels and the solution funnel-path are represented in light gray and green, respectively. The robot (magenta) starts at the top, towards the goal location at bottom (red). Note that the funnels are projected down to x-y subspace for visualisation.

mentioning that cost of funnel-edges in Eq. (10) are defined based on a weighted norm considering x and θ. The (dynamic) obstacle space is characterized by *obstacle-cylinders* in space-time, which can alter as obstacles change their speed and direction. Collision-checking is performed by discretizing the funnel-edges along time, and checking for overlaps with the obstacle set. As noted in Section 5.3.8, bounding volumes help prune the collision-checks, making it tractable.

Given the nature of swing-up control law in Eq. (19), starting from the hanging position ($\theta_0 = 0,\ \omega_0 = 0$), we 'warm-start' the system by injecting a small control input, $u_0 = 0.1$ N, to generate initial angular velocity. For computing funnels using SoS programming, the nonlinear closed-loop dynamics are approximated to polynomial equations using a third-order Taylor-series expansion about the nominal trajectory.

6.2 Car robot

The next example we consider is a ground vehicle equipped with velocity and heading control, navigating through an initially-unknown forest environment with finite sensing horizon. The planner's task is to reach a given goal region starting from any initial robot-pose, while avoiding obstacles newly perceived by the robot's sensor. The state space of the system is 5D consisting of translational position, orientation, linear velocity and angular velocity, $\boldsymbol{x} := [x\ y\ \theta\ v\ \omega]^T \in \mathbb{R}^2 \times \mathbb{S}^1 \times \mathbb{R}^2$. The inputs to the system are linear and angular accelerations, $\boldsymbol{u} := [a\ \alpha]^T \in \mathbb{R}^2$. The nonlinear open-loop car dynamics is given by Eq. (20).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \\ a \\ \alpha \end{bmatrix} \tag{20}$$

We consider the state constraints to be: $\theta \in [0,\ 2\pi)$ rad, $v \in [-2,\ 4]$ m/s, and $\omega \in [-1.5,\ 1.5]$ rad/s. The acceleration inputs are bounded as well, $|a| \leq 2$ m/s^2 and $|\alpha| \leq 1.25$ rad/s^2. We derive linear control laws for tracking desired velocities v_{des}, and heading angles θ_{des}.

$$a = k_v(v_{des} - v) \qquad \alpha = k_p(\theta_{des} - \theta) - k_d\omega \tag{21}$$

The control gains, k_p, k_d and k_v are tuned so as to reach the desired states within $t_f = 3$s. We plug in the feedback control laws, Eq. (21) into the system's equations of motion, Eq. (20) to derive the equivalent closed loop dynamics of the system: $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_d)$, and subsequently construct a library of funnels offline.

In order to **pre-compute** the library, we discretize the desired velocity and orientation space. We consider initial/final velocities to be from the set: $\{-2,\ 0,\ 2,\ 4\}$ m/s, and the desired orientations to be in the range of $[0,\ 2\pi)$ rad, in increments of $\pi/6$ (or 30°). We further constrain the initial and final angular velocities, ω to be 0 rad/s. Note that position (x and y) are cyclic states, and one can translate the funnels along them for motion-plan compossibility (Definition 2). This gives us a total of $12 \times 4 = 48$ choices for initial and final states. Hence, the number of funnels in the library is $48 \times 48 = 2304$. We remark that the non-cyclic states are, $\boldsymbol{x}_{nc} = [\theta\ v\ \omega]^T$, and the inter-compossibility of the funnels is checked by projecting them onto the subspace formed by these three states.

For the **online re-planning** stage, we consider a 2D forest environment of dimensions 50m $\times$ 50m with 25 circular obstacles of random sizes in the range of [2, 4] m, spawned at random locations (see Fig. 9). The sensing radius of the car is 7 m, and the obstacle space (as perceived by the robot) keeps changing as and when new obstacles are encountered. We consider the configuration space, $\mathcal{C} \subseteq \mathbb{R}^2 \times \mathbb{S}^1$ to consist of the robot pose – $[x\ y\ \theta]$, while the workspace, $\mathcal{W} \subseteq \mathbb{R}^2$.

Using the pre-constructed funnel library, we ran **50 trials** of PiP-X (Algorithm 2) with different start/goal configurations and random obstacle locations, and found the algorithm to successfully navigate the ground vehicle to the goal region in 48 runs (**96% success rate**). Fig. 9 shows the time-instances of motion plan executed by the ground vehicle in one such successful trial of PiP-X.

Implementation details: The goal region is defined as a δ-ball with radius 0.1 m around a given goal location. The obstacle sizes are inflated to account for the robot-geometry. We sample positions uniformly at random from the workspace, whereas the orientations are sampled from the discrete space of angles (with resolution $\pi/6$). From the motion primitives library, we search for velocities from the discrete-space of $\{-2,\ 0,\ 2,\ 4\}$ m/s that would drive the system between sampled configurations and corresponding

nearest neighbors (similar to shooting methods). Geodesic distance (for $\mathcal{C}$-space exploration and determining nearest neighbors) and funnel-edge cost are a weighted metric of robot-pose – (x, y and θ), accounting for the difference in units (m and rad). As in the previous cart-pole example, the system equations are approximated to polynomial dynamics about the nominal trajectory using a Taylor-series expansion of order three.

7 Validation in a Simulated Quadrotor

In this section, we test our approach on a quadrotor UAV in simulations, flying through an indoor space with dynamic obstacles. We demonstrate the relevance of invariant sets, and empirically validate completeness and correctness of our algorithm. This section also compares our algorithm with RRT^X, an online sampling-based *geometric* re-planner, and reports execution time. We begin this section by describing the dynamics of a quadrotor along with a nominal position controller, and later discuss our experiments with implementation details.

7.1 System Dynamics & Mission Profile

The equations of motion of a quadrotor UAV are derived using Newton-Euler formulation (Garcia et al. 2006). Considering position, $\boldsymbol{\xi} = [x, y, z]^T$ and the attitude, $\boldsymbol{\eta} = [\phi, \theta, \psi]^T$ of the quadcopter defined in an inertial frame, the dynamics can be written as,

$$\begin{aligned} \dot{\boldsymbol{\xi}} &= \boldsymbol{v} \\ m\dot{\boldsymbol{v}} &= -mg\boldsymbol{e_3} + R\boldsymbol{e_3}T \\ \dot{\boldsymbol{\eta}} &= W_\eta \boldsymbol{\omega} \\ J\dot{\boldsymbol{\omega}} &= -\boldsymbol{\omega} \times J\boldsymbol{\omega} - J_r(\boldsymbol{\omega} \times \boldsymbol{e_3})\Omega + \boldsymbol{M} \end{aligned} \tag{22}$$

where, m is the mass, J the inertia matrix, $\boldsymbol{v}$ the linear velocity and $\boldsymbol{\omega}$ is body angular rates. $\boldsymbol{e_3}$ is $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$, and J_r is inertia of the rotor. $\Omega = \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4$ is the net rotor speed. Ω_i denotes the rotational speed of individual rotors.

$R \in \mathcal{SO}(3)$ is the rotation matrix from body frame to the inertial frame. W_η is the transformation matrix for angular velocities in the body frame to inertial frame. For the specific configuration of rotors as in Fig. 10, Thrust, T and Moment, $\boldsymbol{M}$ are defined as,

$$\begin{aligned} T &= k\sum_{i=1}^{4} \Omega_i^2 \\ \boldsymbol{M} &= \begin{bmatrix} M_\phi \\ M_\theta \\ M_\psi \end{bmatrix} = \begin{bmatrix} kl(\Omega_4^2 - \Omega_2^2) \\ kl(\Omega_3^2 - \Omega_1^2) \\ d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{bmatrix} \end{aligned} \tag{23}$$

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

The position **controller architecture** has a cascaded structure, with a fast inner loop stabilising the attitude and a outer loop tracking the position or velocities (Jaffar et al. 2019). 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 do not require large deviations from nominal hover conditions (Luukkonen 2011). The outer loop tracks the desired position setpoints and is achieved using an LQR controller (Bouabdallah et al. 2004). Equivalently, the inputs to the quadrotor position controller are the desired setpoints: $[x_d,\ y_d,\ z_d,\ \psi_d = 0]^T$.

The **mission profile** is to fly at a set altitude, $z_d = h$ with a zero heading-angle, $\psi_d = 0$. Due to the decoupled yaw-dynamics, we consider only the translational position as configurations, $q := \boldsymbol{\xi} \in \mathcal{C} \subseteq \mathbb{R}^3$ whereas the state space of the feedback-controlled quadrotor is 6D, including velocities as well, $\boldsymbol{x} := [\boldsymbol{\xi}\ \boldsymbol{v}]^T \in \mathcal{S} \subseteq \mathbb{R}^3 \times \mathbb{R}^3$. Owing to a reduced 2.5D workspace, the sampling is in $\mathbb{R}^2$, and it suffices to check for possible collisions in a 2D plane between ellipses and obstacles. The start configuration and goal region are defined in the xy-plane. The funnel library is appropriately constructed, see Fig. 7.

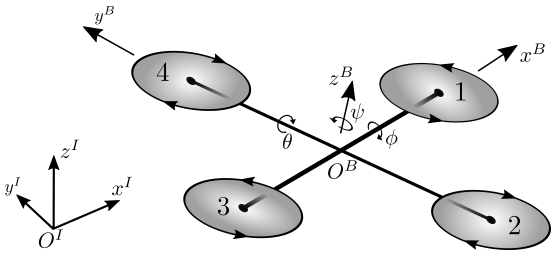

Figure 10. Schematic of a quadrotor with 6DOF (position and attitude) in inertial, I and body-fixed, B frames of reference

7.2 Experimental setup

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

$$\dot{\boldsymbol{\xi}} = \boldsymbol{v} \qquad \dot{\boldsymbol{v}} = \boldsymbol{f}(\boldsymbol{\xi}, \boldsymbol{v}, \boldsymbol{u}) \qquad \boldsymbol{u} = \boldsymbol{\xi}_d \tag{24}$$

The system identification of the translational subsystem, $\boldsymbol{x} = [\boldsymbol{\xi}\ \ \boldsymbol{v}]^T \in \mathcal{S}$, 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 nominal trajectory. An estimate of the required final time of finite horizon, $[0,\ t_f]$ is obtained based on the time taken by the system to reach within the defined goal region of 0.3 m around a desired setpoint. Subsequently, the invariant sets centered around the nominal trajectory are constructed using the methods described in Section 5.4. The Sum-of-Squares optimisation is converted to an SDP by Systems Polynomial Optimisation Toolbox (SPOT) in MATLAB and solved using SeDuMi (Sturm 1999).

The reverse-search algorithm requires backward reachable sets — starting from different initial conditions, the desired setpoint is given as the origin, $\boldsymbol{\xi_d} = \boldsymbol{0}$. The initial xy-positions lie on an ϵ-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

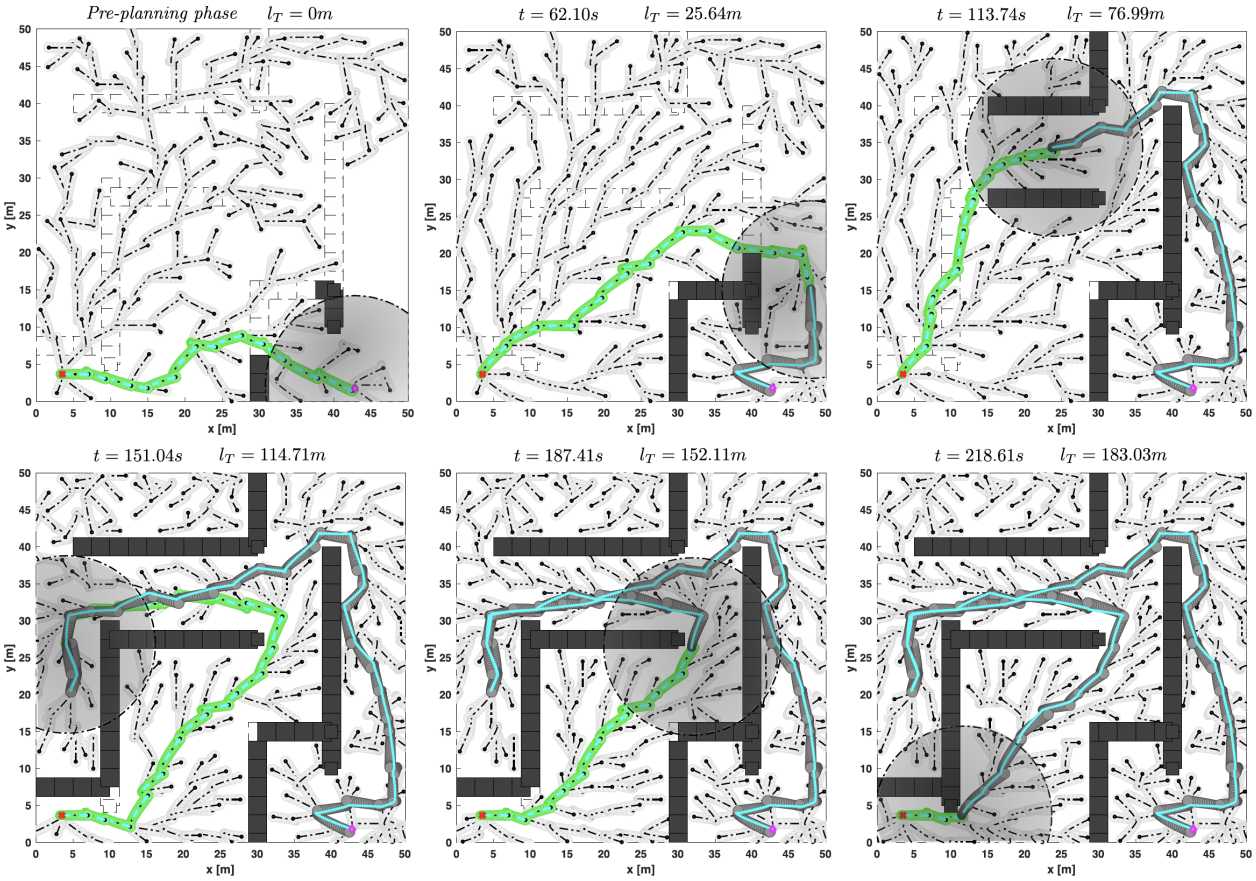

Figure 11. Time instances of motion plan executed by PiP-X on a quadrotor flying with altitude-hold. The start configuration is in the lower-right corner, with the goal location at lower-left corner. The quadrotor senses obstacle-walls (solid rectangles) within sensor-radius (shaded circle), and recomputes 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

starting location. Each funnel in the library is characterised by its key, the nominal trajectory, ellipsoidal invariant sets and corresponding nominal control inputs at each discrete time-step within the finite-time horizon.

In order to verify the motion plan, we develop a higher fidelity model based on the equations of motion in Eq. (22)−(23). In addition to the nominal dynamics, 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.

All the computations are performed on a desktop workstation with a 6-core, 2.6GHz Intel CPU and 32GB RAM. We provide a sample implementation code[‡] of our algorithm, PiP-X (for a quadrotor system) in a maze environment.

7.3 Monte Carlo experiments

A sample run of PiP-X in a user-specified maze environment is shown in Fig. 11. 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 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. (14)−(16). From Fig. 12, 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 subsequent (compossible) *funnel-edges* in the solution funnel-path.

Using Monte Carlo experiments, we test our algorithm in two different kinds of environments – (i) initially unknown and (ii) dynamically changing – maze and random forest, 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.

[‡] https://github.com/khalid2696/pip-x

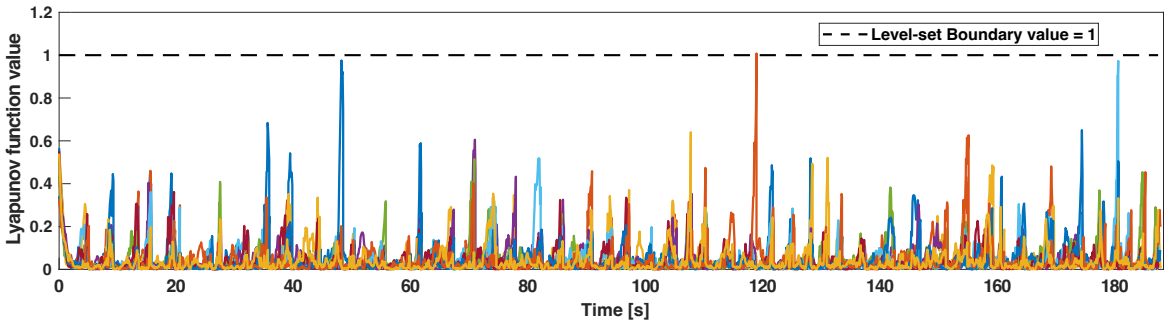

Figure 12. *Normalised* Lyapunov function value of the system's state simulated using the higher-fidelity quadrotor model, with solution funnel-path given by the algorithm across 10 different trials (denoted by different line-colors)

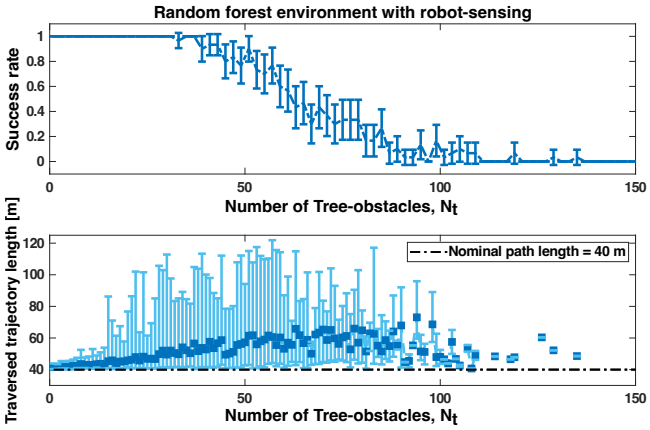

Figure 13. Random forest environment with finite sensing: (a) mean and standard deviation of success rate (b) mean and range of traversed-trajectory length, across 30 trials of each scenario

7.3.1 ***Initially-unknown Maze with limited robot-sensing:*** A maze environment with rectangular walls is designed in a two-dimensional 50m $\times$ 50m workspace, similar to the one in Fig. 11. The robot senses the obstacle-walls within a finite radius of 12 m. We consider 10 different scenarios of start/goal configurations, running 25 trials of each scenario. We observe that our algorithm is *always* capable of computing an initial motion plan, and accordingly replan as new obstacle-walls are perceived, resulting in a 100% success rate across the 10 different scenarios.

7.3.2 ***Initially-unknown random Forest with finite 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 (see Fig. 5-c). The sensing radius of the quadrotor is 12 m. 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 40 m diametrically apart, for uniformity while analysing performance. The goal region is defined to be a δ-ball of radius 0.3 m centered at the goal configuration.

The number of tree-obstacles are varied from $0-150$ in increments of one. 30 different trials are run in each scenario, reporting mean/standard deviation of success rate, and mean/range of traversed-trajectory length (see Fig. 13). The nominal path length — if there are no obstacles, and the robot is holonomic — is 40 m.

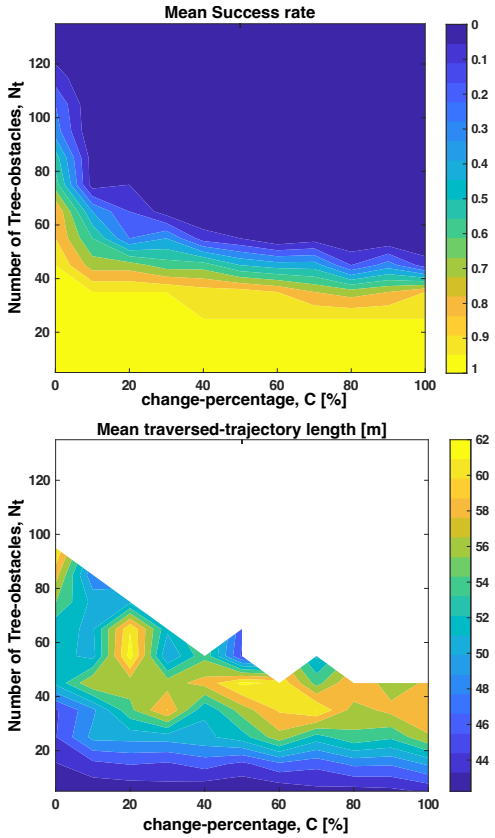

Figure 14. Forest environment with dynamic obstacles: mean of (a) success rate and (b) traversed trajectory length from 25 trials of each scenario

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

7.3.3 ***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 example, a change $C = 20\%$ in a workspace with $N_t = 75$ implies 15 pre-existing trees are removed and 15 new obstacles are added – changing location and size. $C = 0\%$ trivially refers to a static environment.

Taking inputs 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 the form of a contour plot, Fig. 14.

We observe that algorithm failure and the average length of traversed trajectory increase with either increasing number of obstacles, $N_t \geq 45$, or higher level of changes, $C \geq 50$. It completely fails when the environment is densely filled with obstacles or is highly dynamic (upper right triangle of the contour in Fig. 14-a). Note that trajectory length in static environments ($C = 0\%$) with $N_t \geq 95$ is not visualised in Fig. 14-b as they are isolated instances of algorithm successes.

7.4 Discussions

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 path does not 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 it.

7.4.1 ***Computation time analysis:*** The various modules of the online phase (lines 9–25) in Algorithm 2 – sensing, replanning and robot motion have different time complexities and hence, operating frequencies. We study the frequency of the replanner (line 14 of Algorithm 2) because that is the primary scope of this work. In our (possibly unoptimised) implementation in MATLAB, the time taken (on a PC with 6-core, 2.6GHz CPU and 32GB RAM) for each iteration of re-planner in a maze-like environment is in the order of 6.18 ms. This implies the re-planner is capable of running at ∼160 Hz nominally, comparable with existing real-time planners.

The frequency of the online re-planner varies with density of obstacles in the workspace. More obstacle-changes — either due to sensing new obstacles or obstacles moving, appearing/disappearing — tends to require more graph rewiring to perform. To study this, we increase the obstacle density in an initially-unknown 2D forest environment (similar to Fig. 9) and analyse the worst-case re-planning frequency. In the challenging scenario with 50% obstacle-density, the average re-planning frequency (from 100 trials) we encountered is 85.5 Hz (with the least being 43.7 Hz in one trial).

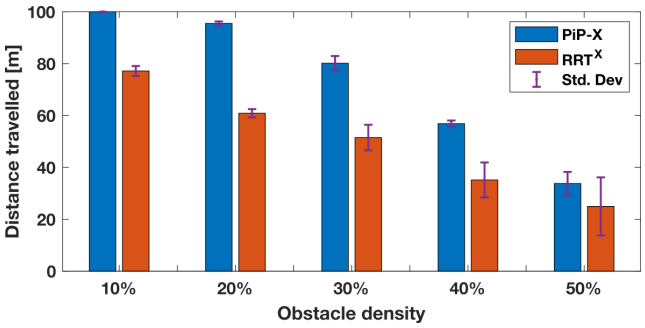

Figure 15. PiP-X v/s RRT^X in a random forest-like environment: Comparing distance travelled by a quadrotor with limited sensing before colliding/freezing. Obstacle density refers to the proportion of obstacle space with respect to the workspace. Mean and standard deviation from 20 trials of each scenario.

We analyse the pre-planning time to iteratively build an RRG of funnels and to keep updating the optimal funnel-path in it (lines 4–7 of Algorithm 2). We report the total average execution time with number of new configurations sampled and added to the funnel-network in Table 1. For context, we also mention the corresponding volume of the funnel-network in proportion to that of the $\mathcal{C}$-space. We do not report the computation time of pre-calculating funnels since it depends on the underlying optimisation methods and toolkits used, and is not the focus of our research.

7.4.2 ***Comparison with RRT^X:*** We compare our approach, PiP-X with RRT^X (Otte and Frazzoli 2016), an online sampling-based *geometric* re-planner in the case of quadrotor flying in an unknown random forest environment (similar to the one in Fig. 9) while perceiving tree-obstacles within its sensor range. We obtain the geometric path from RRT^X in real-time and use the same position tracking controller (as mentioned in Section 7.1) to track the reference trajectory. By varying the obstacle-density (measured as a proportion of the workspace), we analyse how far both the re-planners – PiP-X and RRT^X – are able to fly the quadrotor safely before colliding with the obstacles. The task is to fly a maximum trajectory-length of 100 m to reach the goal from a given start configuration. We run 20 trials (different start/goal configurations) in each environment, characterized by different obstacle densities. The comparison results are illustrated as a bar-plot in Fig. 15.

Additionally, we compare the two algorithms, PiP-X and RRT^X in a **dynamic maze** environment. Similar to the unknown maze environment in Section 7.3.1, we incorporate 'windows' in the obstacle walls that can open (obstacle-deletion) or close (obstacle-addition) at random, emulating a dynamic setup. Change-percentage, C refers to the proportion of windows with respect to the walls, that open and close at a frequency same as that of the robot's sensor. C is varied from 0% (static) to 100% (highly dynamic)

# **configurations sampled**	50	100	150	200	250	300	350	400
% **volume of $\mathcal{C}$-space covered**	19.73%	35.57%	56.09%	72.05%	85.35%	94.99%	98.99%	99.94%
Execution time [s]	0.891	1.625	2.484	4.504	6.950	12.195	32.954	233.207

Table 1. Execution time of our algorithm's pre-planning phase (lines 4–7 of Algorithm 2, PiP-X) – iteratively constructing a funnel-network to probabilistically cover the $\mathcal{C}$-space

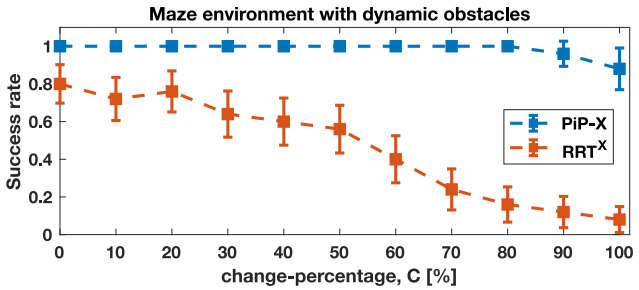

Figure 16. PiP-X v/s RRT^X in a dynamic maze environment: Comparing algorithm success rate. Change percentage refers to the proportion of obstacles that change in location. Mean and standard deviation from 25 trials of each scenario.

in increments of 10. Fig. 16 illustrates the comparison of algorithm success rate between RRT^X and PiP-X in the dynamic maze. Algorithm success is as defined previously — navigating through the maze without colliding with any obstacle walls, and reaching the goal location.

We observe that PiP-X outperforms RRT^X across all scenarios with different obstacle densities (in random forest environment) and change-percentages (in dynamic maze environment). We believe this is because, RRT^X returns paths that are close to obstacles (due to the cost optimality of such paths). Tracking errors by the controller can lead to collisions with obstacles — especially in the dynamic maze environment (see Fig. 16), where the RRT^X planner presumes the quadrotor can fly through the open 'windows' in the maze walls, yet the windows have tight margins of error such that deviations from the planned path are likely to end in collisions. Even if the size of obstacles had been inflated to increase safety, the fact that RRT^X does not consider closed loop dynamics means that there are no guarantees the vehicle would have access to a controller capable of tracking the desired trajectory. Whereas, PiP-X being a feedback motion planner, explicitly considers system dynamics, and controller performance/limitations, and computes kinodynamically feasible plans/replans.

We remark that PiP-X is not suitable in all scenarios and environments. For instance, in the comparative study in random forest environment, the primary reason why PiP-X fails to fly the maximum trajectory-length of 100 m is because of its inability to fit *volumetric* funnel-edges within narrow passages of workspaces densely filled with obstacles.

7.4.3 ***Remarks:*** PiP-X 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 controllable motion plans, implicitly addressing the two-point boundary value problem 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 graph (that represents the *volumetric* funnel-network).

8 Conclusions

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

Funnel-connectivity and its inter-sequencibility is mapped using a *directed-graph data structure* representation, helping us leverage algorithmic 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 cart-pole, car-like robot, and 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 invariant set theory, sampling-based motion planning, and incremental graph-search to create a single framework that enables *feedback motion re-planning* for any general nonlinear robot-system in dynamic workspaces.

Acknowledgements

The authors are grateful to the Robot Locomotion Group at MIT for providing an open-source software distribution for computing funnels. We would like to thank all the anonymous reviewers of this manuscript, as well as the reviewers of the preliminary version of this work that appeared in WAFR '22 for their insightful comments and feedback.

The authors thank Sharan Nayak for all 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.

Funding

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

References

Adiyatov O and Varol HA (2017) A novel RRT*-based algorithm for motion planning in dynamic environments. In: *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE, pp. 1416–1421.

Arslan O, Berntorp K and Tsiotras P (2017) Sampling-based algorithms for optimal motion planning using closed-loop prediction. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4991–4996.

Arslan O and Tsiotras P (2013) Use of relaxation methods in sampling-based algorithms for optimal motion planning. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, pp. 2421–2428.

Arteaga R, Antonio E, Becerra I and Murrieta-Cid R (2021) On the Efficiency of the SST Planner to Find Time Optimal

Trajectories Among Obstacles With a DDR Under Second Order Dynamics. *IEEE Robotics and Automation Letters* 7(2): 674–681.

Åström KJ and Furuta K (2000) Swinging up a pendulum by energy control. *Automatica* 36(2): 287–295.

Bajcsy A, Bansal S, Bronstein E, Tolani V and Tomlin CJ (2019) An efficient reachability-based framework for provably safe autonomous navigation in unknown environments. In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, pp. 1758–1765.

Basescu M and Moore J (2020) Direct NMPC for post-stall motion planning with fixed-wing UAVs. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9592–9598.

Becerra I, Yervilla-Herrera H, Antonio E and Murrieta-Cid R (2021) On the local planners in the RRT* for dynamical systems and their reusability for compound cost functionals. *IEEE Transactions on Robotics* 38(2): 887–905.

Blackmore L, Ono M and Williams BC (2011) Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics* 27(6): 1080–1094.

Borrelli F, Keviczky T and Balas GJ (2004) Collision-free UAV formation flight using decentralized optimization and invariant sets. In: *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, volume 1. IEEE, pp. 1099–1104.

Bouabdallah S, Noth A and Siegwart R (2004) 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)*, volume 3. IEEE, pp. 2451–2456.

Bruce J and Veloso MM (2002) Real-time randomized path planning for robot navigation. In: *Robot soccer world cup*. Springer, pp. 288–295.

Burridge RR, Rizzi AA and Koditschek DE (1999) Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research* 18(6): 534–555.

Connell D and La HM (2017) Dynamic path planning and replanning for mobile robots using RRT. In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, pp. 1429–1434.

Dey D, Liu T, Sofman B and Bagnell J (2012) Efficient optimization of control libraries. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26. pp. 1983–1989.

Ferguson D, Kalra N and Stentz A (2006) Replanning with RRTs. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, pp. 1243–1248.

Frazzoli E, Dahleh MA and Feron E (2005) Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE transactions on robotics* 21(6): 1077–1091.

Gammell JD, Barfoot TD and Srinivasa SS (2020) Batch informed trees (BIT*): Informed asymptotically optimal anytime search. *The International Journal of Robotics Research* 39(5): 543–567.

Garcia PC, Lozano R and Dzul AE (2006) *Modelling and control of mini-flying machines*. Springer Science & Business Media.

Gipson I, Gupta K and Greenspan M (2001) MPK: An open extensible motion planning kernel. *Journal of Robotic Systems* 18(8): 433–443.

Herbert SL, Chen M, Han S, Bansal S, Fisac JF and Tomlin CJ (2017) FaSTrack: A modular framework for fast and guaranteed safe motion planning. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, pp. 1517–1522.

Hsu D, Kindel R, Latombe JC and Rock S (2002) Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research* 21(3): 233–255.

hwan Jeon J, Karaman S and Frazzoli E (2011) 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, pp. 3276–3282.

Jaffar MKM and Otte M (2023) PiP-X: Funnel-based online feedback motion planning/replanning in dynamic environments. In: *International Workshop on the Algorithmic Foundations of Robotics*. Springer, pp. 132–148.

Jaffar MKM, Velmurugan M and Mohan R (2019) A novel guidance algorithm and comparison of nonlinear control strategies applied to an indoor quadrotor. In: *2019 Fifth Indian Control Conference (ICC)*. IEEE, pp. 466–471.

Karaman S and Frazzoli E (2010) Optimal kinodynamic motion planning using incremental sampling-based methods. In: *49th IEEE conference on decision and control (CDC)*. IEEE, pp. 7681–7687.

Karaman S and Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* 30(7): 846–894.

Karl WC, Verghese GC and Willsky AS (1994) Reconstructing ellipsoids from projections. *CVGIP: Graphical Models and Image Processing* 56(2): 124–139.

Kavraki LE, Svestka P, Latombe JC and Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* 12(4): 566–580.

Khalil HK and Grizzle JW (2002) *Nonlinear systems*, volume 3. Prentice hall Upper Saddle River, NJ.

Kleinbort M, Solovey K, Littlefield Z, Bekris KE and Halperin D (2018) Probabilistic completeness of rrt for geometric and kinodynamic planning with forward propagation. *IEEE Robotics and Automation Letters* 4(2): x–xvi.

Koenig S and Likhachev M (2002) D*Lite. *AAAI/IAAI* 15.

Koenig S, Likhachev M and Furcy D (2004) Lifelong planning A*. *Artificial Intelligence* 155(1-2): 93–146.

Kousik S, Vaskov S, Bu F, Johnson-Roberson M and Vasudevan R (2020) Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots. *The International Journal of Robotics Research* 39(12): 1419–1469.

Kuffner JJ and LaValle SM (2000) 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)*, volume 2. IEEE, pp. 995–1001.

Kuwata Y, Teo J, Fiore G, Karaman S, Frazzoli E and How JP (2009) Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on control systems technology* 17(5): 1105–1118.

LaValle SM (2006) *Planning algorithms*. Cambridge university press.

LaValle SM and Kuffner Jr JJ (2001) Randomized kinodynamic planning. *The international journal of robotics research* 20(5): 378–400.

Li Y, Littlefield Z and Bekris KE (2016) Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research* 35(5): 528–564.

Luukkonen T (2011) Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo* 22: 22.

Majumdar A, Ahmadi AA and Tedrake R (2013) Control design along trajectories with sums of squares programming. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, pp. 4054–4061.

Majumdar A and Tedrake R (2013) Robust online motion planning with regions of finite time invariance. In: *Algorithmic foundations of robotics X*. Springer, pp. 543–558.

Majumdar A and Tedrake R (2017) Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research* 36(8): 947–982.

Manchester Z and Kuindersma S (2019) Robust direct trajectory optimization using approximate invariant funnels. *Autonomous Robots* 43(2): 375–387.

Mason M (1985) The mechanics of manipulation. In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2. IEEE, pp. 544–548.

Moore J, Cory R and Tedrake R (2014) Robust post-stall perching with a simple fixed-wing glider using LQR-trees. *Bioinspiration & biomimetics* 9(2): 025013.

Otte M and Frazzoli E (2016) RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *The International Journal of Robotics Research* 35(7): 797–822.

Park C, Pan J and Manocha D (2012) ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In: *Proceedings of the international conference on automated planning and scheduling*, volume 22. pp. 207–215.

Parrilo PA (2003) Semidefinite programming relaxations for semialgebraic problems. *Mathematical programming* 96(2): 293–320.

Penrose M et al. (2003) *Random geometric graphs*, volume 5. Oxford university press.

Ratliff N, Zucker M, Bagnell JA and Srinivasa S (2009) CHOMP: Gradient optimization techniques for efficient motion planning. In: *2009 IEEE international conference on robotics and automation*. IEEE, pp. 489–494.

Ravanbakhsh H, Laine F and Seshia SA (2019) Real-time funnel generation for restricted motion planning. *arXiv preprint arXiv:1911.01532* .

Ravankar A, Ravankar AA, Kobayashi Y, Hoshino Y and Peng CC (2018) Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges. *Sensors* 18(9): 3170.

Reist P, Preiswerk P and Tedrake R (2016) Feedback-motion-planning with simulation-based lqr-trees. *The International Journal of Robotics Research* 35(11): 1393–1416.

Richards A and How JP (2002) Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In: *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 3. IEEE, pp. 1936–1941.

RoboDK (2017) Simulation and OLP for Robots. URL https://robodk.com/.

Schulman J, Duan Y, Ho J, Lee A, Awwal I, Bradlow H, Pan J, Patil S, Goldberg K and Abbeel P (2014) Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research* 33(9): 1251–1270.

Singh S, Landry B, Majumdar A, Slotine JJ and Pavone M (2019) Robust feedback motion planning via contraction theory. *The International Journal of Robotics Research* .

Slotine JJE, Li W et al. (1991) *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ.

Solovey K, Janson L, Schmerling E, Frazzoli E and Pavone M (2020) Revisiting the asymptotic optimality of RRT*. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2189–2195.

Stentz A et al. (1995) The focussed D* algorithm for real-time replanning. In: *IJCAI*, volume 95. pp. 1652–1659.

Sturm JF (1999) Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization methods and software* 11(1-4): 625–653.

Tedrake R, Manchester IR, Tobenkin M and Roberts JW (2010) LQR-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research* 29(8): 1038–1052.

Tobenkin MM, Manchester IR and Tedrake R (2011) Invariant funnels around trajectories using sum-of-squares programming. *IFAC Proceedings Volumes* 44(1): 9218–9223.

Verginis CK, Dimarogonas DV and Kavraki LE (2021) Sampling-based motion planning for uncertain high-dimensional systems via adaptive control. In: *International Workshop on the Algorithmic Foundations of Robotics*. Springer, pp. 159–175.

Verginis CK, Dimarogonas DV and Kavraki LE (2023) KDF: Kinodynamic Motion Planning via Geometric Sampling-Based Algorithms and Funnel Control. *IEEE Transactions on Robotics* 39(2): 978–997.

Webb DJ and Van Den Berg J (2013) Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, pp. 5054–5061.

Zhang X, Liniger A and Borrelli F (2020) Optimization-based collision avoidance. *IEEE Transactions on Control Systems Technology* 29(3): 972–983.

Zucker M, Kuffner J and Branicky M (2007) Multipartite RRTs for rapid replanning in dynamic environments. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, pp. 1603–1609.

Zucker M, Ratliff N, Dragan AD, Pivtoraiko M, Klingensmith M, Dellin CM, Bagnell JA and Srinivasa SS (2013) CHOMP: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research* 32(9-10): 1164–1193.