RICE UNIVERSITY

From High-Level Tasks to Low-Level Motions: Motion Planning for High-Dimensional Nonlinear Hybrid Robotic Systems

by

Erion Plaku

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

Approved, Thesis Committee:

Lydia E. Kavraki, Professor, Chair
Computer Science

Moshe Y. Vardi, Professor
Computer Science

Marcia K. O'Malley, Assistant Professor
Mechanical Engineering & Materials Science

Houston, Texas
July 2008

Abstract

From High-Level Tasks to Low-Level Motions: Motion Planning for High-Dimensional Nonlinear Hybrid Robotic Systems

by

Erion Plaku

A significant challenge of autonomous robotics in transportation, exploration, and search-and-rescue missions lies in the area of motion planning. The overall objective is to enable robots to automatically plan the low-level motions needed to accomplish assigned high-level tasks.

Toward this goal, this thesis proposes a novel multi-layered approach, termed `Sy`nergic `C`ombination of `L`ayers `o`f `P`lanning (`SyCLoP`), that synergically combines high-level discrete planning and low-level motion planning. High-level discrete planning, which draws from research in AI and logic, guides low-level motion planning during the search for a solution. Information gathered during the search is in turn fed back from the low-level to the high-level layer in order to improve the high-level plan in the next iteration. In this way, high-level plans become increasingly useful in guiding the low-level motion planner toward a solution.

This synergic combination of high-level discrete planning and low-level motion planning allows `SyCLoP` to solve motion-planning problems with respect to rich models of the robot and the physical world. This facilitates the design of feedback controllers

that enable the robot to execute in the physical world solutions obtained in simulation.

In particular, `SyCLoP` effectively solves challenging motion-planning problems that incorporate robot dynamics, physics-based simulations, and hybrid systems. Hybrid systems move beyond continuous models by employing discrete logic to instantaneously modify the underlying robot dynamics to respond to mishaps or unanticipated changes in the environment. Experiments in this thesis show that `SyCLoP` obtains significant computational speedup of one to two orders of magnitude when compared to state-of-the-art motion planners.

In addition to planning motions that allow the robot to reach a desired destination while avoiding collisions, `SyCLoP` can take into account high-level tasks specified using the expressiveness of linear temporal logic (LTL). LTL allows for complex specifications, such as sequencing, coverage, and other combinations of temporal objectives.

Going beyond motion planning, `SyCLoP` also provides a useful framework for discovering violations of safety properties in hybrid systems.

Acknowledgments

I thank my advisor, Dr. Lydia Kavraki, for her guidance and support. I am grateful to Dr. Moshe Vardi for the constructive discussions and close collaborations, and to Dr. Marcia O'Malley for her instructive comments. I owe special thanks to my wife, Amarda Shehu, and the rest of my family for unconditional love and support. This thesis is dedicated to the memory of my father, Hekuran Plaku.

This work was supported by NSF (0713623, 0205671, 0308237, CNS-0615328, ITR-0205671), NIH (GM078988), and the Advanced Texas Technology Program (003604-0010-2003). Equipment was supported in part by NSF (EIA-0216467) and the Rice Computational Research Cluster funded by NSF (CNS-0421109 and CNS-0454333) in a partnership between Rice University, AMD and Cray.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of robotics nowadays is marked by an emphasis towards increasing the autonomy of robots in planning and carrying out assigned tasks. The Minerva robotic tour guide [CNN98], the Sony robot dog [FOR07], the Twendy-One robot [ABC07], the robotic vehicles racing in the DARPA Grand Challenge [DAR07], and the IRobot array of domestic and military robots [USN08] are just some examples of robots that exhibit a great degree of autonomy in accomplishing their assigned tasks.

A basic component of autonomy is the ability of the robot to plan the motions needed to accomplish an assigned task. While significant progress has been made, as research in the last forty years has demonstrated, motion planning still constitutes a significant challenge in autonomous robotics:

> Some of the most significant challenges confronting autonomous robotics lie in the area of automatic motion planning. The goal is to be able to specify a task in a high-level language and have the robot automatically compile this specification into a set of low-level motion primitives, or feedback controllers, to accomplish this task [CLH$^+$05].

Due to the complexity of both the robot hardware and the physical world, motion planning generally takes place using simulated and simplified models of the robot

and the environment on which the robot operates. The motion planner employs these models to produce a sequence of motions that in simulation enables the robot to accomplish an assigned task. In order for the robot to execute the simulated motions and accomplish the assigned task in the physical world, feedback controllers are then used to convert the output of the motion planner into low-level commands to the robot hardware.

Moreover, most motion-planning methods to date focus on the simple task of computing a sequence of motions that in simulation allows the robot to move from an initial to a goal destination while avoiding collisions with obstacles. Motion planning is further simplified by computing a sequence of rotations and translations to accomplish this task in a simulated world that models only the geometry of the robot and the obstacles [Lat91, CLH+05, LaV06]. Even with these simplifications, the geometric motion-planning problem is by no means easy, as evidenced by theoretical results that have shown it is PSPACE-complete [Can88a, Can88b]. Despite the hardness theoretical results, great progress has been made in solving challenging geometric motion-planning problems, especially by sampling-based motion planners [KŠLO96, HLM97, LaV98, BMA98, ABD+98a, BOvdS99, LK01, HKLR02, Ist02, SL02, BV02, MRA03, LK04b, MTP+04, JYLS05, BB05, HSAS05, PBC+05, HLK06, KH06, BB07], and many others surveyed in [CLH+05, LaV06].

Motions of a robot in a physical world, however, are governed by the underlying robot dynamics that often impose constraints on velocity, acceleration, and curvature.

Consequently, solution paths obtained in simulation by motion planners that do not take into account robot dynamics but consider only geometric models cannot be easily followed by the robot in the physical world. It is in general difficult and an open problem to design feedback controllers that can convert a geometric solution path into low-level hardware commands that enable the robot to follow the geometric path and thus accomplish the assigned task in the physical world.

This gap between paths produced by geometric motion planning in simulation and the design of feedback controllers that can enable the robot to follow these geometric paths in the physical world underscores the need for incorporating robot dynamics directly into motion planning. Such an approach facilitates the design of feedback controllers, since the solution computed by the motion planner in simulation not only avoids collisions with obstacles, but also respects the robot dynamics.

Motion planning with dynamics, however, poses significant challenges. Modeling the dynamics in addition to the geometry of the robot can considerably increase the dimensionality of the motion-planning problem. Moreover, solutions no longer consist of translations and rotations, but instead of sequences of motions obtained by simulating the robot dynamics. Constraints imposed by the robot dynamics add to the difficulty of finding sequences of motions that allow the robot to reach the goal while avoiding collisions with obstacles.

Current approaches to motion planning with dynamics are usually based on simple adaptations of popular geometric motion planners. Geometric motion planners,

however, are designed to take advantage of the assumption that the possible motions of the robot are purely geometric, i.e., translations and rotations. Such assumption does not hold in the case of motion planning with dynamics, since constraints imposed by the dynamics limit the possible motions of the robot. Research has shown that approaches based on adaptations of geometric motion planners are generally ineffective in solving challenging motion-planning problems with dynamics [LK04a,LK05,BK07, BTK07a,BTK07b,PKV07a,PKV07b,PKV07c,PKV08a,PKV08b,PKV08c,PKV08d].

These limitations become even more pronounced when considering richer models, such as physics-based simulations. Physics-based simulations add an increased level of realism by modeling not only the dynamics and geometry of the robot, but also friction, gravity, and other interactions of the robot with the environment.

Moreover, many robots used in navigation and exploration of unknown and possible hazardous environments can quickly modify the underlying dynamics to respond to mishaps or unanticipated changes in the environment. Such behavior is often modeled by hybrid systems, which go beyond continuous models by employing discrete logic to instantaneously switch to a different operating mode.

Incorporating richer models, such as robot dynamics, physics-based simulations, and hybrid systems, directly into motion planning is crucial, as it facilitates the design of feedback controllers that enable the robot to execute in the physical world the solutions obtained in simulation. This approach adds significant computational challenges to current motion planners, rendering them practically ineffective.

Novel approaches are needed to significantly reduce the computational cost associated with incorporating richer models into motion planning. This has the potential to enable robots employed in service, search-and-rescue missions, and exploration to autonomously plan low-level motions needed to accomplish assigned tasks.

1.1 Contributions

To effectively incorporate rich models, such as robot dynamics, physics-based simulations, and hybrid systems, directly into motion planning, this thesis proposes a novel multi-layered approach, termed `Sy`nergic `C`ombination of `L`ayers `o`f `P`lanning (`SyCLoP`), that seamlessly combines motion planning at different levels of modeling complexity. Fig. 1.1 provides an illustration.

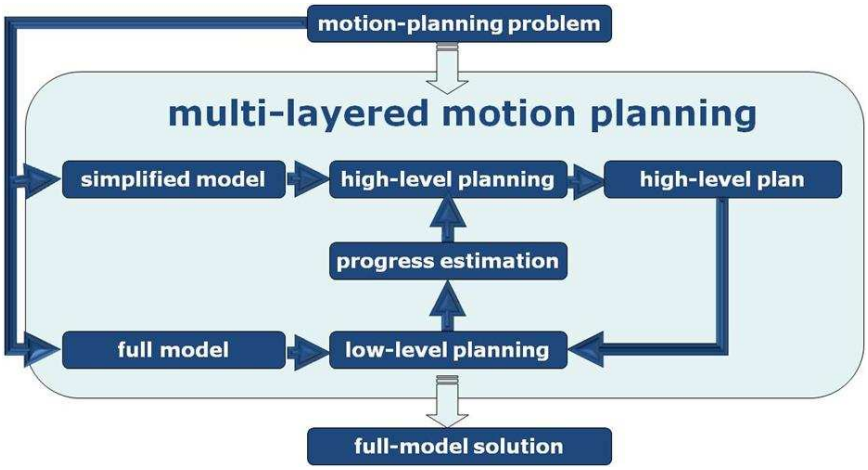


Fig. 1.1: Proposed multi-layered approach, `SyCLoP`, seamlessly combines high-level discrete planning and low-level motion planning.

In the first layer, motion planning takes place in a simplified high-level and discrete model. In the second layer, motion planning is based on the full low-level model of

the robot and the physical world. The high-level discrete planning in the first layer, which draws from research in AI and logic, guides the low-level motion planning in the second layer during the search for a solution.

A distinctive feature and a crucial property of `SyCLoP` is that high-level discrete planning and low-level motion planning are not independent of each-other but in fact work in tandem, as illustrated in Fig. 1.1. At each iteration, high-level discrete planning provides a high-level plan that constitutes a solution to the motion-planning problem under the simplified discrete model. Low-level motion planning attempts in turn to guide the search for a solution under the full model so that it closely follows the current high-level plan. Information gathered during the search, such as the progress made in following the current high-level plan, is fed back from the low-level to the high-level layer in order to improve the high-level plan computed in the next iteration. In this way, high-level plans become increasingly useful in guiding the low-level motion planner toward a solution.

This symbiotic combination of high-level discrete planning and low-level motion planning in `SyCLoP`, as demonstrated in this thesis, has several advantages:

(i) It obtains solutions to the motion-planning problem with respect to the full model of the robot and the physical world in which the robot operates [PKV07a, PKV07b, PKV07c, PKV08a, PKV08b, PKV08c, PKV08d]. This facilitates the design of feedback controllers that enable the robot to follow in the physical world solutions obtained in simulation. (Chapter 4)

(ii) It reduces by one to two orders of magnitude the computational cost in solving challenging problems when compared to current state-of-the-art motion planners [PKV07a, PKV07b, PKV08a, PKV08b, PKV08c, PKV08d]. (Chapters 4–8)

(iii) It incorporates robot dynamics and even physics-based simulations, which increase the realism by modeling friction, gravity, and other interactions of the robot with the environment [PKV07a, PKV08b, PKV08c, PKV08d]. (Chapter 5)

(iv) It is particularly well-suited for hybrid systems, which move beyond continuous models by employing discrete logic to instantaneously modify the underlying robot dynamics to respond to mishaps or unanticipated changes in the environment [PKV07b, PKV07c, PKV08a]. Hybrid systems are often part of sophisticated embedded controllers used in robots exploring unknown and possibly hazardous environments. (Chapter 6)

(v) It incorporates richer tasks expressed in Linear Temporal Logic (LTL) in addition to enabling the robot to move from an initial to a goal placement while avoiding collisions with obstacles [PKV08c]. LTL allows for complex specifications, such as sequencing, coverage, and other combinations of temporal goals, such as "after inspecting a contaminated area A, visit a decontamination station B, before returning to any of the base stations C or D." (Chapter 7)

(vi) Going beyond traditional motion planning, it provides a useful framework for discovering violations of safety properties in hybrid systems [PKV07b, PKV08a,

PKV08c]. Safety properties assert that nothing "bad" happens. For instance, when the hybrid system models air-traffic control, safety properties assert that planes will not come too close to one another. (Chapter 8)

1.2 Organization

Related work is described in Chapter 2. The motion-planning problem, common components used in sampling-based approaches, and a basic search framework for solving motion-planning problems are described in Chapter 3. The proposed multi-layered approach, `SyCLoP`, is described in Chapter 4. Applications of `SyCLoP` to motion-planning problems that incorporate dynamics are presented in Chapter 5. Chapter 5 also includes experiments with several second-order models of robotic vehicles and physics-based simulators. Chapter 6 describes applications of `SyCLoP` to motion planning for hybrid systems, including experiments on a scalable navigation benchmark. Chapter 7 describes applications of `SyCLoP` to incorporate tasks expressed by LTL. Chapter 8 focuses on applications of `SyCLoP` for the falsification of safety properties in hybrid systems. Chapter 8 presents experiments based on falsification of safety properties for an aircraft collision-avoidance protocol in the context of air-traffic management. The thesis concludes in Chapter 9 with a discussion.

1.3 Notation

The section includes a summary of common notation used in this thesis.

Name	Short Description	Defined in
$\top$	boolean value: true	
$\bot$	boolean value: false	
$\circ$	trajectory concatenation	Section 3.2.1
Γ	set of all trajectories	Section 3.1.4
γ	trajectory	Section 3.1.4
γ_{valid}	valid part of trajectory	Section 3.2.3
τ	propositional map	Section 7.2
$\mathcal{A}$	NFA	Definition 7.2.3
Cov	coverage estimate	Section 3.2.5
$\mathcal{D}$	discrete model	Section 4.2
E	discrete transitions in hybrid systems	Definition 6.2.1
f	dynamics flow function	Section 5.1.2
Goal	motion-planning goal function	Section 3.1.3
$G_{\mathcal{R}}$	graph of discrete model	Section 4.2
Guard	guards in hybrid systems	Definition 6.2.1
$\mathcal{H}$	hybrid automaton	Definition 6.2.1
Jump	jumps in hybrid systems	Definition 6.2.1
$\mathcal{P}$	motion-planning problem	Definition 3.1.1
Proj	state-space projection function	Section 3.2.4

continued in next page...

Name	Short Description	Defined in
$\mathcal{Q}$	discrete space in hybrid systems	Definition 6.2.1
$\mathcal{R}$	high-level regions	Section 4.2
$\mathcal{R}_{\text{Goal}}$	goal high-level regions	Section 4.2
$\mathcal{R}_{\text{init}}$	init high-level regions	Section 4.2
$\mathcal{S}$	state space	Section 3.1.1
SampleTraj	trajectory sampling	Section 3.2.2
s_{init}	initial state	Definition 3.1.1
States	associate tree states to region	Section 4.4.2
$\mathcal{T}$	tree data structure	Section 3.3
Traj	tree-trajectory function	Section 3.3
$\mathcal{U}$	control space	Section 5.1.1
Valid	checks if a state is valid	Section 3.1.2
ValidTraj	trajectory validation	Section 3.2.3
$\mathcal{W}$	workspace	Section 5.2.1
$\mathcal{X}$	continuous space in hybrid systems	Definition 6.2.1

Chapter 2

Related Work: Sampling-based Motion Planning

This chapter provides a summary of related work on sampling-based motion planning, which has shown great promise in solving challenging motion-planning problems. The low-level motion planning layer of the proposed multi-layered approach, `SyCLoP`, is also sampling based and draws significantly from progress made in recent years in sampling-based motion-planning research. The chapter starts with an informal definition of the motion-planning problem and a description of early approaches to motion planning. The chapter then focuses on recent approaches that incorporate rich models of the robot and the physical world directly into motion planning in order to facilitate the execution in the physical world of plans produced in simulation.

2.1 Motion Planning with Geometric Models: Generalized Mover's Problem

Stated in its simplest form, the motion-planning problem involves planning a sequence of motions that take a robot from an initial configuration to a final configuration, while avoiding collisions with obstacles in the environment. The robot may be comprised of several rigid objects either moving independently or attached to one another through joints, hinges, and links. A configuration refers to a spatial

arrangement of the robot, and the set of all configurations is referred to as the configuration space. The environment can be a two-dimensional or three-dimensional world, referred to as the workspace, containing obstacles that the robot needs to avoid.

Early on it was shown that the generalized mover's problem, where the robot is comprised of several rigid objects moving independently or connected through joints, was PSPACE-hard [Rei79]. Additional study on exact motion-planning methods for the generalized mover's problem led Schwartz and Sharir [SS88] to an algorithm that was doubly exponential in the degrees of freedom of the robot (subsequent work in real algebraic geometry rendered the algorithm singly exponential [BPR03]). This was followed by Canny's algorithm, which introduced the notion of a roadmap, i.e., a network of 1-dimensional curves that capture the connectivity of the configuration space, and showed that the general mover's problem is PSPACE-complete [Can88a, Can88b]. The algorithms, however, are mainly of theoretical interest due to the prohibitive complexity and difficulty of implementation.

The hardness theoretical results on the generalized mover's problem motivated the development of alternative approaches that do not rely on an explicit computation of the configuration space, but rely instead on an efficient sampling of this space. Several distinct formulations of the sampling-based approach have emerged. One of the first sampling-based planners, Randomized Path Planning (`RPP`) [BL91], utilized a potential field to attract the robot toward the goal, while pushing the robot away from obstacles. When the robot would get stuck in a local minimum of the potential

field, RPP relied on sampling of the configuration space to generate random motions for escaping the local minima. The "Ariadne's Clew" [BMA95,BMA98] used sampling of the configuration space and genetic optimization to guide the exploration of the configuration space toward the goal.

2.1.1 Roadmap Methods

The Probabilistic RoadMap (PRM) [KŠLO96] was the first planner that demonstrated the tremendous potential of sampling-based methods. PRM not only completely decoupled collision checking and planning, but also used sampling in innovative ways that resulted in performance gains that had not been observed earlier. PRM creates a roadmap by first sampling the configuration space. Each sample corresponds to a placement of the robot in the workspace. If the placement does not result in a collision, then the sample is considered valid and it is added to the roadmap. During a second step, neighboring roadmap samples are connected via simple paths that avoid collisions with obstacles. A motion-planning problem is then solved by first connecting the initial and goal configurations to the roadmap, then using graph search on the roadmap to find a path between the initial and goal configurations.

A critical aspect of PRM is the sampling strategy, since PRM relies on sampling to capture the connectivity of the free configuration space. The original PRM implementation [KŠLO96] employed uniform random sampling, which is easy to implement and has been shown to work well in a variety of different problems. It has also been observed, however, that problems where the solution path must go through narrow pas-

sages are particularly challenging, since the probability of generating samples inside narrow passages is low due to these passages' small volume [KŠLO96, HKL+98]. Several sampling strategies were developed to improve sampling inside narrow passages by sampling more around disconnected components of the roadmap [KŠLO96, Kav95], near obstacles [ABD+98a, HKL+98, BOvdS99], on or near the medial axis [GHK99, WAS99, HK00], or using machine learning and workspace information to sample more inside narrow passages [MTP+04, BB05, HSAS05, SHJ+05, HLK06, KH06, ZKB08], and many other strategies surveyed in [CLH+05, LaV06].

2.1.2 Tree Methods

An alternative to roadmap-based approaches is to explore the configuration space by incrementally extending a tree from the initial configuration toward the goal configuration. While a roadmap-based approach attempts to capture the connectivity of the free configuration space so that multiple queries can be solved quickly, the objective of a tree-based approach is to quickly extend the tree toward the goal to solve the one query under consideration. Tree-based approaches were popularized by sampling-based motion planners such as Rapidly-exploring Random Tree (`RRT`) [LaV98, LK01] and Expansive Space Tree (`EST`) [HLM97, HKLR02], which successfully solved challenging motion-planning problems. A vertex in the tree corresponds to a valid sample, while an edge from a sample s' to s'' indicates a valid path connecting s' to s''.

An `RRT` "pulls" the tree toward the unexplored parts of the configuration space by extending the tree toward random samples. At each iteration, a sample s_{rand} is

generated according to some sampling strategy. The closest sample in the tree, s_{near}, is then computed according to a distance metric that defines closeness. A branch in the tree is created by extending a path from s_{near} toward s_{rand}. As in the case of `PRM`, the sampling strategy plays a critical role in the ability of `RRT` to rapidly extend the tree toward the goal configuration.

An `EST` "pushes" the tree to unexplored parts of the configuration space by sampling points away from densely sampled areas. For each sample s in the tree, `EST` maintains a density estimate as a weight $w(s)$, which is usually measured as the number of outgoing edges or the number of neighboring samples. At each iteration, a sample s is selected from the tree with probability inversely proportional to the density estimate $w(s)$, and a branch is created by extending a random path from s.

As in the case of roadmap approaches, the sampling strategy plays a critical role in tree-based approaches. In addition to `RRT` and `EST`, in order to improve the sampling strategy so that the tree quickly extends toward the goal, researchers have proposed numerous methods, such as [SL02, BV02, LL03, JYLS05, LK04a, LK05, BB07, BK07, PKV07a, PKV08b, ZM08] and many others surveyed in [CLH+05, LaV06].

2.1.3 Sampling-based Roadmap of Trees

It is also possible to combine roadmap and tree approaches. Drawing from the success of multi-tree searches in discrete spaces in AI, the Sampling-based Roadmap of Trees (`SRT`) [PBC+05] searches a high-dimensional configuration space by creating a roadmap of trees that integrates the global sampling properties of roadmap-based

planners, such as PRM [KŠLO96], with the local sampling properties of tree-based planners, such as RRT [LaV98,LK01] and EST [HLM97,HKLR02]. As in PRM, SRT constructs a roadmap aimed at capturing the connectivity of the free configuration space. The nodes of the roadmap, however, are not single configurations but trees, which are grown by using tree-based motion planners. The edges of the roadmap correspond to connections between trees, which are also computed by sampling-based tree planners. SRT is shown to be significantly faster and more robust than the roadmap- and the tree-based planners it combines. The multi-tree search in SRT also provides a natural framework for a large-scale distribution based on asynchronous communication, yielding near linear speedup on hundreds of processors [PK05, PK07a].

2.2 Motion Planning with Rich Models: Toward Realistic Applications

The generalized mover's problem considers the motion-planning problem from a purely geometric perspective that ignores the underlying robot dynamics. Motions of a robot in the physical world, however, are governed by dynamics that often impose constraints on the velocity, acceleration, and curvature. As a result, solution paths obtained by motion-planning methods that solve the generalized mover's problem may not be easily executed by the robot in the physical world.

The execution of a solution path obtained in simulation requires the design of feedback controllers that can convert the simulated motions into low-level hardware

commands. The design of feedback controllers is a laborious and challenging task, since it depends on the robot dynamics and the interaction of the robot with the environment. While feedback controllers have been designed that can enable robots with essentially linear dynamics to follow geometric paths, the case of robots with nontrivial dynamics remains open to research [CLH+05, LaV06].

This challenge in designing feedback controllers that can enable complex robots to follow simple geometric paths underscores the need for incorporating robot dynamics directly into motion planning, so that the produced motions obey the physical constraints of the robot. The configuration space is augmented with new parameters necessary to express the robot dynamics. Motion planning then takes place in this augmented configuration space, which is referred to as the state space. Solutions obtained by motion-planning methods that respect the underlying robot dynamics are referred to as trajectories.

Some progress has been made in this direction by adapting popular geometric motion planners, such as `RRT` [LaV98, LK01] and `EST` [HLM97, HKLR02]. To incorporate robot dynamics into motion planning, tree-based approaches, such as `RRT` and `EST`, extend the search tree with trajectories that respect the underlying robot dynamics. Such trajectories are generally computed by propagating the robot dynamics forward in time (see [CLH+05, LaV06], and Chapter 5).

The Path-Directed Subdivision Tree (`PDST`) [LK04a, LK05, Lad06] motion planner, takes this idea a step further and proposes the integration of motion planning with

physics-based simulations that model not only the dynamics of the robot, but also friction, gravity, and other interactions between the robot and the environment in which the robot operates. Motions produced by PDST in simulation have also been executed without much error by modular-chain robots in the physical world [SKYK08]. The work in [GRS+07] also uses physics-based simulations of articulated chains in combination with sampling-based motion planners to effectively solve high-dimensional motion-planning problems for articulated-chain robots.

The work in [BK07] builds upon PDST [LK04a, LK05, Lad06] to incorporate safety constraints in the presence of moving obstacles directly into motion planning. The work is further extended in [BTK07a, BTK07b] to allow for safe replanning not only for one robot, but for a group of robotic vehicles with second-order dynamics.

Sampling-based motion planners have also been adapted to solve motion-planning problems involving flexible or deformable objects [HKW98, GHK99, AOLK00, GLM05, Mol06, SI07], humanoid robots [KKN+02], modular robots [YSS+07, VKR08, SKYK08], and many others surveyed in [CLH+05, LaV06].

Most motion planners to date, however, focus on robots whose underlying dynamics are continuous. Many robots expected to perform complex tasks combine discrete and continuous dynamics. These hybrid systems, designed to explore unknown, dynamic, or possibly hazardous environments, can quickly modify their continuous dynamics to respond to mishaps or unanticipated changes in the environment. Such responses are often realized by employing discrete logic to instantaneously switch be-

tween different operating modes. This combination of discrete logic and continuous dynamics in hybrid systems poses a significant challenge for current motion planners. Some sampling-based motion planners [KEK05, BF04, ND07, EKK04] based on `RRT` [LaV98, LK01] have been adapted to address motion planning for hybrid systems with few modes. The applicability of `RRT`-based motion planners to hybrid systems with a large number of modes remains challenging, since computational efficiency significantly deteriorates as the number of modes increases.

Motion planning with rich models of the robot and the physical world poses significant computational challenges that dramatically increase the computational cost of current motion-planning methods. As discussed in the introduction in Chapter 1, this thesis proposes a novel multi-layered approach, `SyCLoP`, that seamlessly combines motion planning at different levels of modeling complexity. As shown in the rest of this thesis, a significant advantage of `SyCLoP` is that it significantly reduces the computational cost to solve challenging motion-planning problems for robots with dynamics, physics-based simulations, and hybrid systems by one to two orders of magnitude when compared to state-of-the-art motion planners.

Chapter 3

Preliminaries

This chapter defines the motion-planning problem and describes several components used by sampling-based motion planners, including the multi-layered approach, `SyCLoP`, developed in this thesis. The chapter concludes with a description of the underlying tree-search framework commonly used in sampling-based motion planning.

3.1 Motion-Planning Problem

The objective of motion planning is to compute a trajectory that enables the robot to accomplish the assigned task while satisfying constraints such as collision avoidance, and velocity and acceleration bounds along the trajectory. Fig. 3.1 provides an illustration. A formal definition follows.

Definition 3.1.1. (Motion-Planning Problem). *A motion-planning problem is a tuple* $\mathcal{P} = (\mathcal{S}, \textsc{Valid}, s_{\text{init}}, \textsc{Goal})$*, where*

- $\mathcal{S}$ *is a state space consisting of a finite set of variables that completely describe the state of the system (see Section 3.1.1);*
- $\textsc{Valid} : \mathcal{S} \rightarrow \{\top, \bot\}$ *is a state-constraint function, i.e.,* $\textsc{Valid}(s) = \top$ *iff* $s \in \mathcal{S}$ *satisfies the state constraints (see Section 3.1.2);*

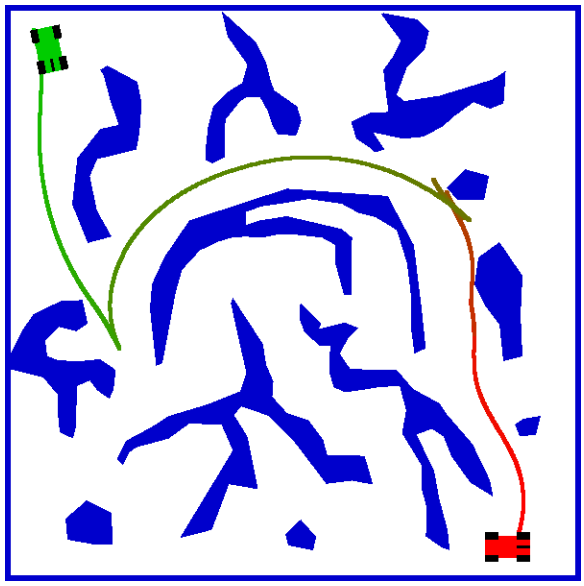


Fig. 3.1: In this motion-planning problem the objective is to find a trajectory that allows the robotic car to reach the goal position starting while avoiding collisions with the obstacles.

- $s_{\text{init}} \in \mathcal{S}$ *is an initial state;*

- $\textsc{Goal} : \mathcal{S} \to \{\top, \bot\}$ *is a goal function, i.e.,* $\textsc{Goal}(s) = \top$ *iff* $s \in \mathcal{S}$ *satisfies the motion-planning goal (see Section 3.1.3);*

A solution to the motion-planning problem $\mathcal{P}$ *is a valid trajectory* $\gamma : [0, T] \to \mathcal{S}$ *(see Section 3.1.4) that starts at* s_{init} *and satisfies the motion-planning goal, i.e.,*

$$\gamma(0) = s_{\text{init}}; \quad \textsc{Goal}(\gamma(T)) = \top; \quad \text{and} \quad \forall t \in [0, T] : \textsc{Valid}(\gamma(t)) = \top.$$

3.1.1 State Space

A state consists of a collection of variable values that completely describe the system at a given instance. The set of all states constitutes the state space, which is denoted by $\mathcal{S}$ and defined as

$$\mathcal{S} = \{s : s \text{ is a state}\}.$$

When a system is composed of multiple robots, then the state of the system is obtained by concatenating the states of each robot in the system. The state space $\mathcal{S}$ of a multi-robot system is then obtained as the Cartesian product of the state spaces $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$ of each robot, i.e.,

$$\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n.$$

The following example illustrates common representations of the state spaces for several simple robots. More complex examples can be found in Chapters 5 and 6.

Example 3.1.1. (State Space of a 2D Point Robot). *Each state can be fully described by the point's position (x, y). Since the state describes only the robot configuration, the state space is commonly referred to as the configuration space.*

(State Space of a 2D Polygonal Robot). *The description of a state of a 2D polygonal robot requires the specification not only of the position (x, y), but also of the orientation θ of the polygon w.r.t to a frame of reference. Similarly as in the case of a 2D point robot, the state space is commonly referred to as the configuration space.*

(State Space of a 3D Polyhedral Robot). *A state (configuration) of a 3D polyhedral robot can be fully described by the position (x, y, z) and the orientation of the polyhedra w.r.t a frame of reference. Orientation in 3D can be described in a variety of ways, e.g., by Euler angles, by an axis and an angle, or by a quaternion.*

(State Space of a Simple Car). *In addition to the configuration (x, y, θ), the state of a simple car can include information about the velocity v and steering angle ψ.*

3.1.2 State Constraints

State constraints indicate a desired invariant that each state should satisfy. In motion planning, it is common to require avoiding collisions with obstacles and, for greater safety, even require that a minimum separation distance be maintained. When planning for an articulated robotic arm, constraints are also imposed on the joint limits in order to keep the rotations at each joint within desired bounds. In motion-planning problems that involve robotic vehicles, state constraints are often used to ensure that the vehicles maintain a reasonable speed and avoid sharp turns.

This thesis allows for a general specification of state constraints as a function $\textsc{Valid} : \mathcal{S} \rightarrow \{\top, \bot\}$, where

$$\textsc{Valid}(s) = \top \text{ iff } s \text{ satisfies the state constraints.}$$

3.1.3 Motion-Planning Goal

The motion-planning goal is specified as desired constraints that a goal state should satisfy. Such constraints could include a desired position or orientation. In motion-planning with dynamics, it is also common to require that a robot's velocity remain within certain bounds.

As in the case of $\textsc{Valid}$, this thesis allows for a general specification of a motion-planning goal as a function $\textsc{Goal} : \mathcal{S} \rightarrow \{\top, \bot\}$, where

$$\textsc{Goal}(s) = \top \text{ iff } s \text{ satisfies the motion-planning goal.}$$

3.1.4 Trajectory

A trajectory indicates the evolution of a system's state w.r.t time.

Definition 3.1.2. (Trajectory). *A trajectory is a function $\gamma : [0, T] \to \mathcal{S}$, parameterized by time $T \in \mathbb{R}^{\geq 0}$. The notation $|\gamma|$ indicates the time duration T. The set of all trajectories is denoted by Γ.*

Note that there is no requirement that γ should be a continuous function. This general definition of a trajectory accommodates hybrid-system trajectories, which, as described in Chapter 6, contain discrete transitions.

3.2 Motion-Planning Components

Sampling-based motion planners, including the multi-layered approach, `SyCLoP`, developed in this thesis, make use of common motion-planning components, such as trajectory concatenation, trajectory sampling, trajectory validation, state-space projections, and coverage estimates.

3.2.1 Trajectory Concatenation

Trajectory concatentation allows sampling-based motion planners to extend a trajectory by concatenating to its end another trajectory.

Definition 3.2.1. (Trajectory Concatenation). *Let $\gamma_1 : [0, T_1] \to \mathcal{S}$ and $\gamma_2 : [0, T_2] \to \mathcal{S}$, where $\gamma_1(T_1) = \gamma_2(0)$. The concatenation of γ_1 by γ_2, written as $\gamma_1 \circ \gamma_2$,*

is another trajectory $\gamma : [0, T_1 + T_2] \to \mathcal{S}$ *defined as*

$$\gamma(t) = \begin{cases} \gamma_1(t), & \text{if } t \in [0, T_1] \\ \gamma_2(t - T_1), & \text{if } t \in (T_1, T_1 + T_2]. \end{cases}$$

3.2.2 Trajectory Sampling

Sampling-based motion planners often employ a trajectory-sampling strategy to generate a trajectory $\gamma : [0, T] \to \mathcal{S}$ that starts at a given state $s \in \mathcal{S}$, i.e., $\gamma(0) = s$.

Definition 3.2.2. (Trajectory Sampling). *Let* $\mathcal{P} = (\mathcal{S}, \textsc{Valid}, s_{\text{init}}, \textsc{Goal})$ *be a motion-planning problem. Given a state* $s \in \mathcal{S}$*, a trajectory-sampling strategy* $\textsc{SampleTraj}(\mathcal{P}, s)$ *is a sampling function that, according to some probability distribution, generates a trajectory* $\gamma : [0, T] \to \mathcal{S}$ *that starts at* s*, i.e.,* $\gamma(0) = s$*.*

The only requirement imposed on $\textsc{SampleTraj}(\mathcal{P}, s)$ is that it should be a sampling function. The purpose of this requirement is to provide the motion planner with alternative trajectories that can start at $s \in \mathcal{S}$. In this way, subsequent calls to $\textsc{SampleTraj}(\mathcal{P}, s)$ produce different trajectories based on the probability distribution and the sampling strategy. This allows the motion planner to extend the search for a solution along different directions.

Note that trajectory sampling depends on the motion-planning problem $\mathcal{P}$ and in particular the robot model. In this way, SampleTraj provides `SyCLoP` with a general formulation that hides away the specifics of a particular motion-planning problem.

Chapters 5 and 6 describe common trajectory-sampling strategies used by `SyCLoP` in the case of motion planning for robot with dynamics and hybrid systems, respectively.

3.2.3 Trajectory Validation

The purpose of trajectory validation is to compute the largest part of a given trajectory $\gamma : [0, T] \rightarrow \mathcal{S}$, starting at time 0, that satisfies the state constraints. Trajectory validation is typically used in combination with trajectory sampling. Given a state $s \in \mathcal{S}$, trajectory sampling generates a trajectory $\gamma : [0, T] \rightarrow \mathcal{S}$ that starts at s. Then, trajectory validation is used to keep only the valid part of γ, starting at s. This allows the motion planner to consider only valid trajectories as it proceeds with a search for a solution to the motion-planning problem.

Definition 3.2.3. (Trajectory Validation). *Let $\mathcal{P} = (\mathcal{S}, \textsc{Valid}, s_{\text{init}}, \textsc{Goal})$ be a motion-planning problem. Given a trajectory $\gamma : [0, T] \rightarrow \mathcal{S}$, the function* $\textsc{ValidTraj} : \Gamma \rightarrow \Gamma$ *computes the largest valid part of γ, starting at time 0, as follows: Let K, $0 \leq K \leq T$, be as large as possible such that*

$$\forall k \in [0, K] : \textsc{Valid}(\gamma(k)) = \top.$$

Then, $\gamma_{\text{valid}} : [0, K] \rightarrow \mathcal{S}$, where

$$\forall k \in [0, K] : \gamma_{\text{valid}}(k) = \gamma(k)$$

denotes the largest valid part of γ starting at time 0.

The implementation of $\textsc{ValidTraj}(\gamma)$ relies on an incremental discretization of γ in order to compute γ_{valid}. At the i-th iteration, $\textsc{ValidTraj}(\gamma)$ checks the validity

of $\gamma(i * \epsilon)$, where $\epsilon > 0$ is a constant. If $\textsc{Valid}(\gamma(i * \epsilon)) = \bot$, then an invalid state is found, so $K = (i - 1) * \epsilon$, and the iteration stops. Otherwise, i is incremented by one until $i * \epsilon > T$. Note that $\epsilon > 0$ should be set to a small value in order to minimize the possibility of skipping over an invalid state, i.e., $\gamma(i * \epsilon) = \top$ and $\gamma((i+1) * \epsilon) = \top$, but $\gamma(t) = \bot$ for some $t \in (i * \epsilon, (i + 1) * \epsilon)$. Such an incremental approach is advocated in [CLH+05, LaV06] and is commonly used by sampling-based motion planners.

3.2.4 State-Space Projections

A projection of a space $\mathcal{S}$ onto another space $\mathcal{Z}$ is obtained via a projection function $\textsc{Proj} : \mathcal{S} \rightarrow \mathcal{Z}$. The multi-layered motion-planning approach `SyCLoP` developed in this thesis uses state-space projections to effectively reduce the dimensionality of $\mathcal{S}$ by projecting $\mathcal{S}$ onto a lower dimensional space $\mathcal{Z}$. Given a set of state samples $\bar{\mathcal{S}} = \{\bar{s}_1, \ldots, \bar{s}_n\}$ from $\mathcal{S}$, the projection of $\bar{\mathcal{S}}$ is obtained as $\bar{\mathcal{Z}} = \{\bar{z}_1, \ldots, \bar{z}_n\}$, where

$$\bar{z}_i = \textsc{Proj}(\bar{s}_i).$$

The objective of the projection is to reduce the dimensionality while at the same time preserve the underlying structure of the original set. For many motion-planning problems, simple projections that consider only some of the state components have been shown to work well in practice [LK04a, LK05, Lad06, PKV07a, PKV08b, PKV08d]. In particular, for motion-planning problems involving robotic vehicles such as cars, differential drives, unicycles, low-dimensional projections are usually obtained by considering only the position component of a state, i.e., $(x, y) \in \mathbb{R}^2$ (resp., $(x, y, z) \in \mathbb{R}^3$)

for a robot operating in a 2D (resp., 3D) workspace. For articulated-arm robots, the position of the end-effector is typically used for the projection.

For other motion-planning problems, such as those arising in reconfigurable robots and computational biology, it is more challenging to design an appropriate projection function that reduces the dimensionality while preserving the underlying structure of the original set. In these cases, dimensionality reduction [Jol86, CC00, HKO01] can provide a viable approach for automatically computing projections onto low-dimensional Euclidean spaces. The author's work in [PK06, PSCK07, PK07b] has developed an effective framework for computing low-dimensional projections that preserve the underlying structure of high-dimensional data remarkably well.

3.2.5 Coverage Estimates

An important issue that arises frequently in sampling-based motion planning relates to the estimation of coverage of a region of the state space by the samples generated by the motion planner. The motion-planning approach `SyCLoP` developed in this thesis, as described in Chapter 4, relies on coverage estimates in order to determine which parts of the state space should be further explored.

Consider a region $\mathcal{Y}$ and a set of samples $\bar{\mathcal{Y}} = \{\bar{y}_1, \ldots, \bar{y}_n\}$ from $\mathcal{Y}$. The objective of a coverage estimate

$$\mathrm{Cov}(\mathcal{Y}, \bar{\mathcal{Y}})$$

is to quantify how well $\bar{\mathcal{Y}}$ covers $\mathcal{Y}$. Coverage estimates were first introduced in the context of Monte Carlo methods as a way to measure the quality of deterministic sam-

pling (also referred to as quasirandom sampling) in comparison to random sampling (see recent books [Nie92, DT97] for details and extensive references on the subject). One such measure that has been widely used is the dispersion. As noted in [EKK04], while dispersion has been used in sampling-based motion planning to generate quasirandom samples [LBL03], its use as a coverage estimate is impeded by the significant cost required to compute it in high dimensions.

A viable way to avoid computational bottlenecks due to the high-dimensionality of the state space is to compute the coverage in a low-dimensional projection onto a Euclidean space. Such an approach, which has been advocated in [CLH+05, LaV06, LK04a, LK05, Lad06, PKV07a, PKV08b, PKV08d], is also followed in this thesis.

1. Given $\bar{\mathcal{Y}} = \{\bar{y}_1, \ldots, \bar{y}_n\}$, compute a low-dimensional projection $\bar{\mathcal{Z}} = \{\bar{z}_1, \ldots, \bar{z}_n\} \subset \mathbb{R}^m$ as described in Section 3.2.4, i.e., $\bar{z}_i = \textsc{Proj}(\bar{y}_i)$.

2. Overlay an implicit grid with n cells over $\mathbb{R}^m$.

3. Compute the coverage by counting the number of grid cells that have at least one sample inside, i.e.,

$$\textsc{Cov}(\mathcal{Y}, \bar{\mathcal{Y}}) = \sum_{i=1}^{n} \begin{cases} 1, & \text{if the } i\text{-th cell contains at least one sample } \bar{z} \text{ from } \bar{\mathcal{Z}} \\ 0, & \text{otherwise.} \end{cases}$$

Note that it is not necessary to maintain an explicit representation of the grid. In fact, a hash data structure is typically used to maintain a list of nonempty grid cells. When a new sample $\bar{y}$ is added to $\bar{\mathcal{Y}}$, then its projection is computed as

$$\bar{z} = \textsc{Proj}(\bar{y}).$$

The cell that $\bar{z}$ belongs to is then added to the list of nonempty grid cells (if not already there). This approach allows for fast updates and minimal memory requirements since the number of nonempty grid cells is always less than or equal to the number of samples. Note that a different spacing can be used along each dimension, allowing the grid to be coarser along some dimensions and finer along others.

3.3 Motion Planning as a Search Problem

Let $\mathcal{P} = (\mathcal{S}, \textsc{Valid}, s_{\text{init}}, \textsc{Goal})$ be a motion-planning problem, as defined in Section 3.1. Motion planning is generally considered as a search problem for a valid trajectory $\gamma : [0, T] \to \mathcal{S}$ that satisfies the motion-planning goal, i.e.,

$$\gamma(0) = s_{\text{init}}, \;\; \textsc{Goal}(\gamma(T)) = \top, \quad \text{and } \forall t \in [0, T] : \textsc{Valid}(\gamma(t)) = \top.$$

Many sampling-based motion planners follow a common framework that searches for a solution by extending in the state space $\mathcal{S}$ a tree rooted at the initial state s_{init}. Pseudocode of the basic search is given in Algo. 1.

A search data structure is maintained as a tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ (Algo. 1:2). A vertex $v(s) \in V_{\mathcal{T}}$ is associated with a state $s \in \mathcal{S}$, where $\textsc{Valid}(s) = \top$. An edge $(v(s_1), v(s_2)) \in E_{\mathcal{T}}$ indicates that a valid trajectory $\gamma_{s_1,s_2} : [0, T] \to \mathcal{S}$ from s_1 to s_2 has been computed, i.e.,

$$s_1 = \gamma_{s_1,s_2}(0), \;\; s_2 = \gamma_{s_1,s_2}(T), \;\; \text{and } \forall t \in [0, T] : \textsc{Valid}(\gamma_{s_1,s_2}(t)) = \top.$$

Initially, $V_{\mathcal{T}} = \{v(s_{\text{init}})\}$ and $E_{\mathcal{T}} = \emptyset$ (Algo. 1:2). As the search proceeds iteratively (Algo. 1:3–10), $\mathcal{T}$ is extended by adding new vertices and edges. At each iteration, a

Algorithm 1 A Basic Search-Tree Framework for the Motion-Planning Problem

Input: $\mathcal{P} = (\mathcal{S}, \textsc{Valid}, s_{\text{init}}, \textsc{Goal})$: motion-planning problem
$t_{\max} \in \mathbb{R}^{>0}$: upper bound on overall computation time
Output: A solution trajectory or $\perp$ if no solution trajectory is found

1: $\textsc{StartClock}$
2: $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$; $V_{\mathcal{T}} \leftarrow \{s_{\text{init}}\}$; $E_{\mathcal{T}} \leftarrow \emptyset$
3: **while** $\textsc{ElapsedTime} < t_{\max}$ **do**
4: $\quad s \leftarrow \textsc{SelectStateFromSearchTree}(\mathcal{P}, \mathcal{T})$
5: $\quad \gamma \leftarrow \textsc{SampleTraj}(\mathcal{P}, s)$ ◇*see Section 3.2.2*
6: $\quad \gamma_{\text{valid}} \leftarrow \textsc{ValidTraj}(\gamma)$ ◇*see Section 3.2.3*
7: $\quad s_{\text{new}} \rightarrow$ last state of γ_{valid}
8: $\quad V_{\mathcal{T}} \leftarrow V_{\mathcal{T}} \cup \{s_{\text{new}}\}$; $E_{\mathcal{T}} \leftarrow E_{\mathcal{T}} \cup \{(s, s_{\text{new}})\}$
9: $\quad$ **if** $\textsc{Goal}(s_{\text{new}}) = \top$ **then**
10: $\quad\quad$ **return** $\textsc{Traj}(\mathcal{T}, s_{\text{new}})$ ◇*compute solution trajectory*
11: **return** $\perp$

state s is first selected from the states already in $\mathcal{T}$ (Algo. 1:4). The search tree $\mathcal{T}$ is then extended by using a trajectory-sampling strategy (see Section 3.2.2) to generate a trajectory $\gamma : [0, T] \rightarrow \mathcal{S}$ that starts at s (Algo. 1:5) and keeping only the valid part γ_{valid} of γ (Algo. 1:6). The vertex $v(s_{\text{new}})$, where s_{new} is the last state of γ_{valid}, is added to $V_{\mathcal{T}}$ (Algo. 1:8). The edge $(v(s), v(s_{\text{new}}))$ is added to $E_{\mathcal{T}}$, and γ is associated with $(v(s), v(s_{\text{new}}))$ (Algo. 1:8). If $\textsc{Goal}(s_{\text{new}}) = \top$, then a solution trajectory is found (Algo. 1:9). In fact, the solution trajectory $\textsc{Traj}(\mathcal{T}, s_{\text{new}})$ (Algo. 1:10) can be computed by concatenating the trajectories associated with the tree edges that connect $v(s_{\text{init}})$ to $v(s_{\text{new}})$, i.e.,

$$\textsc{Traj}(\mathcal{T}, s_{\text{new}}) \stackrel{def}{=} \gamma_{s_{\text{init}}, s_1} \circ \gamma_{s_1, s_2} \circ \cdots \circ \gamma_{s_n, s_{\text{new}}},$$

where $v(s_1), \ldots, v(s_n)$ denotes the sequence of vertices that connects $v(s_{\text{init}})$ to $v(s_{\text{new}})$.

Chapter 4

SyCLoP: Synergic Combination of Layers of Planning

This chapter describes the proposed multi-layered approach, SyCLoP, which synergically combines high-level discrete planning with low-level motion planning to dramatically reduce the computational time for solving challenging motion-planning problems. SyCLoP, as shown later in the thesis, can be applied to motion-planning problems that incorporate robot dynamics, physics-based simulations, hybrid systems, and high-level tasks specified using the expressiveness of linear temporal logic.

4.1 Overall Approach

The efficiency of the search-tree framework presented in Section 3.3 depends on the ability of the motion planner to quickly extend the search tree $\mathcal{T}$ along those directions that can be used for computing a solution trajectory.

SyCLoP utilizes information provided by the problem specification and information gathered during previous exploration steps to guide future explorations closer to obtaining a solution to the given motion-planning problem. This is a concept that has been studied before mainly in the context of geometric motion planning by sampling-based motion planners that construct roadmaps. For example, PRM [KŠLO96] uses

the information of the connectivity of the samples to create more samples in parts of the configuration space where connectivity is low. The work in [BB05] uses nearest-neighbors information in the context of PRM to define the utility of each sample in an information-theoretic sense and only add to the roadmap those samples that increase the overall entropy. The planners in [MTP+04] and [HSAS05] also utilize information in the context of PRM to find appropriate sampling strategies for different parts of the configuration space. In contrast to roadmap methods, traditional tree-based motion planners such as RRT [LaV98,LK01] and EST [HLM97,HKLR02] rely on limited information, such as distance metrics or simple heuristics to guide the search. Although the tree may initially advance quickly, as the time goes on, the growth slows down rapidly as it becomes more and more difficult to find promising directions for the search. These limitations become even more pronounced when solving challenging motion-planning problems that incorporate richer models of the robot and the physical world, such as robot dynamics, physics-based simulations, and hybrid systems. In these cases, the added complexity renders current motion planners practically ineffective. To address some of the limitations observed in tree-based planners in solving challenging problems with dynamics, recent work in [LK04a,LK05,Lad06] and [BK07] rely on a subdivision scheme and potential fields to guide the tree search, respectively.

To effectively guide the search for a solution and overcome the limitations of current motion planners, SyCLoP seamlessly combines high-level discrete planning with low-level motion planning in a multi-layered approach. As mentioned in the

introduction in Chapter 1, in the first layer, planning takes place in a simplified high-level and discrete model of the motion-planning problem. In the second layer, motion planning is based on the full model of the motion-planning problem.

The purpose of high-level planning, which draws from research in AI and logic, is to guide low-level motion planning as it extends $\mathcal{T}$ in search for a solution to the motion-planning problem. The high-level planning provides high-level plans, which constitute solutions to the motion-planning problem under the simplified model.

The rationale for high-level planning is that solutions obtained under the simplified model can be indicative of solutions under the full model of the motion-planning problem. Moreover, from a computational perspective, it is significantly more efficient to obtain solutions under the simplified model than under the full model.

The objective of the low-level planning in the second layer is to extend $\mathcal{T}$ so that it closely follows the current high-level plan. Since a high-level plan constitutes a solution to the motion-planning problem under the simplified model, by closely following the high-level plan, the low-level motion planner might be able to obtain a solution under the full model of the motion-planning problem.

A distinctive feature and a crucial property of `SyCLoP` is the synergic combination of high-level discrete planning and low-level motion planning. Note that it is not known a priori which high-level plan would be the best in guiding the low-level motion planner toward a solution. As the search progresses, the different planning layers in `SyCLoP` exchange information with one another in order to evaluate the feasi-

bility of current high-level plans and compute increasingly feasible high-level plans in future iterations. Aiming to strike a balance between greedy and methodical search, SyCLoP gives priority to highly feasible plans, but at the same time it does not ignore other plans. This is especially relevant in the early stages of the search when more information is needed to properly evaluate the feasibility of different high-level plans.

This synergic combination of high-level discrete planning and low-level motion planning provides SyCLoP with the flexibility to extend the search tree $\mathcal{T}$ along useful directions while able to radically change direction if information from the search suggests other highly feasible plans. As shown in later chapters of this thesis, SyCLoP dramatically reduces the computational cost in solving challenging motion-planning problems that incorporate robot dynamics and physics-based simulations (Chapter 5), hybrid-systems (Chapter 6), and high-level tasks expressed in LTL (Chapter 7).

The rest of this chapter is as follows. The simplified high-level and discrete model of the motion-planning problem is described in Section 4.2. The synergic combination of the high-level discrete planning and low-level motion planning in SyCLoP is described in Section 4.3. Descriptions of the high-level discrete planning and low-level motion planning are provided in Sections 4.4 and 4.5, respectively.

4.2 High-Level Discrete Model of Motion-Planning Problem

The high-level planning layer in SyCLoP operates on a simplified discrete model of the motion-planning problem. Although it is possible to consider other high-

level models, this thesis focuses on discrete models due to their simplicity and the computational efficiency of planning on discrete models as compared to continuous models. Moreover, the use of discrete models allows `SyCLoP` to benefit from research in computer logic and AI, where discrete planning plays an important role.

Discretizations of the motion-planning problem in the context of geometric motion planning appeared early in the literature. Key theoretical results were obtained using discretizations based on decompositions of the collision-free configuration space [Lat91]. As discussed in related work in Chapter 2, the notion of a roadmap, introduced by Canny's algorithm [Can88a], provides a discretization of the motion-planning problem in the form of a graph. Each vertex in the graph is associated with a collision-free region of the configuration space. The union of all the regions associated with the graph vertices corresponds to the collision-free configuration space. An edge in the graph indicates physical adjacency of the regions associated with the end-vertices of the edge. Similar discretizations are also obtained by exact and approximate cell-based decomposition methods, which decompose the collision-free configuration space into a collection of cells (see discussions in [Lat91, CLH+05, LaV06]).

A distinctive feature of `SyCLoP` in contrast to decomposition methods is that `SyCLoP` does not impose any strict requirements on the discrete model of the motion-planning problem. In particular, `SyCLoP` does not require the discrete model to be based on a partition of the collision-free configuration space. This allows `SyCLoP` to consider discrete models that can be computed efficiently, as opposed to the

exponential-time cost required to partition the collision-free configuration space.

In many cases, the discrete model used by the high-level planning layer of `SyCLoP` is based on a simple partition of the state space $\mathcal{S}$ into a finite number of regions

$$\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n\}, \text{ where } \mathcal{S} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \cdots \cup \mathcal{R}_n.$$

Since `SyCLoP` does not require that a region $\mathcal{R}_i \in \mathcal{R}$ should contain only valid states, such partitions can be easily obtained in a variety of ways. For instance, as described in Chapter 5, partitions used for motion-planning problems that incorporate robot dynamics are based on grid and triangular decompositions of the workspace on which the robot operates. In the case of motion-planning for hybrid systems, partitions are based on the discrete logic employed by the hybrid system (see Chapter 6). When incorporating high-level tasks expressed in LTL into motion planning, as described in Chapter 7, partitions are based on the LTL formula specifying the high-level task.

The high-level discrete model is represented in terms of a graph $G_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}})$, whose vertices are regions in the partition $\mathcal{R}$ and whose edges denote adjacency between regions. Region $\mathcal{R}_i \in \mathcal{R}$ is considered adjacent to $\mathcal{R}_j \in \mathcal{R}$ if there exists a trajectory $\gamma : [0, T] \to \mathcal{S}$ that goes directly from a state in $\mathcal{R}_i$ to a state in $\mathcal{R}_j$, i.e.,

$$\gamma(0) \in \mathcal{R}_i, \;\; \gamma(T) \in \mathcal{R}_j, \;\; \text{and} \;\; \forall t \in [0, T] : \gamma(t) \in \mathcal{R}_i \cup \mathcal{R}_j.$$

Then, $V_{\mathcal{R}} = \{v(\mathcal{R}_i) : \mathcal{R}_i \in \mathcal{R}\}$ and

$$E_{\mathcal{R}} = \{(v(\mathcal{R}_i), v(\mathcal{R}_j)) : (\mathcal{R}_i, \mathcal{R}_j \in \mathcal{R}) \wedge (\mathcal{R}_i \text{ is adjacent to } \mathcal{R}_j)\}.$$

Fig. 4.1(a, b) provides an illustration. The high-level discrete model also keeps information about regions associated with the initial state and regions which contain

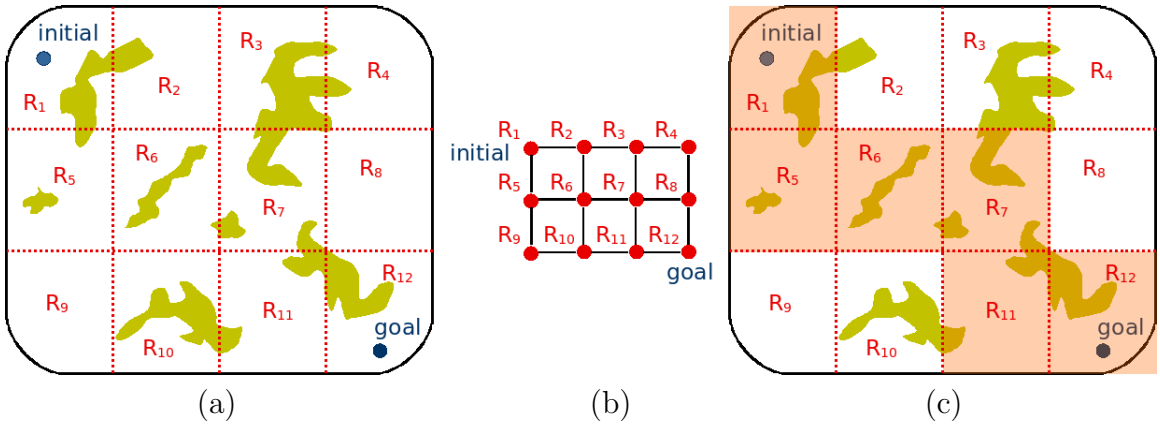


Fig. 4.1: (a) A grid-based partition. (b) The high-level discrete model represented as a graph. (c) Example of a high-level plan.

states that satisfy the motion-planning goal. More specifically,

- $\mathcal{R}_{\text{init}} = \{\mathcal{R}_i : \mathcal{R}_i \in \mathcal{R} \wedge s_{\text{init}} \in \mathcal{R}_i\}$ and
- $\mathcal{R}_{\text{GOAL}} = \{\mathcal{R}_i : \mathcal{R}_i \in \mathcal{R} \wedge (\exists s \in \mathcal{R}_i : \text{GOAL}(s) = \top)\}$.

Putting it all together, the high-level discrete model is a tuple

$$\mathcal{D} = (\mathcal{R}, G_{\mathcal{R}}, \mathcal{R}_{\text{init}}, \mathcal{R}_{\text{GOAL}}).$$

A solution w.r.t to the discrete model is a connected sequence of regions $[\mathcal{R}_{i_j}]_{j=1}^{k}$ that starts at a region in $\mathcal{R}_{\text{init}}$ and ends at a region in $\mathcal{R}_{\text{GOAL}}$, i.e.,

$$\mathcal{R}_{i_1} \in \mathcal{R}_{\text{init}}, \quad \mathcal{R}_{i_k} \in \mathcal{R}_{\text{GOAL}}, \quad \text{and} \quad (v(\mathcal{R}_{i_j}), v(\mathcal{R}_{i_j})) \in E_{\mathcal{R}}, \forall j \in \{1, 2, \ldots, k-1\}.$$

4.3 Interplay of Planning Layers

As discussed in Section 4.1, `SyCLoP` systematically takes advantage of the fact that solutions obtained under the high-level model can be indicative of solutions under the full model of the motion-planning problem. `SyCLoP` uses high-level plans to guide the

low-level motion planner as it extends $\mathcal{T}$ in search for a solution. In turn, information gathered during the search, such as the progress made in following the high-level plan, is fed back from the low-level to the high-level layer. In this way, high-level plans become increasingly useful in guiding the low-level motion planner toward a solution.

Consider a high-level plan $[\mathcal{R}_{i_j}]_{j=1}^{k}$. Recall that $[\mathcal{R}_{i_j}]_{j=1}^{k}$ connects $\mathcal{R}_{i_1} \in \mathcal{R}_{\text{init}}$ to $\mathcal{R}_{i_k} \in \mathcal{R}_{\text{GOAL}}$ (see Section 4.2 and Fig. 4.1(c)). If a solution trajectory $\gamma : [0, T] \to \mathcal{S}$ exists that reaches $\mathcal{R}_{i_1}, \mathcal{R}_{i_2}, \ldots, \mathcal{R}_{i_k}$ in succession, then $[\mathcal{R}_{i_j}]_{j=1}^{k}$ is considered feasible. A feasible high-level plan is indicative of solutions under the full model of the motion-planning problem. Since it is not known a priori which high-level plan is feasible, `SyCLoP` maintains a running weight estimate $w(\left[\mathcal{R}_{i_j}\right]]_{j=1}^{k})$ on the feasibility of $\left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$. A high weight indicates that `SyCLoP` is making significant progress in following the high-level plan, while a low weight indicates little or no progress.

The core part of `SyCLoP`, illustrated in Fig. 1.1 and 4.2, proceeds by repeating the following steps until a solution is found or a maximum amount of time has elapsed:

1. Obtain a high-level plan $\left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$ by a high-level discrete planner operating on the discrete model $\mathcal{D} = (\mathcal{R}, G_{\mathcal{R}}, \mathcal{R}_{\text{init}}, \mathcal{R}_{\text{GOAL}})$ of the motion-planning problem.

2. Use low-level motion planning to extend the search tree $\mathcal{T}$ from one region to its neighbor, as specified by the current high-level plan $\left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$.

3. Update the weights $w(\left[\mathcal{R}_{i_j}\right]_{j=1}^{k})$ on the feasibility estimates of the high-level plans in order to compute increasingly feasible plans in future iterations.

The discrete model can provide `SyCLoP` with many alternative high-level plans.

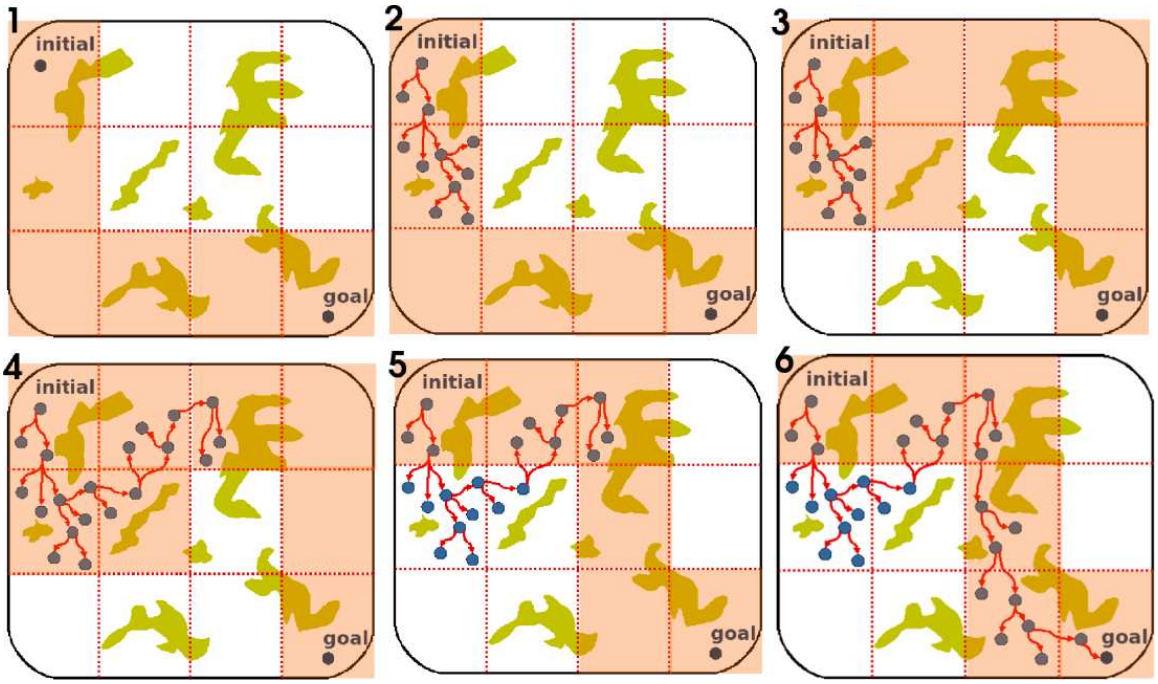


Fig. 4.2: Interplay of high-level discrete planning and low-level motion planning in `SyCLoP`. Given a high-level plan, the low-level motion planner extends the search tree $\mathcal{T}$, so that it closely follows the high-level plan. Information gathered during the search is fed back from the low-level to the high-level layer to compute increasingly feasible high-level plans in future iterations. Obstacles are shown in yellow. High-level plans are shown in light red. Tree vertices are shown as blue circles, while tree edges are shown as red curves.

A central issue is which high-level plan to choose at each iteration from the combinatorially large number of possibilities. Since the feasibility estimates are based on partial information, it is important not to ignore high-level plans associated with lower weights, especially during the early stages of the search.

`SyCLoP` aims to strike a balance between greedy and methodical search. For this reason, `SyCLoP` selects more frequently high-level plans associated with higher feasibility estimates and less frequently high-level plans associated with lower feasibility estimates. Information gathered during the search by the low-level motion planning (such as coverage, regions $\mathcal{R}_{i_j}$ that have been reached, and time spent) is used to

update the weights $w(\left[\mathcal{R}_{i_j}\right]_{j=1}^{k})$ on the feasibility of high-level plans.

As a result, a new high-level plan associated with a high weight could be selected in the next iteration. In turn, by receiving high-level plans $\left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$ that are estimated to be highly feasible, the low-level motion planner is able to make progress and extend $\mathcal{T}$ toward $\mathcal{R}_{i_1}, \mathcal{R}_{i_2}, \ldots, \mathcal{R}_{i_k}$ in succession until it successfully computes a solution.

This synergic combination of high-level discrete planning (Section 4.4) and low-level motion planning (Section 4.5) through the weight estimates on the feasibility of high-level plans is a crucial component of `SyCLoP`. This combination provides `SyCLoP` with the flexibility to extend $\mathcal{T}$ along useful directions while able to radically change direction if information from the search suggests other highly feasible plans, as illustrated in Fig. 4.2. Pseudocode for `SyCLoP` is given in Algo. 2.

4.4 High-Level Discrete Planning: Guiding the Search

The current high-level plan σ is computed at each iteration (Algo. 2:6) by searching the graph $G_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ of the discrete model $\mathcal{D}$ for a connected sequence of regions $[\mathcal{R}_{i_j}]_{j=1}^{k}$ from $\mathcal{R}_{i_1} \in \mathcal{R}_{\text{init}}$ to $\mathcal{R}_{i_k} \in \mathcal{R}_{\text{GOAL}}$. As discussed in Section 4.3, the discrete model can provide `SyCLoP` with combinatorially many alternative high-level plans. Since feasibility estimates are based on partial information, especially during early stages of the search, it is important not to ignore high-level plans associated with low weights. For this reason, `SyCLoP` aims to balance greedy and methodical search in the computation of high-level plans at each iteration of the core loop (Algo. 2:6).

Algorithm 2 `SyCLoP`: Synergic Combination of Layers of Planning

Input: $\mathcal{P} = (\mathcal{S}, \textsc{Valid}, s_{\text{init}}, \textsc{Goal})$: motion-planning problem
$t_{\max} \in \mathbb{R}$: upper bound on overall computation time
Output: A solution trajectory or $\perp$ if no solution trajectory is found

1: $\textsc{StartClock}$
2: $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$; $V_{\mathcal{T}} \leftarrow \{s_{\text{init}}\}$; $E_{\mathcal{T}} \leftarrow \emptyset$ ♢*initialize search tree*
3: $\mathcal{D} = (\mathcal{R}, G_{\mathcal{R}}, \mathcal{R}_{\text{init}}, \mathcal{R}_{\textsc{Goal}}) \leftarrow \textsc{DiscreteModel}(\mathcal{P})$ ♢ *construct discrete model*
4: $\textsc{InitFeasibilityEstimates}(G_{\mathcal{R}}, w)$ ♢*initialize feasibility estimates*
5: **while** $\textsc{time} < t_{\max}$ **do** ♢*core loop interplay: discrete search–motion planning*
6: $\sigma \stackrel{def}{=} [\mathcal{R}_{i_j}]_{j=1}^{k} \leftarrow \textsc{HighLevelPlanning}(\mathcal{D}, w)$ ♢*compute high-level plan*
7: $\sigma_{\text{avail}} \leftarrow \emptyset$ ♢*begin motion-planning step*
8: **for** $j = k, k-1, \ldots, 1$ **do** ♢*start using high-level plan*
9: **if** $\textsc{States}(\mathcal{T}, \mathcal{R}_{i_j}) \neq \emptyset \wedge \text{rand}(0,1) < \frac{1}{1+|\sigma_{\text{avail}}|^2}$ **then**
10: $\sigma_{\text{avail}} \leftarrow \{\mathcal{R}_{i_j}\} \cup \sigma_{\text{avail}}$ ♢*get directions from the high-level plan*
11: **for** several times **do**
12: $\mathcal{R}_{i_j} \leftarrow \textsc{SelectRegion}(\sigma_{\text{avail}}, w)$ ♢ *select region from available regions*
13: $s \leftarrow \textsc{SelectState}(\textsc{States}(\mathcal{T}, \mathcal{R}_{i_j}), w)$ ♢*select state for propagation*
14: $\gamma_{\text{valid}} \leftarrow \textsc{ExtendSearchTree}(\mathcal{T}, s)$ ♢*attempt to extend search tree*
15: $s_{\text{new}} \leftarrow$ last state of γ_{valid}
16: **if** $s_{\text{new}} \neq$ `NULL` **then** ♢*was the tree extended?*
17: $V_{\mathcal{T}} \leftarrow V_{\mathcal{T}} \cup \{s_{\text{new}}\}$; $E_{\mathcal{T}} \leftarrow E_{\mathcal{T}} \cup \{(s, s_{\text{new}})\}$ ♢*add state and trajectory*
18: $\textsc{Update}(\sigma_{\text{avail}})$ ♢*consider for selection newly reached regions*
19: **if** $\textsc{Goal}(s_{\text{new}}) = \top$ **then return** $\textsc{Traj}(\mathcal{T}, s_{\text{new}})$ ♢*solution*
20: $\textsc{UpdateFeasibilityEstimates}(G_{\mathcal{R}}, w)$
21: **return** $\perp$

4.4.1 Balancing Greedy and Methodical Search

An effective strategy that balances greedy and methodical search can be obtained by selecting each high-level plan σ with probability $w(\sigma)/\sum_{\sigma'} w(\sigma')$. This selection process is biased towards high-level plans that are estimated to be highly feasible, since the objective of `SyCLoP` is to quickly compute a solution trajectory. At the same time, since it is not known a priori which σ actually leads to a solution trajectory, the selection process guarantees that each high-level plan has a non-zero probability of being selected. Computationally, however, such strategy is feasible only when it is

practical to enumerate all high-level plans. Due to the discrete model, there is usually a combinatorial number of high-level plans, which makes enumeration impractical.

4.4.2 Estimating the Feasibility of High-Level Plans

To address the issue of selecting a high-level plan among combinatorially many plans, `SyCLoP` associates a weight $w(\mathcal{R}_i)$ with each $\mathcal{R}_i \in \mathcal{R}$. The weight $w(\mathcal{R}_i)$ is a running estimate on the feasibility of including $\mathcal{R}_i$ in the current high-level plan.

The weight $w(\mathcal{R}_i)$ is computed based on information gathered by the low-level motion planner during each exploration of $\mathcal{R}_i$. As it can be imagined, there are many ways that can be used to compute $w(\mathcal{R}_i)$. This thesis focused on simple and efficient computations that were shown to work well in practice for solving challenging motion-planning problems involving robot dynamics, physics-based simulations, hybrid systems, and high-level tasks expressed in LTL.

The weight $w(\mathcal{R}_i)$ in this thesis is computed as

$$w(\mathcal{R}_i) = \frac{\mathrm{vol}^{z_1}(\mathcal{R}_i) * \mathrm{cov}^{z_2}(\mathcal{R}_i)}{\mathrm{time}(\mathcal{R}_i)}, \tag{4.1}$$

where

- $\mathrm{vol}(\mathcal{R}_i)$ is the volume of $\mathcal{R}_i$;
- $\mathrm{cov}(\mathcal{R}_i)$ is an estimate on the coverage of $\mathcal{R}_i$ by the states in $\mathcal{T}$.
- $\mathrm{time}(\mathcal{R}_i)$ is the total time the motion planner has spent exploring $\mathcal{R}_i$;
- z_1, z_2 are normalization constants, where usually $0 < z_1 < z_2 \leq 1$.

The coverage $\text{cov}(\mathcal{R}_i)$ is computed as

$$\text{cov}(\mathcal{R}_i) = \textsc{Cov}(\mathcal{R}_i, \textsc{States}(\mathcal{T}, \mathcal{R}_i)),$$

where $\textsc{Cov}(\mathcal{R}_i, \textsc{States}(\mathcal{T}, \mathcal{R}_i))$ is described in Section 3.2.5 and $\textsc{States}(\mathcal{T}, \mathcal{R}_i)$ denotes the states $s \in \mathcal{T}$ associated with $\mathcal{R}_i$, i.e, $s \in \mathcal{R}_i$.

When $\text{cov}(\mathcal{R}_i)$ is high, then there are many states in $\mathcal{R}_i$ which `SyCLoP` can use to extend $\mathcal{T}$ to the next region in the current high-level plan. Preference is also given to $\mathcal{R}_i$ when it has a large volume, since it allows `SyCLoP` to extend $\mathcal{T}$ in different directions. The term $\text{time}(\mathcal{R}_i)$ ensures that `SyCLoP` does not spend all the computation time extending $\mathcal{T}$ from states associated with one particular $\mathcal{R}_i$. In fact, as $\text{time}(\mathcal{R}_i)$ increases, the likelihood that $\mathcal{R}_i$ is included in the current high-level plan decreases rapidly, allowing `SyCLoP` to spend time extending $\mathcal{T}$ from states associated with other regions. In this way, `SyCLoP` associates a high weight $w(\mathcal{R}_i)$ with $\mathcal{R}_i$ when $\mathcal{R}_i$ has a large volume and has been covered well in a short amount of time.

4.4.3 Computing the High-Level Plan

The computation of a high-level plan is essentially a graph-search algorithm and the literature on this subject is abundant (see [Zha06] for extensive references). The combination of search strategies in this thesis aims to bias the computation toward high-level plans associated with high weights. However, random high-level plans are also used, although less frequently, as a way to correct for errors inherent with the estimates. The use of randomness is motivated by observations made in [GO02,PBC+05],

where random restarts and random neighbors have been suggested as effective ways to unblock the exploration when sampling-based motion planners get stuck.

With high probability, `SyCLoP` computes the high-level plan (Algo. 2:6) as the shortest path by using an adaptation of Dijkstra's shortest path algorithm, where an edge $(v(\mathcal{R}_i), v(\mathcal{R}_j)) \in E_{\mathcal{R}}$ is assigned the weight $1/(w(\mathcal{R}_i) * w(\mathcal{R}_j))$. In this way, `SyCLoP` selects at each iteration a high-level plan that is estimated to be highly feasible for advancing the search toward the goal.

With small probability, `SyCLoP` computes the high-level plan (Algo. 2:6) as a random sequence of edges that connect a region in $\mathcal{R}_{\text{init}}$ to a region in $\mathcal{R}_{\text{GOAL}}$. This computation is carried out by using depth-first search, where the frontier nodes are visited in a random order.

4.5 Low-Level Motion Planning: Explore

The exploration starts by rooting a tree $\mathcal{T}$ at the specified initial state s_{init} (Algo. 2:2). The objective of the motion-planning step (Algo. 2:7–20) is to quickly extend $\mathcal{T}$ from states associated with regions specified by the current high-level plan $\left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$. At each iteration a region R_{i_j} is selected from $\left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$ and explored for a short period of time. The exploration aims to extend $\mathcal{T}$ from $\mathcal{R}_{i_j}$ to $\mathcal{R}_{i_{j+1}}$. For this reason, several states are selected from the states associated with $\mathcal{R}_{i_j}$ and are extended toward $\mathcal{R}_{i_{j+1}}$.

4.5.1 Selecting a Region From the High-Level Plan

The objective is to select a nonempty region $\mathcal{R}_{i_j} \in \left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$ whose exploration causes $\mathcal{T}$ to grow closer to the goal. Since $\left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$ specifies a sequence of neighboring regions that end at a region associated with the motion-planning goal, the order in which the regions appear in the high-level plan provides an indication of how close each region is to the goal. For this reason, `SyCLoP` prefers to select regions that appear toward the end of $\left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$ more frequently than regions that appear at the beginning. `SyCLoP` maintains a set $\mathcal{R}_{\text{avail}}$ of regions that it considers for the selection process. At the beginning of each motion-planning step, $\mathcal{R}_{\text{avail}}$ is set to $\emptyset$ (Algo. 2:7). Then, the current lead $\left[\mathcal{R}_{i_j}\right]_{j=1}^{k}$ is scanned backwards starting at $j = k$ down to 1. If there are states $s \in \mathcal{T}$ associated with $\mathcal{R}_{i_j}$, i.e., $\textsc{States}(\mathcal{T}, \mathcal{R}_{i_j}) \neq \emptyset$, then $\mathcal{R}_{i_j}$ is added to $\mathcal{R}_{\text{avail}}$ with probability $1/(1 + |\mathcal{R}_{\text{avail}}|^2)$ (Algo. 2:8–10). A region R_{i_j} is then selected from R_{avail} (Algo. 2:12) with probability

$$\frac{w(\mathcal{R}_{i_j})}{\sum_{\mathcal{R}_{i_k} \in \mathcal{R}_{\text{avail}}} w(\mathcal{R}_{i_k})},$$

where $w(\mathcal{R}_{i_j})$, defined in Eqn. 4.1, estimates the feasibility of $\mathcal{R}_{i_j}$.

As noted in Section 4.4.2, the selection schemes presented in this thesis are shown to work well on practice for challenging motion-planning problems involving robot dynamics, physics-based simulations, hybrid systems, and high-level tasks expressed in LTL. As it can be imagined, it is possible to develop better selection schemes that can further improve the computational efficiency of `SyCLoP` in solving new and challenging motion-planning problems.

4.5.2 Selecting a State From a High-Level Region

From the states in $\mathcal{T}$ that are associated with $\mathcal{R}_{i_j}$, i.e. $\textsc{States}(\mathcal{T}, \mathcal{R}_{i_j})$, a state s is selected with probability

$$\frac{1}{\text{nsel}(s)} \Big/ \sum_{s' \in \textsc{States}(\mathcal{T}, \mathcal{R}_{i_j})} \frac{1}{\text{nsel}(s')},$$

where $\text{nsel}(s)$ is the number of times s has been selected in the past. This selection schemes follows well-established strategies in motion planning [CLH+05,LaV06,SL02, PKV07b,PKV08a] based on probability distributions that favor those states that have been selected less frequently.

4.5.3 Extending the Search Tree by Trajectory Sampling

Given a state $s \in \mathcal{T}$, the search tree $\mathcal{T}$ is extended from s by using the trajectory-sampling strategy, $\textsc{SampleTraj}(\mathcal{P}, s)$ (see Section 3.2.2), to generate a trajectory $\gamma : [0, T] \to \mathcal{S}$ that starts at s, i.e., $\gamma(0) = s$. Then, starting from $t = 0$, `SyCLoP` checks the validity of γ and keeps only the valid portion of γ (see Section 3.2.2).

4.5.4 Adding a New Branch to the Search Tree

Let $\gamma_{\text{valid}} : [0, K] \to \mathcal{S}$ denote the valid trajectory extended from $s \in \mathcal{T}$. Then, $s_{\text{new}} \stackrel{def}{=} \gamma_{\text{valid}}(K)$ and $(v(s), v(s_{\text{new}}))$ are added to $\mathcal{T}$ (Algo. 2:17). In addition, s_{new} is added to the appropriate region $\mathcal{R}_i$. If $\mathcal{R}_i$ is not in $\mathcal{R}_{\text{avail}}$, then $\mathcal{R}_i$ is added to $\mathcal{R}_{\text{avail}}$ (Algo. 2:18). Thus, when $\mathcal{T}$ reaches new regions, they become available for selection during the next iteration of the motion-planning step.

Chapter 5

Motion Planning with Nonlinear Dynamics

Motion planning that incorporates nonlinear dynamics is greatly motivated by the availability of new robots and the need to produce trajectories that respect the physical constraints in the motion of these robots. This chapter demonstrates the computational efficiency of `SyCLoP` in solving challenging motion-planning problems for robotic systems with nonlinear dynamics. Experiments on nonlinear models of robotic vehicles show significant computational speedups of one to two orders of magnitude when compared to state-of-the-art motion planners.

5.1 Control-based Systems

Many physical systems and in particular robots are often controlled by applying external inputs. As an example, an automatic car is driven by applying acceleration and rotating the steering wheel. The dynamics of a system, which are often nonlinear, describe the evolution of a system's state. This section defines control-based systems using a general formulation that allows treating the dynamics as a black box.

Definition 5.1.1. (Control-based System). *A control-based system is a triple* $\mathcal{M} = (\mathcal{S}, \mathcal{U}, f)$*, where*

- $\mathcal{S}$ *is a state space consisting of a finite set of variables that completely describe*

the state of the system (see Section 3.1.1);

- *$\mathcal{U}$ is a control space consisting of a finite set of input variables that can be applied to the system (see Section 5.1.1);*

- *$f : \mathcal{S} \times \mathcal{U} \times \mathbb{R}^{\geq 0} \rightarrow \mathcal{S}$ is a flow function that simulates the system dynamics when an input is applied to the system for a certain time duration (see Section 5.1.2).*

5.1.1 Control Space

A control is an external input that can be applied to a system in order to affect its behavior. Each control is represented by a collection of variable values. The set of all controls constitutes the control space, which is denoted by $\mathcal{U}$ and is defined as

$$\mathcal{U} = \{u : u \text{ is a control}\}.$$

5.1.2 Dynamics

When a system is at a state $s \in \mathcal{S}$ and a control $u \in \mathcal{U}$ is applied for a duration of $t \in \mathbb{R}^{\geq 0}$ units of time, the system's state evolves according to the underlying dynamics and at the end the system may be at a new state $s_{\text{new}} \in \mathcal{S}$. Such behavior is captured by a flow function, which is defined as follows.

Definition 5.1.2. (Flow Function). *A flow function $f : \mathcal{S} \times \mathcal{U} \times \mathbb{R}^{\geq 0} \rightarrow \mathcal{S}$ is such that, for each $s \in \mathcal{S}$, $u \in \mathcal{U}$, and $t \in \mathbb{R}^{\geq 0}$, $f(s, u, t)$ outputs the new state $s_{\text{new}} \in \mathcal{S}$ obtained by applying the input u for t units of time when the system is at state s. Consistency in the flow function is ensured by the following requirements:*

- **(Identity Property).** *For each $s \in \mathcal{S}$ and $u \in \mathcal{U}$,*

$$s = f(s, u, 0).$$

- **(Transitive Property).** *For each $s \in \mathcal{S}, u \in \mathcal{U}, t_1 \in \mathbb{R}^{\geq 0}, t_2 \in \mathbb{R}^{\geq 0}$,*

$$f(s, u, t_1 + t_2) = f(f(s, u, t_1), u, t_2).$$

Differential Equations

In many physical systems, the dynamics of the state evolution is commonly described by a set of differential equations of the form $g : \mathcal{S} \times \mathcal{U} \to \dot{\mathcal{S}}$ with first, second, or higher order derivatives. In such cases, closed-form solutions (if available) or state-of-the-art numerical integrations that minimize numerical errors can be used for the computation of the flow function f. An example of a kinematic car is provided below. Several examples of second-order dynamics are given in Section 5.3.2.

Example 5.1.1. (`KCar`: Kinematic Car).

- *State Space $\mathcal{S}$: The state $s = (x, y, \theta)$ of a kinematic car model consists of a position $(x, y) \in \mathbb{R}^2$ and an orientation $\theta \in [0, 2\Pi)$.*

- *Control Space $\mathcal{U}$: The kinematic car is controlled by setting the speed (u_0) and steering angle (u_1). The speed and steering control are restricted to $|u_0| \leq \mathrm{v}_{\max} = 3m/s$ and $|u_1| \leq \psi_{\max} = 35°$.*

- *Equations of Motion: $\dot{x} = u_0 \cos(\theta)$, $\dot{y} = u_0 \sin(\theta)$, $\dot{\theta} = u_0 \tan(u_1)/L$, where L is the distance between the front and rear axles.*

Physics-based Simulations

When differential equations become too cumbersome to fully describe the dynamics, a computer program, such as physics-based simulators, can be used for the dynamics simulation. Simulation allows modeling of complex robot dynamics, friction, gravity, and interactions of the robot with the environment, which cannot be easily described analytically. Similar to the abstraction of collision checking in early sampling-based approaches, physics-based simulations allow the motion planner to access the necessary components for planning purposes, while hiding the intricacies of the robot and its interactions with the environment. This general treatment allows `SyCLoP` to handle systems with general nonlinear dynamics.

5.2 Applying `SyCLoP` to Motion Planning with Nonlinear Dynamics

The multi-layered approach `SyCLoP`, described in Chapter 4, can be used in the context of motion planning for control-based systems with nonlinear dynamics. This section describes the high-level discrete model and the trajectory-sampling strategy that are used by `SyCLoP` for motion-planning problems with nonlinear dynamics.

5.2.1 High-Level Discrete Model of Motion-Planning Problem

As discussed in Section 4.2, the high-level discrete model provides a simplified high-level planning layer that is used to effectively guide the low-level motion planner

as it extends the search tree $\mathcal{T}$. Recall that the discrete model is based on a partition

$$\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n\}$$

of the state space $\mathcal{S}$ into different regions. `SyCLoP` computes the partition $\mathcal{R}$ based on a workspace decomposition. For robots operating in 2D (resp., 3D) environments, the workspace, denoted by $\mathcal{W}$, corresponds to a region in $\mathbb{R}^2$ (resp., $\mathbb{R}^3$). Without loss of generality, $\mathcal{W}$ is assumed to be a unit square in 2D and a unit cube in 3D. A projection function $\textsc{Proj} : \mathcal{S} \rightarrow \mathcal{W}$ (see Section 3.2.4) maps each state $s \in \mathcal{S}$ to a point in the workspace $\mathcal{W}$, typically, by extracting the position component from s. Then, the workspace is decomposed into different regions

$$\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_n,$$

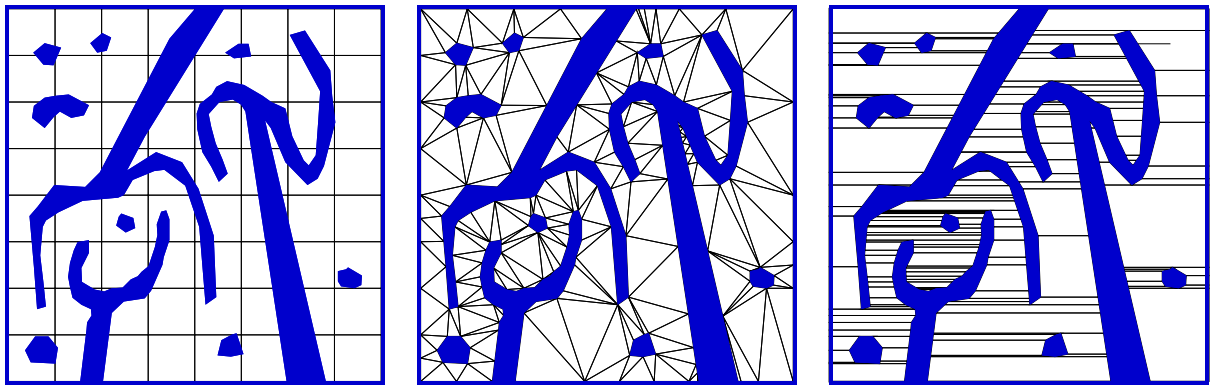
and the state-space partition $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n\}$ is computed as

$$\mathcal{R}_i = \{s \in \mathcal{S} : \textsc{Proj}(s) \in \mathcal{W}_i\}, \quad \text{for } i = 1, 2, \ldots, n.$$

The graph $G_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ of the discrete model is then computed as

- $V_{\mathcal{R}} = \{v(\mathcal{R}_i) : \mathcal{R}_i \in \mathcal{R}\}$ and
- $E_{\mathcal{R}} = \{(v(\mathcal{R}_i), v(\mathcal{R}_j)) : \mathcal{W}_i \text{ is adjacent to } \mathcal{W}_j\}$.

The computation of workspace decompositions is an active research area in computational geometry, and over the years numerous methods have been developed. Simple workspace decompositions can be obtained by imposing a uniform grid over $\mathcal{W}$, where each cell constitutes a decomposition region $\mathcal{W}_i$. Other decompositions can be obtained by triangulations or trapezoidations. Fig. 5.1 provides an example.



(a) uniform grid (b) triangulation (c) trapezoidation

Fig. 5.1: Various workspace decompositions.

The impact of different workspace decompositions on the computational efficiency of `SyCLoP` is studied in Section 5.4.

5.2.2 Low-Level Motion Planning

Applying `SyCLoP` to solve motion-planning problems with dynamics requires defining the trajectory-sampling strategy that is used by the low-level motion-planning layer, as described in Chapter 4. This section describes common strategies that can be applied to all control-based robotic systems.

Trajectory Sampling by Forward Propagation

As described in Section 3.2.2, trajectory sampling provides the necessary mechanism for extending the search tree $\mathcal{T}$ from a state $s \in \mathcal{T}$. Recall that a search tree $\mathcal{T}$ is extended by creating a new branch at s, which is obtained by computing a trajectory $\gamma : [0, T] \to \mathcal{S}$ that starts at s, i.e., $\gamma(0) = s$.

Forward propagation, which is based on the flow function f, allows `SyCLoP` to sam-

ple different trajectories that start at s. A common strategy for forward propagation is to select a control $u \in \mathcal{U}$ and a time duration $T \in [T_{\min}, T_{\max}]$ pseudo-uniformly at random, and compute γ as the trajectory obtained by applying the control u for T units of time when the system is at state s, as defined below:

Definition 5.2.1. (Primitive Trajectory). *Let $\mathcal{M} = (\mathcal{S}, \mathcal{U}, f)$ be a control-based system. A state $s \in \mathcal{S}$, an input $u \in \mathcal{U}$, and a time duration $T \in \mathbb{R}^{\geq 0}$ define a primitive trajectory $\gamma_{s,u,T} : [0, T] \to \mathcal{S}$, such that*

$$\forall t \in [0, T] : \gamma_{s,u,T}(t) = f(s, u, t).$$

An advantage of this random-selection strategy is that it can be applied to any system. For a car-like system, such strategy corresponds to selecting random values for the acceleration and the turning velocity of the steering wheel. A disadvantage is that the resulting trajectories are usually of poor quality. In order to improve the trajectory quality, one could design propagation strategies that are system specific. For the car example, propagation strategies can be designed that allow the car to move straight, make a smooth left or right turn, reverse, and incorporate other common driving strategies. Research in control theory [LaV06] has made significant progress in designing high-quality trajectories for different robotic systems. Such strategies are implemented by applying not just one control, but a sequence of controls for different time durations, as indicated in the following definition:

Definition 5.2.2. (Extended Trajectory). *Let $\mathcal{M} = (\mathcal{S}, \mathcal{U}, f)$ be a control-based system. A trajectory $\gamma_1 : [0, T_1] \to \mathcal{S}$, an input $u_2 \in \mathcal{U}$, and a time duration $T_2 \in \mathbb{R}^{\geq 0}$*

define an extended trajectory $\gamma : [0, T_1 + T_2] \to \mathcal{S}$*, written* $\gamma \stackrel{def}{=} \gamma_1 \circ (u_2, T_2)$*, such that*

$$\gamma(s, u, t) = \begin{cases} \gamma_1(t), & \text{if } t \in [0, T_1], \\ f(\gamma_1(T_1), u_2, t - T_1), & \text{if } t \in (T_1, T_1 + T_2]. \end{cases}$$

(Piecewise Trajectory). *A state* $s \in S$*, a sequence of inputs* $[u_i]_{i=1}^n$*,* $u_i \in \mathcal{U}$*, and a sequence of time durations* $[T_i]_{i=1}^n$*,* $T_i \in \mathbb{R}^{\geq 0}$*, define a piecewise trajectory* $\gamma : [0, T_1 + T_2 + \cdots + T_n] \to \mathcal{S}$*, such that* $\gamma \stackrel{def}{=} \gamma_{s,u_1,T_1} \circ (u_2, T_2) \circ \cdots \circ (u_n, T_n)$*.*

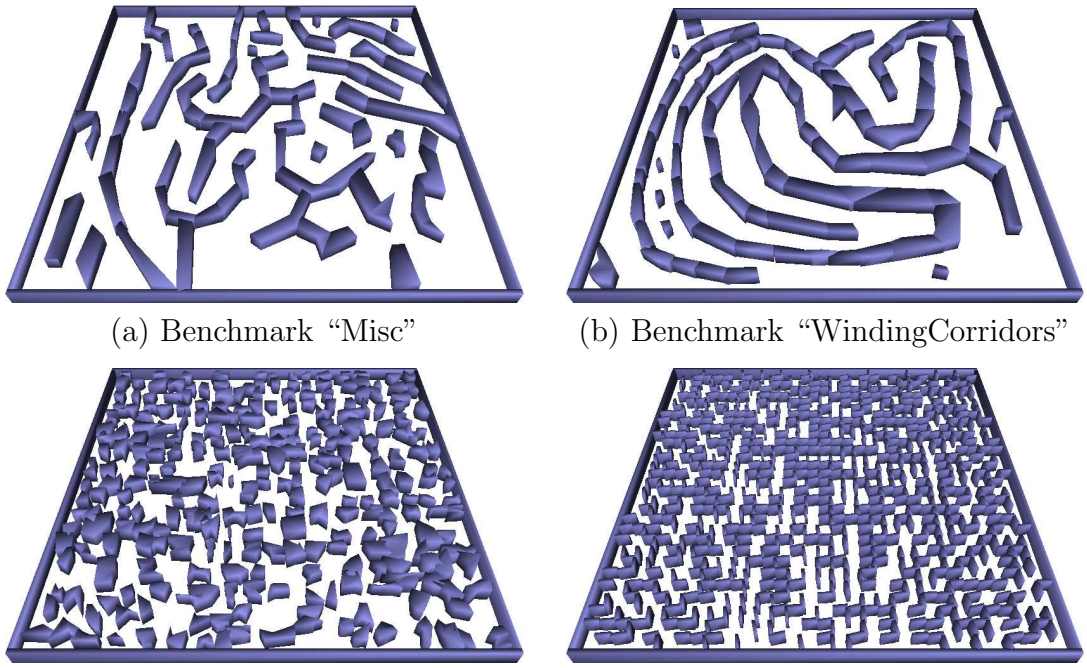
Different propagation strategies can also be combined. In such cases, each propagation strategy can be selected pseudo-uniformly at random or according to some probability distribution that is biased toward strategies that the user deems more appropriate for the motion-planning problem under consideration. `SyCLoP` can use general or system-specific strategies for trajectory sampling.

5.3 Computational Efficiency

The computational efficiency of `SyCLoP` is compared to several state-of-the-art motion-planning methods. Results presented in Section 5.3.4 show significant computational speedups of up to two orders of magnitude and highlight the benefits of synergically combining high-level discrete planning with low-level motion planning.

5.3.1 Motion-Planning Methods used in the Comparisons

`SyCLoP` is compared to `RRT` [LaV98, LK01], `ADDRRT` [JYLS05], and `EST` [HLM97, HKLR02]. Standard implementations were followed, as suggested in the respective re-



(a) Benchmark "Misc" (b) Benchmark "WindingCorridors"

(c) Benchmark "RandomObstacles" (d) Benchmark "RandomSlantedWalls"

Fig. 5.2: Several benchmarks used for the experimental comparisons of `SyCLoP`.

search articles and motion-planning books [CLH+05, LaV06]. These implementations utilize the Object-Oriented Programming System for Motion Planning (OOPSMP) framework [PBK07, PK08] and are well-tested, robust, and efficient. Every effort was made to fine-tune the performance of these motion planners for the experiments.

5.3.2 Models of Robots with Second-Order Dynamics

The robot dynamics are modeled by a set of ordinary differential equations. The models consist of a smooth (second-order) car (`SCar`), unicycle (`SUni`), and differential drive (`SDDrive`). Detailed descriptions can be found in [CLH+05, LaV06].

(`SCar`: Smooth Car).

- State Space $\mathcal{S}$: The state $s = (x, y, \theta, v, \psi)$ consists of the position $(x, y) \in \mathbb{R}^2$,

orientation $\theta \in [0, 2\Pi)$, velocity v, and steering-wheel angle ψ.

- Control Space $\mathcal{U}$: The car is controlled by setting the acceleration u_0 and the rotational velocity of the steering-wheel angle u_1.

- Equations of Motion: $\dot{x} = v\cos(\theta)$; $\dot{y} = v\sin(\theta)$; $\dot{\theta} = v\tan(\psi)/L$; $\dot{v} = u_0$; $\dot{\psi} = u_1$, where L is the distance between the front and rear axles.

(`SUni`: Smooth Unicycle).

- State Space $\mathcal{S}$: The state $s = (x, y, \theta, v, \omega)$ consists of the position $(x, y) \in \mathbb{R}^2$, orientation $\theta \in [0, 2\Pi)$, translational velocity v. and rotational velocity ω.

- Control Space $\mathcal{U}$: The unicycle is controlled by setting the translational u_0 and rotational u_1 accelerations.

- Equations of Motion: $\dot{x} = v\cos(\theta)$; $\dot{y} = v\sin(\theta)$; $\dot{\theta} = \omega$; $\dot{v} = u_0$; $\dot{\omega} = u_1$

(`SDDrive`: Smooth Differential Drive).

- State Space $\mathcal{S}$: The state $s = (x, y, \theta, \omega_L, \omega_R)$ consists of the position $(x, y) \in \mathbb{R}^2$, orientation $\theta \in [0, 2\Pi)$, and left ω_L and right-wheel ω_R rotational velocities.

- Control Space $\mathcal{U}$: The differential drive is controlled by setting the accelerations of the left u_0 and right wheels u_1.

- Equations of Motion: $\dot{x} = 0.5r(\omega_\ell + \omega_r)\cos(\theta)$; $\dot{y} = 0.5r(\omega_\ell + \omega_r)\sin(\theta)$; $\dot{\theta} = r(\omega_r - \omega_\ell)/L$; $\dot{\omega_\ell} = u_0$; $\dot{\omega_r} = u_1$; where L is the length of the axis connecting the wheel centers.

5.3.3 Motion-Planning Benchmarks

The benchmarks used in the experiments are designed to vary in type and difficulty and to test different aspects of motion planning. Fig. 5.2 provides an illustration.

Benchmark "Misc" consists of several obstacles arranged as in Fig. 5.2(a). Random motion-planning problems are created that place the robot in opposite corners of the workspace. The objective is to plan a trajectory that allows the robot to move from one position to another while avoiding collisions with the obstacles. By placing the initial and goal positions in the opposite corners of the workspace, the robot must wiggle its way through the various obstacles and the narrow passages in the workspace.

Benchmark "WindingCorridors" consists of long and winding corridors, as shown in Fig. 5.2(b). Random motion-planning problems are created by placing the robot in two different corridors, either 4 and 5 or 5 and 4 (counting from left to right), respectively. This benchmark is chosen to illustrate the efficacy of motion planning methods in solving problems where even though the initial and goal specification place the robot in neighboring places in the workspace, the solution trajectory is rather long and the robot travels through a large portion of the workspace.

Benchmark "RandomObstacles" consists of a large number of obstacles (278 obstacles) of varying sizes placed at random throughout the workspace, as shown in Fig. 5.2(c). The random placement of the obstacles creates many narrow passages, posing a challenging problem for motion-planning methods, since research [CLH+05, LaV06] has shown that many motion planners have a tendency of getting stuck in

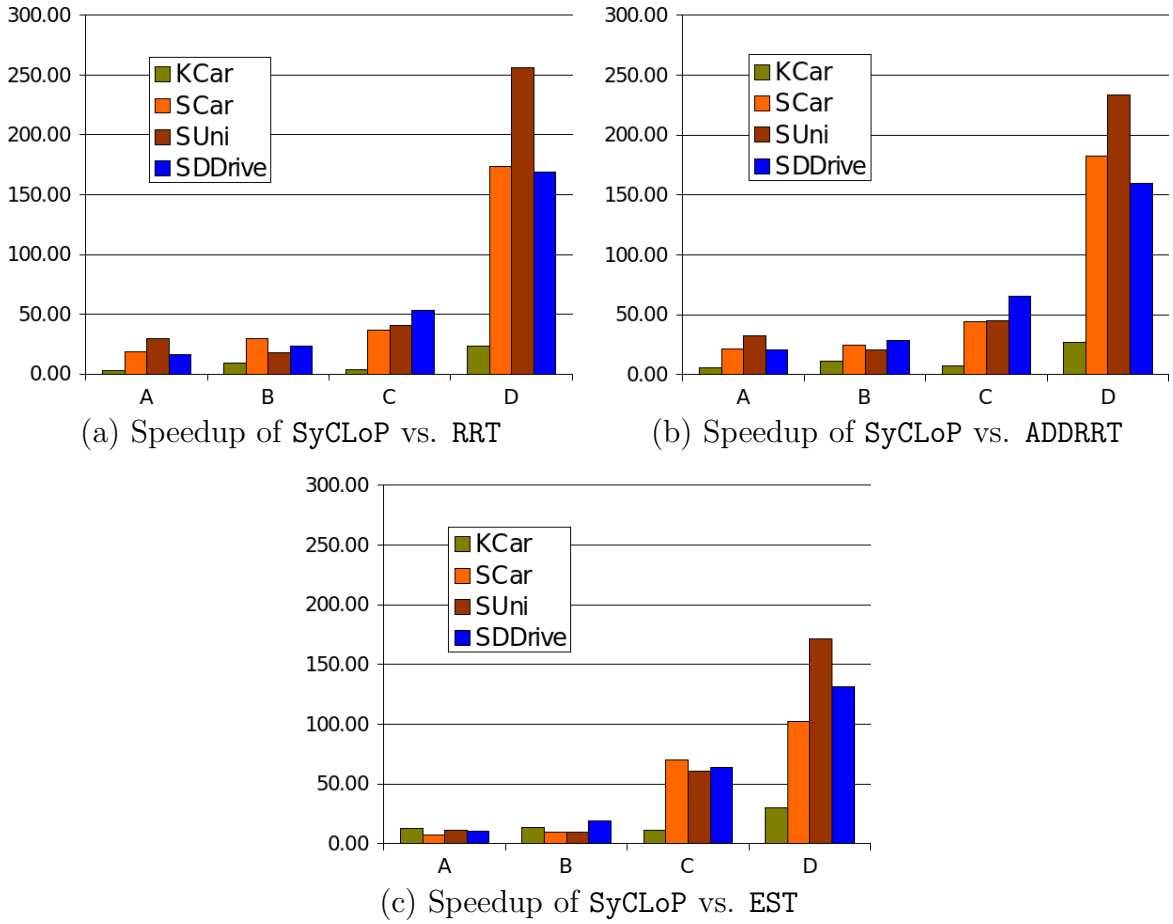
such random environments with narrow passages. Random queries place the robot in opposite sides of the workspace.

Benchmark "RandomSlantedWalls" consists of 890 obstacles resembling slanted walls, as illustrated in Fig. 5.2(d). Initially, a random maze is created using the disjoint set strategy and then only 97% of the maze walls are kept. Knocking down of the maze walls creates multiple passages in the workspace for connecting any two points. The dimensions of the remaining walls are set uniformly at random from the interval $[1/60, 1/90]$ in order to create obstacles of different sizes. Each of the remaining walls is rotated by some angle chosen at random from $[2°, 15°]$, so that the walls are aligned at different angles. This benchmark tests the efficiency of motion-planning methods in finding solutions for problems with multiple passages. Random queries place the robot in opposite sides of the workspace.

5.3.4 Experiments and Results

For each combination of benchmark (Section 5.3.3) and robot model (Section 5.3.2), 30 random motion-planning problems are generated as described in Section 5.3.3. In each instance, the computational time required to solve the query is measured. In each case, the workspace is decomposed using a 32×32 uniform grid. Rice PBC and Cray XD1 ADA clusters were used for code development. Experiments were run on ADA, where each of the processors runs at 2.2GHz and has up to 8GB of RAM.

Fig 5.3 indicates the computational speedup obtained by `SyCLoP` in comparison to the other motion-planning methods used in the experiments. Fig 5.3 shows that



(a) Speedup of SyCLoP vs. RRT

(b) Speedup of SyCLoP vs. ADDRRT

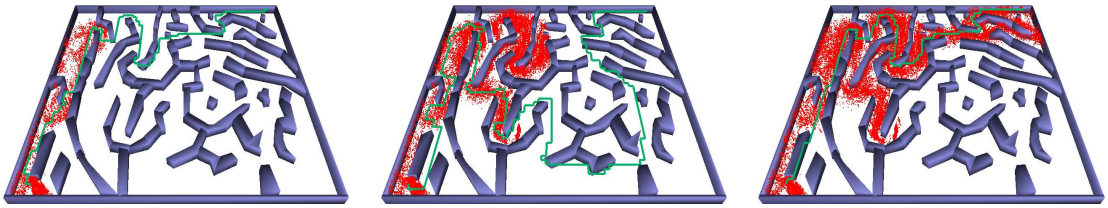
(c) Speedup of SyCLoP vs. EST

Fig. 5.3: Speedup obtained by SyCLoP when compared to RRT, ADDRRT, and EST using various robot models (KCar, SCar, SUni, SDDrive) and motion-planning benchmarks ((A)"Misc" (B) "WindingCorridors" (C) "RandomObstacles" (D) "RandomSlantedWalls").

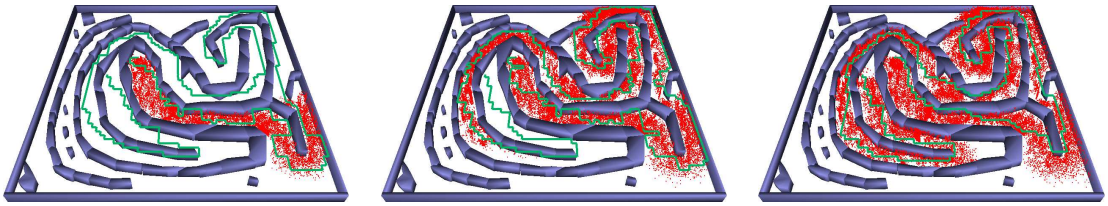
SyCLoP is consistently more efficient than RRT, ADDRRT, and EST. In fact, SyCLoP is one to two orders of magnitude faster. The next section discusses some of the reasons for the observed computational efficiency of SyCLoP.

5.3.5 A Closer Look at the State-Space Exploration

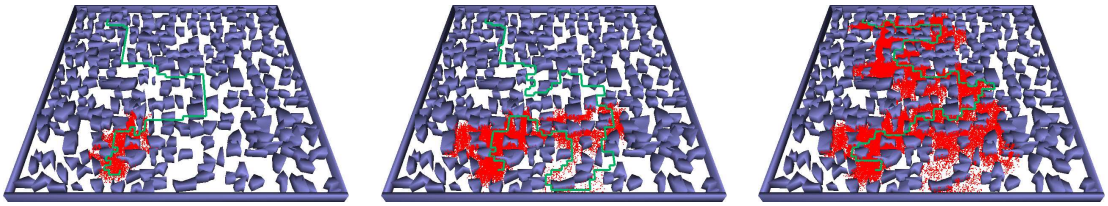
Experimental results presented in Fig. 5.3 indicate that SyCLoP offers considerable computational advantages over state-of-the-art motion-planning methods across a va-



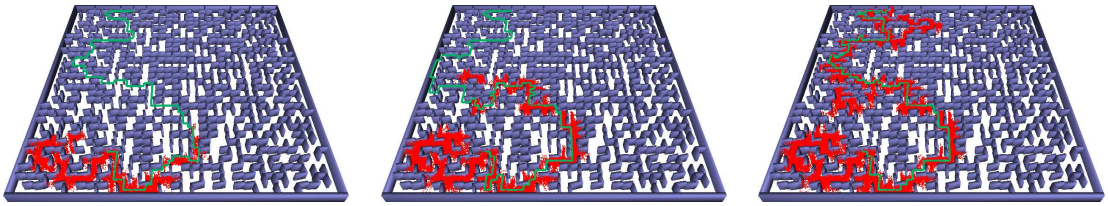
(a) Exploration of "Misc." after 2s, 4s, 8s of running time



(b) Exploration of "WindingTunnels" after 2s, 4s, 8s of running time



(c) Exploration of"RandomObstacles" after 2s, 4s, 8s of running time



(d) Exploration of "RandomSlantedWalls" after 6s, 12s, 24s of running time

Fig. 5.4: Snapshots of the tree exploration by `SyCLoP` with the smooth car (`SCar`) as the robot model. Red dots indicate state projections onto the workspace. The green line in each figure indicates the current high-level plan.

riety of challenging problems. `SyCLoP` computationally outperforms powerful motion planners, such as `RRT`, `ADDRRT`, and `EST` by an order of magnitude on easy problems and as much as two orders of magnitude on more challenging problems.

The understanding of the main reasons for the success of a motion-planning method is in general a challenging issue and subject of much research. This sec-

tion takes a closer look at the exploration done by RRT, ADDRRT, EST, and SyCLoP in order to provide some insights behind the computational efficiency of SyCLoP.

By using nearest neighbors to random states as exploration points, RRT is frequently led toward obstacles where it may remain stuck for some time [CLH+05, LaV06, JYLS05, PBC+05]. Adjusting the exploration step size of RRT, as ADDRRT does, has been shown to alleviate the problem to a certain extent but not in all situations [JYLS05]. The use of ADDRRT incurs additional computational costs, which in some cases, as those observed in this work, outweigh the benefits offered by ADDRRT. However, both in the case of RRT and ADDRRT, as the tree grows large, it becomes more frequent for the nearest neighbors to random states not to be at the frontier of the tree but instead at "inner" nodes of the tree. Consequently, especially in challenging problems where propagation is difficult, these methods end up exploring the same region many times, thus wasting computational time.

EST on the other hand suffers from a different kind of problem. EST directs the search toward less explored regions of the state space. As the tree grows large, the growth of the tree slows down as there are many regions with similar low density distributions. Consequently, EST ends up slowly expanding the tree in all possible directions, which do not necessarily bring the exploration closer to the goal region.

Although these methods have been shown to work well in a variety of settings, the main drawback is that they only use a limited amount of information to guide the exploration. There is generally much more information available to motion-planning

methods that, if properly used, can significantly speed up the computations.

The main strength of SyCLoP is the synergic combination of high-level discrete planning and low-level motion planning. As detailed in Chapter 4 and Section 5.2, the high-level planning provides SyCLoP with high-level plans that guide the low-level motion planner to extend the search tree closer to the goal. The search provides valuable feedback information that is used by SyCLoP to refine the high-level plan for the next motion-planning step. As the search progresses, the high-level plans produced by SyCLoP become more accurate and eventually result in the tree reaching the goal. Fig. 5.4 provides a snapshot of the exploration done by SyCLoP at different time intervals. The tree grows quickly and reaches the goal in a short amount of time.

5.4 Impact of Workspace Decompositions

As shown in Section 5.3.4, SyCLoP significantly reduces the computational time for solving challenging motion-planning problems with nonlinear dynamics. As discussed in Section 5.3.5, the computational efficiency of SyCLoP derives from an effective combination of high-level discrete planning and low-level motion planning. An important aspect of SyCLoP is the role of the workspace decomposition. Understanding this role is important for successful applications of SyCLoP to increasingly challenging problems. As noted earlier, the results presented in Section 5.3.4 were obtained by using a uniform grid decomposition of the workspace. In the experiments presented in this section, SyCLoP uses grid, trapezoidal, and triangular decompositions of various

granularities to solve challenging motion-planning problems with nonlinear dynamics. Workspaces used in the experiments are shown in Fig. 5.5(a) and are designed to vary in type and difficulty and provide representative problems.

5.4.1 Grid Decompositions

Grids with 1×1, 2×2, 4×4, 8×8, 16×16, 32×32, 64×64, and 128×128 cells were used for the experiments. Fig. 5.5(b) provides an illustration.

5.4.2 Triangular Decompositions

Different triangulations were obtained by varying the triangle area. Fig. 5.5(c) shows an illustration. Triangulation T1 is obtained by using Seidel's algorithm as implemented in [NM95]. It is a coarse triangulation and consists primarily of long and thin triangles. Triangulations T2, T3, T4 are computed using the industrial-strength package Triangle [She02] and are obtained by requiring the minimum angle in each triangle to be at least $20°$ and the maximum area of each triangle in T2, T3, and T4 to be at most 0.01, 0.0005, and 0.0002, respectively. Such triangulations are commonly used in mesh generations.

5.4.3 Trapezoidal Decompositions

Trapezoidal decompositions are illustrated in Fig. 5.5(d) and are computed using Seidel's algorithm as implemented in [NM95].

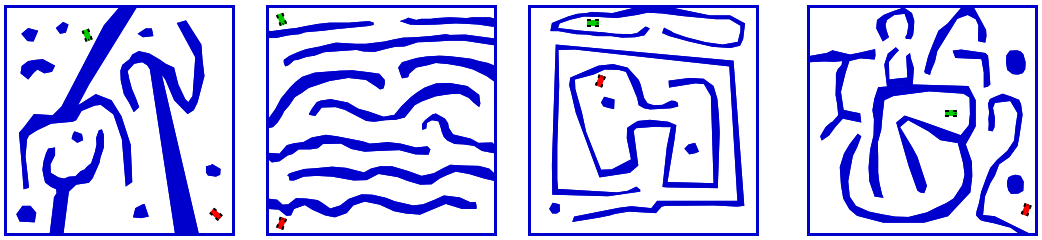
5.4.4 Conforming Delaunay Triangulations

Conforming Delaunay triangulations have been widely used in computational geometry and are similar to Delaunay triangulations for a set of points, which maximize the minimum angle among all possible triangulations, but could potentially differ in some places to take into account polygonal edge constraints by adding additional vertices [She02]. Although in some theoretical cases $O(n^3)$ new vertices are required, in practice the bound is linear [She08]. Fig. 5.5(e) illustrates the conforming Delaunay triangulations as computed by the industrial-strength package Triangle [She02].

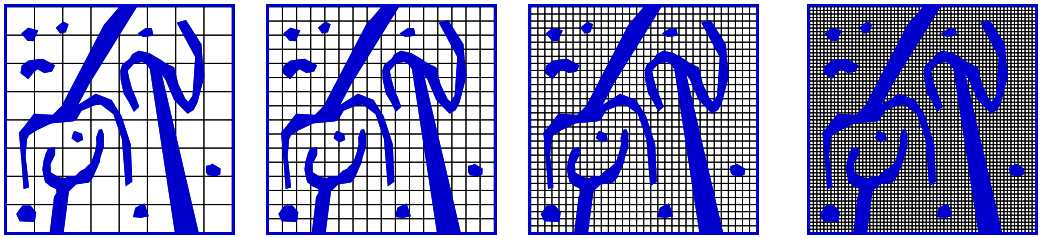
5.4.5 Results

For each combination of workspace, workspace decomposition, and robot model, `SyCLoP` solves 30 random motion-planning problems. The computational efficiency of `SyCLoP` for a given combination is measured as the average time to solve the problems.

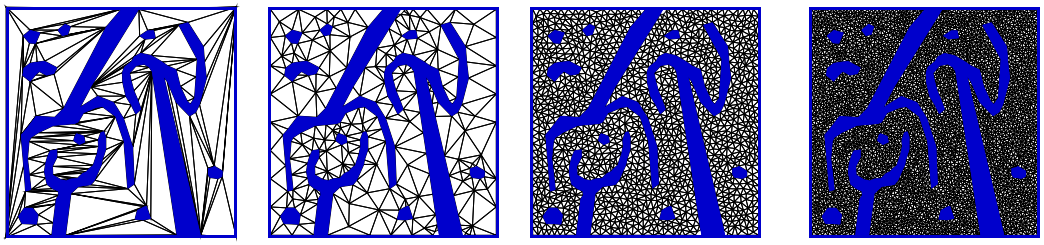
Fig. 5.6 shows that the granularity of the workspace decomposition directly impacts the computational efficiency. `SyCLoP` is faster when the decomposition is neither too fine- nor too-coarse grained. When the decomposition is too fine-grained, the computational advantages offered by the interplay between the high-level planning and motion planning are outweighed by the computational cost associated with high-level plan computations and updates to exploration estimates. On the other end of the spectrum, when the decomposition is too coarse-grained there is no significant advantage by using the high-level plan to guide the motion planner. Finding the right level



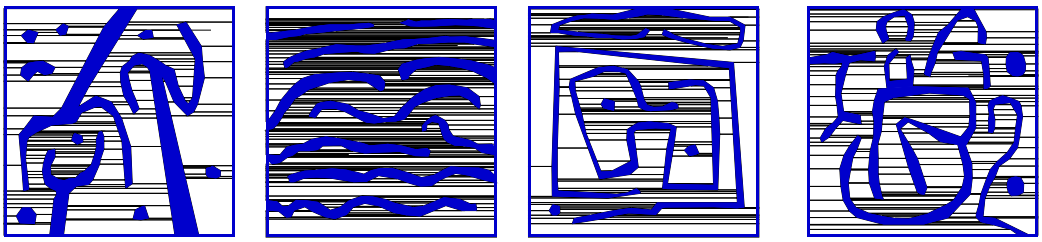
(a) Workspaces used for the experiments



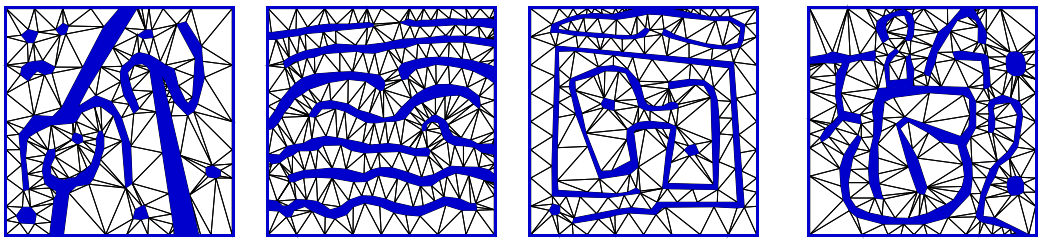
(b) Illustration of some grid decompositions, 8×8, 16×16, 32×32, 64×64



(c) Illustration of triangulations T1, T2, T3, T4

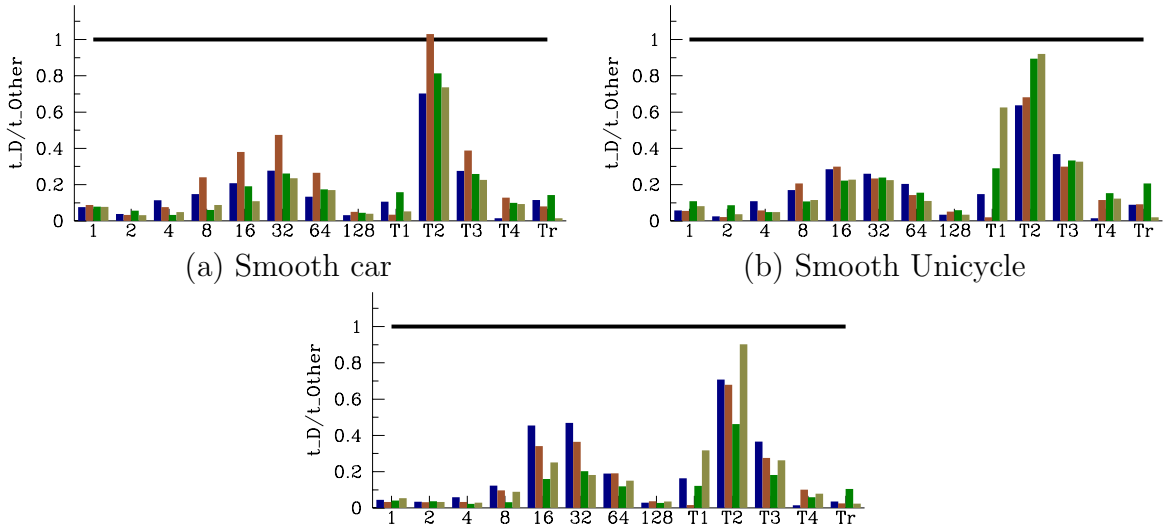


(d) Illustration of trapezoidal decompositions for each workspace



(e) Illustration of conforming Delaunay triangulations for each workspace

Fig. 5.5: (a) Workspaces used for the experiments. Each figure also illustrates a typical query for a second-order car model with the initial state shown in green and the goal state shown in red. (b-e) Illustrations of different workspace decompositions.



(a) Smooth car (b) Smooth Unicycle

(c) Smooth Differential Drive

Fig. 5.6: Bars (from left to right) correspond to the results when using different decompositions (x-axis) on the workspaces in Fig. 5.5. t_D denotes the computational time of `SyCLoP` when using a conforming Delaunay triangulation. t_{Other} denotes the computational time of `SyCLoP` when using a different decomposition. Decompositions 1, 2, 4, 8, 16, 32, 64, and 128 denote grid decompositions. Decompositions T1, T2, T3, T4 denote triangular decompositions. Decomposition Tr denotes trapezoidal decomposition.

of granularity to take full advantage of the computational benefits offered by the interplay between high-level discrete planning and low-level motion planning could further increase the computational efficiency of `SyCLoP` but can require extensive fine-tuning.

Fig. 5.6 also shows that significant computational efficiency can instead be obtained with no fine-tuning by using conforming Delaunay triangulations. In the context of `SyCLoP`, conforming Delaunay triangulations provide a natural workspace decomposition that achieves a balance between coarse- and fine-grained decomposition, and thus allows a remarkably efficient interplay between high-level discrete planning and low-level motion planning in `SyCLoP`.

5.4.6 Advantages of Delaunay Triangulations

It is also interesting to note that similar observations have been made in the context of particle finite element methods, where it has been shown that methods based on Delaunay triangulations are in many cases better suited that grid-based discretizations [L9̈6]. In particular, iterative numerical methods such as the particle finite element method [OnIDPA04] rely on Delaunay triangulations to connect moving fluid particles with a finite element mesh [DPIOA07]. The fast regeneration of the mesh at every time step is crucial to the success of the Langrangian flow formulation. The Extended Delaunay Tesselation [ICO03] allows to generate non-standard meshes combining different geometrical shapes in the same mesh. Moreover, EDT is better suited over grid discretization methods [L9̈6] in efficiently computing boundary domains by simplifying the correct identification of boundary nodes.

Delaunay triangulations also play an important role in surface modeling across different application domains, including robotics, computer graphics, geographic data processing, computer vision, computer aided design, molecular and medical data visualization, and hydrodynamics [AD97, GKM+01, DPIOA07]. Delaunay triangulations offer efficient three-dimensional terrain surface modeling in simulations of off-road vehicles where wheel-surface contact geometry is needed to obtain ground-reaction and friction forces [AD97]. Delaunay triangulations also allow to efficiently keep track of the near environment of an autonomous digital agent [GKM+01] or to efficiently update the free surface of a moving fluid when simulating fluid dynamics [DPIOA07].

Chapter 6

Motion Planning for Hybrid Systems

This chapter describes how the multi-layered approach SyCLoP can be applied to effectively solve challenging motion-planning problems for hybrid systems. Hybrid systems are often part of sophisticated embedded controllers used in robots exploring unknown and possibly hazardous environments. Hybrid systems go beyond continuous models by employing discrete logic to instantaneously modify the underlying robot dynamics to respond to mishaps or unanticipated changes in the environment. While the combination of discrete logic and continuous dynamics poses significant challenges to current motion-planning methods, this combination is particularly well-suited to SyCLoP, which synergically combines high-level discrete planning with low-level motion planning. Experiments show SyCLoP obtains significant computational speedups of one to two orders of magnitude in comparison to state-of-the-art motion planners.

6.1 Introduction

Nowadays robots are expected to explore unknown or hazardous environments, quickly modifying their dynamics to respond to mishaps or unanticipated changes in the environment. For example, a vehicle may be required to modify the dynamics over different terrains due to safety issues, and a reconfigurable robot may change its

shape and use different gaits to climb, crawl, or walk. Such changes in the dynamics are often realized by instantaneously switching to a different operating mode.

A challenging yet important problem is the development of motion planners for these hybrid robotic systems. The challenge lies in that a hybrid system combines discrete and continuous dynamics by associating continuous dynamics with each operating mode and using discrete logic to switch between modes.

Motion planning has generally focused on continuous systems. The use of RRT [LaV98, LK01, LaV06] has recently shown promise as a motion-planning method for hybrid systems with few discrete modes [EKK04, BF04, KEK05, ND07]. The applicability of RRT to more complex hybrid systems, especially systems with a large number of discrete modes and transitions, remains challenging for three main reasons. First, an RRT relies heavily on distance metrics that should indicate how easily the hybrid system can transition from one state to another. Unfortunately, the definition of such distance metrics is difficult, even in the case of continuous systems, since it is not even clear that a distance metric can express this property [ABD+98b]. Second, the growth of an RRT has been shown to significantly slow down as the number of nodes in the tree increases [EKK04, CLH+05, LaV06], limiting the ability of RRT to successfully explore the continuous state spaces of systems with a large number of discrete modes and transitions. Third, the exploration of continuous state spaces by an RRT is local and prone to frequently getting stuck in certain regions [EKK04, PBC+05, PK05, CLH+05].

The limitations of current motion planners in coping with the additional compu-

tational challenges posed by hybrid systems motivates the need for the development of new and effective methods. The combination of high-level discrete planning and low-level motion planning in the multi-layered approach `SyCLoP` is particularly well-suited for hybrid systems, which naturally combine discrete logic with the underlying robot dynamics. As shown in the rest of this chapter, `SyCLoP` obtains significant computational speedups of two orders of magnitude in comparison to state-of-the art motion planners in solving challenging motion-planning problems for hybrid systems.

6.2 Hybrid Systems

A hybrid system combines discrete and continuous dynamics. Continuous dynamics are associated with each mode, and discrete logic determines how to switch between modes. In this thesis, hybrid systems are modeled by hybrid automata [ACH+95].

Definition 6.2.1. (Hybrid Automata). *A hybrid automaton is a tuple*

$$\mathcal{H} = (\mathcal{S}, \textsc{Valid}, E, \textsc{Guard}, \textsc{Jump}, \mathcal{U}, f), \text{ where}$$

- $\mathcal{S} = \mathcal{Q} \times \mathcal{X}$ *is the Cartesian product of the discrete and continuous state spaces;*
- $\mathcal{Q}$ *is a discrete and finite set;*
- $\mathcal{X} = \{\mathcal{X}_q : q \in \mathcal{Q}\}$ *is the collection of the continuous state spaces, where* $\mathcal{X}_q$ *is the continuous state space associated with* $q \in \mathcal{Q}$*;*
- $\textsc{Valid} : \mathcal{S} \to \{\top, \bot\}$ *is a state-constraint function, i.e.,* $\textsc{Valid}(s) = \top$ *iff* $s \in \mathcal{S}$ *satisfies the state constraints;*

- *$E \subseteq \mathcal{Q} \times \mathcal{Q}$ is the set of discrete transitions between discrete states;*

- $\textsc{Guard} = \{\textsc{Guard}_{(q_i,q_j)} : (q_i, q_j) \in E\}$, *where* $\textsc{Guard}_{(q_i,q_j)} \subseteq \mathcal{X}_{q_i}$ *is the guard condition associated with* $(q_i, q_j) \in E$*;*

- $\textsc{Jump} = \{\textsc{Jump}_{(q_i,q_j)} : (q_i, q_j) \in E\}$, *where* $\textsc{Jump}_{(q_i,q_j)} : \textsc{Guard}_{(q_i,q_j)} \to \mathcal{X}_{q_j}$ *is the jump function associated with* $(q_i, q_j) \in E$*;*

- $\mathcal{U} = \{\mathcal{U}_q : q \in \mathcal{Q}\}$, *where* $\mathcal{U}_q$ *is the control space associated with* $q \in \mathcal{Q}$*;*

- $f : \mathcal{S} \times \mathcal{U} \times \mathbb{R}^{\geq 0} \to \mathcal{S}$ *is a flow function that simulates the hybrid-system dynamics when an input is applied to the hybrid-system for a certain time duration.*

The state of the hybrid automaton is a tuple $s = (q, x) \in \mathcal{S}$ that describes both the discrete and the continuous components. The invariant $\textsc{Valid} : \mathcal{S} \to \{\top, \bot\}$ indicates for each state $s \in \mathcal{S}$ whether or not s satisfies the state constraints. The set $E \subseteq \mathcal{Q} \times \mathcal{Q}$ describes which transitions are possible from one mode to another. A discrete transition occurs at a state $s = (q, x) \in \mathcal{S}$ iff s satisfies a guard condition, i.e., $\textsc{Guard}_{(q,q')}(x) = \top$ for some $q' \in \mathcal{Q}$. When a discrete transition occurs, then the current state of the hybrid system is set to a new state $s' = (q', x')$ according to the jump function, where $x' = \textsc{Jump}_{(q,q')}(x)$. The input space $\mathcal{U}_q$ can represent controls, nondeterminism, or uncertainties. As in the case of control-based systems (see Chapter 5), this thesis treats the dynamics of the hybrid system as a black box $f : \mathcal{S} \times \mathcal{U} \times \mathbb{R}^{\geq 0} \to \mathcal{S}$, where for each $s = (q, x) \in \mathcal{U}$, $u \in \mathcal{U}_q$, and $t \geq 0$, $f(s, u, t)$ outputs the new state obtained by following the dynamics when the hybrid

system is at s and u is applied for t units of time. This allows for modeling general nonlinear dynamics. In fact, the only requirement is the ability to simulate the dynamics. When the dynamics in each $q \in \mathcal{Q}$ is given by a set of differential equations $g_q : \mathcal{X}_q \times \mathcal{U}_q \to \dot{\mathcal{X}}_q$, closed-form solutions (if known) or numerical integration can be used for the simulation. Since $\mathcal{X}_q$ can include derivatives of different orders (e.g., velocity, acceleration), g_q can be nonlinear. A hybrid-system trajectory consists of continuous trajectories interleaved with discrete transitions.

Definition 6.2.2. (Continuous Trajectory). *A state $s = (q, x) \in \mathcal{S}$, a time $T \geq 0$, and an input $u \in \mathcal{U}_q$ define a continuous trajectory $\Psi_{s,u,T} : [0, T] \to \mathcal{X}_q$, where*

- $\Psi_{s,u,T}(t) \in \mathcal{X}_q - \{\textsc{Guard}_{(q,q')} : (q, q') \in E\}$, *for* $t \in [0, T)$; *and*
- $(q, \Psi_{s,u,T}(t)) = f(s, u, t)$, *for* $t \in [0, T]$.

(Discrete Transition). *For any state $s = (q, x) \in S$, define*

$$\chi(q, x) = \begin{cases} \chi\left(q', \textsc{Jump}(q, q')(x)\right), & x \in \textsc{Guard}_{(q,q')} \text{ for some } (q, q') \in E, \\ (q, x), & \text{otherwise.} \end{cases}$$

(Continuous Trajectory + Discrete Transition). *The hybrid-system trajectory $\Upsilon_{s,u,T} : [0, T] \to \mathcal{S}$, defined as*

$$\Upsilon_{s,u,T}(t) = \begin{cases} (q, \Psi_{s,u,T}(t)), & 0 \leq t < T \\ \chi(q, \Psi_{s,u,T}(t)), & t = T \end{cases}$$

ensures that discrete transitions at time T, if they occur, are followed.

(Trajectory Extension). *The extension of a trajectory $\Phi : [0, T] \to \mathcal{S}$ by applying*

to $\Phi(T)$ *the input* $u' \in \mathcal{U}$ *for a duration of time* $T' \geq 0$ *is written as* $\Phi \circ (u', T')$*, and it is another trajectory* $\Xi : [0, T + T'] \to \mathcal{S}$ *defined as*

$$\Xi(t) = \begin{cases} \Phi(t), & t \in [0, T] \\ \Upsilon_{\Phi(T), u', T'}(t - T), & t \in (T, T + T']. \end{cases}$$

(Hybrid-System Trajectory). *A hybrid-system trajectory* $\gamma : [0, T] \to \mathcal{S}$ *is defined by a state* $s \in \mathcal{S}$*, a sequence* $u_1, \ldots, u_k \in \mathcal{U}$ *of inputs, and a sequence* $T_1, \ldots, T_k \in \mathbb{R}^{\geq 0}$ *of time durations, where* $T = \sum_{i=1}^{k} T_i$ *and* $\gamma \stackrel{def}{=} \Upsilon_{s, u_1, T_1} \circ (u_2, T_2) \circ \cdots \circ (u_k, T_k)$.

The continuous trajectory $\Psi_{s,u,T}$ is thus obtained by applying the control u to the state s for a duration of T units of time. Moreover, $\Psi_{s,u,T}$ never reaches a guard condition during the time interval $[0, T)$. The trajectory $\Upsilon_{s,u,T}$ is similar to $\Psi_{s,u,T}$, but, unlike $\Psi_{s,u,T}$, $\Upsilon_{s,u,T}$ follows the discrete transitions at time T when they occur.

We note that in the hybrid-system benchmarks used in this thesis the discrete transitions are considered urgent, i.e., a discrete transition is immediately taken once a guard condition is satisfied. There is, however, no inherent limitation of `SyCLoP` in dealing with non-urgent discrete transitions. When discrete transitions are non-urgent, enabled discrete transitions could be taken nondeterministically with some probability or taken only when the invariant is invalid or a combination of both.

6.3 Applying `SyCLoP` to Motion Planning for Hybrid Systems

The combination of high-level discrete planning and low-level motion planning in `SyCLoP`, described in Chapter 4, is particularly well-suited for hybrid systems, which

naturally combine discrete logic with the underlying robot dynamics. This section describes the high-level discrete model used by `SyCLoP` in the case of motion-planning for hybrid systems. This section also describes how to extend the search tree from a selected state. The high-level discrete model and the tree-extension procedure are similar to what `SyCLoP` used in motion planning with dynamics, as described in Chapter 5. The main distinction is that in the case of hybrid systems these procedures need to take into account the combination of discrete logic with the robot dynamics.

6.3.1 High-Level Discrete Model of Motion-Planning Problem

As discussed in Section 4.2, the discrete model of the motion-planning problem provides a simplified high-level planning layer that effectively guides the low-level motion planner as it extends the search tree $\mathcal{T}$. The discrete model in the case of motion-planning for hybrid systems is computed based on a partition of the continuous state spaces associated with the modes of the hybrid system. For each $q \in \mathcal{Q}$, let

$$\mathcal{R}(q) = \{\mathcal{R}_1(q), \ldots, \mathcal{R}_{n_q}(q)\}$$

denote a partition of the continuous state space $\mathcal{X}_q$. The partition of $\mathcal{X}_q$ is usually computed on a low-dimensional projection, as described in Section 5.2.1. Based on this partition, as described in Section 5.2.1, a the graph $G_{\mathcal{R}(q)} = (V_{\mathcal{R}(q)}, E_{\mathcal{R}(q)})$ is constructed as follows:

- $V_{\mathcal{R}(q)} = \{v(\mathcal{R}_i(q)) : \mathcal{R}_i(q) \in \mathcal{R}(q)\}$, and
- $E_{\mathcal{R}(q)} = \{(v(\mathcal{R}_i(q)), v(\mathcal{R}_j(q))) : \mathcal{R}_i(q)$ is adjacent to $\mathcal{R}_j(q)\}$.

In this way, $G_{\mathcal{R}(q)} = (V_{\mathcal{R}(q)}, E_{\mathcal{R}(q)})$ provides a simplified high-level planning layer of the continuous dynamics associated with the mode $q \in \mathcal{Q}$.

`SyCLoP` uses the graphs $G_{\mathcal{R}(q)} = (V_{\mathcal{R}(q)}, E_{\mathcal{R}(q)})$ associated with each $q \in \mathcal{Q}$ and the discrete transitions E of the hybrid system to construct the graph $G_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ of the discrete model for the hybrid system. More specifically,

- $V_{\mathcal{R}} = \bigcup_{q \in \mathcal{Q}} V_{\mathcal{R}(q)}$, and
- $E_{\mathcal{R}} = \left(\bigcup_{q \in Q} E_{\mathcal{R}(q)}\right) \cup \{(v(\mathcal{R}_i(q')), v(\mathcal{R}_j(q''))) : \textsc{Trans}(\mathcal{R}_i(q'), \mathcal{R}_j(q'')) = \top\}$,

where the function $\textsc{Trans}(\mathcal{R}_i(q'), \mathcal{R}_j(q''))$ determines whether or not there is a discrete transition from a state in $\mathcal{R}_i(q')$ to a state in $\mathcal{R}_j(q'')$. This allows $G_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ to capture both the discrete logic and the continuous dynamics of the hybrid system.

Note that the computation of $\textsc{Trans}(\mathcal{R}_i(q'), \mathcal{R}_j(q''))$ depends on the definition of guard $\textsc{Guard}_{(q',q'')} : \mathcal{X}_{q'} \to \{\top, \bot\}$ and jump $\textsc{Jump}_{(q',q'')} : \mathcal{X}_{q'} \to \mathcal{X}_{q''}$ functions. When it is computationally infeasible or expensive to determine if there is a discrete transition from $\mathcal{R}_i(q')$ to $\mathcal{R}_j(q'')$, i.e.,

$$\exists x \in \mathcal{R}_i(q') : \textsc{Guard}_{(q',q'')}(x) = \top \wedge \textsc{Jump}_{(q',q'')}(x) \in \mathcal{R}_j(q''),$$

the definition of $\textsc{Trans}$ can be relaxed. In fact, it is only required that no false negatives are returned, i.e., $\textsc{Trans}(\mathcal{R}_i(q'), \mathcal{R}_j(q'')) = \bot$ when there is a discrete transition from $\mathcal{R}_i(q')$ to $\mathcal{R}_j(q'')$. A false negative would cause `SyCLoP` to miss an edge in $G_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}})$, and as a result, not be able to compute any feasible high-level plans. False positives, i.e., $\textsc{Trans}(\mathcal{R}_i(q'), \mathcal{R}_j(q'')) = \top$ when there is in no discrete transition from $\mathcal{R}_i(q')$ to $\mathcal{R}_j(q'')$, are, however, allowed. A false positive would add

a spurious edge to $E_{\mathcal{R}}$, which could lead to the computation of an infeasible high-level plan. However, as the search progresses, the weight estimates associated with the spurious edge would indicate that such edge should not be included in future high-level plans as it is not helping `SyCLoP` to extend the search tree $\mathcal{T}$. By allowing false positives, the computation of $\textsc{Trans}(\mathcal{R}_i(q'), \mathcal{R}_j(q''))$ can be greatly simplified. In particular, it can be computed as

- $\textsc{Trans}(\mathcal{R}_i(q'), \mathcal{R}_j(q'')) = \top \iff \exists x \in \mathcal{R}_i(q') : \textsc{Guard}_{(q',q'')}(x) = \top$ or
- $\textsc{Trans}(\mathcal{R}_i(q'), \mathcal{R}_j(q'')) = \top \iff (q', q'') \in E$.

6.3.2 Low-Level Motion Planning

Applying `SyCLoP` to solve motion-planning problems for hybrid systems requires taking into account the discrete logic of the hybrid system during the tree-extension step. In particular, accurate simulations require detecting whether a discrete transition will occur, and if it does occur, then localizing the time when the discrete transition occurs. This section describes a common approach for generating trajectories and detecting and localizing discrete events.

Extending the Search Tree

As described in Section 4.5.3, the search tree $\mathcal{T}$ is extended from a given state $s \in \mathcal{S}$ by first generating a trajectory $\gamma : [0, T] \to \mathcal{S}$ that starts at s and then keeping only the valid portion $\gamma_{\text{valid}} : [0, K] \to \mathcal{S}$, $K \leq T$, of the initial trajectory γ.

Let $s = (q, x) \in \mathcal{S}$ be the state selected from the search-tree $\mathcal{T}$. The actual extension of $\mathcal{T}$ from $s = (q, x)$ is computed by the ExtendSearchTree function. In the case of motion-planning for hybrid systems, an input control $u \in U_q$, which could be selected pseudo-uniformly at random or according to some other strategy (see Section 6.4.1), is applied to s for a short duration of time $T > 0$. The function ExtendSearchTree simulates the continuous and discrete dynamics of the hybrid system to obtain the resulting hybrid-system trajectory $\Upsilon_{s,u,T}$, as in Definition 6.2.2. Pseudocode is given in Algo. 3.

Algorithm 3 ExtendSearchTree

Input: $\mathcal{H} = (\mathcal{S}, \textsc{Valid}, E, \textsc{Guard}, \textsc{Jump}, \mathcal{U}, f)$: hybrid system
$s = (q, x) \in S$: starting state
$\epsilon \in \mathbb{R}^{>0}$: integration step
$n_{\text{steps}} \in \mathbb{N}$: number of integration steps
Output: The new state obtained at the end of propagation

```
u ← sample control from U_q
x_0 ← x
for i = 1, 2, ..., n_steps do
    x_i ← ∫_0^ε f_q(x_{i-1}, u)
    if Valid((q, x_i)) = ⊥ then                    ◇current state is not valid
        return (q, x_{i-1})                         ◇ previous valid state
    if (q, x_i) ∈ G_(q,q_new) for some q_new ∈ Q then
        (x_loc, T) ← localize discrete event in time interval ((i − 1) ∗ ε, i ∗ ε]
        return J_(q,q_new)(q, x_loc)
return (q, x_i)
```

The forward propagation follows the continuous dynamics f_q associated with $q \in Q$ and is usually computed based on numerical integration of the ordinary differential equations associated with f_q. This thesis uses 8-th order Prince-Dormand Runge-Kutta numerical integration with adaptive step-size control as implemented in GSL

[GDT+06]. The forward propagation is an iterative procedure. Let n_{steps} denote the number of propagation steps and let $\epsilon > 0$ denote the integration step. Initially, $x_0 = x$ (Algo. 3:2). During the i-th iteration, $x_i \in X_q$ is obtained by numerically integrating the differential equations $f_q(x_{i-1}, u)$ for ϵ units of time (Algo. 3:4).

If (q, x_i) is not valid, then the forward propagation is terminated (Algo. 3:5–6). The previous valid state (q, x_{i-1}) is returned as the new state s_{new} obtained at the end of the forward propagation. The valid hybrid-system trajectory corresponds then to $\Psi_{s,u,T}$ (see Definition 6.2.2), where $T = (i-1) * \epsilon$.

If (q, x_i) is valid, the simulation checks whether the state (q, x_i) satisfies any guard condition, i.e., $(q, x_i) \in G_{(q,q_{\text{new}})}$ for some $q_{\text{new}} \in Q$. If a guard condition is satisfied, then a discrete event has occurred in the time interval $(i - 1 * \epsilon, i * \epsilon]$ (Algo. 3:7). This stage, commonly known as *event detection*, is followed by the *event localization* stage, which localizes the earliest time $T \in ((i-1) * \epsilon, i * \epsilon]$ the guard condition is satisfied (Algorithm 3:8). Variants of bisection or bracketing algorithms, as those found in the classical numerical literature, are commonly employed for the event detection [EKP01]. Once the event is localized, the propagation stops and the corresponding discrete transition is applied to obtain the new state s_{new} (Algo. 3:9). The valid hybrid-system trajectory corresponds then to $\Upsilon_{s,u,T}$ (see Definition 6.2.2).

We note that, due to limitations of floating-point arithmetic, as with any other numerical method in computational mathematics, including symbolic techniques for linear hybrid systems, there will be round-off errors in the simulation of the con-

tinuous dynamics and the event detection and localization of discrete transitions. The approach followed by `SyCLoP` to deal with such numerical errors is similar to the approach followed by other numerical methods for hybrid-system falsification [BF04, KEK05, ND07], which choose the integration step $\epsilon > 0$ to be small in order to minimize such errors. For certain hybrid systems with linear guard descriptions, it is also possible to use more accurate event detection and localization algorithms, such as those surveyed and developed in [EKP01], which come asymptotically close to the boundary of the guard set without overshooting it.

6.4 Computational Efficiency

The computational efficiency of `SyCLoP` is compared to several motion-planning methods that have been developed for hybrid systems. Results presented in Section 6.4.3 show significant computational speedups of up to two orders of magnitude obtained by `SyCLoP`. Similar to the case of motion-planning with nonlinear dynamics (see Section 5.3), this thesis also studies the impact of various decompositions on the computational efficiency of `SyCLoP` in the case of motion planning for hybrid systems.

6.4.1 A Hybrid Robotic System Navigation Benchmark

The hybrid system used in the experiments consists of an autonomous robotic vehicle, whose underlying dynamics change discretely depending on terrain conditions. The choice of this specific system is to provide a concrete, scalable benchmark in

which the competitiveness of `SyCLoP` can be tested. This hybrid-system benchmark, which is motivated by robotics applications, is constructed based on a scalable navigation benchmark proposed in [FI04]. A given environment is divided into $n \times n$ equally sized cells. The hybrid system associates one mode $q_i \in \mathcal{Q}$ with each cell C_i. For each mode, the associated dynamics is specified by a set of ordinary differential equations. A discrete transition $(q_i, q_j) \in E$ occurs when the hybrid system moves from C_i to C_j. When the discrete transition occurs, velocity components of the current continuous state of the robotic vehicle are set to zero. While the navigation benchmark proposed in [FI04] uses linear dynamics, the benchmark in this thesis uses second-order dynamics that are commonly used for modeling cars, differential drives, and unicycles. Details of these models can be found in [CLH+05, LaV06] and Section 5.3.2.

Autonomous driver models

A trajectory-sampling strategy $\textsc{SampleTraj}(\mathcal{P}, s)$ (see Section 3.2.2) could be thought of as playing the role of the automatic driver. At each state, the driver selects the controls that it applies to the system. As described in Section 5.3.2, each vehicle model is controlled by two input values, u_0 and u_1, corresponding to

- `SCar`: acceleration (u_0) and steering-wheel velocity (u_1);
- `SUni`: translational acceleration (u_0) and rotational acceleration (u_1);
- `SCar`: rotational acceleration of left (u_0) and right (u_1) wheels.

The driver models used in this thesis consist of simple if-then-else statements that depend on the state values and the dynamics associated with each mode of the hybrid system. In the first model, `RandomDriver`, u_0 and u_1 are selected pseudo-uniformly at random from $[-\max_0, \max_0]$ and $[-\max_1, \max_1]$, respectively. In the second model, `StudentDriver`, the driver follows an approach similar to stop-and-go. When the speed is close to zero, `StudentDriver` selects u_0 and u_1 as in `RandomDriver`. Otherwise, `StudentDriver` selects controls that reduce the speed. The third model, `HighwayDriver` attempts to maintain the speed within acceptable lower and upper bounds. When the speed is too low, `HighwayDriver` selects controls that increase the speed. When the speed is too high, `HighwayDriver` selects controls that slow down the vehicle. Otherwise, `HighwayDriver` selects controls that do not change the speed considerably. For completeness, a succinct description of the selection strategy for u_0 and u_1 for each driver model and each second-order model is provided in Algo. 4.

Algorithm 4 Autonomous Driver Models

`RandomDriver` $f(a, i, c, L, R)$: return $\text{rnd}(-\max_i, \max_i)$

`StudentDriver` $f(a, i, c, L, R)$:
 if $a \in (0.2, 1]$ then return $\text{rnd}(-L\max_i, R(c-1)\max_i)$
 elif $a \in [-1, -0.2)$ then return $\text{rnd}(R(1-c)\max_i, L\max_i)$
 else return $\text{rnd}(-\max_i, \max_i)$

`HighwayDriver` $f(a, i, c, L, R)$: B=$\{0.4, 0.6, 0.8, 1.0\}$
 if $\exists b \in B \wedge a \in (b-0.2, b]$ then return $\text{rnd}(-Lb\max_i, R(c-b)\max_i)$
 elif $\exists b \in B \wedge a \in [-b, 0.2-b)$ then return $\text{rnd}(R(b-c)\max_i, Lb\max_i)$
 else return $\text{rnd}(-\max_i, \max_i)$
SCar: $u_0 = f(v/v_{\max}, 0, 0.2, 1, 1)$; $u_1 = \text{rnd}(-\max_1, \max_1)$
SUni: $u_0 = f(v/v_{\max}, 0, 0.2, 1, 1)$; $u_1 = f(\omega/\omega_{\max}, 1, 0.2, 1, 1)$
SDDrive: $u_0 = f(a, 0, 1.2, 0, 0.25)$; $u_1 = -f(a, 1, 1.2, 0, 0.25)$; $a = \frac{\omega_\ell + \omega_r}{2\omega_{\max}}$

Modes and discrete transitions

The continuous dynamics associated with each mode $q \in \mathcal{Q}$ is selected pseudo-uniformly at random from `SCar`, `SUni`, and `SDDrive`. The set of discrete transitions E is created using a strategy similar to maze generation based on Kruskal's algorithm. Initially, E is empty and walls are placed between each pair of neighboring cells C_i and C_j. Then, walls are visited in some random order. If the cells divided by the current wall belong to distinct sets, then the wall is removed and the two sets are joined. At the end, each remaining wall is kept with probability $p = 0.9$ to allow for more than one passage from one cell to another. Each time a wall that separates some C_i from C_j is removed, discrete transitions (q_i, q_j) and (q_j, q_i) are added to E.

6.4.2 Experiments

Experiments are performed using the hybrid robotic system described in Section 6.4.1. A problem instance is obtained by fixing the number of modes $|\mathcal{Q}| = n \times n$ and the driver model to `RandomDriver`, `StudentDriver`, or `HighwayDriver`. For each problem instance, 40 random motion-planning problems are created. Each motion-planning problem is created by selecting pseudo-uniformly at random one cell as the initial place and another cell as the goal.

The hybrid robotic system is made increasingly complex by increasing the number of modes. This thesis presents experiments with over one million modes. An important part of experiments is the comparison with previous related work. The

closest work that SyCLoP can be compared to is the application of RRT to hybrid systems [EKK04, KEK05]. This thesis also includes comparisons with a more recent version of RRT, developed in [ND07], that is guided by the star discrepancy coverage measure. To distinguish between RRT and its variant, the acronym RRT[D*] to refer to the star-discrepancy version of RRT [ND07]. This thesis also studies the impact of the high-level discrete planning on the computational efficiency of SyCLoP.

Experiments were run on the Rice Cray XD1 ADA and PBC clusters, where each processor is at 2.2GHz and has up to 8GB RAM. For each experiment, the average computational time in seconds is reported (averages over 40 runs). An upper bound of 3600s was set for each run. SyCLoP constructs the discrete model based on a uniform grid decomposition on a two-dimensional projection, as described in Section 6.3.1.

6.4.3 Results

A summary of the results is shown in Table 6.1. Table 6.1 indicates that SyCLoP is consistently more efficient than RRT. As an example, when RandomDriver is used and $|Q| = 32^2$, RRT requires on average 195.3s, while SyCLoP requires only 2.4s. Similarly, when StudentDriver or HighwayDriver are used and $|Q| = 32^2$, RRT requires on average 210.5s and 219.3s, while SyCLoP requires only 3.4s and 2.9s, respectively. Moreover, as the number of modes is increased SyCLoP remains efficient, while RRT times out. As Table 6.1 shows, RRT times out in all instances with $|Q| \geq 64^2$, while SyCLoP requires on average less than 15s for problem instances with $|Q| = 64^2$ and less than 75s for problem instances with $|Q| = 128^2$.

RandomDriver	$\|Q\|$									
	1^2	2^2	4^2	8^2	16^2	32^2	64^2	128^2	512^2	1024^2
RRT	0.1	0.1	0.3	1.5	16.8	195.3	X	X	X	X
RRT[D*]	0.1	0.9	0.5	4.7	5.1	24.8	411.3	X	X	X
SyCLoP[NoGuide]	0.1	0.1	0.3	3.6	5.7	10.0	147.1	564.8	*X*	*X*
SyCLoP	0.4	0.4	0.6	1.2	1.5	2.4	11.1	66.1	352.7	1198.4

StudentDriver	$\|Q\|$									
	1^2	2^2	4^2	8^2	16^2	32^2	64^2	128^2	512^2	1024^2
RRT	0.1	0.2	0.7	2.4	25.4	210.5	X	X	X	X
RRT[D*]	0.1	1.4	0.3	1.0	4.6	23.2	605.8	X	X	X
SyCLoP[NoGuide]	0.1	0.1	0.4	3.4	5.6	10.3	189.2	576.8	*X*	*X*
SyCLoP	0.4	0.4	0.7	1.3	1.8	3.4	12.4	64.6	294.5	1289.9

HighwayDriver	$\|Q\|$									
	1^2	2^2	4^2	8^2	16^2	32^2	64^2	128^2	512^2	1024^2
RRT	0.1	0.2	0.3	2.9	25.5	219.3	X	X	X	X
RRT[D*]	0.2	0.7	0.2	0.9	3.9	23.7	515.5	X	X	X
SyCLoP[NoGuide]	0.1	0.1	0.4	4.0	5.9	8.8	151.6	514.9	*X*	*X*
SyCLoP	0.4	0.4	0.6	1.3	1.8	2.9	10.9	70.4	288.9	954.8

Table 6.1: Comparison of SyCLoP to other methods as a function of the number of modes $|Q|$ and the driver model. Times are in seconds (averages over 40 runs). Entries marked with X indicate a timeout, which was set to 3600s.

Table 6.1 also indicates that SyCLoP is consistently more efficient than RRT[D*]. The computational advantages of SyCLoP become more pronounced as $|Q|$ is increased. For example, when RandomDriver is used and $|Q| = 64^2$, RRT[D*] requires on average 411.3s. Similarly, when StudentDriver or HighwayDriver are used, RRT[D*] requires 605.8s and 515.5s, respectively, while SyCLoP requires on average less than 15s. Furthermore, RRT[D*] times out as the number of modes is increased to $|Q| = 128^2$, while SyCLoP requires only a short time (less than 75s) to handle such problem instances.

Table 6.1 also shows that SyCLoP scales up reasonably well. While other methods failed to handle large problem instances beyond $|Q| = 128^2$, SyCLoP even when $|Q| =$

1024^2 remains computationally efficient. Overall, the results show the competitiveness of `SyCLoP` for solving challenging motion-planning problems for hybrid systems.

Impact of High-Level Discrete Planning

The second set of experiments focuses on the importance of the high-level discrete planning on `SyCLoP`. We refer to the version of `SyCLoP` that does not use the high-level discrete planner to guide the low-level motion planner as `SyCLoP[NoGuide]`. Table 6.1 shows that `SyCLoP[NoGuide]` is considerably slower than `SyCLoP`. For example, `SyCLoP[NoGuide]` requires on the average 564.8s, 576.8s, and 514.9s when $|Q| = 128^2$ and `RandomDriver`, `StudentDriver`, and `HighwayDriver` are used, respectively, while `SyCLoP` requires only 66.1s, 64.6s, and 70.4s. These results highlight the importance of high-level discrete planning, which, by guiding the low-level motion planner, significantly improves the computational efficiency of `SyCLoP`.

Impact of Decompositions

As shown in Section 5.4 in the case of motion planning with dynamics, decompositions used for the computation of the discrete model can have an impact on the computational efficiency of `SyCLoP`. When using a grid decomposition, while the efficiency increases when the decomposition is neither too fine- nor too coarse-grained, finding the right granularity requires extensive efforts. As an alternative solution, significant computational gains are obtained with no fine-tuning by instead using conforming Delaunay triangulations. Similar observations, as discussed below, can

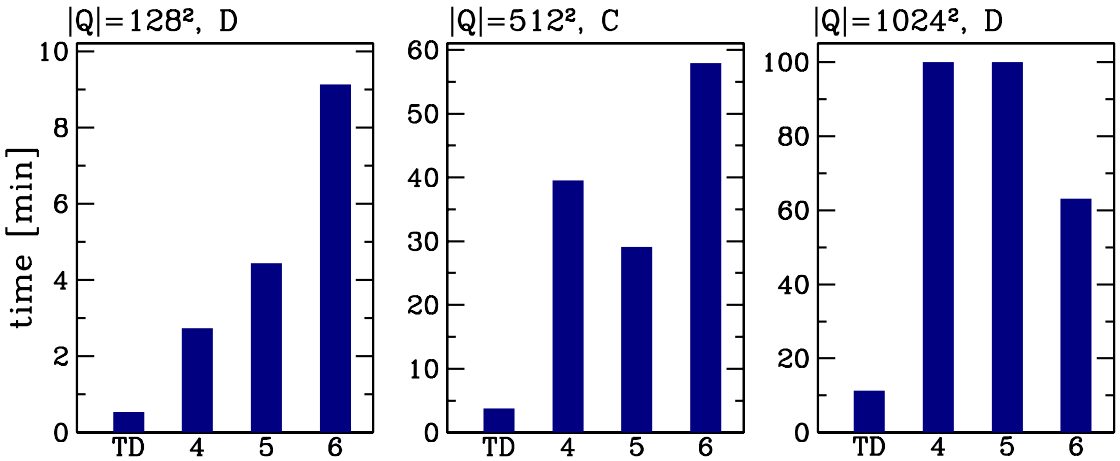


Fig. 6.1: Impact of decompositions on SyCLoP for hybrid systems. Graphs show some typical results when using conforming Delaunay (TD) or grid decompositions with $2^i \times 2^i$ cells, $i = 4, 5, 6$. Finding the right grid size is problem dependent. With no fine-tuning TD yields significant computational speedups.

be made for SyCLoP in the case of hybrid systems as well.

Fig. 6.1 shows the results when SyCLoP uses grid-based decompositions with $2^4 \times 2^4$, $2^5 \times 2^5$, and $2^6 \times 2^6$ cells, and conforming Delaunay triangulations. Fig. 6.1 shows that fine-tuning is necessary in order to find the right grid decomposition so that SyCLoP can take full advantage of the interplay between discrete planning and motion planning. In some cases the best results are obtained when using $2^4 \times 2^4$ grids and in other cases when using $2^5 \times 2^5$ or $2^6 \times 2^6$ grids. Fine-tuning the grid decomposition can, however, require extensive efforts each time SyCLoP is applied to a new problem.

Fig. 6.1 also shows that decompositions based on conforming Delaunay triangulations (see TD in the graphs) offer an alternative solution. As these results indicate, the efficiency of SyCLoP is much higher when using a conforming Delaunay triangulations instead of the various grid decompositions. Furthermore, these significant computational gains are obtained without any fine-tuning.

Chapter 7

Motion Planning with Linear Temporal Logic

This chapter describes the application of `SyCLoP` to automatically plan low-level motions that enable a robot not only avoid collisions with obstacles and reach a desired destination, but also complete high-level tasks specified using the expressiveness of linear temporal logic (LTL). To the best of my knowledge, no other sampling-based motion planner has incorporated LTL directly into motion planning.

7.1 Introduction

When the motion-planning goal is specified as a set of goal states, as is the case with current motion planners [CLH+05, LaV06], it does not matter how a solution trajectory reaches the goal. This makes it difficult or impossible to impose temporal constraints on the solution trajectories. Temporal constraints provide a general way for high-level specifications, such as "the robotic car, after inspecting a contaminated area A, should visit the decontamination station B before visiting any of the base stations C or D," or "the vacuum cleaner should vacuum all the rooms at least twice."

LTL provides the necessary mathematical framework for expressing temporal constraints. LTL has been widely used in model checking of discrete systems in software and hardware [CGP00], and timed systems [BDL+01]. The work in [BBW08] used

LTL to analyze gene networks under parameter uncertainty. The work in [FKGP05] generated trajectories that satisfy LTL constraints on the sequence of triangles visited by a point robot with Newtonian dynamics by using a controller that could drive the point robot between adjacent triangles. Applications of this work [FKGP05] to general systems are limited, however, due to the unavailability of controllers that can generate valid trajectories that allow a robot to navigate between adjacent triangles. In fact, the design of such controllers remains an open problem.

A significant advantage of `SyCLoP` is that it can incorporate high-level tasks expressed in LTL directly into motion planning. This allows `SyCLoP` to compute solution trajectories without relying on controllers, which are difficult to design, in order to steer the robot from one state to another. In this way, `SyCLoP` effectively computes solution trajectories that enable complex robotic systems, such as systems with nonlinear dynamics and hybrid systems, to avoid collisions with obstacles, reach the desired destination, and accomplish the assigned task expressed in LTL.

7.2 Linear Temporal Logic (LTL)

LTL provides the necessary mathematical framework for expressing high-level specifications as temporal constraints. LTL formulas combine propositions with Boolean connectives $\neg$, $\wedge$, $\vee$ and temporal connectives $\mathcal{X}$ (next), $\mathcal{U}$ (until), $\mathcal{R}$ (release), $\mathcal{F}$ (future), $\mathcal{G}$ (globally). This thesis interprets LTL formulas over trajectories $\gamma : [0, T] \rightarrow \mathcal{S}$. As a result of this interpretation, $\mathcal{X}$ is not defined.

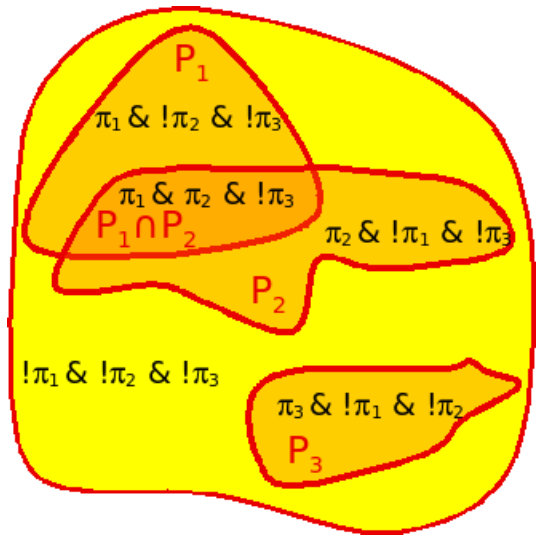


Fig. 7.1: Illustration of a set of propositions and the propositional assignment map.

7.2.1 Propositional Map and Propositional Assignments

Let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ denote the set of propositions. Each proposition π_i is associated with a set of states $P_i \subset \mathcal{S}$. A state $s \in \mathcal{S}$ satisfies proposition π_i iff $s \in P_i$. The map $\tau : \mathcal{S} \to 2^{\Pi}$ maps each state $s \in \mathcal{S}$ to propositional truth values:

$$\tau(s) = \{\pi_i : \pi_i \in \Pi \text{ and } s \in P_i\}.$$

In this way, $\tau(s)$ indicates the propositions satisfied by $s \in S$. Fig. 7.1 provides an illustration.

7.2.2 Propositional Assignments Satisfied by a Trajectory

Consider a trajectory $\gamma : [0, T] \to \mathcal{S}$. At time $T_1 = 0$, γ satisfies some propositional assignment τ_1, where $\tau_1 = \tau(\gamma(0))$. Then, γ continues to satisfy τ_2 until some time T_2, $T_1 < T_2 \leq T$, where γ satisfies some other propositional assignment $\tau_2 = \tau(\gamma(T_2))$. Continuing in this manner, γ satisfies a sequence of propositional assignments $[\tau_i]_{i=1}^{n}$,

where $\tau_i = \tau(\gamma(T_i))$, $\tau_i \neq \tau_{i+1}$, and $0 \leq T_1 < T_2 < \cdots < T_n \leq T$, as indicated in the following definition.

Definition 7.2.1. (Propositional Assignments Satisfied by a Trajectory). *Let $\gamma : [0, T] \to \mathcal{S}$ denote a trajectory. Then, γ satisfies the sequence of propositional assignments $[\tau_i]_{i=1}^n$ where*

- $\tau_i \neq \tau_{i+1}$, *for* $i = 1, \ldots, n-1$
- $\tau_i = \tau(\gamma(T_i))$, *for* $i = 1, \ldots, n$, *where*

$$T_i = \begin{cases} 0, & \text{if } i = 1 \\ \operatorname{argmin}_{T_{i-1} < t \leq T} \ \tau_{i-1} \neq \tau(\gamma(t)), & \text{if } i > 1. \end{cases}$$

We write $\tau(\gamma)$ to denote such sequence, $\tau(\gamma) = [\tau_i]_{i=1}^n$, and say that γ follows $[\tau_i]_{i=1}^n$.

7.2.3 Syntax and Semantics

This section defines the syntax, semantics, and syntactically safe LTL formulas.

Definition 7.2.2. (LTL Syntax). *Every proposition $\pi \in \Pi$ is a formula. If ϕ and ψ are formulas, then $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \mathcal{U} \psi$, $\phi \mathcal{R} \psi$, $\mathcal{F}\phi$, and $\mathcal{G}\phi$ are also formulas.*

(LTL Semantics). *Let $\gamma : [0, T] \to \mathcal{S}$ denote a system trajectory. Let $\gamma_{\geq t}$ denote γ for $t' \geq t$. Then, the semantics of an LTL formula over γ are defined as*

- $\gamma \models \pi$ *for* $\pi \in \Pi$ *if* $\pi \in \tau(\gamma(0))$
- $\gamma \models \neg\phi$ *if* $\gamma \not\models \phi$

- $\gamma \models \phi \wedge \psi$ *if* $\gamma \models \phi$ *and* $\gamma \models \psi$

- $\gamma \models \phi \mathcal{U} \psi$ *if* $\exists t \geq 0$ *such that* $\gamma_{\geq t} \models \psi$ *and* $\gamma_{\geq t_1} \models \phi$ *for all* $0 \leq t_1 < t$, *and*

- $\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$, $\mathcal{F}\phi \equiv \text{true}\mathcal{U}\phi$, $\mathcal{G}\phi \equiv \neg\mathcal{F}\neg\phi$, *and* $\phi \mathcal{R} \psi \equiv \neg(\neg\phi \mathcal{U} \neg\psi)$.

(Syntactically Safe LTL [Sis94]). *An LTL formula ϕ that, when written in positive normal form, uses only the temporal operators $\mathcal{R}$ and $\mathcal{G}$ is syntactically safe. Every syntactically safe formula is a safety formula.*

The following provides several examples of LTL formulas.

- "take measurements from each station A, B, C, D in the area":

$$\mathcal{F}(\pi_A) \wedge \mathcal{F}(\pi_B) \wedge \mathcal{F}(\pi_C) \wedge \mathcal{F}(\pi_D),$$

  where π_i, $i = A, B, C, D$, is the proposition associated with station i.

- "after inspecting a contaminated area A, visit a decontamination station B, before returning to any of the base stations C or D":

$$\mathcal{F}\left(\pi_A \wedge ((\neg\pi_C \wedge \neg\pi_D)\mathcal{U}(\pi_B \wedge \mathcal{F}(\pi_C \vee \pi_D)))\right)$$

7.2.4 Automata Representation

In this work, syntactically safe LTL formulas are translated to nondeterministic finite automata (NFA).

Definition 7.2.3. (NFA). *An NFA is a tuple $\mathcal{A} = (V, \Sigma, \delta, v_0, \text{Acc})$, where V is a finite set of states; $\Sigma = 2^{\Pi}$ is the input alphabet of propositional assignments; δ :*

$V \times \Sigma \to 2^V$ is the transition function; $v_0 \in V$ is the initial state; and $\mathrm{Acc} \subseteq V$ is the set of accepting states. The set of states on which the sequence $[\tau_i]_{i=1}^n$ of propositional assignments ends up when run on $\mathcal{A}$ is defined as

$$\mathcal{A}([\tau_i]_{i=1}^n) = \begin{cases} \delta(v_0, \tau_1), & n = 1 \\ \bigcup_{v \in \mathcal{A}([\tau_i]_{i=1}^{n-1})} \delta(v, \tau_n), & n > 1. \end{cases}$$

$\mathcal{A}$ accepts $[\tau_i]_{i=1}^n$ iff $\mathcal{A}([\tau_i]_{i=1}^n) \cap \mathrm{Acc} \neq \emptyset$.

Let $\gamma : [0, T] \to \mathcal{S}$ denote a trajectory. Let $\tau(\gamma)$ denote the sequence of propositional assignment $[\tau_i]_{i=1}^n$ in the order that they are satisfied by γ (see Section 7.2.2). The automaton $\mathcal{A}$ is said to accept γ iff $\mathcal{A}$ accepts $\tau(\gamma)$.

7.2.5 Problem Statement

Given a hybrid automaton $\mathcal{H} = (\mathcal{S}, \textsc{Valid}, E, \textsc{Guard}, \textsc{Jump}, \mathcal{U}, f)$, propositions Π, and a syntactically safe LTL formula ϕ over Π, compute a sequence $u_1, u_2, \ldots, u_k$ of input controls and a sequence $T_1, T_2, \ldots, T_k$ of times, such that the hybrid-system trajectory $\gamma : [0, T] \to S$, where $T = \sum_{i=1}^k T_i$, and $\gamma \stackrel{def}{=} \Upsilon_{s_{\mathrm{init}}, u_1, T_1} \circ (u_2, T_2) \circ \cdots \circ (u_k, T_k)$, is valid and satisfies ϕ, i.e., $(\forall t \in [0, T] : \textsc{Valid}(\gamma(t)) = \top)$ and $\gamma \models \phi$.

7.3 Applying `SyCLoP` to Motion Planning with LTL

The multi-layered approach `SyCLoP` can incorporate high-level tasks expressed in LTL directly into motion planning. This section describes the modifications needed to be made to the high-level discrete model and the low-level motion-planning layer

of SyCLoP in order to solve LTL motion-planning problems.

7.3.1 High-Level Discrete Model of the Motion-Planning Problem

Consider a valid trajectory $\gamma : [0, T] \to \mathcal{S}$. While in the traditional motion-planning formulation a solution trajectory was found whenever $\textsc{Goal}(\gamma(T)) = \top$, in the LTL motion-planning formulation a solution trajectory must satisfy ϕ, i.e., $\gamma \models \phi$ (see Section 7.2.3). As a result, the discrete model of the motion-planning problem with LTL should incorporate the LTL specification in addition to the state-space partition. The propositions $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ and the propositional map $\tau : \mathcal{S} \to 2^{\Pi}$ induce a partition of $\mathcal{S}$, defined as

$$\mathcal{B} = \{\mathcal{B}(\tau_i) : \tau_i \in 2^{\Pi}\}, \text{ where } \mathcal{B}(\tau_i) = \{s \in \mathcal{S} : \tau(s) = \tau_i\}.$$

Following the definitions and discussion in Section 4.2, the graph $G_{\mathcal{B}} = (V_{\mathcal{B}}, E_{\mathcal{B}})$ of the partition $\mathcal{B}$ is constructed as $V_{\mathcal{B}} = \{v(\mathcal{B}(\tau_i)) : \mathcal{B}(\tau_i) \in \mathcal{B}\}$ and

$$E_{\mathcal{B}} = \{(v(\mathcal{B}(\tau_i)), v(\mathcal{B}(\tau_j))) : (\mathcal{B}(\tau_i), \mathcal{B}(\tau_j) \in \mathcal{B}) \wedge (\mathcal{B}(\tau_i) \text{ is adjacent to } \mathcal{B}(\tau_j))\}.$$

As discussed earlier, the graph $G_{\mathcal{B}} = (V_{\mathcal{B}}, E_{\mathcal{B}})$ by itself is not sufficient for the computation of high-level plans, since in LTL motion planning the goal is expressed as an LTL formula ϕ and not as a set of states, as is the case in traditional motion planning.

We would like to obtain a high-level plan consisting of a sequence $[\mathcal{B}(\tau_i)]_{i=1}^{k}$ such that the associated sequence $[\tau_i]_{i=1}^{k}$ of propositional assignments satisfies ϕ. As shown in [KV01], given a syntactically safe LTL formula ϕ, with an exponential blow-up at most, an NFA $\mathcal{A}_{\phi}$ can be constructed that describes all prefixes satisfying ϕ.

Therefore, $[\tau_i]_{i=1}^{k}$ should end on an accepting state when run on $\mathcal{A}_\phi$.

This is achieved by performing a discrete search (explicit model checking) on the combined product of $G_\mathcal{B} = (V_\mathcal{B}, E_\mathcal{B})$ and $\mathcal{A}_\phi$. The combined product of $G_\mathcal{B} = (V_\mathcal{B}, E_\mathcal{B})$ and $\mathcal{A}_\phi$ can be expressed as a graph $G_\mathcal{R} = (V_\mathcal{R}, E_\mathcal{R})$, where

- $V_\mathcal{R} = \{v(\mathcal{R}(\alpha, \tau_i)) : \alpha \in V_{\mathcal{A}_\phi}, v(\mathcal{B}(\tau_i)) \in V_\mathcal{B}\}$. and
- $E_\mathcal{R} = \{(v(\mathcal{R}(\alpha', \tau_i)), v(\mathcal{R}(\alpha'', \tau_j)) : (\alpha', \alpha'') \in E_{\mathcal{A}_\phi}, (v(\mathcal{B}(\tau_i)), v(\mathcal{B}(\tau_j))) \in E_\mathcal{B}\}$.

In this way, the graph $G_\mathcal{R} = (V_\mathcal{R}, E_\mathcal{R})$ of the discrete model can be used for the computation of high-level plans as described in Section 4.4.

7.3.2 Low-Level Motion Planning

The semantics of certain functions of the low-level motion planner in `SyCLoP` change slightly due to incorporating LTL into motion planning. This section describes all the modifications that are made to the low-level motion planner in `SyCLoP`.

Associating a State in the Search Tree with a High-Level Region

A state $s \in \mathcal{T}$ is associated with $\mathcal{R}(\alpha, \tau_i)$, written $s \in \mathcal{R}(\alpha, \tau_i)$, if the following two conditions are satisfied:

- s satisfies τ_i, i.e., $\tau(s) = \tau_i$, and
- the trajectory $\textsc{Traj}(\mathcal{T}, s)$, when run on the automaton $\mathcal{A}_\phi$, ends up on the automaton state α.

In other words, $\mathcal{R}(\alpha, \tau_i)$ contains those states $s \in \mathcal{T}$ that satisfy τ_i and whose corresponding trajectories $\textsc{Traj}(\mathcal{T}, s)$ have reached the state α in the automaton $\mathcal{A}_\phi$. Thus, $\textsc{States}(\mathcal{T}, \mathcal{R}(\alpha, \tau_i)) = \{s : s \in \mathcal{T} \wedge s \in \mathcal{R}(\alpha, \tau_i)\}$.

Weight Estimate Associated with a High-Level Region

Recall that, as defined in Eqn. 4.1, the weight estimate $w(\mathcal{R}(\alpha, \tau_i))$ associated with the the region $\mathcal{R}(\alpha, \tau_i)$ is computed as

$$w(\mathcal{R}_i) = \frac{\mathrm{vol}^{z_1}(\mathcal{R}(\alpha, \tau_i)) * \mathrm{cov}^{z_2}(\mathcal{R}(\alpha, \tau_i))}{\mathrm{time}(\mathcal{R}(\alpha, \tau_i))}.$$

The meaning of these terms in the case of motion-planning with LTL is as follows:

- $\mathrm{vol}(\mathcal{R}(\alpha, \tau_i)) \stackrel{def}{=} \mathrm{vol}(\mathcal{B}(\tau_i))$;
- $\mathrm{cov}(\mathcal{R}(\alpha, \tau_i)) \stackrel{def}{=} \textsc{Cov}(\mathcal{B}(\tau_i), \textsc{States}(\mathcal{T}, \mathcal{R}(\alpha, \tau_i)))$, where $\textsc{Cov}$ is described in Section 3.2.5;
- $\mathrm{time}(\mathcal{R}(\alpha, \tau_i))$ denotes the time `SyCLoP` has spent extending the search tree $\mathcal{T}$ from states in $\textsc{States}(\mathcal{T}, \mathcal{R}(\alpha, \tau_i))$;
- z_1, z_2 are normalization constants.

Adding a Branch to the Search Tree

Let $v(s_{\mathrm{new}})$ be the new vertex and $(v(s), v(s_{\mathrm{new}}))$ the new edge that should be added to $\mathcal{T}$. Let $\mathcal{A}_\phi(\mathcal{T}, s)$ denote the set of the automaton states where $\tau(\textsc{Traj}(\mathcal{T}, s))$ ends up when run on $\mathcal{A}_\phi$. When computing s_{new}, `SyCLoP` also computes

$$\mathcal{A}_\phi(\mathcal{T}, s_{\mathrm{new}}) = \cup_{\alpha \in \mathcal{A}_\phi(\mathcal{T}, s)} \delta(\alpha, \tau(s_{\mathrm{new}})).$$

If any $\alpha \in \mathcal{A}_\phi(\mathcal{T}, s_{\text{new}})$ is an accepting state, then SyCLoP has successfully computed a solution trajectory. If all the automaton states in $\mathcal{A}_\phi(\mathcal{T}, s_{\text{new}})$ are rejecting states, i.e., not connected to some accepting state, then s_{new} is rejected since in such cases it is not possible to extend TRAJ($\mathcal{T}$, s) and obtain a solution trajectory. If s_{new} is not rejected, then $v(s_{\text{new}})$ and $(v(s), v(s_{\text{new}}))$ are added to $\mathcal{T}$. SyCLoP also associates with $(v(s), v(s_{\text{new}}))$ the trajectory $\gamma_{s,s_{\text{new}}}$ that connects s to s_{new}.

7.4 Experiments

The objective of the experiments is to provide a validation of SyCLoP as a hybrid-system motion-planning method for computing solution trajectories that satisfy specifications expressed by LTL formulas. As discussed in Chapter 6, related work [BF04, KEK05, ND07] can only be used for motion-planning problems where the goal is expressed as a set of states. For these cases, SyCLoP (see Chapter 6) showed significant computational speedups of up to two orders of magnitude. Currently, SyCLoP cannot be experimentally compared to related work [BF04, KEK05, ND07], since extending such work to LTL motion planning remains open to research.

Experiments are performed using increasingly complex LTL formulas. The hybrid-system benchmark consists of a model of an autonomous robotic vehicle driving over different terrains, as described in Section 6.4.1. This benchmark, similar to the one used in [PKV07b, PKV08a], is based on a navigation benchmark proposed in [FI04]. Each terrain corresponds to a mode $q \in \mathcal{Q}$. The continuous dynamics, velocity,

and acceleration depend on terrain conditions and vary from one terrain to another. Second-order dynamics used for modeling cars, differential drives, and unicycles (see Section 5.3.2) are associated with each mode. In each terrain, there are polygons marked as propositions $P_{q_i,k}$ and guards $\textsc{Guard}_{(q_i,q_j)}$. When a guard $\textsc{Guard}_{(q_i,q_j)}$ is satisfied, a discrete transition occurs and the mode of the new state is set to q_j.

The choice of this specific system is to provide a concrete benchmark that is easily scalable to test `SyCLoP` as the complexity of LTL formulas is increased. We created 12 safety properties and 100 instances of the hybrid-system benchmark. Syntactically safe LTL formulas were manually designed in order to provide meaningful properties. Benchmark instances were generated at random in order to test `SyCLoP` over many problems and obtain statistically significant results.

7.4.1 Problem Instances

For each problem instance, the number of modes was set to $n_Q = 10$, number of propositions per mode was set to $n_P = 15$, and number of guards per mode was set to $n_{\textsc{Guard}} = 5$. A random problem instance is generated as follows. First, the second-order continuous dynamics associated with each mode, $1 \leq q \leq n_Q$, is selected pseudo-uniformly at random from those of a car, unicycle, or differential drive. Second, velocity is bounded by $v_{\max}$, where $v_{\max}$ is selected pseudo-uniformly at random from $[3, 6]$m/s. Third, for each mode, n_P propositions and $n_{\textsc{Guard}}$ guards are generated as random polygons.

7.4.2 LTL Specifications

Experiments in this work use the following syntactically safe LTL formulas to specify motion-planning problems:

- sequencing: (with $n = 3, 4, 5, 6$)

  $\phi_1^n = (\pi_0 \mathcal{U} (\pi_1 \wedge (\pi_1 \mathcal{U} (\pi_2 \wedge (\pi_2 \mathcal{U} (\ldots \pi_{n-1} \wedge (\pi_{n-1} \mathcal{U} (\pi_0 \mathcal{U} \pi_n)))))))) ;$

- counting a sequence: (with $n = 1, 2, 3, 4$)

  $\phi_2^n = ((\pi_0 \vee \pi_1)\mathcal{U}(\pi_1 \wedge \underbrace{\Xi(\Xi \cdots (\Xi}_{n}((\pi_0 \vee \pi_1)\mathcal{U}\pi_5)))))),$

  $\Xi(\psi) \stackrel{def}{=} \varsigma_1 \mathcal{U} (\pi_2 \wedge (\varsigma_2 \mathcal{U} (\pi_3 \wedge (\varsigma_3 \mathcal{U} (\pi_4 \wedge (\pi_4 \mathcal{U} (\varsigma_1 \wedge \psi)))))))) ; \varsigma_i \stackrel{def}{=} \pi_0 \vee \pi_i$

- coverage: (with $n = 4, 5, 6, 7$) $\phi_3^n = \bigvee_{i=1}^{n} \mathcal{G}(\pi_i)$

For each problem instance, proposition π_i in the above formulas denotes the i-th proposition among the propositions that were generated when creating the problem instance. A solution trajectory for ϕ_1^n will reach, at some point, $\pi_1, \pi_2, \ldots, \pi_n$ in that order. A solution trajectory for ϕ_2^n will, at some point, reach π_2, π_3, π_4 n-times in that order, and then it will reach π_5. A solution trajectory for ϕ_3^n will, at some point, reach each $\pi_1, \ldots, \pi_n$. Standard tools (such as, scheck [Lat03]) were used to generate an NFA for each ϕ. These tools were also used to generate a minimal DFA for ϕ by converting the NFA to a DFA and then applying minimization.

7.5 Results

Experiments were run on the Rice Cray XD1 ADA and PBC clusters, where each processor is at 2.2GHz and has up to 8GB RAM. Each run of `SyCLoP` uses a single processor, i.e., no parallelism. For each LTL motion-planning problem specified by a formula ϕ described in Section 7.4.2, we report the computational time of `SyCLoP` in seconds, averaged over 100 problem instances. A summary of the results is shown in Fig. 7.2 and 7.3.

The results in Fig. 7.2 indicate that `SyCLoP` was capable of computing solution trajectories for all problem instances. Even for the most challenging problem in the experiments (instances where the safety property is specified by ϕ_3^7), `SyCLoP` was able to compute a solution trajectory in less than five minutes. We also observe that the computational time increases sub-linearly (Fig. 7.2(a, b)) or sub-quadratically (Fig. 7.2(c)) with the number of automaton states. These results provide promising initial validation. Since none of the related methods [BF04,KEK05,ND07] can be used for comparisons (as discussed, these methods cannot handle LTL safety properties), we ran experiments with a random version of `SyCLoP` in order to provide a basis for the results. In the random version, the combination of model checking and motion planning is ignored: at each iteration, a state s is selected uniformly at random from all the states in $\mathcal{T}$, and then $\mathcal{T}$ is extended from s as described in Section 4.5. The random version could not solve any of the problem instances. It always exceeded the allowed time $t_{\max}$, even though $t_{\max}$ was set to one hour per problem instance.

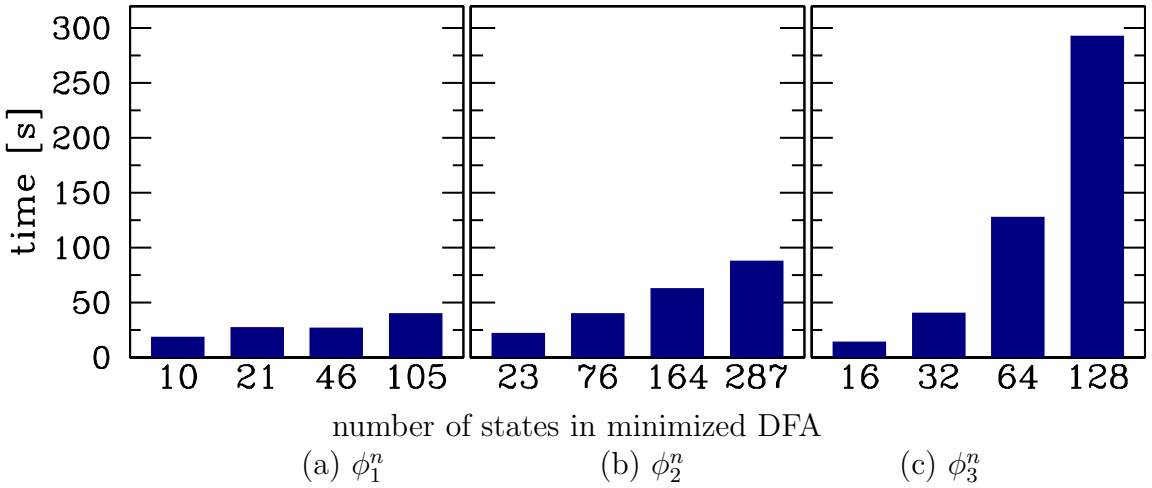


Fig. 7.2: Computational time (in seconds, averaged over 100 runs) of `SyCLoP` when computing solution trajectories for motion-planning problems with various LTL formulas.

We also compared the computational efficiency of `SyCLoP` when using an NFA as computed by standard tools (scheck [Lat03]), a minimal NFA constructed by hand, or a minimal DFA (scheck -d [Lat03]) for ϕ. The motivation for these experiments comes from the work in [AEF+05], which shows significant speedup when using DFA instead of NFA in the context of model checking. Fig. 7.3 shows a summary of the results for various cases of ϕ_2^n. As indicated in Fig. 7.3, `SyCLoP` is only slightly more computationally efficient when using minimal NFAs (constructed by hand) instead of minimal DFAs, even though the minimal NFAs had significantly fewer states than the corresponding minimal DFAs. As concluded in [AEF+05], DFAs offer certain computational advantages that can offset the drawbacks of a possibly exponential increase in size. In particular, a DFA search has a significantly smaller branching factor, since there is exactly one transition that can be followed. This observation is also supported by the comparison of minimized DFAs to non-minimal NFAs (as

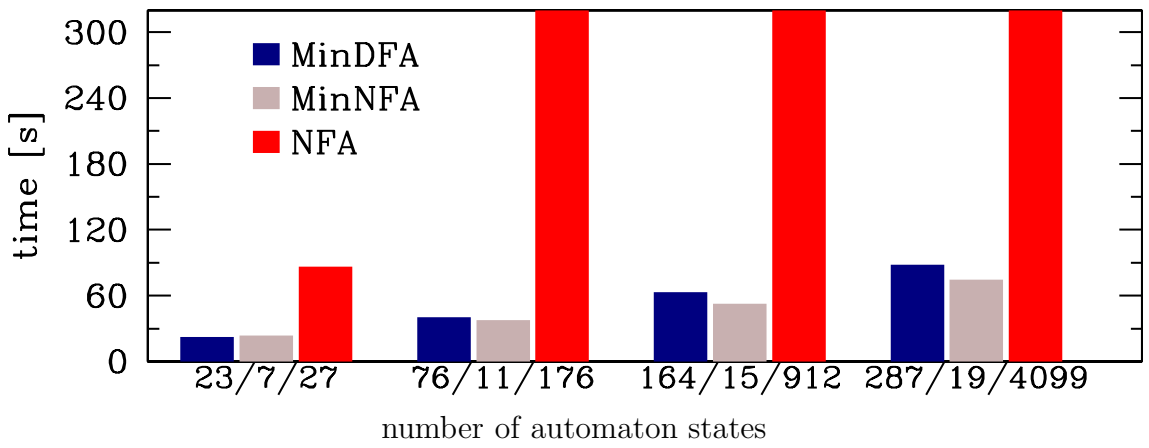


Fig. 7.3: Comparison of the computational time (in seconds, averaged over 100 runs) of `SyCLoP` when using a minimal DFA, a minimal NFA constructed by hand, or an NFA constructed by standard tools for the LTL motion-planning problems specified by ϕ_2^n.

computed by standard tools, such as scheck [Lat03]), since in such cases, as shown in Fig. 7.3, there is significant speedup when using minimized DFAs. Therefore, the non-minimized NFA, as obtained from standard tools, should also be determinized and minimized.

Chapter 8

Falsification of Safety Properties in Hybrid Systems

As hybrid systems are often part of devices operating in safety-critical situations, the validation of safety properties becomes increasingly important. Safety properties assert that nothing "bad" happens to the system. This chapter highlights the connection between motion planning and the falsification of safety properties in hybrid systems. It then describes the applications of `SyCLoP` for the falsification of safety properties in hybrid systems with nonlinear dynamics and high-dimensional continuous spaces. Experiments on an aircraft conflict-resolution protocol in the context of air-traffic management demonstrate the computational efficiency of `SyCLoP`, which obtains up to two orders of magnitude speedup when compared to related work.

8.1 Verification of Safety Properties in Hybrid Systems

Hybrid systems play an increasingly important role not only in robotics, but also in transportation networks as part of sophisticated embedded controllers used in the automotive industry and air-traffic management, or in manufacturing processes, and even medicine and biology as part of medical devices monitoring health conditions [TPS98, PC00, GL04, BEKK05, PAM+05]. As discussed in Chapter 6, a hybrid system

combines discrete and continuous dynamics. Continuous dynamics are associated with each mode, and discrete logic determines how to switch between modes. As an example, a hybrid system may model air traffic control, where the modes correspond to the cruising of the planes and the discrete logic models conflict-resolution protocols.

As hybrid systems are employed in safety-critical situations, verification of safety properties becomes increasingly important [ACH+95, TMBO03]. Safety properties assert that nothing "bad" happens, e.g., "the robotic car will not shift to reverse when driving at high velocities," or "the concentrations in a regulatory gene network will not increase beyond certain levels." A hybrid system is considered safe if unsafe states cannot be reached starting from initial safe states.

The hybrid-system verification problem has traditionally been formulated as a reachability analysis on the state space of the hybrid system. In the forward reachability formulation, safety verification is equivalent to showing that the set of states reachable from the initial states does not intersect the set of unsafe states. In the backward reachability formulation, safety is guaranteed by showing that the set of states that can reach an unsafe state does not intersect the initial set of states.

Over the years a rich theory has been developed for this problem as well as numerous methods [ACH+95, HKPV95, Pur95, Hen96, LPY99, LL98]. Initial approaches included enumeration and symbolic methods originally developed for discrete systems [CGP00]. Tools such as KRONOS [Yov97] and UPPAAL [BDL+01] have been used for the verification of real-time hardware and software, and HyTech [HHWT97]

has been used for the verification of hybrid systems with linear dynamics. Research has also focused on abstraction methods that make verification more amenable to analysis by constructing a simplified model that simulates the original system [AHLP00, GPB05, CK03, ADI06]. The simplified model is usually obtained by eliminating variables that do not influence safety properties, mapping each domain to a smaller domain, or constructing finite-state models that group states that satisfy the same predicates. Alternative methods have also been developed that approximate the reachable set [ACH+95, SK00, BT00, ADM02, TMBO03, SK03]. Tools such as d/dt [ADM02], Checkmate [SK00], VeriSHIFT [BT00] use polyhedra or ellipsoids to overapproximate the reachable set, and other tools use level sets to compute convergent approximations [TMBO03].

8.2 From Verification to Falsification

Unfortunately, even for safety properties where verification is equivalent to reachability checking, decidability holds only for hybrid systems with simple continuous dynamics (essentially some types of linear dynamics) [ACH+95, HKPV95, TMBO03, Mit07]. In light of these theoretical results, it is no surprise that the most efficient complete algorithms for hybrid-system verification have a single- or double-exponential dependency on the dimension of the state space and are generally limited in practicality to hybrid systems with up to six dimensions, simple dynamics, and few or no input controls [ACH+95, TMBO03, Mit07].

These hardness theoretical results underscore the need for the development of alternative methods that perhaps satisfy weaker forms of completeness, but can handle more general hybrid systems. In fact, recent computational methods developed in [EKK04,BF04,KEK05,BCLM06,ND07,JFA+07], even though unable to determine that a hybrid system is safe, are capable of handling nonlinear hybrid systems and finding unsafe behaviors when such systems are unsafe.

In essence, the focus in these recent approaches shifts from *verification* to *falsification*, which often is the main focus of model checking in industrial applications [CFF+01]. Falsification studies the following problem: *Can a hybrid-system witness trajectory be produced from a safe state to an unsafe state when such trajectories exist?*

8.3 Applying `SyCLoP` to Hybrid-System Falsification

This thesis approaches hybrid-system falsification from a motion-planning perspective. While the objective in motion planning for hybrid systems is to reach a desired goal, the objective in hybrid-system falsification is to find a trajectory that leads to an unsafe state. That is, the objective of hybrid-system falsification is the complement of the motion-planning goal. This allows the direct application of motion-planning methods for hybrid systems to also be used for the falsification of safety properties. Therefore, the multi-layered approach, `SyCLoP`, developed in this thesis can be used for the falsification of safety properties in hybrid systems, as

described in Chapter 6. An advantage of SyCLoP is that it offers significant computational speedups of up to two orders of magnitude over related work [KEK05, ND07]. When a hybrid system is safe, it may not be possible to prove that unsafe states are unreachable. Such an approach trades completeness for the ability to discover safety violations for complex hybrid systems with nonlinear dynamics that current verification methods cannot handle.

8.3.1 Falsification of LTL Safety Properties in Hybrid Systems

A significant advantage of SyCLoP is that they can take into account specifications expressed by LTL formulas (see Chapter 7). In the context of motion planning, such specifications correspond to high-level tasks. In the case of hybrid-systems, LTL can be used to specify safety properties. LTL allows for complex specifications that cannot otherwise be expressed as a set of states, as it is the case in current falsification methods [BF04, KEK05, ND07, PKV07b, PKV08a]. Other methods, such as [HHMWT00, DPR07], have been used however to verify LTL safety properties for hybrid systems where the continuous dynamics can have nonlinear terms but only first-order derivatives. Another limitation is that these methods [HHMWT00,DPR07] cannot handle high-dimensional systems such as those used in robotic vehicles or air-traffic management. As described in Chapter 7, SyCLoP can be used as a hybrid-system falsification method, that, in contrast to related work [BF04, KEK05, ND07], can handle LTL safety properties, controls, and general nonlinear dynamics.

8.4 Experiments

The rapid increase in air traffic is stressing the capabilities of current mostly human operated air traffic management systems (ATMS). Enhanced automation of future ATMS is seen as a viable approach to improve the system and alleviate the task of human operators [SMT+95, TPS98, BP00, LLL00, TMG01, GL04]. An important aspect of current and future ATMS is the guarantee of aircraft safety. A safety conflict occurs when two aircraft come closer to one another than a desired minimum distance. Conflict-avoidance procedures used in current ATMS first determine possible conflicts by predicting future aircraft positions and then resolve possible conflicts by modifying aircraft routes. In order to simplify the task of human operators, researchers are focusing on the development of conflict-resolution protocols which take into account traffic and environmental conditions and even the possibility of malicious attacks. Such conflict-resolution protocols are usually modeled as hybrid systems [SMT+95, TPS98, BP00, LLL00, TMG01, GL04]. As the complexity of these conflict-resolution protocols increases, the validation of safety properties becomes even more challenging and surpasses the capabilities of current hybrid-system verification tools.

This section presents experiments with an aircraft conflict-resolution protocol with high-dimensional continuous state spaces. Such experiments demonstrate the effectiveness of `SyCLoP`. Comparisons to related work show computational speedups of up to two orders of magnitude.

8.4.1 Aircraft Conflict-Resolution Protocol

The aircraft conflict-resolution protocol, which has been widely used in [TMBO03, BF04, KEK05, ND07], tests the computational efficiency of `SyCLoP` when also dealing with high-dimensional continuous states spaces. The continuous state space is $X = X_1 \times X_2 \times \cdots \times X_N$, where X_i is the continuous state space associated with the i-th aircraft. Each aircraft i has three continuous state variables (x_i, y_i, θ_i), where (x_i, y_i) denotes the position and θ_i denotes the orientation. This thesis presents experiments with up to 20 aircraft (60 continuous dimensions), which is considerably larger than instances considered in related work (5 aircraft in [KEK05] and 8 aircraft in [ND07]). The continuous dynamics of the i-th aircraft are given by

$$\begin{aligned}
\dot{x}_i &= v\cos(\theta_i) + (-u_1\sin(\theta_i) + d_2\cos(\theta_i))(-\sin(\theta_i)) \\
\dot{y}_i &= v\cos(\theta_i) + (-u_1\sin(\theta_i) + d_2\cos(\theta_i))(\cos(\theta_i)) \\
\dot{\theta}_i &= \textsc{Protocol}(i)
\end{aligned}$$

where v is a constant forward velocity; $u_1, u_2 \in [-w, w]$ is the wind disturbance; and $\textsc{Protocol}(i)$ determines the yaw rate. The discrete logic is incorporated in the computation of $\textsc{Protocol}(i)$, which is based on a conflict-resolution protocol that aims to safely bring all aircrafts from their initial positions $(x_i^{\text{init}}, y_i^{\text{init}})$ to their goal positions $(x_i^{\text{goal}}, y_i^{\text{goal}})$ while avoiding collisions with each other.

As in [EKK04, KEK05, ND07], the function $\textsc{Protocol}(i)$ switches depending on the modes associated with the aircrafts. At the initial position, the i-th aircraft is in heading mode, $q = 1$, and rotates with an angular velocity $\dot{\theta}_i = \textsc{Protocol}(i) =$

$\theta_{\text{goal}} - \theta_i$ until it points toward the goal position, where $\theta_{\text{goal}} \in [-\pi, \pi)$ is computed as the directed angle between the x-axis and $(x_i^{\text{goal}}, y_i^{\text{goal}})$. Once reaching the desired goal heading, the i-th aircraft switches to cruising mode, $q = 2$, and cruises toward the goal with angular velocity $\dot{\theta}_i = \textsc{Protocol}(i) = 0$. If two aircrafts i and j get close to each-other, i.e., within p distance, then both aircrafts enter an avoid mode, $q = 2$. During the avoid mode, both aircrafts i and j make an instantaneous turn by $-90°$ and then follow a half-circle with constant angular velocity $\dot{\theta}_i = \textsc{Protocol}(i) = c$ and $\dot{\theta}_j = \textsc{Protocol}(j) = c$. At the end of the half circle, each aircraft makes instantaneous turns until pointing toward their own goal positions, and then the aircrafts return to cruise mode. It is also possible that during the avoid mode between aircrafts i and j, another aircraft k comes within p distance to i. In this case, aircrafts i and k make instantaneous turn by $-90°$ and execute the same avoid procedure as above. When an aircraft reaches the goal position, it stays there and it is no longer involved in the collision-avoidance protocol. A violation of the safety property occurs if at any point two aircraft come within d ($d < p$) distance from each other.

We initially experimented with the benchmark in [EKK04], which has 5 aircraft (15 continuous dimensions). As in [EKK04], the avoidance distance was set to $p = 5.25km$ and the collision distance was set to $d = 1km$. The translational velocity was set to $0.3km/s$ and the angular velocity was set to $c = 0.03rad/s$. The maximum wind disturbance was set to $w = 0.1$. For the benchmark used in [EKK04], all computational methods tested in the experiments, `RRT`, `RRT[D*]`, `SyCLoP`, were able to

compute witness trajectories in a matter of a few seconds (less than 10s). The methods would quickly find collisions that resulted from the aircrafts making instantaneous $-90°$ turns during the avoid mode and bumping into each other as they followed the respective half-circles. Fig. 8.1 provides an illustration.

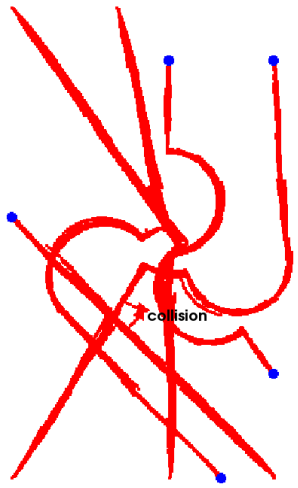


Fig. 8.1: A collision between two aircraft is quickly found after a few seconds (less than 10s). The exploration is shown in red. Goal positions are shown as blue circles.

8.4.2 Safer Aircraft Conflict-Resolution Protocol

In order to make the protocol safer, when two aircrafts i and j enter an avoid mode, each aircraft determines whether it would be best to make a $-90°$ or a $90°$ instantaneous turn. Let $\text{halfcircle}_i(a_i)$ denote the half-circle made by the i-th aircraft following an a_i-degree instantaneous turn, where $a_i \in \{-90°, 90°\}$. The half-circle $\text{halfcircle}_i(a_i)$ is defined similarly. The decision which half-circle to take is based on maximizing the minimum distance between the two aircraft when they follow the

half circles with constant angular velocity $\dot{\theta}_i = \textsc{Protocol}(i) = (-\text{sign}(a_i))c$ and $\dot{\theta}_j = \textsc{Protocol}(j) = (-\text{sign}(a_j))c$. This safer protocol eliminates those collisions which could be avoided by making the appropriate $-90°$ or $90°$ instantaneous turn instead of always turning by $-90°$, as it is the case in [EKK04, KEK05, ND07]. This safer protocol makes it more challenging to compute witness trajectories. A witness trajectory involving 10 aircrafts, as computed by `SyCLoP`, is shown in Fig. 8.2.

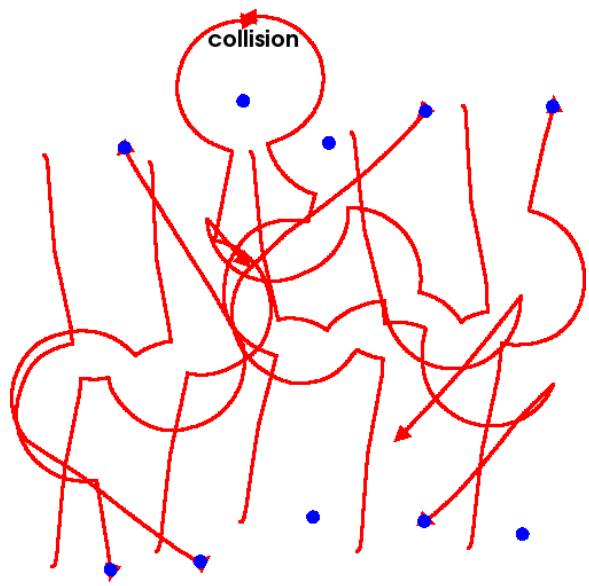


Fig. 8.2: Example of a witness trajectory that indicates a collision between two aircrafts in a scenario involving 10 aircrafts. Blue circles indicate goal positions.

8.4.3 Experimental Settings

A problem instance is obtained by specifying the number N of the aircrafts, the initial $(x_i^{\text{init}}, y_i^{\text{init}})$, and the goal $(x_i^{\text{goal}}, y_i^{\text{goal}})$ positions for each aircraft i. The experiments carried out in related work [TMBO03, BF04, KEK05, ND07] relied on one

problem instance, where the initial and goal positions were set by hand. In order to test `SyCLoP` across different problem instances, we use an automatic procedure to generate random problem instances. This allows a more comprehensive testing that better characterizes the computational efficiency of each method. As noted earlier, in the hand-designed problem instance, all the computational methods (`RRT`, `RRT[D*]`, and `SyCLoP`) solved the problem in less than 10s.

In the automatic procedure for generating a random benchmark instance, half of the aircrafts are placed from left to right at the top and the other half are placed at the bottom at a safe distance from each other. The aircrafts placed at the top have goal positions at the bottom, and the aircrafts placed at the bottom have goal positions placed at the top. More precisely, let $h = N/2$. The gap between aircrafts is set to $\text{gap} = (2.85 + 0.04 * (N - 10)) * p$, which corresponds to $2.85p$ for $N = 10$; $3.05p$ for $N = 15$; and $3.25p$ for $N = 20$. Then, for each $i = 1, \ldots, h$, which corresponds to the first half of the aircrafts, x_i^{init} is selected pseudo-uniformly at random from $[\text{init}_i, \text{init}_i + p]$, where $\text{init}_i = -500km + (i - 1) * \text{gap}$; y_i^{init} is selected pseudo-uniformly at random from $[250, 350]km$; x_i^{goal} is selected pseudo-uniformly at random from $[\text{goal}_i, \text{goal}_i + p]$, where $\text{goal}_i = -420km + (i - 1) * \text{gap}$; and y_i^{goal} is selected pseudo-uniformly at random from $[-350, -250]km$. For each $i = h, \ldots, N$, which corresponds to the second half of the aircrafts, x_i^{init} is selected pseudo-uniformly at random from $[\text{init}_i, \text{init}_i + p]$, where $\text{init}_i = -500km + (i - h - 1) * \text{gap}$; y_i^{init} is selected pseudo-uniformly at random from $[-350, -250]km$; x_i^{goal} is selected pseudo-

method	number of aircrafts $N = 10$	$N = 15$	$N = 20$
`SyCLoP`	30.35s	42.67s	77.61s
`RRT`	242.15	394.40s	1973.11s
`RRT[D*]`	X	X	X

Table 8.1: Comparison of the computational efficiency for solving the aircraft conflict-resolution problem with respect to the number of aircrafts N. For each N, the computational efficiency of each method is measured as the median computational time obtained on 200 random instances of the aircraft conflict-resolution problem. Entries marked with X indicate a timeout, which was set to 3600s.

uniformly at random from $[\text{goal}_i, \text{goal}_i + p]$, where $\text{goal}_i = -420km + (i - h - 1) * \text{gap}$; and y_i^{goal} is selected pseudo-uniformly at random from $[250, 350]km$.

Experiments were carried out with $N = 10, 15, 20$ aircrafts, which correspond to continuous state spaces with $30, 45, 60$ dimensions, respectively. For a fixed N, 200 problem instances were generated using the randomized procedure described in Section 8.4.3. Each method was run on each problem instance. A timeout of 3600s was imposed on each run. The median computational time is reported for each method.

8.5 Results

Table 8.1 provides a summary of the results. In each case, `SyCLoP` is several times faster than `RRT` and `RRT[D*]`. As the number of aircrafts is increased, the computational advantages of `SyCLoP` become more pronounced. In fact, `SyCLoP` obtains computational speedups of up to two orders of magnitude in comparison to related work in hybrid-system falsification.

Chapter 9

Discussion

The work in this thesis was motivated by recent advances in autonomous robotics, where robots such as Twendy-One, vehicles racing in the DARPA Grand Challenge, and the IRobot array of domestic and military robots have shown a great degree of autonomy in accomplishing their assigned task.

This thesis focused on motion planning. Motion planning, which constitutes a fundamental component of autonomy, also presents a significant challenge in autonomous robotics. The overall objective is to provide robots with the ability to automatically plan the motions needed to accomplish an assigned task.

Toward this goal, this thesis developed a multi-layered approach, `SyCLoP`, that seamlessly combines high-level discrete planning and low-level motion planning. A distinctive feature and a crucial property of `SyCLoP` is that high-level discrete planning and low-level motion planning are not independent but in fact work in tandem. High-level discrete planning, which draws from research in AI and logic, guides the low-level motion planning during the search for a solution. Information gathered during the search is in turn fed back from the low-level to the high-level layer in order to improve the high-level plan in the next iteration. In this way, high-level plans become increasingly useful in guiding the low-level motion planner toward a solution.

This synergic combination of high-level discrete planning and low-level motion

planning, as demonstrated in this thesis, allows SyCLoP to effectively solve challenging motion-planning problems that incorporate robot dynamics, physics-based simulations, and hybrid systems. Moreover, SyCLoP yields significant computational advantages of one to two orders of magnitude when compared to related work.

In addition to traditional motion planning, which focuses on planning motions that allow the robot to reach a desired destination while avoiding collisions, SyCLoP can take into account high-level tasks specified using the expressiveness of linear temporal logic (LTL). LTL allows for complex task specifications, such as sequencing, coverage, and other combinations of temporal objectives.

This thesis also provides a seemingly surprising contribution of motion planning in hybrid-system falsification. Analogies between motion-planning trajectories in traditional robotics applications and witness trajectories that show a hybrid system violating a safety property are exploited to apply SyCLoP for the falsification of safety properties in hybrid systems. We note that SyCLoP allows specification of safety properties expressed by syntactically safe LTL formulas for hybrid systems with general nonlinear dynamics.

The synergic combination of high-level discrete planning and low-level motion planning in SyCLoP demonstrated the ability of motion planning to take into account rich models of robots and the physical world. This provides an important step toward the overall goal of enabling robots in the physical world to autonomously accomplish high-level tasks. In combination with control theory, localization, mapping, vision,

and probabilistic reasoning, this has the potential to significantly increase the autonomy of robots employed in service, search-and-rescue missions, exploration, and navigation to accomplish their assigned tasks.

Bibliography

[ABC07] ABC. Japanese housework robot unveiled (2007)

[ABD+98a] Amato NM, Bayazit B, Dale L, Jones C, and Vallejo D. OBPRM: An obstacle-based PRM for 3d workspaces. In International Workshop on Algorithmic Foundations of Robotics, 156–168. Houston, Texas (1998)

[ABD+98b] Amato NM, Bayazit OB, Dale LK, Jones C, and Vallejo D. Choosing good distance metrics and local planners for probabilistic roadmap methods. In IEEE International Conference on Robotics and Automation, 630–637. Leuven, Belgium (1998)

[ACH+95] Alur R, Courcoubetis C, Halbwachs N, Henzinger TA, Ho PH, Nicollin X, Olivero A, Sifakis J, and Yovine S. The algorithmic analysis of hybrid systems. Theoretical Computer Science 138(1):3–34 (1995)

[AD97] Agarwal PK and Desikan PK. An efficient algorithm for terrain simplification. In ACM-SIAM Symposium on Discrete Algorithms, 139–147. New Orleans, Louisiana (1997)

[ADI06] Alur R, Dang T, and Ivančić F. Counterexample-guided predicate abstraction of hybrid systems. Theoretical Computer Science 354(2):250–271 (2006)

[ADM02] Asarin E, Dang T, and Maler O. The d/dt tool for verification of hybrid systems. In International Conference on Computer Aided Verification, Lecture Notes in Computer Science, 365–370. Copenhagen, Denmark (2002)

[AEF+05] Armoni R, Egorov S, Fraer R, Korchemny D, and Vardi M. Efficient LTL compilation for SAT-based model checking. In International Conference on Computer-Aided Design, 877–884. San Jose, California (2005)

[AHLP00] Alur R, Henzinger TA, Lafferriere G, and Pappas G. Discrete abstractions of hybrid systems. Proceedings of the IEEE 88(7):971–984 (2000)

[AOLK00] Anshelevich E, Owens S, Lamiraux F, and Kavraki LE. Deformable volumes in path planning applications. In IEEE International Confer-

ence on Robotics and Automation, 2290–2295. San Francisco, California (2000)

[BB05] Burns B and Brock O. Toward optimal configuration space sampling. In Robotics: Science and Systems, 105–112. Cambridge, Massachusetts (2005)

[BB07] Burns B and Brock O. Single-query motion planning with utility-guided random trees. In IEEE International Conference on Robotics and Automation, 3307–3312. Rome, Italy (2007)

[BBW08] Batt G, Belta C, and Weiss R. Temporal logic analysis of gene networks under parameter uncertainty. IEEE Transactions of Automatic Control 53:215–229 (2008)

[BCLM06] Branicky MS, Curtiss MM, Levine J, and Morgan S. Sampling-based planning, control, and verification of hybrid systems. Control Theory and Applications 153(5):575–590 (2006)

[BDL+01] Behrmann G, David A, Larsen KG, Mller O, Pettersson P, and Yi W. Uppaal - present and future. In IEEE Conference on Decision and Control, 2881–2886. Orlando, Florida (2001)

[BEKK05] Belta C, Esposito J, Kim J, and Kumar V. Computational techniques for analysis of genetic network dynamics. International Journal of Robotics Research 24(2–3):219–235 (2005)

[BF04] Bhatia A and Frazzoli E. Incremental search methods for reachability analysis of continuous and hybrid systems. In Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 2993, 142–156. Philadelphia, Pennsylvania (2004)

[BK07] Bekris KE and Kavraki LE. Greedy but safe replanning under kinodynamic constraints. In IEEE International Conference on Robotics and Automation, 704–710. Rome, Italy (2007)

[BL91] Barraquand J and Latombe JC. Robot motion planning: A distributed representation approach. International Journal of Robotics Research 10(6):628–649 (1991)

[BMA95] Bessiere P, Mazer E, and Ahuactzin JM. Planning in continuous space with forbidden regions: The Ariadne's clew algorithm. In International Workshop on Algorithmic Foundations of Robotics, 39–47. Wellsley, Massachusetts (1995)

[BMA98] Bessiere P, Mazer E, and Ahuactzin JM. The Ariadne's Clew algorithm. Journal of Artificial Intelligence Research 9:295–316 (1998)

[BOvdS99] Boor V, Overmars MH, and van der Stappen AF. The Gaussian sampling strategy for probabilistic roadmap planners. In IEEE International Conference on Robotics and Automation, 1018–1023. Detroit, Michigan (1999)

[BP00] Bicchi A and Pallottino L. On optimal cooperative conflict resolution for air trafficmanagement systems. IEEE Transactions on Intelligent Transportation Systems 1(4):221–231 (2000)

[BPR03] Basu S, Pollack R, and Roy MF. Algorithms in Real Algebraic Geometry. Springer-Verlag (2003)

[BT00] Botchkarev O and Tripakis S. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 1790, 73–88. Pittsburgh, Pennsylvania (2000)

[BTK07a] Bekris KE, Tsianos K, and Kavraki LE. A decentralized planner that guarantees the safety of communicating vehicles with complex dynamics that replan online. In IEEE/RSJ International Conference on Intelligent Robots and Systems, 3784–3790. San Diego, California (2007)

[BTK07b] Bekris KE, Tsianos K, and Kavraki LE. A distributed protocol for safe real-time planning of communicating vehicles with second-order dynamics. In International Conference on Robot Communication and Coordination. Athens, Greece (2007)

[BV02] Bruce J and Veloso M. Real-time randomized path planning for robot navigation. In IEEE/RSJ International Conference on Intelligent Robots and Systems, 2383–2388. Lausanne, Switzerland (2002)

[Can88a] Canny J. The Complexity of Robot Motion Planning. MIT Press, Cambridge, Massachusetts (1988)

[Can88b] Canny J. Some algebraic and geometric computations in PSPACE. In ACM Symposium on Theory of Computing, 460–469. Chicago, IL (1988)

[CC00] Cox T and Cox M. Multidimensional Scaling. Chapman & Hall, 2nd edition (2000)

[CFF+01] Copty F, Fix L, Fraer R, Giunchiglia E, Kamhi G, Tacchella A, and Vardi M. Benefits of bounded model checking at an industrial setting. In International Conference on Computer Aided Verification, Lecture Notes in Computer Science, vol. 2102, 436–453. Paris, France (2001)

[CGP00] Clarke EM, Grumberg O, and Peled DA. Model Checking. MIT Press (2000)

[CK03] Chutinan C and Krogh BH. Computational techniques for hybrid system verification. IEEE Transactions on Automatic Control 48(1):64–75 (2003)

[CLH+05] Choset H, Lynch KM, Hutchinson S, Kantor G, Burgard W, Kavraki LE, and Thrun S. Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press (2005)

[CNN98] CNN. Mechanical museum aide explains how robots developed (1998)

[DAR07] DARPA. Tartan racing wins 2 million prize for darpa urban challenge (2007)

[DPIOA07] Del Pin F, Idelsohn SR, Onate E, and Aubry R. The ALE/Langrangian particle finite element method: A new approach to computation of free-surface flows and fluid-object interactions. Computers and Fluids 36(1):27–38 (2007)

[DPR07] Damm W, Pinto G, and Ratschan S. Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. International Journal of Foundations of Computer Science 18(1):63–86 (2007)

[DT97] Drmota M and Tichy RF. Sequences, discrepancies and applications. Springer, Berlin (1997)

[EKK04] Esposito J, Kim J, and Kumar V. Adaptive RRTs for validating hybrid robotic control systems. In International Workshop on Algorithmic Foundations of Robotics, 107–132. Zeist, Netherlands (2004)

[EKP01] Esposito J, Kumar V, and Pappas G. Accurate event detection for simulation of hybrid systems. In Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, 204–217. Rome, Italy (2001)

[FI04] Fehnker A and Ivancic F. Benchmarks for hybrid systems verification. In Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, 326–341. Stanford, California (2004)

[FKGP05] Fainekos GE, Kress-Gazit H, and Pappas G. Temporal logic motion planning for mobile robots. In IEEE International Conference on Robotics and Automation, 2020–2025. Barcelona, Spain (2005)

[FOR07] FORBES. Sony's robot dog has pups (2007)

[GDT+06] Galassi M, Davies J, Theiler J, Gough B, Jungman G, Booth M, and Rossi F. GNU Scientific Library Reference Manual. Network Theory Ltd, 2 edition (2006)

[GHK99] Guibas LJ, Holleman C, and Kavraki LE. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In IEEE/RSJ International Conference on Intelligent Robots and Systems, 254–260. Kyongju, Korea (1999)

[GKM+01] Goldenstein S, Karavelas M, Metaxas D, Guibas L, Aaron E, and Goswami A. Scalable nonlinear dynamical systems for agent steering and crowd simulation. Computers and Graphics 25(6):983–998 (2001)

[GL04] Glover W and Lygeros J. A stochastic hybrid model for air traffic control simulation. In Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 2993, 372–386. Philadelphia, Pennsylvania (2004)

[GLM05] Gayle R, Lin MC, and Manocha D. Constraint-based motion planning of deformable robots. In IEEE International Conference on Robotics and Automation, 1046–1053. Barcelona, Spain (2005)

[GO02] Geraerts R and Overmars M. A comparative study of probabilistic roadmap planners. In International Workshop on Algorithmic Foundations of Robotics, 43–58. Nice, France (2002)

[GPB05] Giorgetti N, Pappas GJ, and Bemporad A. Bounded model checking for hybrid dynamical systems. In IEEE Conference on Decision and Control, 672–677. Seville, Spain (2005)

[GRS+07] Gayle R, Redon S, Sud A, Lin MC, and Manocha D. Efficient motion planning of highly articulated chains using physics-based sampling. In IEEE International Conference on Robotics and Automation, 3319–3326. Rome, Italy (2007)

[Hen96] Henzinger T. The theory of hybrid automata. In IEEE Symposium on Logic in Computer Science, 278–292. Piscataway, New Jersey (1996)

[HHMWT00] Henzinger TA, Horowitz B, Majumdar R, and Wong-Toi H. Beyond HYTECH: Hybrid systems analysis using interval numerical methods. In Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 1790, 130–144. Pittsburgh, Pennsylvania (2000)

[HHWT97] Henzinger TA, Ho PH, and Wong-Toi H. HyTech: A model checker for hybrid systems. Software Tools for Technology Transfer 1:110–122 (1997)

[HK00] Holleman C and Kavraki LE. A framework for using the workspace medial axis in PRM planners. In IEEE International Conference on Robotics and Automation, 1408–1413. San Francisco, California (2000)

[HKL+98] Hsu D, Kavraki LE, Latombe JC, Motwani R, and Sorkin S. On finding narrow passages with probabilistic roadmap planners. In International Workshop on Algorithmic Foundations of Robotics, 141–154. Wellesley, Massachusetts (1998)

[HKLR02] Hsu D, Kindel R, Latombe JC, and Rock S. Randomized kinodynamic motion planning with moving obstacles. International Journal of Robotics Research 21(3):233–255 (2002)

[HKO01] Hyvärinent A, Karhunen J, and Oja E. Independent Component Analysis. John Wiley & Sons (2001)

[HKPV95] Henzinger T, Kopke P, Puri A, and Varaiya P. What's decidable about hybrid automata? In ACM Symposium on Theory of Computing, 373–382. Las Vegas, Nevada (1995)

[HKW98] Holleman C, Kavraki LE, and Warren J. Planning paths for a flexible surface patch. In IEEE International Conference on Robotics and Automation, 21–26. Leuven, Belgium (1998)

[HLK06] Hsu D, Latombe JC, and Kurniawati H. On the probabilistic foundations of probabilistic roadmap planning. International Journal of Robotics Research 25(7):627–643 (2006)

[HLM97] Hsu D, Latombe JC, and Motwani R. Path planning in expansive configuration spaces. In IEEE International Conference on Robotics and Automation, 2719–2726. Albuquerque, New Mexico (1997)

[HSAS05] Hsu D, Sánchez-Ante G, and Sun Z. Hybrid PRM sampling with a cost-sensitive adaptive strategy. In IEEE International Conference on Robotics and Automation, 3885–3891. Barcelona, Spain (2005)

[ICO03] Idelsohn SR, Calvo N, and Onate E. Polyhedrization of an arbitrary point set. Computer Methods in Applied Mechanics and Engineering 192(22-24):2649–2668 (2003)

[Ist02] Isto P. Constructing probabilistic roadmaps with powerful local planning and path optimization. In IEEE/RSJ International Conference on Intelligent Robots and Systems, 2323–2328. Lausanne,Switzerland (2002)

[JFA+07] Julius AA, Fainekos GE, Anand M, Lee I, and Pappas GJ. Robust test generation and coverage for hybrid systems. In Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 4416, 329–342. Pisa, Italy (2007)

[Jol86] Jolliffe I. Principal Components Analysis. Springer-Verlag, New York (1986)

[JYLS05] Jaillet L, Yershova A, LaValle SM, and Simeon T. Adaptive tuning of the sampling domain for dynamic-domain RRTs. In IEEE/RSJ International Conference on Intelligent Robots and Systems, 4086–4091. Edmonton, Canada (2005)

[Kav95] Kavraki LE. Random Networks in Configuration Space for Fast Path Planning. Ph.D. thesis, Stanford University, Stanford, California (1995)

[KEK05] Kim J, Esposito JM, and Kumar V. An RRT-based algorithm for testing and validating multi-robot controllers. In Robotics: Science and Systems, 249–256. Boston, Massachusetts (2005)

[KH06] Kurniawati H and Hsu D. Workspace-based connectivity oracle: An adaptive sampling strategy for PRM planning. In International Workshop on Algorithmic Foundations of Robotics, Springer Tracts in Advanced Robotics. New York, NY (2006)

[KKN+02] Kuffner J, Kagami S, Nishiwaki K, Inaba M, and Inoue H. Dynamically-stable motion planning for humanoid robots. Autonomous Robots 12(1):105–118 (2002)

[KŠLO96] Kavraki LE, Švestka P, Latombe JC, and Overmars MH. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12(4):566–580 (1996)

[KV01] Kupferman O and Vardi M. Model checking of safety properties. Formal methods in System Design 19(3):291–314 (2001)

[L9̈6] Löhner R. Progress in grid generation via the advancing front technique. Engineering with Computers 12(3-4):186–210 (1996)

[Lad06] Ladd AM. Motion Planning for Physical Simulation. Ph.D. thesis, Rice University, Houston, Texas (2006)

[Lat91] Latombe JC. Robot Motion Planning. Kluwer, Boston, Massachusetts (1991)

[Lat03] Latvala T. Efficient model checking of safety properties. In Model Checking Software, Lecture Notes in Computer Science, vol. 2648, 74–88. Portland, Oregon (2003)

[LaV98] LaValle SM. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Department, Iowa State University, Ames, Iowa (1998)

[LaV06] LaValle SM. Planning Algorithms. Cambridge University Press, Cambridge, Massachusetts (2006)

[LBL03] LaValle S, Branicky M, and Lindemann S. On the relationship between classical grid search and probabilistic roadmaps. International Journal of Robotics Research 23(7–8):673–692 (2003)

[LK01] LaValle SM and Kuffner JJ. Randomized kinodynamic planning. International Journal of Robotics Research 20(5):378–400 (2001)

[LK04a] Ladd AM and Kavraki LE. Fast tree-based exploration of state space for robots with dynamics. In International Workshop on Algorithmic Foundations of Robotics, 297–312. Utrecht/Zeist, Netherlands (2004)

[LK04b] Ladd AM and Kavraki LE. Using motion planning for knot untangling. International Journal of Robotics Research 23(7-8):797–808 (2004)

[LK05] Ladd AM and Kavraki LE. Motion planning in the presence of drift, underactuation and discrete system changes. In Robotics: Science and Systems, 233–241. Boston, Massachusetts (2005)

[LL98] Livadas C and Lynch N. Formal verification of safety-critical hybrid systems. In Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 1386, 253–272. Berkeley, California (1998)

[LL03] Lindemann SR and LaValle SM. Current issues in sampling-based motion planning. In Int. Symp. Robotics Research, Springer Tracts in Advanced Robotics, 36–54. Siena, Italy (2003)

[LLL00] Livadas C, Lygeros J, and Lynch NA. High-level modeling and analysis of the traffic alert and collision avoidance system. Proceedings of the IEEE 88(7):926–948 (2000)

[LPY99] Lafferriere G, Pappas G, and Yovine S. A new class of decidable hybrid systems. In Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 1569, 137–151. Berg en Dal, Netherlands (1999)

[Mit07] Mitchell IM. Comparing forward and backward reachability as tools for safety analysis. In Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 4416, 428–443. Pisa, Italy (2007)

[Mol06] Path planning for deformable linear objects. IEEE Transactions on Robotics 22(4):625–636 (2006)

[MRA03] Morales M, Rodriguez S, and Amato N. Improving the connectivity of PRM roadmaps. In IEEE International Conference on Robotics and Automation, 4427–4432. Taipei, Taiwan (2003)

[MTP+04] Morales M, Tapia L, Pearce R, Rodriguez S, and Amato NM. A machine learning approach for feature-sensitive motion planning. In International Workshop on Algorithmic Foundations of Robotics, 361–376. Utrecht/Zeist, Netherlands (2004)

[ND07] Nahhal T and Dang T. Test coverage for continuous and hybrid systems. In International Conference on Computer Aided Verification, Lecture Notes in Computer Science, vol. 4590, 449–462. Berlin, Germany (2007)

[Nie92] Niederreiter H. Random Number Generation and Quasi-Monte Carlo Methods. Society for Industrial and Applied Mathematics (1992)

[NM95] Narkhede A and Manocha D. Fast polygon triangulation based on Seidel's algorithm. In Graphics Gems 5, 394–400. Academic Press (1995)

[OnIDPA04] Oñate E, Idelsohn SR, Del Pin F, and Aubry R. The particle finite element method. an overview. International Journal of Computational Methods 1(2):267–307 (2004)

[PAM+05] Piazza C, Antoniotti M, Mysore V, Policriti A, Winkler F, and Mishra B. Algorithmic algebraic model checking I: Challenges from systems biology. In International Conference on Computer Aided Verification,

Lecture Notes in Computer Science, vol. 3576, 5–19. Edinburgh, Scotland (2005)

[PBC+05] Plaku E, Bekris KE, Chen BY, Ladd AM, and Kavraki LE. Sampling-based roadmap of trees for parallel motion planning. IEEE Transactions on Robotics 21(4):597–608 (2005)

[PBK07] Plaku E, Bekris KE, and Kavraki LE. OOPS for Motion Planning: An Online Open-source Programming System. In IEEE International Conference on Robotics and Automation, 3711–3716. Rome, Italy (2007)

[PC00] Pepyne D and Cassandras C. Optimal control of hybrid systems in manufacturing. Proceedings of the IEEE 88(7):1108–1123 (2000)

[PK05] Plaku E and Kavraki LE. Distributed sampling-based roadmap of trees for large-scale motion planning. In IEEE International Conference on Robotics and Automation, 3879–3884. Barcelona, Spain (2005)

[PK06] Plaku E and Kavraki LE. Quantitative analysis of nearest-neighbors search in high-dimensional sampling-based motion planning. In International Workshop on Algorithmic Foundations of Robotics, Springer Tracts in Advanced Robotics. New York, NY (2006)

[PK07a] Plaku E and Kavraki LE. Distributed computation of the knn graph for large high-dimensional point sets. Journal of Parallel and Distributed Computing 67(3):346–359 (2007)

[PK07b] Plaku E and Kavraki LE. Nonlinear dimensionality reduction using approximate nearest neighbors. In SIAM InternationalConference on Data Mining, 3711–3716. Minneapolis, Minnesota (2007)

[PK08] Plaku E and Kavraki LE. An object-oriented programming system for motion planning. IEEE Transactions on Robotics (2008). Under review

[PKV07a] Plaku E, Kavraki LE, and Vardi MY. Discrete search leading continuous exploration for kinodynamic motion planning. In Robotics: Science and Systems. Atlanta, Georgia (2007)

[PKV07b] Plaku E, Kavraki LE, and Vardi MY. Hybrid systems: From verification to falsification. In International Conference on Computer Aided

Verification, Lecture Notes in Computer Science, vol. 4590, 468–481. Berlin, Germany (2007)

[PKV07c] Plaku E, Kavraki LE, and Vardi MY. A motion planner for a hybrid robotic system with kinodynamic constraints. In IEEE International Conference on Robotics and Automation, 692–697. Rome, Italy (2007)

[PKV08a] Plaku E, Kavraki LE, and Vardi MY. Hybrid systems: From verification to falsification by combining motion planning and discrete search. Formal Methods in System Design (2008). Under review [invited]

[PKV08b] Plaku E, Kavraki LE, and Vardi MY. Impact of workspace decompositions on discrete search leading continuous exploration (DSLX) motion planning. In IEEE International Conference on Robotics and Automation, 3751–3756. Pasadena, California (2008)

[PKV08c] Plaku E, Kavraki LE, and Vardi MY. Motion planning with linear temporal logic. IEEE Transactions on Robotics (2008). In preparation

[PKV08d] Plaku E, Kavraki LE, and Vardi MY. SyCLoP: Synergic combination of layers of planning. International Journal of Robotics Research (2008). In preparation

[PSCK07] Plaku E, Stamati H, Clementi C, and Kavraki LE. Fast and reliable analysis of molecular motion using proximity relations and dimensionality reduction. Proteins: Structure, Function, and Bioinformatics 67(4):897–907 (2007)

[Pur95] Puri A. Theory of Hybrid Systems and Discrete Event Systems. Ph.D. thesis, University of California at Berkeley, Berkeley, California (1995)

[Rei79] Reif J. Complexity of the mover's problem and generalizations. In IEEE Symposium on Foundations of Computer Science, 421–427. San Juan, Puerto Rico (1979)

[She02] Shewchuk JR. Delaunay refinement algorithms for triangular mesh generation. Computational Geometry: Theory and Applications 22(1-3):21–74 (2002)

[She08] Shewchuk JR. General-dimensional constrained delaunay and constrained regular triangulations, i: Combinatorial properties. Discrete & Computational Geometry 39:580–637 (2008)

[SHJ+05] Sun Z, Hsu D, Jiang T, Kurniawati H, and Reif J. Narrow passage sampling for probabilistic roadmap planners. IEEE Transactions on Robotics 21(6):1105–1115 (2005)

[SI07] Saha M and Isto P. Manipulation planning for deformable linear objects. IEEE Transactions on Robotics 23(6):1131–1150 (2007)

[Sis94] Sistla A. Safety, liveness and fairness in temporal logic. Formal Aspects of Computing 6:495–511 (1994)

[SK00] Silva BI and Krogh BH. Formal verification of hybrid systems using CheckMate: a case study. In American Control Conference, 1679–1683. Chicago, Illinois (2000)

[SK03] Stursberg O and Krogh BH. Efficient representation and computation of reachable sets for hybrid systems. In Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 2623, 482–497. Prague, Czech Republic (2003)

[SKYK08] Sucan I, Kruse JF, Yim M, and Kavraki LE. Kinodynamic motion planning with hardware demonstration. In IEEE/RSJ International Conference on Intelligent Robots and Systems. Nice, France (2008)

[SL02] Sánchez G and Latombe JC. On delaying collision checking in PRM planning: Application to multi-robot coordination. International Journal of Robotics Research 21(1):5–26 (2002)

[SMT+95] Sastry S, Meyer G, Tomlin C, Lygeros J, Godbole D, and Pappas G. Hybrid control in air traffic management systems. In IEEE Conference on Decision and Control, vol. 2, 1478–1483. New Orleans, Louisiana (1995)

[SS88] Schwartz JT and Sharir M. A survey of motion planning and related geometric algorithms. Artificial Intelligence 37:157 – 169 (1988)

[TMBO03] Tomlin CJ, Mitchell I, Bayen A, and Oishi M. Computational techniques for the verification and control of hybrid systems. Proceedings of the IEEE 91(7):986–1001 (2003)

[TMG01] Tomlin CJ, Mitchell I, and Ghosh R. Safety verification of conflict resolution maneuvers. IEEE Transactions on Intelligent Transportation Systems 2(2):110–120 (2001)

[TPS98] Tomlin CJ, Pappas GJ, and Sastry SS. Conflict resolution for air traffic management: a case study in multi-agent hybrid systems. IEEE Transactions on Automatic Control 43(4):509–521 (1998)

[USN08] USNews. The robot revolution may finally be here: As the vacuuming roomba clears the way, medical and military bots also show promise (2008)

[VKR08] Varshavskaya P, Kaelbling LP, and Rus D. Automated design of adaptive controllers for modular robots using reinforcement learning. International Journal of Robotics Research 27(3-4):505–526 (2008)

[WAS99] Wilmarth SA, Amato NM, and Stiller PF. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In IEEE International Conference on Robotics and Automation, 1024–1031. Detroit, Michigan (1999)

[Yov97] Yovine S. Kronos: A verification tool for real-time systems. International Journal of Software Tools for Technology Transfer 1:123–133 (1997)

[YSS+07] Yim M, Shen WM, Salemi B, Rus D, Moll M, Lipson H, and Klavins E. Modular self-reconfigurable robot systems: Challenges and opportunities for the future. IEEE Robotics & Automation Magazine 14(1):43–52 (2007)

[Zha06] Zhang W. State-space Search: Algorithms, Complexity, Extensions, and Applications. Springer Verlag, New York, NY (2006)

[ZKB08] Zucker M, Kuffner J, and Bagnell JA. Adaptive workspace biasing for sampling-based planners. In IEEE International Conference on Robotics and Automation, 3757–3762. Pasadena, California (2008)

[ZM08] Zhang L and Manocha D. An efficient retraction-based RRT planner. In IEEE International Conference on Robotics and Automation, 3743–3750. Pasadena, California (2008)