To appear in Journal of Artificial Intelligence Research Submitted 10/17

ScottyActivity: Mixed Discrete-Continuous Planning with Convex Optimization

Enrique Fernández-González EFERNAN@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar St, Office: 32-224 Cambridge, MA 02139 USA

Brian Williams WILLIAMS@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar St, Office: 32-224 Cambridge, MA 02139 USA

Erez Karpas KARPASE@TECHNION.AC.IL
Technion Israel Institute of Technology
Technion City, Haifa 32000, Israel

Abstract

The state of the art practice in robotics planning is to script behaviors manually, where each behavior is typically generated using trajectory optimization. However, in order for robots to be able to act robustly and adapt to novel situations, they need to plan these activity sequences autonomously. Since the conditions and effects of these behaviors are tightly coupled through time, state and control variables, many problems require that the tasks of activity planning and trajectory optimization are considered together. There are two key issues underlying effective hybrid activity and trajectory planning: the sufficiently accurate modeling of robot dynamics and the capability of planning over long horizons. Hybrid activity and trajectory planners that employ mixed integer programming within a discrete time formulation are able to accurately model complex dynamics for robot vehicles, but are often restricted to relatively short horizons. On the other hand, current hybrid activity planners that employ continuous time formulations can handle longer horizons but they only allow actions to have continuous effects with constant rate of change, and restrict the allowed state constraints to linear inequalities. This is insufficient for many robotic applications and it greatly limits the expressivity of the problems that these approaches can solve. In this work we present the *ScottyActivity* planner, that is able to generate practical hybrid activity and motion plans over long horizons by employing recent methods in convex optimization combined with methods for planning with relaxed plan graphs and heuristic forward search. Unlike other continuous time planners, ScottyActivity can solve a broad class of robotic planning problems by supporting convex quadratic constraints on state variables and control variables that are jointly constrained and that affect multiple state variables simultaneously. In order to support planning over long horizons, ScottyActivity does not resort to time, state or control variable discretization. While straightforward formulations of consistency checks are not convex and do not scale, we present an efficient convex formulation, in the form of a Second Order Cone Program (SOCP), that is very fast to solve. We also introduce several new realistic domains that demonstrate the capabilities and scalability of our approach, and their simplified linear versions, that we use to compare with other state of the art planners. This work demonstrates the power of integrating advanced convex optimization techniques with discrete search methods and paves the way for extensions dealing with non-convex disjoint constraints, such as obstacle avoidance.

1. Introduction

Due to advances in mechanical design and control, the capabilities of robots have been improving at a dramatic rate over the last few years. However, most robots operating in the real world are not autonomous. For example, in the recent DARPA Robotics Challenge, participants demonstrated very impressive humanoid robot behaviors, such as walking and climbing stairs, grabbing and using power tools to drill holes and even driving cars (Fallon, Kuindersma, Karumanchi, & Tedrake, 2015). However, all of these behaviors were controlled remotely by a human operator. Other robots are used routinely in scientific missions, such as the ones performed by Woods Hole Oceanographic Institution (WHOI) in which autonomous underwater vehicles (AUVs) are sent to collect data of scientific interest determined by experts. Although these robots operate mostly on their own while the mission is underway, they often execute fixed scripts that are hand-written by experts in a tedious, time-consuming and error-prone way. Teleoperation or script-based methods do not scale in a cost efficient way and are not appropriate for situations with long communication delays (e.g. space) or where not all the information is known in advance. For this and other cases it is desirable to have autonomous robots capable of reasoning about their goals and the environment they operate in.

Tasks that do not require reasoning over discrete conditions or effects are successfully handled by modern control approaches, such as dynamic motion planning, model-predictive control or trajectory optimization. While these methods show excellent results in dealing with complex non-linear dynamics in tight environments, they do not scale for long time horizons, and are often used either reactively or in very short missions, but not in long term robotic mission planning.

In order to plan missions for autonomous robot vehicles with state-dependent goals that are subject to temporal deadlines and coordination constraints, it becomes essential to model their dynamics with sufficient accuracy. Planners handling these missions need to consider, as a minimum, the allowed velocities that these vehicles can travel with, as well as be able to subject these robots to complex state constraints, such as being inside regions or maintaining certain distances with other robots. Planning for these missions involves reasoning over discrete and continuous conditions and effects, as well as the previously mentioned robot dynamics, coordination constraints and temporal deadlines. As a consequence, the times, states and control variables of these robots become tightly coupled. In order to find high quality feasible plans, it becomes necessary to perform joint hybrid activity and trajectory planning. Finally, typical missions often span many hours or days. Often, some activities execute in short times, such as activating or deactivating sensor suites, while others can take very long, such as traversing long distances. Since all these activities can be intrinsically connected, it is essential to handle all of them effectively over long horizons.

Over the last few years, the robotics community has had tremendous success with trajectory optimization and other sophisticated methods to control robots with many degrees of freedom and very complicated dynamics. It is now common to see, not only in simulation, but even more impressively, in real hardware, demonstrations of robots walking in complex terrains, climbing stairs, running and even jumping. However, as impressive as these demonstrations are, the robotics community has not often considered the activity planning

problem, and researchers have resorted to sequencing these complex behaviors computed using trajectory optimization by hand. These approaches have been, in general, restricted to limited horizons in which fixed time discretization works well. Therefore, these methods do not address the problem of planning missions of robot vehicles over long horizons in which autonomous activity planning is required.

In order to reason effectively with both the discrete and continuous behaviors that robots exhibit, Kongming (Li & Williams, 2008; Li, 2010) was introduced as the first approach that merged activity planning and trajectory optimization to generate practical plans for real robots. By reasoning with both discrete and continuous effects depending on control variables, Kongming made important advances in merging both worlds, and demonstrated its practical usefulness in real underwater robot missions. Unfortunately, Kongming's approach requires a fixed time discretization that hinders its ability to scale well to missions with long horizons in which short and long lived activities coexist.

On the other hand, heuristic forward search (HFS) approaches for activity planning have shown over the last two decades immense progress in the scale of the problems that they can solve. A large part of their success comes from the use of very effective heuristics such as delete-relaxations (Hoffmann & Nebel, 2001) or, more recently, landmarks (Richter & Westphal, 2010). The COLIN planner (Coles, Coles, Fox, & Long, 2012), for example, presented a heuristic forward search with delete relaxations approach to handle both temporal planning and continuous time-dependent effects. Like previous planners such as LPSAT (Wolfman & Weld, 1999) and LPGP (Long & Fox, 2003) that interleaved linear programming with discrete search, COLIN uses linear programs to test the consistency of partial states. Since COLIN does not discretize time, it can handle long horizons with short and long activities well. However, COLIN is not sufficiently expressive for solving robot hybrid activity and trajectory planning problems. COLIN only supports continuous time-dependent effects with constant rates of change and state variable constraints that are simple linear inequalities. While it can be used for some robotic hybrid activity and trajectory planning problems, this requires defining many activities with multiple discretized values for the different desired rates of change of the continuous effects (such as the velocities of each robot). This does not scale well to practical problems, as we demonstrate in our empirical evaluation section. Furthermore, many typical robot planning problems like the ones that will be discussed in this work cannot be modeled with COLIN since the state constraints are only limited to simple linear inequalities. As we will discuss later, our approach draws inspiration from COLIN in that we also interleave discrete heuristic forward search with the solving of mathematical programs.

Finally, over the last few years, the robotics community has shown a great interest in the combined Task and Motion Planning problem (TAMP) (Srivastava, Fang, Riano, Chitnis, Russell, & Abbeel, 2014; Cambon, Alami, & Gravot, 2009; Garrett, Lozano-Pérez, & Kaelbling, 2014; Lozano-Pérez & Kaelbling, 2014). Many of these approaches combine state of the art discrete activity planners with specialized motion planners for robotic applications. In order to do that, these approaches often discretize the robot states on demand as they need to, and commit early to these discretized states as soon as they find one that works. Most of the work in TAMP has focused on manipulation tasks, where planners need to handle very complex and highly non-linear and non-convex constraints over many degrees of freedom. While these approaches have shown impressive results in this domain, they

are often limited to classical activity planning, with no temporal constraints, and in which dynamics are largely ignored. They often use discretized steps and are limited to short horizons, and therefore these approaches are generally not applicable to the robot vehicle mission planning problems that we target in this work.

In this work we present ScottyActivity, a hybrid activity and trajectory planner designed to effectively plan missions for robot vehicles over long horizons. By supporting convex state constraints and control variables that often model controllable rates of change or robot velocities, ScottyActivity can model a wide range of real robotic missions. By using a continuous time formulation, ScottyActivity scales effectively to missions with long durations, even in the presence of both short and long span activities. ScottyActivity exploits recent advances in convex optimization and avoids discretizing state or control variables, which in turn prevents early commitment to bad discretization choices. ScottyActivity uses continuous convex optimization to choose these continuous states, control variables and times, but leaves this choice open until the final plan is found, allowing a great amount of flexibility to the optimizer until the end, which others have shown to provide great advantages (Toussaint, 2015).

Our planner is possible thanks to three insights. First, due to recent advances in optimization, a restricted form of quadratically constrained programs, called Second Order Cone Programs (SOCPs), can be solved efficiently for real world problems. Second, nearly all of the requirements of the robot vehicle missions that ScottyActivity targets can be encoded with cone constraints, with the exception of a non-convex term resulting from the product of control variables and time. Third, an encoding trick allows us to eliminate this non-convex term, resulting in a SOCP encoding that is very fast to solve and that our planner repeatedly uses to test the consistency of partial plans. Our SOCP encoding allows us to impose upper bound constraints on the norm of vectors of control variables (e.g. $v_x^2 + v_y^2 \leq v_{max}^2$), enforce convex quadratic state constraints (such as being inside ellipsoidal regions or ensuring a maximum distance between objects) and use the same control variables in as many simultaneous effects as needed. We do this without resorting to time, state or control variable discretization, which allows us to maintain the high performance of continuous time heuristic forward search (HFS) planners. While SOCPs are harder to solve than the linear programs that other continuous time planners use, our encoding naturally degrades to a, more efficient, linear program when the features that require quadratic constraints are not used in the problem.

The work presented here is an extension of our earlier work in which we introduced the SOCP optimization model (Fernandez-Gonzalez, Karpas, & Williams, 2017) and our initial iteration of the ScottyActivity planner (Fernandez-Gonzalez, Karpas, & Williams, 2015). Our first version introduced the use of control variables in a continuous hybrid planning approach but uses a much more restrictive consistency checking method and is limited to significantly less expressive robot applications. Compared to our later publication, this work presents many additions. First, we introduce a new search algorithm called obj-EHC that attempts to produce better plans by breaking heuristic ties with the problem objective function. Second, we present extended benchmarks in which we compare the performance of this new search algorithm to the standard EHC and we show that the returned plans are significantly better in typical robotic planning problems. Third, we add a section in which we prove that invariant convex conditions can be enforced efficiently by imposing

them at switch points. Fourth, we add additional benchmarks that show the importance of using continuous time (by comparing with Kongming) and continuous control variables (by comparing with POPF). Among the new benchmarks, we also include a comparison of our approach against a mixed integer optimization method, and show that we find plans of comparable or better quality two orders of magnitude faster. Finally, we provide many more details and additional explanations in all sections of the paper, especially in the approach, convex optimization model, heuristic and problem statement.

In order to plan efficiently over long horizons, ScottyActivity makes two fundamental assumptions: the absence of obstacles and the restriction of robot behaviors to first-order linear dynamics. In order to deal with obstacles, we propose the *ScottyPath* planner, that uses heuristic forward search and the convex model described in this work to find obstacle-free state trajectories for ScottyActivity plans. We expand the work presented here and we describe ScottyPath and its interaction with ScottyActivity in another work (Fernandez-Gonzalez, 2017). The dynamics restriction is strong, but ScottyActivity can still plan effectively common long duration robotic missions, where robot behaviors can be approximated with linear dynamics. Since ScottyActivity supports control variables and convex quadratic state constraints, we can solve rich robotic planning missions that cannot be modeled with current activity planners. However, in order to consider more detailed dynamics during plan execution, we propose an additional planner, which we are currently developing, that uses a receding-horizon formulation. This planner simultaneously combines a detailed dynamics discrete-time formulation over a limited horizon with the same simplified linear dynamics continuous-time formulation described in this work for the rest of the long horizon in the plan.

The organization of this paper is as follows. In Section 2 we describe the background in planning with continuous change that supports ScottyActivity. In Section 3 we present a motivating scenario that illustrates a real world problem that could not be solved with approaches prior to our planner. Our problem statement is described in Section 4. Section 5 describes and justifies the piecewise linear restriction on trajectories that ScottyActivity imposes and that allows us to handle complex convex conditions very efficiently. Section 6 describes the details of our planner. Our convex optimization model, one of the key innovations of our work, is presented in Section 7. Section 8 presents the experimental evaluation of our planner, including several new benchmarking domains that illustrate the new capabilities of ScottyActivity. Finally, Section 9 presents the conclusions of this work as well as avenues for future research. We leave for the appendix the new region-based system that we use to succinctly represent the unique effects and constraints that ScottyActivity supports and that are hard to express with regular Planning Domain Definition Language (PDDL) (Appendix B).

2. Related Work

While no other planner that we are aware of solves the exact same problem that our planner does, a multitude of approaches have emerged over the last decade to plan with mixed discrete-continuous change or with combined task and motion planning, and researchers have applied these planning techniques to real robotic missions. In this section we discuss the most relevant approaches and their relation with ScottyActivity.

2.1 Planning Graph Approaches

One of the first attempts to integrate planning with continuous actions was made by the Kongming planner (Li & Williams, 2008), that uses an approach that mixes an analog to the Planning Graph from Graphplan (Blum & Furst, 1997) and mixed integer optimization. Due to the features it supports, Kongming is perhaps the planner that is closest to ScottyActivity in the type of problems that both can solve. Kongming is the first activity planner to support control variables (such as controllable velocities in moving robots) affecting continuous effects. One of the main innovations introduced by Kongming consists in representing continuous effects with flow tubes, that are abstractions of the infinite number of trajectories that a continuous action can produce. Another key innovation introduced by Kongming is the Hybrid Flow Graph, the continuous analog to Graphplan's Planning Graph. Hybrid actions (i.e. actions with both discrete and continuous conditions or effects) connect initial state regions to goal regions after some fixed duration using the flow tubes generated from the system dynamics. Kongming expands the Hybrid Planning Graph with alternating action and fact layers until the goal conditions are non-mutex in the last fact layer. The problem is then encoded as a mixed logic linear (non-linear) program that contains the system dynamics constraints on the state variables and logic constraints on binary variables for the discrete conditions and effects and that is similar to Blackbox's SAT encoding (Kautz & Selman, 1999). Kongming alternates between trying to solve the ML(N)LP and adding additional layers to the graph until the ML(N)LP solver returns a solution.

Later, the Kongming planner was extended to support temporally-extended goals by reformulating them into durative actions with effects that add specific predicates that need to hold at the end of the plan. This extension also added the capability of supporting actions with flexible durations (Li, 2010; Li & Williams, 2011).

Although Kongming's approach is innovative, it suffers from performance degradation issues in medium to large problems due to the fixed time-step discretization that its graph layers are subject to. In problems in which the planning horizon is moderately large and where short and long lived activities coexist, this involves creating many layers. As the number of layers increases, identifying mutex relations becomes exponentially more complicated. This also slows down significantly the ML(N)LP solver, as each additional layer adds many more additional variables and constraints. Kongming inspires ScottyActivity in its representation of continuous effects that depend on continuous control variables.

2.2 Heuristic Forward Search Planners

Heuristic forward search approaches have dominated all planning competitions since the development of the FF planner (Hoffmann & Nebel, 2001). This approach has been extended to address problems requiring temporal coordination (Coles, Fox, Long, & Smith, 2008; Eyerich, Matmüller, & Röger, 2009). Later, the COLIN planner (Coles et al., 2012) extended these previous approaches to problems with linear time-varying processes. COLIN solves temporal planning problems with continuous effects as defined in version 2.1 of the Planning Domain Definition Language (PDDL2.1) (Fox & Long, 2003). In these problems actions can have continuous time-varying effects, whose rate of change is specified by a fixed value. In order to solve the mixed discrete-continuous planning problem, COLIN tests the consistency of partial plans with linear programs, which have been used before by other planners such as

LPSAT (Wolfman & Weld, 1999) and LPGP (Long & Fox, 2003). In these linear programs the decision variables are the continuous state variables and the times at which actions are executed. The continuous linear time-varying effects and the temporal relations between actions are encoded as constraints. The linear program is also used at each step of the search to determine the minimum and maximum possible bounds for each state variable in order to prune actions that cannot possibly be feasible at that point of the search. COLIN's heuristic is based on the Temporal Relaxed Planning Graph and delete relaxations. Similarly to MetricFF (Hoffmann, 2003), COLIN's planning graph expansion keeps track of the bounds of state variables, that are only allowed to grow (and never get smaller) as a result of the change produced by the continuous effects. Although COLIN is an efficient and capable planner, it is not expressive enough to solve the kind of robotic problems we want to target. COLIN only supports continuous effects with fixed rates of change, according to the following equation:

$$x(t_{end}) = x(t_{start}) + \text{rate-of-change} \cdot (t_{end} - t_{start}) \tag{1}$$

COLIN's formulation cannot represent continuous controllable rates of change (control variables), that are required to model, for example, the controllable velocities of moving vehicles. We can simulate this by creating multiple actions with discretized fixed rates of change. However, as we will show later, this solution does not scale well, especially in the case where multiple control variables actuating in multiple state variables are needed.

COLIN was later extended by POPF (Coles, Coles, Fox, & Long, 2010), that improves its performance by using a partial order representation of the underlying state, and OPTIC (Benton, Coles, & Coles, 2012), that supports preferences as introduced in PDDL3 (Gerevini, Haslum, Long, Saetti, & Dimopoulos, 2009). Although these planners lack the ability to represent robot behaviors naturally (due to, among others, the absence of control variables), they have been used in multiple robotic applications due to their robustness and scalability. For example, POPF was used to find automated inspection plans of underwater installations (Cashmore, Fox, Larkworthy, Long, & Magazzeni, 2014). However, in this case the planner did not explicitly consider the continuous motion of the robot, but instead chose a mission path by selecting discrete waypoints that were previously generated using random sampling motion planning techniques.

In some situations it is necessary to consider explicitly the robot motions during planning to ensure, for example, that temporal and spatial constraints are always satisfied. This is the reason why we have developed the ScottyActivity planner. ScottyActivity extends the expressivity of previous heuristic forward search planning approaches by, among others, adding support for continuous control variables, that are essential to model robotic domains.

However, ScottyActivity is not the only planner that has addressed the problem of planning with control parameters. Recently, POPCORN (Savas, Fox, Long, & Magazzeni, 2016) formalized the notion of continuous control parameters as an additional element of actions that are chosen by the planner. Contrary to ScottyActivity, POPCORN's control parameters can only be used in discrete numeric effects and not as rates of change in continuous effects. Other recent approaches have also considered continuous control parameters, but are limited to discrete time and change (Pantke, Edelkamp, & Herzog, 2016).

All the planning approaches discussed previously are limited to linear continuous effects. However, over the last few years the planning community has made attempts to use

the previous planning formalisms in non-linear settings. One popular approach consists in interleaving temporal planning with an external domain-specific solver capable of reasoning with complex non-linear change. This approach has been used successfully to solve power balancing problems in an electricity network (Piacentini, Alimisis, Fox, & Long, 2013). An alternative approach extends COLIN to handle a limited form of non-linear continuous monotonic effects using an iterative convergence method that repeatedly solves linear programs (Bajada, Fox, & Long, 2015). However, the assumptions required by this approach are not compatible with typical robot behaviors such as the ones that arise in the robotic mission planning targeted by this work. Most of the other approaches to planning with non-linear effects have focused on PDDL+ problems and are described in the next section.

2.3 PDDL+ Planners

PDDL+ (Fox & Long, 2006) significantly enhances the expressivity of PDDL by introducing processes, events and must-happen semantics. While non-linear change can also be represented with PDDL2.1, the rich semantics of PDDL+ processes and events, that make it easier to specify must-happen physical behaviors, has encouraged researchers to support non-linear effects in most PDDL+ planners. The first of such planners was UPMurphi (Della Penna, Magazzeni, Mercorio, & Intrigila, 2009), that uses an uninformed discretize and validate approach. This approach was recently improved by the DiNo planner (Piotrowski, Fox, Long, Magazzeni, & Mercorio, 2016) by introducing an efficient heuristic that vastly outperforms UPMurphi. Lately, the planning community has also explored SMT based techniques to solve PDDL+ planning problems with good results (Bryce, Gao, Musliner, & Goldman, 2015; Bogomolov, Magazzeni, Minopoli, & Wehrle, 2015; Cashmore, Fox, Long, & Magazzeni, 2016).

Most PDDL+ planners are, in general, more expressive than our planner in several ways, like their support of processes, events, must-happen semantics and continuous change that is non-linear in time. However, their semantics do not represent robot dynamics accurately, since they do not support control variables and are unable to model, for example, the motivating scenario described in Section 3.1. Several of these planners also suffer from scalability issues since they require time and state discretization.

2.4 Combined Trajectory and Motion Planning Approaches (TAMP)

Over the last few years, the robotics community has recently expressed a significant interest in the combined task and motion planning problem, and many interesting approaches have emerged.

One of the most interesting ones integrates off-the-shelf task and motion planners by using a novel representational symbolic abstraction (Srivastava et al., 2014). The architecture of this planner alternates between solving discrete symbolic planning problems with abstracted poses, verifying and refining those plans using a motion planner and generating additional discrete poses when needed. This approach is innovative in that whenever the motion planner is unable to find collision free paths between poses, objects are removed one by one in order to discover which one is responsible for the plan failure. This information is then added to the symbolic planning problem that can then decide that an object in the way should first be moved to a different location. A related approach(Cambon et al., 2009)

also interweaves a symbolic planner (MetricFF) with a roadmap motion planner. In this case the symbolic planning formulation is extended to include geometric constraints and a roadmap for each movable element is maintained and extended as necessary.

On the other hand, the FFRob planner(Garrett et al., 2014) approaches this problem by extending the heuristic ideas of the symbolic FF planner to motion planning by using a semantic attachments strategy. A state for FFRob is composed of simple true/false literals and also domain-dependent literals (such as reachability conditions) whose true/false value is determined by a test function that is evaluated lazily on demand. FFRob maintains during search a conditional reachability graph that represents the connectivity of the sampled configurations.

Another interesting approach by *Lozano-Perez et al* frames this problem as a geometric constraint-satisfaction problem (Lozano-Pérez & Kaelbling, 2014). The goal, in this case, consists in finding a sequence of activities forming a plan skeleton in which the specific robot and object poses are not bound until the CSP is solved by an off-the-shelf constraint satisfaction solver. However, the domains of the unbound variables need to be arbitrarily discretized in advance. Most of the planners discussed above approach the problem by bringing the continuous world of motion planning into the discrete symbolic planning formulation. This is often done by discretizing the continuous space either in advance or lazily during runtime as needed. The main advantage of this idea is that they can then use slightly modified off-the-shelf symbolic and motion planners. An important disadvantage, though, is the need to perform a discretization whose size needs to be chosen in advance.

Alternative approaches have tried to go in the opposite direction, by bringing the discrete symbolic world into the continuous space. These approaches use symbolic planning to generate sequences of actions in which the values of the continuous variables are not bound and then formulate large optimization problems to resolve these values. The main advantage of these methods is that no arbitrary discretization is required and that the solver can choose the best values for the continuous variables at the end in order to optimize for some criteria. This approach would have seemed intractable a few years ago, but impressive advances in trajectory optimization and non-linear solvers have made this a real possibility. For instance, (Toussaint, 2015) demonstrates this approach in a manipulation problem in which a robot picks and places cylinders and plates from a table in order to assemble the highest possible stable tower. Toussaint generates action sequences using a relatively simple symbolic planning approach but, more interestingly, solves a large optimization problem at the end in order to find the best final and intermediate positions of all the objects in order to assemble the highest tower.

Most of the approaches discussed so far have dealt almost exclusively with manipulation problems, which present on itself very complicated domain specific challenges. These studies often neglect temporal constraints or robot dynamics as these are not necessary in most manipulation problems.

The ScottyActivity planner considers problems where the robot behaviors are subject to dynamics and where temporal and state constraints are intrinsically connected. ScottyActivity needs to solve a problem in which robot behaviors and tasks need to be selected while constructing control policies jointly, in order to satisfy the problem constraints. We discuss this *hybrid activity and motion planning problem* in the next section and provide an example scenario that we later use to construct our problem statement.

3. Hybrid Activity and Trajectory Planning

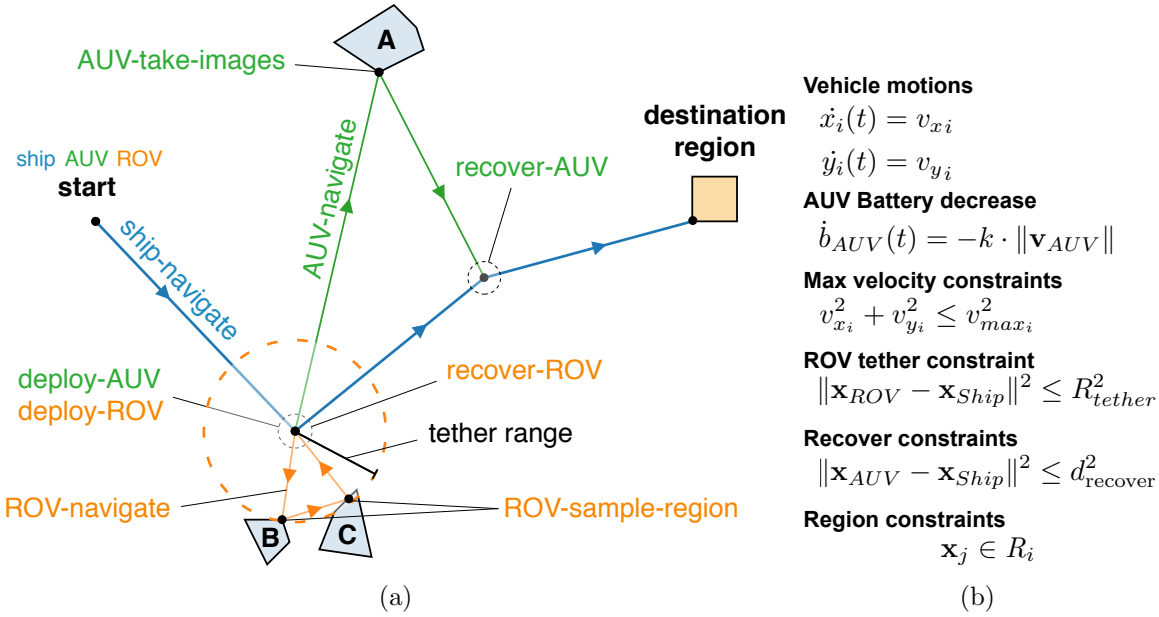
Common robotic applications require reasoning with continuous and discrete robot behaviors. These behaviors specify, for example, how robots move according to their dynamics and are often subject to state constraints. Mission goals often involve a combination of temporal and continuous and discrete state constraints in order to specify, for example, that a robot needs to visit a certain location before the deadline, or that a sample has to be acquired by the end of the mission. In order to satisfy the mission, a hybrid plan needs to specify not only what robot behaviors have to be executed and when, but also how to execute them by providing a control plan. These control plans specify, for example, the velocities that the robots need to follow while executing a given behavior. Since the robot behaviors and mission specifications are tightly connected by temporal and state constraints, the selection of the robot behaviors, their schedule and their control plans and trajectories need to be determined jointly. For example, a ship may need to meet an autonomous underwater vehicle (AUV) to pick it up. The location where the pickup takes place may not matter as long as it happens within the temporal deadlines and before the AUV runs out of battery. By jointly considering the mission constraints and the dynamics and actuation constraints of the vehicles, we can determine the appropriate location where the pickup should take place and the trajectories of the vehicles in order to minimize the distance traveled by the ship. The previous is an example of a *hybrid activity and trajectory planning problem*, in which the behaviors/activities that need to be selected, their schedule and their control and motion plan are selected jointly.

In order to solve robotic hybrid activity and trajectory planning problems we envision the *Scotty Planning System*, that decomposes the problem into simpler ones that are solved by specialized modules. First, the *ScottyActivity* planner, the object of this work, solves the hybrid activity trajectory planning problem with two assumptions: the absence of obstacles, and the restriction to first-order linear dynamics. In order to handle obstacles, the *ScottyPath* planner builds on ScottyActivity's convex optimization model presented in this work to compute obstacle free trajectories for activity plans found by ScottyActivity. These two planners and their interaction are described in detail in another work (Fernandez-Gonzalez, 2017). In order to relax the assumption of first-order dynamics and offline planning, we envision *MPCScotty*, the third planner in the Scotty system. This planner, which is under active research at the moment, executes the plans found by the combination of the previous two. MPCScotty uses a receding-horizon approach that can simultaneously handle detailed dynamics with a discrete time formulation over a limited horizon and first-order dynamics with continuous time over the rest of the plan. We describe the ScottyActivity planner in this work. Therefore, we restrict the planning problems to those without obstacles and with first-order linear dynamics.

We now describe a motivating example of a ScottyActivity hybrid activity and trajectory planning problem that is based on a real mission. This example is also used to describe the needs of our planner and to later provide a formal problem statement in Section 4.

3.1 Example Motivating Scenario

Our example scenario is modeled after a real underwater science mission to the Kolumbo volcano off the coast of Santorini (Greece) that will take place in 2019. The Kolumbo volcano



(a) (b)

Figure 1: Our motivation scenario exhibits interesting constraints such as the maximum distance constraint between the ship and the AUV (b). The solution returned by our planner shows that ScottyActivity is able to select the best position of the ship for deploying and recovering the AUV and the ROV in order to satisfy the constraints and without requiring discretization (a).

is an active submarine volcano that is geologically and biologically very interesting due to the hydrothermal vents present in its caldera. Moreover, subsea CO_2 pools not known to exist before were recently discovered nearby (Camilli, Nomikou, Escartin, Ridao, Mallios, Kilias, Argyraki, Andreani, Ballu, Campos, Deplus, Gabsi, Garcia, Gracias, Hurtos, Magi, Mevel, Moreira, Palomeras, Pot, Ribas, Ruzie, & Sakellariou, 2015). In 2019, Woods Hole Oceanographic Institute (WHOI) will lead an expedition to the Kolumbo caldera in which autonomous planning algorithms developed at MIT will be used to coordinate and plan the activities and trajectories of multiple vehicles. This mission is funded in part by the NASA PSTAR program as an analog mission to explore issues such as limited communications and harsh environments that future robotic missions to Europa and other moons in the Solar System will experience. In particular, in the third stage of this mission ScottyActivity and other planners will be used to coordinate the trajectories and scientific activities of an autonomous underwater vehicle (AUV), a ship and a Remotely Operated Vehicle (ROV) tethered to the ship. We model our example scenario after this stage of the Kolumbo mission (Figure 1).

In our example scenario, the ship is initially transporting both the AUV and the ROV to the science site. The AUV needs to take images at region A, while the ROV needs to take samples in regions B and C. All three vehicles need to reach the destination region at the end, can navigate on their own and have their own v_x, v_y velocities. The velocities can be freely chosen, but their norms are upper-bound constrained ($v_{x_i}^2 + v_{y_i}^2 \leq v_{max_i}^2 \quad \forall i \in \{\text{ship}, \text{ROV}, \text{AUV}\}$). Whenever the ROV is deployed, the ship needs to remain still at the deployment location until the ROV is recovered again. Moreover, the ROV is tethered to the ship, and therefore it can only move within a circle centered at the ship with radius the tether length ($(x_R - x_S)^2 + (y_R - y_S)^2 \leq R_{tether}^2$). Both the AUV and the ROV can be picked up when at most 2 meters away from the ship. The AUV can navigate on its own once deployed, but it has a finite battery that limits how long it can travel on its own ($\dot{b}_{\text{AUV}} = -k \cdot \|\mathbf{v}_{\text{AUV}}\|$).

Figure 1 shows a valid plan for this mission that presents interesting characteristics. First, by stationing the ship and deploying the ROV at the appropriate location, both sampling regions can be visited without violating the tether range constraint, which saves time and fuel. Second, the AUV battery is not large enough to reach the destination region on its own. Therefore, the ship meets the AUV at a non fixed nor discretized intermediate location, picks it up and transports it to the destination region. Finally, the battery decrease function is also interesting, as it depends on the norm of the velocity of the AUV. In this work we show how ScottyActivity can solve this problem by integrating heuristic forward search and convex optimization.

The example scenario described in this section showcases the needs that need to be modeled in order to solve this type of problems. First, we need a way to express the requirements of the robot behaviors that allow them to move according to their controllable velocities, or the fact that the AUV battery gets drained according to the velocity that the AUV moves at. Moreover we need to model the state constraints that have to be satisfied at certain points in the mission: e.g. the ROV always satisfying the tether range and the AUV staying within region A while the images are taken, to name a few. Finally, we need to be able to specify an objective to minimize, such the total duration of the mission or the

distance traveled by the ship. The description of this example scenario in the PDDL-like syntax used by ScottyActivity is shown in Section 4.

4. Problem Statement

In this section we define ScottyActivity's problem statement. ScottyActivity is designed to solve a broad range of robotic planning problems like the one presented in Section 3.1. We first describe the elements we need to describe the hybrid problem we solve, which we define at the end of this section.

4.1 State

Definition 1 (*State*)**.** The *state* of the system, $\mathbf{s} = \langle \mathbf{x}, \mathbf{p} \rangle$, is a tuple of *state variables*, $\mathbf{x}$, and *propositional variables*, $\mathbf{p}$. The state variables are given by a vector of real valued variables, $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle \in \mathbb{R}^n$. The propositional variables are given by a vector of propositions that can be true or false, $\mathbf{p} = \langle p_1, p_2, \ldots, p_l \rangle$.

Definition 2 (*Resource*)**.** A *resource* is a type of state variable that is subject to special conditions that we describe later in this section.

In the example scenario, the state variables are the x and y coordinates of the ship, the AUV and the ROV as well as the battery level of the AUV. Additionally, the battery level is a state variable that is a resource. The propositional variables indicate, for example, whether the images and samples have been taken, or whether the AUV and ROV are deployed or onboard the ship.

4.2 Control Variables and Continuous Effects

State variables change their value through continuous effects operating on them. These continuous effects may be seen as behaviors that enable or disable robot dynamics. The change on the state variables due to the presence of continuous effects depends on the *control variables* that these effects make use of.

Definition 3 (*Control Variables,* $\mathbf{c}$)**.** The control variables vector, $\mathbf{c}$, is a vector of control variables $\mathbf{c} = \langle c_1, c_2, \ldots, c_m \rangle$, where each control variable c_j is a real valued parameter that is continuously controllable within its fixed lower and upper bounds, c_{j_l} and c_{j_u}.

In the example scenario, the control variables vector is a vector of the v_x, v_y velocities of the ship, the AUV and the ROV, $\mathbf{c} = \langle v_{x\,ship}, v_{y_{ship}}, v_{x\,AUV}, v_{y\,AUV}, v_{x\,ROV}, v_{y\,ROV} \rangle$.

Control variables can be subject to control variable constraints. We restrict the control variable constraints to convex quadratic constraints for reasons that will become apparent when we describe, in Section 7, the convex optimization program that ScottyActivity uses.

Definition 4 (*Convex Quadratic Control Variable Constraint*)**.** A *convex quadratic control variable constraint* is a constraint in the form of $g(\mathbf{c}) \leq 0$, where $g : \mathbb{R}^m \to \mathbb{R}$ is a convex quadratic function operating on the vector of control variables, $\mathbf{c}$.

Recall that any quadratic function $g : \mathbb{R}^m \to \mathbb{R}$ can be written as $g(\mathbf{c}) = \mathbf{x}^T A\mathbf{x} + \mathbf{k}^T\mathbf{x} + b$, where $A \in \mathbb{R}^{m \times m}$, $\mathbf{k} \in \mathbb{R}^m$, and $b \in \mathbb{R}$ are constants. Function g is convex if and only if

A is a positive semidefinite matrix (Boyd & Vandenberghe, 2004). Note that since any linear expression is also a convex quadratic expression, it is also possible under our model to subject control variables to linear inequality constraints.

In the example scenario, there are three convex quadratic constraints operating on the control variables. These three constraints limit the l^2-norm of the velocities of each of the three vehicles, $\|\mathbf{v}_v\|_2 \leq v_{max_v} \quad \forall v \in \{\text{ship}, \text{ROV}, \text{AUV}\}$. Since l^2-norms are convex quadratic functions, the previous constraint can be written as a convex quadratic constraint: $v_{x_v}^2 + v_{y_v}^2 - v_{max_i}^2 \leq 0$.

As described in the beginning of this section, continuous effects produce the change in state variables. In order to maintain a continuous time formulation and enable ScottyActivity to plan efficiently for long horizons, we restrict continuous effects to first order dynamics. That is, a continuous effect operating on a state variable induces a rate of change per unit time in the state variable that is a function of the control variables vector and that does not depend on time. In practice, this means that the change in state variables due to continuous effects is linear in time.

Definition 5 (*Continuous Effect*)**.** A *continuous effect* is a tuple $\langle x, f \rangle$ where x is the state variable that is subject to the effect and $f : \mathbb{R}^m \to \mathbb{R}$ is a function of the control variables vector. An active continuous effect eff on variable x induces a rate of change $\dot{x}_{eff}(t)$ on x such that $\dot{x}_{eff}(t) = f(\mathbf{c}(\mathbf{t}))$.

Note that while the change produced by a continuous effect is linear in time, the values of the control variables are allowed to change throughout the duration of the continuous effect. The change produced in state variable x due to continuous effect *eff* being active from t_a to t_b is given by:

$$\Delta x_{eff}(t) = \int_{t_a}^{t} f(\mathbf{c}(\tau))d\tau, \quad t_a \leq t \leq t_b \tag{2}$$

Continuous effects are additive. That is, multiple continuous effects can be operating on a state variable x during an interval of time. In that case, the rate of change of state variable x is the sum of the rates of change induced by each continuous effect:

$$\dot{x}(t) = \sum_{eff \in E_x(t_a,t_b)} \dot{x}_{eff}, \qquad t_a \leq t \leq t_b \tag{3}$$

$$x(t) = x(t_a) + \sum_{eff \in E_x(t_a,t_b)} \Delta x_{eff}(t), \qquad t_a \leq t \leq t_b \tag{4}$$

where $E_x(t_a, t_b)$ is the set of active continuous effects operating on x between t_a and t_b.

Note that as we described above, state variables only change their value due to the presence of continuous effects operating on them. Therefore, the schedule of what continuous effects are applied to each state variable and when these effects are active, together with the trajectory of the control variables, $\mathbf{c}(t)$, is sufficient to fully determine the trajectory of all state variables, $\mathbf{x}(t)$.

While the description above can be used with any choice of f, we allow two types of continuous effects in this work. These are described below.

4.2.1 Controllable Linear Time-varying Continuous Effects

The first type of continuous effect that we describe induces a rate of change that is a linear combination of the control variables.

Definition 6 (*Controllable linear time-varying continuous effect, CLTE*)**.** A *CLTE* is defined by a tuple $\langle x, \mathbf{k} \rangle$, where x is the state variable subject to the effect and $\mathbf{k} \in \mathbb{R}^m$ is a constant vector. A *CLTE* changes a state variable linearly in time with a rate of change that is a linear combination of its control variables. The change in state variable x up to time t due to an ongoing *CLTE* that started at time 0 is given by

$$\Delta x_{CLTE}(t) = \int_0^t \mathbf{k}^T \cdot \mathbf{c}(\tau) d\tau \tag{5}$$

In the example scenario, the *CLTEs* allow the vehicles to move according to their control variable velocities. There are six *CLTE* effects, one for each position variable of each of the three vehicles. For example, the two CLTE effects operating each on x_{auv} and y_{auv} induce the rates of change $\dot{x}_{AUV} = v_{x\,AUV}$ and $\dot{y}_{AUV} = v_{y\,AUV}$.

4.2.2 Resource-constrained Norm Effects

The other type of continuous effects that we allow describe a change in a state variable that depends on the l^2-norm of a vector of control variables. This type of effect is useful to model, for example, how the battery of a vehicle decreases as a function of the magnitude of the velocity that the vehicle moves with. For reasons that will become apparent when we introduce our convex optimization model in Section 7, we restrict this effect to only be applied to state variables that are *resources*.

Definition 7 (*Resource-constrained norm effect, RNE*)**.** A *RNE* is given by a tuple $\langle x, k, \mathbf{c}_e, f \rangle$, where $k \in \mathbb{R}_{>0}$ is a positive real constant, $\mathbf{c}_e$ is a vector of m or fewer control variables, and $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a real nonnegative function. A *RNE* decreases the value of a constrained state variable (resource) x with a rate of change proportional to the function f of the l^2-norm of $\mathbf{c}$, $f(\|\mathbf{c}\|_2)$.

The change on a continuous state variable x up to time t due to an ongoing *RNE* that started at time 0 is given by:

$$\Delta x_{RNE}(t) = -\int_0^t k \cdot f(\|\mathbf{c}_e(\tau)\|) d\tau, \quad k \geq 0 \tag{6}$$

In this work we consider two types of *RNEs*. A *linear norm effect* (LNE), $\langle x, k, \mathbf{c}_e, x \to x \rangle$, produces a decrease rate proportional to the norm of $\mathbf{c}_e$. A *linear squared norm effect* (LSNE), $\langle x, k, \mathbf{c}_e, x \to x^2 \rangle$, produces a decrease rate proportional to the square of the norm of $\mathbf{c}_e$.

In the example scenario, the *navigate* activity has a *LNE* effect that makes the battery of the AUV decrease with a rate proportional to the norm of its velocity ($\dot{b}_{AUV}(t) = -k \cdot \|\mathbf{v_{AUV}}\|$). This is equivalent to stating that the battery decrease is proportional to the distance traveled by the AUV. The same effect depending on the square of the norm ($\|\mathbf{v_{AUV}}\|^2$) can be achieved by using a *LSNE* effect instead of the *LNE* one. Since continuous effects are additive, it is also possible to define a change in the battery that is a linear combination of these terms and other *CLTE* effects.

4.3 State Constraints

Robots are often subject to state constraints. Some of these constraints may be intrinsic to their dynamics or modes of operation. For example, the AUV in the example scenario can only continue moving while its battery level is greater than 0. Similarly, the ROV can only separate from the ship a distance smaller than the length of the tether than connects them. Other state constraints, however, are mission dependent. For example, in order to successfully capture the images required for the mission, the AUV needs to be inside region A while the pictures are taken.

We restrict the state constraints in our model to convex quadratic constraints. This restriction allows us to perform efficient checks for constraints that need to be maintained over arbitrarily long durations, as explained in Section 5, and to use an efficient convex model and a state of the art convex quadratic solver, as explained in Section 7. However, note that convex quadratic constraints are sufficient to express a wide range of real-world constraints that appear in typical robotic missions, like the ones shown in the example scenario.

Similarly to the convex quadratic control variable constraints defined in Section 4.2, state variable constraints are defined as follows:

Definition 8 (*Convex Quadratic State Constraint*)**.** A *convex quadratic state constraint* is a constraint in the form of $g(\mathbf{x}) \leq 0$, where $g : \mathbb{R}^n \to \mathbb{R}$ is a convex quadratic function operating on the vector of state variables, $\mathbf{x}$.

Additionally, for reasons that we explain in Section 7, we restrict resources to only be subject to *greater or equal than* constraints.

As mentioned before, the example scenario presents multiple state constraints. The ROV tether constraint, for example, is represented with the convex quadratic constraint: $(x_{ROV} - x_{ship})^2 + (y_{ROV} - y_{ship})^2 - R^2_{tether} \leq 0$. The constraint that forces the AUV to remain inside the images region is also a convex quadratic constraint. In particular, the images region is a polyhedron and the constraint is, therefore, a linear inequality constraint in the form of $A \cdot \mathbf{x}_{AUV} \leq \mathbf{b}$. The battery level of the AUV is a resource that is subject to always be nonnegative, $b_{AUV} \geq 0$.

4.4 Hybrid Durative Activities

In the course of a typical robotic mission, robot behaviors may need to be engaged or disengaged at different times and state constraints may become active depending on the behaviour currently being executed or the goal that the robot is currently trying to achieve. While the main focus of our planner is dealing with continuous behaviors efficiently over long horizons, we also need to be able to reason with discrete conditions and effects in order to find plans for typical robotic missions. We use durative activities like the ones that have long been used by the activity planning community to model the switched behaviors described before. In particular, we use *hybrid durative activities*, which are similar to PDDL2.1 (Fox & Long, 2003) activities except for some differences that we highlight below.

Definition 9 (*Hybrid Durative Activity*)**.** A *hybrid durative activity* a is given by the tuple $\langle dur, pre_{\vdash}, eff_{\vdash}, pre_{\leftrightarrow}, eff_{\leftrightarrow}, pre_{\dashv}, eff_{\dashv} \rangle$, where:

- *dur* is the tuple $\langle d_l, d_u \rangle$ that describes the minimum and maximum duration of a. As in PDDL 2.1, the duration of the activity is assumed to be controllable by the planner.
- $pre_{\vdash}$($pre_{\dashv}$) are the conditions that must hold at the *start* (*end*) of a. These conditions can be of two types. First, a propositional condition can require a proposition $p_j \in \mathbf{p}$ to hold. Second, a condition can be a state variable constraint as defined in Section 4.3.
- $eff_{\vdash}$($eff_{\dashv}$) are the *starting* (*ending*) effects of a that indicate the resulting change in the state as a result of applying the activity. A set of effects $eff_x, x \in \{\vdash, \dashv\}$ consists of:
  - eff_x^-, the set of propositions to be deleted (set to *false*) from the state.
  - eff_x^+, the set of propositions to be added (set to *true*) to the state.
- $pre_{\leftrightarrow}$ are the invariant conditions of a that must hold throughout the duration of the activity. These can also be propositions that need to hold or state variable constraints.
- $eff_{\leftrightarrow}$ are the *continuous effects* of a as defined in Section 4.2 that are active while the activity is being executed.

The main differences between *hybrid durative activities* and PDDL2.1 durative activities are two. First, we support continuous effects that depend on control variables. Second, we support convex quadratic constraints. The PDDL2.1 standard allows arbitrary polynomial expressions, which include convex quadratic constraints. However, most PDDL2.1 planners support only linear constraints, and we are not aware of any other activity planner that supports convex quadratic constraints.

4.5 The PDDL-S Problem

Having defined the elements that model robot behaviors, activities and constraints, we now proceed to define formally the problem that the ScottyActivity planner solves. ScottyActivity solves a hybrid activity and trajectory optimization planning problem that we call a *PDDL-S* planning problem.

Definition 10 (*PDDL-S Problem*)**.** A *PDDL-S* problem is a planning problem given by the tuple $\langle I, G, CV, A, O \rangle$, where:

- $I = \langle \mathbf{x}_0, \mathbf{p}_0 \rangle$ is the initial state, which is a complete assignment to the state variables, $\mathbf{x}_0 = \mathbf{x}(0)$, and propositional variables, $\mathbf{p}_0 = \mathbf{p}(0)$ at the beginning.
- $G = \langle S_G, P_G \rangle$ is the goal. The goal consists of a set of state variable constraints, S_G, that need to be satisfied at the end of the plan and a set of propositional variables P_G whose value needs to be *true* at the end of the plan.
- $CV = \langle \mathbf{c}, CC \rangle$ is the tuple of the vector of control variables, $\mathbf{c}$, and the convex quadratic constraints operating on the control variables, CC, as defined in Section 4.2.
- A is the set of hybrid durative activities.

- O is the objective function.

The objective function of a *PDDL-S* problem is the function that the planner aims to minimize. We restrict the objective O to a linear combination of one or more of the following terms:

- The total duration of the plan (plan makespan).
- The value of a state variable at the end of the plan. In the case of resources, their coefficients are limited to negative values (i.e. resources can only be maximized).
- The sumproducts of the norms (or squared norms) of a vector of control variables and the durations it was active for, i.e. $\int_0^T \|\mathbf{c}_e(\tau)\|^{\{1,2\}}\, d\tau$.

For example, in the motivating scenario, one of the terms minimized is $\int \|\mathbf{v_{SHIP}}\|\, dt$, which minimizes the distance traveled by the ship. A similar term involving, instead, the square of the norm is also possible. This is useful when the control effort needs to be minimized. This is typically achieved by minimizing the sum of the squares of the control variables throughout the plan, which is the same as minimizing the square of the norm of a control variable vector of all control variables ($\int \|\mathbf{c}(\tau)\|^2\, d\tau = \int \sum_{c_i} c_i^2(\tau) d\tau$).

The main differences between PDDL2.1 problems and PDDL-S problems lie in the definition of the continuous effects and control variables supported by our hybrid durative activities.

4.6 Solution to a PDDL-S Problem

The solution to a PDDL-S problem is given by a plan.

Definition 11 (*PDDL-S Plan*)**.** A *PDDL-S Plan* is a tuple $\langle S, \mathbf{f_c} : \mathbb{R} \to \mathbb{R}^m \rangle$, where

- S is the activity schedule, and is given by a list of triples $\langle a, t_s, d \rangle$, where a is an activity, t_s its start time and d its duration. The latest end time of all activities is denoted by T and corresponds to the plan makespan.
- $\mathbf{f_c} : [0, T) \to \mathbb{R}^m$ is the control trajectory. The control trajectory assigns a value to all control variables at every time t between the start and the end of the plan.

For reasons explained in Section 5, the output of ScottyActivity is a PDDL-S plan with piecewise constant control.

Definition 12 (*PDDL-S Plan with Piecewise Constant Control*)**.** A *PDDL-S Plan with Piecewise Constant Control* is a PDDL-S plan in which the control trajectory is a piecewise constant function where the change points occur at the start and end times of the activities in the plan schedule.

Given a *PDDL-S plan*, the trajectories of all state variables are fully determined at all times throughout the duration of the plan. In order to compute the value of a state variable x_i at time t, $x_i(t)$, it suffices to apply from 0 to t the continuous effects of the activities for the durations and values of the control variables specified in the plan.

A *valid PDDL-S plan* satisfies all constraints defined by the PDDL-S problem.

Definition 13 (*Valid PDDL-S Plan*)**.** A *valid PDDL-S plan* is a *PDDL-S plan* such that:

1. For each activity a which starts at time t, all its discrete and continuous *at start* conditions are satisfied right before time t.
2. For each activity a which ends at time t, all its discrete and continuous *at end* conditions are satisfied right before time t.
3. For each activity a which is ongoing at time t, all its discrete and continuous *over all* conditions are satisfied at time t.
4. The trajectories of the control variables satisfy each global control variable constraint throughout the duration of the plan.
5. The final state at the end of the plan satisfies the goal constraints.

An *optimal PDDL-S plan* is a *valid PDDL-S plan* such that objective O takes the minimum possible value. ScottyActivity finds *valid PDDL-S plans* but it is not guaranteed to find *optimal PDDL-S plans* due to the greedy nature of its search algorithm, as will be explained in Section 6.

5. Efficient Satisfaction of Convex Conditions over Arbitrarily Long Horizons Through Piecewise Linear State Trajectories

In this section we present ScottyActivity's approach to satisfying convex state constraints through arbitrarily long durations in an efficient manner. State constraints are imposed by activities as their *at start*, *at end* and *over all* conditions. One issue the planner must address is to satisfy state constraints throughout continuous time, and not just at sampled events. The second issue is that the method for satisfying state constraints must scale computationally to long horizon problems.

Most robotic planning algorithms enforce these constraints by discretizing states at fixed timesteps and imposing the constraints at each discretized state. Many of these algorithms ignore the constraints between the discretized points of the trajectory. While this strategy does not guarantee that the conditions are satisfied at all times, this usually works well when the discretization time is sufficiently small. Unfortunately, these approaches do not work well for long time horizons, or when different activities happen at a different time scale.

One of the advantages of the ScottyActivity planner is that it performs equally well for short and long time horizons and different time scales since time is not discretized. While avoiding time discretization greatly helps performance, it imposes the challenge of how to enforce maintenance conditions throughout horizons that can be arbitrarily long. In order to enforce invariant conditions in an efficient manner, we restrict the trajectories of the control variables chosen by the planner to be piecewise constant. We do not restrict how many segments the planner can select. Therefore, the trajectories of the control variables are given by:

$$\mathbf{c}(t) = \left[c_1(t), c_2(t), \ldots, c_{\|CV\|}(t)\right] = \mathbf{c}_j, \quad t_j \leq t < t_{j+1}, \quad j \in 0 \ldots N-1 \tag{7}$$

, where N is the total number of piecewise constant segments, t_j, called a *switch point*, is the starting time of segment j, $\mathbf{c}(t)$ is the vector of values of all control variables in CV at time t and $\mathbf{c}_j$ is the vector of constant values of all control variables during segment j, between t_j and t_{j+1}.

As described in the problem statement (Section 4), continuous change in the state variables only occurs as a result of the action of continuous effects. Recall that the change in a state variable x subject to a continuous effect from the time the effect starts, t_a, to time t can be expressed as:

$$\Delta x_{eff}(t) = \int_{t_a}^{t} f(\mathbf{c}(\tau))d\tau, \quad t_a \leq t \leq t_b \tag{8}$$

, where f is a function of the control variables that depends on the type of continuous effect. Since the control variables are piecewise constant, $f(\mathbf{c}(\tau))$ is piecewise constant and the previous equation can be simplified to:

$$\Delta x_{eff}(t) = f(\mathbf{c}_l)(t_{l+1} - t_a) + \sum_{k=l+1}^{l+n-1} f(\mathbf{c}_k)(t_{k+1} - t_k) + f(\mathbf{c}_{l+n})(t - t_{l+n}) \tag{9}$$

$$t_l \leq t_a \leq t_{l+1} \leq \ldots \leq t_{l+n} \leq t \tag{10}$$

, where t_l is the last switch point before t_a and there are n switch points between t_a and t. Equation (9) shows that the change in state variable x is linear in time. Since all continuous effects can be expressed in this way and are additive, the piecewise constant restriction on control variables results in state variable trajectories that are piecewise linear in time. The beginning and ends of the linear segments correspond to the switch points of the trajectory of control variables.

Therefore, the trajectory of the state variables in the linear segment l between consecutive switch points t_l and t_{l+1} can be written as:

$$\mathbf{x}(t) = \mathbf{x}(t_l) + \mathbf{C}_{\mathbf{x}}(t_l \to t_{l+1}) \cdot (t - t_l), \quad t_l \leq t \leq t_{l+1} \tag{11}$$

$$\mathbf{x}(t) = \left[x_1(t) \ldots x_{\|V\|}(t)\right]^T \tag{12}$$

$$\mathbf{C}_{\mathbf{x}}(t_l \to t_{l+1}) = \left[C_{x_1}(t_l \to t_{l+1}) \ldots C_{x_{\|V\|}}(t_l \to t_{l+1})\right]^T \tag{13}$$

, where $C_{x_i}(t_l \to t_{l+1})$ is a constant value that represents the constant rate of change in state variable x_i due to all the continuous effects operating on the state variable and that is a function of the constant vector of control variables $\mathbf{c}_l$ at segment l.

The piecewise linear restriction on state variables is very useful for our planner. It allows us to impose conditions over long horizons in an efficient way, without needing to resort to time discretization. We can do this as long as the conditions are *convex*, which they are given our problem statement. In effect, convexity properties ensure that a linear segment is fully contained in a convex set as long as the ends of the segment are contained in the set. As a consequence, in order to ensure that an *invariant* convex state condition is satisfied at all times, we only need to ensure that the convex condition is satisfied at the *switch points*

of the state trajectory. The switch points can be separated in time arbitrarily and therefore this is an efficient way to enforce invariant conditions. This is shown with the following lemma and its corresponding proof.

Lemma 1. *If the switch points of the piecewise linear state trajectory are contained in a convex set, the full trajectory is contained in the convex set.*

Proof. A set $S \subseteq \mathbb{R}^n$ is convex if and only if $\forall \mathbf{a}, \mathbf{b} \in S$ and $\forall \lambda \in [0, 1]$ we have that $(1 - \lambda)\mathbf{a} + \lambda\mathbf{b} \in S$ as well. Since equation (11) describes a straight line of a segment of the piecewise linear trajectory, it can be reformulated in terms of the switch points (or extreme points of the segments) as $\mathbf{x}(t) = \mathbf{x}(t_a) + \frac{t-t_a}{t_b-t_a}(\mathbf{x}(t_b) - \mathbf{x}(t_a))$. Taking $\mathbf{a} = \mathbf{x}(t_a), \mathbf{x} = \mathbf{x}(t_b), \lambda = \frac{t-t_a}{t_b-t_a}$, we get that $\mathbf{x}(t) = (1-\lambda)\mathbf{a}+\lambda\mathbf{b}$. Since S is a convex set and $a, b \in S$, from the definition of convexity, $\mathbf{x}(t) \in S$ as well. This is true for any value of $t \in [t_a, t_b]$ and for any value of the control variables, as long as they are constant, which they are, given the piecewise constant restriction imposed earlier. □

The restriction of piecewise constant control variables and piecewise linear state trajectories does not affect the completeness of our planner.

Theorem 1 (Completeness of PDDL-S Plans with Piecewise Constant Control)**.** *If a PDDL-S problem has a solution, there always exists a solution that is a PDDL-S plan with piecewise constant control.*

Similarly, this restriction does not affect the optimality of the plans that can be obtained.

Theorem 2 (Optimality of PDDL-S Plans with Piecewise Constant Control)**.** *The optimal solution to a PDDL-S problem, if one exists, is a PDDL-S plan with piecewise constant control.*

In effect, our linear time dynamics and the absence of obstacles, curvature constraints or other non-convex constraints ensure that any problem solvable with an arbitrarily changing state trajectory is also solvable with a piecewise linear one. Furthermore, since both the state conditions and the objective are required to be convex, the piecewise linear restriction of state variables does not affect optimality either under our action model (the optimal solution is the piecewise linear one). We present the proof for both theorems in Appendix A.

As we will explain in Section 6, the maintenance conditions are only added or removed whenever an activity starts or ends (an event). This means that the invariant conditions do not change in between events. Therefore, our approach method will place the switch points of the piecewise linear trajectories at the starts and ends of activities, as this is sufficient to capture the requirements of the problem.

5.1 Typical Maintenance Convex Conditions in Robotic Applications

We now proceed to give examples of useful convex conditions that commonly arise in robotic planning problems and that, as we have seen, our planner can enforce efficiently.

- **Remain inside a convex region** (Figure 2a). As explained in the previous region, we can enforce that a robot remains inside a convex region while moving by imposing

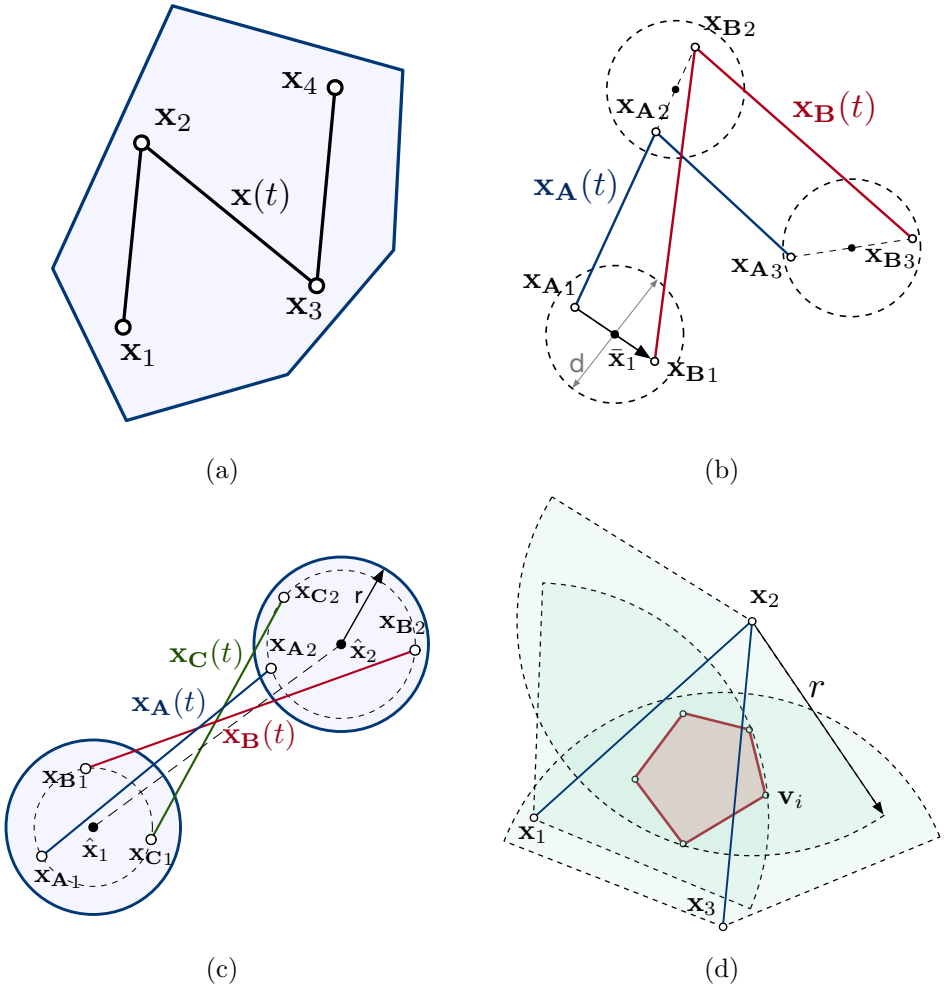


Figure 2: Interesting convex conditions that arise in robotic planning problems..

that the switch points of the trajectory be inside the convex region. Since trajectories are piecewise linear, ensuring that the switch points are contained in the convex region guarantees that the full trajectory is also contained.

- **Proximity constraint** (Figure 2b). We can impose that agents A and B stay within a maximum distance d while performing an activity. Let $\mathbf{x_A}(t)$ and $\mathbf{x_B}(t)$ be the piecewise linear trajectories of A and B. Imposing that both agents are within distance d is equivalent to imposing that $\mathbf{x_A}(t) - \mathbf{x_B}(t) \in C_d$ where C_d is a circle of radius d centered in the origin. Since C_d is a convex set and $\mathbf{x_A}(t)$ and $\mathbf{x_B}(t)$ are piecewise linear, $\bar{\mathbf{x}}(t) = \mathbf{x_A}(t) - \mathbf{x_B}(t)$ is piecewise linear too, and we only need to impose that

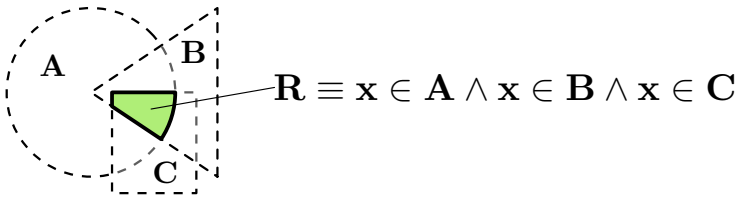


Figure 3: Complex condition defined by the intersection of simple convex sets.

$\bar{\mathbf{x}}(t_j) \in C_d$ at each switch point j to ensure that A and B are always within distance d.

- **N vehicles inside a circle of radius** r (Figure 2c). The center of the n vehicles is given by $\hat{\mathbf{x}}(t) = \frac{\sum^n \mathbf{x_i}(t)}{n}$. The condition that all vehicles remain inside a circle of radius r is equivalent to the n conditions $||\mathbf{x_i}(t) - \hat{\mathbf{x}}(t)|| \leq r$. Since $\mathbf{x_i}(t) - \hat{\mathbf{x}}(t)$ is piecewise linear due to being a linear combination of piecewise linear trajectories and since the norm constraint defines a convex set, the n conditions only need to be imposed at the switch points to ensure that all n vehicles are always inside the circle of radius r at all times.

- **Coverage constraint of a polygonal region** (Figure 2d). We can similarly impose that a circle of radius r centered at a moving vehicle always covers a fixed polygonal region. This is equivalent to satisfying the convex constraints $||\mathbf{x}(t) - \mathbf{v_i}|| \leq r$ for each of the $\mathbf{v}_1 \ldots \mathbf{v}_m$ vertices of the polygon. Again, since the m constraints are convex and the motion of the vehicle is piecewise linear, we only need to impose the constraints at each of the switch points of the trajectory of the vehicle.

Further since the intersection of convex sets is convex, we can represent complicated convex regions by intersecting simple convex primitives. Figure 3 shows, for example, how a complex region can be achieved by intersecting a circle, a triangle and a rectangle. Ensuring that a vehicle remains inside such region can be achieved by imposing that the switch points of the trajectories remain inside the circle, the rectangle and the triangle at the same time.

6. Planning Approach

In this section we describe how the ScottyActivity planner works. We begin by providing a high level overview of the ScottyActivity planner. We then describe how the successor search states are generated, the heuristic that we use and the algorithms that guide the search. A core element of the ScottyActivity planner is the convex mathematical program that we use to test the consistency of partial states. As explained earlier, our formulation makes it possible to use both continuous time and continuous control variables, which allows ScottyActivity to plan for long horizons efficiently. Since this is one of the main contributions of our work, we describe the full details of our mathematical program in Section 7.

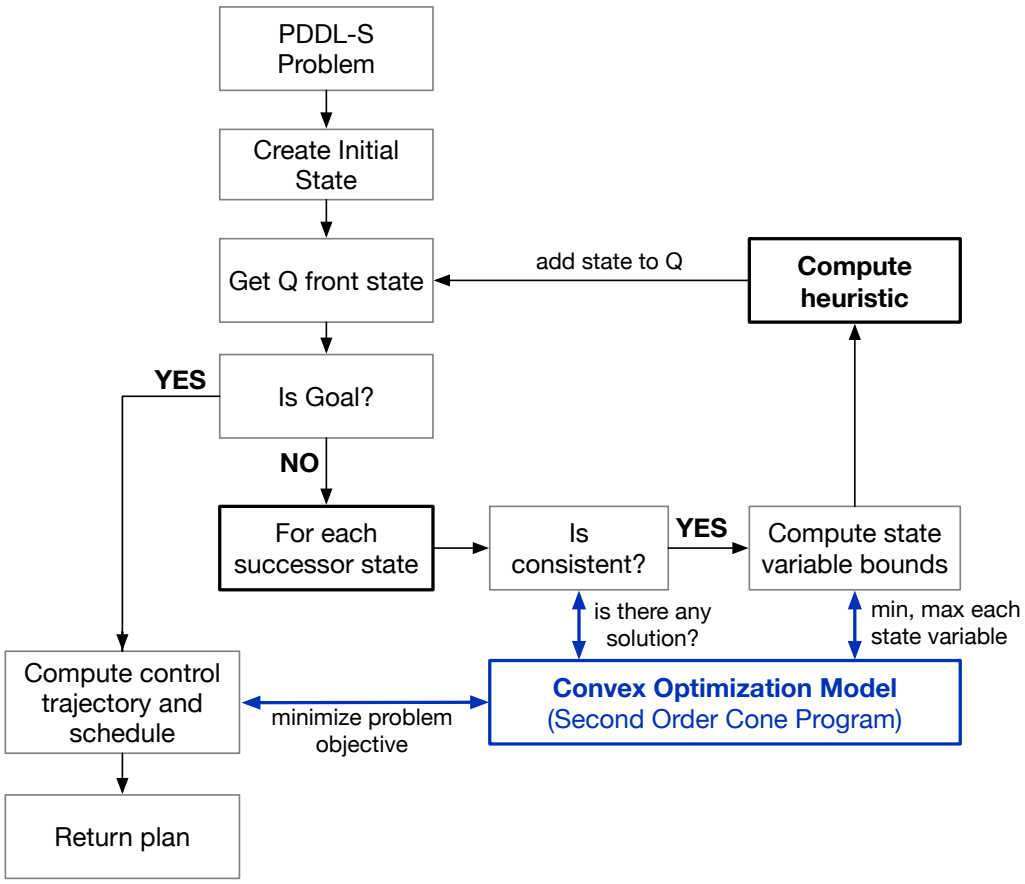


Figure 4: Informal diagram describing the overall flow of the ScottyActivity planner. The blue box indicates that the convex model is used as a module that is queried at different stages of the planning process and is not part of the flow.

6.1 ScottyActivity In a Nutshell

Recall that the plan that ScottyActivity generates consists both of an activity schedule and a control plan, given as a piecewise constant trajectory of the control variables. ScottyActivity generates the activity schedule and the control plan concurrently, through a heuristic forward search that selects the order of the activities in the schedule and a mathematical program that tests the feasibility of the schedule by finding a valid control trajectory, state trajectory and the execution times of the activities. An informal diagram describing how the ScottyActivity planner works is presented in Figure 4. We provide detailed descriptions of each step of the algorithm in this section.

Our method draws inspiration from COLIN (Coles et al., 2012), LPGP (Long & Fox, 2003), LPSAT (Wolfman & Weld, 1999) and previous planners (Stefik, 1981) in that the discrete search is interleaved with consistency checks using an optimization approach. We use heuristic forward search to find an ordered sequence of starts and ends of activities that are analogous to the start and end snap actions used by many temporal planners (Long & Fox, 2003; Coles et al., 2008). We call each start or end of an activity an *event*. We

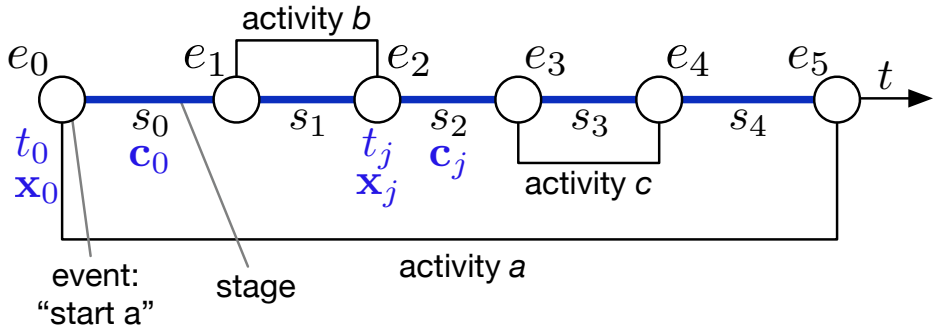


Figure 5: A plan skeleton is given by an ordered sequence of start and end events, e_j. Plan skeletons do not have an assigned control or state trajectory, or event execution times. A mathematical program is used to test the consistency of plan skeletons by finding feasible values for the control trajectory $\mathbf{c}(t)$, state trajectory $\mathbf{x}(t)$ and event execution times, t_j.

assume that no two events can co-occur, and that they are totally ordered. This allows the planner to consider the introduction of only one event during each planning step. While some heuristic forward search planners allow some flexibility in the order of these events (Coles et al., 2010), we leave this extension for future work. Note that, while events cannot co-occur, activities can (Figure 5).

We call a *stage* the period of time between consecutive events. In the piecewise constant control trajectory that ScottyActivity finds, control variables have constant values during stages, and the value changes can happen at each event. As described in Section 5, the state variables change linearly in time during stages.

Every search state defines a partial schedule of totally ordered events, that we call a plan skeleton. The plan skeleton only defines the sequence of starts and ends of activities that are selected and in what order. Search states are constructed so that their plan skeletons satisfy all the discrete conditions imposed by the activities in the partial schedule. However, plan skeletons do not define the control trajectory, the state trajectory or the execution times of the events. In order to check that the plan skeleton can satisfy the continuous constraints, we solve an optimization problem that tries to find a feasible control trajectory, state trajectory and the execution times of the activities. For intermediate search states, this optimization problem is used as a feasibility check. The values returned by the solver are, therefore, discarded. When the search state that satisfies the goal conditions is found, the optimization problem is used one more time to find the control trajectory and activity execution times that minimize the problem objective. These values along with the event schedule are returned as the solution plan.

Successor search states extend the parent plan skeleton with a new start or end event at the end. The optimization problem for successor plan skeletons is solved from scratch every time, and no intermediate control trajectories are reused. There are multiple reasons why that is the case. First, we do not know what future events could be added to the plan later, and therefore we want to avoid early commitment to values that could make the plan infeasible later. Second, by not committing early to values found by the solver for

incomplete plan skeletons, we can optimize the problem objective throughout the complete plan skeleton when the goal is found.

Like many other heuristic forward search planners, we use common greedy search algorithms that work well for relatively large planning problems. Our heuristic is based on the Temporal Relaxed Planning Graph (Coles et al., 2008) and it provides an estimate of the remaining number of start and end activities to reach the goal.

Our mathematical program is convex. Therefore our consistency checks are complete. However, we use an incomplete greedy search algorithm. As is standard with most greedy HFS planners, we optionally resort to a complete A* search when the incomplete greedy search algorithms fail. In practice, A* is much slower than the greedy search algorithms and we do not report results using A* in this work.

ScottyActivity is not an optimal planner in that we do not guarantee that the returned plan is the best possible plan according to the problem objective. However, since our optimization problem is convex, we guarantee that the instantiation of the plan skeleton selected based on the heuristic is optimal. That is, for the order of starts and ends of activities found by the search, there exist no other control trajectories or activity execution times than the ones returned by ScottyActivity that can produce a better objective value.

In the next sections, we describe the different components of our planner in detail.

6.2 Selecting Successor States

Algorithm 1: Get-Activities

Input: A PDDL-s planning problem (PP) and a search state (S).
Output: A list of helpful applicable activities A_H and list of activities that are applicable but not helpful at S, $A_{A\setminus H}$.

Algorithm

```
1 A_H ← {}, A_{A\H} ← {}
2 for a in PP.A do
3     if DISCRETE-APPLICABLE(S, a) and CONTINUOUS-CONSISTENT(S, a)
4         if IS-HELPFUL(S, a)
5             PUSH(A_H, a)
          else
6             PUSH(A_{A\H}, a)
7 return A_H, A_{A\H}
```

Each search state contains a plan skeleton (ordered list of events), a set of predicates that hold at this state, a set of activities that have started but not ended at this state, and the independently achievable lower and upper bounds for each state variable (as will be explained later).

In order to explore successor states, we need to determine what events (starts or ends of activities) can be applied next. This entails checking that both the discrete and the continuous conditions and effects of the event are consistent with the current state. Since the discrete state is fully described by the search state, the discrete check can be done

Algorithm 2: Test-Consistency

Input: A PDDL-S planning problem (PP) a search state (S) and the partial plan leading to that state (p).

Output: A state with the independently achievable lower and upper bounds for each state variable or *nil* if the state is not consistent.

Algorithm

```
1 prog ← BUILD-PROGRAM(PP, p)
2 if not IS-FEASIBLE(prog)
3     return nil
4 for x in PP.V do
5     x_min ← MINIMIZE(prog, x)
6     x_max ← MINIMIZE(prog, −x)
7     SET-BOUNDS (S, x, x_min, x_max)
8 return S
```

directly (Line 3 of Algorithm 1). However, we cannot do the same with the continuous conditions and effects, since the execution times, durations, state variables, and control variables depend on each other in a complex way. In order to reject infeasible successor states, we construct a mathematical program containing all the constraints defined by the activities in the plan skeleton. Invalid states are those whose associated mathematical programs do not have a feasible solution (Algorithm 2).

We use Second Order Cone Programs (SOCPs) for this check for the following reason. First, SOCPs can effectively represent all the continuous conditions and dynamics that our planner requires (as described in Section 4). Second, SOCPs are a class of convex optimization problems and are commonly solved with complete algorithms that will return no solution only when the program does not have a solution. This is an important characteristic since it ensures that only infeasible states are pruned. Third, SOCPs can be solved very efficiently with interior-point methods available in off-the-shelf solvers. Finally, being convex optimization problems, the solutions of the SOCPs returned by the solvers are guaranteed to be optimal, which is important for reasons that will be explained later. Note that gradient-based algorithms solving general non-linear programs do not present these characteristics. They are orders of magnitude slower (which would directly translate into the planning time since this is, by far, the dominant time). They are not complete (as they can get stuck in local minima) and they are not optimal. Framing all the conditions and dynamics as a convex program is not straightforward and is one of the key innovations of our planner, and is described in detail in Section 7.

Solving the optimization for every candidate successor state that meets the discrete conditions is expensive, and dominates the running time of our planner. For this reason, we compute the feasible lower and upper bounds for each state variable independently with our mathematical program. We do that using the same method that COLIN uses: we solve the optimization problem twice per state variable to minimize and maximize the state variable at the state (Lines 4-7 in Algorithm 2). These bounds constitute an over-approximation of the reachable space at the last event of the plan skeleton. We use the bounds to prune activities

whose state conditions are necessarily not compatible with the state variable bounds, and, therefore, can never be satisfied (`CONTINUOUS-CONSISTENT` method in Algorithm 1). For linear inequality conditions in the form of $\sum k_i x_i \leq c$, this can be done by computing the lower bound of the expression according to the state variable bounds, and checking whether it satisfies the condition. The lower bound on the expression can be computed with the following equation:

$$l_b = \sum \min(k_i x_{imin}, k_i x_{imax}) \tag{14}$$

, where k_i are the constant coefficients of the expression and x_{imin} and x_{imax} are the bounds of state variable x_i. The linear inequality can only be satisfied if $l_b \leq c$. Activities having at least one linear constraint where this condition is not met are pruned from the list of applicable activities at that point in the search. Note that since the optimization is convex (and the returned solution is therefore optimal), the computed bounds of each state variables are guaranteed to be the maximum and minimum values that the state variable could reach. Therefore, this method only prunes infeasible activities. Note, however, that this test using the variable bounds does not guarantee that the linear condition can be satisfied in practice, as each variable lower and upper bound is computed independently. This is not a problem, since the consistency check that uses the optimization model can later reject states with unsatisfiable conditions.

Unfortunately, the previous test is not straightforward in the case of general convex quadratic constraints, as it involves checking the intersection of arbitrary conic shapes (the convex quadratic constraints) and hypercubes (given by the bounds of each state variable). In practice, we use linear over-approximations for those conditions and handle them as in the linear case. This linear over-approximations can be entered manually or computed automatically using the region system described in Appendix B.4. Again, using linear over-approximations only affects the efficiency of our pruning method, but not the completeness of the algorithm.

Finally, the state variable bounds are also needed to define the first layer of the heuristic, as we explain in the next section.

6.3 Relaxed Hybrid Plan Heuristic

The heuristic that ScottyActivity uses is based on the Temporal Relaxed Planning Graph (TRPG). The TRPG assigns to each layer in the planning graph a timestamp corresponding to the earliest time when each layer could be reached (Do & Kambhampati, 2003). In order to handle continuous effects with control variables, such as vehicle dynamics, our heuristic uses two ideas. First, we use *flow tubes* to represent all possible state trajectories resulting from the application of activities with continuous effects. Second, each fact layer is annotated with the minimum and maximum values that each state variable could independently take in that layer. This idea of tracking the bounds for each state variable is borrowed from MetricFF (Hoffmann, 2003).

A flow tube is a compact encoding that describes all possible trajectories resulting from the application of a continuous effect on a state variable, that is, its reachability region. Figure 6 shows a flow tube that represents the reachability region of state variable x when subject to first order dynamics $\dot{x}(t) = v(t)$ from $x_0 = x(t_0)$ for a duration between d_l and

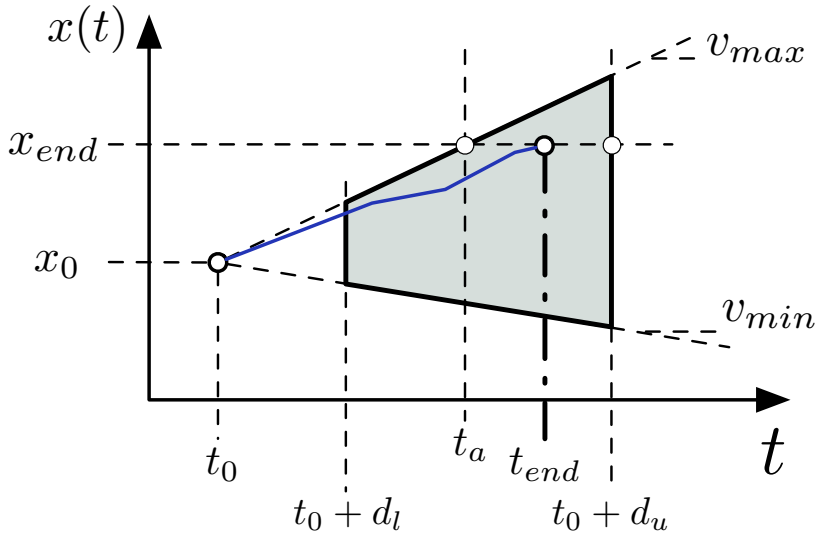


Figure 6: The shaded region is a flow tube that represents the reachable region for state variable x when subject to a linear time-varying effect($\Delta x(t) = v(t) \cdot t$) for a duration between d_l and d_u

d_u. Under first order dynamics, as is the case in ScottyActivity problems, the velocity $v(t)$ is a control variable that is continuously controllable within its actuation bounds of v_{min} and v_{max} for the duration of the activity. The shaded region in Figure 6 is the flow tube and is computed by propagating the initial point with the extremal actuation and temporal constraints. This region represents the values that x can take at the end of the activity. The blue line shows one possible trajectory. Note that the example final value x_{end} can be reached as soon as at t_a if the maximum velocity value v_{max} is chosen, or as late as $t_o + d_u$ if a lower value is used. Flow tubes have been used successfully for temporally and spatially flexible execution of hybrid plans, in applications such as biped walking (Hofmann & Williams, 2006, 2015). Moreover, flow tubes are the basic building block used by Kongming (Li & Williams, 2008) to represent continuous behaviors in its Hybrid Flow Graph. In our heuristic, we use flow tubes to express how the reachable bounds of state variables grow between the consecutive layers of the temporally relaxed planning graph. In our case, the minimum and maximum actuation bounds are obtained by combining the minimum and maximum bounds of all the continuous effects acting on each state variable at a given time.

In practice, our heuristic works in a similar way to COLIN's (Coles et al., 2008), except for some differences that we describe in this section. As is the case for many other planners, the heuristic value is the number of start or end events needed to reach the goal in the relaxed graph. We follow the standard procedure to generate the relaxed planning graph that we summarize next. The initial fact layer is defined by the state at which the heuristic is being computed. The relaxed graph is generated by adding alternating activity and fact layers. All activities that could be applied at a given fact layer are added to the current activity

layer. The procedure finishes when the goal conditions are contained in a fact layer, or when no more activities can be added to the graph. The delete relaxation procedure ensures that at the next fact layer additional activities may become applicable, but activities that were previously applicable always remain applicable in posterior layers. In order to do that, only the discrete *add effects* of an activity applied at a layer are incorporated into the next fact layer, while the discrete *delete effects* are ignored (Hoffmann & Nebel, 2001).

In the spirit of MetricFF, the continuous conditions and effects are handled by tracking the minimum and maximum bounds of each state variable at each fact layer. These bounds are used to test whether an action is applicable at a given layer. The continuous effects active between layers grow these bounds between consecutive layers. We describe now how the bounds for each state variable are grown while creating the relaxed graph, since the presence of control variables in ScottyActivity problems requires a slightly different approach compared to COLIN. We later describe how these bounds are used to test whether a continuous condition can be satisfied at a given layer.

When computing the heuristic for a state, the state variable bounds for the first layer are known in advance. Recall from Section 6.2 that these bounds are computed by solving the convex optimization problem twice per state variable in order to find the minimum and maximum possible values. In order to compute the state variable bounds in the next layers, COLIN keeps track of the maximum positive and lowest negative rates of change (gradients) that each state variable could be subject to at each layer. These rates of change are additive and come from the continuous effects of ongoing activities at the current layer. The minimum and maximum possible bounds for each state variable in layer $i+1$ are computed by extending the bounds at layer i with the maximum positive and minimum negative gradients multiplied by the time elapsed between layers $i+1$ and i, $\Delta t = t_{i+1} - t_i$. In the spirit of delete relaxations, the bounds can only grow from one layer to the next. Contrary to COLIN, the rates of change in ScottyActivity are not fixed, as they depend on the continuous controllable control variables, and these can take any value within their actuation bounds. As explained in the beginning of this section, we use flow tubes to represent how the bounds of the state variables grow in the presence of continuous effects depending on continuous control variables. In practice, we use the minimum and maximum actuation bounds of each control variable to compute the minimum negative and maximum positive rates of change at a given layer. The growth of the state variable bounds from a layer to the next are then computed in the same way as COLIN does.

As an example, consider a layer in which an activity *navigate* with linear time-varying continuous effects $\dot{x}(t) = v_x(t)$ and $\dot{y}(t) = v_y(t)$ is active. Assume that the control variable v_x has actuation limits of $(-1, 2)$ while v_y is constrained to be within its $(1, 3)$ bounds. The minimum negative and maximum positive gradients operating on x at that layer are then $\nabla x^- = -1$, $\nabla x^+ = 2$ respectively. The gradients on y are $\nabla y^- = 0$ and $\nabla y^+ = 3$. Note that $\nabla y^- = 0$ since the bounds of state variables from one layer to the next are only allowed to grow. These gradients define how the boundaries of the flow tubes grow with time (as shown in Figure 6). If an activity *turbo-boost-x* that gave an additional boost to the x velocity of $(-2, 2)$ became active at a later layer, the minimum and maximum possible gradients for x would become $\nabla x^- = -3$ and $\nabla x^+ = 4$ from that layer on, as effects are additive. The lower and upper bounds for state variable x at layer $i+1$, x_{Li+1} and x_{Ui+1} are then computed as:

$$x_{Li+1} = x_{Li} + \nabla x^- \cdot (t_{i+1} - t_i) \tag{15}$$
$$x_{Ui+1} = x_{Ui} + \nabla x^+ \cdot (t_{i+1} - t_i) \tag{16}$$

The bounds computed in the previous manner are used to test whether each continuous condition of an activity can be satisfied at a given layer using the same method employed by MetricFF. We now describe how this test is performed for linear inequalities. Any linear inequality can be expressed as:

$$\sum_j k_j \cdot x_j + c \leq 0, \tag{17}$$

where k_j and c are constants and x_j are the state variables. A linear equality condition can be satisfied at a given layer i if the intersection between the half-space defined by the inequality and the hypercube $R_i = \{\mathbf{z} \in \mathbb{R}^n \mid x_{jLi} \leq z_j \leq x_{jUi}\}$ whose sides are the bounds of each state variable at layer i is non null. For inequalities expressed in the form of Equation (17), this condition is equivalent to asserting that the lower bound of the left hand side of the inequality at layer i, B_i, is smaller or equal than 0. We can compute this lower bound B_i using the bounds of the state variables with the following expression:

$$B_i = \sum_i \min(k_i \cdot x_{jLi}, k_i \cdot x_{jUi}) + c, \tag{18}$$

Activities having linear inequality conditions that cannot be satisfied at a given layer according to the previous test may become satisfiable in a future layer if there are active continuous effects that expand the state variable bounds sufficiently in the right direction. In order to ensure that the relaxed graph is fully expanded, COLIN iteratively computes the next future time when one such unmet condition will become satisfiable and adds a layer at that time until all activities become applicable or all linear conditions that could ever be satisfied become satisfiable. The future point in time, if any, when an unsatisfiable linear inequality will become satisfiable is computed as follows. The linear condition is unsatisfiable because its lower bound B_i is greater than 0. By using the negative and positive gradients of each state variable at that layer, we can compute the negative gradient of the left hand side of the inequality, ∇e_i^- at layer i:

$$\nabla e_i^- = \sum_j \min(k_j \cdot \nabla x_{j_i}^-, k_j \cdot \nabla x_{j_i}^+) \leq 0 \tag{19}$$

If $\nabla e_i^- < 0$, the lower bound of the left hand side of the inequality, B, decreases with time, and the linear inequality will become satisfiable when B becomes negative, which will happen first after Δt_e units of time.

$$\Delta t_e = \frac{B_i}{|\nabla e_i^-|}$$

The next layer can then be created at time $t_i + \Delta t_e$, where t_i is the time of the current layer, with the hopes that satisfying this previously unmet linear condition may make some new activity applicable at that time. If $\nabla e_i^- = 0$ the linear inequality will never become

satisfiable in the future, unless another continuous effect that makes this gradient negative becomes active in a later layer.

Also new in our heuristic is that we need to take into account convex quadratic conditions, that COLIN does not support. As we have just described, computing the future times when linear inequalities will become satisfiable can be done in a straightforward manner using Section 6.3. However, computing the future times when an unmet arbitrary convex quadratic condition will become satisfiable cannot be done with an analytical expression in an efficient manner. Our solution is to use linear over-approximations to the quadratic constraints in the heuristic, which are then handled as explained previously. In other words, our heuristic operates on a relaxed problem, where the relaxation replaces convex quadratic constraints with their linear over-approximation, on top of using the delete relaxation. If the user specifies the convex quadratic conditions as primitives, such as ellipsoidal regions, our planner computes the linear over-approximations automatically. For example, for ellipsoidal region conditions, we compute the axis aligned bounding box that contains the region. Otherwise, the user can specify the approximations directly by providing lists of linear inequalities. These linear over-approximations constitute valid relaxations since they ensure that actions can always be executed earlier than they would be if the actual quadratic constraints were used.

The final difference in our heuristic compared to COLIN's is that *resource-constrained norm effects* (RNE) also need to be considered. These effects only reduce the availability of constrained resources and their application can only make activities infeasible (and never new activities possible). Therefore, in the spirit of delete relaxations, these effects are ignored in the heuristic. The optimization-based consistency check, that accurately computes these effects, rejects states that become infeasible due to this and the search follows through a different route. However, there is room to improve the current heuristic and make it aware of these interactions. We leave handling these effects more accurately in the heuristic for future work.

6.4 Search Strategies

ScottyActivity implements two search algorithms. The first is Enforced Hill-Climbing (EHC) , which has been widely used with success by satisficing planners (Hoffmann & Nebel, 2001). The second is a variation of EHC that we call *obj-EHC* that guides the search to plans that produce better objectives more efficiently. Additionally, ScottyActivity can optionally fall back to A* when these incomplete search algorithms fail. However, this is not discussed further in this work since our A* implementation is the standard one commonly used in other planners.

EHC (Algorithm 3) is a very popular greedy search algorithm that drops the open list every time it finds a state with a lower heuristic value. As a consequence, EHC is not complete. Since the heuristic is the estimated number of start or end activities to reach the goal, our EHC algorithm completely ignores the problem objective to guide the search. The `TEST-GOAL` procedure (Line 15) checks that the goal has been reached and, in that case, solves the optimization with the problem objective in order to generate the solution plan with the activity schedule and control trajectory. This plan is guaranteed to be optimal conditioned on the sequence of events chosen by the search.

Algorithm 3: Scotty-Plan-Ehc

Input: A PDDL-S planning problem PP.
Output: A valid PDDL-s plan or *nil* if no plan could be found.
Algorithm

```
1  S_0 ← MAKE-STATE(PP.I)
2  p_0 ← {}
3  h_best ← GET-HEURISTIC(S_0)
4  Q ← [<S_0, p_0, h_0 >]
5  while not IS-EMPTY(Q) do
6      S, p ← POP(Q)
7      has-valid-descendants ← nil
8      A_H, A_{A\H} ← GET-ACTIVITIES(PP, S)
9      for a in A_H + A_{A\H} do
10         S_new ← APPLY(S, a)
11         p_new ← p + {a}
12         S_new ← TEST-CONSISTENCY(PP, S_new, p_new)
13         if S_new
14             has-valid-descendants ← true
15             p_sol ← TEST-GOAL(S_new, p_new)
16             if p_sol
17                 return p_sol
18             h_new ← GET-HEURISTIC(S_new)
19             if h_new < h_best
20                 Q ← {<S_new, p_new >}
21                 h_best ← h_new
22                 break
               else
23                 if h_new < ∞
24                     PUSH(Q, < S_new, p_new >)
25         if all a ∈ A_H explored
             and has-valid-descendants
               /* Do not explore non-helpful                    */
26             break
27 return nil
```

Algorithm 4: Scotty-Plan-Obj-Ehc

Input: A PDDL-S planning problem PP.
Output: A valid PDDL-s plan or *nil* if no plan could be found.
Algorithm

```
1  S_0 ← MAKE-STATE(PP.I)
2  p_0 ← {}
3  h_best ← GET-HEURISTIC(S_0)
4  Q ← MAKE-PQUEUE()
5  PUSH(Q, < h_best, 0 >, < S_0, p_0 >)
6  while not IS-EMPTY(Q) do
7      S, p ← POP(Q)
8      has-valid-descendants ← nil
9      if S.h < h_best
10         h_best ← S.h
11         CLEAR(Q)
12     A_H, A_{A\H} ← GET-ACTIVITIES(PP, S)
13     for a in A_H + A_{A\H} do
14         S_new ← APPLY(S, a)
15         p_new ← p + {a}
16         S_new ← TEST-CONSISTENCY(PP, S_new, p_new)
17         if S_new
18             has-valid-descendants ← true
19             obj_new ← MINIMIZE-OBJECTIVE(PP, p_new)
20             p_sol ← TEST-GOAL(S_new, p_new)
21             if p_sol  return p_sol
22             h_new ← GET-HEURISTIC(S_new)
23             if h_new < ∞
24                 PUSH(Q, < h_new, obj_new >, < S_new, p_new >)
25         if all a ∈ A_H explored
            and has-valid-descendants
               /* Do not explore non-helpful                    */
26             break
27 return nil
```

While EHC is fast, ignoring the problem objective often leads to EHC making misguided choices that result in highly suboptimal plans. This happens because states that have the same heuristic value (in term of the number of activities to reach the goal) may have very different costs as specified by the problem objective. In order to improve the quality of the plans found by ScottyActivity, we introduce the *obj-EHC* algorithm (Algorithm 4), that is a variation of EHC that attempts to improve this issue. The obj-EHC algorithm uses a priority queue to sort the open states by the heuristic value (first) and, in case of a tie, by the cost incurred in that state so far according to the problem objective (second). Contrary to EHC, obj-EHC computes the heuristic value and the cost-so-far of all the helpful descendants of the current state, as opposed to dropping the queue and descending into a state as soon as a heuristic lower than the prior best one is seen. In the spirit of EHC, obj-EHC drops the queue when a state with a lower heuristic than the incumbent is removed from the priority queue. This provides a reasonable compromise between finding better quality plan and preserving the speed of EHC. However, dropping the queue makes obj-EHC also an incomplete search algorithm.

In order to compute the cost so far for a given state, we solve the same optimization problem that we use to test consistency with the objective of minimizing the cost as specified by the problem objective (Line 19). Note that the cost computed this way only provides an indication of the cost incurred so far, but does not provide any estimate of the future cost that may be incurred in by future activities. That is, this is the current cost of the plan skeleton and not a heuristic. Computing an estimate of the future cost would probably help the search considerably. However, this computation is not straightforward since it involves solving an optimization over possible future choices of activities with their conditions and effects. We leave this extension for future work.

Note that state expansion in obj-EHC is more computationally expensive than in EHC, since obj-EHC requires solving one additional optimization problem (the one minimizing the cost) on top of the optimization problems used to compute the bounds for each state variable. Moreover, obj-EHC maintains a priority queue, which is not needed in EHC. Finally, obj-EHC requires expanding all children of each state as opposed to stopping right away when a state with a better heuristic value is found. In Section 8 we compare both, discuss the advantages and disadvantages of each and show that the quality of the plans returned by obj-EHC is significantly higher in general.

Both search algorithms explore by default only the helpful activities of each state. These are the activities that appear in the first layer of the relaxed planning graph (Hoffmann & Nebel, 2001). However, we cannot model accurately some of the non-linear effects in our heuristic (i.e. *resource-constrained norm effects*, RNEs). As a result, the heuristic sometimes fails to identify activities that need to take place as helpful when RNEs are used. To alleviate this, we allow our search to explore the applicable, but non-helpful, descendant activities if the current state does not have any valid successors (Line 25).

Finally, ScottyActivity can, like other planners, fall back to A* search if EHC or obj-EHC do not find a plan. In practice A* search is very slow in non-trivial problems and we do not show the performance of our planner using this search method.

7. Consistency Checking Through Convex Optimization

Recall that ScottyActivity's convex optimization model is used for multiple purposes during planning. First, the model is used to test the consistency of partial plans. Second, the model is also used to compute the minimum and maximum reachable bounds for each state variable at every search state. Third, to compute the cost of a state according to the problem objective if the obj-EHC algorithm is being used. Fourth, the model is used one last time when the goal is reached in order to compute the optimal activity execution times and the control trajectory that minimizes the problem objective. In all these situations, the model has the same decision variables and constraints. The difference lies in the objective used.

The straightforward mathematical model that describes the characteristics of the continuous effects and other constraints defined in Section 4 is non-linear and non-convex. A key innovation of our work consists in describing an alternative encoding that is convex and can be represented as a Second Order Cone Program (SOCP), a class of convex quadratically constrained programs. Being able to use a SOCP model is important for several reasons. First, there are very efficient solvers for SOCP problems that are orders of magnitude faster than traditional gradient-based methods typically used to solve general non-linear problems. Since every state needs a feasibility check, ScottyActivity solves thousands of optimization problems for typical planning problems, and the runtime of the solver accounts for the majority of ScottyActivity's planning time. Second, unlike general non-linear problems, convex optimization problems can be solved with complete algorithms. As a consequence, ScottyActivity's feasibility check is complete and we are guaranteed to only reject states that are infeasible. Finally, unlike general non-linear problems, convex problems do not have local minima. As a consequence, the solution returned by convex solvers is optimal. This is important for two reasons. First, the minimum and maximum bounds computed for each state variable are guaranteed to be as large as possible, which ensures that no valid states are rejected when using the bounds check described in Section 6.2. Second, the plan returned when the goal is found is guaranteed to produce the lowest possible objective for the sequence of starts and ends of activities that the heuristic guided search finds.

SOCPs are harder to solve than LPs. However, they can still be solved in polynomial-time with interior point algorithms. Moreover, our model only uses cone constraints when the problem has convex quadratic state or control variable constraints, or *resource-constrained norm effects*. When these characteristics are absent, our model becomes a linear program. That is, we only suffer the performance degradation of transitioning from LP to SOCP when the problem requires it.

An important advantage of our approach is that, since the optimization is solved in full at every step of the search, we do not make early commitments to the event times or values of the state or control variables. That is, the optimization can make some choices for event times and state variables at some state in the search, and completely different ones at a later descendant search node. This is important since we prevent early bad choices that could lead to infeasible or suboptimal plans later on and we leave as much flexibility as possible for the solver to make the best choice according to the objective (Toussaint, 2015).

We now present an example that we use throughout the rest of this section to illustrate what the decision variables are and how the SOCP model is built.

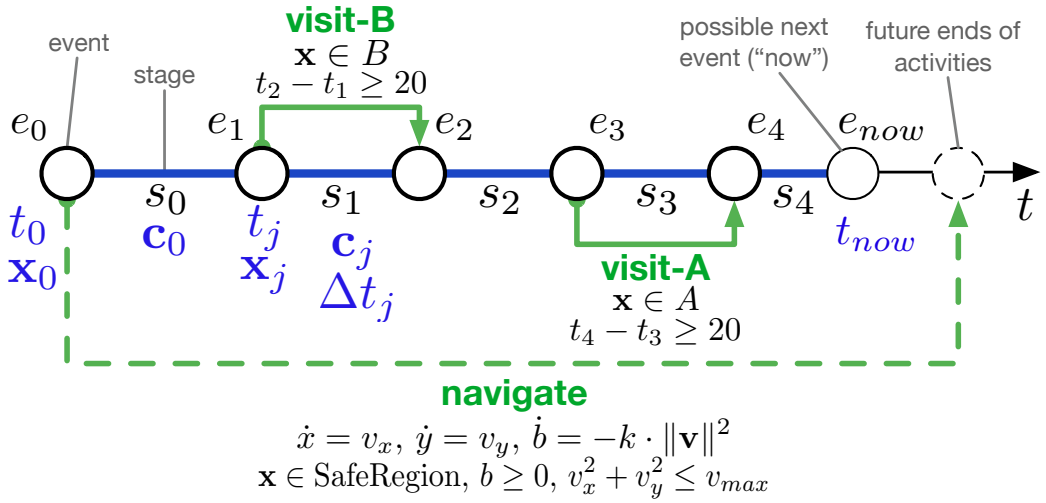


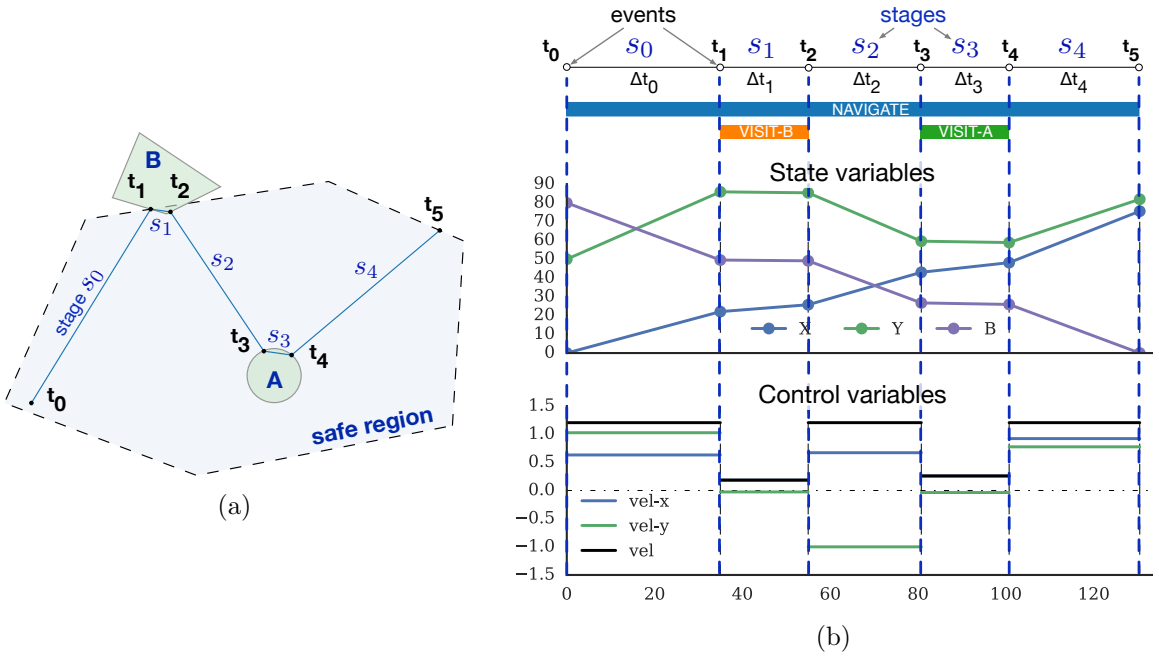
Figure 7: Example plan skeleton when the start of *navigate*, the start and end of *visit-A* and the start and end of *visit-B* events have been added but the end of *navigate* is not part of the plan yet.

7.1 Example

We present an example problem in which a vehicle has to visit regions A and B and stay inside them for at least 20 seconds. While doing so, it must always stay inside the safe region and travel with a maximum velocity of 2 m/s. The vehicle has a limited battery that decreases at a rate proportional to the square of the norm of its velocity. At the end, the vehicle must also reach the maximum possible x and y values.

This example problem is modeled with three activities: *navigate*, *visit-A* and *visit-B*. Assume, for clarity, that the search has already found a plan skeleton with 5 events, in which the start of *navigate* is the first event and the start and end of *visit-B* and *visit-A* happen sequentially while *navigate* takes place. Assume, however, that the end of *navigate* has not been placed in the plan yet. The corresponding plan skeleton is shown in Figure 7. Recall that plan skeletons only define the order of the selected start or end events and that the values of the times, and the control and state trajectory are determined by solving the optimization problem. Figure 7 also shows the constraints imposed by this plan skeleton.

The solution to this example problem once the search is completed and the final *end-navigate* event is placed is presented in Figure 8. Note that our planner automatically chooses the intermediate points and the times of the vehicle state and control trajectory in order to respect the constraints and minimize the objective. In particular note that while the objective includes minimizing the total time, the planner chooses very small velocities during the *visit-B* and *visit-A* activities. The reason for this apparent contradiction is that these activities force the vehicle to remain inside their regions for at least 20 seconds. Since there is no point in leaving these regions early, the planner decides to move very slowly since large speeds decrease the vehicle battery significantly, and this battery is needed to move in the x and y coordinates at the end of the plan. The solution shown is guaranteed to be optimal with respect to the given choice of activities. Note, however, that if the planner had



(a) (b)

Figure 8: Resulting state and control trajectories for the example problem. The optimization chooses the *switch points* in order to satisfy the constraints and obtain the best objective of maximum x and y with minimum time.

chosen to visit A before B the solution returned would be very different and the objective reached would be worse.

We describe throughout the rest of this section how our convex model is built.

7.2 Preliminary Definitions and Decision Variables

Recall that the events in a plan skeleton are totally ordered and that we call *stage* the period of time between consecutive events. The control variables take constant values during stages and, as a consequence, the state variables change linearly in time during these stages. Recall, as well, that given the event execution times and the piecewise constant control trajectory, the state trajectory is fully determined.

A plan skeleton with N events, $e_0, \dots e_{N-1}$, has $N-1$ stages, labeled $s_0, \dots, s_{N-2}$. We call stage j, s_j, the one starting at event j and ending at event $j+1$. Execution times and state variable values are associated to events, while the constant values of the control variable trajectory are associated to stages.

The execution times of the events, $t_j \in \mathbb{R}$, are decision variables in our model. The values of the state variables at each event, $\mathbf{x}_j = \mathbf{x}(t_j) \in \mathbb{R}^n$ are also decision variables in our model. While we need to determine the constant values of the control variables during each stage, $\mathbf{c}_j \in \mathbb{R}^m$, these are not decision variables in our model for reasons that we explain later in this section. Instead, we use other proxy decision variables that we define later and whose value we use to compute the values of the control variables once the optimization problem is solved. For the sake of clarity in our model, we also define the duration of each

stage, Δt_j, as a decision variable whose value is constrained to be $\Delta t_j = t_{j+1} - t_j$. Other auxiliary decision variables are defined for different purposes, and are introduced as needed in this section.

The execution time and the values of the state variables at the first event are constrained to take the initial conditions defined in the PDDL-S problem. In our example, $t_0 = 0, x_0 = 0, y_0 = 50, \text{battery}_0 = 80$.

7.3 Temporal Constraints

Events in the plan skeleton are totally ordered. We enforce this order by constraining the events to be separated by at least ε time ($\Delta t_j \geq \varepsilon$). This is consistent with the PDDL2.1 semantics (Fox & Long, 2003). We do this because solvers can only enforce non-strict inequality constraints. For each event representing the end of an activity, the constraint that the minimum and maximum duration of that activity is respected is added to the program:

$$d_l(a) \leq t_e(a) - t_s(a) \leq d_u(a), \quad \forall a \in P \tag{20}$$

, where $t_s(a)$ and $t_e(a)$ are the event times of the start and end of activity a, $d_l(a)$ and $d_u(a)$ are the lower and upper bounds of the duration and each a is an activity in the plan skeleton.

In the example, the vehicle has to remain in regions A and B for at least 20 seconds. Therefore, the constraints $t_2 - t_1 \geq 20$ and $t_4 - t_3 \geq 20$ are added to the program.

7.4 State Constraints

Each event is the start or end of an activity. For each event, e_j, which is the start (end) of an activity, we add the constraints that enforce that the value of the state variables, $\mathbf{x}_j$, satisfy the start (end) state conditions of the activity.

Moreover, the values of the state variables at e_j, $\mathbf{x}_j$, need to satisfy all the maintain (*over all*) state conditions of all the activities that have started before event e_j but have not ended yet. Recall from Section 5 that, since state conditions are restricted to be convex and the state trajectories are piecewise linear, enforcing the maintain conditions of an activity at all events that take place during the activity execution ensures that the conditions are always satisfied at any continuous point in time during the activity.

In the example, the *visit-B* activity requires that the vehicle remains in region B while the activity takes place. Since B is a polygonal region, this is achieved with the constraint $H(x_j\, y_j) \leq \mathbf{h}$, $j \in \{1, 2\}$, where $H \in \mathbb{R}^{2\times 2}$ and $\mathbf{h} \in \mathbb{R}^{2\times 1}$ are the constants of the linear inequalities that represent the polygon. The conditions for the *visit-A* activity are similar, except that the region is a circle and the constraints take the quadratic form of $(x_j - c_x)^2 + (y_j - c_y)^2 \leq R^2$, $j \in \{3, 4\}$. Moreover, since events e_1, e_2, e_3 and e_4 take place while the *navigate* activity is ongoing, the state variables at those events need to satisfy its *over all* conditions: remain inside the safe region and keep the battery level above 0.

7.5 State Change

State variables change their value between consecutive events e_j and e_{j+1} due to the presence of continuous effects active during the stage between those events, s_j. Therefore, the value of state variable k at event $j+1$ is given by:

$$x_k(t_{j+1}) = x_k(t_j) + \sum_{eff_i \in E_{x_k}(j)} \Delta x_{keff_i}(j) \tag{21}$$

, where $E_{x_k}(j)$ is the set of all active continuous effects operating on state variable x_k at stage j. Each $\Delta x_{keff_i}(j)$ is a decision variable describing the change due to a continuous effect during stage j. Each type of continuous effect imposes different constraints on this value, as we describe in the next section.

7.6 Control Variables and Continuous Effects

The change produced by a *controllable linear time-varying continuous effect (CLTE)* on state variable x_k during stage j is described by:

$$\Delta x_{kCLTE}(j) = \left(\sum_i k_i \cdot c_i(j) \right) \cdot (t_{j+1} - t_j) \tag{22}$$

, where $c_i(j)$ is the constant value that the i-th control variable takes during stage j. Given that the event times are decision variables, the previous equation is non-linear and non-convex if the control variables are also decision variables. To overcome this problem, the first version of ScottyActivity in (Fernandez-Gonzalez et al., 2015) relaxes this non-linear constraint to a linear interval equation on the state variables. With this relaxation, control variables are not needed as decision variables and, instead, its lower and upper bounds are used in the interval equations. Using this relaxation, however, limits the problems that can be solved. For instance, with this relaxation the same control variable cannot be used in more than one effect simultaneously, since the interval equations would, in practice, allow the solver to choose two different values for the same control variable (e.g. a high velocity to make a vehicle drive fast and simultaneously a low value to make it consume less battery). Another important limitation of this relaxation is that control variables are completely independent of each other and they can only be limited by fixed bounds. Other constraints on the control variables are not possible within this framework. For instance, a moving vehicle would only have independent bounded velocity control variables on x and y. The magnitude of its velocity cannot be limited, as this requires a norm constraint acting on both the v_x and v_y velocities. Therefore, the vehicle would essentially be able to move much faster in diagonal directions than in the x and y directions.

These limitations are addressed here with a different encoding. Instead of using interval equations, we define the decision variable $c_{i\Delta t_j}$ for every control variable being used in each stage j. This decision variable represents the value $c_{i\Delta t_j} = c_i \cdot \Delta t_j$. However this constraint is not added directly, to avoid adding a non-linearity and non-convexity constraint. Instead, the new decision variable $c_{i\Delta t_j}$ is subject to the linear inequality constraints

$$c_{il} \cdot \Delta t_j \leq c_{i\Delta t_j} \leq c_{iu} \cdot \Delta t_j \tag{23}$$

where c_{il} and c_{iu} are the constant lower and upper bounds of control variable c_i. We can use these new decision variables to turn (22) into a linear equation, as we describe later.

Note that, in effect, instead of asking the solver to pick a value for each control variable, we ask the solver to pick the times of the events and the product of the values of the control variables and the elapsed time between consecutive events. The key idea that makes this encoding work is that, for most purposes, we only need the values of the products of control variables and time intervals instead of the actual values of the control variables. A key advantage of this formulation is that, contrary to the first version of ScottyActivity, control variables can now be used in as many continuous effects as needed, since the $c_{i\Delta t}$ decision variables can appear in other constraints, and the values will be consistent with each other. Moreover, we can now impose constraints on the control variables. For example, we can impose that two control variables, c_1, c_2 always satisfy $c_1 + c_2 \leq 5$. The constraint cannot be encoded directly since c_1 and c_2 are not explicit decision variables. However, we can multiply the equation by Δt_j and impose the condition $c_{1\Delta t_j} + c_{2\Delta t_j} \leq 5 \cdot \Delta t_j$ in every stage in which they are active.

Similarly, we can also impose maximum l_2-norm constraints on sets of control variables. In many robotic applications, like the motivating example in Section 3.1, it is useful to represent the velocity of a robot with its principal components (v_x, v_y) but still limit the total velocity by some fixed amount. This can be expressed with the constraint $||\mathbf{c}|| \leq v_{max}$, where $\mathbf{c} = [v_x, v_y]^T$ is a vector of control variables. To encode this constraint, we multiply the whole equation again by Δt_j to obtain the equation

$$\|\mathbf{c}\| \cdot \Delta t_j = \left\|\mathbf{c_{\Delta t_j}}\right\| \leq v_{max} \cdot \Delta t_j \tag{24}$$

The previous inequality is not a linear constraint, but a particular case of a second order cone constraint. Second order cone constraints are given by:

$$\|\mathbf{Ax} + \mathbf{b}\| \leq \mathbf{c}^T\mathbf{x} + \mathbf{d} \tag{25}$$

where $\mathbf{x}$ is the vector of decision variables and the rest are constant parameters. Since $\mathbf{c_{\Delta t}} = [v_{x\Delta t}, v_{y_{\Delta t}}]^T$ and Δt are decision variables, we can transform (24) into (25) by taking $\mathbf{b} = \mathbf{d} = \mathbf{0}$ and choosing constants $\mathbf{A}$ and $\mathbf{c}$ as needed.

In the example, the velocities of the vehicle, v_x and v_y are each restricted to be within $(-2, 2)$. Therefore, we define the variables $v_{x\Delta t_j}$ and $v_{y_{\Delta t_j}}$ for each stage and constraint them as:

$$-2 \cdot \Delta t_j \leq v_{x\Delta t_j} \leq 2 \cdot \Delta t_j \tag{26}$$

$$-2 \cdot \Delta t_j \leq v_{y_{\Delta t_j}} \leq 2 \cdot \Delta t_j \tag{27}$$

$$\tag{28}$$

Moreover, the magnitude of the velocity of the vehicle is also constrained to not exceed 2. This is achieved with the following cone constraint for each stage:

$$v_{x\Delta t_j}{}^2 + v_{y_{\Delta t_j}}{}^2 \leq 2^2 \cdot \Delta t_j{}^2 \tag{29}$$

7.6.1 CLTE Effects

Using the newly defined $c_{i\Delta t_j}$ variables, the originally non-linear CLTE equation (22) can then be rewritten as the linear equation

$$\Delta x_{kCLTE}(j) = \sum_i k_i \cdot c_{i\Delta t_j} \tag{30}$$

As an example, consider that the change in state variable x during stage s_1 (during the *visit-B* activity) in the example scenario is given by:

$$x_2 = x_1 + v_{x\Delta t_1} \tag{31}$$

7.6.2 RNE Effects

The second type of continuous effects, *resource constrained norm effects (RNE)*, are described with cone constraints. In this work we focus on *linear norm effects* (LNE) and *linear squared norm effects* (LSNE). Recall that the change due to a LNE effect on a constrained resource (state variable r_k) is given by

$$\Delta r_{kLNE}(j) = -k_{LNE} \cdot \|\mathbf{c}_e\| \cdot \Delta t_j, \quad k_{LNE} \geq 0 \tag{32}$$

where k_{LNE} is a non-negative real constant and $\mathbf{c}_e$ is the vector of the control variables involved in the effect. Equation (32) is not convex and cannot be encoded directly. However, we can transform (32) into a cone constraint by defining a new decision variable b that acts as a bound. The equation is then rewritten as

$$\|\mathbf{c_{\Delta t_j}}\| \leq b, \quad b \geq 0 \tag{33}$$

$$\Delta r_{kLNE}(j) = -k_{LNE} \cdot b \tag{34}$$

Equations (32) and (34) do not represent the same, since b is simply an upper-bound on $||\mathbf{c_{\Delta t_j}}||$. This is the reason why we have to restrict these effects to constrained resources. In general, the bound b will not be tight and the computed value for resource r_k may not be accurate. However, these equations can accurately model that a constrained resource decreases with the norm of a control variable vector and the bound will become tight to ensure that a resource never dips below some threshold. This is useful to model, for example, how a vehicle's battery decreases as a function of the speed it is traveling at, regardless of the x, y direction. Unfortunately, we cannot use resource constrained norm effects to impose that a certain resource is below some level, since the artificial bound b could take any arbitrary large value to satisfy the constraint trivially without changing the actual value of the norm of the control variable vector. Modeling such condition would require handling a constraint in the form of $||\mathbf{c_{\Delta t_j}}|| \geq b$. However, this is a non-convex constraint that we do not support since it cannot be represented with a SOCP program. In practice, we have not found many problems in which this is required, but we leave this extension for future work.

The second resource constrained norm effect that we support is the *linear squared norm effect* (LSNE), which is subject to the same limitations as the LNE effect, but in which the decrease rate of the resource variable is proportional to the squared norm:

$$\Delta r_{kLSNE}(j) = -k_{LSNE} \cdot \|\mathbf{c}\|^2 \cdot \Delta t_j, \quad k_{LSNE} \geq 0 \tag{35}$$

Again, equation (35) is non-convex, but we can use the same principle as before to represent it as a SOCP constraint. Since $\mathbf{c_{\Delta t_j}} = \mathbf{c} \cdot \Delta t_j$, we can write

$$\|\mathbf{c}\|^2 \cdot \Delta t_j = \frac{\mathbf{c_{\Delta t_j}}^T \mathbf{c_{\Delta t_j}}}{\Delta t_j} \leq b, \quad b \geq 0 \tag{36}$$

where b is, again, an auxiliary positive bound variable. Finally, equation (36) can be rewritten as

$$\mathbf{c_{\Delta t_j}}^T \mathbf{c_{\Delta t_j}} \leq b \cdot \Delta t_j \tag{37}$$

$$\Delta r_{kLSNE}(j) = -k_{LSNE} \cdot b \tag{38}$$

which is a *rotated cone constraint*, a valid type of convex SOCP constraint, since b and Δt_j are positive.

In the example, the vehicle battery decreases with a rate proportional to the square of the vehicle velocity. This is modeled with the *LSNE* effect present in the *navigate* activity. As explained in the previous section, we use auxiliary decision variables to model this change. For each stage, we define the upper bound decision variables $u_{LSNE}(j)$ for each stage that are subject to the rotated cone constraint described in eq. (36). In particular, the constraint takes the form:

$$\mathbf{v_{\Delta t_j}}^T \mathbf{v_{\Delta t_j}} \leq u_{LSNE}(j) \cdot \Delta t_j, \quad u_{LSNE}(j) \geq 0 \tag{39}$$

where $\mathbf{v_{\Delta t_j}} = (v_{x\Delta t_j}\, v_{y_{\Delta t_j}})$. Then, the battery is updated in each stage according to $b_{j+1} = b_j - k_{LSNE} \cdot u_{LSNE}(j)$. Although the battery value computed with this model may not be accurate (since the upper-bound decision variables could take arbitrarily large values), these constraints ensure that the battery is never depleted.

7.7 Partial Skeleton Plans

The constraints presented so far are sufficient to represent full plans. However, ScottyActivity also needs to test the consistency of partial skeleton plans. In these partial plans there may be ongoing activities that have started but not ended yet. We use the same t_{now} trick that COLIN (Coles et al., 2012) uses to detect inconsistencies due to future temporal constraints being violated. An event at t_{now} is placed after all the other events in the partial plan. The state variables at this event are updated according to the active continuous effects and subject to the invariant conditions of activities that have started but not ended yet. For each of these ongoing activities, a decision variable describing the future time where it will end is added and constrained to occur after t_{now} and to respect the activity duration constraints. This helps identify partial plans that are not feasible because future temporal deadlines cannot be met. The 'now' event is also the point at which each state variable is minimized and maximized to find the bounds of each state variable at the end of the given plan skeleton.

In the example problem activity *navigate* has not ended yet (Figure 7). Therefore, we place the auxiliary 'now' event, e_{now}, after the last event in the skeleton (the end of *visit-A*) but before the future end of the *navigate* activity.

7.8 Objective

When the model is used to test the feasibility of a plan skeleton, no objective is needed. When the model is used to compute the bounds of each state variable, the problem is solved $2n$ times to minimize and maximize each of the n state variables at the end of the plan. The $2n$ objectives are, therefore: $\{x_k(t_{now}), -x_k(t_{now})\}$ [1].

The problem objective needs to be used in the model in two situations: when determining the accumulated cost of a plan skeleton if the obj-EHC algorithm is being used, and when finding the optimal execution times and control trajectory for the full plan skeleton that satisfies the goal discrete conditions. Recall that the problem objective is a linear combination of the total plan time, the values of the state variables at the end and the sumproducts of the norms (or squared norms) of vectors of control variables and the durations they are active for. The first two are straightforward to model, since the total plan time and the values of the state variables are, respectively, the execution time of the last event in the plan and the values of the state variables at that time. The last option allows us to minimize actuation control.

The sumproduct of the norm of a control variable vector $\mathbf{c}_v$ and the times it is active for is given by

$$\sum_j \|\mathbf{c}_v\| \cdot \Delta t_j \tag{40}$$

Note that if $\mathbf{c}_v$ represents the velocity of a vehicle, the previous is equivalent to the distance traveled by that vehicle. In order to minimize eq. (40) using our convex model, we use the same approach we followed to represent *LNE* effects (eq. (33)). For each stage, we define a bound decision variable b_j that satisfies

$$||\mathbf{c_{\Delta t_j}}|| \leq b_j, \quad b_j \geq 0 \tag{41}$$

Then, minimizing $\sum_j \|\mathbf{c}_v\| \cdot \Delta t_j$ is equivalent to minimizing the sum of the bound variables for each stage ($\sum_j b_j$). We use an analogous approach to minimize the sumproduct of the squared norm. In this case the bound variables are constrained with rotated second order cone constraints (as in the case of the *LSNE* effects):

$$\mathbf{c_{\Delta t_j}}^T \mathbf{c_{\Delta t_j}} \leq b_j \cdot \Delta t_j, \quad b_j \geq 0 \tag{42}$$

In the example, the problem objective is a combination of minimizing the total time and maximizing the sum of the x and y coordinates at the end. For the full plan as shown in Figure 8a, this is achieved with the minimization objective $0.7t_5 - x_5 - y_5$.

8. Experimental Results

In this section we evaluate the scalability of our planner in both synthetic domains and real expressive robotic scenarios. First, we use synthetic domains to illustrate why maintaining both continuous time and continuous control variables, as ScottyActivity does, is essential

1. For valid states, the bounds for the state variables always need to be computed after the feasibility check. Therefore, for efficiency purposes, our model is never used without an objective in practice. The first model solved is simultaneously determining the feasibility of the plan and minimizing the first state variable.

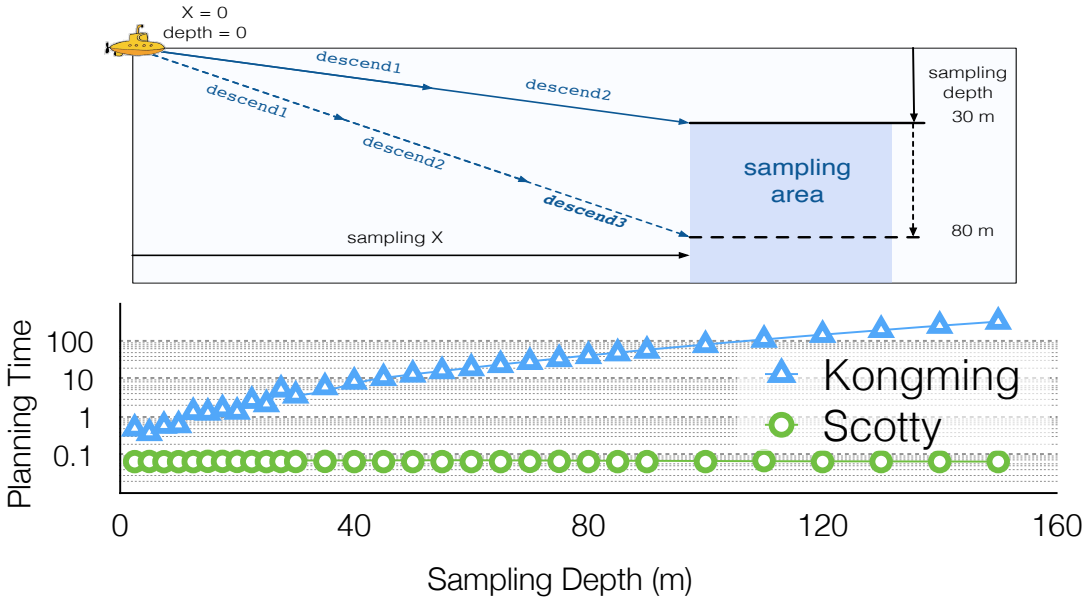


Figure 9: Example scenario that shows the problems of discretizing time. Planning time is shown in seconds.

to plan efficiently over long horizons. We then present in Section 8.2 three robotic domains that ScottyActivity is designed to solve efficiently and show the scalability of our approach in these problems. Finally, we present in Section 8.3 a comparison of ScottyActivity against a mixed-integer solution (MIP). This MIP approach uses the same model from Section 7 to represent state and control variables, times and constraints. Instead of selecting activities using heuristic forward search, the MIP method makes this decision with integer variables. We show that our planner performs at least two orders of magnitude faster than the MIP method in medium to large problems. In our tests, we used an Intel Core i7-3770 3.40 GHz processor, and Gurobi 7 as ScottyActivity's internal convex optimization solver.

8.1 Synthetic Benchmarks

In our first example, we compare the performance of ScottyActivity against Kongming in a robotic sampling scenario that shows why Kongming's time discretization can be a problem in common applications and how ScottyActivity overcomes this issue. We then compare ScottyActivity's performance against POPF (Coles et al., 2010), a state of the art planner capable of planning with continuous effects. Recall that POPF does not support control variables and therefore needs to represent different rates of change with multiple discretized actions. This example shows that representing control variables continuously is essential in robotic applications and that the alternative discretization of control variables is not a scalable approach.

	2D AUV 1	2D AUV 2	3D AUV	Firefighting 1	Firefighting 2
Kongming	3.633	9.736	13.063	1.505	20.202
ScottyActivity	0.054	0.025	0.192	0.03	0.372

Table 1: Comparison between Kongming and ScottyActivity in several domains. Results show planning time in seconds.

8.1.1 Discretization of Time

Kongming is the first mixed discrete-continuous activity planner that performs simultaneous activity and motion planning as a tightly coupled problem. It does so by allowing continuous control variables and by using an efficient representation of all possible trajectories with flow tubes. However, while Kongming's approach is very innovative, we argued in Section 2 that its fixed time discretization hinders its performance in problems that require long planning horizons and where both short and long lived activities coexist. In this section we present a simple robotic scenario that highlights this issue and we show that ScottyActivity's continuous time formulation avoids this problem.

The problem consists of a simple autonomous underwater vehicle (AUV) sampling mission as shown in Figure 9. In this problem the AUV needs to reach a certain depth range in order to take a sample. We parameterize this scenario in terms of such sampling depth. The AUV can use the action descend to modify its depth according to the bounded control variable *descent-rate*. Because Kongming discretizes time in constant time steps, increasing the target sampling depth forces Kongming to create additional fact and action layers and, additionally, more variables that the MILP solver needs to consider. As a result, the performance of Kongming degrades very quickly with the target sampling depth as shown in Figure 9. On the other hand, ScottyActivity's performance is constant (and orders of magnitude better than Kongming's). This is an expected result. Because ScottyActivity does not discretize time, it solves essentially the same problem regardless of the target sampling depth. Only one *descend* activity is needed and it is only the parameters of the mathematical optimization that change in each instance of the problem. For the sake of completeness we also benchmarked ScottyActivity in other domains in which Kongming was used. These domains, described in detail in (Li, 2010), typically showcase one or more mobile robots moving in a 2D or 3D environment and completing objectives that involve visiting different locations. Table 1 shows that ScottyActivity exhibits a large performance advantage over Kongming in these domains as well.

8.1.2 Discretization of Control Variables

In this section we argue that continuous control variables are essential to efficiently solve robotic planning problems. In order to do this, we compare the performance of ScottyActivity and POPF in a synthetic robotic domain. While POPF is a very efficient planner, it only supports linear continuous effects with previously defined, fixed rates of change, as opposed to the continuously controllable rates of change (control variables) that ScottyActivity supports. In this synthetic domain a mobile vehicle needs to visit 6 regions. For the sake of simplicity the order of the regions is fixed in this problem. The objective of this problem consists in minimizing the plan makespan. In order to move, the vehicle uses the

	Planner	A	t	S	L	O	l	v_{avg}
	ScottyActivity (EHC)	1	0.81	24	24	151.11	203.65	1.35
POPF	4 directions	4	1.07	326	38	200.22	376.40	1.88
	8 directions	8	1.77	531	36	192.22	415.90	2.16
	5 steps	24	11.84	3530	36	277.17	413.92	1.49
	7 steps	48	22.66	6168	32	342.56	413.01	1.21
	9 steps	80	59.62	15037	32	469.92	459.06	0.98
	11 steps	120	133.35	31160	32	430.13	446.35	1.04

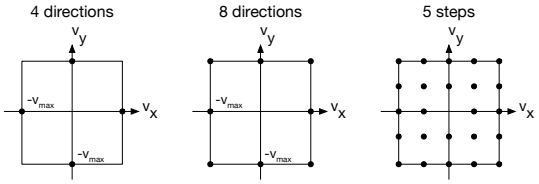
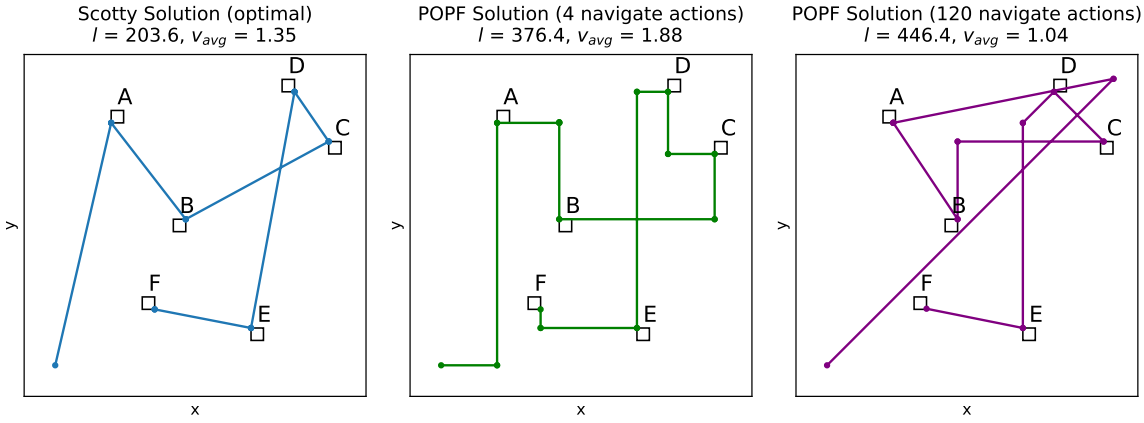


Table 2: Discretization example results. **A**: number of navigate actions in the domain; **t**: Planning time in seconds; **S**: Number of nodes expanded; **L**: Plan length in number of actions; **O**: Makespan (objective value) of the plan returned; ***l***: Length of the path traveled; v_{avg}: Average speed of the vehicle throughout the plan. The diagrams on the right show the discretization performed in some of the problem instances. Each black dot represents a *navigate* activity with its given v_x and v_y. For ScottyActivity, any velocity value within the square is allowed.



(a) (b) (c)

Figure 10: Trajectories of the discretization example. In this domain, the solution returned by ScottyActivity is optimal (a). With 4 navigate actions, the solution is worse than optimal and harder to find (b). Adding more actions, the problem becomes much harder to solve, and the solution returned gets worse (c)

navigate activity. This activity has two CLTE effects, each operating in state variables x and y respectively:

```
(increase (x) (* (vx) #t))
(increase (y) (* (vy) #t))
```

For ScottyActivity, v_x and v_y are continuous control variables that can take any value between -2 and 2. Since POPF only supports fixed rates of change, the values of v_x and v_y have to be fixed in advance. Therefore, we create multiple *navigate* activities with different discretization values for the velocities. We compare the performance of ScottyActivity using EHC to POPF in problem instances using different discretization values (from 4 navigate activities to up to 120 navigate activities).

The results are shown in Table 2. The diagrams on the right show the discretizations performed for some of the problem instances. Since the visit order of the regions has been fixed, the plan returned by ScottyActivity is optimal. Given that ScottyActivity only needs one *navigate* activity to represent all possible velocities, this problem can be solved very quickly. As expected, the results show that as the number of *navigate* activities increases, it becomes harder for POPF to find a solution since it needs to explore more states.

Figure 10 shows the optimal trajectory returned by ScottyActivity and the trajectories returned by POPF for different discretizations. A finer discretization (more *navigate* activities), provides the chance of finding better plans, since more fine grained control of the velocities is possible, which in turn allows better navigation headings to be selected. However, the results show that the plans returned by POPF are not only much harder to find, but they also do not get better with more *navigate* activities. We hypothesize that this is due to the greedy nature of the Enforced Hill Climbing algorithm that POPF uses, and the fact that the number of actions to the goal and not the objective is what guides the search. As an example consider the two first actions of the plan shown in Figure 10c, which are the navigate activities with velocities $(0.8, 0.8)$ and $(-2, -0.4)$ respectively. Because EHC commits to those first two actions with their fixed velocities and headings, POPF's linear program has no other option than taking the vehicle all the way to the top right corner in order to be able to reach region A afterwards.

This discretization example shows that even in simple domains, continuous control variables provide a large advantage over discretized rates of change.

8.2 Evaluation in Robotic Domains

To the best of our knowledge, there are currently no prior benchmarks available that can exploit the features of our planner. Therefore, in order to showcase the new capabilities of our planner and to show that our optimization framework is fast and scalable, we present three new expressive robotic domains and benchmark our planner against them. Since no other planner can solve these domains, we also provide simplified, linear version of some of these domains that we use to compare our planner to POPCORN (Savas et al., 2016). We compare against this planner since it supports control parameters, which are essential for these domains. The simplified domains do not capture the full expressivity of these problems. However, because POPCORN supports control parameters, we can still represent some aspects of these problems within its formalism. We also compare against

the first version of our planner, Scotty1 (Fernandez-Gonzalez et al., 2015), since it uses a much simpler optimization model than the one presented in this work.

We describe these domains in this section. The PDDL description of these that ScottyActivity takes as input is provided in Appendix D.

8.2.1 The AUV Domain

In this domain an Autonomous Underwater Vehicle (AUV) needs to visit and take samples at multiple regions. This domain is similar to POPCORN's 2D-AUV-Power domain, that is based on prior Kongming and Scotty domains. There are two main differences between POPCORN's domain and ours. First, since POPCORN does not support controllable rates of change, the effects of the *glide* action are modeled as discrete numeric displacements on the x, y variables at the end of the action, whereas we model the motion as a continuous effect that takes place while the action is being executed. Moreover, since POPCORN only supports linear constraints, its authors model the maximum power of the vehicle as a simple linear constraint on the displacements at the end of the action (e.g. $3d_x + 4d_y \leq 60$), while we can use the new features of our SOCP model to limit the magnitude of the velocity ($v_x^2 + v_y^2 \leq v_{max}^2$). The objective of this domain consists in minimizing the plan makespan. The simplified linear version of this domain is similar except that we place no constraints on the v_x, v_y velocities other than their simple independent bounds.

8.2.2 The ROV Domain

This domain is based on the motivating example presented in Section 3.1 but without an AUV. As in the motivating example, the Remotely Operated Vehicle (ROV) needs to take samples in multiple regions and end, together with the ship, in the destination region. Note that our planner decides where to station the ship while the ROV is taking samples, and that good selections of that position may allow the ROV to visit several regions without having to be recovered by the ship first. The objective for this domain minimizes a linear combination of the plan makespan and the distance traveled by the ship. Figure 11 shows an example solution plan returned by our planner for problem 6 of this domain.

In the simplified linear version of this domain we remove the velocity norm constraints. Furthermore, the maximum distance constraints, which are modeled with the convex quadratic constraints of being inside a circle, are replaced by a simpler linear polygonal over approximation of such a circle (an octagon in this case). Since we cannot model the distance traveled by the ship without quadratic constraints, the simplified version only optimizes the makespan of the plan.

8.2.3 The Air Refueling Domain

In this domain an autonomous Unmanned Aerial Vehicle (UAV) needs to take pictures of several regions before landing at the destination location. Since the UAV has limited fuel, it needs to refuel in-air from a tanker plane. While refueling, both planes can keep moving but they need to stay within a maximum distance. The UAV fuel decreases as a function of the distance traveled and the square of the velocity ($\dot{f} = -k_1 v - k_2 v^2$). As in the ROV domain, the objective for this domain is to minimize a linear combination of the plan makespan and the distance traveled by the tanker plane. In instances 11-20 there is an additional UAV,

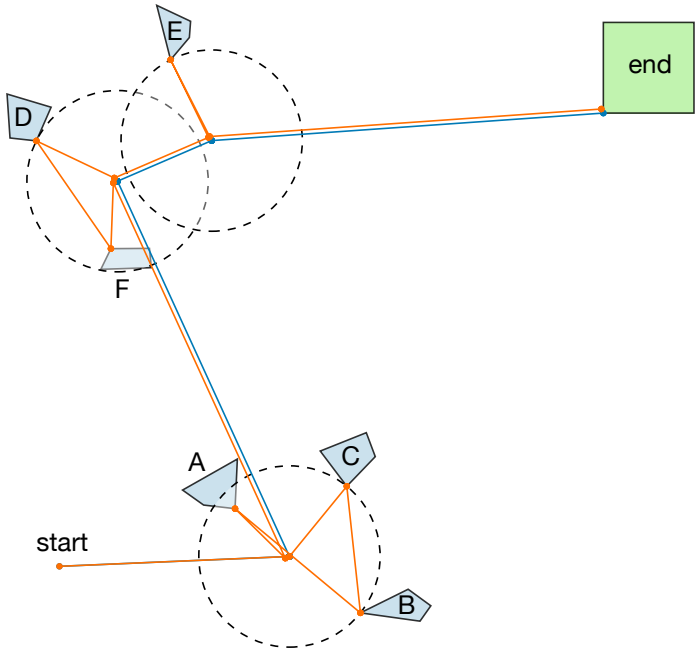


Figure 11: Trajectories of ship (blue) and ROV (orange) in problem 06 of the ROV-regions domain. Note how the planner selects ship positions so that the ROV can take samples at multiple regions without having to reposition the ship and while observing the tether range constraint.

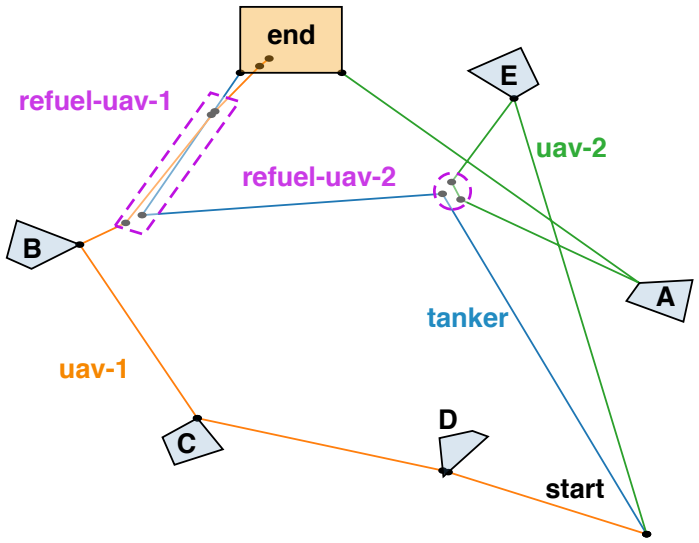


Figure 12: Example solution for instance 15 of the refueling domain with a tanker plane (blue) and two UAVs (orange and green). Note that the planner finds a trajectory for the tanker that allows it to refuel both UAVs as needed.

but only one UAV can refuel from the tanker plane at a time. Figure 12 shows an example solution plan returned by our planner for instance 15 of this domain.

This domain is challenging for several reasons. First, the planner needs to consider the simultaneous trajectories of multiple vehicles and also their fuel levels. Second, and more importantly, while our optimization model supports the *resource-constrained norm effects* (such as the fuel decrease depending on the norm or squared norm of the velocity), the heuristic does not consider these effects directly. Therefore, our planner only chooses the refuel activities when reaching other regions becomes infeasible due to having insufficient fuel.

We do not present a simplified version of this domain because POPCORN would not be able to solve a linear alternative. The reason is that the *refuel* activity requires continuous effects since both the tanker and the UAV have to be flying simultaneously while staying close to each other. POPCORN cannot model this since the numeric change can only be applied at the beginning or end of an activity and not continuously in time. This domain also requires that the fuel of each UAVs decreases as a function of the magnitude of their velocities, which POPCORN does not support either.

8.2.4 Results

We evaluated ScottyActivity with the two search approaches discussed in Section 6.4 in these robotic domains. The results are shown in Table 3. As seen in column T, our convex optimization model, a key contribution of our work, is solved very quickly. The mean optimization time per problem grows for more complicated instances since these have more state variables and require more activities, which results in far more decision variables and constraints at later stages of the search. However, most optimization problems are solved in less than 10 ms on average for small to medium domain instances and in less

	AUV											ROV											Air Refueling										
	EHC					obj-EHC						EHC					obj-EHC						EHC					obj-EHC					
	t	L	S	N	T	t	L	S	N	T	$O_{\%}$	t	L	S	N	T	t	L	S	N	T	$O_{\%}$	t	L	S	N	T	t	L	S	N	T	$O_{\%}$
01	0.53	4	4	17	2	0.54	4	4	21	2	0.0	1.16	16	19	153	3	1.20	16	19	172	3	0.0	0.97	8	11	111	3	1.08	8	12	133	3	-0.0
02	0.55	8	10	41	2	0.64	8	9	46	1	0.0	1.74	20	35	281	4	1.52	20	25	226	4	-0.0	1.68	12	19	191	5	1.93	12	20	221	6	-0.0
03	0.66	12	18	73	2	0.80	12	15	76	2	0.0	2.79	24	57	457	4	1.90	24	32	289	4	-0.1	2.44	16	30	301	6	2.72	16	28	309	6	-0.0
04	0.84	16	28	113	2	0.90	16	22	111	2	-5.7	4.34	36	79	633	5	3.51	36	48	433	6	-0.2	5.78	18	74	669	7	3.52	18	36	397	7	26.1
05	0.94	20	40	161	2	0.98	20	30	151	2	-4.0	7.26	40	119	953	6	4.32	40	55	496	7	-14.7	3.82	20	45	433	7	4.74	20	43	464	8	-8.5
06	0.89	20	40	161	2	0.97	20	30	151	2	-13.3	10.71	52	157	1225	7	7.24	52	74	651	9	-21.2	5.68	24	60	583	8	5.57	22	50	541	8	-12.1
07	1.18	24	54	217	2	1.11	24	39	196	2	-9.5	16.38	56	213	1665	9	9.05	56	83	734	11	-15.5	11.18	28	98	927	11	5.69	-	55	585	8	-
08	1.24	24	54	217	2	1.12	24	39	196	2	-31.4	19.75	68	236	1822	9	14.84	68	108	950	14	-12.1	10.93	32	96	916	11	11.13	30	77	808	12	-30.1
09	1.46	28	70	281	2	1.34	28	49	246	2	-19.1	32.24	72	338	2607	11	17.48	72	119	1053	15	-21.0	12.03	32	105	997	11	13.29	32	94	945	12	-77.0
10	1.49	28	70	281	2	1.50	28	49	246	3	-20.4	35.09	84	350	2659	12	21.96	84	136	1203	16	-4.7	17.14	38	115	1124	14	15.05	34	100	1041	13	-63.0
11	1.87	32	88	353	3	1.90	32	60	301	3	-14.0	40.10	88	392	2993	12	25.47	88	150	1313	17	-23.5	2.14	10	18	289	5	3.60	10	29	494	6	-0.5
12	1.86	32	88	353	3	1.62	32	60	301	3	-33.5	56.63	100	451	3410	15	31.83	100	176	1490	20	-21.7	10.77	14	64	1025	10	-	-	-	-	-	-
13	2.29	36	108	433	3	1.92	36	72	361	3	-33.1	52.82	96	412	3094	15	34.41	104	186	1556	20	-23.2	15.43	16	80	1281	11	-	-	-	-	-	-
14	2.30	36	108	433	3	2.03	36	72	361	3	-23.4	68.63	108	497	3683	17	39.55	108	207	1699	21	-25.9	21.33	18	98	1569	13	-	-	-	-	-	-
15	2.85	40	130	521	3	2.35	40	85	426	3	-26.1	87.77	120	586	4315	18	45.00	120	224	1827	22	-16.6	49.61	22	165	2581	19	-	-	-	-	-	-
16	2.75	40	130	521	3	2.34	40	85	426	3	-19.9	95.51	124	630	4659	18	48.94	124	237	1930	23	-17.6	60.67	24	191	2907	20	-	-	-	-	-	-
17	3.57	44	154	617	3	3.10	44	99	496	4	-32.0	119.82	136	712	5301	20	58.95	136	259	2128	25	-21.9	78.57	26	222	3373	23	-	-	-	-	-	-
18	4.48	48	180	721	4	3.42	48	114	571	4	-38.4	151.10	140	885	6531	21	64.24	140	272	2245	26	-19.7	658.30	32	1147	17003	38	-	-	-	-	-	-
19	5.04	52	208	833	4	4.04	52	130	651	4	-35.9	161.05	144	923	6802	21	70.79	144	288	2373	27	-19.6	563.87	34	906	13972	39	-	-	-	-	-	-
20	6.15	56	238	953	4	4.69	56	147	736	4	-40.8	218.39	156	1181	8658	22	82.21	156	314	2591	29	-20.5	249.25	36	419	6210	39	-	-	-	-	-	-

Table 3: Benchmarking results. **t**: Planning time in seconds; **L**: Plan length in number of actions; **S**: Number of nodes expanded; **N**: Number of optimization problems solved; **T**: Mean optimization time for each optimization problem in milliseconds; $\mathbf{O}_{\%}$: Relative objective value achieved by obj-EHC compared to EHC. Values with '-' denote problems that could not be solved in 1200 seconds.

than 50 ms for larger ones. This is important since large domain instances require solving tens of thousands of optimization problems, as seen in the table (column N). This kind of performance would not be possible if we used a non-convex non-linear optimization model with a general purpose non-linear optimizer.

Table 3 also illustrates the performance characteristics of the EHC search algorithm compared to the obj-EHC, that breaks heuristic ties based on the objective of each state, as discussed in Section 6.4. Column $O_{\%}$ shows the relative value of the objective of the obj-EHC approach compared to EHC. Negative values indicate an improvement in the objective. As expected, obj-EHC produces better plans in general. This improvement is very significant in some instances, showing a reduction in the objective of more than 50% (Figure 13). As explained in Section 6, obj-EHC is more computationally expensive as it requires an extra optimization problem minimizing the objective per state and it checks all the helpful descendants of each expanded node, as opposed to immediately picking the one with lower than the incumbent best heuristic. Therefore, we expected that obj-EHC would take longer than EHC to find plans. However, the results show that this is not the case, and that obj-EHC explores less states, finds better plans and takes less time in general than EHC. In particular, in the ROV domain, obj-EHC takes less than half the time to find plans than EHC for more difficult instances. We believe the objective guidance is being very effective in these domains, in which taking a sample in a nearby region or a further one looks the same in number of actions, but very different in terms of the objective.

However, obj-EHC is not always better than EHC. In particular, obj-EHC struggles in the Air Refueling domain with two UAVs. This is a challenging domain since the need to refuel is not accurately captured in the heuristic. Since the refueling activity requires the tanker plane to move wherever the UAV that wants to refuel is, this activity is expensive in terms of the objective, but is not deemed to be useful by the heuristic. This happens because

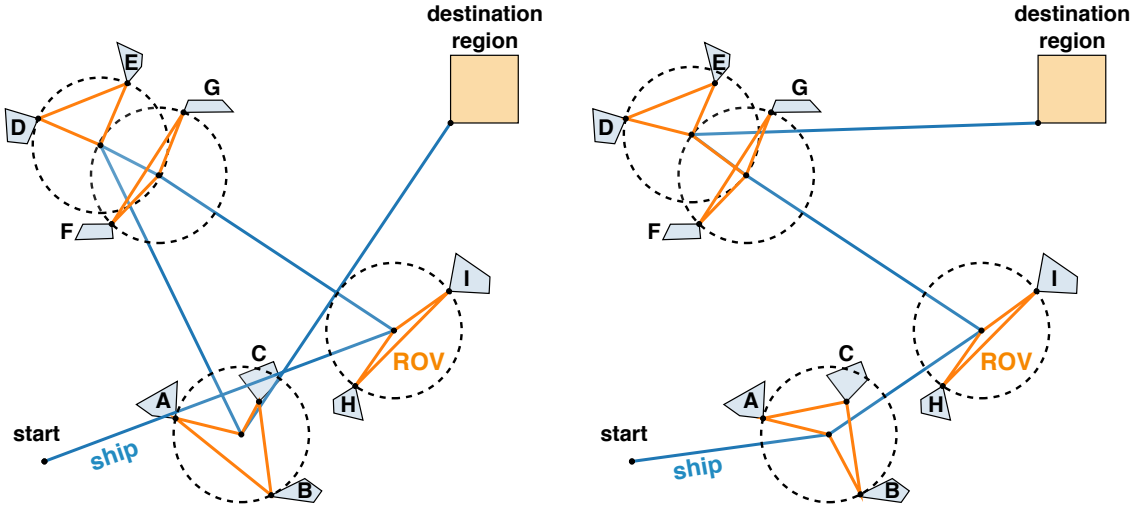


Figure 13: Plans found by ScottyActivity for problem 9 of the ROV domain using EHC search (a) and obj-EHC (b). The objective is a combination of the plan makespan and the distance traveled by ship. obj-EHC finds a better plan than EHC, with a 21% improvement in the objective, by taking samples at closer regions first.

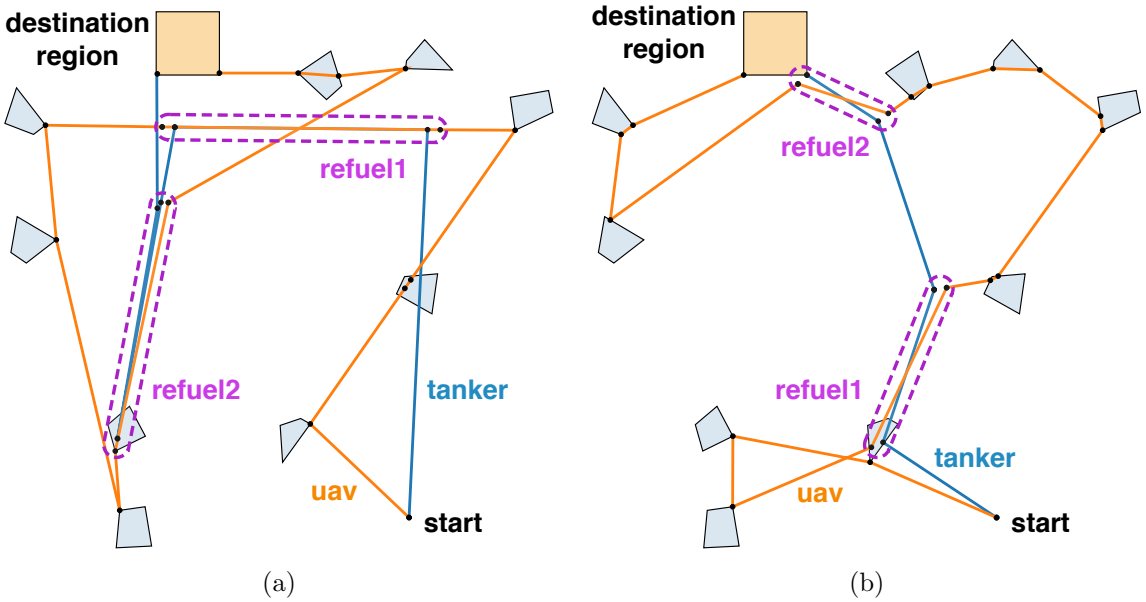


Figure 14: Plans found by ScottyActivity for problem 9 of the Air Refueling domain using EHC search (a) and obj-EHC (b). The objective is a combination of the plan makespan and the distance traveled by tanker plane. Note how the plan found by obj-EHC is much better (77% improvement in the objective).

	AUV-simplified									ROV-simplified								
	ScottyAct(EHC)					**Scotty1**		**POPCORN**		**ScottyAct**(EHC)					**Scotty1**		**POPCORN**	
	t	L	S	N	T	t	L	t	L	t	L	S	N	T	t	L	t	L
01	0.54	4	4	17	2	0.48	4	0.05	4	0.86	16	19	153	2	0.89	16	0.57	16
02	0.55	8	10	41	1	0.49	8	0.15	8	1.25	20	35	281	2	1.48	20	1.95	20
03	0.61	12	18	73	1	0.56	12	0.30	12	1.74	24	57	457	2	2.33	24	3.59	24
04	0.72	16	28	113	1	0.64	16	0.58	16	2.54	36	79	633	3	5.58	36	6.53	36
05	0.80	20	40	161	1	0.77	20	0.94	20	4.07	40	119	953	3	7.54	40	16.34	40
06	0.83	20	40	161	1	0.74	20	0.91	20	5.46	52	156	1214	3	12.79	52	24.77	52
07	0.92	24	54	217	1	0.92	24	1.50	24	7.85	56	213	1621	4	16.73	56	49.84	56
08	0.99	24	54	217	1	0.94	24	1.52	24	9.06	68	233	1753	4	55.40	84	77.21	68
09	1.15	28	70	281	1	1.24	28	2.22	28	14.36	72	328	2478	5	86.75	96	107.27	72
10	1.13	28	70	281	1	1.14	28	2.23	28	15.53	84	345	2565	5	119.23	100	150.69	84
11	1.43	32	88	353	1	1.44	32	3.29	32	18.34	88	396	2952	5	96.59	96	175.83	88
12	1.42	32	88	353	1	1.43	32	3.26	32	24.79	100	450	3335	6	142.33	108	242.64	92
13	1.49	34	92	369	2	1.53	34	3.89	34	23.36	96	411	3023	6	126.13	104	278.46	96
14	1.48	34	92	369	2	1.57	34	3.90	34	30.35	108	495	3639	7	180.93	116	343.66	108
15	1.80	38	114	457	2	2.12	38	4.88	38	38.62	120	578	4261	7	254.07	128	460.56	112
16	1.75	38	114	457	2	2.19	38	5.60	38	45.26	124	660	4826	8	158.47	108	525.74	116
17	2.22	42	138	553	2	2.50	42	6.58	42	56.49	136	743	5469	8	204.34	120	617.31	128
18	2.81	46	164	657	2	3.53	46	9.91	46	68.51	140	890	6484	8	271.64	132	783.90	140
19	3.31	50	192	769	2	4.12	50	13.01	50	73.40	144	926	6737	9	391.01	152	834.64	136
20	3.93	54	222	889	2	5.30	54	17.18	54	98.11	156	1177	8479	9	500.96	164	1028.76	148

Table 4: Benchmarking results for simplified domains **t**: Planning time in seconds; **L**: Plan length; **S**: Number of nodes expanded; **N**: Number of optimization problems solved; **T**: Mean optimization time for each optimization problem in milliseconds.

the fuel consumption is modeled with a *resource constrained norm effect*. Since this effect is not considered by the heuristic, the heuristic cannot predict, in advance, that refueling will be necessary. Therefore, the algorithm tends to prefer other irrelevant actions that are not useful (such as taking the same sample over and over again). In some instances, this causes the search to exhaust all options and fail (such as in problem 7), or to continuously pick wrong activities until the time limit is reached (in problems 12 and higher).

Finally, we evaluate ScottyActivity in the simplified linearized versions of the domains as described earlier (Table 4). Recall that our optimization model degrades gracefully to an LP when the problem solved does not require quadratic constraints. Therefore, these results let us answer the question of what is the performance penalty of switching from a linear program formulation to a SOCP one. As seen in the table, the difference between the mean optimization time for the linear problems solved in the simplified domains and the SOCP ones from the full domains is very small for the simpler instances and significant for more difficult instances. However, this difference is always well within an order of magnitude. Moreover, we should highlight that the linearized version of the domains is significantly simpler, as it not only linearizes some constraints (such as the ROV tether range ones) but also drops many other constraints, like the norm ones or the ones required to minimize the traveled distances. We can conclude that using SOCPs for consistency checking is not only practical, but that the performance trade-off is well worth it considering the added expressivity that they provide. Finally, we compare our planner in these simplified domains

against Scotty1 and POPCORN. Since our optimization model is significantly more complex than theirs, even in these linear domains, given the extra variables and constraints that we require, we expected that our planner would be slower. However, Table 4 shows that this is not the case and our planner performs significantly better. We hypothesize that this is due to the superior performance of the Gurobi solver compared to the solvers used by Scotty1 (CPLEX 12.4) and POPCORN (lpsolve 5.5). Additionally, POPCORN's test were kindly run on a slower i5-M540 2.53GHz processor by its authors, since they could not share the planner binary with us.

8.3 Comparison with a Mixed Integer Approach

As we explain in Section 6, when the plan skeleton that describes the chosen starts and ends of activities and their order is known, the state and control trajectories can be found efficiently by solving a convex optimization problem. The hybrid activity and trajectory planning problem reduces to finding the plan skeleton. This is hard due to the highly combinatorial nature of the problem. In this work we argue that this problem is best solved using heuristic forward search with a delete relaxation heuristic, as this method has proven to be very effective in activity planning problems. However, this problem could also be solved using a mixed-integer optimization approach. Mixed-integer solvers use branch and bound search algorithms. The branch and bound search is often guided by the solution to relaxed optimization problems in which the integer variables are allowed to take continuous values. We show in this section that our approach based on heuristic forward search performs at least two orders of magnitude better than an alternative mixed-integer approach.

The mixed-integer formulation that we use in this section employs the same decision variables and constrains for the state and control variables and for the activity durations, continuous conditions and effects, as presented in Section 7. An important difference between the mixed-integer approach that we use in this section and ScottyActivity is that the former can only find the solution to a planning problem when it is provided with a fixed length for the plan. This length indicates the maximum number of start and end activities (events) that the plan can have. The reason for this limitation is that the decision variables and constraints are fixed in the optimization problem. ScottyActivity, on the other hand, determines the required number of start and end activities as part of its search process. A well-known approach to solving planning problems with a mixed-integer approach when the plan length is not known consists in iteratively solving the problem with increasing plan lengths until a solution is found or the maximum plan length or time limit is exceeded. However, in the results shown in this section we provided the MIP planner with a fixed plan length, that matched, for each problem, the length of the plan found by ScottyActivity using the obj-EHC algorithm.

In order to select the start and end activities and their order, the MIP approach uses integer variables. For each step in the fixed length plan, we create a binary variable for each start or end activity that indicates whether the activity is selected for such step in the plan. The fixed plan length indicates the maximum number of start or end activities, but the MIP planner is allowed to find a shorter plan by selecting no-op activities at the last steps of the fixed length plan. Additional binary variables are created to represent the

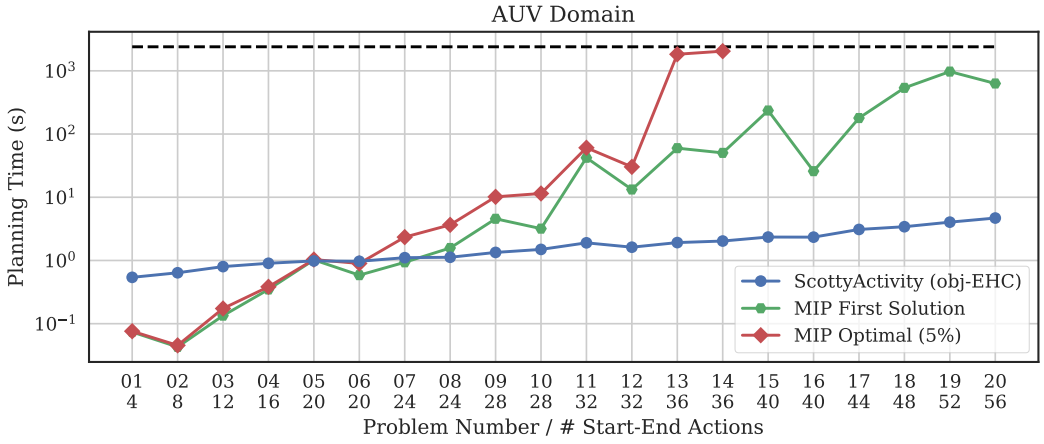
propositions that hold at each step in the plan. We use the standard big-M method to enable or disable constraints when an activity is selected.

We compared the performance of this MIP planner against ScottyActivity using the obj-EHC algorithm in the AUV and ROV domains from Section 8.2. We used a time limit of 2400 seconds for the MIP solver for each problem, and we used Gurobi 7.5 as the MIP solver. The MIP solver was able to solve all the problems in the AUV domain (Figure 15a). For small problems, up to instance 5, that require less than 20 events, the MIP planner was able to find a solution faster than ScottyActivity. In many cases, this solution was also the optimal (to 5% tolerance) for the given fixed length. However, the performance of the MIP solver degrades quickly as the complexity of the problem increases. Problems needing more than 40 events were solved two orders of magnitude slower than ScottyActivity. For example, the MIP planner found a feasible solution for instance 19 of the AUV domain in 970 seconds, while ScottyActivity solved the same problem in about 5 seconds. The MIP planner was able to find optimal solutions (proved to 5 % tolerance) for instances 14 and lower, which need fewer than 36 events.

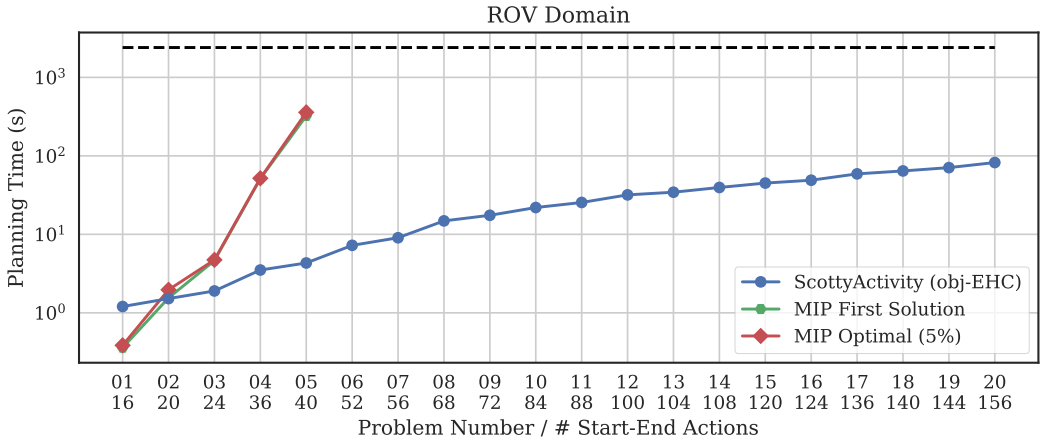
The ROV domain proved to be more challenging for the MIP solver (Figure 15b). In this case, the MIP solver was only able to solve the first five problems within the maximum time limit of 2400 seconds. Instance 5, requiring 40 events was solved by the MIP planner in 320 seconds. The same problem was solved by ScottyActivity using the obj-EHC algorithm in about 7 seconds. One of the reasons why the ROV domain is more challenging for the MIP planner is that the problems are more complex and require a higher number of activities. However, we speculate that this is not the only reason. In the AUV domain, the MIP planner was able to find solutions for problems requiring up to 56 events, while it was not able to find a solution for a problem requiring 52 events in the ROV domain. The reason this may happen is that the ROV domain not only requires longer plans, but the number of possible activities is also significantly higher in this domain than in the AUV domain. Furthermore, there are several activities that are mostly discrete, such as the deployment and recovery activities. For this activities the planner is probably not finding useful relaxations.

Another interesting comparison between the ScottyActivity planner and the MIP approach is the quality of the returned plans. We report, in Figure 16, the objective of the plans found by ScottyActivity and the MIP planner. For the MIP planner we are interested in the objective of the first solution found, the best solution found within the time limit and the optimal solution for the fixed plan length. The results show the ratio between the objective of the MIP solutions and the ScottyActivity solution for the AUV (Figure 16a) and ROV (Figure 16b) domains when using the obj-EHC algorithm. As seen in the figures, the obj-EHC algorithm returns fairly high quality solutions. In particular, the solutions found by ScottyActivity with the obj-EHC algorithms were always better than the first solution found by the MIP planner, even when this solution was often found two orders of magnitude slower. This results show the improvement of the obj-EHC algorithm when compared to the standard EHC algorithm, whose solutions are often comparable to the first solutions found by the MIP planner. One of the benefits of the MIP approach is that it is able to find optimal solutions for a fixed plan length. However, these solutions are often computationally expensive to find.

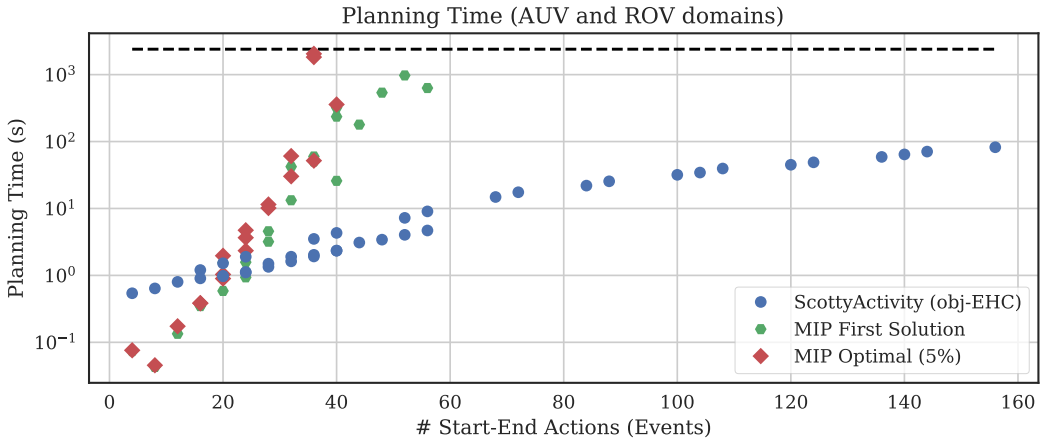
These results indicate that the MIP approach does not scale to large planning problems. Moreover, it is important to highlight that the MIP planner was provided with the number of



(a)

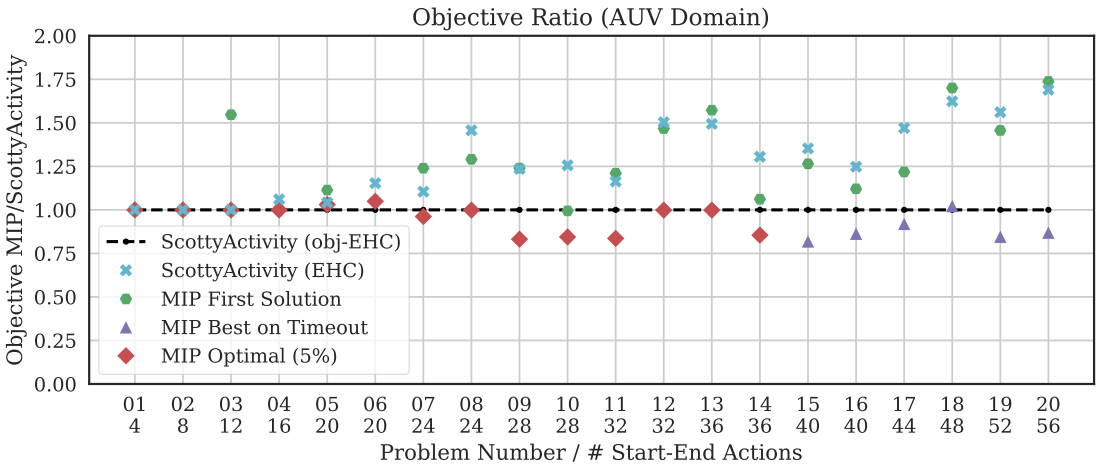


(b)

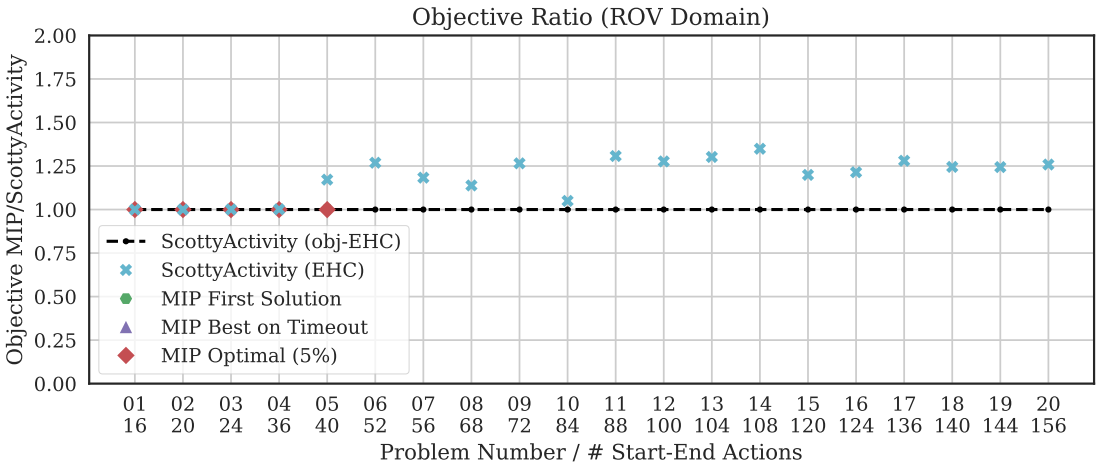


(c)

Figure 15: Planning time for ScottyActivity (obj-EHC algorithm) and the MIP approach in the AUV (a) and ROV (b) domains. The planning time for the first MIP solution (green) and optimal to 5% tolerance (red) are shown. Figure (c) shows the planning time for both domains as a function of the required number of start/end actions to solve the problem. A time limit of 2400 seconds was used.



(a)



(b)

Figure 16: Objective ratio between the MIP approach and ScottyActivity (obj-EHC algorithm) in the AUV (a) and ROV (b) domains. The figures show the ratio achieved by the first MIP solution (green), the optimal to 5% tolerance (red), and the best MIP solution found before the timeout (purple). A time limit of 2400 seconds was used.

required events for each planning problem. The runtime performance would be significantly worse if the MIP planner had to solve problems of increasing fixed-length until a solution was found. Additionally, the Gurobi solver is designed to use multiple cores simultaneously, while our search algorithm is single-core. While we do not believe that a mixed-integer approach is the best method to solve large hybrid activity and trajectory planning problems, it is a useful approach for solving small planning problems that require optimal solutions. Moreover, it may be possible to use a mixed-integer formulation to improve plans that are first found using a heuristic forward search approach. We leave this work for future research.

9. Conclusions and Future Work

While robots are quickly becoming more capable and accessible, there is still an unfulfilled need for controlling them efficiently and autonomously. In this work, we have presented ScottyActivity, a mixed discrete-continuous planner that aims to narrow the gap between the needs of robotic missions and the capabilities of state of the art automated planning techniques. We have shown that by allowing convex quadratic constraints on state variables and control variables that are jointly constrained and that affect multiple state variables simultaneously, we can model more realistic robotic missions that were not possible with previous planning approaches. As seen in the experimental results, our approach is scalable and performs equally or better than prior art in simplified problem instances that do not use the extended capabilities of our planner.

The advances introduced by our planner are possible due to several insights. First, we build upon previous work that inspires and makes possible our planner. Kongming illustrated the need for tightly coupled task and motion planning within a single planning framework and showed the expressive power of its approach in new robotic domains. COLIN showed the effectiveness of using heuristic forward search to handle the discrete part of the planning problem with linear programs handling the continuous effects and temporal constraints. Second, recent advances in convex optimization from the operations research community make it possible to solve Second Order Cone Programs (SOCPs) very efficiently, to the point that we can solve several SOCPs to test the feasibility of each search state and still solve the full planning problem in a reasonable time. Third, while the continuous change equations that our planner handles are inherently non-linear and non-convex, we propose an equivalent second order cone model that is efficient. This is enabled, in part, through a clever selection of decision variables that reduces the non-convex component to a linear constraint that restores convexity. Without this convex model ScottyActivity would not be possible, since a general non-linear solver would have to be used, which would have a performance orders of magnitude slower than SOCP solvers and, perhaps more importantly, would not be complete due to local minima. Fourth, while the TRPG-based heuristic used by COLIN and ScottyActivity is only applicable to linear state constraints, we show that we can exploit this heuristic by first producing automatic linear over approximations to the quadratic constraints and that these perform well in practice. Finally, we introduce several new robotic domains that showcase the new capabilities of our planner and that can be used as a benchmark by the planning community.

While this work advances the expressivity of the hybrid problems that can be solved with heuristic forward search techniques, ScottyActivity makes strong assumptions that we ought

to address. First, our planner only supports linear dynamics and practical constraints such as maximum curvature cannot be handled in our model. Moreover, ScottyActivity does not consider obstacles, since their associated non-convex constraints cannot be directly encoded in our convex optimization model. Finally, while returned plans are optimal conditioned on the chosen sequence of activities, our heuristic does not explicitly take the objective into account, and we cannot make any guarantees with respect to the optimality of the chosen sequence. As described in Section 3, in order to handle obstacles the ScottyPath planner, which we present in another work (Fernandez-Gonzalez, 2017), uses the convex model introduced in this work to find obstacle free trajectories and schedules from ScottyActivity plans. It does so by combining heuristic forward search and the convex model in order to find sequences of obstacle-free safe convex regions for each robot throughout the mission. In order to handle more complex dynamics, we propose MPC-Scotty, an additional online receding-horizon planner that is currently under development. This planner uses a single optimization model that combines a detailed dynamics discrete-time formulation over a limited horizon with a continuous-time linear dynamics formulation over the remaining long horizon of the full mission.

Acknowledgments

This work was partially supported by the Exxon Mobil Corporation, and the United States - Israel Binational Science Foundation (BSF). We would like to thank Emre Savas for running the POPCORN benchmarks. We would also like to thank all the members of the MIT MERS lab for endless insightful discussions and for their constant support throughout the years. Finally, we would also like to acknowledge the JAIR reviewers and editor for their thoughtful reviews, which helped us make significant improvements to this paper.

Appendix A. Proofs of Completeness and Optimality of PDDL-S Plans with Piecewise Constant Control

In this section, we provide the proofs for the completeness and optimality theorems stated in Section 5.

A.1 Completeness of PDDL-S Plans with Piecewise Constant Control

Assume the PDDL-S Problem has a valid PDDL-S Plan solution, p, with an arbitrary control trajectory. This solution p is characterized by:

- The schedule of its N start or end events, given by their execution times t_i with $i = 0, \ldots, N-1$.
- The control trajectory $\mathbf{c}(t)$.

We prove the completeness of PDDL-S plans with piecewise constant control by construction. We use the original valid PDDL-S plan p to construct the PDDL-S plan with piecewise constant control p_c in the following way:

- The execution times of the events, t_{ci}, are the same as the times of the original solution, $t_{ci} = t_i$. Since they are the same, we use refer to them as t_i from this point.
- The control trajectory, $\mathbf{c_c}(t)$, is a piecewise constant trajectory. Recall that a *stage*, s_i, is the period of time between two consecutive events taking place at t_i and t_{i+1}. The value of the vector of control variables is constant during each *stage*, $\mathbf{c_c}(t) = \mathbf{c_c}_i \in \mathbb{R}^m, i \in [t_i, t_{i+1})$, for $i = 0, \ldots, N-2$. Each constant value $\mathbf{c_c}_i$ at each stage takes the value of the average of the original control trajectory $\mathbf{c}(t)$ during that stage:

$$\mathbf{c_c}_i = \frac{1}{\Delta t_i} \int_{t_i}^{t_{i+1}} \mathbf{c}(t) dt, \tag{43}$$

  where $\Delta t_i = t_{i+1} - t_i$ is the duration of the stage.

Lemma A.1. *The piecewise constant control trajectory of p_c, $\mathbf{c_c}(t)$ satisfies all the control variable constraints at all times.*

Proof. Recall that each control variable constraint is restricted to be a convex constraint on the control variable vector in the form of $g(\mathbf{c}(t)) \leq 0$. Using (43), we can write:

$$g(\mathbf{c_c}_i) = g\left(\frac{1}{\Delta t_i} \int_{t_i}^{t_{i+1}} \mathbf{c}(t) dt\right) \tag{44}$$

We make use of Jensen's Inequality to assist our proof (Jensen, 1906). Jensen's Inequality states the following:

$$g\left(\int_a^b h(t) f(t) dt\right) \leq \int_a^b g\left(h(t)\right) f(t) dt \tag{45}$$

, where:

- f is a non-negative function such that $\int_a^b f(t)dt = 1$.
- h is any real-valued measurable function.
- g is convex over the range $[a, b]$.

We use the vector version of (45) on (44) by identifying terms as follows: $a = t_i$, $b = t_{i+1}$, $g = g$, $f(t) = \frac{1}{\Delta t_i}$, $\mathbf{h}(t) = \mathbf{c}(t)$. Note that Jensen's conditions hold since g is convex. This allows us to write:

$$g(\mathbf{c}_{\mathbf{c}i}) = g\left(\frac{1}{\Delta t_i}\int_{t_i}^{t_{i+1}} \mathbf{c}(t)dt\right) \leq \int_{t_i}^{t_{i+1}} g(\mathbf{c}(t))\frac{1}{\Delta t_i}dt \tag{46}$$

Since p is a valid PDDL-S plan, $g(\mathbf{c}(t)) \leq 0$ at every stage and, therefore, (46) reduces to $g(\mathbf{c}_{\mathbf{c}i}) \leq 0$ during stage s_i. Since this is applicable for every stage and every control variable constraint, we conclude that the piecewise constant control trajectory of p_c, $\mathbf{c}_{\mathbf{c}}(t)$, satisfies the control variable constraints. □

Lemma A.2. *The state variables that are not resources take the same values in the PDDL-S plan with piecewise constant control, p_c, as in the original plan, p.*

Proof. Recall that state variables that are not resources can only change their value due to active CLTE effects. Recall, as well, that the change in one such state variable x_j during a stage s_i is given by:

$$\Delta x_j(j) = \sum \Delta x_{j_{eff}}, \tag{47}$$

where *eff* represents each of the active CLTE effects during stage s_i. Recall as well from Section 4.2.1 that the change due to a CLTE on a state variable x during stage s_i is given by:

$$\Delta x_{CLTE}(j) = \int_{t_i}^{t_{i+1}} \mathbf{k}^T \cdot \mathbf{c}_{\mathbf{c}}(\tau)d\tau \tag{48}$$

In the case of the plan with piecewise constant control, $\mathbf{c}_{\mathbf{c}}(t)$ takes the constant value $\mathbf{c}_{\mathbf{c}i}$ during stage s_i:

$$\Delta x_{CLTE}(j) = \int_{t_i}^{t_{i+1}} \mathbf{k}^T \cdot \mathbf{c}_{\mathbf{c}i}d\tau = \mathbf{k}^T \cdot \frac{1}{\Delta t_i}\int_{t_i}^{t_{i+1}} \mathbf{c}(t)dt \int_{t_i}^{t_{i+1}} d\tau = \int_{t_i}^{t_{i+1}} \mathbf{k}^T \cdot \mathbf{c}(\tau)d\tau \tag{49}$$

(49) proves that the cumulative change due to CLTE effects during a stage s_i is the same regardless of whether the original control trajectory $\mathbf{c}(t)$ or its averaged value during the stage, $\mathbf{c}_{\mathbf{c}i}$, is used. Since the initial variables are the same, and the cumulative change during each stage is the same, the values of the state variables that are not resources are the same ($\mathbf{x}_i = \mathbf{x}(t_i) = \mathbf{x}_{\mathbf{c}}(t_i) = \mathbf{x}_{\mathbf{c}i}$) at each event in the original plan, p, and the plan with piecewise constant control, p_c. □

Corollary A.1. *The state trajectory of the PDDL-S plan with piecewise constant control, p_c, satisfies all the state constraints on the state variables that are not resources.*

Proof. Since the values are the same at the events, $\mathbf{x_c}_i = \mathbf{x}(t_i)$, and p is valid, p_c satisfies all the state constraints at the events. The overall maintenance conditions between events are also satisfied by $\mathbf{x_c}(t)$. This is the case because the overall conditions during stages between consecutive events are convex by definition and need to be satisfied also at those events. Section 5 proves by convexity that this is the case. □

Lemma A.3. *The resource variables take greater or equal values at the events in the PDDL-S plan with piecewise constant control, p_c, than in the original plan p.*

Proof. The change during stages on resources due to CLTE effects is the same as in the case of state variables that are not resources. The remaining change in resources is due to LNE or LSNE effects.

Recall that the change during stage s_i in resource r due to LNE effect is given by:

$$\Delta r_{LNE}(i) = -k \cdot \int_{t_i}^{t_{i+1}} \|\mathbf{c_c}(t)\| dt \tag{50}$$

Substituting for the constant value of $\mathbf{c_c}(t)$ during the stage:

$$\Delta r_{LNE}(i) = -k \cdot \left\| \int_{t_i}^{t_{i+1}} \mathbf{c}(t) \frac{1}{\Delta t_i} \right\| \int_{t_i}^{t_{i+1}} dt \tag{51}$$

We can use Jensen's inequality (45) again by identifying terms: $a = t_i$, $b = t_{i+1}$, $g(\mathbf{x}) = \|\mathbf{x}\|$, $f(t) = \frac{1}{\Delta t_i}$, $\mathbf{h}(t) = \mathbf{c}(t)$. The conditions for Jensen's inequality hold since norms are convex functions. Therefore, we obtain:

$$\left\| \int_{t_i}^{t_{i+1}} \mathbf{c}(t) \frac{1}{\Delta t_i} \right\| \cdot \Delta t_i \leq \int_{t_i}^{t_{i+1}} \|\mathbf{c}(t)\| \frac{1}{\Delta t_i} dt \cdot \Delta t_i \tag{52}$$

Note that (52) is the factor that multiplies $-k$ for the cumulative decrease of the LNE effect in the original plan with arbitrary control trajectory. Therefore, the decrease due to the LSE effect is smaller in the case of the piecewise constant control trajectory than in the plan with the arbitrary control trajectory.

Similarly, for LSNE effects, the change in a resource during stage s_i is given by:

$$\Delta r_{LSNE}(i) = -k \cdot \int_{t_i}^{t_{i+1}} \|\mathbf{c_c}(t)\|^2 dt \tag{53}$$

Substituting for the constant value of $\mathbf{c_c}(t)$ during he stage:

$$\Delta r_{LSNE}(i) = -k \cdot \left\| \int_{t_i}^{t_{i+1}} \mathbf{c}(t) \frac{1}{\Delta t_i} \right\|^2 \int_{t_i}^{t_{i+1}} dt \tag{54}$$

We can use Jensen's inequality (45) one more time with the correspondence: $a = t_i$, $b = t_{i+1}$, $g(\mathbf{x}) = \|\mathbf{x}\|^2$, $f(t) = \frac{1}{\Delta t_i}$, $\mathbf{h}(t) = \mathbf{c}(t)$. Jensen's conditions hold again since g is a convex function. It is a convex function because it is the composition of a convex non-decreasing function (x^2 is non-decreasing over the positive reals) and a convex function (the norm function). We then obtain:

$$\left\| \int_{t_i}^{t_{i+1}} \mathbf{c}(t) \frac{1}{\Delta t_i} \right\|^2 \cdot \Delta t_i \leq \int_{t_i}^{t_{i+1}} \|\mathbf{c}(t)\|^2 \frac{1}{\Delta t_i} dt \cdot \Delta t_i \tag{55}$$

Again, (55) is the factor that multiplies $-k$ for the cumulative decrease of the LSNE effect in the original plan with arbitrary control trajectory. Therefore, the decrease due to the LSNE effect is also smaller in the case of the piecewise constant control trajectory.

From Lemma A.2, the change produced on resources due to CLTE effects is the same in p_c as in p. However, the resources in p_c experience a smaller decrease in p_c during stages due to LSE or LSNE effects than in p. As a consequence, resources have always larger or equal values at the events in p_c with piecewise constant control trajectories than in p with its arbitrary control trajectory. □

Corollary A.2. *The resource variables in the PDDL-S plan with piecewise constant control satisfy all the constraints they are subject to.*

Proof. Recall that resources can only be subject to greater than some value constraints. Since p is a valid plan, the resources in p satisfy the constraints they are subject to. Since the resources take values greater or equal at the events in p_c than in p, the resources satisfy all their constraints at the events. Due to convexity and as proved in Section 5, the resources also satisfy the constraints at all intermediate points between events. □

Lemma A.4. *p_c is a valid PDDL-S plan.*

Proof. p_c is a valid PDDL-S plan since:

1. All temporal constraints are satisfied since the event execution times of p_c are the same of p, and p satisfies all temporal constraints.
2. All the propositional constraints (discrete conditions and effects of activities) are satisfied since p satisfies all propositional constraints and the ordering of the starts and ends of activities is the same.
3. The piecewise constant control trajectory satisfies all the control variable constraints (Lemma A.1)
4. The state variables satisfy all the state constraints (Corollaries A.1 and A.2).

□

Theorem 1 (Completeness of PDDL-S Plans with Piecewise Constant Control)**.** *If a PDDL-S problem has a solution, there always exists a solution that is a PDDL-S Plan with piecewise constant control.*

Proof. For every valid PDDL-S plan with an arbitrary control trajectory, it is also possible to construct a valid PDDL-S plan with a piecewise constant control trajectory (Lemma A.4). Therefore, there are no solvable PDDL-S problems that cannot be solved with a PDDL-S plan with piecewise constant control. □

A.2 Optimality of PDDL-S Plans with Piecewise Constant Control

Recall that the minimization objective of a PDDL-S problem is given as a linear combination of the following types of terms:

- The total duration of the plan (plan makespan)
- The value of a state variable at the end of the plan. In the case of a resource, its coefficient can only be negative (i.e. resources can only be maximized).
- $\int_0^T \|\mathbf{c}_e(\tau)\| d\tau$, where $\mathbf{c}_e$ is a vector of control variables and T is the plan makespan. Its coefficient can only be nonnegative (i.e. this term can be minimized but not maximized).
- A similar term with the same conditions with the square of the norm, $\int_0^T \|\mathbf{c}_e(\tau)\|^2 d\tau$.

Corollary A.3. *The makespan is the same in p_c than in p.*

Proof. All the events take place at the same time in p_c than in p. Therefore, the last event takes place at the same time in both plans. □

Corollary A.4. *Any state variable that is not a resource takes the same value at the end in p_c than in p.*

Proof. Since the values of the state variables that are not resources take the same value at all events in p_c than in p (Lemma A.2), they also take the same value at the end of the plan. □

Corollary A.5. *Any resource variable takes a larger value at the end of the plan in p_c than in p.*

Proof. Since the values of the resources take larger value at all events in p_c than in p (Lemma A.3), they also take larger values at the end of the plan. □

Lemma A.5. *$\int_0^T \|\mathbf{c}_e(\tau)\| d\tau$ takes a smaller or equal value in p_c than in p.*

Proof. The proof is similar to that of Lemma A.3. We can write the term as a sum of integrals over each stage in the plan:

$$\int_0^T \|\mathbf{c}_e(\tau)\| d\tau = \sum_{i=0}^{N-2} \int_{t_i}^{t_{i+1}} \|\mathbf{c}_e(\tau)\| d\tau \tag{56}$$

In a PDDL-S plan with piecewise constant control, $\mathbf{c}_e(t) = \mathbf{c}_{\mathbf{c}e}(t)$ takes a constant value $\mathbf{c}_{\mathbf{c}ei}$ during each stage s_i. Therefore, each stage term can be written as:

$$\int_{t_i}^{t_{i+1}} \|\mathbf{c}_{\mathbf{c}ei}\| d\tau = \left\| \int_{t_i}^{t_{i+1}} \mathbf{c}_e(t) \frac{1}{\Delta t_i} \right\| \cdot \Delta t_i \leq \int_{t_i}^{t_{i+1}} \|\mathbf{c}_e(t)\| \frac{1}{\Delta t_i} dt \cdot \Delta t_i \tag{57}$$

, where the equality follows from the definition of $\mathbf{c}_\mathbf{c}$ (43), and the inequality is, again, Jensen's inequality (45), and it is applied in the same way as in (52). We can then write:

$$\int_0^T \|\mathbf{c}_{\mathbf{c}e}(\tau)\|d\tau = \sum_{i=0}^{N-2} \int_{t_i}^{t_{i+1}} \|\mathbf{c}_{\mathbf{c}e}(\tau)\|d\tau \leq \sum_{i=0}^{N-2} \int_{t_i}^{t_{i+1}} \|\mathbf{c}_e(\tau)\|d\tau = \int_0^T \|\mathbf{c}_e(\tau)\|d\tau, \quad (58)$$

which proves that the term is smaller for p_c than for p. □

Lemma A.6. *$\int_0^T \|\mathbf{c}_e(\tau)\|^2 d\tau$ takes a smaller or equal value in p_c than in p.*

Proof. The proof is analogous to the proof for Lemma A.5. The difference is that Jensen's inequality is applied as in (55). That leads us to

$$\int_0^T \|\mathbf{c}_{\mathbf{c}e}(\tau)\|^2 d\tau = \sum_{i=0}^{N-2} \int_{t_i}^{t_{i+1}} \|\mathbf{c}_{\mathbf{c}e}(\tau)\|^2 d\tau \leq \sum_{i=0}^{N-2} \int_{t_i}^{t_{i+1}} \|\mathbf{c}_e(\tau)\|^2 d\tau = \int_0^T \|\mathbf{c}_e(\tau)\|^2 d\tau, \quad (59)$$

, which proves that the term is smaller for p_c than for p. □

Lemma A.7. *A PDDL-S plan with piecewise constant control, p_c, has better objective value than the plan with arbitrary control trajectory that it was formed from, p.*

Proof. The objective of a PDDL-S Plan is a linear combination of the terms presented above. Due to Corollaries A.3 and A.4, all the terms that could be negative or positive take the same value in p_c than in p. Due to Corollary A.5, all the terms that can only be negative are larger in p_c than in p. Finally, due to Lemmas A.5 and A.6, all the terms that can only be positive are smaller in p_c than in p. Therefore, p_c has a smaller or equal objective value than p. □

Theorem 2 (Optimality of PDDL-S Plans with Piecewise Constant Control)**.** *The optimal solution to a PDDL-S problem, if one exists, is a PDDL-S Plan with piecewise constant control.*

Proof. Since there always exists a PDDL-S plan with piecewise constant control for any valid PDDL-S plan for a PDDL-S problem (Theorem 1) and this plan always has a better or equal objective value (Lemma A.7), the optimal plans for PDDL-S problems are PDDL-S plan with piecewise constant control. □

Appendix B. Expressing PDDL-S Problems

ScottyActivity uses control variables to define continuous change and allows more complex state constraints than most activity planners. Since these features cannot be expressed with standard PDDL, we define an additional syntax that we describe in this section. We base our syntax in PDDL2.1 (Fox & Long, 2003) since it provides most of the expressiveness that we need, except for control variables. While PDDL+ (Fox & Long, 2006) significantly advances the expressiveness of PDDL2.1, its syntax does not define control variables (or parameters) and its more advanced features, such as processes and events, are not supported by our planner.

We now proceed to describe the additions to the PDDL syntax that we need to express ScottyActivity domains. The rest of the PDDL2.1 syntax is the same. We conclude this section by showing how the example scenario is expressed using this syntax.

B.1 Control Variables and Global Constraints on Control Variables

As opposed to Scotty (Fernandez-Gonzalez et al., 2015), where control variables were defined to be local to activities, control variables are now global in ScottyActivity. The reason for this change is twofold. First, in Scotty control variables could only affect one state variable at a time, and therefore global control variables could offer no benefits and could, in fact, lead to modeling errors. Second, allowing control variables to be global provides additional flexibility in how domains are defined. For example, suppose that an electrical car recharge station can recharge simultaneously cars A and B, but that the maximum current out of the station is limited. We could use control variables i_A and i_B to represent the current entering each car while recharging. While the recharge processes of each car could be represented with independent activities, we could impose a global constraint $i_A + i_B \leq i_{max}$ that ensures that the maximum current of the recharge station is respected. This could not be represented if the currents were local control variables to each recharge activity.

As explained in Section 4, each control variable $c_i \in CV$ is defined by its lower (c_{il}) and upper (c_{iu}) bounds. Each control variable is defined with the following syntax, that needs to appear before the definitions of activities:

```
(:control-variable <cvar-name>
 :bounds (and (>= ?value <lower-bound>)
              (<= ?value <upper-bound>)))
```

The global constraints on control variables can be of two types: linear constraints and norm constraints. We use the keyword `control-constraint` to define named groups of linear control variable constraints such as:

```
(:control-constraint <constraint-name>
 :condition (and ({<=, >=} <lin-expr-left> <lin-expr-right>)
                                    ...))
```

, where `<lin-expr-{left,right}>` are linear expressions on the control variables $c_i \in CV$.

Norm constraints on control variables are expressed by first defining vectors of control variables. Control variable vectors are lists of control variables that can optionally specify a maximum norm according to the following syntax:

```
(:control-variable-vector <vector-name>
```

```
:control-variables ((<cv1> <cv2> ... <cvn>))
:max-norm <max-norm-value>)
```

Control variable vectors are not formally required to define our effects or constraints and, therefore, they are not presented in our problem statement section. However, they are practical to express norm constraints (e.g. $\|\mathbf{c}\| \leq c_{max}$), as well as RNE effects and objectives. Therefore, we use them throughout the rest of this section.

B.2 Continuous Change with CLTE and RNE Effects

Since CLTE effects are effects in which the rate of change is expressed as a linear combination of control variables, we use a syntax very similar to that of PDDL2.1 for continuous effects with the difference that, instead of using constant rates of change, we use control variables. In particular, activities can contain effects like the following:

```
({increase,decrease} <state-var> <cvar-time-expr>)
```

, where `<cvar-time-expr>` is a linear combination of control variables multiplied by the `#t` term. As an example (**increase** (x) (∗ (vx) **#t**)) denotes that state variable x is subject to a rate of change that corresponds to the control variable v_x.

RNE effects, that depend on the norm of control variable vectors, are defined similarly by using the keywords `norm` or `norm-sq` to denote the norm or squared norm of control variable vectors. For example (**decrease** (b) (∗ 0.45 (**norm** (velocity) **#t**))) represents that the state variable b is subject to a rate of change of $-0.45\|\mathbf{velocity}\|$, where `velocity` is a vector of control variables as defined above.

B.3 Objectives

Similarly to PDDL2.1, the optional objective of a planning problem is defined with the keyword `metric`, according to the following syntax:

```
(:metric minimize <linear-expr>)
```

As described in Section 4, the minimization objective is a linear combination of the following types of terms:

- State variables at the end of the plan ($x_i \in V$)
- Plan makespan (using the keyword `total-time`)
- Sum-products of the norms or squared norms of control variable vectors and the durations they are active for. Since control variables are restricted to be piecewise constant, the minimization objective given by the expression `norm(velocity)`, for example, minimizes the sum of the norm of the velocity control variable vector in each segment multiplied by the duration of that segment ($\sum_j \|\mathbf{v}(j)\| \cdot \Delta t_j$). In the case of a velocity, this is equivalent to minimizing the distance traveled by a vehicle. However, a similar term involving the squared norm of a control variable vector is also possible.

B.4 Representing Advanced Convex State Constraints Through State Space Regions

PDDL2.1 syntax allows activity conditions on state variables to be defined using polynomial expressions. Since ScottyActivity only supports convex quadratic state conditions, polynomial expressions are able to represent all the conditions that we can handle. However, manually specifying state conditions using polynomial equations is cumbersome and repetitive, since the same conditions often need to be used in different parts of the domain definition. Moreover, the convex quadratic conditions that ScottyActivity can handle can become fairly complicated and hard to write using direct equations. Polynomial state constraints are still allowed in ScottyActivity. However, for the reasons outlined above, we have developed a region-based system to express state conditions that arise naturally in robotic scenarios in a more succinct and clear way.

We call a *region* the part of the state-space that is allowed by a condition. Regions are defined as a collection of *primitive regions* defined on one or more *parameters*. Analogous to the parameters in PDDL activities, the region parameters are placeholders that are replaced by expressions of state variables when the regions are used in actual activity conditions. Regions are defined before activities with the following syntax:

```
(:region <region-name>
 :parameters (?par1 ?par2 ...)
 :condition (and <primitive-region1(parameter-expr1)>
                 <primitive-region2(parameter-expr2)>
                 ...                                    )
 [:linear-approximation (and ({<=,=,>=} <lin-exp-left> <lin-exp-right>)
                             ...)])
```

Regions defined in this way represent the intersection of the provided primitive regions. Since the primitive regions are convex, their intersection is also convex. We only support regions defined as intersections (`and`) and not disjunctions (`or`) because the latter are not guaranteed to be convex. ScottyActivity supports both linear conditions as well as quadratic conditions, as long as they are convex. For reasons that will be explained in Section 6, the heuristic needs linear over-approximations for quadratic conditions. Our quadratic region primitives, such as circles, automatically add these linear over-approximations (such as surrounding boxes). In general, automatic linear over-approximations cannot be deduced easily from arbitrary quadratic equations. Therefore, users specifying conditions manually using quadratic equations need to provide a list of linear inequalities that over approximate the quadratic region using the optional parameter `linear-approximation`.

Regions defined in this way can then be used in the *at start*, *at end* or *over all* conditions of activities. Region conditions are defined with the following syntax:

```
(inside (<region-name> <par-expr1(state-variables)>
                       <par-expr2(state-variables)
                       ...                        ))
```

Note that we only support `inside` and not `outside` conditions. In effect, being inside a convex region constitutes a convex condition, while being outside a convex region does not. The quadratic solvers that ScottyActivity uses do not support non-convex conditions. However, even if they did, we would not be able to guarantee that invariant conditions are

satisfied throughout a piecewise linear trajectory by only checking the switch points (like we showed we can do with convex conditions in Section 5).
Regions used in conditions need to be supplied with a list containing the values to be used in place of their parameters. Each parameter can be given as a state variable, or as an expression involving one or more state variables. This provides additional flexibility to compose complicated regions from simpler ones. As an example, let us consider a sampling region defined by a rectangle.

```
(:region sampling-region
  :parameters (?x ?y)
  :condition (and (in-rect (?x ?y) :corner (5 -10) :width 20 :height 30)))
```

The region `sampling-region(?x, ?y)` has parameters `?x` and `?y`. Note that these parameters are not state variables, but simply placeholders that serve to indicate what the internal primitive region `in-rect` should be defined with respect to. The primitive region `in-rect(?x, ?y)` defines the rectangular 2d region given by the inequalities $\text{corner}_x \leq ?x \leq \text{corner}_x + \text{width}$ and $\text{corner}_y \leq ?y \leq \text{corner}_y + \text{height}$. The `sampling-region` can now be used to indicate, for example, that an AUV needs to remain inside the region while a sample is being taken:

```
(over all (inside (sampling-region (x-auv) (y-auv))))
```

, where `x-auv` and `y-auv` are state variables that denote the position of the AUV. Note that `sampling-region` can be reused in a different activity condition just by feeding a different set of parameters as required. For example, we could indicate that the ship needs to remain in the same region while another activity is being executing just by using (**inside** (sampling−region (x−ship) (y−ship))). The expressions passed as parameters to the regions can be arbitrarily complex, which makes it very simple to specify complex conditions. For example, we could impose that instead of requiring the AUV or the ship to be inside the region, we would like the center point of the two vehicles to remain in the region by using the following expression:

```
(over all (inside (sampling-region (/ (+ (x-auv) (x-ship)) 2)
                                   (/ (+ (y-auv) (y-ship)) 2))))
```

The internal expression engine in ScottyActivity propagates the parameter expressions in order to obtain the resulting linear conditions $\text{corner}_x \leq \frac{1}{2}(x_{AUV} + x_{ship}) \leq \text{corner}_x + \text{width}$ and $\text{corner}_y \leq \frac{1}{2}(y_{AUV} + y_{ship}) \leq \text{corner}_y + \text{height}$. Expressions passed as parameters to regions can be arbitrarily complex as long as the resulting final expression remains convex quadratic, due to the reasons explained before.

B.4.1 Primitive Regions

In the current version of our planner we have implemented multiple *primitive regions* such as `in-rect`. Because the robotic planning problems we have spent more time on can be expressed with two dimensional coordinates, the primitives expressed here are 2d conditions. However, our system is not restricted to two dimensions, and additional region primitives can be defined for any number of dimensions naturally.

- Manual convex quadratic inequalities or linear equalities: ({<=,=,>=} <expr−left> < expr−right>)

- Rectangles: (**in−rect** (?x ?y) **:corner** (<c_x> <c_y>) **:width** <w> **:height** <h>)
  The rectangle primitive enforces the conditions $c_x \leq ?x \leq c_x + w$ and $c_y \leq ?y \leq c_y + h$. Since these conditions are linear, this primitive does not add linear approximations.

- Convex polygons: (**in−poly** (?x ?y) **:vertices** ((<v_{1x}> <v_{1y}>) ... (<v_{nx}> <v_{ny}>)))
  The convex polygon is given by a list of its vertices. Again, no linear approximation equations are needed in this case.

- Circles: (**in−circle** (?x ?y) **:center** (<c_x> <c_y>) **:r** <r>)
  The circle primitive enforces the convex quadratic condition $(?x - c_x)^2 + (?y - c_y)^2 \leq r^2$. This primitive also adds the linear over-approximation conditions of an square of center (c_x, c_y) and side $2r$ surrounding the circle.

- Maximum distance between entities: (**max−distance** ((?x1 ?y1) (?x2 ?y2)) **:d** <d>)
  This primitive ensures that point entities of positions $(?x1, ?y1)$ and $(?x2, ?y2)$ always remain within distance d by enforcing the quadratic condition $(?x1 - ?x2)^2 + (?y1 - ?y2)^2 \leq d^2$. This primitive also provides the linear over-approximation given by the equations $|?x1 - ?x2| \leq d$ and $|?y1 - ?y2| \leq d$.

  Note that, thanks to our powerful parameter expression system, the condition represented by `max-distance` can also be represented with a `circle` region in which the parameters are the x,y components of the difference vector between the entities:

```
(:region max-distance-region
         :parameters (?x1 ?y1 ?x2 ?y2)
         :condition (in-circle (- ?x1 ?x2) (- ?y1 ?y1)
         :center (0 0) :r <d>))
```

- Other previously defined regions: (**in−region** <region−name> (<par−expr1> ...))
  A region can also be composed by the intersection of previously defined regions. This makes it easy to define complex regions from other simpler, reusable regions.

Appendix C. Example Scenario in PDDL-S Syntax

We now proceed to represent our motivation example from Section 3.1 with the syntax we have defined in Appendix B. We start by defining the state variables of the problem, which are the x, y positions of the ship, (x_s, y_s), the AUV, (x_a, y_a) and the ROV, (x_r, y_r). Furthermore, the battery of the AUV is also a state variable in this problem. State variables are defined using the same syntax from PDDL2.1:

```
(:functions
 (xs) (ys) (xr) (yr)
 (xa) (ya) (ba))
```

There are six control variables in our problem, the x, y velocities of each vehicle. The maximum velocity of each vehicle is upper-bound constrained. As an example, the velocity of the ship (`vel-ship`) is defined and constrained with the following declaration:

```
(:control-variable vx-s
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable vy-s
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable-vector vel-ship
 :control-variables ((vx-s) (vy-s))
 :max-norm 2)
```

We define the rectangular region `mission-region` to limit the area that any of the vehicles can visit. Other regions, such as the sampling regions are polygons and are defined with their vertices according to the following syntax:

```
(:region mission-region
 :parameters (?x ?y)
 :condition  (and (in-rect (?x ?y) :corner (0 -100) :width 500 :height 500)))
(:region regionA
 :parameters (?x ?y)
 :condition
 (and (in-poly (?x ?y) :vertices ((125 0) (130 -15) (115 -30) (100 0)))))
```

Since the ROV is tethered to the ship, it always needs to remain within a distance that is the length of the tether. Moreover, the AUV and the ROV can only be recovered by the ship when they are close enough. These two conditions are represented with the regions `rov-range` and `recover-range` defined next:

```
(:region rov-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition  (and
              (max-distance ((?x1 ?y1) (?x2 ?y2)) :d 60)))

(:region recover-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition  (and
              (max-distance ((?x1 ?y1) (?x2 ?y2)) :d 2)))
```

In order to enforce the tether constraint, the ROV and the ship are forced to remain in the `rov-range` region while they move. We present the `navigate-ROV` activity next to show how this is expressed:

```
(:durative-action navigate-ROV
```

```
:duration  (and (>= ?duration 0.1) (<= ?duration 200))
:condition (and
            (at start (rov-still))
            (over all (rov-deployed))
            (over all (inside (mission-region (xr) (yr))))
            (over all (inside (rov-range (xr) (yr) (xs) (ys)))))
:effect    (and
            (at start (not (rov-still)))
            (at end (rov-still))
            (at start (not (rov-positioned)))
            (at end (rov-positioned))

            (increase (xr) (* (vx-r) #t))
            (increase (yr) (* (vy-r) #t))))
```

As explained in the beginning of this section, the continuous change in the position of the ROV is represented with a CLTE effect that indicates, for example, that the rate of change of the position (x_r) is proportional to the control variable that represents the v_x velocity of the ROV. RNE effects, such as the one that decreases the AUV battery are expressed in a similar way. The continuous effects of the `navigate-AUV` activity are given by:

```
(increase (xa) (* (vx-a) #t))
(increase (ya) (* (vy-a) #t))
(decrease (ba) (* 1 (norm (vel-auv)) #t ))
```

, which indicates that the rate of change of the battery is proportional to the space traveled by the AUV ($\dot{b}_a = -\|\mathbf{v}_a\|$). The `navigate-AUV` also has an invariant condition (**over all** (>= (b_a) 0)) that ensures that the battery constraint is respected.

Finally, the minimization objective in this problem is the sum of the total plan length and the distance traveled by the ship. This objective is defined as:

```
(:metric minimize (+ (* 1 (total-time) )
                     (* 1 (norm (vel-ship))))))
```

The other discrete conditions and effects, as well as the discrete objectives are defined as in PDDL2.1 and are omitted here for the sake of brevity.

Appendix D. Benchmark Domains

In this appendix we provide the PDDL sources of some of the instances of the benchmark domains described in Section 8.2.

D.1 The AUV Domain

The PDDL of instance 3 of the AUV domain is provided next. In this domain the AUV needs to visit regions A, B and C. The other instances (1-20) are analogous. The only difference between the original and the simplified, linearized version is that the later does not include the maximum norm constraint (indicated with the **:control-variable-vector** statement). The domain file (`auv03-domain.pddl`) is displayed next:

```
(define (domain auv-2D-3)
(:predicates
  (sample-takenA)
  (sample-takenB)
  (sample-takenC)
  (can-move))
(:functions (x) (y))

;; Control Variables
(:control-variable vel-x
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable vel-y
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable-vector vel-auv
 :control-variables ((vel-x) (vel-y))
 :max-norm 2)

;; Regions
(:region mission-region
  :parameters (?x ?y)
  :condition (and (in-rect (?x ?y) :corner (0 0) :width 100 :height 100)))
(:region regionA
  :parameters (?x ?y)
  :condition (and (in-rect (?x ?y) :corner (80 70) :width 10 :height 10)))
(:region regionB
  :parameters (?x ?y)
  :condition (and (in-rect (?x ?y) :corner (55 40) :width 5 :height 5)))
(:region regionC
  :parameters (?x ?y)
  :condition (and (in-rect (?x ?y) :corner (30 30) :width 10 :height 10)))

;; Activities
(:durative-action glide
 :duration (and (>= ?duration 0.1) (<= ?duration 200))
 :condition (and (at start (can-move))
                 (over all (inside (mission-region (x) (y)))))
 :effect    (and (at start (not (can-move)))
                                (at end (can-move))
                 (increase (x) (* (vel-x) #t))
                                (increase (y) (* (vel-y) #t))))

(:durative-action take-sampleA
```

```
 :duration (and (>= ?duration 2) (<= ?duration 8))
 :condition (and  (at start (can-move))
                  (over all (inside (regionA (x) (y))))
                  (at end (inside (regionA (x) (y)))))
:effect (and      (at start (not (can-move)))
                  (at end (can-move))
                  (at end (sample-takenA))))

(:durative-action take-sampleB
 :duration (and (>= ?duration 2) (<= ?duration 8))
 :condition (and  (at start (can-move))
                  (over all (inside (regionB (x) (y))))
                  (at end (inside (regionB (x) (y)))))
:effect (and      (at start (not (can-move)))
                  (at end (can-move))
                  (at end (sample-takenB))))

(:durative-action take-sampleC
 :duration (and (>= ?duration 2) (<= ?duration 8))
 :condition (and  (at start (can-move))
                  (over all (inside (regionC (x) (y))))
                  (at end (inside (regionC (x) (y)))))
:effect (and      (at start (not (can-move)))
                  (at end (can-move))
                  (at end (sample-takenC)))))
```

The problem file (`auv03-problem.pddl`) is displayed next:

```
(define (problem auv-3D-problem-3)
  (:domain auv-2D-1)
  (:init
    (can-move)
    (= (x) 0) (= (y) 0))
  (:goal (and
           (sample-takenA)
           (sample-takenB)
           (sample-takenC))))
(:metric minimize (+ (* 1 (total-time))))
```

D.2 The ROV Domain

We provide the PDDL sources for instance 6 of the ROV domain. In this problem, the ROV needs to visit regions A, B, C, D, E and F (Figure 11).

D.2.1 Original (Quadratic) Version

The domain file (`rov06-domain.pddl`) is displayed below:

```
(define (domain rov-6)
(:predicates
  (rov-deployed)(rov-onboard)
  (rov-navigating)(rov-still)
  (rov-positioned)(ship-arrived)
  (mission-ongoing)
  (sample-takenA)(sample-takenB)
        (sample-takenC)(sample-takenD)
        (sample-takenE)(sample-takenF))

(:functions (xs) (ys) (xr) (yr))

;; Control Variables
(:control-variable vx-s
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable vy-s
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable-vector vel-ship
 :control-variables ((vx-s) (vy-s))
 :max-norm 2)
(:control-variable vx-r
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable vy-r
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable-vector vel-rov
 :control-variables ((vx-r) (vy-r))
 :max-norm 2)

;; Regions
(:region mission-region
  :parameters (?x ?y)
  :condition (and (in-rect (?x ?y) :corner (0 0) :width 100 :height 100)))
(:region region-port
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((80.00000 80.00000)(80.00000
    90.00000)(90.00000 90.00000)(90.00000 80.00000)(80.00000 80.00000)))))
(:region regionA
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((39.37217 36.35934)(39.62838
    41.83741)(33.58334 38.41339)(35.90700 36.75789)(39.37217 36.35934)))))
(:region regionB
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((53.20386 24.86533)(59.77362
    23.94972)(60.97144 25.64728)(58.46709 27.47164)(53.20386 24.86533)))))
(:region regionC
  :parameters (?x ?y)
```

```
  :condition (and (in-poly (?x ?y) :vertices ((54.84244 42.09887)(53.85109
    44.74345)(48.76991 42.71553)(51.70078 38.83075)(54.84244 42.09887)))))
(:region regionD
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((14.22096 82.10052)(14.54697
    77.25059)(17.45469 76.93250)(19.08229 80.64577)(14.22096 82.10052)))))
(:region regionE
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((32.26246 85.87668)(34.33392
    88.33325)(34.46927 90.12637)(30.73706 91.88235)(32.26246 85.87668)))))
(:region regionF
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((30.13904 62.94699)(29.93304
    65.07422)(25.68036 65.04391)(24.56301 62.75958)(30.13904 62.94699)))))

(:region rov-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition (and (max-distance ((?x1 ?y1) (?x2 ?y2)) :d 10)))
(:region recover-range
  :parameters (?x1 ?y1 ?x2 ?y2)
  :condition (and (max-distance ((?x1 ?y1) (?x2 ?y2)) :d 0.5)))

;; Activities
(:durative-action navigate-ship
 :duration (and (>= ?duration 0.1) (<= ?duration 200))
 :condition (and (over all (mission-ongoing))
                 (over all (rov-onboard))
                 (over all (inside (mission-region (xs) (ys))))
                 (over all (inside (mission-region (xr) (yr)))))
 :effect (and
          (increase (xs) (* (vx-s) #t))
          (increase (ys) (* (vy-s) #t))
          ;; Also move ROV state variables
          (increase (xr) (* (vx-s) #t))
          (increase (yr) (* (vy-s) #t))))

(:durative-action navigate-ROV
 :duration (and (>= ?duration 0.1) (<= ?duration 200))
 :condition (and (over all (mission-ongoing))
                 (at start (rov-still))
                 (over all (rov-deployed))
                 (over all (inside (mission-region (xr) (yr))))
                 (over all (inside (rov-range (xr) (yr) (xs) (ys)))))
 :effect     (and (at start (not (rov-still)))
                 (at end (rov-still))
                 (at start (not (rov-positioned)))
                 (at end (rov-positioned))
                 (increase (xr) (* (vx-r) #t))
                 (increase (yr) (* (vy-r) #t))))

(:durative-action deploy-ROV
 :duration  (and (>= ?duration 10) (<= ?duration 10))
 :condition (and (over all (mission-ongoing))
                 (at start (rov-onboard)))
 :effect (and (at start (not (rov-onboard)))
```

```
              (at end (rov-still))
              (at end (rov-deployed))))

(:durative-action recover-ROV
 :duration  (and (>= ?duration 40) (<= ?duration 40))
 :condition (and (over all (mission-ongoing))
                 (at start (rov-deployed))
                 (over all (rov-still))
                 (over all (rov-positioned))
                 (over all (inside (recover-range (xr) (yr) (xs) (ys)))))
 :effect (and (at start (not (rov-deployed)))
              (at end (rov-onboard))
              (at start (not (rov-positioned)))))

(:durative-action arrive-port
:duration (and (>= ?duration 2) (<= ?duration 2))
:condition (and (over all (rov-onboard))
                (over all (inside (region-port (xs) (ys)))))
:effect (and (at end (ship-arrived))
             (at start (not (mission-ongoing)))))


(:durative-action take-sampleA
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionA (xr) (yr))))
                (at end (inside (regionA (xr) (yr)))))
:effect (and (at end (sample-takenA))
             (at end (not (rov-positioned)))))

(:durative-action take-sampleB
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionB (xr) (yr))))
                (at end (inside (regionB (xr) (yr)))))
:effect (and (at end (sample-takenB))
             (at end (not (rov-positioned)))))

(:durative-action take-sampleC
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionC (xr) (yr))))
                (at end (inside (regionC (xr) (yr)))))
:effect (and (at end (sample-takenC))
             (at end (not (rov-positioned)))))
```

```
(:durative-action take-sampleD
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionD (xr) (yr))))
                (at end (inside (regionD (xr) (yr)))))
:effect (and (at end (sample-takenD))
             (at end (not (rov-positioned)))))

(:durative-action take-sampleE
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionE (xr) (yr))))
                (at end (inside (regionE (xr) (yr)))))
:effect (and (at end (sample-takenE))
             (at end (not (rov-positioned)))))

(:durative-action take-sampleF
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionF (xr) (yr))))
                (at end (inside (regionF (xr) (yr)))))
:effect (and (at end (sample-takenF))
             (at end (not (rov-positioned))))))
```

The ROV instance 6 problem file:

```
(define (problem rov-problem-6)
(:domain rov-6)
(:init (rov-onboard)
       (rov-still)
       (mission-ongoing)
       (= (xs) 20.0)(= (ys) 30.0)
       (= (xr) 20.0)(= (yr) 30.0))
(:goal (and (sample-takenA)(sample-takenB)
            (sample-takenC)(sample-takenD)
            (sample-takenE)(sample-takenF)
            (rov-onboard)(ship-arrived))))
(:metric minimize (+ (* 0.1 (total-time) )
                     (* 2.5 (norm-sq (vel-ship)))))
```

D.2.2 Simplified, Linearized Version

In the linearized version the velocities of the AUV and the ship are not norm constrained and therefore, the **:control-variable-vector** statements do not appear in the domain file. Moreover, the original convex quadratic constraints that represent the maximum distance between the ROV and the ship (represented with the `rov-range` and `recover-range` re-

gions) are replaced by linear approximations (in particular, octagon over-approximations). Therefore the `rov-range` and `recover-range` regions are described in the following way:

```
(:region rov-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition (and
             (<= (+ (* 1.0 (- ?x1 ?x2)) (* 2.414 (- ?y1 ?y2))) 24.142)
             (<= (+ (* -1.0 (- ?x1 ?x2)) (* 2.414 (- ?y1 ?y2))) 24.142)
             (<= (+ (* -2.414 (- ?x1 ?x2)) (* -1.0 (- ?y1 ?y2))) 24.142)
             (<= (+ (* -2.414 (- ?x1 ?x2)) (* 1.0 (- ?y1 ?y2))) 24.142)
             (<= (+ (* -1.0 (- ?x1 ?x2)) (* -2.414 (- ?y1 ?y2))) 24.142)
             (<= (+ (* 1.0 (- ?x1 ?x2)) (* -2.414 (- ?y1 ?y2))) 24.142)
             (<= (+ (* 2.414 (- ?x1 ?x2)) (* -1.0 (- ?y1 ?y2))) 24.142)
             (<= (+ (* 2.414 (- ?x1 ?x2)) (* 1.0 (- ?y1 ?y2))) 24.142)))

(:region recover-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition (and
             (<= (+ (* 1.0 (- ?x1 ?x2)) (* 2.414 (- ?y1 ?y2))) 1.207)
             (<= (+ (* -1.0 (- ?x1 ?x2)) (* 2.414 (- ?y1 ?y2))) 1.207)
             (<= (+ (* -2.414 (- ?x1 ?x2)) (* -1.0 (- ?y1 ?y2))) 1.207)
             (<= (+ (* -2.414 (- ?x1 ?x2)) (* 1.0 (- ?y1 ?y2))) 1.207)
             (<= (+ (* -1.0 (- ?x1 ?x2)) (* -2.414 (- ?y1 ?y2))) 1.207)
             (<= (+ (* 1.0 (- ?x1 ?x2)) (* -2.414 (- ?y1 ?y2))) 1.207)
             (<= (+ (* 2.414 (- ?x1 ?x2)) (* -1.0 (- ?y1 ?y2))) 1.207)
             (<= (+ (* 2.414 (- ?x1 ?x2)) (* 1.0 (- ?y1 ?y2))) 1.207)))
```

D.3 The Air Refueling domain

As an example, we provide the PDDL sources for instance 15 of the Air Refueling domain. In this problem there are two UAVs that need to take images in six regions, while refueling from a single tanker airplane. For the reasons explained in Section 8.2.3, no linearized version is provided for this domain.

Domain (`onair15-domain.pddl`):

```
(define (domain onair-refuel-15)
(:predicates
  (can-start)
  (uav-canfly)(uav-flying)(uav-available)
  (uav2-canfly)(uav2-flying)(uav2-available)
  (tanker-flying)
  (mission-ongoing)
  (arrived)
  (photo-takenA)(photo-takenB)
  (photo-takenC)(photo-takenD)(photo-takenE))
(:functions (xt) (yt) (xb) (yb) (bb) (xb2) (yb2) (bb2))

;; Control Variables
(:control-variable vx-t
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable vy-t
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable-vector vel-tanker
 :control-variables ((vx-t) (vy-t))
 :max-norm 2)
(:control-variable vx-b
 :bounds (and (>= ?value -3.0) (<= ?value 3.0)))
(:control-variable vy-b
 :bounds (and (>= ?value -3.0) (<= ?value 3.0)))
(:control-variable-vector vel-uav
 :control-variables ((vx-b) (vy-b))
 :max-norm 3)
(:control-variable vx-b2
 :bounds (and (>= ?value -3.0) (<= ?value 3.0)))
(:control-variable vy-b2
 :bounds (and (>= ?value -3.0) (<= ?value 3.0)))
(:control-variable-vector vel-uav2
 :control-variables ((vx-b2) (vy-b2))
 :max-norm 3)
(:control-variable vx-b-ref
 :bounds (and (>= ?value -0.5) (<= ?value 0.5)))
(:control-variable vy-b-ref
 :bounds (and (>= ?value -0.5) (<= ?value 0.5)))
(:control-variable-vector vel-uav-ref
 :control-variables ((vx-b-ref) (vy-b-ref))
 :max-norm 0.5)
(:control-variable bat-recharge-rt
 :bounds (and (>= ?value 0.5) (<= ?value 10)))

;; Regions
(:region mission-region
 :parameters (?x ?y)
```

```
 :condition (and (in-rect (?x ?y) :corner (0 0) :width 100 :height 100)))
(:region end-region
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((30.00000 80.00000)(30.00000
    90.00000)(40.00000 90.00000)(40.00000 80.00000)(30.00000 80.00000)))))
(:region regionA
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((69.28348 48.10923)(68.00933
    45.38239)(73.61835 42.22267)(74.51618 48.55133)(69.28348 48.10923)))))
(:region regionB
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((8.00984 57.59487)(7.01760
    51.92697)(9.45458 50.25484)(14.20403 53.92992)(8.00984 57.59487)))))
(:region regionC
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((23.52966 20.52394)(28.28920
    22.87291)(25.77673 27.59659)(22.34778 24.69332)(23.52966 20.52394)))))
(:region regionD
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((49.99606 18.74888)(54.37759
    24.80803)(52.85137 25.66706)(49.60897 24.57518)(49.99606 18.74888)))))
(:region regionE
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((59.35168 78.26495)(57.61885
    83.77747)(52.45846 80.30299)(56.94561 76.10759)(59.35168 78.26495)))))

(:region refuel-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition (and (max-distance ((?x1 ?y1) (?x2 ?y2)) :d 2)))

;; Activities
(:durative-action fly-tanker
 :duration (and (>= ?duration 0.1) (<= ?duration 2000))
 :condition (and (at start (can-start))
            (over all (inside (mission-region (xt) (yt)))))
:effect (and (at start (not (can-start)))
             (at start (mission-ongoing))
             (at end (not (mission-ongoing)))
             (at start (tanker-flying))
             (at end (not (tanker-flying)))
             (increase (xt) (* (vx-t) #t))
             (increase (yt) (* (vy-t) #t))))

(:durative-action fly-uav
:duration (and (>= ?duration 0.1) (<= ?duration 2000))
:condition (and (at start (mission-ongoing))
                (at start (uav-canfly))
                (over all (inside (mission-region (xb) (yb))))
                (over all (>= (bb) 0)))
:effect (and (at start (not (uav-canfly)))
             (at start (uav-flying))
             (at end (not (uav-flying)))
             (increase (xb) (* (vx-b) #t))
             (increase (yb) (* (vy-b) #t))
             (decrease (bb) (* 0.1 (norm-sq (vel-uav)) #t))
```

```
            (decrease (bb) (* 1.1 (norm (vel-uav)) #t))))

(:durative-action fly-uav2
:duration (and (>= ?duration 0.1) (<= ?duration 2000))
:condition (and (at start (mission-ongoing))
                (at start (uav2-canfly))
                (over all (inside (mission-region (xb2) (yb2))))
                (over all (>= (bb2) 0)))
:effect (and (at start (not (uav2-canfly)))
             (at start (uav2-flying))
             (at end (not (uav2-flying)))
             (increase (xb2) (* (vx-b2) #t))
             (increase (yb2) (* (vy-b2) #t))
             (decrease (bb2) (* 0.1 (norm-sq (vel-uav2)) #t))
             (decrease (bb2) (* 1.1 (norm (vel-uav2)) #t))))

(:durative-action refuel-uav
:duration (and (>= ?duration 0.5) (<= ?duration 20))
:condition (and (over all (tanker-flying))
                (over all (uav-flying))
                (over all (<= (bb) 100))
                (at start (uav-available))
                (over all (inside (refuel-range (xt) (yt) (xb) (yb)))))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (increase (bb) (* (bat-recharge-rt) #t))))

(:durative-action refuel-uav2
:duration (and (>= ?duration 0.5) (<= ?duration 20))
:condition (and
            (over all (tanker-flying))
            (over all (uav-flying))
            (over all (<= (bb2) 100))
            (at start (uav2-available))
            (over all (inside (refuel-range (xt) (yt) (xb2) (yb2)))))
:effect (and
         (at start (not (uav2-available)))
         (at end (uav2-available))
         (increase (bb2) (* (bat-recharge-rt) #t))))

(:durative-action arrive-airport
 :duration (and (>= ?duration 2) (<= ?duration 2))
 :condition (and (at start (mission-ongoing))
                 (at start (uav-flying))
                 (at start (uav2-flying))
                 (at start (uav-available))
                 (at start (uav2-available))
                 (over all (inside (end-region (xt) (yt))))
                 (over all (inside (end-region (xb) (yb))))
                 (over all (inside (end-region (xb2) (yb2)))))
:effect (and (at end (arrived))
             (at start (not (mission-ongoing)))))

(:durative-action take-photoA
:duration (and (>= ?duration 15) (<= ?duration 15))
```

```
:condition (and (over all (mission-ongoing))
                (over all (uav-flying))
                (over all (inside (regionA (xb) (yb))))
                (at end (inside (regionA (xb) (yb))))
                (at start (uav-available)))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (at end (photo-takenA))))

(:durative-action take-photoA2
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav2-flying))
                (over all (inside (regionA (xb2) (yb2))))
                (at end (inside (regionA (xb2) (yb2))))
                (at start (uav2-available)))
:effect (and (at start (not (uav2-available)))
             (at end (uav2-available))
             (at end (photo-takenA))))

(:durative-action take-photoB
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav-flying))
                (over all (inside (regionB (xb) (yb))))
                (at end (inside (regionB (xb) (yb))))
                (at start (uav-available)))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (at end (photo-takenB))))

(:durative-action take-photoB2
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav2-flying))
                (over all (inside (regionB (xb2) (yb2))))
                (at end (inside (regionB (xb2) (yb2))))
                (at start (uav2-available)))
:effect (and (at start (not (uav2-available)))
             (at end (uav2-available))
             (at end (photo-takenB))))

(:durative-action take-photoC
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav-flying))
                (over all (inside (regionC (xb) (yb))))
                (at end (inside (regionC (xb) (yb))))
                (at start (uav-available)))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (at end (photo-takenC))))

(:durative-action take-photoC2
:duration (and (>= ?duration 15) (<= ?duration 15))
```

```
:condition (and (over all (mission-ongoing))
                (over all (uav2-flying))
                (over all (inside (regionC (xb2) (yb2))))
                (at end (inside (regionC (xb2) (yb2))))
                (at start (uav2-available)))
:effect (and (at start (not (uav2-available)))
             (at end (uav2-available))
             (at end (photo-takenC))))

(:durative-action take-photoD
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav-flying))
                (over all (inside (regionD (xb) (yb))))
                (at end (inside (regionD (xb) (yb))))
                (at start (uav-available)))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (at end (photo-takenD))))

(:durative-action take-photoD2
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav2-flying))
                (over all (inside (regionD (xb2) (yb2))))
                (at end (inside (regionD (xb2) (yb2))))
                (at start (uav2-available)))
:effect (and (at start (not (uav2-available)))
             (at end (uav2-available))
             (at end (photo-takenD))))

(:durative-action take-photoE
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav-flying))
                (over all (inside (regionE (xb) (yb))))
                (at end (inside (regionE (xb) (yb))))
                (at start (uav-available)))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (at end (photo-takenE))))

(:durative-action take-photoE2
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav2-flying))
                (over all (inside (regionE (xb2) (yb2))))
                (at end (inside (regionE (xb2) (yb2))))
                (at start (uav2-available)))
:effect (and (at start (not (uav2-available)))
             (at end (uav2-available))
             (at end (photo-takenE)))))
```

Problem (`onair15-problem.pddl`):

```
(define (problem onair-problem-1)
```

```
 (:domain onair-refuel-1)
 (:init (can-start)
        (uav-canfly)(uav-available)
        (uav2-canfly)(uav2-available)
        (= (xt) 70.0)(= (yt) 10.0)
        (= (xb) 70.0)(= (yb) 10.0)
        (= (xb2) 70.0)(= (yb2) 10.0)
        (= (bb) 100)(= (bb2) 100))
 (:goal (and
         (photo-takenA)(photo-takenB)
         (photo-takenC)(photo-takenD)
         (photo-takenE)
          (arrived))))
(:metric minimize (+ (* 5 (total-time) )
                     (* 20 (norm (vel-tanker)))))
```

References

Bajada, J., Fox, M., & Long, D. (2015). Temporal Planning with Semantic Attachment of Non-Linear Monotonic Continuous Behaviours. In *IJCAI 2015, Proceedings of the 24th International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, July 25-31, 2015*.

Benton, J., Coles, A. J., & Coles, A. (2012). Temporal Planning with Preferences and Time-Dependent Continuous Costs.. *ICAPS 2012*.

Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1), 281–300.

Bogomolov, S., Magazzeni, D., Minopoli, S., & Wehrle, M. (2015). PDDL+ Planning with Hybrid Automata: Foundations of Translating Must Behavior. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pp. 42–46.

Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, New York, NY, USA.

Bryce, D., Gao, S., Musliner, D. J., & Goldman, R. P. (2015). SMT-Based Nonlinear PDDL+ Planning.. *AAAI-17*, 3247–3253.

Cambon, S., Alami, R., & Gravot, F. (2009). A Hybrid Approach to Intricate Motion, Manipulation and Task Planning. *The International Journal of Robotics Research*, *28*(1), 104–126.

Camilli, R., Nomikou, P., Escartin, J., Ridao, P., Mallios, A., Kilias, S. P., Argyraki, A., Andreani, M., Ballu, V., Campos, R., Deplus, C., Gabsi, T., Garcia, R., Gracias, N., Hurtos, N., Magi, L., Mevel, C., Moreira, M., Palomeras, N., Pot, O., Ribas, D., Ruzie, L., & Sakellariou, D. (2015). The Kallisti Limnes, carbon dioxide-accumulating subsea pools. *Scientific Reports*, *5*(1), srep12152.

Cashmore, M., Fox, M., Larkworthy, T., Long, D., & Magazzeni, D. (2014). AUV mission control via temporal planning. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 6535–6541. IEEE.

Cashmore, M., Fox, M., Long, D., & Magazzeni, D. (2016). A Compilation of the Full PDDL+ Language into SMT. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, pp. 79–87.

Coles, A. J., Coles, A., Fox, M., & Long, D. (2010). Forward-Chaining Partial-Order Planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, pp. 42–49.

Coles, A. J., Coles, A., Fox, M., & Long, D. (2012). COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research (JAIR)*, *44*, 1–96.

Coles, A., Fox, M., Long, D., & Smith, A. (2008). Planning with Problems Requiring Temporal Coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pp. 892–897.

Della Penna, G., Magazzeni, D., Mercorio, F., & Intrigila, B. (2009). UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*.

Do, M. B., & Kambhampati, S. (2003). Sapa: A Multi-objective Metric Temporal Planner. *J Artif Intell Res(JAIR)*, *20*, 155–194.

Eyerich, P., Matmüller, R., & Röger, G. (2009). Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*.

Fallon, M., Kuindersma, S., Karumanchi, S., & Tedrake, R. (2015). An Architecture for Online Affordance-based Perception and Whole-body Planning. *Journal of Field ...*.

Fernandez-Gonzalez, E. (2017). *Generative Multi-Robot Task and Motion Planning Over Long Horizons*. Ph.D. thesis, Massachusetts Institute of Technology.

Fernandez-Gonzalez, E., Karpas, E., & Williams, B. C. (2015). Mixed Discrete-Continuous Heuristic Generative Planning Based on Flow Tubes. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 1565–1572.

Fernandez-Gonzalez, E., Karpas, E., & Williams, B. C. (2017). Mixed Discrete-Continuous Planning with Convex Optimization. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 1–7.

Fox, M., & Long, D. (2003). PDDL2. 1: An Extension to PDDL for Expressing Temporal Planning Domains.. *J Artif Intell Res(JAIR)*.

Fox, M., & Long, D. (2006). Modelling Mixed Discrete-Continuous Domains for Planning.. *J Artif Intell Res(JAIR)*.

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2014). FFRob: An efficient heuristic for task and motion planning. *lis.csail.mit.edu*.

Gerevini, A. E., Haslum, P., Long, D., Saetti, A., & Dimopoulos, Y. (2009). Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, *173*(5-6), 619–668.

Hoffmann, J. (2003). The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *J Artif Intell Res(JAIR)*, *20*, 291–341.

Hoffmann, J., & Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. *J Artif Intell Res(JAIR)*, *14*, 253–302.

Hofmann, A., & Williams, B. C. (2006). Exploiting spatial and temporal flexibility for plan execution of hybrid, under-actuated systems. *AAAI 2006*.

Hofmann, A., & Williams, B. C. (2015). Temporally and spatially flexible plan execution for dynamic hybrid systems. *Artificial Intelligence*, pp. –.

Jensen, J. L. W. V. (1906). Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, *30*(0), 175–193.

Kautz, H. A., & Selman, B. (1999). Unifying SAT-based and Graph-based Planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pp. 318–325.

Li, H., & Williams, B. C. (2011). Hybrid Planning with Temporally Extended Goals for Sustainable Ocean Observing. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*.

Li, H. X. (2010). *Kongming: a generative planner for hybrid systems with temporally extended goals*. Ph.D. thesis, Massachusetts Institute of Technology.

Li, H. X., & Williams, B. C. (2008). Generative Planning for Hybrid Systems Based on Flow Tubes. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, pp. 206–213.

Long, D., & Fox, M. (2003). Exploiting a Graphplan Framework in Temporal Planning. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, pp. 52–61.

Lozano-Pérez, T., & Kaelbling, L. P. (2014). A constraint-based method for solving sequential manipulation planning problems. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 3684–3691. IEEE.

Pantke, F., Edelkamp, S., & Herzog, O. (2016). Symbolic discrete-time planning with continuous numeric action parameters for agent-controlled processes. *Mechatronics*, *34*, 38–62.

Piacentini, C., Alimisis, V., Fox, M., & Long, D. (2013). *Combining a temporal planner with an external solver for the power balancing problem in an electricity network*. Twenty-Third International ....

Piotrowski, W. M., Fox, M., Long, D., Magazzeni, D., & Mercorio, F. (2016). Heuristic Planning for PDDL+ Domains. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 3213–3219.

Richter, S., & Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*.

Savas, E., Fox, M., Long, D., & Magazzeni, D. (2016). Planning Using Actions with Control Parameters. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, pp. 1185–1193.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S. J., & Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pp. 639–646. IEEE.

Stefik, M. (1981). Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, *16*(2), 111–139.

Toussaint, M. (2015). Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 1930–1936.

Wolfman, S. A., & Weld, D. S. (1999). The LPSAT Engine & Its Application to Resource Planning.. *IJCAI 2016*.