HAL
open science

Cyber-Physical Planning: Deliberation for Hybrid Systems with a Continuous Numeric State

Arthur Bit-Monnot, Luca Pulina, Armando Tacchella

To cite this version:

Arthur Bit-Monnot, Luca Pulina, Armando Tacchella. Cyber-Physical Planning: Deliberation for Hybrid Systems with a Continuous Numeric State. International Conference on Automated Planning and Scheduling (ICAPS), Jun 2018, Delft, Netherlands. hal-02931996

HAL Id: hal-02931996

https://laas.hal.science/hal-02931996v1

Submitted on 7 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Cyber-Physical Planning: Deliberation for Hybrid Systems with a Continuous Numeric State

Arthur Bit-Monnot and **Luca Pulina**
University of Sassari, Sassari, Italy
{afbit, lpulina}@uniss.it

Armando Tacchella
University of Genoa, Genoa, Italy
armando.tacchella@unige.it

Abstract

Cyber-physical systems pose unique deliberation challenges, where complex strategies must be autonomously derived and executed in the physical world, relying on continuous state representations and subject to safety and security constraints. Robots are a typical example of cyber-physical systems where high-level decisions must be reconciled with motion-level decisions in order to provide guarantees on the validity and efficiency of the plan.

In this work we propose techniques to refine a high-level plan into a continuous state trajectory. The refinement is done by translating a high-level plan into a nonlinear optimization problem with constraints that can encode the intrinsic limitations and dynamics of the system as well as the rules for its continuous control. The refinement process either succeeds or yields an explanation that we exploit to refine the search space of a domain-independent task planner. We evaluate our approach on existing PDDL+ benchmarks and on a more realistic and challenging rover navigation problem.

1 Introduction

The research field of model-based deliberation has given rise to different strands, depending on whether they target a general view of intelligence (in AI) or systems with a physical embodiment (as in Robotics). This separation of concerns has enabled the development of a wealth of efficient dedicated methods targeting each field's specific problems but that have been only loosely integrated. We are today reaching the limits of what can be achieved through this separation of concerns in terms of efficiency, scalability and safety of complex *autonomous* Cyber-Physical Systems (CPS) (Rajan and Saffiotti 2017).

Indeed, research in AI and more specifically in planning and scheduling has been mostly concerned with the high-level discrete decisions that must be taken into account to build a successful and high-quality plan while abstracting away most of the lower level details of the system. On the other hand, decision problems for CPS have to deal with the physics associated with the state of the system and are often deeply concerned with the metric and temporal environment in which the system evolves.

Motivation Consider a planetary exploration rover composed of a mobile base and a robotic arm with an end effector allowing for object manipulation. A rover can perform of a variety of tasks including: collecting a soil sample in a given area of the map, taking a picture of a landmark and collecting a previously identified rock at given coordinates. The rover's batteries are continuously discharging and can be recharged in some predefined areas. To achieve its objectives, a rover is required to navigate in a mapped environment while avoiding obstacles and ensuring that its battery level remains above a safety threshold.

At the most abstract level, the problem requires formulating a (discrete) plan that allocates and schedules tasks to the rover so that all high-level goals are fulfilled. Such a high-level plan must eventually be refined through lower-level decisions, especially regarding the trajectory of the rover. Such a refined plan must be such that all motions implied by the high-level actions have a corresponding trajectory that is feasible with respect to the robot's kinematics and dynamics. Furthermore, the trajectories chosen should not impact the validity of the imposed properties of the plan by, e.g., causing a deadline to be missed or a rover to run out of battery.

Of course, it might be the case that a high-level plan cannot be refined to a low level trajectory, because of the discrepancies between the high-level model and actual continuous dynamics of the environment.

Outline Our objective in this paper is to tackle the problem of mixed discrete-continuous planning in which an autonomous agent must reconcile the discrete, task-level, decisions with the need to eventually refine them down to a continuous trajectory that can be leveraged by a low-level controller. In particular, we are interested in problems that require fine-grained control of the continuous state evolution as it typically occurs in navigation problem.

We first introduce a general method for refining discrete decisions into continuous trajectories with fine-grained control. We build for this on nonlinear optimization techniques as a mean to fulfill a set of constraints on the continuous state evolution. Second, we detail how this refinement process can be integrated with domain-independent task planners where a task level plan is refined into a continuous trajectory and feedback is provided as a result of failure to do so.

2 Running Example

As a concise running example, we adapt the *1-dimensional car problem* by Fox and Long (2006). The continuous state of a car is a tuple $(d, v) \in \mathbb{R}^2$ where d is the distance to the origin and v is the velocity. The car can be in two operating modes, denoted by a boolean state variable *running* that is true iff the engine is turned on. The driver can control the acceleration, denoted by a control parameter $a \in [-1, 1]$, which has no effect when engine is turned off. The continuous state evolution is given by the following differential equations:

$$\dot{d} = v \qquad \dot{v} = \begin{cases} a, & \text{if } \textit{running} = \textit{true} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

In addition, the velocity should be null whenever the engine is off (i.e. $\textit{running} \vee v = 0$).

3 Definitions

Definition 1 (Hybrid Domain)**.** A hybrid domain Σ is a tuple $(\mathcal{S}, \mathcal{V}_x, \mathcal{V}_u, \textit{Flow}, \mathcal{A}, \textit{Trans}, \textit{Guards}, \textit{Inv})$ where:

- $\mathcal{S}$ is a finite set of discrete states
- $\mathcal{V}_x = \{x_1, \ldots, x_n\}$ is a set of continuous state variables. We denote as $X = \mathbb{R}^n$ the set of possible assignments.
- $\mathcal{V}_u = \{u_1, \ldots, u_m\}$ is a set of control variables. Let us denotes as $U = \mathbb{R}^m$ the set of possible assignments.
- $\textit{Flow} : \mathcal{S} \times X \times U \to \dot{X}$ is the flow function that gives the evolution of continuous state variables in a discrete state ($s \in \mathcal{S}$) under a given control ($u \in U$).
- $\mathcal{A}$ is a finite set of symbols denoting actions
- $\textit{Trans} : \mathcal{A} \times \mathcal{S} \to \mathcal{S}$ denotes the transition function that gives the discrete state resulting in the applying an action in a given discrete state.
- $\textit{Guards} : \mathcal{A} \to 2^{\mathcal{S} \times X}$, the guards, give the set of possible assignments to continuous variables for a transition to occur in a given discrete state.
- $\textit{Inv} : \mathcal{S} \to 2^{X \times U}$, the invariants, gives the set of allowed assignments to continuous and control variables in a given discrete state $s \in \mathcal{S}$.

Our running example is translated into a hybrid domain with *(i)* $\mathcal{S}$ composed of two discrete states corresponding to ($\textit{running} = \textit{true}$) and ($\textit{running} = \textit{false}$), *(ii)* two continuous state variables $\mathcal{V}_x = \{d, v\}$, and *(iii)* one control variable $\mathcal{V}_u = \{a\}$. The flow function is given by Eq. (1). $\mathcal{A}$ contains two instantaneous discrete actions that respectively turn-on and turn-off the engine. The corresponding effects and conditions are encoded in transitions *Trans* and guards *Guards*.

Definition 2 (Hybrid Problem)**.** A hybrid problem is a tuple $\mathcal{H} = (\Sigma, s_0, x_0, G)$ where:

- Σ is a hybrid domain,
- $s_0 \in \mathcal{S}$ is the initial discrete state,
- $x_0 \in X$ is the initial continuous state,
- $G \subseteq 2^{\mathcal{S} \times X}$, the goal, is the set of accepting final states.

Given a sequence of instantaneous events (*happenings*), we use a *hybrid time domain* to refer to the (absolute) occurrence time of each happening.

Definition 3 (Hybrid Time Domain)**.** A *hybrid time domain* with an *horizon* $T \in \mathbb{R}_{\geq 0}$ and n happenings is given by a sequence of times $\langle t_0, t_1, \ldots, t_{n+1} \rangle$ such that $t_0 = 0$, $t_{n+1} = T$ and $t_i \leq t_{i+1}$.

Definition 4. A *plan* Π over a hybrid time domain Φ_Π with horizon T and n happenings is a tuple $(\pi_\Pi, \Phi_\Pi, u_\Pi)$ where:

- $\pi_\Pi = \langle a_1, \ldots, a_n \rangle$ is a sequence of actions such that $a_i \in \mathcal{A}$ is the discrete action to take at the ith happening.
- $\Phi_\Pi = \langle t_0, \ldots, t_{n+1} \rangle$, the hybrid time domain, acts as a schedule by associating each action a_i to a timestamp t_i.
- $u_\Pi(t) : [0, T] \to U$ is the control to apply at time t.

For a given hybrid problem and plan, the discrete state immediately after each happening i is given by $s_i = \textit{Trans}(a_i, s_{i-1})$. The continuous state at time $t \in [t_j, t_{j+1}]$ derives from the activity function: $\dot{x}(t) = \textit{Flow}(s_i, x(t), u(t))$. Fixing $x(0) = x_0$ as given in the problem definition allows a complete simulation of the evolution of $x(t)$ over $[0, T]$.

Definition 5 (Solution)**.** A plan Π with n happenings is a solution to a hybrid problem if and only if:

- $\forall i \in \{1..n\} : (s_{i-1}, x(t_i)) \in \textit{Guards}(a_i)$, i.e., the guards of the ith action are satisfied.
- $\forall i \in \{0..n\}, t \in [t_i, t_{i+1}] : (x(t), u(t)) \in \textit{Inv}(s_i)$, i.e., the invariants associated to a discrete state s_i hold.
- $(s_n, x(T)) \in G$, where T is the temporal horizon of Π and G is the goal set, i.e., the last state is an accepting state.

4 Decision Procedure for the Refinement Problem

Let us first focus on the problem of refining a discrete plan (i.e., a sequence of instantaneous actions) into a control strategy and the associated continuous state trajectory.

Definition 6 (Refinement problem)**.** Given a hybrid problem $\mathcal{H}$ and a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$, the *refinement problem* is the one of finding:

- a hybrid time domain $\Phi = \langle t_0, \ldots, t_{n+1} \rangle$, and
- a control function $u(t)$ defined over $[t_0, t_{n+1}]$

such that the plan (Φ, π, u) is a solution to $\mathcal{H}$.

In this section, we detail a decision procedure for the refinement problem. We defer the treatment of the generation of discrete state sequences to Section 5.

4.1 Plan Refinement as Nonlinear Optimization

Given the computational cost induced by nonlinear constraint satisfaction problems (Nocedal and Wright 1999), we consider the use of unconstrained nonlinear optimization techniques for the generation of continuous state trajectories. In this process, we build on the insights of (Rösmann, Hoffmann, and Bertram 2017) that used a similar, though more specialized, approach in the context of trajectory optimization for robotic navigation.

Least-square optimization problem Let $\boldsymbol{b}$ be a real valued vector representing the variables of a system. In an optimization problem the objective is to find a vector $\boldsymbol{b}^*$ that minimizes an objective function O:

$$\boldsymbol{b}^* = \arg\min_{\boldsymbol{b}} O(\boldsymbol{b}) \tag{2}$$

An optimization problem is said to be a *least-square optimization problem* if the objective function is of the form:

$$O(\boldsymbol{b}) = \sum_k r_k(\boldsymbol{b})^2 \tag{3}$$

where each $r_k(\boldsymbol{b})$ term is called a *residual* and typically represents an individual error that should be minimized in a solution $\boldsymbol{b}^*$. We are further interested in *nonlinear* least-square optimization problems each r_k can take an arbitrary form.

4.2 Refinement as Nonlinear Least Squares

Let us define $Q = X \times U \times \mathbb{R}^+$ such that a vector $q \in Q$ is composed of an assignment $x(q) \in X$ to continuous state variables $\mathcal{V}_x$, an assignment $u(q) \in U$ to control variables $\mathcal{V}_u$ and a temporal delay $\delta t(q) \in \mathbb{R}^+$.

Definition 7 (Refinement)**.** The *refinement* of a given discrete state trajectory with n happening is a trajectory $\mathcal{T}$:

$$\mathcal{T} = \langle \boldsymbol{q}_0^0, q_1^0, \ldots, q_{m_0}^0, \boldsymbol{q}_0^1, q_1^1, \ldots, q_{m_n}^n, \boldsymbol{q}_0^{n+1} \rangle$$

such that:

- each happening i is associated to a state $\boldsymbol{q}_0^i \in Q$,
- between two happenings i, $i+1$ the state evolution is given by a sequence of m_i continuous states $\langle q_1^i, \ldots, q_{m_i}^i \rangle$
- for each $q \in \mathcal{T}$, $\delta t(q) \in \mathbb{R}^+$ denotes the time elapsed until the next state in $\mathcal{T}$.

In a preliminary step, let us assume that the "shape" of the solution trajectory $\mathcal{T}$ is known, that is, for each happening i we are given the number m_i of intermediate states until the next happening.

Decision variables Determining an actual trajectory for $\mathcal{T}$ boils down to giving, for each $q \in \mathcal{T}$ a value $f(q) \in \mathbb{R}$ to each variable $f \in Q$. We can thus define the set of real valued decision variables $V_{\mathcal{T}}$ as:

$$V_{\mathcal{T}} = \bigcup_{q \in \mathcal{T}} \{f(q) : f \in Q\}$$

The input vector of the optimization problem can be obtained by laying out variables of $V_{\mathcal{T}}$ into a vector $\boldsymbol{b}$. E.g. $\boldsymbol{b} = [d(q_0), v(q_0), a(q_0), \delta t(q_0), d(q_1), v(q_1), \ldots]$ for our running example.

From constraints to errors The set of constraints $C_{\mathcal{T}}$ that must hold for a trajectory $\mathcal{T}$ to be valid is obtained by taking the conjunction of the following constraints that respectively enforce (1) invariants, (2) guards, (3) flow constraints, (4) start conditions, and (5) goal conditions:

1	$(x(q_j^i), u(q_j^i)) \in \mathit{Inv}(s_i)$	$\forall i, j$
2	$(s_{i-1}, x(q_0^i)) \in \mathit{Guards}(a_i)$	$\forall i \in \{1..n\}$
3	$\frac{x(q_{j+1}^i) - x(q_j^i)}{\delta t(q_j^i)} = \mathit{Flow}(s_i, x(q_j^i), u(q_j^i))$	$\forall i, j < m_i$
4	$x(q_0^0) = x_0$	–
5	$(s_n, x(q_0^{n+1})) \in G$	–

At this point, let us note that each constraint in $C_{\mathcal{T}}$ can be expressed solely as a function of the decision variables $V_{\mathcal{T}}$ and thus of $\boldsymbol{b}$. Indeed, the a_i and s_i terms of the original constraints are fixed and known from the definition of the refinement problem. By systematically substituting them, one can express each constraint in $C_{\mathcal{T}}$ as a boolean function $c_k(\boldsymbol{b})$.

To complete our mapping to least-square optimization, we must now express each constraint $c_k(\boldsymbol{b})$ as a real valued residual term $r_k(\boldsymbol{b})$ such that $r_k(\boldsymbol{b}) = 0$ implies $c_k(\boldsymbol{b}) = \mathit{true}$

We do this with a recursive transformation $e(\cdot)$ that maps a boolean expression into a real valued one. $e(\cdot)$ is defined recursively by the following transformations rules, where a, b and c are boolean expressions, x and y are real valued expressions and $\mathit{ite}(a, b, c)$ is a conditional expression that evaluates to b if $a = \mathit{true}$ and to c otherwise.

Original term	Rewritten term
$e(\mathit{true})$	0
$e(false)$	∞
$e(a \vee b)$	$\min\{\lvert e(a)\rvert, \lvert e(b)\rvert\}$
$e(a \wedge b)$	$\lvert e(a)\rvert + \lvert e(b)\rvert$
$e(x = y)$	$y - x$
$e(x \leq y)$	$\max\{x - y, 0\}$
$e(x < y)$	$\max\{x - y - \epsilon, 0\}$
$e(ite(a, b, c))$	$ite(a, e(b), e(c))$

It is easy to verify that $e(a) = 0$ iff $a = \mathit{true}$ and we can define $r_k(\boldsymbol{b}) \stackrel{\text{def}}{=} e(c_k(\boldsymbol{b}))$. The objective function is defined as the squared sum of all residuals, thus defining a least-square minimization problem:

$$\boldsymbol{b}^* = \arg\min_{\boldsymbol{b}} O_{\mathcal{T}}(\boldsymbol{b})$$

$$O_{\mathcal{T}}(\boldsymbol{b}) = \sum_k r_k(\boldsymbol{b})^2$$

4.3 Solution of the Optimization Problem

Recall that our optimization problem is defined for a particular solution shape $\mathcal{T}$. The solution we are looking for is thus a pair $(\mathcal{T}^*, \boldsymbol{b}^*)$ such that $O_{\mathcal{T}^*}(\boldsymbol{b}^*) = 0$.

Solution Approximation Finding such a solution is typically computationally very expensive and we instead propose to look for an approximate solution $(\hat{\mathcal{T}}, \hat{\boldsymbol{b}})$ where each residual is bounded above by a finite small quantity $e_{\max}$, the maximal error tolerated on a constraint violation.

Also important for defining an approximate solution is the choice of the time discretization, i.e., the time elapsed between two consecutive states in $\hat{\mathcal{T}}$ (for which we use the notation $\delta t(\cdot)$). A bound $\delta t_{\max}$ on this temporal gap can be expressed directly as a constraint $\forall q \in \mathcal{T} : \delta t(q) \leq \delta t_{\max}$.

Provided $e_{\max}$ and $\delta t_{\max}$, an approximate solution is a pair $(\hat{\mathcal{T}}, \hat{\boldsymbol{b}})$ such that:

$$\begin{aligned} &\forall k, r_k(\hat{\boldsymbol{b}}) \leq e_{\max} \\ &\forall q \in \hat{\mathcal{T}}, \delta t(q) \leq \delta t_{\max} \end{aligned} \tag{4}$$

Resolution Scheme Before diving into the resolution process let us first give an intuition of how numeric solvers based on some form of gradient descent would proceed in minimizing a single first-order constraint between two states in our model.

Example 1. Consider two states q_1 and q_2 related by the following constraint that imposes a velocity of 1:

$$\dot{d}(q_1) = \frac{d(q_2) - d(q_1)}{\delta t(q_1)} = 1$$

and an incumbent solution $\boldsymbol{b}$ corresponding to the assignment:

$$d(q_1) \leftarrow 0 \qquad d(q_2) \leftarrow 1 \qquad \delta t(q_1) \leftarrow 0.1$$

The residual at $\boldsymbol{b}$ would be $1 - \frac{1-0}{0.1} = -9$.

A solver relying on the gradient to decide how to update the solution would analyze the first order partial derivatives and conclude that the residual can be decreased by *(i)* decreasing $d(q_2)$, *(ii)* increasing $d(q_1)$ or *(iii)* increasing $\delta t(q_1)$. The solver would then compute an update to the incumbent solution that follows the descent direction indicated by the gradient. Here, the updated incumbent could, e.g., correspond to the following assignment:

$$d(q_1) \leftarrow 0.2 \qquad d(q_2) \leftarrow 0.8 \qquad \delta t(q_1) \leftarrow 0.5$$

corresponding to a velocity of 1.2 and a residual of -0.2. □

From example 1, it should be obvious that our formulation allows the solver to act both on the values of state variables as well as on the *temporal gap between states*. We exploit this property in our resolution procedure as follows. Starting with the smallest possible $\mathcal{T}$ containing one state for each happening and a randomly generated incumbent solution $\boldsymbol{b}$, we use a nonlinear least-square solver to minimize constraint violations, resulting a new incumbent $\boldsymbol{b}'$. $\boldsymbol{b}'$ can be though as a first, very rough, approximation of the final solution. Except in trivial cases, it will likely be the case that the $\delta t(\cdot) \leq \delta t_{\max}$ constraints cannot by satisfied without also violating the other constraints. In such a case, the solver would look for an intermediate solution with $\delta t(\cdot)$ terms above the threshold wherever this helps in minimizing the residuals of other constraints. We treat the presence of $\delta t(\cdot) > \delta t_{\max}$ in $\boldsymbol{b}'$ as evidence that the current solution shape $\mathcal{T}$ is too tight and needs to be extended to obtain a valid solution.

Scaling solution shapes

Definition 8 (Band)**.** For a given solution shape with n happenings:

$$\mathcal{T} = \langle \boldsymbol{q}_0^0, q_1^0, \ldots, q_{m_0}^0, \boldsymbol{q}_0^1, q_1^1, \ldots, q_{m_n}^n, \boldsymbol{q}_0^{n+1} \rangle,$$

we define the *band* of the ith happening as the sequence of $m_i + 1$ states between the ith happening and the next one:

$$\mathcal{B}_i = \langle \boldsymbol{q}_0^i, \ldots, q_{m_i}^i \rangle$$

Given an incumbent $\boldsymbol{b}$, let $\overline{\delta t_i}$ be the average value of $\delta t(q)$ for each $q \in \mathcal{B}_i$. Having $\overline{\delta t_i} > \delta t_{\max}$, indicates that the band is too tight while $\overline{\delta t_i} < \delta t_{\max}$ indicates that the band is too large. We define a new solution shape $\mathcal{T}'$ where the number of states in each band is given by m_i':

$$m_i' = m_i \times \min \left\{ \frac{\overline{\delta t_i}}{\delta t_{\max}}, 2 \right\}$$

The new incumbent solution $\boldsymbol{b}'$ extrapolates its values from $\boldsymbol{b}$. Namely, for all states $q' \in \mathcal{B}_i'$, $\delta t(q')$ is given a value of $\delta t_{\max} \times \frac{m_i'}{m_i}$. For each variable $f \in Q$, the value of $f(q')$ is linearly interpolated from the two enclosing states in B_i.

We later refer to this as the $\textsc{Scale}(\mathcal{T}, \boldsymbol{b})$ procedure that returns the new solution shape and associated incumbent solution $(\mathcal{T}', \boldsymbol{b}')$

Minimization A wealth of algorithmic approaches have been devised for solving nonlinear least-square optimization problems such as the Gauss-Newton and Levenberg-Marquardt (LM) algorithms (Nocedal and Wright 1999). We use the LM approach that has previously shown to be robust and efficient on similar problems in robotics (Rösmann et al. 2012). We here give short presentation of the LM method and refer to the reader to Nocedal and Wright (1999) for further information.

Defining as $r(\boldsymbol{b})$ the vector of individual residuals $[r_0(\boldsymbol{b}), r_1(\boldsymbol{b}), \ldots]^\top$, LM iteratively refine the incumbent solution as $\boldsymbol{b} \leftarrow \boldsymbol{b} + \Delta\boldsymbol{b}$ where the update $\Delta\boldsymbol{b}$ is obtained by solving the linear system:

$$(\boldsymbol{J}^\top \boldsymbol{J} + \lambda \boldsymbol{I}) \times \Delta\boldsymbol{b} = -\boldsymbol{J}^\top \times r(\boldsymbol{b}) \tag{5}$$

where $\boldsymbol{J}$ denotes the jacobian of the objective function at $\boldsymbol{b}$, $\boldsymbol{I}$ is the identity matrix and λ is a damping factor. In the case of least squares minimization problems, $\boldsymbol{J}$ can be computed by combining the individual partial derivatives of residuals $r_k(\boldsymbol{b})$. The damping factor λ is defined such that higher λ results in smaller $\Delta\boldsymbol{b}$. It is used by the LM algorithm to dynamically control the step size in the case of nonlinear surfaces. Provided with an initial estimate $\boldsymbol{b}$, the LM method will iteratively refine it with a $\Delta\boldsymbol{b}$ update until a stopping condition is met (e.g., no progress is made or the objective is below a target threshold).

The greatest cost of LM lies in solving the linear system of Eq. (5) that has a complexity of $\mathcal{O}(n^3)$. Let us observe that our model is defined such that a given incumbent r_k only refers to a small subset of the variables of the system. For an invariant constraint (e.g. $d(q) = 0$), those are limited to the variables of a single state q, while constraints involving derivatives might only refer to the variables of two adjacent states. The direct consequence is that the jacobian $\boldsymbol{J}$ is a sparse, banded matrix. This enables the use of sparse matrix algorithms for the resolution of Eq. (5), and notably of sparse Cholesky factorization (Davis 2006) that can provide orders of magnitude improvements for such problems and has notably be exploited in graph optimization frameworks (Rainer, Grisetti, and Konolige 2011).

We define the procedure $\textsc{Minimize}(\boldsymbol{b}, C)$ as repeatedly refining $\boldsymbol{b}$ by means of an LM iteration, where the residuals corresponds to the errors associated to a set of constraints of $C : \{e(c) \mid c \in C\}$. The procedure stops once a local minimum or a maximum number of iterations is reached, yielding a new incumbent solution $\boldsymbol{b}'$.

Resolution Procedure The complete resolution process is given in Alg. 1. The $\textsc{Solve}(C, \mathcal{T}, \boldsymbol{b}, k)$ procedure first attempt to minimize constraint violation within the current solution shape $\mathcal{T}$, resulting in a new incumbent solution $\boldsymbol{b}'$. If $\boldsymbol{b}'$ can be shown to be a solution (Eq. (4)), then the solver returns the solution $(\mathcal{T}, \boldsymbol{b}')$. Otherwise, if the maximum number of iterations has been reached, the solver stops. When more iterations are allowed, the current solution will be scaled before repeating the process. We start from the smallest solution shape $\mathcal{T}_0$ containing a single state in each band and a randomly drawn initial incumbent $\boldsymbol{b}_0$.

Algorithm 1 Decision procedure for a set of constraints C, a solution shape T, a current solution estimate $\boldsymbol{b}$ and a current step k.

```
procedure SOLVE(C, T, b, k)
    b' ← MINIMIZE(b, C)
    if is-solution(b', C) then
        return (T, b')
    else if k = k_max then
        return Failure
    else
        (T_next, b_next) ← SCALE(T, b')
        return SOLVE(C, T_next, b_next, k + 1)
    end if
end procedure
```

Exploiting Constraint Orders We now detail how one can exploit structural differences in constraints to consider a small set of the simplest constraints in the first iterations and iteratively extend this set towards more complex constraints until the solution is valid with respect to all constraints of the problem.

Let us define a hierarchy between variables corresponding to the order of their associated derivatives. The order $\ell(x)$ of a state or control variable $x \in \mathcal{V}_x \cup \mathcal{V}_u$ is defined as $\ell(x) = 1 + \max_y l(y)$, where $y \in \mathcal{V}_x$ is a continuous variable such that the variable x appears in the right hand side of a flow constraint $\dot{y} = e$. Applying this definition to the car domain would result in, $\ell(d) = 1$, $\ell(v) = 2$ and $\ell(a) = 3$.

Definition 9 (Constraint Order)**.** We define the order $\ell(c)$ of a constraint c as $\ell(x)$ if c is a flow constraint of the form $\dot{x} = e$ and 0 otherwise.

This notion of order directly relates to the order of a derivative. A state invariant $d(q) \geq 0$ would have the order 0 while a constraint involving a first derivative (e.g. $\dot{d} = v$) would have an order of 1.

Let C_ℓ be the subset of constraints in $C_\mathcal{T}$ whose order is lesser than or equal to ℓ. In Algorithm 2, we redefine the resolution procedure as $\textsc{DeepeningSolve}(\ell, \mathcal{T}, \boldsymbol{b})$. For a given order ℓ, it invokes our previous SOLVE procedure but limit its scope to constraints in C_ℓ. Once a solution is found for ℓ, the solver continues at order $\ell + 1$ until an order $\ell_{\max}$ is reached such that $C_{\ell_{\max}} = C_\mathcal{T}$.

This scheme as the advantage of restricting its focus on the simplest constraints when searching for a solution. This is for instance beneficial when invoking $\textsc{DeepeningSolve}(0, \mathcal{T}_0, \boldsymbol{b}_0)$ in which the first action of the solver would be to transform the randomly drawn incumbent solution $\boldsymbol{b}_0$ into one where all state invariants hold before proceeding to satisfy first or second derivative constraints.

Algorithm 2 Alternative decision procedure that considers subsets of constraints of increasing order.

```
procedure DEEPENINGSOLVE(ℓ, T, b)
    res ← SOLVE(C_ℓ, T, b, 0)
    if res = Failure then
        return Failure
    else
        (T̂, b̂) ← res
        if C_ℓ = C_T then
            return (T̂, b̂)
        else
            return DEEPENINGSOLVE(ℓ + 1, T̂, b̂)
        end if
    end if
end procedure
```

5 Mixed Discrete and Continuous Planning

In section 4, we focused on refining a sequence of discrete actions into a continuous state trajectory and the associated control function. In this section, we outline how such a discrete plan can be computed by domain-independent planners and how we exploit failures of the refinement process to guide the search for alternative discrete plans.

5.1 Discrete State Trajectories from Domain-Independent Planners

Definition 10 (Classical planning problem)**.** A *classsical planning problem* is a tuple $(\mathcal{S}, \mathcal{A}, \mathit{Trans}, \mathcal{P}, s_0, \mathcal{S}_G)$ where $\mathcal{S}$ is a finite set of discrete states; $\mathcal{A}$ is a set of symbols denoting actions; $\mathit{Trans} : \mathcal{A} \times \mathcal{S} \to \mathcal{S}$, is the state-transition function; $\mathcal{P} : \mathcal{A} \to 2^\mathcal{S}$, the preconditions, associates each action to the set of states in which it is applicable; $s_0 \in \mathcal{S}$ is the initial state; and $\mathcal{S}_G \subseteq \mathcal{S}$ is the set of goal states.

From this definition, let us first note that $\mathcal{S}$, $\mathcal{A}$, *Trans* and s_0 map one-to-one with the eponimous terms in hybrid problems (Def. 1 and 2). To extract a classical planning problem as a discrete abstraction of a hybrid problem, we must only define how preconditions and goals relate to guards and accepting states, their counterparts in hybrid problems.

From $G \in 2^{\mathcal{S} \times X}$, the accepting states of a hybrid problem, we can define $\mathcal{S}_G$ as the projection of G on $2^\mathcal{S}$. Similarly, given $\mathit{Guards}(a) \in 2^{\mathcal{S} \times X}$ the guards of an action a, we can define $\mathcal{P}(a)$ as the projection of $\mathit{Guards}(a)$ on $2^\mathcal{S}$.

For practical purposes, it is sound to overapproximate $\mathcal{P}$ and $\mathcal{S}_G$ since the original guards and accepting states will be checked in the refinement process. For instance, if the guards of the hybrid domain are defined as conjunctions of boolean formulas over discrete and continuous state variables, it is sound to keep only these conjuncts that exclusively refer to discrete state variables when building the precondition formulas of $\mathcal{P}$.

This simple definition allows us to derive a discrete abstraction of a hybrid problem $\mathcal{H}$ in the form of a classical planning problem. The solution to a classical planning problem being a sequence of actions, one can *(i)* leverage domain-independent task planners to produce a discrete plan π, and *(ii)* use the decision procedure of the previous section to refine π into a solution to the original hybrid problem.

5.2 No-good Extraction

The naïve scheme for solving hybrid problems would be to have a discrete planner generate causally valid plans until one can be refined to a valid continuous trajectory. This indeed constitutes the basis of our resolution strategy and we now detail an extension to exploit the refinement failures to prune the search space of the discrete planner.

After a refinement failure, let us denote as Z the set of violated constraints in the last incumbent solution. We say that two constraints c_1 and c_2 are adjacent if they share a common variable. Let $\{Z_1, \ldots, Z_k\}$ be the connected components of Z under the adjacency relation. Each Z_j denotes a set of constraints that contribute to making the refinement problem unsatisfiable.

For the purpose of no-good extraction, we limit our analysis to Z_j such that all constraints in it are part of a single band $\mathcal{B}_i$. Z_j constitutes an invariant: it must be the case that these constraints are not all active in the same band.

Recall that each band $\mathcal{B}_i$ is associated to a unique discrete state s_i. Let us define the *enabling assignment* of a constraint c as the partial assignment $ea(c)$ to state variables in $\mathcal{S}$ that led to impose the constraint c. We find this enabling assignment by intersecting the full discrete state assignment s_i with the state variables of $\mathcal{S}$ present in in the original constraint formulation. E.g. if a constraint *running* $\vee$ $(v = 0)$ is violated in a state corresponding to the assigment $\{running \leftarrow \textit{false}, \textit{crashed} \leftarrow \textit{false}\}$, its enabling assignment would be $\neg$*running*.

Avoiding the conflict to be repeated requires that at least one of the constraints in Z_j is not enabled, which corresponds to the proposition $p_j = \bigvee_{c \in Z_j} \neg ea(c)$. Upon failure of the refinement procedure, we return each such proposition p_j. The discrete solver can use it to discard from its search space any state in which a p_j does not hold.

6 Experimental Evaluation

We here present an experimental evaluation of our planner (HYDRA), first with a comparison to PDDL+ planners and then on a more involved domain representing the navigation of a planetary exploration rover.

HYDRA uses LCP (Bit-Monnot 2018), an SMT-based domain-independent task planner, as our discrete solver. We exploit the incremental solving capability of the underlying SMT solver to encode no-goods.

The refinement solver is configured to allow a maximum of 30 scaling steps each leading to at most 30 iterations of the LM method. $\delta t_{\max}$ is set to 0.1 on all attempted problems while the maximum residual error, $e_{\max}$, is set to 10^{-4}.

Domains are modeled in a specific language that closely mirrors the definition of the hybrid domain and whose presentation is beyond the scope of this paper.

6.1 Comparison with PDDL+ Solvers

Our planning model differs from PDDL+ in its support of continuous control. For instance, PDDL+ models of the car domain encodes the control of the acceleration through two discrete actions *accelerate* and *decelerate* that respectively increase and decreases the acceleration by a fixed amount of one. Distance, velocity and acceleration are all part of the state of the system. This model has the consequence that the flow of the system between two discrete actions is fixed as control change is only allowed in discrete actions.

We compare our performance with DiNo (Piotrowski et al. 2016) and ENHSP (Scala et al. 2016a) on the car and descent domains from DiNo's benchmarks. The car domain is the one used as our running example with an additional constraint to model the drag when reaching high velocities. It is provided with 10 instances in wich the maximum allowed acceleration is increased.

Powered descent models a powered spacecraft landing on a celestial body. The vehicle gains velocity due to the force of gravity. The available action is to fire the thrusters of the spacecraft to decrease its velocity by burning fuel which in turn reduces its total mass. The thrust duration is limited by the amount of propellant available. We use the 20 original instances that differ by the distance that must be covered before landing.

Results are given in Table 1. It can be seen that HYDRA generally outperforms DiNo with overall shorter runtimes. On the solution quality (makespan), both solver provided similar results with none strictly dominating the other. Interesting to note is that on the car problem, HYDRA is not affected by the increase in allowed velocity. This in fact makes sense: for DiNo allowing larger accelerations increases its state space while for HYDRA it only relaxes some constraints on the state trajectory.

ENHSP scales better than DiNo on the car problem with runtimes within a factor 2 of HYDRA's. It however fails to solve any of the descent problem with $\delta t_{\max} = 0.1$. ENHSP's performance appeared to be very dependent on the choice of the temporal delta: with $\delta t_{\max} = 1$ it would solve all problems within a handful of seconds.

SMTPlan+ (Cashmore et al. 2016) is not included in the comparison as it timed out on all instances. Note that the domains include non-polynomials functions that are not fully supported by SMTPlan+.

We could not evaluate the efficiency of no-good learning on these benchmarks since for both domains, the second discrete solution was always refinable.

6.2 Rover Navigation

PDDL+ domains from DiNo' set of benchmarks are limited to a single dimension with limited control capabilities. To further assess our planner, we introduce a more challenging problem of planning rover activities where the rover has to move in 2D space subject to placement and orientation constraints (for manipulation and obstacle avoidance) as well as kinodynamic constraints on velocity and acceleration. The

	Car			Descent (1–10)			Descent (11–20)		
	DiNo	ENHSP	HYDRA	DiNo	ENHSP	HYDRA	DiNo	ENHSP	HYDRA
1	**3.18**	5.93	3.75	**1.49**	–	3.43	12.62	–	**5.47**
2	13.81	6.64	**3.0**	**2.30**	–	3.7	69.60	–	**6.78**
3	30.70	6.12	**3.11**	4.43	–	**3.3**	18.54	–	**7.5**
4	51.49	6.39	**2.9**	11.44	–	**4.11**	62.36	–	**7.45**
5	84.77	5.31	**3.14**	9.17	–	**4.51**	63.64	–	**9.69**
6	96.27	7.67	**2.87**	6.94	–	**4.48**	78.91	–	**10.43**
7	128.17	7.11	**2.98**	9.96	–	**4.52**	**3.67**	–	6.39
8	162.02	6.79	**3.18**	9.93	–	**4.55**	–	–	**34.93**
9	206.03	6.77	**2.76**	65.09	–	**4.56**	–	–	**8.2**
10	216.75	7.33	**2.92**	3.80	–	**4.62**	–	–	**7.75**

Table 1: Runtimes in seconds for DiNo, ENHSP and HYDRA for the 10 car domains and the 20 planetary descent domains. – indicates a timeout after 30 minutes. All runtimes were obtained on an Intel Core i5-7200U @ 2.50GHz. Best result is given in bold font.

problem requires to decide the course of action of a planetary rover that must take pictures of particular landmarks, collect rocks and sample soil in predefined areas.

The continuous state of a rover is a tuple (x, y, θ, v) where $(x, y) \in \mathbb{R}^2$ denotes its coordinates in a 2D euclidian space, $\theta \in \mathbb{S}$ is its orientation and $v \in \mathbb{R}$ is its forward velocity. The car is controlled by two parameters: $a \in \mathbb{R}$ being its acceleration and $s \in \mathbb{S}$ the steering angle of the front wheels. Using the models for nonholonomic wheeled vehicles (Laumond, Sekhavat, and Lamiraux 2006), the state evolution of the rover is given by the following differential equations:

$$\dot{x} = v \times cos(\theta)$$
$$\dot{y} = v \times sin(\theta)$$
$$\dot{\theta} = v \times tan(s)/L$$
$$\dot{v} = a$$

where L is the distance between the front and rear wheels.

The rover is further subject to velocity, acceleration and kinematic constraints. Constraints for obstacle avoidance and presence in an area are encoded as inequalities on the distance to an *implicit surface* (Gomes et al. 2009). Formulation of these constraints are omitted for the sake of space but can be easily adapted from existing formulations in the robotics literature on trajectory optimization (Rösmann, Hoffmann, and Bertram 2017).

The discrete part of the domain contains actions *pick*, *take-picture*, and *sample-soil*. The first two actions impose distance and orientation requirements to the rover's pose with respect to a landmark in the map while *sample-soil* requires the rover to be in a given, rather large, area. All actions thus restrict the position of the rover but leave some freedom that can be leveraged to optimize the global trajectory in the map.

We experimented on 10 variations of the problem with up to six tasks. It was relatively easy for LCP to find an initial discrete plan which it always provided in less than a second.

We hand tuned four of the problems so that the most obvious solution at the discrete level had no valid refinement. On these four domains, no-good acquisition proved useful, reducing the total number of failures from 12 to 5 as a result of forbidding different ordering of the same plan. In all problems of this domain that emphasizes trajectory planning, the bottleneck appeared to be in the plan refinement part.

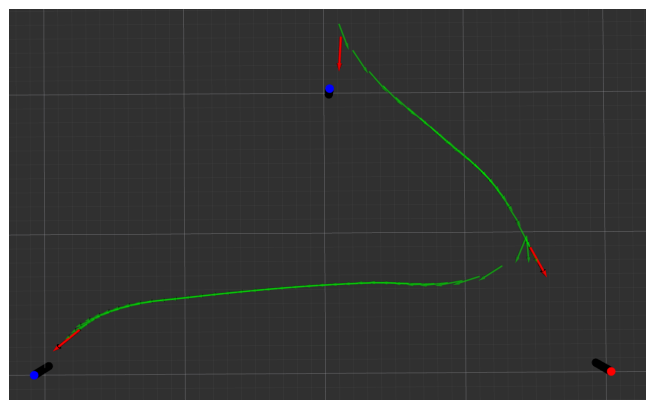

Figure 1: A kinematically feasible sequence of trajectories for a rover with car-like dynamics. The task level plan has the rover starting in a given position (top), drive to take a picture of the red object and drive to a position from which it can pick the blue object (bottom left). Red arrows denote the position of the rover at happenings of the plan while green arrows represent intermediate states.

During refinement of these problems, the average number of states in the incumbent trajectory was 837 and corresponds to an average runtime of 4 millisecond per LM iterations. Total time for a refinement tentative is in the order of the second, with more time spent to failed attempts as the solver exhausts its budget before abandoning.

An example trajectory in given in Figure 1. It should be noted that intermediate poses are not fixed but only subject to distance and orientation constraints with respect to landmarks of the map. This is illustrated by the choice of the position to take a picture that is automatically chosen to be close to the origin and destination and such that the maneuver to go to the next waypoint is kept efficient (note that the rover moves backward while turning before advancing toward the last target pose).

Most existing PDDL+ planners were unable to even approach the problem: the absence of support for trigonometric functions in planners prevented a realistic encoding of even the simplest robot motion models (e.g. the Dubins car (LaValle 2006)) in PDDL+. While ENHSP supports trigonometric functions, it failed to solve the simplest point-to-point navigation problem with no obstacles for the Dubins car. Our understanding is that the heuristic used by ENHSP fails to capture the complex relationship between the orientation θ of the rover and its (x, y) coordinates when evaluating the distance to the goal.

7 Related Work

Our definition of a hybrid domain closely relates to *hybrid automata* (Kowalewski et al. 2009) but differs by explicitly considering the control variables $\mathcal{V}_u$. In hybrid automata, the control variables are embedded in the discrete states which fully define the control mode (making the number of possible controls finite). Here the set of possible controls is potentially infinite (but is subject to invariants) and can be changed without a discrete state transition. Our definition also places some restrictions with respect to the usual definition of a hybrid automaton, namely by only allowing for deterministic state transitions and forbidding jumps on continuous variables.

Support for continuous numeric change in AI planning has primarily been done by integrating numeric decision procedures in a forward search setting. COLIN (Coles et al. 2012) support linear continuous change by integrating forward search with a linear programming solver. Similarly, ScottyActivity (Fernández-González, Williams, and Karpas 2018) is still limited to linear continuous change but additionally supports convex quadratic constraints by relying on a Second Order Cone Program (SCOP). An important limitation of ScottyActivity is the convexity requirement on constraints that prevents, e.g., the encoding of obstacles in robotic navigation problems. ENHSP (Scala et al. 2016a) relies on an eager time-discretization combined with elaborated heuristics to explore its search space in a forward manner. This schemes allows ENHSP to support a richer set of mathematical functions (and notably transcendental functions).

PDDL+ (Fox and Long 2006) allows the definition of processes that describe the continuous evolution of a numeric state and is now supported by several planners (Cashmore et al. 2016; Bryce et al. 2015; Piotrowski et al. 2016). One limitation with respect to our objective is that processes conveniently model the dynamics of the system but fails short when trying to specify the operating limits of a controller. Again fined-grained control could in theory be obtained by multiplying the number of actions, but one would quickly hit the scalability limits of current planners. The work of Scala et al. (2016b) is a notable exception in its capability to handle the long actions sequence that are necessary for trajectory planning, but imposes important limitation on the feasible trajectories and has no support for time which prevents its use for constraining the state evolution beside simple invariants.

Work in motion planning has seen steady progress with now well established methods and tools (LaValle 2006) with the last ten years seeing important interest in the problem of joint *ask and Motion Planning (TAMP)*. However, most of the work in TAMP has focused on geometrically demanding manipulation problems with a single agent, no support for time and minimal interactions regarding task-level requirements as reviewed by Lagriffoul (2016). An interesting approach is taken by Lagriffoul and Andres (2016) on identifying the cause of an infeasible plan refinement. Dantam et al. (2016) exploit similar insights to feed an SMT solver with additional clauses when solving joint task and motion planning problems. Mansouri and Pecora (2014) take the different approach of directly encoding geometric requirements into a constraint-based planner with dedicated propagation techniques. While relevant, work on refining a task level plan into kinematically feasible trajectories, typically decouples the task and trajectory planing parts and assumes *a priori* known target poses (Pecora et al. 2018). For our purpose, an inherent limitation is that these techniques are highly specialized to find collision free trajectories which is only a subpart of the problem.

8 Discussion and Conclusion

We explored the refinement of task-level plans into continuous state trajectories through nonlinear optimization techniques and provided an integration with domain-independant task planners through no-good acquisition.

The refinement technique is very general and makes no assumption on the structure of the numeric state, with constraints allowing to encode, e.g., invariants, flow, object placement or kinematics. The refinement of a task-level plan allows continuous control over the state trajectory which we demonstrated on trajectory planning for a rover with differential constraints.

An important area for future work is to support the exploration of distinctive topologies in the continuous state space. In the current resolution process, the refinement process could stop in a local optimum in which avoidable constraint violations remains. While this limitation did not manifest itself in the attempted problems, it could force the domain modeler to explicitly drive the search of the discrete plan toward distinct continuous trajectories. Even though very specialized to their particular problems, we believe the work in robotics and motion planning would constitute a good entry point for tackling this problem, notably with the notion of homotopic paths that captures most of our requirements (Schmitzberger et al. 2002; Jaillet and Simeon 2008).

Acknowledgements

The research of Arthur Bit-Monnot and Luca Pulina has been funded by the EU Commission's H2020 Program under grant agreement N.732105 (CERBERO project). The research of Luca Pulina has been also partially funded by the Sardinian Regional Project PROSSIMO (POR FESR 2014/20-ASSE I) and the FitOptiVis (ID: 783162) project.

References

Bit-Monnot, A. 2018. A Constraint-based Encoding for Domain-Independent Temporal Planning. In *International Conference on Principles and Practice of Constraint Programming (CP)*.

Bryce, D.; Gao, S.; Musliner, D.; and Goldman, R. 2015. SMT-based Nonlinear PDDL+ Planning. In *AAAI Conference on Artificial Intelligence*.

Cashmore, M.; Fox, M.; Long, D.; and Magazzeni, D. 2016. A Compilation of the Full PDDL+ Language into SMT. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: Planning with Continuous Linear Numeric Change. *Journal of Artificial Intelligence Research (JAIR)* 44.

Dantam, N. T.; Kingston, Z. K.; Chaudhuri, S.; and Kavraki, L. E. 2016. Incremental Task and Motion Planning: A Constraint-Based Approach. In *Robotics: Science and Systems Conference*.

Davis, T. A. 2006. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia.

Fernández-González, E.; Williams, B. C.; and Karpas, E. 2018. ScottyActivity: Mixed Discrete-Continuous Planning with Convex Optimization. *Journal of Artificial Intelligence Research (JAIR)* 62.

Fox, M., and Long, D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *Journal of Artificial Intelligence Research (JAIR)* 27.

Gomes, A.; Voiculescu, I.; Jorge, J.; Wyvill, B.; and Galbraith, C. 2009. *Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms*. Springer-Verlag London.

Jaillet, L., and Simeon, T. 2008. Path deformation roadmaps: Compact graphs with useful cycles for motion planning. *International Journal of Robotics Research (IJRR)* 27(11-12).

Kowalewski, S.; Garavello, M.; Guéguen, H.; Herberich, G.; Langerak, R.; Piccoli, B.; Polderman, J. W.; and Weise, C. 2009. *Handbook of Hybrid Systems Control: Theory, Tools, Applications*. Cambridge University Press.

Lagriffoul, F., and Andres, B. 2016. Combining task and motion planning: A culprit detection problem. *International Journal of Robotics Research (IJRR)*.

Lagriffoul, F. 2016. On Benchmarks for Combined Task and Motion Planning. In *RSS Workshop on Task and Motion Planning*.

Laumond, J. P.; Sekhavat, S.; and Lamiraux, F. 2006. Guidelines in nonholonomic motion planning for mobile robots. In *Robot Motion Planning and Control*.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

Mansouri, M., and Pecora, F. 2014. More Knowledge on the Table: Planning with Space, Time and Resources for Robots. In *International Conference on Robotics and Automation (ICRA)*.

Nocedal, J., and Wright, S. J. 1999. *Numerical Optimization*. Springer Series in Operations Research.

Pecora, F.; Andreasson, H.; Mansouri, M.; and Petkov, V. 2018. A Loosely-Coupled Approach for Multi-Robot Coordination, Motion Planning and Control. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Piotrowski, W.; Fox, M.; Long, D.; Magazzeni, D.; and Mercorio, F. 2016. Heuristic planning for PDDL+ domains. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Rainer, K.; Grisetti, G.; and Konolige, K. 2011. g2o : A General Framework for Graph Optimization. In *International Conference on Robotics and Automation (ICRA)*.

Rajan, K., and Saffiotti, A. 2017. Towards a science of integrated AI and Robotics. *AI* 247.

Rösmann, C.; Feiten, W.; Wösch, T.; Hoffmann, F.; and Bertram, T. 2012. Trajectory modification considering dynamic constraints of autonomous robots. In *German Conference on Robotics (Robotik)*.

Rösmann, C.; Hoffmann, F.; and Bertram, T. 2017. Integrated online trajectory planning and optimization in distinctive topologies. *Robotics and Autonomous Systems* 88.

Scala, E.; Haslum, P.; Thiebaux, S.; and Ramirez, M. 2016a. Interval-based relaxation for general numeric planning. In *European Conference on Artificial Intelligence (ECAI)*.

Scala, E.; Ramirez, M.; Haslum, P.; and Thiebaux, S. 2016b. Numeric Planning with Disjunctive Global Constraints via SMT. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Schmitzberger, E.; Bouchet, J.; Dufaut, M.; Wolf, D.; and Husson, R. 2002. Capture of homotopy classes with probabilistic road map. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.