# A Planning Based Framework for Controlling Hybrid Systems

**Johannes Löhr, Patrick Eyerich, Thomas Keller and Bernhard Nebel**
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Georges-Köhler-Allee 52
79110 Freiburg, Germany
{loehr, eyerich, tkeller, nebel}@informatik.uni-freiburg.de

## Abstract

The control of dynamic systems, which aims to minimize the deviation of state variables from reference values in a continuous state space, is a central domain of cybernetics and control theory. The objective of action planning is to find feasible state trajectories in a discrete state space from an initial state to a state satisfying the goal conditions, which in principle addresses the same issue on a more abstract level. We combine these approaches to switch between dynamic system characteristics on the fly, and to generate control input sequences that affect both discrete and continuous state variables. Our approach (called Domain Predictive Control) is applicable to hybrid systems with linear dynamics and discretizable inputs.

## Introduction

Where is the need for planning in cybernetics? Modeling the real world by the use of differential equations is the base for controlling dynamic systems. Modern control methods like Model Predictive Control (Rawlings and Mayne 2009) can deal with real world restrictions like saturation of input signals or constraints on state variables. However, it is still an issue to control systems with reconfigurable dynamics (e.g mode switches) or logical dependencies between input signals or state transitions, aspects that are a central part of action planning. The intersection of planning and control was part of previous research (Anthony 1981) that utilizes a separate control system to reach set points, which are commanded by a planning system. Autonomous system reconfigurations of space engines were put into practice by Williams and Nayak (1996). The main contribution of this paper is to show how planning can be used to generate input signals directly for systems with numerical and logical states (known as hybrid systems) taking their dynamics and reconfigurations into account.

This research is motivated by the need of autonomy of spacecraft, which have to reach their goals in the absence of permanent communication links and in the presence of failures and uncertainties. Spacecraft can be modeled as hybrid dynamic systems with continuous and logical state variables. Continuous state variables (like position and velocity or attitude and angular rate) can be affected by actuators, e.g., thrusters, or reaction-wheels, and measured by sensors like star tracker or gyroscopes. Logical state variables (like information about the currently active equipment, available equipment or identified failures) are usually changed manually or via simple rules triggered in failure cases. The main idea of this paper is to utilize planning algorithms that take into account information about the system and its environment to autonomously find *feasible state transitions* from the initial hybrid state to a desired goal state. Using planning techniques to control specific dynamic systems has been of interest recently, e.g., in rover dynamics (Della Penna et al. 2010) or in the efficient usage of battery packs (Fox, Long, and Magazzeni 2011). Rather than focusing on a specific application explicitly, we show how to tackle such problems from a general point of view.

The remainder of the paper is structured as follows. First, we briefly characterize the state space description and control problem of continuous dynamic systems and extend it to our purpose. Then, we introduce a framework to control hybrid systems by planning and show how to derive a suitable planning domain from the dynamics and input signals via discretization. Furthermore, necessary planner requirements and modifications are formulated. We apply our approach on exemplary dynamic systems with additional constraints in the subsequent section. Finally, we discuss the results and give an outlook on future work.

## Continuous and Hybrid Systems

A *dynamic system* is a model of the real physical processes (called *plants*) based on a set of linear differential equations

$$\dot{\mathbf{x}}_n(t) = A\,\mathbf{x}_n(t) + B\,\mathbf{u}_n(t), \tag{1}$$

where $\mathbf{x}_n \in \mathbb{R}^p$ is the vector of *state variables*,[1] $\mathbf{u}_n \in \mathbb{R}^m$ is the vector of *input signals*, $A$ is the *dynamic matrix* of dimension $p \times p$, and $B$ is the *input matrix* of dimension $p \times m$. Using a vector based description rather than scalar equations makes it very easy to consider coupled dynamics, as the time derivative of any state variable can be influenced by all other variables in $\mathbf{x}_n$.

*Feedforward control* of dynamic systems generates input sequences

$$\mathbf{u}_n(t),\ t \in [t_a, t_b]$$



[1] We use the subscript $n$ to tag numerical state variable vectors and the subscript $l$ to tag logical state variable vectors.

guiding the system from its initial state $\mathbf{x}_n(t_a)$ to a desired *set point* $\mathbf{x}_n(t_b)$, specified externally, within the time interval $[t_a, t_b]$.

A vector of *q measurements* $\mathbf{y}_n \in \mathbb{R}^q$, provided by sensors, is given by

$$\mathbf{y}_n(t) = C\,\mathbf{x}_n(t) + D\,\mathbf{u}_n(t), \tag{2}$$

where $C$ is the *measurement matrix* and $D$ is the *direct input-output matrix*.

Methods like *Optimal Control* (Geering 2007) or *Model Predictive Control* (Wang 2009) can be used to control dynamic systems. Optimal Control finds time– or cost–optimal input sequences $\mathbf{u}_n(t)$ by minimizing a cost function. In Model Predictive Control an optimal input subsequence $\mathbf{u}_n(t)$ with $t \in [t_a, t_a + t_c]$ is generated by evaluating the states $\mathbf{x}_n(t) \in [t_a, t_a + t_p]$ of a prediction model within a receding horizon $t_p$.

We refer to dynamic systems enriched with some logical state variables as *hybrid systems*. An overview on hybrid dynamical models is given by Heemels, Schutter, and Bemporad (2001). More concretely, the *state space matrices* $A$, $B$, $C$, and $D$ of the hybrid system are dependent on a vector $\mathbf{x}_l$ of Boolean variables, allowing for system reconfigurations (i.e., mode switches) which shall be exploited actively in the planning process. This leads to the state space description of the system:

$$\dot{\mathbf{x}}_n(t) = A(\mathbf{x}_l)\,\mathbf{x}_n(t) + B(\mathbf{x}_l)\,\mathbf{u}_n(t). \tag{3}$$

Analogously, the measurement Equation 2 is extended to

$$\mathbf{y}_n(t) = C(\mathbf{x}_l)\,\mathbf{x}_n(t) + D(\mathbf{x}_l)\,\mathbf{u}_n(t). \tag{4}$$

In the following, we refer to the vector of all the logical and numeric variables of a state as $\mathbf{x} := [\mathbf{x}_n, \mathbf{x}_l]^T$ and write shortly $s_i$ to denote the values that are taken by the state variables $\mathbf{x}$ at time point $t_i$.

## Domain Predictive Control

We present an architecture for planning based control of hybrid systems. The name *Domain Predictive Control* is chosen in the style of Model Predictive Control (Wang 2009), where control signals are generated by evaluation of model based state predictions. We use a similar approach based on domain models which are further abstractions of the dynamic model.

### Architecture

The first step towards autonomy is a suitable architecture which combines planning and hybrid systems, as shown in Figure 1. The planner uses an abstract domain model of the plant for predicting future states.[2] Based on this model and the knowledge of the current system state and the goal specification, the planner continuously generates input signals by replanning. This yields several advantages regarding robustness. If the plant is disturbed by non-modeled dynamics or

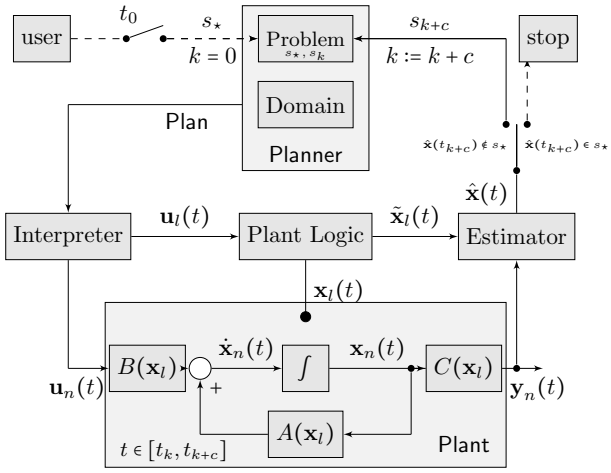


Figure 1: Architecture of Domain Predictive Control for hybrid systems based on continuous (re-) planning.

exogenous effects, there are discrepancies between the prediction and the real dynamics that tend to grow with the size of the prediction horizon. Continuous replanning reduces this effect since it is basically a control-feedback loop, taking into account new information about the system or the environment.

The user triggers the planner at time $t_0$ and provides a set of goals $s_\star$. The current estimated system state $\hat{\mathbf{x}}(t_0)$ is the initial state for the planner. The planner generates a sequence of actions $\langle a_1, \ldots, a_i, \ldots, a_c \rangle$ having durations $\delta_i$ that provides a numeric input signal $\mathbf{u}_n(t)$, where $t \in [t_k, t_{k+c}]$ and $t_{k+c} = t_k + \sum_{i=1}^{c} \delta_i$ and a logical input signal $\mathbf{u}_l(t)$. The parameter $c$ denotes the number of state transitions to perform before replanning is triggered if the goal was not reached.[3] The numeric input signal affects the numeric state variables $\mathbf{x}_n(t)$ as in Equation 3, whereas $\mathbf{u}_l(t)$ changes the logical state variables $\mathbf{x}_l(t)$ of the plant logic. As the state space matrices of the plant dynamics in Equations 3 and 4 are dependent on the logical state $\mathbf{x}_l$, system changes can actively be planned.

In general, it is necessary to estimate states since only measurements $\mathbf{y}_n$ are known in realistic applications. While numeric states have to be observed or estimated (Simon 2006), deviations in the logical state $\mathbf{x}_l$ from the expected logical state $\tilde{\mathbf{x}}_l$ (e.g., caused by failures) can be identified via Failure Detection and Isolation (Isermann 2006) or model-based diagnosis (Struss 1997) methods. As we focus on the planning aspects of the architecture in this paper, we assume full knowledge of the hybrid state vector such that $\hat{\mathbf{x}} = \mathbf{x}$.

The planner solves the new planning problem that is updated with the new hybrid state $s_{k+c} = \hat{\mathbf{x}}(t_{k+c})$. Furthermore, $k$ is set to $k+c$. This process is repeated until $s_{k+c} \in s_\star$, thereby generating a trajectory of $z$ state transitions, driven by $\mathbf{u}_n(t_k),\ k \in [0, 1, \ldots, z]$ control signals, from the initial state $s_0 = \hat{\mathbf{x}}(t_0)$ to a goal state $s_z = \hat{\mathbf{x}}(t_z) \in s_\star$.

[2] As the current information of the plant is required for prediction of future states in the planning process, it is implicitly assumed that the input $u_n(t)$ cannot affect the measurement $y_n(t)$ at the same time. Thus $D = 0$ in the system.

[3] Setting $c = 1$ achieves the best reactivity to exogenous events, but is also most costly from a computational point of view.

If the domain model is complex or the minimal plan length is high, the planner might fail to find a solution in reasonable time. To capture such cases we have implemented two termination criteria. The first is a receding planning horizon, limiting the search depth of the planner. The second is a limitation on the number of expanded states. If one of these criteria triggers, the planner generates a *promising trajectory* that either leads to a goal state or, if no such state has been reached, to a state that is nearest to the goal according to the heuristic information.

As mentioned, our architecture resembles the idea of Model Predictive Control. Instead of a dynamic model, we use a domain model for prediction and utilize a planner for optimization. In the following, we outline the generation process of such a domain model.

**Domain Model** In this section we show that the control problem of hybrid systems can be expressed in a planning language featuring basic mathematical operations. Let $\mathbf{x}_n = [v_1, v_2, ..., v_p]$ be the state variables vector consisting of numeric variables $v_n$ and $\mathbf{u}_n(t), t \in [t_k, t_k + \delta_i]$ be a continuous input vector signal which shall be modeled as an action. A finite set of such actions $\mathcal{O}$ describes the *domain model*. The effect $e$ of action $a_i$ on the numeric state $\mathbf{x}_n(t_k)$ can be predicted as

$$e(a_i) : \mathbf{x}_n(t_{k+1}) = \Phi_i \, \mathbf{x}_n(t_k) + \Psi_i \tag{5}$$

where $\Phi_i$ is the homogeneous solution, obtained by the matrix exponential function[4]

$$\Phi_i = e^{A\delta_i} \tag{6}$$

and $\Psi_i$ is the inhomogeneous solution of Equation 3, driven by the input signal $\mathbf{u}_n(t)$

$$\Psi_i = \int_{t_k}^{t_k+\delta_i} e^{A(t_k+\delta_i-\tau)} B \, \mathbf{u}_n(\tau) \, d\tau. \tag{7}$$

The state space matrices $A$ and $B$ are assumed to be constant within the duration of one action. Nevertheless, reconfigurations of the system can be planned on the fly since state space matrices can be changed by actions as well.

The advantage of our approach is that the quite complex computations to solve Equations 6 and 7 can be performed as a preprocessing step. Effects as in Equation 5 can be represented using only basic mathematical operations and computed by a numeric planner during the planning process.

Domains of this kind are similar to the class of switched, piecewise affine systems (Heemels, Schutter, and Bemporad 2001). A piecewise affine system (PWA) is given by

$$\mathbf{x}_n(k+1) = A_i \, \mathbf{x}_n(k) + B_i \, \mathbf{u}_n(k) + f_i, \; \mathbf{x}_n(k) \in \Omega_i.$$

In a domain model $A_i = \Phi_i$ and $B_i \, \mathbf{u}_n(k)$ is restricted to $\Psi_i$. The additional external offset $f_i$ can be either considered in Equation 5 or absorbed into the inhomogeneous solution. The system dynamics depends on the state $\mathbf{x}_n$ that corresponds to a subset $\Omega_i$ of the state space $\bigcup_{\forall i \in \mathcal{O}} \Omega_i \subseteq \mathbb{R}^n$.

[4]It is worth to mention that the matrix exponential can be approximated by the series $e^{A\delta} = \sum_{k=0}^{\infty} \frac{(A\delta)^k}{k!}$.

Note that the PWA constraint $\Omega_i \cap \Omega_j = \varnothing, \; \forall i \neq j$ needs not to be satisfied in a domain model, however. Instead, we explicitly allow for overlapping subsets $\Omega_i$ and use a planner to switch to the most suitable dynamics in order to guide the system into a state satisfying the goal conditions.

**Goal Specification** It is possible that the goal cannot be reached precisely due to the use of discretized effects. Therefore, we define the goal set as a conjunction of intervals $v_i = g_i \pm \epsilon_i$. Such intervals can easily be specified as two conjunctive goals: $v_i \leq g_i + \epsilon_i$ and $v_i \geq g_i - \epsilon_i$, where $\epsilon_i$ denotes the sufficient accuracy of the desired set point.

## Planning Formalism

The domain model is an action based description of the plant dynamics with all possible inputs and constraints. A general approach to handle the continuous dynamics of a system is to relax them to a discrete system, e.g., to a Finite State System (Della Penna et al. 2010) or a Final State Temporal System (Fox, Long, and Magazzeni 2011), and interleave between discretization and validation. Our approach also incorporates discretization techniques. However, currently we use a fixed discretization of time steps and plan to incorporate varying durations later.

The formalism we use to generate planning tasks is PDDL 2.1 level 3 (Fox and Long 2003), which means that we require the underlying planning system to support durative actions and numeric variables. Indeed, the generated tasks make heavy use of numeric variables and in general contain variables that do not change monotonically but can both increase and decrease, a feature early numeric planning systems are not able to cope with (Hoffmann 2002). Since, beside vector additions, we also need to perform matrix-vector multiplications, the requirements for planning system that can deal with such domains are quite high and pose some interesting challenges for the development of more sophisticated techniques to solve numeric planning tasks. Nevertheless, we show in the following that our approach is feasible and that we can achieve very good results using a state-of the art temporal numeric planner, Temporal Fast Downward (TFD) (Eyerich, Mattmüller, and Röger 2009), out of the box. With some minor modifications mainly regarding the heuristic, described in the following sub section, we are able to generate complex plans involving lots of numerical operations in nearly real time.

For the remainder of the paper we need some definitions. We borrow them from Eyerich, Mattmüller, and Röger (2009) and omit some of the details for the sake of simplicity. First, we define a *temporal numeric planning task* as a tuple $\Pi = \langle \mathcal{V}, s_0, s_\star, \mathcal{A}, \mathcal{O} \rangle$ with the following components: $\mathcal{V}$ is a set of *state variables* $v$, partitioned into a set $\mathcal{V}_l$ of *logical variables* with finite domains $\mathcal{D}_v$ and a set of *numeric variables* $\mathcal{V}_n$ with domains $\mathcal{D}_n = \mathbb{R} \cup \{\perp\}$ ($\perp$ for *undefined*). The initial state $s_0$ is given by a variable assignment (a *state*) over all fluents in $\mathcal{V}$ and the set of goal states is defined by a partial state $s_\star$ over $\mathcal{V}$. A *partial state* $s'$ is a state restricted to a subset of the fluents. $\mathcal{A}$ is the set of *axioms* and $\mathcal{O}$ is a finite set of *durative actions*. A durative action $\langle C, E, \delta \rangle$ consists of a triple $C = \langle C_\vdash, C_\leftrightarrow, C_\dashv \rangle$ of

partial variable assignments over $\mathcal{V}_l$ (called its *start*, *persistent*, and *end condition*, respectively), a tuple $E = \langle E_\vdash, E_\dashv \rangle$ of *start* and *end effects* and a duration variable $\delta \in \mathcal{V}_n$. $E_\vdash$ and $E_\dashv$ are finite sets of *conditional effects* $\langle c, e \rangle$. The *effect condition* $c = \langle c_\vdash, c_\leftrightarrow, c_\dashv \rangle$ is defined analogously to the operator condition $C$. A *simple effect* $e$ is either a logical effect of form $v = w$ or a numeric effect of form $v \circ v'$, where $v, v' \in \mathcal{V}_n$ and $\circ$ is one of the operators +=, −=, ∗=, /=, and :=, respectively.

A *time-stamped state* $\mathcal{S} = \langle t, s, E, C_\leftrightarrow, C_\dashv \rangle$ consists of a *time stamp* $t \geq 0$, a *state* $s$, a set $E$ of *scheduled effects*, and two sets $C_\leftrightarrow$ and $C_\dashv$ of persistent and end conditions. A scheduled effect $\langle \Delta t, c_\leftrightarrow, c_\dashv, e \rangle$ consists of the remaining time $\Delta t \geq 0$ (until the instant when the effect triggers), persistent and end effect conditions $c_\leftrightarrow$ and $c_\dashv$ over $\mathcal{V}$, and a simple effect $e$. The conditions in $C_\leftrightarrow$ and $C_\dashv$ are annotated with time increments $\Delta t \geq 0$ and have to hold until instant $t + \Delta t$ (exclusively) for persistent conditions and at instant $t + \Delta t$ for end conditions.

A *trajectory* $\mathcal{S} = \langle s_0, s_1, \ldots, s_z \rangle$ in a temporal planning task is a sequence of time-stamped states such that for $1 \leq i \leq z$ an intermediate state $s_i$ can be generated out of its predecessor state $s_{i-1}$ by either inserting an additional action $a$ starting at time stamp $(s_{i-1}.t)+\epsilon$, thereby applying all start effects of $a$ and scheduling its end effects, or by progressing time via an artificial *advance time operator* to the next time point on which a scheduled action $a$ ends (applying all end effects of $a$). A trajectory is *valid* if all the start, overall and end conditions of all involved actions are satisfied at their appropriate time. A valid trajectory is called a *plan* if $s_z$ complies with the goal specification $s_\star$.

**Planning** For solving the generated planning tasks we use Temporal Fast Downward (TFD) (Eyerich, Mattmüller, and Röger 2009). Out of the box, TFD works quite nicely for our purposes. In fact, it was the only planner of several candidates that was able to solve certain planning tasks generated by our approach. Other systems we have tried include LPG (Gerevini, Saetti, and Serina 2003), SAPA (Do and Kambhampati 2003), Crikey3 (Coles et al. 2008), and POPF2 (Coles et al. 2010; 2011). None of these systems was able to produce plans for our generated problems as they are not able to deal with numeric axioms, could not handle numeric variables that do not change in a monotone way but can both increase and decrease, did not support multiplication at all, or did just not terminate. However, despite making heavy use of complex numeric calculations and utilizing the power of numeric axioms, our produced planning tasks are legal PDDL 2.1 level 3 tasks of relatively moderate size, so it is very likely that other planning systems can be tweaked relatively easily to work on them.

It turned out that the incorporated heuristic used in TFD has two weaknesses that can lead to poor guidance in certain domains: First, our representation often requires the same action to be applied several times consecutively to reach the goal, a behavior that is not covered by TFD's standard heuristic, where each appearance of an action instance that changes a numeric variable is only counted once. Second, TFD's heuristic only checks for actions that can possibly change a numeric value in a way that seems to be promising to fulfill a yet unsatisfied comparison condition, thereby not distinguishing between several different actions able to do so. Furthermore, all our example applications share a very interesting property that seems to be present in a lot of practical problems as well: numeric variables change usually in a continuous way and the amount of change often corresponds to the cost of the actions producing that change. Therefore, when approximating the cost of satisfying a yet unsatisfied comparison condition, it is often sufficient to compare only the actual value of the appropriate numeric variables with their desired ones, and not consider the actions that might produce that change. Therefore, we implemented a new type of heuristic that sums up the differences between the desired values and the current values of unsatisfied comparison conditions.

The last modification we made is to enable TFD to terminate early based on different criteria like the elapsed time, the number of expanded states, and the maximal number of actions that can be applied sequentially, respectively. If one of these termination rules triggers, TFD returns the trajectory to the time-stamped state that seems to be most promising in terms of the heuristic estimation.

## Exemplary Applications

In this section we put Domain Predictive Control into practice, thereby showing two important aspects of it. The first is that we plan with and in excess of system transitions, shown in a fictive slew maneuver of a space telescope. The second is the incorporated replanning strategy that allows us to deal with large state spaces, shown in an exemplary ball maze. Both applications focus on the main advantages of planning in hybrid systems.

### Example I: Space Telescope Slew Maneuver

This example shows how to generate a planning model out of physical equations and how we can use planning to generate an input sequence that both performs state transitions and system transitions. The goal is to slew a space telescope from a certain attitude into another attitude in order to look at a new point in space. A common way to point to an arbitrary direction is to slew around a rotation axis with angle $\phi$, also known as Euler axis and Euler angle[5] (Wertz 1994). The challenge in this example is that the telescope has to zoom on the new object during the slew, in order to save time. Due to the movement of the heavy optical units orthogonally to the rotation axis, the satellites moment of inertia changes significantly.

**State Space Differential System** There are some basic physical equations which are necessary to derive the differential system. We start with the angular momentum, which is defined by

$$H = J\,\omega,$$

[5]This enables a non vectorial description of the physical equations, used for simplicity. Note that the moment of inertia changes in general, if the rotation axis changes.

where $J$ is the moment of inertia around the rotation axis and $\omega$ is the angular rate. Any change of the angular momentum $H$ needs an external torque $T$, given by

$$J\dot{\omega} = T.$$

The change of the angle is given by

$$\dot{\phi} = \omega,$$

which finally leads to the state space description of the dynamic system (see Equation 3)

$$\begin{bmatrix}\dot{\phi}\\ \dot{\omega}\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}\phi\\ \omega\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{J}\end{bmatrix} T \tag{8}$$

| | Precondition | Numerical Eff. | Logical Eff. |
|---|---|---|---|
| $a_0$ | - | $\Phi_0\ \mathbf{x}$ | - |
| $a_1$ | $\mathbf{x}_2 < 1\ \frac{\text{deg}}{\text{s}} \wedge \neg x_l)$ | $\Phi_1\ \mathbf{x} + \Psi_1$ | - |
| $a_2$ | $\mathbf{x}_2 < 1\ \frac{\text{deg}}{\text{s}} \wedge \neg x_l)$ | $\Phi_2\ \mathbf{x} + \Psi_2$ | - |
| $a_3$ | $\mathbf{x}_2 < 1\ \frac{\text{deg}}{\text{s}} \wedge x_l)$ | $\Phi_3\ \mathbf{x} + \Psi_3$ | - |
| $a_4$ | $\mathbf{x}_2 < 1\ \frac{\text{deg}}{\text{s}} \wedge x_l)$ | $\Phi_4\ \mathbf{x} + \Psi_4$ | - |
| $a_5$ | $\mathbf{x}_2 < 1\ \frac{\text{deg}}{\text{s}} \wedge \neg x_l)$ | $\Phi_5\ \mathbf{x}$ | $x_l := \texttt{true}$ |

Table 1: Actions of the domain based description of the hybrid system of Example I.

**Domain Model** The continuous state space system from Equation 8 has to be discretized in time and for several input sequences. The external torque $T$ is produced by thrusters, which act with a certain lever on the satellite's center of mass. They can either be switched on or switched off and provide a torque around the rotation axis of 1 Nm. A discretization time of 5 s using Equation 6 yields the homogeneous solution

$$\Phi_i = \begin{bmatrix}1 & 5\\ 0 & 1\end{bmatrix},\ i \in \{0, 1, ..., 4\}.$$

The moment of inertia $J$ is 1500 kg m$^2$ while the telescope is zoomed and 1000 kg m$^2$ while it is not zoomed. Therefore, the input matrix of the hybrid system is either

$$B_{x_l=0} = \begin{bmatrix}1\\ \frac{1}{1000}\end{bmatrix} \quad \text{or} \quad B_{x_l=1} = \begin{bmatrix}1\\ \frac{1}{1500}\end{bmatrix}$$

depending on the Boolean state variable $x_l$ which denotes whether the telescope is zoomed or not. Applying a torque in positive direction ($i = 1$) and negative direction ($i = 2$) with respect to the rotation axis, as in Equation 7 leads to the inhomogeneous solutions

$$\Psi_1 = \begin{bmatrix}0.0125\\ 0.005\end{bmatrix} \qquad \Psi_2 = \begin{bmatrix}-0.0125\\ -0.005\end{bmatrix}.$$

The inhomogeneous solutions for applying a torque in positive direction ($i = 3$) and negative direction ($i = 4$) after the zoom operation are

$$\Psi_3 = \begin{bmatrix}0.0083\\ 0.0033\end{bmatrix} \qquad \Psi_4 = \begin{bmatrix}-0.0083\\ -0.0033\end{bmatrix}$$

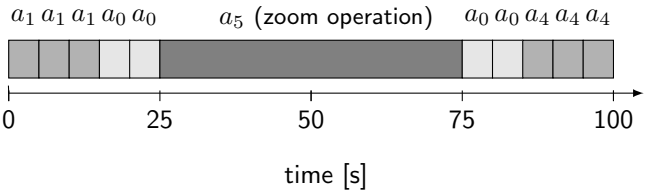


Figure 2: Plan from the initial state to the goal state. The total plan duration is 100 s.

since the moment of inertia has changed. If no thruster is fired, the transition of the states is affected solely by the homogeneous solution and $\Psi_0 = \mathbf{0}$ per definition. Finally, the effect of an action $i$ is given by

$$\mathbf{x}(t_{k+1}) = \Phi_i\ \mathbf{x}(t_k) + \Psi_i, \quad i \in [0, 4],$$

where $k$ indicates the number of previous actions. This approach leads to a set of actions, with preconditions shown in Table 1. The domain model is directly derived from the hybrid system.

**Zoom Transition** In this example, a system transition is needed to reach the goal state. Instantaneous mode switches are often sufficient to describe the change of the system – in this example a more complex, time variant transition is necessary, which itself interacts with the state variables. How the effect of the zoom operation ($a_5$) complying with Equation 5 is derived is shown in the following.

As mentioned above, the movement of the optical units changes the moment of inertia $\Delta J = J_{x_l=1} - J_{x_l=0}$ within the zoom time of $\delta_5$, which has a direct impact on the angular rate of the satellite. As it is prohibited to fire a thruster during the zoom operation, the angular momentum $H$ is constant. Let us assume that the moment of inertia changes linearly over $\delta_5$, then the angular rate is given by

$$\omega(t_k + t) = \frac{J_0}{(J_0 + \frac{\Delta J}{\delta_i} t)}\ \omega_k,$$

where $\omega_k$ is the angular rate before the zoom operation has started. The angle during the zoom operation is given by the integration of the angular rate

$$\phi(t) = \int^{\delta_5} \omega\ dt + \phi_k = \frac{\delta_5}{\Delta J} J_0 \ln(\frac{\Delta J}{\delta_5} t + J_0)\Big|^{\delta_5}\ \omega_k + \phi_k.$$

With concrete values, $\delta_5 = 50$ s and $\Delta J = 500$ kg m$^2$, the transition matrix of the zoom operation is given by

$$\Phi_5 = \begin{bmatrix}1 & 40.55\\ 0 & \frac{2}{3}\end{bmatrix}$$

with the logical effect $x_l := \texttt{true}$.

We chose this example to show how we can make solutions from more complex system equations applicable to our planning domain.

**Planning Problem** In this section, an exemplary instance of the domain is introduced. The planning problem is given by the initial hybrid state $s_0$ and the goal $s_\star$.

$$s_0 = \begin{bmatrix}0.463\\ 0\\ \texttt{false}\end{bmatrix} \qquad s_\star = \begin{bmatrix}1.483 \pm \epsilon_1\\ 0 \pm \epsilon_2\\ \texttt{true}\end{bmatrix}$$

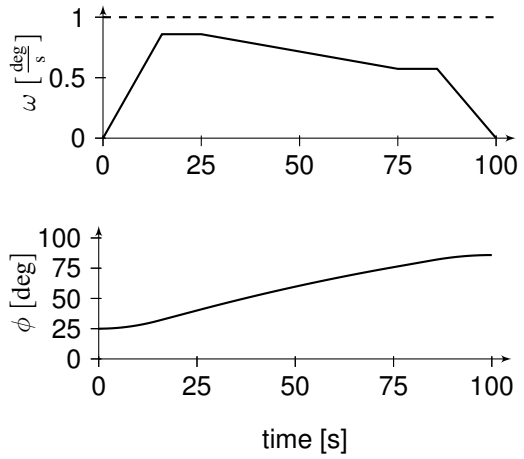


Figure 3: Graphs showing the angle (lower graph) and angular rate (upper graph) during the slew maneuver. The sensor constraint is shown as dashed line in the upper subfigure.

The initial state corresponds to an initial angle of 25 deg with an initial angular rate of 0 $\frac{\text{deg}}{\text{s}}$, where the telescope of the satellite is not zoomed. The desired attitude is 85 deg and the final angular rate 0 $\frac{\text{deg}}{\text{s}}$. The accuracy of the goal state is sufficient for $\epsilon_1 = 0.2$ deg and $\epsilon_2 = 0.02$ $\frac{\text{deg}}{\text{s}}$, since the satellite switches then into a closed loop fine pointing mode. Additionally, the sensors of the satellite require angular rates lower than 1 $\frac{\text{deg}}{\text{s}}$.

**Results** The planning problem (which is basically a control problem of hybrid systems) is difficult to model and to solve with classical methods in cybernetics. Task planning enables a straight forward, domain-based description of the hybrid system by the use of actions with preconditions and effects, as shown in Table 1. The planning problem can be solved by any planner that can deal with numerical variables, in our case TFD. The solution is a feasible state transition, as shown in Figure 2. The plan is numerically simulated in MATLAB. The resulting continuous profiles of the angle and angular rate are shown in Figure 3. The sensor constraint on the angular rate is marked as a dashed line. Finally, the attitude of the satellite during the slew maneuver is shown in the lower graph of Figure 3.

## Example II: Ball Maze

In our second example we use our approach to automatically guide a ball through a ball maze by inclining a plate around two axes. While this problem might look like a classical path planning problem at first sight, it is actually quite more complex as the trajectory of the ball cannot be influenced directly. Instead, we have to consider the differential coupling of velocity and accelerations. So, we search for a sequence of plate-inclinations that lets the ball roll to the goal area. Additionally, as the ball is not allowed to fall into one of the holes in the plate, the state space is strongly restricted, a circumstance that is challenging for classical control methods.

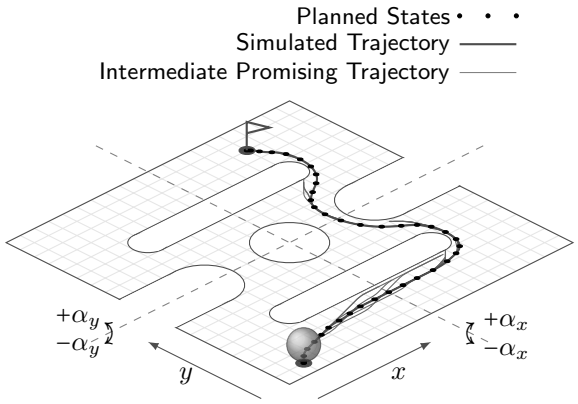


Figure 4: Ball maze with planned trajectory from the initial point to the desired set point.

**State Space Differential System** We model the ball as a frictionless mass on the plate. The plate can be inclined in both positive and negative directions by a discrete angle $\alpha$ on each axis. Depending on $\alpha$, the gravitational force acts on the ball by

$$F_x = m \cdot g \cdot \sin\alpha_x; \quad F_y = m \cdot g \cdot \sin\alpha_y. \tag{9}$$

The forces accelerate the mass by

$$\ddot{x} = \frac{F_x}{m} \qquad \ddot{y} = \frac{F_y}{m} \tag{10}$$

where the accelerations equal the time derivatives of the velocities $v_x$ and $v_y$

$$\ddot{x} = \dot{v}_x \qquad \ddot{y} = \dot{v}_y \tag{11}$$

and the velocities are the time derivatives of the $(x, y)$ position on the plate

$$\dot{x} = v_x \qquad \dot{y} = v_y. \tag{12}$$

Equations 10 to 12 yield the state space differential system

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v}_x \\ \dot{v}_y \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{m} & 0 \\ 0 & \frac{1}{m} \end{bmatrix} \begin{bmatrix} F_x \\ F_y \end{bmatrix}.$$

**Domain Model** Assuming the mass of the ball is 2 kg, $g = 9.81 \frac{\text{m}}{\text{s}^2}$, and the plate can be inclined by $\alpha = \pm 2.92$ deg around both axes for the duration $\delta = 0.5$ s, the homogeneous solution is

$$\Phi_i = \begin{bmatrix} 1 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

for $i \in \{0, 1, ..., 8\}$, and the inhomogeneous solutions for all inclination combinations are

$$\Psi_1 = \begin{bmatrix} 0.0625 \\ 0 \\ 0.25 \\ 0 \end{bmatrix} \quad \Psi_2 = \begin{bmatrix} -0.0625 \\ 0 \\ -0.25 \\ 0 \end{bmatrix} \quad \Psi_3 = \begin{bmatrix} 0 \\ 0.0625 \\ 0 \\ 0.25 \end{bmatrix} \quad \Psi_4 = \begin{bmatrix} 0 \\ -0.0625 \\ 0 \\ -0.25 \end{bmatrix}$$

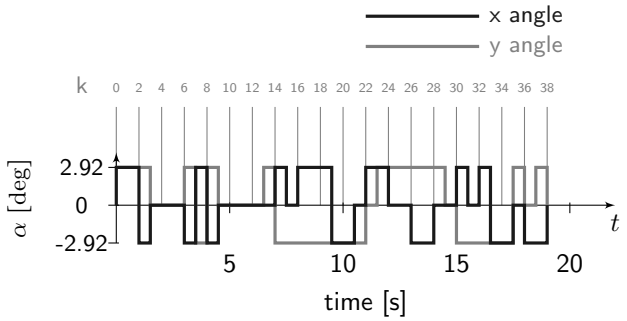


Figure 5: Planned input sequence from $k = 1$ to $k = 38$, that solves the planning problem presented in Experiment II.

$$\Psi_5 = \begin{bmatrix} -0.0625 \\ 0.0625 \\ -0.25 \\ 0.25 \end{bmatrix} \quad \Psi_6 = \begin{bmatrix} 0.0625 \\ 0.0625 \\ 0.25 \\ 0.25 \end{bmatrix} \quad \Psi_7 = \begin{bmatrix} 0.0625 \\ -0.0625 \\ 0.25 \\ -0.25 \end{bmatrix} \quad \Psi_8 = \begin{bmatrix} -0.0625 \\ -0.0625 \\ -0.25 \\ -0.25 \end{bmatrix}$$

If the plate is not inclined, the transition of the state is affected solely by the homogeneous solution and $\Psi_0 = \mathbf{0}$ per definition. The state space restrictions can be considered in the domain model using preconditions as shown in Table 2. The holes are approximated by five rectangular areas

$$R_k = \{(x, y) : x \in [x_-, x_+] \wedge y \in [y_-, y_+]\}$$

with an additional restriction defined by the plate edges

$$R_6 = \{(x, y) : x \in \mathbb{R} \setminus [0, 10] \vee y \in \mathbb{R} \setminus [0, 10]\}.$$

We generate valid transitions by modeling these restrictions as preconditions for all actions

$$p_{\vdash} : (x, y) \notin \left(R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6\right)$$

Note that, as $p_{\vdash}$ is a general precondition for all actions, we can model it as a derived predicate in PDDL.

**Planning Problem** We present an example instance of the ball maze domain and show the solution generated by our approach. Consider the following initial state $s_0$ and goal specification $s_\star$:

$$s_0 = \begin{bmatrix} 1 \\ 0.5 \\ 0 \\ 0 \end{bmatrix} \qquad s_\star = \begin{bmatrix} 7.5 \pm \epsilon_1 \\ 9 \pm \epsilon_2 \\ 0 \pm \epsilon_3 \\ 0 \pm \epsilon_4 \end{bmatrix} \tag{13}$$

where $\epsilon_1 = \epsilon_2 = 0.1$ m and $\epsilon_3 = \epsilon_4 = 0.01$ $\frac{\text{m}}{\text{s}}$.

| k | $x_+$ | $x_-$ | $y_+$ | $y_-$ | $p_{\vdash}$ = |
|---|---|---|---|---|---|
| 1 | 8 | 2 | 3 | 2 | $[(x > 8) \vee (x < 2) \vee (y > 3) \vee (y < 2)] \wedge$ |
| 2 | 6 | 4 | 6 | 4 | $[(x > 6) \vee (x < 4) \vee (y > 6) \vee (y < 4)] \wedge$ |
| 3 | 4 | 0 | 6 | 4 | $[(x > 4) \vee (x < 0) \vee (y > 6) \vee (y < 4)] \wedge$ |
| 4 | 10 | 6 | 6 | 4 | $[(x > 10) \vee (x < 6) \vee (y > 6) \vee (y < 4)] \wedge$ |
| 5 | 8 | 2 | 8 | 7 | $[(x > 8) \vee (x < 2) \vee (y > 8) \vee (y < 7)] \wedge$ |
| 6 | 10 | 0 | 10 | 0 | $[(x > 0) \wedge (x < 10) \wedge (y > 0) \wedge (y < 10)]$ |

Table 2: Obstacles of the ball maze of Example II.

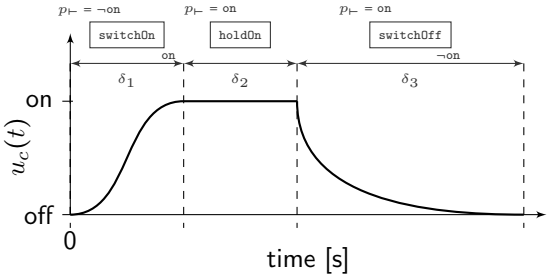


Figure 6: Switching between discrete input signals often generates continuous signals. This can be easily taken into account by the use of logical preconditions and effects.

**Results** The generated input sequence that guides the ball into the desired set point is shown in Figure 5, while the planned trajectory is presented in Figure 4, where the black dots mark the planned states and the black line is the result of the numerical MATLAB simulation. The time discretization of the system is very fine grained, making it difficult for the planner to generate complete plans in reasonable time. Moreover, the system is a multi input system (with inputs $F_x$ and $F_y$) which increases the branching factor of the search space. Therefore, the continuous replanning approach presented in this paper is very useful. In Figure 4, the intermediate promising trajectories are visualized as gray lines. Since the planner has knowledge of the intermediate predicted states, arbitrary preconditions can be checked and infeasible branches are pruned. The solution in this example was found by expanding 15000 states in each planning step, taking about 2 seconds per step. If less than 5000 states are expanded, the system typically is trapped in the labyrinth in our experiments.

## Discussion

The decomposition of continuous state variables depends directly on the action durations. There is a trade-off between planer performance and domain model accuracy: Short durations lead to finer resolutions of the mapping from the continuous to the discrete state space. On the other hand, they increase the trajectory length, making it more difficult for the planner to generate solutions. We use continuous replanning to relax the impact of time discretization.

Moreover, the number of actions depends directly on the number of possible input signals. Since $\mathcal{O}$ is limited, the domain model will suite the real plant only in an appropriate way if the input signals are limited as well. In cybernetics, control signals are often the result of optimizations which lead to arbitrary signal shapes. In planning, the discretization of arbitrary amplitudes of control signals is a rough approximation. Therefore, the domain model might not represent all capabilities of the real plant. In these cases, classical methods of cybernetics are clearly superior.

We focus on plants with discrete control signals (like the thrusters in Example 1). The resulting signals, however, are often continuous functions $u_c(t)$ in time, as depicted in Figure 6. Effects are generated in a way allowing time continuous input signals (see Equation 7). These transient effects can be covered by the domain model. It is easy for task plan-

ners to ensure that input sequences are feasible by checking preconditions. Therefore, Domain Predictive Control is capable to contain a realistic representation of the system in the domain model if the input signals of the plant are discrete with possible transition effects, as shown in Figure 6.

## Future Work

In future work we plan to enhance Domain Predictive Control in several ways for example by taking Gaussian uncertainties into account or by considering non-linearity in plants. Furthermore, we will try to find new powerful heuristics for problems making heavy use of numeric variables. We also want to compare Domain Predictive Control to related methods and investigate the effects of a planner in the control loop. We will evaluate Domain Predictive Control for deep space missions in order to achieve robustness to unexpected events in the environment or the system itself.

## Conclusion

We have presented Domain Predictive Control, an approach that utilizes action planning for the control of dynamic systems. We showed how planning domains can be derived from linear time invariant systems, which is an important class of systems in cybernetics. Using these domains, we presented how a planning system can be used to guide a hybrid system from an initial state into a state satisfying the goal conditions. An important feature of utilizing action planning for controlling hybrid systems is that beyond regular state transitions system transitions can be considered as well. On the one hand, the discretization aspect of the approach delimits the application field as discussed. On the other hand, exactly the types of systems our approach is suited best for, which are systems with discrete inputs, system transitions, and many logical dependencies, are difficult to be handled by classical control methods by now.

## Acknowledgments

This work was supported by the German Aerospace Center (DLR) and EADS Astrium-Satellites as part of the Project "Kontiplan" (50 RA 1010).

## References

Anthony, R. N. 1981. *Planning and Control Systems: A Framework for Analysis*. Boston and Mass: Division of Research, Graduate School of Business Administration, Harvard University.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with Problems Requiring Temporal Coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 892–897. AAAI Press.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of the Twenty International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 42–49.

Coles, A.; Coles, A.; Clark, A.; and Gilmore, S. 2011. Cost-Sensitive Concurrent Planning under Duration Uncertainty for Service Level Agreements. In *Proceedings of the Twenty First International Conference on Automated Planning and Scheduling (ICAPS 2011)*, 34–41.

Della Penna, G.; Intrigila, B.; Magazzeni, D.; and Mercorio, F. 2010. Planning for Autonomous Planetary Vehicles. In *Proceeding of the Sixth International Conference on Autonomic and Autonomous Systems*, 131–136.

Do, M. B., and Kambhampati, S. 2003. Sapa: A Multi-objective Metric Temporal Planner. *Journal of Artificial Intelligence Research* 20:155–194.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 130–137. AAAI Press.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20:61–124.

Fox, M.; Long, D.; and Magazzeni, D. 2011. Automatic Construction of Efficient Multiple Battery Usage Policies. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling*, 74–81.

Geering, H. P. 2007. *Optimal Control with Engineering Applications*. Berlin [u.a.]: Springer.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning Through Stochastic Local Search and Temporal Action Graphs in LPG. *Journal of Artificial Intelligence Research* 20:239–290.

Heemels, W.; Schutter, B. D.; and Bemporad, A. 2001. Equivalence of Hybrid Dynamical Models. *Automatica* 37:1085–1091.

Hoffmann, J. 2002. Extending FF to Numerical State Variables. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*, 571–575.

Isermann, R. 2006. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Berlin: Springer.

Rawlings, J. B., and Mayne, D. Q. 2009. *Model Predictive Control: Theory and Design*. Madison and Wis: Nob Hill Publishing.

Simon, D. 2006. *Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches*. New Jersey: John Wiley.

Struss, P. 1997. Fundamentals of Model-Based Diagnosis of Dynamic Systems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, 480–485.

Wang, L. 2009. *Model Predictive Control System Design and Implementation using MATLAB ®*, volume 1 of *Advances in industrial control*. London: Springer.

Wertz, J. R. 1994. *Spacecraft Attitude Determination and Control*. Dordrecht and and Boston: Kluwer Academic Publishers.

Williams, B. C., and Nayak, P. P. 1996. A Model-Based Approach to Reactive Self-Configuring Systems. In *Proceedings of the National Conference on Artificial Intelligence*, 971–978.