Automatic Control Software Synthesis for Quantized Discrete Time Hybrid Systems

Vadim Alimghuzin, Federico Mari, Igor Melatti,
Ivano Salvo, Enrico Tronci

Department of Computer Science
Sapienza University of Rome
via Salaria 113, 00198 Rome
email: {alimghuzin,mari,melatti,salvo,tronci}@di.uniroma1.it
Vadim Alimghuzin is also with the Department of Computer Science and Robotics
Ufa State Aviation Technical University
12 Karl Marx Street, Ufa, 450000, Russian Federation

November 5, 2018

Abstract

Many *Embedded Systems* are indeed *Software Based Control Systems*, that is control systems whose controller consists of *control software* running on a microcontroller device. This motivates investigation on *Formal Model Based Design* approaches for automatic synthesis of embedded systems control software. This paper addresses control software synthesis for discrete time *nonlinear* systems. We present a methodology to overapproximate the dynamics of a discrete time nonlinear hybrid system $\mathcal{H}$ by means of a discrete time *linear* hybrid system $\mathcal{L}_{\mathcal{H}}$, in such a way that controllers for $\mathcal{L}_{\mathcal{H}}$ are guaranteed to be controllers for $\mathcal{H}$. We present experimental results on the inverted pendulum, a challenging and meaningful benchmark in nonlinear Hybrid Systems control.

1 Introduction

Many *Embedded Systems* are indeed *Software Based Control Systems* (SBCSs). An SBCS consists of two main subsystems: the *controller* and the *plant*. Typically, the plant is a physical system consisting, for example, of mechanical or electrical devices whereas the controller consists of *control software* running on a microcontroller. In an endless loop, the controller reads *sensor* outputs from the plant and sends commands to plant *actuators* in order to guarantee that the *closed loop system* (that is, the system consisting of both plant and controller) meets given *safety* and *liveness* specifications (*System Level Formal Specifications*).

Software generation from models and formal specifications forms the core of *Model Based Design* of embedded software [21]. This approach is particularly interesting for SBCSs since in such a case system level (formal) specifications are much easier to define than the control software behavior itself.

The typical control loop skeleton for an SBCS is the following. Measure x of the system state from plant *sensors* go through an *analog-to-digital* (AD) conversion, yielding a *quantized* value $\hat{x}$. A function `ctrlRegion` checks if $\hat{x}$ belongs to the region in which the control software works correctly. If this is not the case a *Fault Isolation and Recovery* (FDIR) procedure is triggered, otherwise a function `ctrlLaw` computes a command $\hat{u}$ to be sent to plant *actuators* after a *digital-to-analog* (DA) conversion. Basically, the control software design problem for SBCSs consists in designing software implementing functions `ctrlLaw` and `ctrlRegion`.

For SBCSs, system level specifications are typically given with respect to the desired behavior of the closed loop system. The *control software* (that is, `ctrlLaw` and `ctrlRegion`) is designed using a *separation-of-concerns* approach. That is, *Control Engineering* techniques (e.g., see [10]) are used to design, from the closed loop system level specifications, *functional specifications* (*control law*) for the *control software* whereas *Software Engineering* techniques are used to design control software implementing the given functional specifications. Such a separation-of-concerns approach has several drawbacks.

First, usually control engineering techniques do not yield a formally verified specification for the control law when quantization is taken into account. This is particularly the case when the plant has to be modelled as a *Hybrid System*, that is a system with continuous as well as discrete state changes [5, 1, 16, 4]. As a result, even if the control software meets its functional specifications there is no formal guarantee that system level specifications are met since quantization effects are not formally accounted for.

Second, issues concerning computational resources, such as control software *Worst Case Execution Time* (WCET), can only be considered very late in the SBCS design activity, namely once the software has been designed. As a result, the control software may have a WCET greater than the sampling time. This invalidates the schedulability analysis (typically carried out before the control software is completed) and may trigger redesign of the software or even of its functional specifications (in order to simplify its design).

Last, but not least, the classical separation-of-concerns approach does not effectively support design space exploration for the control software. In fact, although in general there will be many functional specifications for the control software that will allow meeting the given system level specifications, the software engineer only gets one to play with. This overconstrains a priori the design space for the control software implementation preventing, for example, effective performance trading (e.g., between number of bits in AD conversion, WCET, RAM usage, CPU power consumption, etc.). We note that the above considerations also apply to the typical situation where Control Engineering techniques are used to design a control law and then tools like Simulink are used to generate the control software.

The previous considerations motivate research on Software Engineering methods and tools focusing on control software synthesis (rather than on control law synthesis as in Control Engineering). The objective is that from the plant model (as a hybrid system), from formal specifications for the closed loop system behavior and from *Implementation Specifications* (that is, number of bits used in the quantization process) such methods and tools can generate correct-by-construction control software satisfying the given specifications.

The tool QKS [23] synthesise control software for *Discrete Time Linear Hybrid Systems* (DTLHSs). However, the dynamics of many interesting hybrid systems cannot be directly modeled by linear predicates. The focus of the present paper is control software synthesis for *nonlinear* Discrete Time Hybrid Systems.

1.1 Our Main Contributions

We model the controlled system (plant) as a *Discrete Time Hybrid System* (DTHS), that is a discrete time hybrid system whose dynamics is modeled as a *predicate* (possibly non linear) over a set of continuous as well as discrete variables that describe system state, system inputs and disturbances.

System level safety as well as liveness specifications are modeled as sets of states defined, in turn, as predicates. In our setting, as always in control problems, liveness constraints define the set of states that any evolution of the closed loop system should eventually reach (*goal states*). Using an approach similar to the one in [20], in [24] it has been proven that both existence of a controller and existence of a *quantized* controller for DTHSs are undecidable problems, even for very restricted classes of DTHSs. Accordingly, we can only hope for non complete or semi-algorithms.

In this paper we present a general approach to deal with discrete time non-linear hybrid systems. The basic idea is to overapproximate the behaviour of a DTHS $\mathcal{H}$ by means of a DTLHS $\mathcal{L}_{\mathcal{H}}$. Stemming from Corollary 3, that ensures that controllers for $\mathcal{L}_{\mathcal{H}}$ are guaranteed to be controllers for $\mathcal{H}$, we synthesize control software by giving as input to the tool QKS [23] the linear plant model $\mathcal{L}_{\mathcal{H}}$, the desired quantization schema, and system level formal specifications.

Since $\mathcal{L}_{\mathcal{H}}$ dynamics overapproximates the dynamics of $\mathcal{H}$, the controllers that we synthesize are inherently *robust*, that is they meet the given closed loop requirements *notwithstanding* nondeterministic small *disturbances* such as variations in the plant parameters. Tighter overapproximations makes finding a controller easier, whereas coarser overapproximations makes controllers more robust. As in the linear case, the automatically generated software has a *Worst Case Execution Time* (WCET) guaranteed to be linear in the number of bits of the state quantization schema. Moreover, control software computes commands in such a way that the closed loop system follows a (near) *time optimal* strategy to reach the goal [15].

We present experimental results on the inverted pendulum benchmark [22], a challenging and well studied example in control synthesis.

1.2 Related Work

Control Engineering has been studying control law design (e.g., optimal control, robust control, etc.), for more than half a century (e.g., see [10]). Also *Quantized Feedback Control* has been widely studied in control engineering (e.g. see [14]). However such research does not address hybrid systems (our case) and, as explained above, focuses on control law design rather than on control software synthesis (our goal). Furthermore, all control engineering approaches model *quantization errors* as statistical *noise*. As a result, correctness of the control law holds in a probabilistic sense. Here instead, we model quantization errors as nondeterministic (*malicious*) *disturbances*. This guarantees system level correctness of the generated control software (not just that of the control law) with respect to *any* possible sequence of quantization errors.

When the plant model is a *Linear Hybrid Automaton* (LHA) [1, 4] reachability and existence of a control law are both undecidable problems [19, 20]. This, of course, has not prevented devising effective (semi) algorithms for such problems. Examples are in [4, 16, 13, 30, 28, 9]. Control software synthesis for continuous time linear systems (no switching) has been implemented in the tool PESSOA [25]. Such an approach exploits suitable finite state abstraction (e.g. see [27, 26]) to synthesize a control law computing commands from real valued state measures (no quantization). The control software is then generated by passing to Simulink such a control law. In the same wavelength, [31] generates a control strategy from a finite abstraction of a *Piecewise Affine Discrete Time Hybrid Systems* (PWA-DTHS). Also the Hybrid Toolbox [7] considers PWA-DTHS. Such a tool outputs a feedback control law that is then passed to Matlab in order to generate control software. Finite horizon control of PWA-DTHS has been studied using a MILP based approach. See, for example, [8]. Explicit finite horizon control synthesis algorithms for discrete time (possibly non-linear) hybrid systems have been studied in [12] and citations thereof.

We note that all such approaches do not account for state feedback quantization since they all assume *exact* (i.e. real valued) state measures. Thus, as explained above, they do not offer any formal guarantee about system level correctness of the generated software, which is instead our focus here.

Quantization can be seen as a sort of abstraction, which has been widely studied in a hybrid system formal verification context (e.g., see [2, 3]). Note however that in a verification context abstractions are designed so as to ease the verification task whereas in control software synthesis quantization is a design requirement since it models a hardware component (AD converter) which is part of the specification of the control software synthesis problem. Indeed, in our setting, we have to design a controller *notwithstanding* the nondeterminism stemming from the quantization process. As a result, the techniques used to devise clever abstractions in a verification setting cannot be directly used in our synthesis setting where quantization is given.

The tool QKS [23] synthesize control software from system level specification for Discrete Time Linear Hybrid Systems whenever a a constructive sufficient condition for control software existence holds. Here, we address control software synthesis for a more general class of discrete time hybrid systems.

In the context of Hybrid Systems verification, the overapproximation of Hybrid Systems with Linear Hybrid Systems has been studied in [18] and [17]. Such works consider dense time models, and focus on verification rather than control synthesis. Moreover, we observe that we can obtain tighter approximations, since

DTLHSs allow us to model system dynamics with predicates that mix present and next state variables.

Correct-by-construction software synthesis in a finite state setting has been studied, for example, in [6, 29, 11]. Such approaches cannot be directly used in our context since they cannot handle continuous state variables.

Summing up, to the best of our knowledge, no previously published result is available about automatic generation of correct-by-construction control software from a DTHS model of the plant, *system level formal specifications* and *implementation specifications* (*quantization*, that is number of bits in AD conversion).

2 Background

We denote with $[n]$ an initial segment $\{1,\ldots,n\}$ of the natural numbers. We denote with $X = [x_1,\ldots,x_n]$ a finite sequence (list) of variables. By abuse of language we may regard sequences as sets and we use $\cup$ to denote list concatenation. Each variable x ranges on a known (bounded or unbounded) interval $\mathcal{D}_x$ either of the reals or of the integers (discrete variables). We denote with $\mathcal{D}_X$ the set $\prod_{x\in X}\mathcal{D}_x$. To clarify that a variable x is *continuous* (i.e. real valued) we may write x^r. Similarly, to clarify that a variable x is *discrete* (i.e. integer valued) we may write x^d. Analogously X^r (X^d) denotes the sequence of real (integer) variables in X. Finally, boolean variables are discrete variables ranging on the set $\mathbb{B} = \{0, 1\}$. If x is a boolean variable we write $\bar{x}$ for $(1-x)$.

2.1 Predicates

An *expression* $E(X)$ over a list of variables X is an expression of the form $\sum_{i\in[n]} a_i f_i(X)$, where $f_i(X)$ is a possibly nonlinear function over X and a_i are rational constants. $E(X)$ is a *linear expression* if each $f_i(X)$ is a projection (i.e. $f_i(X) = x_i$), i.e. if it is a linear combination of variables $\sum_{i\in[n]} a_i x_i$. A constraint is an expression of the form $E(X) \leq b$, where b is a rational constant. In the following, we also write $E(X) \geq b$ for $-E(X) \leq -b$.

Predicates are inductively defined as follows. A *constraint* $C(X)$ over a list of variables X is a predicate over X. If $A(X)$ and $B(X)$ are predicates over X, then $(A(X) \wedge B(X))$ and $(A(X) \vee B(X))$ are predicates over X. Parentheses may be omitted, assuming usual associativity and precedence rules of logical operators. A *conjunctive predicate* is a conjunction of constraints. For conjunctive predicates we will also write: $E(X) = b$ for $((E(X) \leq b) \wedge (E(X) \geq b))$ and $a \leq x \leq b$ for $x \geq a \wedge x \leq b$, where $x \in X$.

A *valuation* over a list of variables X is a function v that maps each variable $x \in X$ to a value $v(x) \in \mathcal{D}_x$. Given a valuation v, we denote with $X^* \in \mathcal{D}_X$ the sequence of values $[v(x_1), \ldots, v(x_n)]$. By abuse of language, we call valuation also the sequence of values X^*. A *satisfying assignment* to a predicate P over X is a valuation X^* such that $P(X^*)$ holds. If a satisfying assignment to a predicate P over X exists, we say that P is *feasible*. Abusing notation, we may denote with P the set of satisfying assignments to the predicate $P(X)$. Two predicates P and Q over X are *equivalent*, denoted by $P \equiv Q$, if they have the same set of satisfying assignments. Two predicates P and Q are *equisatisfiable* if P is feasible iff Q is feasible.

A variable $x \in X$ is said to be *bounded* in P if there exist a, $b \in \mathcal{D}_x$ such that $P(X)$ implies $a \leq x \leq b$. A predicate P is bounded if all its variables are bounded.

Given a constraint $C(X)$ and a fresh boolean variable (*guard*) $y \notin X$, the *guarded constraint* $y \rightarrow C(X)$ (if y then $C(X)$) denotes the predicate $((y = 0) \vee C(X))$. Similarly, we use $\bar{y} \rightarrow C(X)$ (if not y then $C(X)$) to denote the predicate $((y = 1) \vee C(X))$. A *guarded predicate* is a conjunction of either constraints or guarded constraints. It is possible to show that, if a guarded predicate P is bounded, then P can be transformed into an equivalent (bounded) conjunctive predicate [24].

2.2 Labeled Transition Systems

A *Labeled Transition System* (LTS) is a tuple $\mathcal{S} = (S, A, T)$ where S is a (possibly infinite) set of states, A is a (possibly infinite) set of *actions*, and $T : S \times A \times S \rightarrow \mathbb{B}$ is the *transition relation* of $\mathcal{S}$. We say that T (and $\mathcal{S}$) is *deterministic* if $T(s, a, s') \wedge T(s, a, s'')$ implies $s' = s''$, and *nondeterministic* otherwise. Let $s \in S$ and $a \in A$. We denote with $\mathrm{Adm}(\mathcal{S}, s)$ the set of actions admissible in s, that is $\mathrm{Adm}(\mathcal{S}, s) = \{a \in A \mid \exists s' : T(s, a, s')\}$ and with $\mathrm{Img}(\mathcal{S}, s, a)$ the set of next states from s via a, that is $\mathrm{Img}(\mathcal{S}, s, a) = \{s' \in S \mid T(s, a, s')\}$. A *run* or *path* for an LTS $\mathcal{S}$ is a sequence $\pi = s_0, a_0, s_1, a_1, s_2, a_2, \ldots$ of states s_t and actions a_t such that $\forall t \geq 0$ $T(s_t, a_t, s_{t+1})$. The length $|\pi|$ of a finite run π is the number of actions in π. We denote with $\pi^{(S)}(t)$ the $(t+1)$-th state element of π, and with $\pi^{(A)}(t)$ the $(t+1)$-th action element of π. That is $\pi^{(S)}(t) = s_t$, and $\pi^{(A)}(t) = a_t$.

Given two LTSs $\mathcal{S}_1 = (S, A, T_1)$ and $\mathcal{S}_2 = (S, A, T_2)$, we say that $\mathcal{S}_1$ *refines* $\mathcal{S}_2$ (notation $\mathcal{S}_1 \sqsubseteq \mathcal{S}_2$) iff $T_1(s, a, s')$ implies $T_2(s, a, s')$ for each state $s, s' \in S$ and action $a \in A$. The refinement relation is a partial order on LTSs.

2.3 LTS Control Problem

A *controller* for an LTS $\mathcal{S}$ is used to restrict the dynamics of $\mathcal{S}$ so that all states in the initial region will reach in one or more steps the goal region. In the following, we formalize such a concept by defining strong solutions to an LTS control problem. In what follows, let $\mathcal{S} = (S, A, T)$ be an LTS, I, $G \subseteq S$ be, respectively, the *initial* and *goal* regions of $\mathcal{S}$.

Definition 1 *A* controller *for $\mathcal{S}$ is a function $K : S \times A \to \mathbb{B}$ such that $\forall s \in S$, $\forall a \in A$, if $K(s,a)$ then $\exists s'\ T(s,a,s')$.* $\mathrm{dom}(K)$ *denotes the set of states for which at least a control action is enabled. Formally,* $\mathrm{dom}(K) = \{s \in S \mid \exists a\, K(s,a)\}$. $\mathcal{S}^{(K)}$ *denotes the* closed loop system, *that is the LTS $(S, A, T^{(K)})$, where $T^{(K)}(s,a,s') = T(s,a,s') \wedge K(s,a)$.*

We call a path π *fullpath* [6] if either it is infinite or its last state $\pi^{(S)}(|\pi|)$ has no successors (i.e. $\mathrm{Adm}(\mathcal{S}, \pi^{(S)}(|\pi|)) = \varnothing$). We denote with $\mathrm{Path}(s,a)$ the set of fullpaths starting in state s with action a, i.e. the set of fullpaths π such that $\pi^{(S)}(0) = s$ and $\pi^{(A)}(0) = a$.

Given a path π in $\mathcal{S}$, we define $J(\mathcal{S}, \pi, G)$ as follows. If there exists $n > 0$ s.t. $\pi^{(S)}(n) \in G$, then $J(\mathcal{S}, \pi, G) = \min\{n \mid n > 0 \wedge \pi^{(S)}(n) \in G\}$. Otherwise, $J(\mathcal{S}, \pi, G) = +\infty$. We require $n > 0$ since our systems are nonterminating and each controllable state (including a goal state) must have a path of positive length to a goal state. Taking $\sup \varnothing = +\infty$ and $\inf \varnothing = -\infty$, the *worst case distance* of a state s from the goal region G is $J_{\mathrm{strong}}(\mathcal{S}, G, s) = \sup\{J_s(\mathcal{S}, G, s, a) \mid a \in \mathrm{Adm}(\mathcal{S}, s)\}$, being $J_s(\mathcal{S}, G, s, a) = \sup\{J(\mathcal{S}, G, \pi) \mid \pi \in \mathrm{Path}(s,a)\}$.

Definition 2 *A* control problem *for $\mathcal{S}$ is a triple $\mathcal{P} = (\mathcal{S}, I, G)$. A* strong *solution (or simply a solution) to $\mathcal{P}$ is a controller K for $\mathcal{S}$, such that $I \subseteq$* $\mathrm{dom}(K)$ *and for all $s \in$* $\mathrm{Dom}(K)$, *$J_{strong}(\mathcal{S}^{(K)}, G, s)$ is finite.*

An optimal *solution to $\mathcal{P}$ is a solution K^* to $\mathcal{P}$ s.t. for all solutions K to $\mathcal{P}$, for all $s \in \mathcal{D}_X$ we have: $J_{strong}(\mathcal{S}^{(K^*)}, G, s) \leq J_{strong}(\mathcal{S}^{(K)}, G, s)$. The* most general optimal (mgo) solution *to $\mathcal{P}$ is an optimal solution $\overline{K}$ to $\mathcal{P}$ s.t. for all optimal solutions K to $\mathcal{P}$, for all $s \in \mathcal{D}_X$, for all $u \in \mathcal{D}_U$ we have: $K(s,u) \to \overline{K}(s,u)$. It is easy to see that this definition is well posed (i.e., the mgo solution is unique) and that $\overline{K}$ does not depend on I.*

3 Discrete Time Hybrid Systems

In this section we introduce our class of *Discrete Time Hybrid Systems* (DTHS for short), together with the DTHS representing the inverted pendulum on which our experiments will focus. Moreover, we will define in Sect. 3.2 the *Quantized Control Problem*.

Definition 3 *A* Discrete Time Hybrid System *is a tuple* $\mathcal{H} = (X, U, Y, N)$ *where:*

- $X = X^r \cup X^d$ *is a finite sequence of real* (X^r) *and discrete* (X^d) present state *variables. We denote with* X' *the sequence of* next state *variables obtained by decorating with* $'$ *all variables in* X.
- $U = U^r \cup U^d$ *is a finite sequence of* input *variables.*
- $Y = Y^r \cup Y^d$ *is a finite sequence of* auxiliary *variables. Auxiliary variables are typically used to model* modes *(e.g., from switching elements such as diodes) or "local" variables.*
- $N(X, U, Y, X')$ *is a conjunctive predicate over* $X \cup U \cup Y \cup X'$ *defining the* transition relation *(*next state*) of the system.* N *is* deterministic *if* $N(x, u, y_1, x')$ $\wedge$ $N(x, u, y_2, x'')$ *implies* $x' = x''$*, and* nondeterministic *otherwise.*

A DTHS is bounded *if the predicate N is bounded. A DTHS is* deterministic *if N is deterministic. A DTHS is* linear*, and we call it DTLHS if N is a conjunction of linear constraints.*

Since any bounded guarded predicate can be transformed into a conjunctive predicate (see Sect. 2.1), for the sake of readability we will use bounded guarded predicates to describe the transition relation of bounded DTHSs. To this aim, we will also clarify which variables are boolean, and thus may be used as guards in guarded constraints.

Example 1 *Let us consider a simple inverted pendulum [22], as shown in Fig. 1. The system is modeled by taking the angle* θ *and the angular velocity* $\dot{\theta}$ *as state variables. The input of the system is the torquing force u, that can influence the velocity in both directions. Moreover, the behaviour of the system depends on the pendulum mass m, the length of the pendulum l and the gravitational acceleration g. Given such parameters, the motion of the system is described by the differential equation* $\ddot{\theta} = \frac{g}{l} \sin\theta + \frac{1}{ml^2} u$.

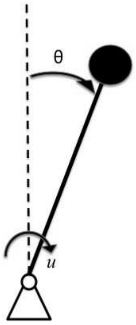

Figure 1: Inverted Pendulum with Stationary Pivot Point.

In order to obtain a state space representation, we consider the following normalized system, where x_1 is the angle θ and x_2 is the angular speed $\dot{\theta}$.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \frac{g}{l}\sin x_1 + \frac{1}{ml^2}u \end{cases} \tag{1}$$

The DTHS model $\mathcal{H}$ for the pendulum is the tuple (X,U,Y,N), where $X = \{x_1,x_2\}$ is the set of continuous state variables, $U = \{u\}$ is the set of input variables, and $Y = \varnothing$. Differently from [22], we consider the problem of finding a discrete controller, whose decisions maybe “apply the force clockwise” ($u = 1$), “apply the force counterclockwise” ($u = -1$)”, or “do nothing” ($u = 0$). The intensity of the force will be given as a constant F. Finally, the discrete time transition relation N is obtained from the equations in (1) by introducing a constant T that models the sampling time. N is the predicate $(x_1' = x_1 + Tx_2) \wedge (x_2' = x_2 + T\frac{g}{l}\sin x_1 + T\frac{1}{ml^2}Fu)$.

The semantics of DTHSs is given in terms of LTSs.

Definition 4 *Let $\mathcal{H}$ = (X, U, Y, N) be a DTHS. The dynamics of $\mathcal{H}$ is defined by the Labeled Transition System* $\mathrm{LTS}(\mathcal{H})$ = ($\mathcal{D}_X$, $\mathcal{D}_U$, $\tilde{N}$) *where:* $\tilde{N} : \mathcal{D}_X \times \mathcal{D}_U \times \mathcal{D}_X \to \mathbb{B}$ *is a function s.t.* $\tilde{N}(x,u,x') \equiv \exists y \in \mathcal{D}_Y : N(x,u,y,x')$. *A* state *$x$ for $\mathcal{H}$ is a* state *x for* $\mathrm{LTS}(\mathcal{H})$ *and a* run *(or* path*) for $\mathcal{H}$ is a run for* $\mathrm{LTS}(\mathcal{H})$ *(Sect. 2.2).*

3.1 DTHS Control Problem

A DTHS control problem $(\mathcal{H},I,G)$ is defined as the LTS control problem ($\mathrm{LTS}(\mathcal{H})$, I, G). To accommodate quantization errors, always present in software based controllers, it is useful to relax the notion of control solution by tolerating an (arbitrarily small) error ε on the continuous variables. This leads to the definition of ε-solution. Let ε be a nonnegative real number, $W \subseteq \mathbb{R}^n \times \mathbb{Z}^m$. The ε-*relaxation* of W is the set (*ball* of radius ε) $\mathcal{B}_\varepsilon(W) = \{(z_1, \ldots z_n,\ q_1, \ldots q_m) \mid \exists (x_1, \ldots, x_n, q_1, \ldots q_m) \in W$ and $\forall i \in [n]\ |z_i - x_i| \leq \varepsilon\}$.

Definition 5 *Let $(\mathcal{H}, I, G)$ be a DTHS control problem and ε be a nonnegative real number. An ε solution to $(\mathcal{H}, I, G)$ is a solution to the LTS control problem $(\mathrm{LTS}(\mathcal{H}), I, \mathcal{B}_\varepsilon(G))$.*

Example 2 *Let T be a positive constant (sampling time). We define the DTHS $\mathcal{H}$ = $(\{x\}, \{u\}, \varnothing, N)$ where x is a continuous variable, u is a boolean variable, and $N(x, u, x') \equiv [\overline{u} \rightarrow x' = x + (\frac{5}{4} - x)T] \wedge [u \rightarrow x' = x + (x - \frac{3}{2})T]$. Let $\mathcal{P}$ = ($\mathcal{H}$, I, G) be a control problem, where $I \equiv -2 \leq x \leq 2.5$, and $G \equiv x = 0$. A controller may drive the system near enough to the goal $x = 0$, by enabling a suitable action in such a way that $x' < x$ when $x > 0$ and $x' > x$ when $x < 0$. If the sampling time T is small enough with respect to ε (for example $T < \frac{\varepsilon}{10}$), the controller: $K(x, u) = (-2 \leq x \leq 0\ \wedge\ \overline{u})\ \vee\ (0 \leq x \leq \frac{11}{8}\ \wedge\ u)\ \vee\ (\frac{11}{8} \leq x \leq 2.5\ \wedge\ \overline{u})$ is an ε solution to $(\mathcal{H}, I, G)$. Observe that, that any controller K' such that $K'(\frac{5}{4}, 0)$ holds is not a solution, because since $N(\frac{5}{4}, 0, \frac{5}{4})$ holds, the closed loop system $\mathcal{H}^{(K)}$ may loop forever along the path $\frac{5}{4}, 0, \frac{5}{4}, 0 \ldots$.*

Example 3 *The typical goal for the inverted pendulum in Example 1 is to turn the pendulum steady to the upright position, starting from any possible initial position, within a given speed interval. In our experiments, the goal region is defined by the predicate $G(X) \equiv (-\rho \leq x_1 \leq \rho) \wedge (-\rho \leq x_2 \leq \rho)$, where $\rho \in \{0.05, 0.1\}$, and the initial region is defined by the predicate $I(X) \equiv (-\pi \leq x_1 \leq \pi) \wedge (-4 \leq x_2 \leq 4)$.*

3.2 Quantized Control Problem

In order to manage real variables, in classical control theory the concept of *quantization* is introduced (e.g., see [14]). Quantization is the process of approximating a continuous interval by a set of integer values. In the following we formally define a quantized feedback control problem for DTHSs.

A *quantization function* γ for a real interval $I = [a, b]$ is a non-decreasing function $\gamma : I \mapsto \mathbb{Z}$ s.t. $\gamma(I)$ is a bounded integer interval. We will denote $\gamma(I)$ as $\hat{I} = [\gamma(a), \gamma(b)]$. The *quantization step* of γ, notation $\|\gamma\|$, is defined as $\sup\{\ |w - z| \mid w, z \in I \wedge \gamma(w) = \gamma(z)\}$. For ease of notation, we extend quantizations to integer intervals, by stipulating that in such a case the quantization function is the identity function.

Definition 6 *Let $\mathcal{H} = (X, U, Y, N)$ be a DTHS, and let $W = X \cup U \cup Y$. A* quantization $\mathcal{Q}$ *for $\mathcal{H}$ is a pair (A, Γ), where:*

- *A is a predicate over W that explicitely bounds each variable in W. For each $w \in W$, we denote with A_w its* admissible region *and with $A_W = \prod_{w \in W} A_w$.*
- *Γ is a set of maps $\Gamma = \{\gamma_w \mid w \in W$ and γ_w is a quantization function for $A_w\}$.*

Let $W = [w_1, \ldots w_k]$ and $v = [v_1, \ldots v_k] \in A_W$. We write $\Gamma(v)$ for the tuple $[\gamma_{w_1}(v_1), \ldots, \gamma_{w_k}(v_k)]$. Finally, the quantization step $\|\Gamma\|$ *is defined as* $\sup\{\, \|\gamma\| \mid \gamma \in \Gamma\}$.

A control problem admits a *quantized* solution if control decisions can be made by just looking at quantized values. This enables a software implementation for a controller.

Definition 7 *Let $\mathcal{H} = (X, U, Y, N)$ be a DTHS, $\mathcal{Q} = (A, \Gamma)$ be a quantization for $\mathcal{H}$ and $\mathcal{P} = (\mathcal{H}, I, G)$ be a DTHS control problem. A $\mathcal{Q}$* Quantized Feedback Control *(QFC) solution to $\mathcal{P}$ is a $\|\Gamma\|$ solution $K(x, u)$ to $\mathcal{P}$ such that $K(x, u) = \hat{K}(\Gamma(x), \Gamma(u))$ where $\hat{K} : \Gamma(A_X) \times \Gamma(A_U) \to \mathbb{B}$.*

Example 4 *Let $\mathcal{P}$ be as in Example 2. Let us consider the quantization (A, Γ) where $A = I$ and $\Gamma = \{\gamma_x\}$ where $\gamma_x(x) = \lfloor x \rfloor$. The set $\Gamma(A_x)$ of quantized states is the integer interval $[-2, 2]$. No $\mathcal{Q}$ QFC solution can exist, because defining both $\hat{K}(1, 1)$ and $\hat{K}(1, 0)$ allows infinite loops to be potentially executed in the closed loop system. Of course, the controller K in Example 2 can be obtained as a quantized controller decreasing the quantization step, for example by taking $\tilde{\Gamma} = \{\tilde{\gamma}_x\}$ where $\tilde{\gamma}_x(x) = \lfloor 8x \rfloor$.*

4 DTLHS overapproximation of DTHSs

In [23], we presented the tool QKS that given a DTLHS control problem $\mathcal{P} = (\mathcal{H}, I, G)$ and a quantization schema as input, yields as output control software implementing a most general optimal quantized controller for $\mathcal{P}$, whenever a sufficient condition holds. In this section we show how a DTHS $\mathcal{H}$ can be overapproximate by a DTLHS $\mathcal{L}_{\mathcal{H}}$, in such a way that $\mathrm{LTS}(\mathcal{H}) \sqsubseteq \mathrm{LTS}(\mathcal{L}_{\mathcal{H}})$. The following theorem ensures that controllers for $\mathcal{L}_{\mathcal{H}}$ are guaranteed to be controllers for $\mathcal{H}$.

4.1 DTHS linearization

Let $C(V)$, with $V \subseteq X \cup U \cup Y \cup X'$, be a constraint in N that contains a nonlinear function as a subterm. Then $C(V)$ has the shape $f(R,W)+E(V) \leq b$, where $R \subseteq V^r$ is a set of n real variables $\{r_1,\ldots,r_n\}$, and $W \subseteq V^d$ is a set of discrete variables. For each $w \in \mathcal{D}_W$, we define the function $f_w(R)$ obtained from f, by instanciating discrete variables with w, i.e $f_w(R) = f(R,w)$. Then $C(V)$ is equivalent to the conjunctive predicate $\bigwedge_{w\in\mathcal{D}_W}[f_w(R)+E(V) \leq b]$. In order to make the overapproximation tighter, we partition the domain $\mathcal{D}_R$ of each function $f_w(R)$ into m hyperintervals $I_1, I_2 \ldots I_m$, where $I_i = \Pi_{j\in[n]}[a^i_j, b^i_j]$. In the following $R \in I_i$ will denote the conjunctive predicate $\bigwedge_{j\in[n]} a^i_j \leq r_j \leq b^i_j$.

Let $f^+_{w,i}(R)$ and $f^-_{w,i}(R)$ be over- and under- linear approximations of $f_w(R)$ over the hyperinterval I_i, i.e. such that $R \in I_i$ implies $f^-_{w,i}(R) \leq f_w(R) \leq f^+_{w,i}(R)$. Taking $|\mathcal{D}_W| \times n$ fresh continuous variables $Y = \{y_{w,i}\}_{w\in\mathcal{D}_W, i\in[n]}$, we define the conjunctive predicate $\tilde{C}(V,Y)$:

$$\begin{aligned}&\textstyle\bigwedge_{w\in\mathcal{D}_W}\bigwedge_{i\in[m]}[y_{w,i}+E(V) \leq b]\\ &\quad\wedge \textstyle\bigwedge_{w\in\mathcal{D}_W}[\bigvee_{i\in[m]}[R \in I_i \wedge f^-_{w,i}(R) \leq y_{w,i} \leq f^+_{w,i}(R)]]\end{aligned}$$

By introducing $|\mathcal{D}_W| \times n$ fresh boolean variables $Z = \{z_i\}_{w\in\mathcal{D}_W, i\in[n]}$, $\tilde{C}(V,Y)$ can be translated into the following equisatisfiable conjunctive predicate $\overline{C}(V,Y,Z)$:

$$\begin{aligned}&\textstyle\bigwedge_{w\in\mathcal{D}_W}\bigwedge_{i\in[m]}[y_{w,i}+E(V) \leq b]\\ &\quad\wedge \textstyle\bigwedge_{w\in\mathcal{D}_W}\bigwedge_{i\in[m]} z_{w,i} \to f^-_{w,i}(R) \leq y_{w,i} \leq f^+_{w,i}(R)\\ &\wedge \textstyle\bigwedge_{w\in\mathcal{D}_W}\bigwedge_{i\in[m]} z_{w,i} \to R \in I_i \wedge \bigwedge_{w\in\mathcal{D}_W}\sum_{i\in[m]} z_{w,i} \geq 1\end{aligned}$$

As a result, this transformation eliminates a nonlinear subexpression of a constraint $C(V)$ and yields a constraint $\overline{C}(V,Y,Z)$ such that $\exists Y,Z[\overline{C}(V,Y,Z) \Rightarrow C(V)]$. Given a DTHS $\mathcal{H} = (X,U,Y,N)$, without loss of generality, we may suppose that the transition relation N is a conjunction $\bigwedge_{i\in[m]} C_i(X,U,Y,X')$ of constraints. By applying the above transformation to each nonlinear subexpressions occurring in N, we obtain a conjunction of linear constraints $\overline{N} \equiv \bigwedge_{i\in[m]}\overline{C}_i(X,U,\overline{Y},X')$, such that $\overline{N} \Rightarrow N$. Hence, starting from a DTHS $\mathcal{H}$, we find a DTLHS $\mathcal{L}_{\mathcal{H}} = (X,U,\overline{Y},\overline{N})$, whose dynamics overapproximate the dynamics of $\mathcal{H}$.

Theorem 1 *Let $\mathcal{H} = (X,U,Y,N)$ be a DTHS and let $\mathcal{L}_{\mathcal{H}}$ be its linearization. Then we have that $LTS(\mathcal{H}) \sqsubseteq LTS(\mathcal{L}_{\mathcal{H}})$.*

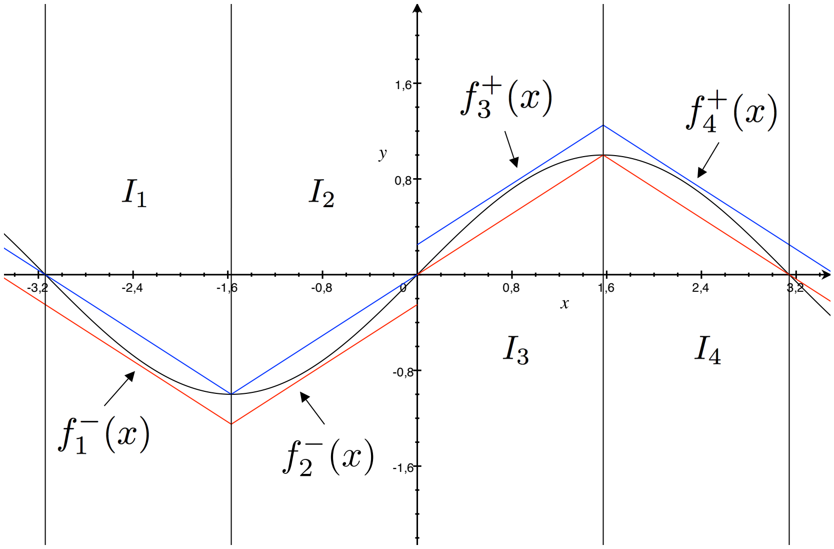

Figure 2: Linearization of $\sin x$ in $[-\pi,\pi]$.

Theorem 2 *Let $\mathcal{S}_1 = (S, A, T_1)$ and $\mathcal{S}_2 = (S, A, T_2)$ be two LTSs, and let K be a solution for the LTS control problem $(\mathcal{S}_2, I, G)$. If $\mathcal{S}_1$ refines $\mathcal{S}_2$ and for all $s \in S$ $Adm(\mathcal{S}_1, s) = Adm(\mathcal{S}_1, s)$, then K is a solution also for $(\mathcal{S}_1, I, G)$.*

Proof 1 *(Sketch) The proof is by induction on $n = J_{\text{strong}}(\mathcal{S}_2^{(K)}, G, s)$. If $n = 1$ and $K(s,a)$, then $\text{Img}(\mathcal{S}_2, s, a) \subseteq G$. Since $\mathcal{S}_1 \sqsubseteq \mathcal{S}_2$, we also have that $\text{Img}(\mathcal{S}_1, s, a) \subseteq \text{Img}(\mathcal{S}_2, s, a) \subseteq G$. Moreover, $Adm(\mathcal{S}_1, s) = Adm(\mathcal{S}_2, s)$ implies that there exists at least a transition of the shape $T_1(s, a, s')$ with $s' \in G$ and thus $J_{\text{strong}}(\mathcal{S}_1^{(K)}, G, s) = 1$ too. This implies that $\{s \mid J_{\text{strong}}(\mathcal{S}_1^{(K)}, G, s) = 1\} = \{s \mid J_{\text{strong}}(\mathcal{S}_2^{(K)}, G, s) = 1\}$. The inductive step is similar, by substituting G with the set of states $\{s \mid J_{\text{strong}}(\mathcal{S}_2, G, s) = n - 1\}$.*

Corollary 3 *Let $\mathcal{H} = (X, U, Y, N)$ be a DTHS and let $\mathcal{L}_{\mathcal{H}}$ be its linearization. Let K be a solution for the DTLHS control problem $(\mathcal{L}_{\mathcal{H}}, I, G)$. Then K is a solution also for the DTHS control problem $(\mathcal{H}, I, G)$.*

Example 5 *The DTHS $\mathcal{H} = (X, U, \varnothing, N)$ model for the inverted pendulum in Ex. 1 contains the nonlinear function $\sin x_1$. We define the linearization $\mathcal{L}_{\mathcal{H}} = (X, U, Y, \tilde{N})$ as follows. In order to exploit sinus periodicity, we consider the equation $x_1 = 2\pi y_k + y_\alpha$, where y_k represents the period in which x_1 lies and $y_\alpha \in [-\pi, \pi]$ represents the actual x_1 inside a given period.*

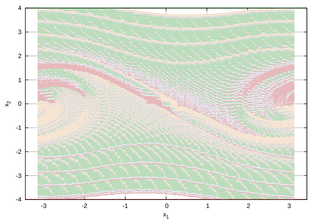

Figure 3: Controllable region for $F = 0.5$, $T = 0.1$, and $b = 9$.

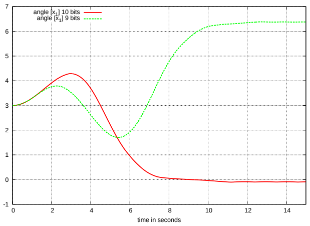

Figure 4: Trajectories for $\mathcal{H}^{(K^{(9)}_{0.5})}$ and $\mathcal{H}^{(K^{(10)}_{0.5})}$ starting from $(x_1, x_2) = (\pi, 0)$.

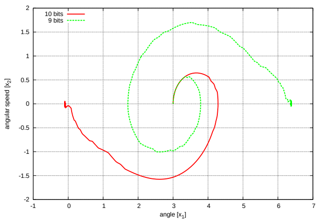

Figure 5: Same trajectories of Fig. 4 in the phases space.

This allows us to apply our linearization to $y_\alpha \in [-\pi, \pi]$ *only. We partition the interval* $[-\pi, \pi]$ *into four sub-intervals* I_1, I_2, I_3, I_4 *as shown in Fig. 2. For* $y_\alpha \in I_1 = [-\pi, -\frac{\pi}{2}]$ *we define* $f_1^+(y_\alpha)$ *as the line passing through points* $(-\pi, \sin(-\pi))$ *and* $(-\frac{\pi}{2}, \sin(-\frac{\pi}{2}))$*, i.e.* $f_1^+(y_\alpha) = -0.6369 y_\alpha + 2$*. Moreover, we define* $f_1^-(y_\alpha)$ *as the line which is tangent to the curve* $\sin y_\alpha$ *at* I_1 *medium point, i.e.* $f_1^-(y_\alpha) = 0.7073(y_\alpha + 0.785) - 0.7068$*. Functions* $f_2^\pm$, $f_3^\pm$ *and* $f_4^\pm$ *are obtained analogously.*

Finally, we have that $Y = Y^d \cup Y^r = \{y_k, y_q, z_1, z_2, z_3, z_4\} \cup \{y_\alpha\}$ *and* $\tilde{N} \equiv (x_1' = x_1 + 2\pi y_q + T x_2) \wedge (x_2' = x_2 + T\frac{g}{l} y_\alpha + T\frac{1}{ml^2} F u) \wedge x_1 = 2\pi y_k + y_\alpha \wedge \bigwedge_{i=1}^4 z_i \to f_i^- \leq y_\alpha \leq f_i^+ \wedge \bigwedge_{i=1}^4 z_i \to x_1 \in I_i \wedge \sum_{i=1}^4 z_i \geq 1$.

4.2 Linearization: a systematic approach

When nonlinear subexpressions are $\mathcal{C}^2$ functions, a systematic approach to compute linear overapproximations of a DTHS makes use of Taylor polinomial of degree 1 as piecewise affine functions that over- and under-approximate the value of a $\mathcal{C}^2$ function. Let $f(x)$ be a $\mathcal{C}^2$ function of n real variables over a given interval I. By Taylor's theorem, we may derive *linear* under- and over-approximations for $f(x)$ around a given point $x_0 \in I$ as follows. Namely, we have that exists $t \in [0, 1]$ such that $f(x) = f(x_0) + \triangledown f(x_0)(x - x_0) + \frac{1}{2}(x - x_0)^T H(x + t(x - x_0))(x - x_0)$, being H the Hessian matrix of f. If we know two real numbers m and M that are the minimum and the maximum value of $\frac{1}{2}(x - x_0)^T H(x + t(x - x_0))(x - x_0)$, in a given interval around x_0. In this case we can choose $f^+(x) = f(x_0) + \triangledown f(x_0)(x - x_0) + M$ and $f^-(x) = f(x_0) + \triangledown f(x_0)(x - x_0) + m$.

5 Experimental Results

In this section we present our experiments that aim at evaluating effectiveness of our linearization technique.

5.1 Experimental Settings

We present experimental results obtained by using QKS [23] on the inverted pendulum described in Example 1. In order to let QKS handle such a case study, we linearize the DTHS $\mathcal{H}$ in Example 1 with the DTLHS $\mathcal{L}_{\mathcal{H}}$ of Example 5. In all our experiments, as in [22] we set parameters l and m in such a way that $\frac{g}{l} = 1$ (i.e. $l = g$) and $\frac{1}{ml^2} = 1$ (i.e. $m = \frac{1}{l^2}$). As for the quantization, we set $A_{x_1} = [-1.1\pi, 1.1\pi]$ and $A_{x_2} = [-4, 4]$, and we define $A = A_{x_1} \times A_{x_2} \times A_u$. Moreover, we use uniform quantization functions dividing the domain of each state variable (x_1, x_2) into 2^b equal intervals, where b is the number of bits used by AD conversion. The resulting quantization is $\mathcal{Q}_b = (A, \Gamma_b)$, with $\|\Gamma_b\| = \frac{8}{2^b}$. Since we have two quantized variables (x_1, x_2) each one with b bits, the number of quantized (abstract) states is exactly 2^{2b}. Finally, the initial region I and goal region G are as in Ex. 3, thus the DTHS [DTLHS] control problem we consider is $P = (\mathcal{H}, I, G)$ $[(\mathcal{L}_{\mathcal{H}}, I, G)]$.

We run QKS for different values of the remaining parameters, i.e. F (force intensity), ρ (goal tolerance), T (sampling time), and b (number of bits of AD). For each of such experiments, QKS outputs a control software K in C language. In the following, we sometimes make explicit the dependence on F and b by writing $K_F^{(b)}$. In order to evaluate performance of K, we use an *inverted pendulum simulator* written in C. The simulator computes the next state by using Eq. (1) of Ex. 1, thus simulating a path of $\mathcal{H}^{(K)}$. Such simulator also implements the following features:

- random disturbances (up to 4%) in the next state computation are introduced, in order to assess K robustness w.r.t. non-modelled disturbances;
- Eq. (1) is translated into the discrete time version by means of a simulation time step T_s much smaller than the sampling time T used in $\mathcal{H}$ (and $\mathcal{L}_{\mathcal{H}}$). Namely, $T_s = 10^{-6}$ seconds, whilst $T = 0.01$ or $T = 0.1$ seconds. This allows us to have a more accurated simulation. Accordingly, K is called each 10^4 (or 10^5) simulation steps of $\mathcal{H}$. When K is not called, the last chosen action is selected again (*sampling and holding*).

All experiments have been carried out on an Intel(R) Xeon(R) CPU @ 2.27GHz, with 23GiB of RAM, Kernel: Linux 2.6.32-5-686-bigmem, distribution Debian GNU/Linux 6.0.3 (squeeze).

5.2 Underactuated Inverted Pendulum ($F = 0.5$)

In order to stabilize an *underactuated* inverted pendulum (i.e. when $F < 1$) from the hanging position to the uprigth position, a controller needs to find a non obvius strategy that consists of swinging the pendulum once or more times to gain enough momentum. We show that QKS is able to synthesize such a controller by running it on $\mathcal{L}_{\mathcal{H}}$ where $F = 0.5$ (note that in [22] $F = 0.7$). Results are in Tab. 1, where each row corresponds to a QKS run. Columns meaning in Tab. 1 are as follows. Columns b, T and ρ show the corresponding inverted pendulum parameters. Column $|K|$ shows the size of the C code for $K_{0.5}^{(b)}$. Finally, columns **CPU** and **RAM** show the computation time (in seconds) and RAM usage (in KB) needed by QKS to synthesize $K_{0.5}^{(b)}$.

As for $K_{0.5}^{(b)}$ performance, it is easy to show that by reducing the sampling time T and the quantization step (i.e. increasing b), we increase the quality of $K_{0.5}^{(b)}$ in terms of ripple, set-up time and coverage. In fact, Fig. 4 shows the simulations of $\mathcal{H}^{(K_{0.5}^{(9)})}$ and $\mathcal{H}^{(K_{0.5}^{(10)})}$. As we can see, $K_{0.5}^{(10)}$ drives the system to the goal with a smarter trajectory, with one swing only. This have a significant impact on the set-up time (the system stabilizes after about 8 seconds when controlled by $K_{0.5}^{(10)}$ instead of about 10 seconds required when controlled by $K_{0.5}^{(9)}$). Fig. 3 shows that the *controllable region* of $K_{0.5}^{(9)}$ (i.e., $\mathrm{dom}(K_{0.5}^{(9)})$) covers almost all states in the admissible region that we consider. Different colors mean different set of actions enabled by the controller. We observe that the mgo solution enables more than one action in a significant portion of the controllable region. The control software, however, is generated in such a way that one action is chosen in each state. Finally, Fig. 10 shows the ripple of x_1 for $\mathcal{H}^{(K_{0.5}^{(10)})}$ inside the goal. Note that such ripple is very low (0.018 radiants).

5.3 Very Underactuated Inverted Pendulum ($F = 0.3$)

We succeeded to find controllers for the inverted pendulum for any value of F down to 0.3, with $T = 0.1$ seconds and $\rho = 0.1$. However, simulations show that

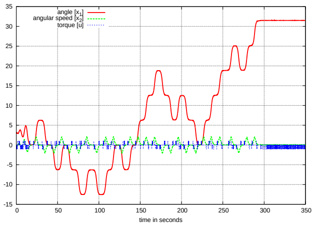

Figure 6: Simulation for $\mathcal{H}^{(K_2^{(11)})}$ starting from $(x_1, x_2) = (\pi, 0)$.

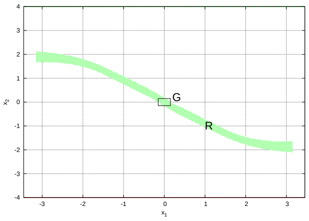

Figure 7: States turned directly to the goal with $F = 0.3$.

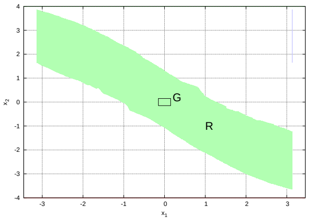

Figure 8: States turned directly to the goal with $F = 2$.

the behaviour of the resulting closed loop system is somewhat puzzling. As it is shown in Fig. 6 for $\mathcal{H}^{(K_{0.3}^{(11)})}$, after three swings the pendulum is correctly driven to the goal, but at that point the controller is not able to maintain the plant inside the goal. In fact, the controller let the pendulum fall and makes it do a complete round in order to reach again the upright position. This behaviour is repeated 27 times, before the $K_{0.3}^{(11)}$ makes pendulum stabilize into the goal region.

As already noted in [22], all controllers for underactuated pendulum use two very different strategies to stabilize the system depending on the initial state. When the angle is positive and the speed is negative (and in a suitable range that depends on F), the controller turns directly the pendulum into the upright position. Symmetrically, this also happens when the angle is negative and the speed is positive. Otherwise the controller let the pendulum fall down to gain enough momentum (or to smoothly slow down it). Therefore, starting from very near states may lead the system to follow very different trajectories. Reducing F squeezes the region of states from which the pendulum is directly turned into the upright position. As Fig. 7 shows, when F is equal to 0.3, we have a rather pathological situation: the frontier between the two strategies lies *inside* the goal region. The controller sometimes is unable to keep the system inside the goal, because disturbances introduced by the simulator make the system cross the frontier between the two strategies. When this frontier lies far enough from the goal (see Fig. 8 for the case $F = 2$), this phenomenon is essentially harmless and leads, at worst, to suboptimal strategies.

5.4 Overactuated Pendulum ($F = 2$)

When F is greater than 1, finding a control strategy is less challenging. It is worth noting however that, even in this case, our approach allows us to find controllers

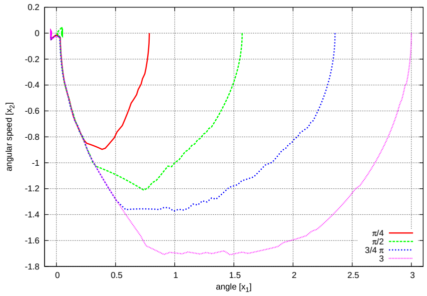

Figure 9: $\mathcal{H}^{(K_2^{(11)})}$ phases space (starting from $x_1 \in \{\frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, 3\}$).

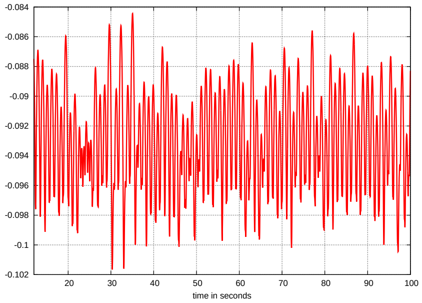

Figure 10: Ripple of x_1 for $\mathcal{H}^{(K_{0.5}^{(10)})}$.

Table 1: Experimental Results for inverted pendulum with $F = 0.5$.

b	T	ρ	$\lvert K \rvert$	CPU	MEM
8	0.1	0.1	2.73e+04	2.56e+03	7.72e+04
9	0.1	0.1	5.94e+04	1.13e+04	1.10e+05
10	0.1	0.1	1.27e+05	5.39e+04	1.97e+05
11	0.01	0.05	4.12e+05	1.47e+05	2.94e+05

that hardly can be synthesized by means of traditional analytical methods. In Fig. 9, we show trajectories in the phases space of $\mathcal{H}^{(K_2^{(11)})}$ with $T = 0.01$ seconds, $\rho = 0.05$, and starting values for x_1 are in $\{\frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, 3\}$ and $x_2 = 0$. $\mathcal{H}^{(K_2^{(11)})}$ follows highly non-smooth trajectories: $K_2^{(11)}$ drives the system along an optimal approach to the goal. Before joining this ideal path to the goal, the controller, in order to optimize the set up time, drives the system at the maximum possible "cruising" speed that allows the pendulum to be stopped in the goal. For higher values of F, this cruising speed is even higher.

6 Conclusions

We presented an automatic methodology to sinthesize control software for nonlinear Discrete Time Hybrid Systems. The control software is correct-by-construction with respect both System Level Formal Specifications of the closed loop system and Implementation Specification, namely the quantization schema. Our experimental results on the inverted pendulum benchmark show the effectiveness of our approach and that we synthesize near optimal controllers that hardly can be designed by using traditional analytical methods of Control Engineering.

The present work can be extended in several directions. First of all, it would be interesting to consider control synthesis of controllers that are optimal with respect a cost function given as input of the control problem, rather than simply time-optimal. Another natural possible future research direction is to investigate fully symbolic control software synthesis algorithms based, for example, on efficient quantifier elimination procedures, in order to efficiently deal with Hybrid Systems with several continuous state variables.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3 – 34, 1995.

[2] R. Alur, T.H. Henzinger, G. Lafferriere, G. Pappas. Discrete abstractions of hybrid systems. *Proc. of the IEEE*, 88(7):971–984, 2000.

[3] R. Alur, T. Dang, F. Ivančić. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. on Embedded Computing Sys.*, 5(1):152–199, 2006.

[4] R. Alur, T.A. Henzinger, P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Softw. Eng.*, 22(3):181–201, 1996.

[5] R. Alur, P. Madhusudan. Decision problems for timed automata: A survey. In *SFM*, LNCS 3185, pp. 1–24, 2004.

[6] P. Attie, A. Arora, E.A. Emerson. Synthesis of fault-tolerant concurrent programs. *ACM Trans. on Pr. Lang. Syst.*, 26(1):125–185, 2004.

[7] A. Bemporad. Hybrid Toolbox - User's Guide, 2004. http://www.ing.unitn.it/~bemporad/hybrid/toolbox.

[8] A. Bemporad, N. Giorgetti. A sat-based hybrid solver for optimal control of hybrid systems. In *HSCC*, LNCS 2993, pp. 126–141, 2004.

[9] M. Benerecetti, M. Faella, S. Minopoli. Revisiting synthesis of switching controllers for linear hybrid systems. In CDC-ECC, pp. 4753–4758, 2011.

[10] W.L. Brogan. *Modern control theory (3rd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[11] A. Cimatti, M. Roveri, P. Traverso. Strong planning in non-deterministic domains via model checking. In *AIPS*, pp. 36–43, 1998.

[12] G. Della Penna, D. Magazzeni, A. Tofani, B. Intrigila, I. Melatti, E. Tronci. *Automated Generation of Optimal Controllers through Model Checking Techniques*, volume 15 of *LNEE*. Springer, 2008.

[13] G. Frehse. Phaver: algorithmic verification of hybrid systems past hytech. *Int. J. Softw. Tools Technol. Transf.*, 10(3):263–279, 2008.

[14] M. Fu, L. Xie. The sector bound approach to quantized feedback control. *IEEE Trans. on Automatic Control*, 50(11):1698–1711, 2005.

[15] A. Girard. Synthesis using approximately bisimilar abstractions: time-optimal control problems. In CDC, pp. 5893 –5898, 2010.

[16] T.A. Henzinger, P.-H. Ho, H. Wong-Toi. Hytech: A model checker for hybrid systems. *STTT*, 1(1):110–122, 1997.

[17] T.A. Henzinger, P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In *CAV*, LNCS 939, pp. 225–238, 1995.

[18] T.A. Henzinger, B. Horowitz, R. Majumdar, H. Wong-Toi. Beyond hytech: Hybrid systems analysis using interval numerical methods. In *HSCC*, LNCS 1790, pp. 130–144, 2000.

[19] T. A. Henzinger, P. W. Kopke. Discrete-time control for rectangular hybrid automata. In *ICALP*, pp. 582–593, 1997.

[20] T.A. Henzinger, P.W. Kopke, A. Puri, P. Varaiya. What's decidable about hybrid automata? *J. of Comp. and Sys. Sc.*, 57(1):94–124, 1998.

[21] T.A. Henzinger, J. Sifakis. The embedded systems design challenge. In *FM*, LNCS 4085, pp. 1–15, 2006.

[22] G. Kreisselmeier, T. Birkhölzer. Numerical nonlinear regulator design. *IEEE Trans. on on Automatic Control*, 39(1):33–46, 1994.

[23] F. Mari, I. Melatti, I. Salvo, E. Tronci. Synthesis of quantized feedback control software for discrete time linear hybrid systems. In *CAV*, LNCS 6174, pp. 180–195, 2010.

[24] F. Mari, I. Melatti, I. Salvo, E. Tronci. Quantized feedback control software synthesis from system level formal specifications. *CoRR*, abs/1107.5638v1, 2011.

[25] M. Mazo, A. Davitian, P. Tabuada. Pessoa: A tool for embedded controller synthesis. In *CAV*, LNCS 6174, pp. 566–569, 2010.

[26] G. Pola, A. Girard, P. Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516, 2008.

[27] P. Tabuada. Approximate simulation relations and finite abstractions of quantized control systems. In *HSCC*, pp. 529–542, 2007.

[28] C. Tomlin, J. Lygeros, S. Sastry. Computing controllers for nonlinear hybrid systems. In *HSCC*, LNCS 1569, pp. 238–255, 1999.

[29] E. Tronci. Automatic synthesis of controllers from formal specifications. In *ICFEM*, pp. 134–143. IEEE, 1998.

[30] H. Wong-Toi. The synthesis of controllers for linear hybrid automata. In *CDC*, pp. 4607–4612 vol. 5. IEEE, 1997.

[31] B. Yordanov, J. Tumova, I. Cerna, J. Barnat, C. Belta. Temporal logic control of discrete-time piecewise affine systems. *To Appear in IEEE Transactions On Automatic Control*, 2012.