2012 15th Euromicro Conference on Digital System Design

Open Problems in Verification and Refinement of Autonomous Robotic Systems

Davide Bresolin, Luigi Di Guglielmo, Luca Geretti, Riccardo Muradore, Paolo Fiorini and Tiziano Villa
Dipartimento di Informatica – Università di Verona, Italy
{davide.bresolin, luigi.diguglielmo, luca.geretti, riccardo.muradore, paolo.fiorini, tiziano.villa}@univr.it

***Abstract*—The relevance of formal verification methods is widely recognized in the computer science and embedded systems community. Recently, such methods have been introduced also within the control community, to help designers in developing control architectures for complex robotics systems. Robotic systems typically mix continuous and discrete behaviors that cannot be modeled faithfully using neither continuous-only nor discrete-only formalisms. The interaction of continuous and discrete dynamics makes the formal treatment of this kind of systems computationally very demanding, and justifies the need of studying new methods and algorithms. In this paper, we outline the current state-of-the-art, and describe some open problems in verification, refinement and implementation of autonomous robotic systems. We motivate the relevance of our analysis by means of an Autonomous Robotic Surgery test case.**

***Index Terms*—Hybrid systems, Autonomous robotic systems, Formal verification, Refinement, Implementation**

I. Introduction

In engineering practice, the analysis of a complex system is usually carried out via simulation, exploring one of the possible system executions at a time, similarly to how software testing is performed. On the other hand, formal verification methods aim at exploring all possible executions of the system, in order to be certain that a property of interest holds in all cases. In this approach the properties that the system must respect are expressed in some logical formalism, while the design under verification is modeled as a finite-state transition system. This reduces the verification problem to the problem of establishing the truth of a formula over a model, through an exhaustive exploration of the states reachable by the system and its related behaviors. Nowadays, model checking and other formal verification methods are common practice in the design flow of many hardware and software producers, and their importance for automating design, verification and refinement of embedded systems is widely recognized in the scientific community.

Formal verification methods have been recently introduced also within the control community [1], to help designers in developing control architectures for complex robotics systems. Robotic systems are a typical example of *hybrid systems*, mixing continuous and discrete behaviors that cannot be modeled faithfully using continuous-only or discrete-only formalisms. The presence of mixed dynamics makes the formal treatment of this kind of systems considerably hard, thus justifying the need of researching novel approaches that are both more efficient and more effective.

This research was partly supported by the EU projects FP7-ICT-223844 CON4COORD and FP7-ICT-270396 I-SUR

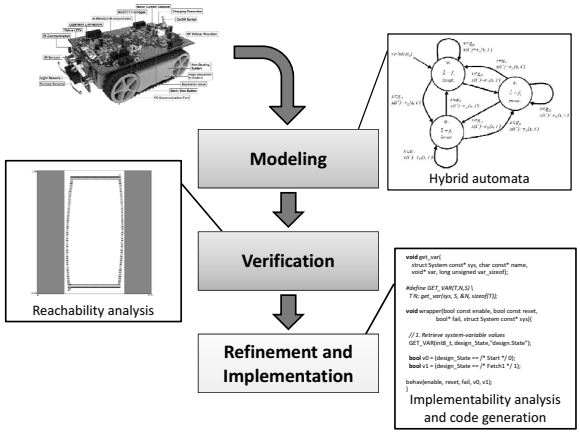


Fig. 1. Verification and refinement of robotic systems.

A simple, yet powerful, formalism for the modeling of hybrid systems is the one of *hybrid automata* [2], which extends the usual definition of finite state automata with continuous variables that evolve according to dynamics characterizing each discrete state. In the last years, a wide spectrum of algorithmic techniques has been studied in the control and computer science communities to solve the problems of simulation, verification and control synthesis for hybrid automata. Current state-of-the-art tools can verify hybrid systems with complex nonlinear dynamics [3], [4], or linear systems with a large number of continuous variables [5], thus becoming of interest also for real test-cases and application domains.

Another phase of the design flow where the interplay of continuous and discrete behaviors makes things complicated is the refinement and implementation of hybrid models. Indeed, formal verification techniques for hybrid automata usually rely on unimplementable assumptions, such as the *synchrony hypothesis*, i.e., the capability of performing any computation in zero time units and forcing a change in the dynamics of the hybrid system with no delays. Thus, the verification results on the correctness of the ideal model of the system cannot be directly applied to a real implementation. For this reason, new semantics for hybrid automata, which do not rely on synchrony or other unrealistic assumptions, have been proposed in the literature [6], [7]. By formally verifying the correctness of the system using such semantics, it is possible to determine

978-0-7695-4798-5/12 $26.00 © 2012 IEEE
DOI 10.1109/DSD.2012.96

the *performance bounds* to be satisfied by any conservative concrete hardware/software device that implements the system.

In this paper, we outline the current state-of-the-art, and describe some open problems in verification, refinement and implementation of autonomous robotic systems. We motivate the relevance of our analysis by a test case coming from a challenging frontier of modern robotics, namely, Autonomous Robotic Surgery.

II. Modeling with hybrid automata

A hybrid system is a system whose behaviors cannot be characterized faithfully using either discrete or continuous models. It consists of a discrete part that operates in a continuous environment, and for this reason, it is sensitive not only to time-driven phenomena but also to event-driven ones.

Hybrid automata in particular are a powerful modeling formalism for the design and verification of hybrid systems. Intuitively, a hybrid automaton is a finite-state automaton with continuous variables that evolve according to dynamics specified at each discrete node (or *location*). A state of a hybrid automaton is defined as a pair $\langle l; v \rangle$, where l is a location and $v \in \mathbb{R}^n$ is an assignment of values for the continuous variables (here n is the number of continuous variables of the automaton). An execution of a hybrid automaton corresponds to a sequence of transitions alternating continuous and discrete evolution. In a continuous evolution step, the location does not change while time passes and the evolution of the state variables follows the dynamic law associated with the current location. A discrete evolution step instead consists of a change of the current location and possibly of the value of the state variables, in accordance with a reset function associated with the transition. The interleaving of continuous and discrete evolution steps is decided by the invariant of the location, which must be true for the continuous evolution to go on, and by the guard predicates, which must be true for a discrete transition to be activated. Guards and invariants are not necessarily complements of each other: when both the invariant and one or more guards are true, both the continuous evolution and activation of discrete transitions are allowed, and the behavior of the automaton becomes nondeterministic.

III. Formal verification of hybrid automata

Of particular importance in the verification of hybrid systems is the computation of the reachable set [8], which consists of all the states that can be reached under the dynamical evolution from a given initial state set. In the following, given a hybrid automaton H, we will distinguish between two kinds of reachable sets:

- *Finite-time reachable set*: given a set of states X_0 and a time point t, we denote with $ReachSet_H(X_0, t)$ the set of all states that can be reached from X_0 by a finite trajectory within the $[0, t]$ time interval;
- *Infinite-time reachable set*: given a set of states X_0, we denote with $ReachSet_H(X_0)$ the set of all states that can be reached from X_0 by a trajectory of arbitrary time length.

Finite-time and infinite-time reachability are the main tools to verify properties of hybrid automata. Indeed, checking safety properties (i.e., "something bad never happens") reduces to the infinite-time reachability problem. Suppose we wish to verify that a safety property φ holds for a hybrid automaton H; in other words, that φ remains true for all possible executions starting from a set X_0 of initial states. Then we only need to prove that $ReachSet_H(X_0) \subseteq Sat(\varphi)$, where $Sat(\varphi)$ is the set of states where φ is true. Finite-time reachability can be used to verify timed safety properties, that is, whether the property φ is true at some execution time t. Finally, by pairing finite-time and infinite-time reachability, also timed stability properties can be verified, by checking whether φ is always true after some execution time t. Other properties, such as liveness (i.e., "something good eventually happens") and general stability properties (i.e., "the system eventually reaches the good region and never leaves it"), require more complex techniques to be verified, based on transformations on the model [9], [10].

Unfortunately, the reachability problem is not decidable in general: there is no algorithm that can compute $ReachSet_H(X_0)$ in an exact way without imposing strong restrictions on the dynamics of the automaton [11]. In particular, the results in [12]–[14] suggest that except under very restrictive settings, it is not possible to attain decidability without imposing that the continuous variables be reset to a constant value after *every* discrete transition. From a modeling point of view, this severely restricts the expressiveness of the formalism. Indeed, when the continuous variables represent physical quantities, as in the case of digital systems that interact with plants through sensors and actuators, this resetting restriction is completely unrealistic, since the values of physical quantities usually evolve continuously and thus are not reset by discrete transitions.

Nevertheless, formal verification methods can be applied to hybrid automata also in the general case. Suppose we can compute an over-approximation S to $ReachSet_H(X_0)$, i.e., a set $S \supseteq ReachSet_H(X_0)$. Then if S is a subset of $Sat(\varphi)$, so is the reachable set, and the automaton H respects the property. Conversely, if we can compute an under-approximation S to $ReachSet_H(X_0)$ (that is, a set $S \subseteq ReachSetH(X_0)$) that contains at least one point outside $Sat(\varphi)$, we have proved that H does not respect the safety property φ.

For these reasons, many tools for reachability analysis of hybrid automata are based on approximation techniques. The challenge in this case is to find the best approximations of the reachable set. HyTech [15] computes approximations to the reachable set for hybrid automata with piecewise-constant continuous dynamics. VeriShift [16], PHAVer [17] and SpaceEx [5] can handle affine dynamics and guards. CheckMate [18] and HSOLVER [3] are able to deal with nonlinear dynamics, but can verify safety properties only. KeYmaera [19] uses automated theorem proving techniques, and performs parametric verification of nonlinear hybrid sys-

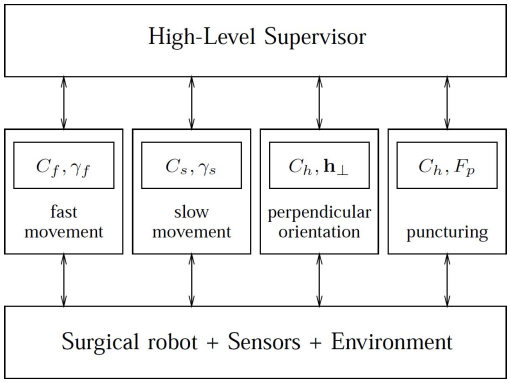


Fig. 2. High-level control architecture.

tems. Finally, ARIADNE [4], [20], [21] exploits a rigorous *function calculus* to achieve upper- and lower- approximations of the reachable set of nonlinear hybrid systems [22], [23] and perform parametric proving and disproving.

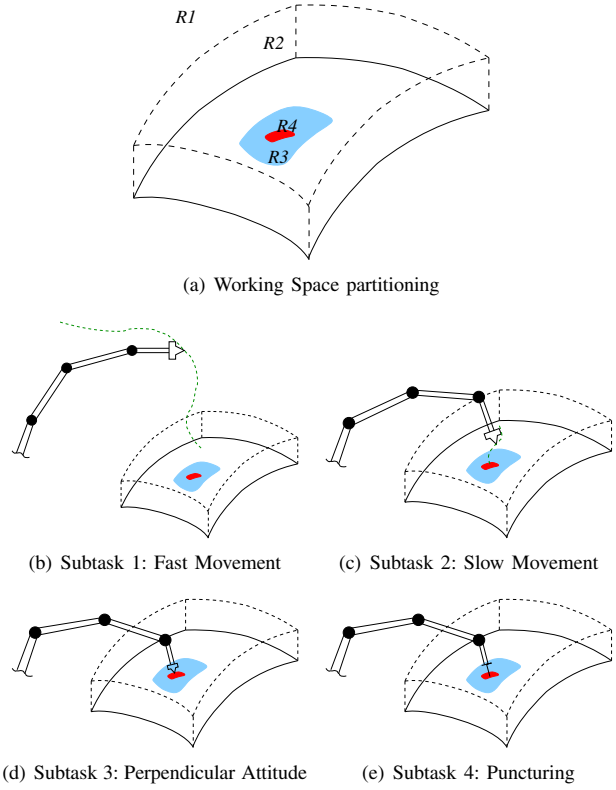
IV. Verification of Complex Systems: the Robotic Surgery Test Case

In the last few decades, robotics played a relevant role in the progress of surgery. Teleoperated surgery has already proved its advantages by improving safety, accuracy, reproducibility, and decreasing human fatigue, but more is expected when advanced features, such as force feedback, will become commonplace in commercial robotic systems [24]–[27]. Another challenging frontier is Autonomous Robotic Surgery (ARS), whose objective is to perform simple tasks without the direct intervention of surgeons [28], allowing them to focus on the most difficult aspects of the intervention only. Moreover, thanks to the dexterity of current robots, it is well possible to accomplish several simple tasks with improved precision and speed.

In order to make these new technologies acceptable for patients and surgeons, ARS demands methods and models for assessing its quality, especially in terms of security and robustness w.r.t. disturbances and unexpected events. Proper design of such critical systems can greatly benefit from the availability of methodologies and software tools that can automatically analyze or even synthesize parts of the system, while providing definite guarantees on the accuracy of the analysis and on the performance of the synthesized modules. A first step in this direction is [29], where formal verification methods based on reachability analysis have been successfully applied to the problem of designing an overall control architecture that guarantees the successful accomplishment of a simple surgical task, independently from uncertainties and unmodeled subsystems.

A. Modeling

A robotic manipulator consists of a discrete control part that operates in a continuous environment, and thus is a typical example of a hybrid system. Hence, hybrid automata theory provides the proper mathematical tools to model and



(a) Working Space partitioning

(b) Subtask 1: Fast Movement

(c) Subtask 2: Slow Movement

(d) Subtask 3: Perpendicular Attitude

(e) Subtask 4: Puncturing

Fig. 3. Surgical action subtasks.

control surgical actions by means of a suitable supervisor automaton. The automaton will specify the discrete task states (e.g., tissue approach, contact and cut), continuous dynamics within each state, and transitions between states. Adaptation mechanisms can also be introduced into the automaton so that state parameters can reflect the real values of the environment parameters. The control of the surgical action will be aware of the current medical situation, and therefore, it will interact with the intraoperative reasoning process.

The surgical task that we analyze in this paper is the same considered in [29]: puncturing, that is, the act of penetrating a biological tissue with a needle. To obtain a sensible model of the task, we first decompose it into a sequence of atomic subtasks that should be correctly accomplished to guarantee success of the overall action. The high level architecture of the model is shown in Figure 2. The high-level supervisor is responsible for switching between four different subtasks, according to the nominal operation conditions, to the operation constraints (maximum speed, force, etc.), and to the partition of the workspace depicted in Figure 3(a):

R_1: region far away from the patient;
R_2: region near the patient's tissue;
R_3: surface on the tissue where the tool could touch the patient due to tracking errors and/or unmodeled dynamics;
R_4: target surface on the tissue. It is assumed to be surrounded by the region R_3.

Every subtask is defined by a local controller that is respon-

sible for controlling the execution of the atomic actions. The ARS procedure goes through the following phases:

- **Fast Movement**: in this phase, the end-effector of the robot should approach the patient's tissue starting from its home position and following some reference trajectory γ_f. Since the manipulator is assumed to be far away from the patient at the beginning (region R_1), this movement can be relatively fast (Figure 3(b)). The controller responsible for following γ_f is C_f;
- **Slow Movement**: when the end-effector is close to the tissue (region R_2), the movement should slow down in order to approach the target position on the patient by reducing the tracking error and improving robustness (Figure 3(c)). The corresponding controller and reference trajectory in this phase are C_s and γ_s;
- **Perpendicular Attitude**: as needle deflection and tissue deformation are major problems for accurate puncturing, with no loss of generality we assume that the end effector of the surgical robot needs to be perpendicular to the tissue before puncturing. While keeping the touching point fixed, the robot should move its wrist in order to have the tool orthogonal to the surface (Figure 3(d)). The controller C_h has to align the force $\mathbf{h}$ at the end effector to the nominal force $\mathbf{h}_\perp$, where the only non-zero component should be the force along the needle direction;
- **Puncturing**: the last subtask consists of changing the tool at the end effector and executing the puncturing. The robot should remove the force sensor and insert the mechanical structure hosting the needle (Figure 3(e)).

The hybrid automaton that describes the puncturing test bench is pictured in Figure 4. Each location of the automaton corresponds to one of the subtasks identified above, with the exception of location *stop*, that represents the successful accomplishment of the complete task. Double lines enclose the initial location *fast*. Transitions describe the switching from one subtask to another and yield a sequence of atomic actions that models the complete surgical action, and are defined as follows:

- the transition from *fast* to *slow* becomes active when the Cartesian coordinate $\mathbf{x}$ of the end effector computed by using the direct kinematic tells us that the end effector is now inside R_2;
- the transition from *slow* to *perp* is activated as soon as the end effector touches the tissue, i.e., when the force $\mathbf{h}$ at the end effector becomes positive;
- the manipulator remains in location *perp* until the tool is perpendicular to the tissue $\mathbf{h} = \mathbf{h}_\perp$, then the transition is activated and the automaton moves to *puncturing*;
- the last transition becomes active after the puncturing action is executed.

B. Formal Verification

Once a sensible model of the system has been defined, formal verification techniques can be applied to prove that the planned sequence of subtasks can successfully accomplish the surgical action, by expressing the requirements of interest as questions about reachability properties of the hybrid automaton model. In [29], the following properties for the automatic puncturing test case have been analyzed:

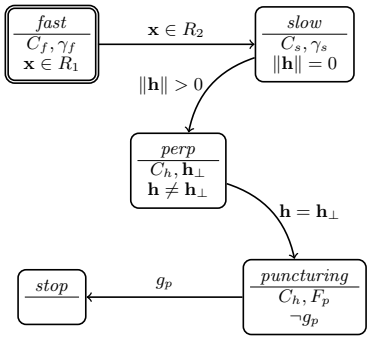


Fig. 4. Automaton for the surgical test bench.

P1 The force applied to the patient by the end effector is always less than a given threshold, except for the puncturing subtask.

P2 The task is feasible, and the position of the needle at the end of the task is always inside the target region R_4.

By using the library ARIADNE, the authors determined how the value of some design parameters, like the uncertainty on the position of the patient and the proportional gain of the force loop in the controller of the *perp* location, affected the truth of properties **P1** and **P2**. The parametric analysis yielded intervals on the chosen parameters where the properties held.

V. Open Problems

The results obtained in [29] are potentially interesting to engineers and technicians to assess the correctness of the surgical system and to identify design changes that help meeting the specifications.

Unfortunately this analysis, like any analysis based on the ideal model of hybrid automata, is based on several assumptions (working hypotheses) that cannot be guaranteed using real devices. Indeed, hybrid automata can be considered perfect and instantaneous devices. They adopt a notion of time and evaluation of continuous variables based on dense sets of values (usually the set of real numbers $\mathbb{R}$). Thus, they can sample the state of the hybrid system at any instant in time, and are capable of instantaneously evaluating guard predicates and reacting by performing changes in the operating mode of the system without any delay.

While these aspects are convenient at the modeling level, any system that relies for its correctness on such precision and instantaneity cannot be implemented by any hardware/software device, no matter how fast it is. In a typical implementation of a robotic system, instead, the supervisor samples the state of the system at periodic discrete time instants which are typically given by the clock frequency of the processor that implements the supervisor. The values of the relevant variables are observed by sensors, that are affected by noise and have a finite precision. Finally, the output signals generated by the

supervisor, once transmitted to the actuators, will effect the desired change in the operating mode only after a small (but not zero) delay.

For instance, according to the automaton in Figure 4, it is clear that the transition from *fast* to *slow* cannot be instantaneous because the position of the end-effector is derived from the measurements obtained from the robot encoders (through the forward kinematics). Such encoders have a finite sample rate and their values are quantized and affected by noise. This implies that the transition to the state *slow* happens with a certain delay and with an intrinsic (and unavoidable) uncertainty in space. Similarly, the transition from *slow* to *probing* depends on the accuracy of the force sensor: the measurement noise does not allow the transition to be triggered by the "ideal condition" $\|\mathbf{h}\| > \mathbf{0}$. A threshold has to be defined according to the quality of the sensor (response time and noise) and on the specific application.

These basic observations are given only to highlight how seemingly harmless assumptions, taken for granted during the verification procedure, can yield large and unacceptable errors in the final implementation of the system. The discrepancy between the ideal system model and real system implementations can invalidate the formal verification completely. Dropping the infinite accuracy and instantaneous reaction hypotheses can affect not only the truth of the properties of interest, but also the stability of the whole system, that is not guaranteed anymore when there are delays in the switching between the different local controllers. Consequently, it is of paramount importance to refine the results obtained during the verification phase by removing the previous assumptions and quantifying their effects analytically.

This is particularly true in the surgical framework we are working in, and motivates the need of a rigorous and accurate post-design refinement phase which can guarantee that the formal results obtained in the verification phase still hold for the real implementation of the control architecture. In addition, this phase can also be used as a design tool to drive the choice of sensors and actuators.

VI. Refinement and implementation of hybrid automata

To handle the semantic gap between the high level model and its corresponding implementation described in Section V, a number of alternative semantics for hybrid automata have been proposed in the literature.

One of the first proposals [30] introduces a digitalized semantics for timed automata based on the non-instantaneous observability of events. This semantics models the fact that any implementable control strategy cannot observe the physical environment continuously, but only at discrete time instants. The authors consider timed automata as timed specifications of the hybrid system and study in what sense the corresponding timed traces can be recognized by a digital controller. To do this, they introduce the concept of *time-triggered automata* . A time-triggered automaton is essentially a time table for a digital supervisor describing what the supervisor should do at a given time point. The advantage of such time-triggered automata is that they can be easily transformed into executable programs (i.e., the concrete control strategies). The authors prove as main result that one can effectively decide whether a time-triggered automaton correctly recognizes the timed traces, i.e., the time-triggered automaton implements the timed specification of the hybrid system. Unfortunately, the authors underline that the systematic synthesis of time-triggered automata recognizing the required set of timed traces is still an open issue.

An alternative semantics for timed automata that allows to synthesize an implementable control strategy in a systematic way is proposed in [6]. This new semantics, named Almost-As-Soon-As-Possible (AASAP), relies on the continuous time behavior and infinite precision of hybrid automata. However, it takes into account the digital and imprecise aspects of the hardware on which the timed automaton $\mathcal{C}$ representing the supervisor is being executed, by relaxing the "synchrony hypothesis": the semantics allows the supervisor to react within Δ time units when a synchronization or a control action has to take place, rather than instantaneously. The designer acts as if the synchrony hypothesis were true, i.e., he/she models the environment $\mathcal{E}$ and the supervisor $\mathcal{C}$ without referring to the reaction delay. Such delay is taken into account during the verification/refinement phase: the authors use reachability analysis to look for the largest value Δ for which the relaxed supervisor is still receptive w.r.t. the environment in which it will be embedded, and it is still correct w.r.t. the properties that the original instantaneous model $\mathcal{C}$ has to enforce. Such a Δ-relaxed controller represents an implementable control strategy if $\Delta > 0$. A complete tool chain, implementing this approach for the refinement of embedded systems, where the environment can be a general hybrid automata (and not only a timed automata, as in the original work), has been recently presented in [31]. However, it is worth noting that the AASAP approach requires the supervisor $\mathcal{C}$ and environment $\mathcal{E}$ to be modeled as separate automata interacting with each other by means of synchronization events only. Moreover, since there are no restrictions on the dynamics of the environment, the problem of synthesizing such a value Δ may not be decidable.

The work in [7] proposes a similar approach for synthesizing implementable control strategies from timed automata. Given a timed automaton $\mathcal{C}$ modeling a control strategy, the authors look for the existence of a new timed automaton $\mathcal{C}'$ whose behaviors satisfy the properties of $\mathcal{C}$ and are implementable, i.e., they do not rely on the synchrony hypothesis. To identify such an automaton, the authors try to derive $\mathcal{C}'$ from $\mathcal{C}$. In particular, $\mathcal{C}'$ is obtained by shrinking the guards of the original automaton $\mathcal{C}$ so that, by assuming bounded reaction delays for synchronization and control actions, all behaviors of $\mathcal{C}'$ are non-blocking (i.e., shrinkability problem). This means that all timing requirements satisfied by the instantaneous automaton $\mathcal{C}$, such as critical deadlines, are strictly respected by $\mathcal{C}'$. In general, shrinking a model $\mathcal{C}$ may remove too many behaviors and even introduce deadlocks. Thus, constraints to guide the shrinking process are required. For this reason, the

authors have theoretically investigated the constraints which guarantee the preservation of the non-blocking behaviors in $\mathcal{C}'$, and have shown that deciding the shrinkability problem of a timed automaton $\mathcal{C}$ can be checked in EXPTIME.

Notice that modifying the semantics may not be the only way to enforce the implementability. Indeed, in [32] the authors ask the question whether similar results can be obtained without introducing a new semantics, but acting on modeling instead, thanks to the introduction of new assumptions on the program type or execution platform by means of changes on the corresponding models. The authors propose an implementation methodology for timed automata which allows to transform a timed automaton into a program and to check whether the execution of this program on a given platform satisfies a desired property. Unlike the works in [6] and [7], an open problem of this approach is how to guarantee that, when a platform P is replaced by a "better" platform P', a program proved correct for P is also correct for P'. The authors reported examples where this does not hold for a reasonable assumption of a "better" platform, namely, when P and P' are identical, but P' provides a periodic digital clock running twice as fast as the one of P. The reason is that a program using the faster clock has a higher "sampling rate" and thus may generate more behaviors than a program using the slower clock, leading to a violation of the properties.

A. Refinement using the AASAP approach

Among all the approaches summarized above, only the works in [6], [31] provide practical solutions to the refinement problem. In particular, in [6] it is formally proved that by verifying the correctness of the system using the AASAP semantics, it is possible to determine if the supervisor of the modeled hybrid system can be refined. In fact, a concrete supervisor realization can only take a control decision as soon as it detects the required conditions through sensors. Only at that point, after small computation and actuation delays, it is able to carry out the necessary actions to the controlled plant. Thus, the supervisor can only perform a control switch almost as soon as possible.

Due to the fact that none of the existing analysis and verification tools for hybrid systems is able to natively verify hybrid models against safety properties using the AASAP semantics, the only way to work around this limitation is to transform the original hybrid model. The transformed model, interpreted using the classical semantics for hybrid automata, represents a conservative abstraction of the original model interpreted using the AASAP semantics, and can be verified without the need to develop an ad-hoc tool. In particular, this transformation consists of:

- *the relaxation of the supervisor clock precision*: clocks (i.e., continuous variables measuring the elapsing of time) can be modeled only with a finite precision and consequently they are rounded according to the HW platform characteristics;
- *the relaxation of the instantaneousness of the supervisor regarding the reaction to clock timeouts and events*: any reaction to a timeout and an incoming or outgoing event introduces delays that depend on the HW platform characteristics.

In the new *relaxed model* the idea of *almost as soon as possible* is modeled by the parameter Δ, representing both an upper-bound on the delays for sensing and emitting synchronization events from the supervisor to the controlled plant, and an upper-bound on the imprecision of the clocks. In particular:

- Δ *relaxes the clocks precision*. The guards of the transitions that involve the evaluation of the continuous variables are parameterized by Δ, simulating rounding of clocks;
- Δ *relaxes the reactions to timeouts*. Any transition that can be taken by the automaton becomes urgent after a small delay modeled by the parameter Δ;
- Δ *relaxes the reactions to events*. A distinction is made between the occurrence of an event in the sender (occurrence) and the acknowledgement of the event by the receiver (perception). The time difference between occurrence and perception of an event is bounded by Δ.

Notice that the relaxed model exhibits a superset of the original behavior, the latter corresponding to a relaxed value $\Delta = 0$ for the parameter. Consequently, in order for the relaxed model to be *implementable* at all, a necessary condition is that there exists a value $\Delta > 0$ for which the required properties are satisfied. Moreover, by determining the maximum value for Δ such that the relaxed model still satisfies the required properties, it is possible to know how much the model behavior can be relaxed while preserving the system correctness.

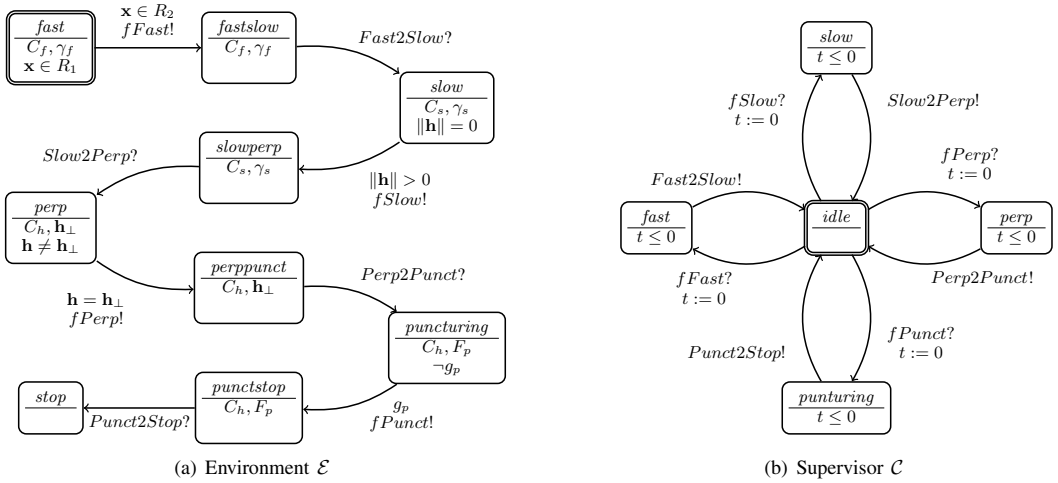
Currently, the AASAP approach requires the supervisor to be modeled by a special kind of timed automata, named *Elastic controllers*, featuring the following restrictions:

1) only urgent transitions are allowed;
2) the guards must be closed expressions;
3) communication with the environment is allowed only through events.

It must be remarked that, while the restriction on the continuous dynamics (timers only) and on the guards are perfectly reasonable from the supervisor implementation viewpoint, the fact that communication is limited to discrete events only represents a major limitation in terms of applicability of the method, since it makes any supervisor interacting with the environment through shared variables very difficult to represent: in principle, one must add an event for every possible value of the finite-precision shared variables.

B. Refinement of the puncturing test case

To be able to satisfy the requirements for performing the refinement analysis described above on the puncturing test case described in Section IV, we used a straightforward transformation to re-define the original hybrid model in Figure 4 into an equivalent hybrid model in which the supervisor and the control tasks are described by separate automata. This model,



(a) Environment $\mathcal{E}$ (b) Supervisor $\mathcal{C}$

Fig. 5. Automata for refinement analysis.

noted as $\mathcal{M}$, is depicted in Figure 5. $\mathcal{M}$ is characterized by an Elastic controller $\mathcal{C}$ modeling the supervisor, and a hybrid automaton $\mathcal{E}$ modeling the set of subtasks with the corresponding local controllers. Notice that the two automata interact only by means of input and output synchronization events (marked by question and exclamation mark suffixes, respectively): the controlled automaton $\mathcal{E}$ sends notifications to the supervisor $\mathcal{C}$ when a relevant event occurs, like the transition from a space region to another (i.e., $fFast$), or the fact that the end effector touches the patient's body (i.e., $fSlow$), is perpendicular to it (i.e. $fPerp$), or has penetrated the tissue (i.e., $fPunct$). Then it moves to intermediate locations (i.e., $fastslow$, $slowperp$, $perppunct$, $punctstop$) to make possible the reception of the supervisor commands (i.e., $Fast2Slow$, $Slow2Perp$, $Perp2Punct$, $Punct2Stop$) which enable the controlled plant to switch from one local controller to another (i.e., $fast$, $slow$, $perp$, $puncturing$, $stop$). Notice that the supervisor automaton $\mathcal{C}$ is able to instantaneously generate the switch commands for the controlled automaton $\mathcal{E}$, due to the use of the constraint $t \leq 0$ as invariant.

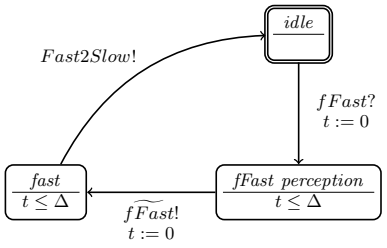


Fig. 6. The portion of the relaxed supervisor implementing the transitions between *idle* and *fast*.

According to the AASAP approach, from such a model $\mathcal{M}$ it is possible to automatically extract a new model $\mathcal{M}'$ that, when proved to be correct against the hybrid system specifications, guarantees the possibility of refining the original hybrid model. The model $\mathcal{M}'$ is given by the composition of $\mathcal{E}$ with a relaxed supervisor $\mathcal{C}_\mathcal{R}$, obtained from $\mathcal{C}$ by applying modifications which remove the unrealistic assumptions of hybrid automata. Figure 6 depicts the portion of $\mathcal{C}_\mathcal{R}$ that takes care of the transition between the local controllers *fast* and *slow* of the robot removing the supervisor instantaneity in handling the $fFast$ and $Fast2Slow$ synchronization events. In particular, the supervisor automaton $\mathcal{C}$ is relaxed by introducing a new location (i.e., $fFast\ perception$) used to model the small amount of time ($t \leq \Delta$) elapsing between the occurrence of the event $fFast$ notified by $\mathcal{E}$ and its perception in $\mathcal{C}$ due to sensors latency. Only after this perception (i.e., $\widetilde{fFast}$) the supervisor $\mathcal{C}$ can issue a control command (i.e., $Fast2Slow$) to force $\mathcal{E}$ to switch between the tasks $fastslow$ and $slow$. Notice that the command is not issued immediately, but only after a small delay ($t \leq \Delta$) that models the time required to forward the command through actuators.

Then, by verifying $\mathcal{M}'$ against properties **P1** and **P2**, we can obtain the maximum value Δ_{max} for the parameter Δ that returns a relaxed safe system. As proved in [6], the expression

$$\Delta_{max} > 4\Delta_P + 3\Delta_L \tag{1}$$

relates Δ_{max} to the actual constraints to which the discrete supervisor has to adhere. These constraints are the clock period Δ_P, and the worst-case-time required for treating incoming and outgoing synchronization events Δ_L. It holds that, if Δ_{max} is strictly greater than zero, then any discrete implementation $\mathcal{C}_\mathcal{D}$ of the supervisor such that Δ_P and Δ_L satisfy the above expression is able to control the environment $\mathcal{E}$, while respecting the properties.

Moreover, the value of Δ_{max} can be used to drive the choice of sensors and actuators for the implementation of the system. Suppose that the designer has the possibility to choose between different hardware platforms for the control board, and different sensors and actuators for the robotic manipulator, with different tradeoffs between price and performance. The choice of the control board affects the value of Δ_P (clock precision), while the choice of the sensors and actuators affects the value of Δ_L (delay due to sensing and actuation). To minimize the final cost of the system, but still guaranteeing correctness, one

has to choose the cheapest combination of components for which the values of Δ_P and Δ_L still satisfy (1).

VII. Conclusion

In this paper we discussed the application of formal methods for the verification, refinement and implementation of autonomous robotic systems. By means of an Autonomous Robotic Surgery test case, we showed how they can be of great impact in helping designers and developers when assessing the correctness of the system, and in guiding the choice of the hardware/software platform and of sensors and actuators to be used in the actual system implementation.

To conclude, we point out some open challenges and problems that the research community must face to make formal verification methods adopted as common practice also in the design flow of commercial robotic systems.

Robotic systems usually need a large number of continuous variables, paired with complex nonlinear dynamics, to be described faithfully. For instance, the robotic manipulator presented in Section IV requires at least 12 variables to be modeled completely. This makes their formal verification computationally very demanding, if not completely outside the capabilities of currently available tools.

The refinement and implementation phase is even more challenging. Most of the methodologies proposed in the literature are in an early stage and lack a comprehensive theoretical study. None of them can address completely the problems we pointed out in Section V, either due to a lack of expressivity or tool support.

Hence, the application of formal methods to autonomous robotic systems calls for the development of both new theories for the analysis of hybrid systems, and of new and improved tools supporting them.

References

[1] H. Kress-Gazit, "Robot challenges: Toward development of verication and synthesis techniques," *IEEE Robotics & Automation Magazine*, vol. 18, no. 3, pp. 22–23, 2011.

[2] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," *Hybrid systems*, pp. 209–229, 1993.

[3] S. Ratschan and Z. She, "Safety verification of hybrid systems by constraint propagation based abstraction refinement," *ACM Transactions in Embedded Computing Systems*, vol. 6, no. 1, 2007.

[4] "Ariadne: An open tool for hybrid system analysis." [Online]. Available: http://ariadne.parades.rn.cnr.it

[5] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *Proc. of CAV 2011*, ser. LNCS. Springer, 2011, vol. 6806, pp. 379–395.

[6] M. De Wulf, L. Doyen, and J. F. Raskin, "Almost ASAP Semantics: from Timed Models to Timed Implementations," *Formal Aspects of Computing*, vol. 17, no. 3, pp. 319 – 341, 2005.

[7] O. Sankur, P. Bouyer, and N. Markey, "Shrinking timed automata," in *Proc. of IARCS Annual Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, ser. Leibniz International Proceedings in Informatics, vol. 13, 2011, pp. 90–102.

[8] T. Henzinger and V. Rusu, "Reachability verification for hybrid automata," *Hybrid Systems: Computation and Control*, pp. 190–204, 1998.

[9] S. Bogomolov, C. Mitrohin, and A. Podelski, "Composing reachability analyses of hybrid systems for safety and stability," in *Automated Technology for Verification and Analysis*, ser. LNCS, 2010, vol. 6252, pp. 67–81.

[10] A. Podelski and S. Wagner, "Region stability proofs for hybrid systems," *Formal Modeling and Analysis of Timed Systems*, pp. 320–335, 2007.

[11] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" *Journal of Computer and System Sciences*, vol. 57, no. 1, pp. 94–124, 1998.

[12] A. Puri and P. Varaiya, "Decidability of Hybrid Systems With Rectangular Differential Inclusions," in *Proc. of International Conference on Computer Aided Verification (CAV)*, 1994, pp. 95–104.

[13] J. McManis and P. Varaiya, "Suspension Automata: A Decidable Class of Hybrid Automata," in *Proc. of International Conference on Computer Aided Verification (CAV)*, 1994, pp. 105–117.

[14] T. Henzinger, "Hybrid Automata With Finite Bisimulations," in *Proc. of International Colloquium on Automata, Languages and Programming (ICALP)*, 1995, pp. 324–335.

[15] T. A. Henzinger, P. H. Ho, and H. Wong-Toi, "HyTech: A model checker for hybrid systems," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 110–122, 1997.

[16] O. Botchkarev and S. Tripakis, "Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations," in *Proc. of Hybrid Systems: Computation and Control*, ser. LNCS, vol. 1790. Springer, 2000, pp. 73–88.

[17] G. Frehse, "Phaver: algorithmic verification of hybrid systems past hytech," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 10, pp. 263–279, 2008.

[18] E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald, "Abstraction and counterexample-guided refinement in model checking of hybrid systems," *Internat. J. Found. Comput. Sci.*, vol. 14, no. 4, pp. 583–604, 2003.

[19] A. Platzer and J.-D. Quesel, "Keymaera: A hybrid theorem prover for hybrid systems," in *Proc. of IJCAR 2008*, ser. LNCS. Springer, 2008, vol. 5195, pp. 171–178.

[20] A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, and A. Sangiovanni-Vincentelli, "Ariadne: a framework for reachability analysis of hybrid automata," in *Proc. of the Int. Symp. on Mathematical Theory of Networks and Systems (MTNS'06)*, 2006.

[21] L. Benvenuti, D. Bresolin, A. Casagrande, P. Collins, A. Ferrari, E. Mazzi, A. Sangiovanni-Vincentelli, and T. Villa, "Reachability computation for hybrid systems with ariadne," in *Proceedings of the 17th IFAC World Congress*, 2008.

[22] P. Collins, "Semantics and computability of the evolution of hybrid systems," *SIAM J. Control Optim.*, vol. 49, pp. 890–925, 2011.

[23] P. Collins, D. Bresolin, L. Geretti, and T. Villa, "Computing the evolution of hybrid systems using rigorous function calculus," in *Proc. of the 4th IFAC Conf. on Analysis and Design of Hybrid Systems (ADHS 12)*, 2012.

[24] K. Cleary and C. Nguyen, "State of the art in surgical robotics: Clinical applications and technology challenges," *Computer Aided Surgery*, vol. 6, no. 6, pp. 312–328, 2001.

[25] C. Marohn and C. Hanly, "Twenty-first century surgery using twenty-first century technology: Surgical robotics* 1," *Current Surgery*, vol. 61, no. 5, pp. 466–473, 2004.

[26] E. Guglielmelli, M. J. Johnson, and T. Shibata, "Guest editorial special issue on rehabilitation robotics," *Robotics, IEEE Transactions on*, vol. 25, no. 3, pp. 477–480, Jun. 2009.

[27] J. Desai and N. Ayache, "Editorial Special Issue on Medical Robotics," *The International Journal of Robotics Research*, 2009.

[28] D. Botturi and P. Fiorini, "Optimal control for autonomous task execution," in *Decision and Control, 2005 and 2005 European Control Conference. 44th IEEE Conference on*, 2005, pp. 3525–3530.

[29] R. Muradore, D. Bresolin, L. Geretti, P. Fiorini, and T. Villa, "Robotic surgery: Formal verification of plans," *IEEE Robotics & Automation Magazine*, vol. 18, no. 3, pp. 24–32, 2011.

[30] P. Krčál, L. Mokrushin, P. Thiagarajan, and W. Yi, "Timed vs. time-triggered automata," in *Proc. of International Conference on Concurrency Theory (CONCUR)*, 2004, pp. 340–354.

[31] D. Bresolin, L. Di Guglielmo, L. Geretti, and T. Villa, "Correct-by-construction code generation from hybrid automata specification," in *International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2011, pp. 1660–1665.

[32] K. Altisen and S. Tripakis, "Implementation of Timed Automata: an Issue of Semantics or Modeling?" in *Proc. of International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2005, pp. 273–288.