Science of Computer Programming 245 (2025) 103314

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

A provably safe controller for the needle-steering problem using online strategy synthesis ☆

Sascha Lehmann [a], Antje Rogalla [a,*], Maximilian Neidhardt [b], Alexander Schlaefer [b], Sibylle Schupp [a]

[a] Institute for Software Systems, Hamburg University of Technology, Am Schwarzenberg-Campus 3, 21073 Hamburg, Germany
[b] Institute of Medical Technology and Intelligent Systems, Hamburg University of Technology, Am Schwarzenberg-Campus 3, 21073 Hamburg, Germany

ARTICLE INFO

Keywords:
Model checking
Timed game
Strategy synthesis
Controller synthesis
Needle steering

ABSTRACT

Autonomous systems often address complex planning problems, which require both prospective action planning and retrospective data evaluation. Timed games could aid since they automatically synthesize strategies that, provably correct, solve those planning problems; yet, they assume a static model of the environment, which is not realistic for autonomous systems. However, many autonomous systems are control applications, which employ sensors that capture system behavior at run time and can thus compensate for incomplete knowledge at modeling time. In this paper, we propose an *online strategy synthesis*, which, based on offline strategy synthesis on the one hand and on sensor information about the current state of the physical world on the other hand, derives formal safety guarantees while reacting and adapting to environment changes. We formalize the needle-steering problem from medical robotics, i.e., the problem of navigating a (flexible and beveled) needle through partially unknown tissue towards a target without damaging its surroundings, by interpreting it as a timed game. Further, we introduce a new representation of its environment through different region types that determine the acceptance of action plans and trigger local correcting actions. We present an algorithm for online strategy synthesis and, for the given region representation, formally prove that it returns safe online controllers. The algorithm is implemented on top of Uppaal Stratego. For two medical applications of needle steering, *peridural anesthesia* and *predefined needle trajectory*, we demonstrate the necessity of online adjustments in a series of simulations with various degrees of initial knowledge about the environment, and show that the overhead of online synthesis remains practical.

1. Introduction

In cyber-physical systems (CPS), one commonly deals with the task of directing a controllable entity through its environment towards a target state defined by local or global goals, without violating frame properties imposed by safety requirements. For applications of limited complexity with directly measurable environmental parameters (e.g., thermostats), practical control solutions already exist [1]. However, the environment of complex systems is often uncertain, changing, and not entirely measurable, rendering

☆ This work was partially funded by DFG SCHU 2479, DFG SCHL 1844 and i^3 lab initiative (internal funding id T-LP-E01-WTM-1801-02).
* Corresponding author.
E-mail address: antje.rogalla@tuhh.de (A. Rogalla).

https://doi.org/10.1016/j.scico.2025.103314
Received 6 June 2024; Received in revised form 12 March 2025; Accepted 1 April 2025

Available online 4 April 2025

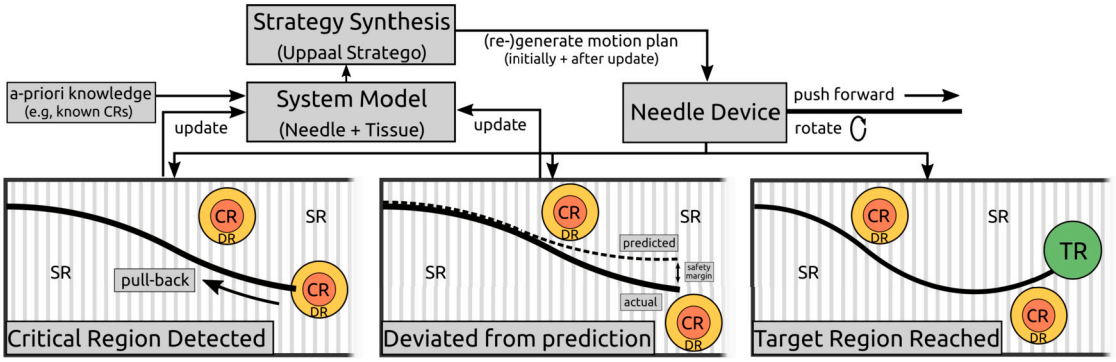


Fig. 1. The workflow of online strategy synthesis for the needle steering problem (sr/tr/dr/cr = safe/target/detection/critical region, see Sec. 4.3).

```
SUCCESSFUL STRATEGY GENERATION NR: 2
MOTION PLAN NR PER STRATEGY SYNTHESIS STEP: [56, 0, 0, 22]
[10:55:14][ Strat] Random motion plan selected.
[10:55:14][ Strat] Sending instruction message 'move f_0.0_r_180_f_32.0_r_180_f_2.0\n' ...
[10:55:14][ Strat] Sending instruction message 'move f_0.0_r_180_f_32.0_r_180_f_2.0\n' ... finished.
[10:55:14][Needle] Received instruction: "move f_0.0_r_180_f_32.0_r_180_f_2.0"
[10:55:14][Needle] Rotating by 180.0 degree ...
[10:55:14][Needle] Rotating by 180.0 degree ... finished.
[10:55:14][Needle] Stepping forward by 32.0 ... [start progress: 28.0]
[10:55:19][ Strat] DR reached. Adapt needle and identify new strategy.
FORCE EXCEEDED NR: 2
[10:55:19][ Strat] Readjust needle and identify new strategy.
[10:55:19][ Strat] Sent message "stopped" to ('127.0.0.1', 37006).
[10:55:19][ Strat] Wait for message "Stop true" at ('127.0.0.1', 37006) ...
[10:55:19][Needle] Sent message "Stop true" to ('127.0.0.1', 1340).
[10:55:19][ Strat] Received message "Stop true" at ('127.0.0.1', 37006).
[10:55:19][Needle] Sent message "Stopped Movement" to ('127.0.0.1', 1339).
[10:55:19][ Strat] Wait for message "Stopped Movement" or "move ready" ...
[10:55:19][Needle] Needle device stopped.
[10:55:19][ Strat] Sent message "start" to ('127.0.0.1', 37006).
[10:55:19][Needle] Sent message "Start true" to ('127.0.0.1', 1340).
[10:55:19][ Strat] Wait for message "Start true" at ('127.0.0.1', 37006) ...
[10:55:19][ Strat] Received message "Start true" at ('127.0.0.1', 37006).
[10:55:19][ Strat] Received message "Stopped Movement".
TOTAL NUMBER OF PULL BACKS: 4
[10:55:19][ Strat] Pull back needle by -30.0 units ...
[10:55:19][ Strat] Sending instruction message 'move f_-30.0\n' ...
```

Fig. 2. Excerpt from the simulation steps after the second successful strategy generation.

a comprehensive model of such environments for safety prediction infeasible. Deriving safety guarantees via formal methods imposes further limitations on feasible model complexities. Using a partial model instead, global safety guarantees might not hold anymore as soon as reality and model deviate.

In practice, safety is only required for the actual trace of the real system. An *online* approach based on model updates on-the-fly splits the task of deriving global guarantees into a series of safety verifications on locally valid models with limited scope. That way, one can guarantee safety for all possible near futures of the current system state. However, two main problems persist: first, one still needs to guarantee that the system does not reach critical sections due to local deviations of the predicting model. Second, one needs to decide when to adapt the model to the real system. We approach these problems by splitting the environment into discrete regions—target, safe, critical, and detection regions—based on which we can derive safety-preserving actions. A safe controller will then reach a target region via safe regions without entering any critical regions, which are early detected in surrounding detection regions.

Fig. 1 gives an overview of the workflow of online strategy synthesis (OnSS) for the case of needle steering. Leaving the specifics of needle steering for a moment aside, the general workflow reads as follows: first, the model is initialized with data of the real system's starting state. Then, an initial strategy is synthesized offline via reachability check. Afterwards, the system is executed and its tracked trace matched against the current motion plan model. If the needle deviates too much, the model is again updated, and a new strategy is synthesized. If no strategy can be found, or if a detection region (DR) is reached, the system is *readjusted*, i.e., rolled back to a previously visited and known state, and the model is updated to the new system state after readjustment. If the initial state is re-reached via rollbacks, and again no strategy is found, the process is aborted (and may be started anew under different conditions). If the target region is reached, the process succeeds. Fig. 2 illustrates the executing, stopping, and readjusting steps.

Described in general terms as above, OnSS is applicable to a wide range of dynamic cyber-physical systems. Yet, each of the steps displayed in Fig. 1—detecting deviations or critical regions, readjusting and updating the system model—is application-specific

and the safety of a controller online, at run time, depends on a proper formal model of the environment. In this paper, we provide such model for the class of needle-steering problems. Needle steering, i.e., the task of steering a flexible and beveled needle through soft tissue, is a field of its own in medical robotics and presents an interesting application for OnSS since tissue behavior cannot be precisely modeled and critical regions therefore are known at run time only.

Overall, we make the following contributions:

1. We formalize the safety problem of dynamic autonomous systems by introducing online strategy synthesis. Employing an existing offline strategy synthesis approach, OnSS is based on the formal method of timed games and can give formal safety guarantees for autonomous systems in uncertain environments.
2. We introduce an abstract region representation of the environment (for the real system and model) where critical regions are unreachable by construction; and we provide concrete definitions of regions for the needle-steering problem.
3. We allow for unpredictable needle-tissue interactions and present a novel architecture in which online controller synthesis is interleaved with offline synthesis. Our algorithm for online controller synthesis, OCS, manages incomplete knowledge about critical regions and unexpected needle moves. We prove three important properties: reachability, safety, and conditional termination: whenever a strategy is executed, it executes safely.
4. We apply OnSS to two applications of needle steering, peridural anesthesia and predefined needle trajectory. In a series of software simulations we illustrate its workings, including the need for online adjustments, and show that its performance is acceptable.

In order to stick to reality, we have defined some actual parameters' values of the software simulation by running experiments using a real needle, gelatin phantoms to simulate soft tissue, and two cameras, image processing software, and an oscilloscope to estimate the position of the needle. In the following, we will use the term *hardware experiments* to refer to them, as well as *hardware configuration* to refer to the actual experiment settings.

The remaining paper is structured as follows: we discuss related work in Sec. 2 and provide preliminary definitions, in particular the definition of timed games and winning strategies, in Sec. 3. In Sec. 4, we formalize the needle-steering problem. After a brief summary of the needle-steering problem in medical robotics (Sec. 4.1), we elaborate on its interpretation as a timed game (Sec. 4.2), introduce the representation of the environment first through abstract, then through concrete regions (Sec. 4.3 and 4.4), and, lastly, in (Sec. 4.5), provide the underlying formal model, a network of timed automata, implemented in Uppaal Stratego [2]. The OnSS itself is the subject of Sec. 5, where we motivate the shift from offline to an online game (Sec. 5.1), present and discuss the algorithm OCS (Sec. 5.2), and prove relevant properties, including the safety of the needle-steering controller at run time (Sec. 5.3). Afterwards, we report simulation results in Sec. 6 and conclude our work in Sec. 7. This paper is an extended version of our contribution to the Third Workshop on Formal Methods for Autonomous Systems (FMAS'21) [3]: we present the algorithm for online controller synthesis, a refined formulation of its theoretical properties, formal proofs of those properties, and an empirical evaluation using two medical applications of needle steering, peridural anesthesia, and predefined needle trajectory.

Notation We use *italics* to denote timed automata and their components, in particular action transitions (e.g., *do_ push*). We use `teletype` for code, (e.g., `Check_CR_Reached`), most notably C++ definitions as supported by Uppaal's implementation of timed automata; and we use SMALL CAPS (e.g., FITCIRCLE) for (pseudo) code operating on the model.

2. Related work

When rigorous guarantees are needed for control software, a common method of choice are (timed) games: instead of verifying the controller afterwards, the controller is synthesized automatically, so that it is correct by construction. Timed games are an extension of timed automata by uncontrollable transitions, dating back to the turn of the century [4][5][6]. While one research focus is on general algorithms for controller synthesis for reachability timed games [7] and the generation of time-optimal strategies [8] [9][10], another research direction refines the formalization of reachability timed games, by adding weights on locations and transitions [11] to generate cost-optimal or cost-bounded strategies [12][13][14][15][16][17]. Since the 2010s another extension of timed games is studied: probabilistic timed games, e.g., $1\frac{1}{2}$ or $2\frac{1}{2}$ games. In a $1\frac{1}{2}$ game, one player follows deterministically a strategy and the other half player is the environment, represented by probabilistic choices over action transitions and probability distributions over delay transitions; in a $2\frac{1}{2}$ game, both players follow a strategy but take into account the probabilistic behavior of the environment [18][19][20][21]. Timed games are also studied for winning condition specifications in different logics [22][23][24]. Most of this research, however, addresses questions of algorithmic decidability assuming special configurations (e.g., one clock automata and stopwatch cost [15]) or provably optimal strategies (optimal-time, time-bound, optimal-cost, or cost-bound), and simply assumes that models are available. In contrast, we address the situation where not all information is available at modeling time.

Few works exist that relax the requirement of complete information. Bacci et al. investigate the partially observable oil-pump control problem and deal with imprecise knowledge of energy rates by assigning imprecision to updates of edges to the energy-timed automata, where transitions can both consume and generate resources [25]. Another approach to solve controller synthesis under partial observability is the template-based controller synthesis by Finkenbeiner and Peter, where automatic abstract refinement reduces incrementally the set of valuation of parameters until only safe states are reachable [26]. Cassez et al. [27] permit the situation of incomplete observations and generate a controller strategy by transforming the game into an equivalent game with

complete information. While our approach could also be understood as transformational, they address more general games than we (2 player games) but require strategies to be stuttering-invariant, i.e., invariant under delays.

In automated planning, a large body of works exist dealing with different kinds of uncertainty. Applicable to our problem of uncertainty about critical regions, are approaches like FOND and contingent planning, both permitting actions with unknown effects [28,29]. Yet both assume that effects can be enumerated or described as partial function, which is not the case for the second source of uncertainty in the needle-steering problem, the needle-tissue interaction. Closest to our online synthesis are replanning approaches, which, similar to our feedback loop, run a plan for a slightly different problem online and as long as it is safe to do, but are prepared to replan otherwise [30,31]; often though, they require a connected state space, which, again, is different in our needle-tissue interaction model. In general terms of automated planning our online synthesis approach can be described as a partially observable, non-deterministic, dynamic problem with a non-finite, non-enumerable state space. Since we utilize offline synthesis, which could be characterized as a static, finite, fully observable, thus classical planning problem [32], we proceed similar to the translation-based approach to contingent planning [33], although the differences of the two planning problems entail relaxations of different nature. Also different from automated planning, strategy synthesis does not return a single execution sequence, but a function ("strategy") that advises the controller deterministically during the game which action to take to guarantee reaching a target.

Yet another strand of research deals with tool support for timed games. Early tools include SynthKro, a module of the Kronos [34] tool suite, and FlySynth [35] for dense-time systems. While controller synthesis in SynthKro was realized by a fix-point iteration on symbolic state representations [4][5][6][7], in FlySynth the first on-the-fly synthesis on time-abstracting quotient graphs [36][37] was implemented. Prominent tools of the second generation are Synthia [38], the first tool to use abstraction refinement for state-space optimizations, and Uppaal Tiga [39] implementing a symbolic on-the-fly synthesis algorithm [40]; Uppaal Tiga has been further developed to Uppaal Stratego [2]. We build our algorithm for online controller synthesis on top of Uppaal Stratego and propose a practical approach to solve timed games with unknown requirements. Recently, strategy synthesis and spatial model checking were combined in a tool chain that integrates the Contract Automata Library (CATLib) and the spatial model checker VoxLogicA [41].

Timed games have been successfully applied to scheduling problems [42][43], railway, cruise, and traffic light control problems [44][45][46][47], power management [48][49][50] and other applications as playing ball [51], floor heating [52], and needle steering [53]. With the exception of the floor-heating game, however, all games are offline games. While the needle-steering game inspired the work in this paper, it presents an offline reachability timed game for steering a flexible needle in homogeneous tissue. Ignoring the fact that tissue in reality is inhomogeneous and has anatomic obstacles, it is an example of an offline game that is not applicable in real medical applications. In contrast, the floor-heating game [52] presents a compositional online synthesis approach to control the floor heating system in a house for a short period, based on current sensor data. However, the approach assumes complete knowledge of the process, which we do not.

In medical robotics, the use of flexible needles presents a particular research challenge: to drive flexible needles to a soft tissue target, the needle-tissue interaction needs to be known. First approaches include nonholonomic models to estimate the needle trajectory in homogeneous tissue [54], but they do not include tissue movement due to insertion forces. Hence, FEM modelings with linear elastic tissue have been developed [55]. Yet, simulations of needle-tissue interaction offline, solely based on clinical image data, are of limited use when applied to a real clinical needle insertion as the image data does not provide details of the mechanical properties of the respective tissues. A few adaptive online models are now available that include uncertainties from tissue inhomogeneities [56], needle buckling and slip-stick-movement during needle insertion [57], or moving targets [58]. Those adaptive models can be applied in data-driven methods [59] or with real-time sensoring [60,58]. Still, neither method can provide any formal guarantees. Closely related to our project is the work [61] from the Motion Planning for Steerable Needles project, which considers both motion and sensing uncertainty, computes a policy rather than a static path, and formalizes the needle-steering problem, yet differently from our approach: as partially observable Markov decision process (POMDP), while our approach is based on model checking.

3. Preliminaries

We start the preliminaries to our work by recapitulating the definition of timed automata. Relevant for our application are *networks* of timed automata, since they enforce synchronization between needle motion and tissue response; relevant for our online approach are *extended* timed automata, since they introduce data variables, which are subject to online updates, as well as *urgent* and *committed* locations, which prohibit time delays. In this section, we also define the *run* in a timed game automaton and *reachability* and *safety games*.

Definition 1 *(Timed automaton [62]).* A timed automaton $TA = (L, L_0, Act, C, E, Inv, Atom, \mathcal{L})$ is a tuple where L is a finite set of locations, $L_0 \subseteq L$ is a finite set of initial locations, $Atom$ is a set of atomic propositions, and Act is a finite set of actions. C is a finite set of nonnegative real-valued clocks, $\mathcal{B}(C)$ is a set of conjunctions ("clock constraints") of the form $x \bowtie c$ or $x - y \bowtie c$ for clocks x, y, $c \in \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$, $E \subseteq L \times \mathcal{B}(C) \times Act \times 2^C \times L$ is a transition relation, $Inv : L \to \mathcal{B}(C) \cup \{\top\}$ is a mapping from locations to clock constraints and the constant *true* ($\top$), and $\mathcal{L} : L \to 2^{Atom}$ is a labeling function.

Definition 2 *(Network of timed automata [63,64]).* Given $n > 1$ timed automata $TA_i = (L_i, L_{0,i}, Act_i, C, E_i, Inv_i, \text{Atom}_i, \mathcal{L}_i)$. Their composition forms the network $TA = (L^n, L_0^n, Act^n, C, Inv, \text{Atom}^n, \mathcal{L})$ where $L^n = L_1 \times \cdots \times L_n$ defines the location vectors, $L_0^n = l_{0,1} \times \cdots \times l_{0,n}$ the initial location; $Act^n = Act_1 \cup \cdots \cup Act_n$ and $\text{Atom}^n = \text{Atom}_1 \cup \cdots \cup \text{Atom}_n$ define the actions and atomic propositions of the network; Inv and $\mathcal{L}$ take location vectors and are component-wise defined. The semantics of the network is defined as the transition system $(S, s_0, \longrightarrow)$ where $S = L^n \times \mathbb{R}^C$ is the set of states, $s = (\bar{l}, u) \in S$ is a pair of location vector $\bar{l}$ and clock valuation

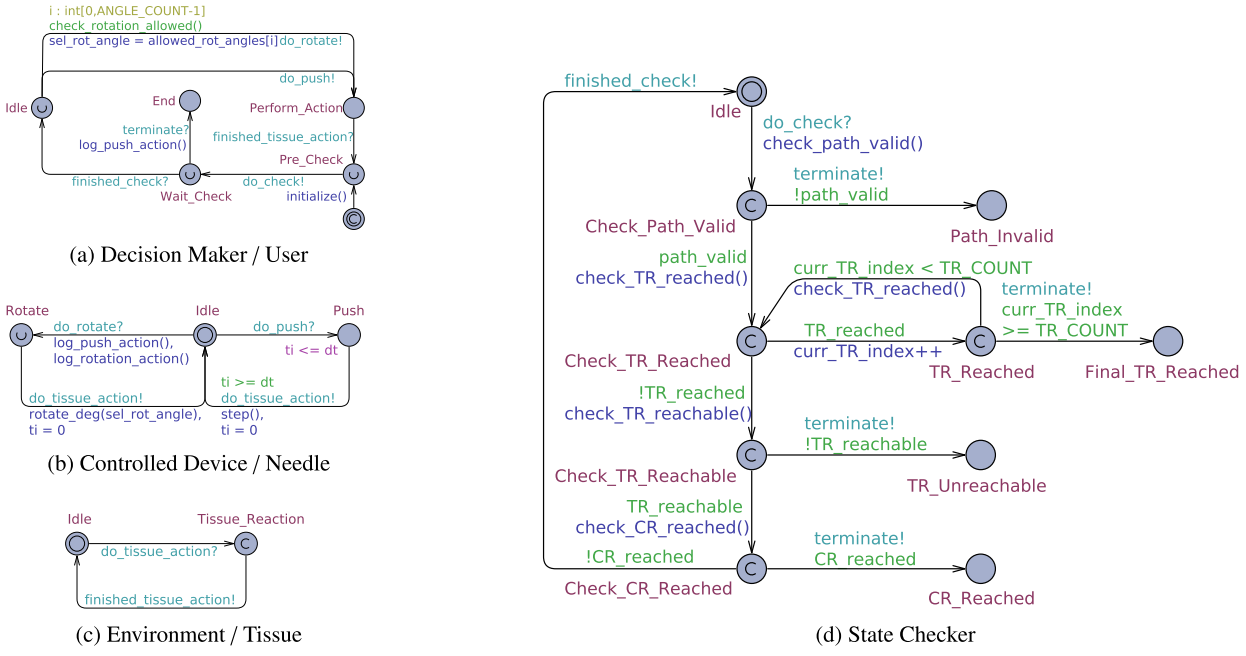


Fig. 3. The Uppaal model components [66].

function $u : C \to \mathbb{R}$, and $s_0 = (L_0^n, u_0)$ is the initial state with $u_0(x) = 0 \forall x \in C$. The transition relation $\longrightarrow \subseteq S \times S$ defines 3 kinds of transitions: (i) $d \in \mathbb{R}_{\geq 0}$ defines a *delay transition*, $(\bar{l}, u) \xrightarrow{d} (\bar{l}, u + d)$, if $\forall d' : 0 \leq d' \leq d \to u + d' \in \mathcal{I}(\bar{l})$; (ii) the edge $\bar{l} \xrightarrow{agr} (\bar{l}[l'_i/l_i]$ defines an *action transition*, $(\bar{l}, u) \xrightarrow{a} (\bar{l}[l'_i/l_i], u')$, if $u \in g$, $u' = [r \mapsto 0]u$, and $u' \in \mathcal{I}(\bar{l}[l'_i/l_i])$ where $g \in \mathcal{B}(C)$ denotes a guard, r the set of clocks that are reset, and $\bar{l}[l'_i/l_i]$ the vector $\bar{l}$ where l_i is replaced by l'_i; and (iii) the handshaking edge $l_i \xrightarrow{a?g_i r_i} l'_i$ and $l_j \xrightarrow{a!g_j r_j} l'_j$ defines a *synchronizing action transition*,

$$(\bar{l}, u) \xrightarrow{a} (\bar{l}[l'_i/l_i, l'_j/l_j], u'),$$

if $u \in (g_i \wedge g_j)$, $u' = [r_i \cup r_j \mapsto 0]u$, and $u' \in \mathcal{I}(\bar{l}[l'_i/l_i, l'_j/l_j])$.

Definition 3 *(Extended timed automaton [65]).* Let L, L_0, C, Inv, and $\mathcal{L}$ as in the TA definition. An extended timed automaton (ETA) is a tuple $(L, L_0, Act, C, E, Inv, Atom, \mathcal{L}, V, L_U, L_C, Ch_{Bi}, Ch_{Br})$, where $E \subseteq L \times \mathcal{B}(C) \times Act \times 2^C \times A(V) \times L$ is a set of edges as in TAs but with additional variable assignments $va \in A(V)$, V is a set of variables. $L_U \subseteq L$ and $L_C \subseteq L$ are sets of urgent and committed locations (with $L_U \cap L_C = \emptyset$), and Ch_{Bi} and Ch_{Br} are binary and broadcast synchronization channels (with $Ch_{Bi} \cup Ch_{Br} \subseteq$ Act and $Ch_{Bi} \cap Ch_{Br} = \emptyset$). The notion of networks can be lifted to networks of ETAs.

Example 1. The model of a needle in Fig. 3b is an ETA (and part of a network of 4 ETAs). The needle ETA consists of 3 locations (*Idle*, *Push*, and *Rotate*), where *Idle* is the initial location (marked by two circles) and *Rotate* is an urgent location (marked by a *U*). The automaton has one clock, *ti*; the two transitions from locations *Push* and *Rotate* with the action *do_tissue_action* reset the clock to its initial value $ti = 0$ and return to *Idle*. The expression *ti<=dt* is an invariant of the location *Push*, while *ti>=dt* serves as guard; together, invariant and guard ensure that the action transition from *Push* to *Idle* is taken exactly when $ti = dt$. Very important are the synchronization messages: *do_tissue_action!* synchronizes with the (receiving) action *do_tissue_action?* from the tissue automaton in Fig. 3c, while *do_push?* and *do_rotate?* listen to the user automaton in the same figure. Finally, `step()`, `rotate_deg()`, and the two log functions refer to C++ functions.

Definition 4 *(Timed game automaton, runs).* A *timed game automaton TGA* is a timed automaton with a set of Actions *Act* partitioned into *controllable* actions $\texttt{Act}_c$ and *uncontrollable* actions $\texttt{Act}_u$. The semantics of a timed (game) automaton $TS_{TGA} = (S, s_0, A, \to, Atom, L)$ is defined by its associated timed transition system (see [62] and [67]); the definition can be lifted to *networks* of TGAs (see, e.g., [68]).

A *run* is a sequence of alternating delay and action transitions in $\texttt{TS}_{\texttt{TGA}}$. An initial run $\rho : s_0 \xrightarrow{d_0} s'_0 \xrightarrow{a_0} s_1 \xrightarrow{d_1} s'_1 \xrightarrow{a_1} s_2 \ldots (d_i \in \mathbb{R}^+)$ starts in the initial state s_0. The induced set of states on ρ is denoted $\texttt{States}(\rho) = \{s_0, s'_0, s_1, s'_1, s_2, \ldots\}$ [67]. A maximal run is either infinite or ends in a terminal state, which is a state where neither delay nor action transitions are possible. $\texttt{Runs}(s, \texttt{TS}_{\texttt{TGA}})$ denotes the set of runs starting in s. $\texttt{Runs}(\texttt{TS}_{\texttt{TGA}})$ and $\texttt{Runs}_{\texttt{max}}(\texttt{TS}_{\texttt{TGA}})$ denote the set of initial runs and initial, maximal runs.

Example 2. Consider again the model of a needle in Fig. 3. In the associated timed transition system, states are pairs of location and clock value, e.g., $(Idle,\ ti=0)$. An example of an initial run is $(Idle,\ ti=0) \xrightarrow{d_0} (Idle,\ ti=d0) \xrightarrow{do\_push} (Push,\ ti=d0) \xrightarrow{dt-d_0} (Idle,\ ti=dt) \xrightarrow{do\_tissue\_action} (Idle,\ ti=0) \longrightarrow \cdots$. In the transition system of the network of ETAs, an initial run starts with the location vector in which all four ETAs are in *Idle* and then either stays in the initial state or proceeds with either of the two synchronizations *do_push* or *do_rotate*. We return to that example in Sec. 4.5.

Definition 5 *(Reachability and safety games).* A *timed game* is a pair $(\mathtt{TS}_{\mathtt{TGA}}, \Psi)$, where $\mathtt{TS}_{\mathtt{TGA}}$ is the associated timed game transition system of a timed game automaton *TGA* and Ψ a winning condition [23]. A *reachability timed game* is a timed game with a *reachability* winning condition defining a set of goal states $S_{\mathtt{goal}} \subseteq S$ [40]. A run $\rho \in \mathtt{Runs}(\mathtt{TS}_{\mathtt{TGA}})$ is winning for $S_{\mathtt{goal}}$ if $\mathtt{States}(\rho) \cap S_{\mathtt{goal}} \neq \emptyset$. A *safety timed game* is a timed game with a *safety* winning condition defining a set of locations to avoid $S_{\mathtt{avoid}} \subseteq S$ [40]. A run $\rho \in \mathtt{Runs}_{\mathtt{max}}(\mathtt{TS}_{\mathtt{TGA}})$ is winning for $S_{\mathtt{avoid}}$ if $\mathtt{States}(\rho) \cap S_{\mathtt{avoid}} = \emptyset$. Given a $\mathtt{TS}_{\mathtt{TGA}}$. A *strategy* f over $\mathtt{TS}_{\mathtt{TGA}}$ is a partial function from $\mathtt{Runs}(\mathtt{TS}_{\mathtt{TGA}})$ to $\mathtt{Act}_c \cup \epsilon$, where ϵ denotes the situation where the best next controlled action is to *not* carry out any action but to wait.

Informally, a strategy f is *winning* from s if all maximal runs in its outcome are winning runs; for a formal definition, we refer to [13] and [40].

Remark 1. Winning strategies can be thought of as trees of instructions on how to achieve the winning condition, and may therefore be distinguished from the concretely executing actions, which form a linear sequence. In the following we use the term motion plan when we want to highlight that (executable) sequence.

4. Formalizing the needle-steering problem

We reported already that timed games are used for controller synthesis (Sec. 2) and that they themselves build on timed automata as formal model (Sec. 3). We now apply this game- and modeling notion to the needle-steering problem. We show how needle steering can be interpreted as a timed game, and what kind of game it is (Sec. 4.2), and we provide a formal model of this class of application, a network of four (extended) timed automata in Uppaal in Sec. 4.5. Our new representation of the environment is presented in Sec. 4.3 and Sec. 4.4. We start off, however, with a brief characterization of needle steering from the perspective of medical robotics.

4.1. The needle-steering problem in medical robotics

Placing biopsy needles in deep tissue structures is a common medical procedure. To increase the accuracy in reaching the target tissue, physicians typically use some form of image guidance, e.g., X-ray or ultrasound. However, even after successful initial alignment, the needle may deviate from the planned trajectory, e.g., due to tissue deformation and needle deflection. Hence, the needle may need to be retracted and realigned multiple times, potentially leading to increased trauma and longer interventions. The idea of needle steering is to use flexible needles such that the path of the needle through the tissue can be controlled. One approach uses needles with a bevel-shaped tip that can be rotated. For a fixed rotation angle, the needle will deflect substantially, while a continuous rotation of the needle allows following a straight path. However, note that the motion of the needle depends not only on its velocity, orientation, and the exact shape of its tip but also on unknown parameters — most importantly, the (in-) homogeneity of the surrounding tissue and its elasticity. Generally, stiffer tissue will cause larger deflections, and inhomogeneities can result in abrupt longitudinal needle movements. So far, a sufficiently complete and patient-specific model of the needle-tissue interaction cannot be obtained in advance. Hence, path planning typically involves iterative approaches where the path is frequently updated during insertion, e.g., using rapidly exploring trees (see Sec. 2). As a result, a path that reaches the target and avoids critical structures like nerves and vessels is determined. Note that this approach does not include formal guarantees.

4.2. Needle steering as a timed game

Considering needle steering as a game, the game involves the user (robot), the device (a flexible needle), and the tissue. Users are full players, i.e., they are in control of their actions. The needle and tissue, on the other hand, respond. Since they respond in possibly unexpected ways—the needle might bend, the patient's motion might move organs—they force the user to react. In our model, the set of user actions is abstracted to 3 movements: moving the needle forward, rotating it by a certain angle, or pulling it back (only in case of unexpected tissue response).

Since needle and tissue cannot outplay the user in any systematic way, we consider the offline needle-steering game as a 1-player game, where the set of actions consists of exactly two controllable actions $\mathrm{Act} = \mathrm{Act}_c = \{do\_push,\ do\_rotate\}$. The game is a timed game since clocks are needed to determine in short time intervals periodically the next action and perform safety checks (see, e.g., Fig. 4). Different from the common 2-player timed games, however, waiting does not help to win the game.

Given the user's primary goal of reaching a particular tissue T in the body, needle steering is, prima facie, a reachability game (Sec. 3). For such target location T, one winning run of the reachability game is a (concrete) motion plan. In illustration, the following sequence of controlled actions

$$init \xrightarrow{do\_push} s_1 \xrightarrow{do\_rotate} s_2 \xrightarrow{do\_push} s_3 \xrightarrow{do\_rotate} s_5 \xrightarrow{do\_push} T$$

for states s_i, initial state *init*, and target state T presents a motion plan with which, since the target T is reached, the user wins the reachability game. In practice, the target tissue is marked before a medical treatment session starts (for example, using gold markers, which can be detected by image sensors). Thus, it is known at modeling time how the target can be detected and the reachability game can therefore be played classically offline. In the syntax of Uppaal Stratego, the corresponding query in Timed Computational Tree Logic (TCTL) is as follows

$$strategy\ ReachFinalTR = control : A <> M.T$$

where $A <>$ is the usual TCTL operator for "always possible", T the target in model M, and *control* Uppaal's keyword for offline synthesis [40]; for re-usability, the resulting strategies can be stored, here under the name *ReachFinalTR*. The output of Uppaal's offline strategy synthesis is a compact representation of a set of strategies.

Yet, a needle-steering application is more than just a reachability game. It also has to observe a number of safety constraints, since the needle must not navigate, e.g., through organs or other tissue. For the game, one therefore needs to distinguish safe states from unsafe, critical ones. Different from other timed games, however, the safety of a state is data-dependent and needs to be computed upon each state transition. From that data dependence, two major design decisions for the game follow. For one, since not all data is available at modeling time, it necessitates the move from an offline to an online game, which motivates this paper and which we discuss below (see Sec. 5.1). Second, all controllable actions in Act_c need to be guarded, that is, action transitions in the underlying timed automaton are of the form

$$s_1 \xrightarrow{g, do\_push} s_2,$$

for a guard g, so that the safety of the action can be checked before it executes. Invoking a safety check for each action, turns the previous winning strategy-synthesis for reachability games in a synthesis for strategies that are additionally safe: whenever a safety check fails, the action or even an associated path fragment is discarded, and other paths are considered for synthesis of a safe strategy. Note that at implementation level, clocks (timers) of the underlying timed automata can ensure that those safety checks are periodically triggered and that the interval at which the checks take place is defined in dependence of the velocity of the needle to ensure that the previous *do_push* has not already entered a critical region before the next check (see Sec. 4.5).

In conclusion, the offline strategy synthesis just described yields needle trajectories that are discrete sequences of actions where, at synthesis time, in between any two actions a safety check took place.

4.3. Abstract region representation

The safety checks from the previous subsection require a representation of unsafe, critical regions. Since the environment is partly unknown, such representation must come with safety margins, so that an application has time to react. We therefore require the construction of a safety cordon around each critical region and introduce the notion of *detection regions*, which must surround critical regions so that they are always entered before any critical region is. In needle steering, where critical states are punctured organs, the organ's tissue is always deformed for some time before it is punctured; the time span of that deformation, thus, defines the detection region.

For any given needle position, our abstract region representation partitions the state space explored so far in 4 regions; moving towards the target, more and more states are explored [69]. The 4 regions are:

Safe regions (SR)	Regions that do not violate safety
Critical regions (CR)	Regions that violate safety
Detection regions (DR)	Transitional regions between SRs and CRs where upcoming CRs (= safety violations) can be detected
Target regions (TR)	Regions that we want to reach with the controllable entity

Detection regions (DR), by definition, are constructed so that critical regions are never entered; DRs themselves, along with safe and target regions are informally "safe": i.e., $(SR \cup TR \cup DR) \cap CR = \emptyset$, and SR, TR, DR are pairwise disjoint. Further, we require region types to be discrete so that any finite space can be covered by a finite number of regions.

4.4. Region representation for the needle-steering problem

For any particular problem class, the abstract region representation from the previous subsection needs to be concretized. In needle-steering applications, the SRs are the uncritical tissue areas, the CRs are hardened tissue and organs, the TR is the targeted placement position of the needle, and the DRs are pre-rupture deformation sections around critical sections. For the concretization, we represent CRs and TRs geometrically through spheres, and DRs by measured quantities at needle tips; SRs are not represented at all since they need not be detected.

In more detail, for needle-steering applications, one can assume that the transition to a detection region, DR, is always measurable: due to deformations in the tissue, one can always observe an increasing force on the needle tip when moving to another—possibly

critical—tissue type. In our specific applications of needle steering (Sec. 6), we ensure that the deformation section, spanning around $5-6$ mm, is larger than the measurable step size of the used sensors, which provide new force and position data with a frequency of 150 Hz (i.e., every 33.35 μm of needle progress at a speed of 5 mm/s) with a conservative maximum error of 3 mm. Obviously, those parameters depend on the body region (e.g., spinal cord, breast, or liver) and the choice of image-guided technique, and vary with each needle-steering application or scenario. In any case, those parameters need to be known beforehand (and determined empirically).

Concretizing critical and target regions as spheres, we use initial, partial knowledge about surrounding organs or bones, e.g., vertebral bodies, and model a critical region as a collection of overlapping spheres that cover that region. The radius of a sphere is defined for the specific tissue region via the assumed size of physical obstacles and inhomogeneities; the specifics of the scenario also determine how to define which position deviation can be tolerated to still ensure that the needle cannot enter known critical regions even if it deviates from the plan. The safety check during strategy synthesis (Sec. 4.2) then calculates for each critical region whether the distance of the needle tip to the center sphere is greater or less than the specified radius. Fig. 4 lists one of the major checks, the function `Check_CR_Reached`, and also shows that our implementation in Uppaal Stratego stores critical regions in the two data structures `CR_center_sphere[CR_COUNT][3]` and `CR_radius[CR_COUNT]`; we get back to those data structures in Sec. 5.1. Worthwhile noting, were all CRs known statically (as well as the needle motion), strategy synthesis could take place entirely offline; yet, this assumption is not realistic. Before we explain how to move the offline game to an online game, we complete the description of the offline setting with a summary of the underlying formal model.

4.5. The formal model: a network of timed automata

Timed games, as we have seen, build on timed automata (Sec. 3). For the needle-steering game we provide a network of 4 timed automata: *User*, *Needle*, *Tissue*, and, for the safety checks, *Checker*. In Uppaal parlance, all models are templates as they are instantiated anew in each iteration of the online loop. Typical for that kind of model, the sizes of the automata are small, ranging from 2 to 10 locations with just one or two action transitions per location.

In our network, *Tissue* and *Device* synchronize via *do_tissue_action*, while the *User*, in control of the actions, synchronizes with the *Device* via *do_push* and *do_rotate*; the user's third action, *do_pull* (see Sec. 4.2) is not part of the model since it is relevant only for model updates in an online setting, and would otherwise incur path fragments of infinite lengths: $s \xrightarrow{do\_push} s' \xrightarrow{do\_pull} s$. The *User* also synchronizes with the *Checker* via *do_check*, which carries out different safety checks (Sec. 4.2), including the function `Check_CR_Reached` we have shown already (Fig. 4). This function and all other checks are written in C++ and are part of the code section of the Uppaal network. The C++ part further includes customized data structures for our region representation of the geometrical space, physics functions and a data representation for the needle and its motion per time step, or when rotated, as well as linear algebra functions (matrices and matrix-vector operations include auxiliary functions) for geometry. For completeness, we list the network in 3; for the details of the network, including its code part, we refer to our previous work [66].

5. Online strategy synthesis

The uncertainties of needle motion and tissue behavior are the two main reasons why needle steering should not be interpreted as an offline game: a controller following a merely offline generated strategy could produce a needle trajectory in the real world that differs from the one produced in the offline model so, that the needle might no longer be able to reach the target or might be about to navigate through a critical region (see Fig. 1). The online approach we propose, in contrast, allows for detecting both in time and re-initiating the synthesis safely with an adjusted model. The online approach requires sensor measurements of the executing application on the one hand, and an aptly prepared model on the other hand.

5.1. From offline to online game

For the needle-steering problem, two measurements suffice, both concerning the needle tip: a position sensor, which provides the current location of the needle, and a force sensor on the tip, which observes an increase of force. The latter can be used to detect the upcoming transition to a critical region (see Sec. 4.4), the former to determine whether the target remained reachable at all. Specific for our approach, not the actions but a timer controls when those values are read; and they are used only to decide whether

```
bool Check_CR_Reached() {
  for (i = 0; i < CR_COUNT; i++) {
    tip_center_dist = sqrt(
          pow(CR_center[i][0]-tip_pos[0],2)
        + pow(CR_center[i][1]-tip_pos[1],2)
        + pow(CR_center[i][2]-tip_pos[2],2));
    tip_region_dist = tip_center_dist - (CR_radius[i] + deviation);
    if (tip_region_dist <= 0) {return true;}}
  return false;}
```

Fig. 4. The `Check_CR_Reached` function in Uppaal for safety checks.

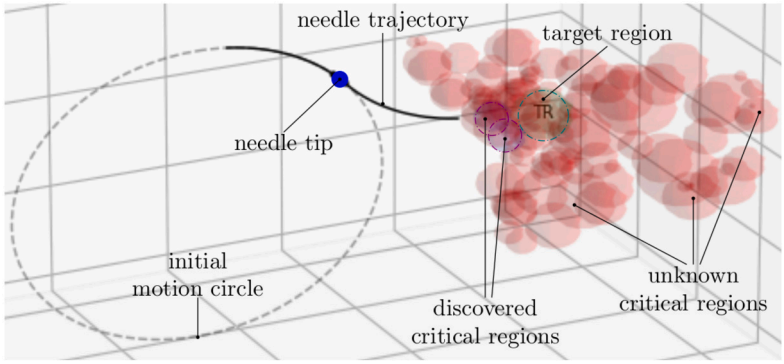


Fig. 5. Online discovery of a critical region (CR) of a scenario with no initial knowledge about the environment. Red spheres: unknown CRs, purple spheres: online discovered CRs. One CR has already been discovered; the needle tip just detected a second CR and is about to pull back. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

the current plan must be discarded. Preparing the model accordingly is easy and restricted to its data section: the parameters of the needle tip need to be represented explicitly so that they can be bound to changing sensor values, and the data structures for critical regions need to be dynamic so that each online discovered critical region can be added. Fig. 4 above displayed already the most important variables: the 3-dimensional array that represents the needle by its tip, `tip_pos[3] = {x, y, z}`, and the two data structures for critical regions (`CR_center_sphere[CR_COUNT][3]` and `CR_radius[CR_COUNT]`), which are both dynamic. The variable `CR_COUNT` is an internal counter that is incremented with each new critical region detected. Thus, no critical region is ever discarded, only new ones can be added. For an intuition of the effect of moving synthesis from offline and to online, Fig. 5 illustrates the online exploration of the environment: CRs may be unknown (marked red), but if they are on the needle path, they will be discovered as CR (marked purple); the needle then pulls back to follow a different path and, successively, other CRs might be discovered.

For flexible needles with bevel tip, which take circular paths through the tissue, the possibility of a deviation implies that their motion circle is known dynamically only. Once a deviation has been detected, thus, a separate fitting step determines the actual motion circle before the next round of strategy generation commences and, if needed, adjusts its radius. The fitting step, too, is based on the actual location of the needle tip that the position sensor provides and depends on the needle feed, which is the mechanism of inserting a needle along its longitudinal axis (in our underlying hardware experiments, we empirically determined a needle feed of 20 mm).

Before we explain the algorithm OCS for online controller synthesis, we summarize which physical quantities must be available and properly set for the online approach to work:

1. Detection regions can be detected by measuring the force on the needle tip. The frequency of force measurement and corresponding data handling is within the time span in which tissue may deform before rupture.
2. (Safe) values for the radius of critical regions are known as well as allowances before a needle position is flagged as deviating (see the parameter `deviation` in Fig. 4).
3. The new motion circle can be computed dynamically; if based on a needle feed, it is known which feed yields a sufficiently accurate circle.
4. The time between the online update of the model and the generation of the new strategy is short enough so that any changes in the state of the real world do not render the new strategy unsafe; in particular are changes in the state of the real world not sudden.

We are now ready to present the main contribution, the algorithm OCS.

5.2. OCS: an algorithm for online controller synthesis

Algorithm 1 outlines the steps of OCS. We denote by SynthesizeStrategy the algorithm SOTFTR for strategy generation in timed games [40] and use this algorithm for offline synthesis. Compared to the original offline algorithm (l.12), online synthesis runs in a loop, the *feedback loop*, updating the timed game model and re-generating the winning strategy based on newly available run-time information. The loop terminates if the needle reaches the target position (l.26-28) or no more winning strategy can be synthesized from the initial position (l.13-14).

The algorithm's input is, firstly, the reachability real-timed game, where its goal, here T, is not yet reached; secondly, the assumed critical regions radius defining the radius of newly added critical regions to the model; third, the needle tip's initial and target position (l.1-4). Initially, the needle's list of position and force data is empty, no safe motion plan exists to steer the needle to the target, and *motion_circle*, the transformation matrix for computing the needle's circular path, is the identity (l.5-8). The FitCircle routine starts by pushing forward the needle (in our experiment by 20 *mm*), monitors the position data, and based on the needle feed, approximates the actual motion circle matrix of the needle (l.9). The InitializeTG routine (re-)initializes the timed game model with the currently

Algorithm 1 OCS : Online Strategy Synthesis for a timed reachability game.

Require: *TGM*, reachability winning condition, assumed critical regions radius, initial position, target position
Ensure: True if the target position is reached, false otherwise

```
 1: M, T ← TGM, reachability winning condition
 2: cr_radius ← assumed critical regions radius
 3: init_pos, current_pos ← initial position
 4: target_pos ← target position
 5: motion_circle ← identity matrix                          ▷ transformation matrix for circular path
 6: motion_plan ← None                                       ▷ no motion plan exists
 7: target_reached, force_exceeded, deviation_exceeded ← false
 8: pos_data, force_data ← []
 9: pos_data, current_pos, motion_circle ← FitCircle         ▷ computes motion circle
10: while target_reached = false do                          ▷ target is not yet reached
11:     M ← InitializeTG(CURRENT_POS, MOTION_CIRCLE)         ▷ initializes the actual TGM
12:     motion_plan ← SynthesizeStrategy(M, T)               ▷ synthesizes a strategy
13:     if motion_plan = None ∧ current_pos = init_pos then  ▷ no safe motion plan at the initial position
14:         return target_reached                            ▷ target unreachable
15:     end if
16:     if motion_plan = None ∧ current_pos ≠ init_pos then  ▷ no safe motion plan
17:         current_pos ← Pullback, continue                 ▷ retract needle to previous position
18:     end if
19:     start MonitorExecution, start Execute(MOTION_PLAN)
20:     while motion_plan ≠ None ∧ target_reached = false do ▷ feedback loop starts
21:         wait(safe_time_window)                           ▷ runs motion plan safely
22:         pos_data, force_data ← MonitorExecution          ▷ adds new observations
23:         force_exceeded ← CheckSafety(FORCE_DATA)         ▷ checks safety
24:         current_pos, deviation_exceeded ← CheckDeviation(POS_DATA,MOTION_PLAN)  ▷ checks deviation
25:         target_reached ← CheckTargetPos(CURRENT_POS, TARGET_POS)  ▷ checks target reached
26:         if target_reached then                           ▷ case 1: target reached
27:             stop MonitorExecution, stop Execute(MOTION_PLAN)
28:             return target_reached                        ▷ target reached
29:         end if
30:         if force_exceeded then                           ▷ case 2: critical region detected
31:             stop MonitorExecution, stop Execute(MOTION_PLAN)
32:             motion_plan = None                           ▷ new safe motion plan required
33:             M ← AddCriticalRegion(CR_RADIUS)             ▷ add new critical region to TGM
34:             current_pos ← Pullback                       ▷ retract needle to previous position
35:         end if
36:         if deviation_exceeded then                       ▷ case 3: needle tip deviates
37:             stop MonitorExecution, stop Execute(MOTION_PLAN)
38:             motion_plan = None                           ▷ new safe motion plan required
39:             pos_data, current_pos, motion_circle ← FitCircle  ▷ adjusts motion circle
40:         end if
41:     end while
42: end while
43: return target_reached                                    ▷ target reached
```

available knowledge about the actual system behavior, the current needle tip position, and the motion circle matrix (l.11). After, the SynthesizeStrategy($\mathcal{M}, T$) routine, if possible, generates a winning strategy using classical offline controller synthesis of timed games. A winning strategy defines an executable sequence of actions, a motion plan. If multiple winning strategies exist, one strategy resp. motion plan is selected randomly. If the strategy synthesis fails but the needle is not yet back at the initial position, the needle will retract (in our experiments: by 30 *mm*) (l.17), and the outer loop restarts.

If the strategy synthesis was successful, the Execute(*MOTION_PLAN*) thread executes the motion plan instructions while the MonitorExecution thread monitors the needle trajectory and, once *safe_time_window* has elapsed, saves its position and force data (l.22). The routines CheckSafety(*FORCE_DATA*), CheckDeviation(*POS_DATA*), and CheckTargetPos(*CURRENT_POS, TARGET_POS*) check the newly available data (l.23-25). The feedback loop stops or finishes the motion plan execution for three reasons: first, the needle reaches the target position (l.26-28); second, the force data values rise too high relative to each other (l.30-34); and third, the actual needle position deviates too much from its planned position (l.36-39). If *force_exceeded* is true, the safety of the actual motion plan is no longer given and the plan must be rejected. For the following strategy synthesis step, AddCriticalRegion(*CR_RADIUS*) updates the *TGM* set of critical regions by the newly encountered one (l.33) and, for safety reasons, the needle is pulled back; then, the outer loop restarts. If *deviation_exceeded* is true, FitCircle reinitializes the motion circle matrix before the outer loop restarts. The OCS repeats the steps of initializing the *TGM*, synthesizing and executing a new strategy, monitoring and verifying the execution, and, if necessary, retracting the needle, and adapting and updating the timed game according to the latest information until the target is reached or the target is unreachable.

5.3. Reachability and safety

In this section we prove two important properties of algorithm OCS, reachability and safety, and show under which conditions the algorithm fails to terminate. Since the proofs refer to properties of the underlying offline synthesis, we state all relevant properties upfront in the following fact.

Fact 1. Offline synthesis, denoted as SynthesizeStrategy, refers to the algorithm SOTFTR (Symbolic On-The-Fly Algorithm for Timed Reachability Games) from the literature [40]. The following properties hold:

1. A strategy is a function from execution sequences of a transition system to actions that instruct the controller in any given state which action to take. A winning strategy is a strategy that guarantees that the controller reaches the goal [5].
2. SynthesizeStrategy solves the reachability game: it returns a winning strategy, if one exists; and every strategy it returns is a winning strategy (see [40], proof for algorithm SOTFTR).
3. SynthesizeStrategy terminates (see [40], proof for algorithm SOTFTR).

Theorem 1 *(Reachability). Let $\mathcal{M}$ be a network of timed game automata, let $TR_{\mathcal{M}} = \{t\}$ be the target region with target t, and denote by $Region(\mathcal{M}) = SR_{\mathcal{M}} \cup TR_{\mathcal{M}} \cup DR_{\mathcal{M}} \cup CR_{\mathcal{M}}$ the set of regions in $\mathcal{M}$. Let $\rho = s_0 \to s_1 \to \ldots \to s_n$ ($s_i \in Region(\mathcal{M})$, $n \geq 0$) be the finite trajectory computed by algorithm OCS. Then, $s_n = t$, i.e., OCS computes a winning strategy for the reachability game.*

Proof. Assume a finite motion plan. This motion plan has been generated in l.12 of the algorithm, by the offline synthesis. Subsequently, this motion plan is either followed (l.19) or aborted (l.27,l.31,l.37), but never modified. The claim now follows from Fact 1.(2). □

Theorem 2 *(Safety). Let $\mathcal{M}$ be a network of timed game automata and let $Region(\mathcal{M})$ be as before. Denote by $Safe(\mathcal{M}) = SR_{\mathcal{M}} \cup TR_{\mathcal{M}} \cup DR_{\mathcal{M}}$, $Safe(\mathcal{M}) \subseteq Region(\mathcal{M})$, the subset of safe regions of $\mathcal{M}$. Let $\rho = s_0 \to s_1 \to \ldots \to s_n$ ($s_i \in Region(\mathcal{M})$, $n \geq 0$) be the finite trajectory computed by algorithm OCS. It holds: if $s_0 \in Safe(\mathcal{M})$, then $s_i \in Safe(\mathcal{M})$ for all states s_i, $i > 0$, in ρ, i.e., OCS computes a winning strategy for the safety game.*

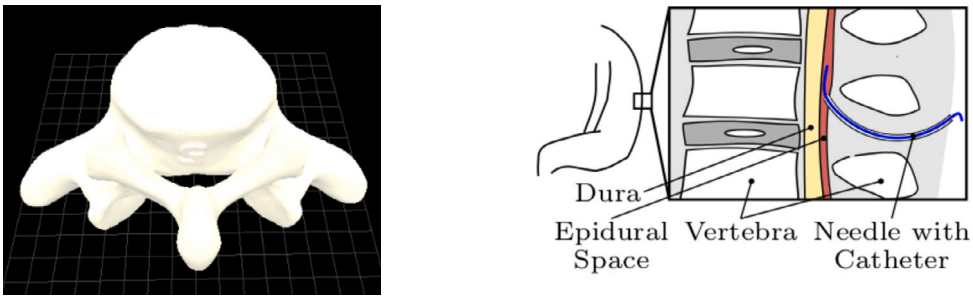
Proof. We prove the claim by induction on the length of the sequence ρ. If ρ is of length 1, the claim follows from the fact that the action $s_0 \to s_1$ is guarded with a safety check (Sec. 4.2, Sec. 4.5) and that, by assumption, s_0 is safe. The inductive step follows with a similar argument. □

Lastly, we turn to the termination condition of our algorithm. From Fact 1(3) it is known that offline synthesis terminates. Since all other subroutines of OCS terminate as well, it suffices to directly consider the feedback loop. In the following, we call the current start state a *previous position* if it served as start state in a previous invocation of SynthesizeStrategy and we call a motion circle *new* if it differs from any previous motion circle computed by FitCircle for a previous position.

Theorem 3 *(Conditional termination). Consider the invocation of SynthesizeStrategy in line l.12 of the algorithm, assume it is invoked for at least the second time, i.e., the while-loop has run at least once, and let p be the start state for the current invocation of SynthesizeStrategy. Algorithm OCS may fail to terminate if p is a previous position and one of the two conditions holds (i) in between its two SynthesizeStrategy invocations, no critical region has been detected; or (ii) its motion circle is new. Otherwise, algorithm OCS terminates.*

Proof. Proof by case distinction.

- Case 1: p is a previous position but in between its two SynthesizeStrategy invocations, a critical region has been detected. Case 1 applies necessarily if the loop body in the preceding iteration was left because of an exceeding force at the needle tip, and it might apply if the loop body was left because of a needle deviation; additionally assume in the latter case that the motion circle is not new. In either case, SynthesizeStrategy, deterministically, generates the same function ("motion plan") as before for p. In the meantime, however, a critical region has been detected and the needle was rotated. Thus, some guards become false and some previous paths are blocked; returning to p repeatedly, therefore ultimately reduces the motion plan to the loop termination condition None (unless the target has been reached).
- Case 2: p is a previous position but either the motion circle is new or no critical region has been detected in between its two SynthesizeStrategy invocations. The former case may result from FitCircle, the latter applies if the loop bodies in between were all left because of a needle deviation, i.e., the needle has been jumping around erratically. In the former case, a new strategy is generated each time, and there is no bound on the number of times SynthesizeStrategy can be invoked. In the latter case, SynthesizeStrategy re-generates the strategy from before. But neither did the set of enabled transitions change (this is different from Case 1) nor does the algorithm check for state or strategy duplication. As a consequence, SynthesizeStrategy could be invoked infinitely often.
- Case 3: p is not a previous position. Case 3 can never apply if in the previous iteration the loop body was left because of an exceeding force, since pullback always returns to a previous position, but it may result from needle deviation. Since the tissue region is bound and the needle position is represented by integers, only a finite number of non-previous positions exist. □



(a) System environment: Vertebra

(b) Cross-section of three vertebrae (depicted in white) of the spine

Fig. 6. *Peridural anesthesia* (PA).

Theorem 4. *Upon termination, Algorithm OCS returns a strategy that ensures both reachability and safety, provided such strategy exists.*

Proof. Follows directly from Theorem 1 and 2. □

Remark 2. Whether or not such a strategy exists depends on the layout of critical regions and the moves of the needle, and will change in the course of OCS in two ways: on the one hand, more critical regions are discovered, restricting the search space for winning strategies. On the other hand, the needle may deviate slightly from its predicted position, so that targets become reachable that previously were not.

5.4. Implementation

We implemented the online concepts covered in Sec. 5 as a Python program. The implementation consists of three main components: the needle device, the online strategy synthesizer, and the externally called Uppaal Stratego tool. The needle and strategy synthesizer communicate via a TCP interface, which is used to exchange needle motion instructions (e.g., *do_rotate* instructions) and measured data. Such interface allows a simple exchange of the needle data source (e.g., using either a virtual or an actual physical needle), as long as the component in use handles the given set of instructions. The online strategy synthesizer continuously checks for deviations of force increase rates and needle positions, and, when the pre-defined bounds are exceeded, initiates respective countermeasures; in our concrete application, these countermeasures are a pullback and slight push when a critical region is detected, and a position correction in the model when the observed position deviates, each followed by the synthesis of a new strategy. Whenever an initial or updated strategy is needed, the synthesizer spawns the strategy synthesis routine of Uppaal Stratego as a separate process (using Uppaal's `verifyta` module with `-print-strategies` flag), and extracts concrete instruction sequences from the returned strategy data afterwards. Uppaal Stratego itself uses the model introduced in Sec. 4.5, whose XML representation (i.e., the standard file format of Uppaal models) is updated with the current knowledge about the needle position and orientation, as well as the detected critical regions. As core Uppaal does not support *float* variables, all affected data variables are scaled by a given factor (1000) and cast into the *integer* domain; to mitigate the state-space explosion problem, we only allow 4 rotation angles, restrict the number of rotations, and require interleaving push actions [66].

6. Experiments

In this section, we demonstrate OCS using simulations of two applications of needle steering, *peridural anesthesia* (PA) and *predefined needle trajectory* (PNT). In PA, epidural anesthesia is administered by inserting an anesthetic into the epidural space, which is the target region while the dura and the vertebrae represent critical structures that need to be avoided (Fig. 6b). PNT is a more general approach, where the flexible needle should follow a predefined trajectory, avoiding critical structures and reaching the target (Fig. 7b) but may encounter unknown obstacles, which it must not puncture. For both applications, strategies are motion plans, i.e., sequences of instructions for steering the needle, and the task for OnSS is to synthesize motion plans that cross no critical region and reach the target, provided it is reachable at all.

6.1. Setting

For both applications, we keep the model of the environment, including critical regions and target region, fixed throughout the experiments. In both cases, we model critical and target regions as spheres. For PA, we took a vertebrae model in STL format (Fig. 6a) as input, reduced the size to 50 vertices and converted the triangles of the reduced vertebrae model into spheres by calculating the incenter and inradius of each triangle. We modeled the vertebral body by 100 critical regions and placed the target region in the gap of the vertebral body. For PNT, we arbitrarily selected the needle trajectory that describes a push forward by 30 mm, followed by a rotation by 180^0, followed by another push forward by 30 mm (Fig. 7b). A realistic assumption for PNT is that critical regions form a tunnel towards the target region. As in [66], we took the trajectory and, modeling critical regions as spheres as well, arranged 48

Table 1
Hyperparameters deviation tolerance and threshold: impact on the number of deviations.

parameter	# deviations		time in sec.		# target reached	# timeouts
settings	avg	range	avg	range	simulations	after 300 sec.
PA-k100-dev041-c5	0.2	[0 , 1]	25.3	[17.0 , 33.7]	4	26
PA-k100-dev043-c5	1.8	[0 , 4]	101.7	[17.0 , 250.9]	13	17
PA-k100-dev045-c5	1.2	[0 , 4]	91.2	[17.1 , 209.4]	26	4
PA-k100-dev047-c5	0.5	[0 , 3]	41.5	[17.0 , 153.8]	30	0
PA-k100-dev049-c5	0.1	[0 , 1]	34.2	[17.0 , 96.5]	30	0
PA-k100-dev045-c1	0.9	[0 , 4]	84.2	[17.3 , 227.3]	14	16
PA-k100-dev045-c3	1.0	[0 , 5]	73.4	[17.0 , 225.2]	22	8
PA-k100-dev045-c5	1.2	[0 , 4]	91.2	[17.1 , 209.4]	26	4
PA-k100-dev045-c7	0.4	[0 , 3]	35.6	[17.1 , 151.8]	29	1
PA-k100-dev045-c9	0.8	[0 , 5]	61.6	[17.0 , 242.0]	28	2

Table 2
Hyperparameter initial position: impact on the number of initial strategies.

simulation	initial position in mm	target reachable?	# initial strategies
PA-vertebrae-k100-p000	(0,0,0)	yes	20
PA-vertebrae-k100-p500	(5,0,0)	yes	3
PA-vertebrae-k100-p050	(0,5,0)	yes	22
PA-vertebrae-k100-p005	(0,0,5)	yes	10
PA-vertebrae-k100-p3000	(30,0,0)	no	0

spheres with a radius of 5 mm as critical regions at a distance of 15 mm on circular orbits around the trace; we placed the target region at the end of the needle trace (Fig. 7a).

For both applications, we assume six different degrees of initial knowledge about the vertebrae body (for PA) and the tunnel (for PNT), namely, whether 0%, 20%, 40%, 60%, 80%, or 100% of the critical regions are known before the experiment runs. Using purple for (previously unknown but now) discovered and red for unknown critical regions, Fig. 5 illustrates for PA the worst case of 0% knowledge of the environment, while Fig. 7a is an example of partial initial environmental knowledge for PNT where 40% of the tunnel obstacles are known. In the (unrealistic) best case of 100% knowledge, all critical regions are known already offline, and offline strategy synthesis would suffice to generate safe strategies if one assumes an ideal needle. A real needle, however, may deviate from the planned trajectory by physical reasons—for the flexibility of the needle tip or the texture of the tissue. Therefore, even full knowledge of the environment might require online adjustments, though with a low probability.

In all experiments, the critical and target regions are virtual, and so is the needle. The *virtual needle* performs the same steps as a real needle would. Since the real needle may deviate from its predicted trajectory, its actual motion circle needs to be determined anew whenever a critical region was detected and the needle pulled back. By analogy, and to also account for the possibility of deviation, we apply this fitting step of the motion circle to the virtual needle as well but approximate it only (by allowing a deviation of up to 2 mm). For the experiments, we used Linux Mint 20.3 Cinnamon 5.4.7 system with Intel Core i7-9750H CPU and 16 GB RAM, and version 4.1.20-7 of Uppaal Stratego.

6.2. Hyperparameters

For all simulations, we specify the permissible needle position deviation and force excess, defined by the deviation/force *tolerance* and a *threshold*, and the initial needle position via hyperparameters. The threshold determines the maximum number of position/-force data that may consecutively exceed the tolerance. This section demonstrates the effects of different parameter settings, using simulations of the application PA.

For the deviation hyperparameters tolerance and threshold, we carried out 30 simulations each for 10 different parameter settings. Identical for all simulations are the initial position in mm (0, 0, 0), the target position in mm (0,-52,-25), the target radius of 5 mm, and the force tolerance and threshold. In addition, we assume 100% knowledge (=k100) of the environment, i.e., strategy generation knows precisely how the vertebral body is positioned in the tissue, which largely rules out the detection of critical regions within the OCS (for possible issues, we refer to the discussion on Fig. 8a). For the parameter settings PA-k100-**dev041**-c5 to PA-k100-**dev049**-c5, we specify that the threshold is 5 and vary the deviation tolerance of the needle position from 0.41 mm to 0.49 mm. For the parameter settings PA-k100-dev045-**c1** to PA-k100-dev045-**c9**, we specify that the deviation tolerance is 0.45, and vary the threshold from 1 to 9 allowed consecutive detected deviations. Since our simulations are based on approximations of the needle movement, we use these values in our OCS to force strategy regenerations based on deviations. If the target has not yet been reached, we abort the simulations after 5 minutes.

We present the results in Table 1: Columns 5 and 6 show how many of the 30 experiments had a timeout and in how many simulations the OCS successfully guided the needle to the target. For the successful simulations, columns 2 and 3 show the average and range of the number of deviations and simulation times each. The parameter settings PA-k100-dev043-c5 to PA-k100-dev049-

c5 show that, the smaller the deviation tolerance, the higher the average number of deviations of the needle, the simulation time, and the number of timeouts. For example, in PA-k100-dev041-c5, each of the four successful simulations has at most one deviation. However, there are 26 timeouts, which are 9 unsuccessful simulations more than in PA-k100-dev043-c5. For PA-k100-dev045-c1 to PA-k100-dev045-c7 in the lower half of the table, the relations are similar, though less pronounced. Most visible is the parameter's influence on the number of timeouts, which is cut in half per parameter setting. Generally, the smaller the permissible threshold, the more frequently the simulations do not reach the target within 5 minutes. The figures of PA-k100-dev045-c9 do not confirm that trend, though, but this attributed to the random selection of strategies.

The second important hyperparameter is the initial position of the needle: its choice impacts the number of motion plans that can be generated initially. In Table 2, we show the results for 5 simulations with the initial positions in mm: (0,0,0), (5,0,0), (0,5,0), (0,0,5), and (30,0,0); note that the first 4 positions are very close to each other. Across all simulations we keep fix the following hyperparameters: the target position in mm (0,-52,-25), the target radius of 5 mm, the deviation threshold of 5, the permissible deviation of 2 mm, and the force tolerance and threshold; with a permissible deviation of 2 mm, we exclude strategy regenerations due to small vibrations or other forms of jitter a needle realistically is subject to. In addition, we assume (again) complete knowledge of the environment, which excludes strategy generation due to detected critical regions. Thus, our simulation outputs correspond to those of an offline strategy synthesis. The simulations show that it depends on the initial position whether a strategy can be generated at all and, if so, how many. For the initial positions (0,0,0), (5,0,0), (0,5,0), (0,0,5), even though they are close, the number of generated motion plans ranges from 3 to 22. The target cannot be reached from position (30,0,0).

The values for the hyperparameters force tolerance and threshold cannot be chosen freely as they depend on the tissue properties and the needle specification. We remark that we confirmed the values used in the following two experiments in separate experiments in our hardware configuration. Based on the force tolerance and the needle speed, we can calculate how to set the safety radius around newly detected critical regions. In our simulations, we assume the needle's speed to be constant. For all following experiments, the initial position in mm is (0,0,0), and the deviation tolerance and threshold are 2 mm and 5. This parameter setting rules out strategy regenerations due to needle jitter. Finally, experiment 1 (Sec. 6.3) and experiment 2 (Sec. 6.4) assume different positions of the vertebral body and the target point relative to the initial point for the application PA.

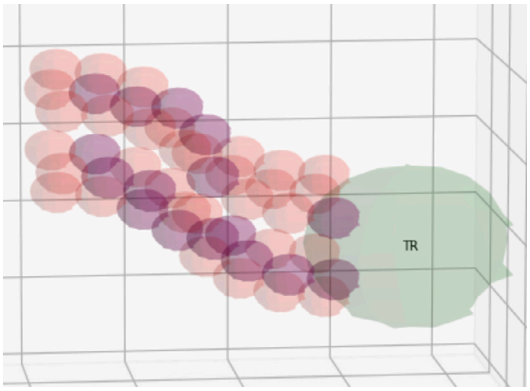
6.3. Experiment 1

In Experiment 1, we place the target so that it is reachable. For PNT and PA, we run for each of the 6 degrees of initial knowledge 30 simulations each and report the number of detected critical regions (Figs. 7c, 8a) as well as the simulation time (Figs. 7d, 8b). In Table 3, we detail for one simulation each the operations specific for OCS and list the number of motion plans that are available after each adjustment step. We now discuss each kind of result separately.

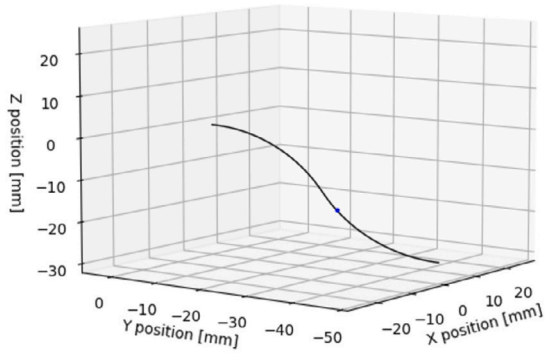
In Figs. 7c, 8a, we plot the number of detected critical regions per initial degree of knowledge. Since critical regions are only detected when a motion plan fails to remain safe, this number is typically much lower than the total number of critical regions in the environment; if the number of detected critical regions is 0, it means that the initial, offline generated strategy is safe already along the entire needle trajectory. The plots show, as expected, that the number of detected critical regions decreases as the initial knowledge of the environment increases, though for PNT slightly faster than for PA. For PA, the simulations with 0% knowledge result in the detection of up to 6 CRs (see the green bar), while with 100% knowledge at most 1 CR was detected; in between, up to 5-4-2-1 CRs were detected for the knowledge levels 20%-40%-60%-80%. For PNT, the maximal number of detected critical regions forms the sequence 7-6-6-4-0-0 and the plot is, overall, more scattered. Generally, the vertebrae environment of PA makes it more likely that the initial motion plan is already safe throughout; once faced with a critical region, however, it is more difficult for a motion plan to get back to a safe trajectory. For PNT, the situation is the other way around: it is unlikely that the initial motion plan with which the tunnel is entered, will stay safe as the needle moves along, but unsafe deviations from the trajectory can be corrected with fewer operations.

Altogether, Figs. 7c, 8a illustrate the necessity of OnSS — many motion plans encounter one or more critical regions, and none of them would be safe if they were not adjusted online, by OCS. The two figures show at the same time that initial motion plans can already succeed even for partial knowledge only—in which case no online adjustments are needed. For PNT, this is the case the first time for 40% knowledge (see the blue bar), for PA, somewhat unexpected, already at 0% knowledge, where as many as 8 of the 30 simulation get by without adjustments; this number increases for higher degrees of initial knowledge (to 8, 15, 19, 19, 26). Those high numbers confirm that initial motion plans for PA may avoid critical regions even without knowing them. Finally, an interesting result is shown by the simulations for PA with 100% knowledge, where a critical region is detected in 4 simulations (see the yellow bar in Fig. 8a). In theory, complete knowledge should never result in a motion plan that entails the detection of a critical region. In fact, those 4 cases are artifacts of our overapproximation of critical regions, which we embed in detection regions (recall Fig. 1): none of those 4 motion plans crosses a critical region, but they touch on detection regions and therefore cause OCS to dismiss them and initiate strategy regeneration.

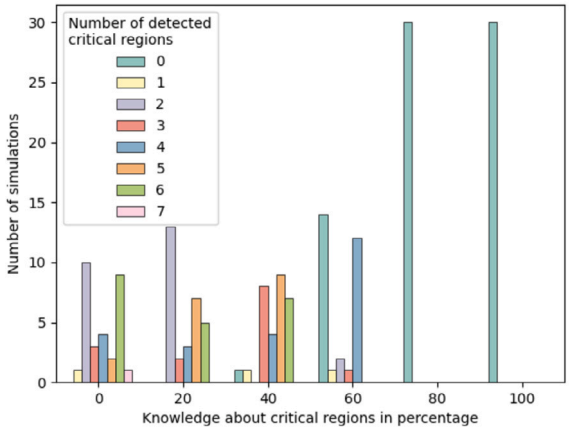
Our decision of overapproximating critical regions has a second consequence, which can be seen in Figs. 7d, 8b. As OCS proceeds and more and more critical regions are detected, it might happen that their surrounding detection regions overlap so that they not only block paths through critical regions, as desired, but at the same time block paths that, unharmfully, stay in detection regions. In the end, targets could thus become unreachable. The experiment in Fig. 7d also shows that PNT is quite vulnerable to that issue: in 60 out of 180 ($= 6 \cdot 30$) simulations the target becomes unreachable; for PA, this is the case for 4 simulations only (Fig. 8b). The two applications differ also with



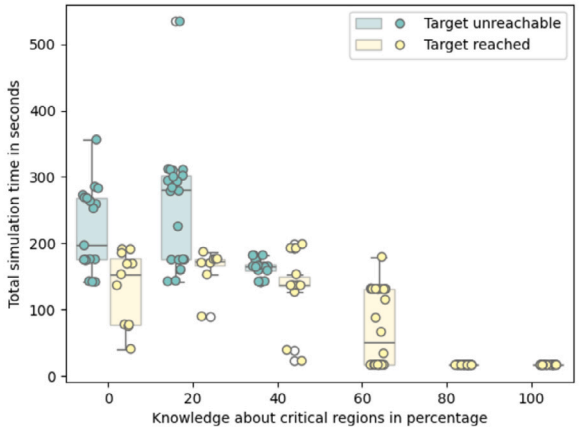
(a) PNT 40 % initial environmental knowledge



(b) Needle trajectory "f_30.0_r_180.0_f_30.0"



(c) PNT: Detected critical regions per simulation



(d) PNT: Simulation time per simulation

Fig. 7. Application *predefined needle trajectory* (PNT) - experiment 1.

respect to their run-time behavior: compared to 0% and 20% initial knowledge, the median performance of PA improves significantly for 40% initial knowledge, yet stays roughly there for all higher levels of initial knowledge. PNT, on the other hand, with its high number of unreachable targets, becomes practical only from 40% knowledge onwards and high-performant from 80% onwards.

The results in Table 3 are based on 1 simulation per initial knowledge level each. The OCS cycle—of aborting a motion plan, readjusting the needle, and generating a new plan, or at least attempting to—is broken down in its major steps to provide a profile for each simulation. As example we consider the simulation for PNT with 20% initial knowledge, which lists 3 CRs detected, 7 pullbacks, and 2 *back to initial state*. 7 pullbacks for 3 critical regions indicate that a single pullback cannot rectify the situation in all 3 cases and suggests that the 3 adjustment attempts require 2 or 3 pullbacks. Reaching the initial state with a sequence of 2 or 3 pullbacks, at the same time, suggests that 2 of the 3 adjustments take place close to the starting point of the navigation.

The last column in Table 3 provides a coarse-grained profile: it counts how many motion plans are generated in each iteration of the feedback loop (or 0 if no such motion plan exists). Naturally, this number is highest at the beginning of OCS where fewer obstacles are known than in the course of online discovery and are at the start point also higher for lower levels of initial knowledge than for higher. For 100% knowledge (and no deviation), all initial plans are safe over the whole distance the needle travels (for possible issues due to overapproximation, see above discussion). For all other knowledge levels, the further course of OCS depends both on the particular initial plans selected by the algorithm and on the particular adjustment attempts, which might make many motion plans possible in one go, or leave none at all. In more detail, the entry [328, 22, 0, 10, 0, 0, 0, 1] for PNT-k20 is read as follows: initially, 328 motion plans exist. One plan is selected and executes until it hits a CR. Then, strategy regeneration is invoked, but now only 22 strategies can be found. Again, one strategy is selected and, again, executes until a CR is hit. Now the adjustment was not successful since no strategies could be generated, making further adjustments necessary. The new needle position opens the possibility of 10 strategies, followed by 3 unsuccessful strategy-generation attempts. After the third unsuccessful attempt, the needle is pulled back far enough so that a strategy exists, and in fact reaches the target. Comparing the numbers for PNT and PA, the range for the number of initial motion plans is larger for PNT (from 0 to almost 421) and the spikes are more distinct (e.g., from 5 to 144 plans). One time more, we see that the different layouts in the tunnel (PNT) and the vertebrae (PA) scenario determine the number

Table 3
OCS profile for six selected PNT and PA simulations (target reachable).

simulation	time in sec.	# deviations	# CRs	# pullbacks	# back to initial state	# strategies per synthesis
PNT-k0	153.0	0	4	8	2	[421, 22, 0, 10, 70, 0, 0, 0, 1]
PNT-k20	137.3	0	3	7	2	[328, 22, 0, 10, 0, 0, 0, 1]
PNT-k40	194.3	0	6	10	2	[316, 11, 0, 5, 144, 8, 3, 0, 0, 0, 1]
PNT-k60	33.8	0	1	1	0	[10, 67]
PNT-k80	16.9	0	0	0	0	[2]
PNT-k100	16.9	0	0	0	0	[2]
PA-k0	174.8	0	3	9	2	[56, 0, 0, 21, 0, 0, 3, 0, 0, 6]
PA-k20	352.8	0	2	6	2	[56, 0, 0, 21, 0, 0, 14]
PA-k40	341.9	0	1	3	1	[54, 0, 0, 19]
PA-k60	75.4	0	1	3	1	[21, 0, 0, 3]
PA-k80	75.8	0	1	3	1	[20, 0, 0, 2]
PA-k100	17.7	0	0	0	0	[20]

Table 4
OCS profile for six selected PA simulations (target unreachable).

simulation	time in sec.	# deviations	# CRs	# pullbacks	# back to initial state	# strategies per synthesis
PA-k0	681.5	0	4	12	3	[36, 0, 0, 3, 0, 0, 0, 22, 0, 3, 0, 0, 0]
PA-k20	122.7	0	2	6	2	[21, 0, 7, 0, 0, 0, 0]
PA-k40	51.2	0	1	2	1	[15, 0, 0]
PA-k60	431.4	0	2	6	1	[17, 0, 0, 2, 0, 0, 0]
PA-k80	13.0	0	0	0	0	[0]
PA-k100	13.7	0	0	0	0	[0]

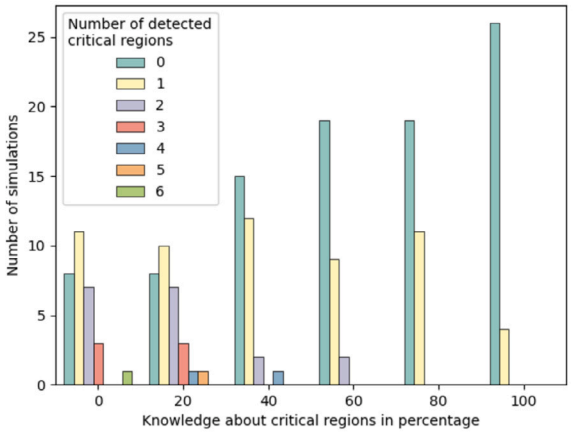
of options for navigating the needle safely; fewer options suggest more constraints and, generally, lead the needle more directly to its target.

6.4. Experiment 2

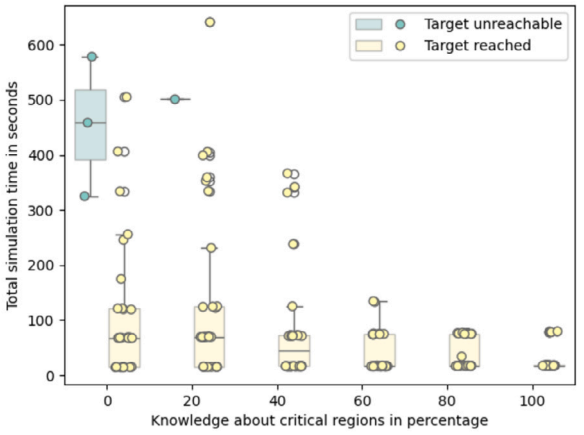
Given a particular environment, a particular target is not necessarily reachable. Since strategy synthesis is an optimistic algorithm, i.e., it assumes that a solution exists, it is interesting to see how OCS behaves when its task is infeasible. For experiment 2, we place the target and vertebrae so that it is not safely reachable even with 100% initial knowledge of all obstacles. Apart from that, we use the same setup as in experiment 1 and report the same numbers,
though we restrict ourselves to just the PA scenario: based on 30 simulations for each of the 6 knowledge levels, Figs. 8c, 8d list the number of detected critical regions and simulation times; based on 1 simulation run per knowledge level, Table 4 provides a simulation profile and the numbers of generated strategies.

As in experiment 1, the number of detected critical regions decreases as the knowledge of the environment increases (Fig. 8c). Since the target cannot be reached, however, strategy generation at 100% (and 80%) knowledge now returns no critical regions simply because no strategy exists (and not because they would safely be circumnavigated). For all other knowledge levels, adjustment and regeneration attempts are made as before, yet and as expected in the unreachable case, at larger numbers. At the same time, the total number of detected critical regions is comparatively low and comparable to the numbers in experiment 1, which implies, that the algorithm gets stuck quite early and, from a medical perspective, does not rummage in the tissue too broadly. Naturally, the larger number (compared to experiment 1) of regeneration attempts is reflected in higher run times, of up to almost a quarter hour for 0% and 60% initial knowledge (see Fig. 8d). While those numbers are about factor 3 higher than in the case of a reachable target, informal responses by practitioners suggest that they are still acceptable in practice. Perhaps unintuitively, the values for 60% knowledge in Fig. 8d are higher than for 40% knowledge or lower. Manual inspection confirms that this behavior is an artifact of the randomized choice of critical regions per initial knowledge: incidentally, a region that controls the initial number of motion plans and is known at lower knowledge levels, was no longer critical for 60% knowledge. This also explains the unintuitive higher number of initial motion plans for 20% knowledge compared to 40% and 60% initial knowledge. Lastly, comparing the OCS profile and the numbers of strategy generations in Table 4 with the figures in experiment 1, one observes, again as expected, more pullbacks, unsuccessful strategy generations, and longer sequences of unsuccessful generations in a row (see, e.g., the 4th-7th generation attempt at 20%: [21, 0, 7, *0, 0, 0, 0*]). Altogether, experiment 2 illustrates that the efforts and costs of OCS increase when the target is unreachable.

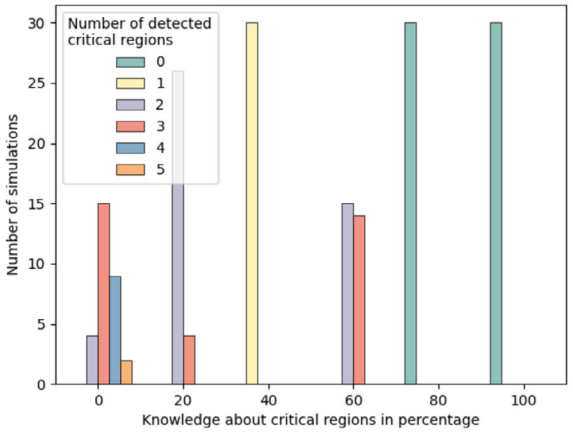
From an application point one would prefer paying more for the reachable rather than the unreachable case, when no treatment will take place. Alas, the optimistic approach of strategy synthesis proceeds differently, and in the end, as experiment 2 also shows, the costs remain practical.



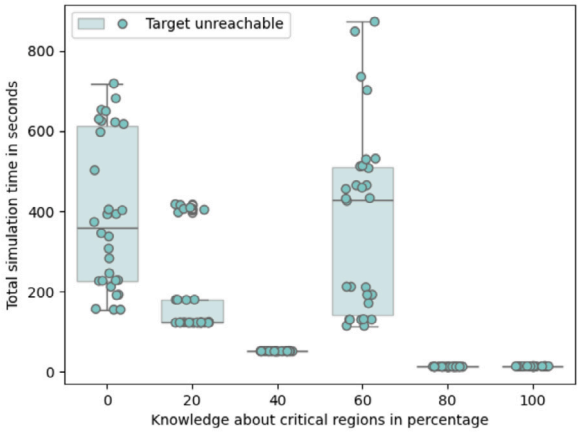
(a) PA (target reachable): Detected critical regions per simulation



(b) PA (target reachable): Simulation time per simulation



(c) PA (target unreachable): Detected critical regions per simulation



(d) PA (target unreachable): Simulation time per simulation

Fig. 8. Application *peridural anesthesia*: experiment 1 (target reachable) and experiment 2 (target unreachable).

6.5. Threats to validity

We now discuss the four kinds of threats to the validity of the experiments reported. Threats to construct validity, first, present motion-plan computations that differ from the ones a physician could devise. We distinguish two cases. The first case is that the physician finds a way to steer the needle to the target, while our algorithm determines that the target is not reachable. This case can actually happen because of overapproximations and simplifications in our algorithm—the region representation is an over-approximation and the needle movement has been abstracted to three operations—and cannot be entirely avoided; yet it presents no safety problem. The other case is that our algorithm finds a motion plan, but in reality no motion plan exists because the needle inevitably would hit a critical region. In principle, this case could happen since our safety argument is relative to the particular values of speed, force, and detection regions, and those values might not be synchronized (see Sec. 5.1). We mitigate this threat by determining these values not just mathematically, but through hardware simulations. The same safety problem could occur if critical regions would not be properly covered by detected regions. We currently double-check coverage visually; a check in software would be possible as well. A fundamental threat to construct validity, finally, is the gap between simulation and reality. Again, there are two cases. If our model assumptions do not apply (e.g., if the needle would be straight or the target could move), there is no real threat since the algorithm is not applicable at all. If our models are suitable but their parameters are poorly set, however, our algorithm might compute poor motion plans. To mitigate this kind of threat, we complement software simulations with hardware experiments using real needles and sensors. Yet, we can only partially close the gap to reality: the tissue, though in hardware, always is simulated only, i.e., it is no real human tissue.

Threats to internal validity, next, are posed by the choice of configuration parameters, in particular the limit of tolerance of needle movements (along with the allowance of threshold) and the orientation of the needle at the initial location. If those parameters are chosen poorly, the needle might, e.g., just jump around and the algorithm might not behave as the user had expected. So that our

algorithm is not unjustly held accountable, we address the risk of misinterpretations by raising awareness that parameters values need to be chosen carefully (see Table 1).

External validity of needle-steering planners, third, is very restricted from the outset to a (small) selection of applications and relative to the underlying models of the needle, the needle dynamics, the sensor, and the tissue. In the literature, an approach often is demonstrated on just one or two needle-steering applications, along with very fine-grained modeling assumptions. Accordingly, the experiments we presented generalize to other PA and PNT scenarios (with different number or size of CRs, different values for force, speed, and deviation parameters), provided the modeling assumptions are preserved; in line with other needle-steering publications, we make no claims otherwise. Of course, since the algorithm itself is application-agnostic, the two case studies may serve as blueprint for other needle-steering applications, or controllers in other domains.

Threats to conclusion validity, lastly, could be bias in the reporting of simulation data, especially since the variance between two simulation runs can be large. We mitigate this threat by not aggregating the numbers but listing them per simulation run (Table 3, Table 4). Finally, the overall goal with the two case studies was to show the feasibility of our approach: that needle steering can be based on model checking—and that conclusion can be drawn from two running scenarios.

7. Summary and future work

In this paper, we presented online strategy synthesis, OnSS, a workflow to realize controller synthesis for real-timed games of partial knowledge, where safety-critical information only becomes available at run time. We demonstrated the concepts of OnSS for the needle-steering problem, a well-known class of applications in safety-critical medical robotics. For needle steering, both the needle behavior and the patient's motion render offline motion plans unsafe since at run time safety-critical tissue might have changed its place. We provide an algorithm for online controller synthesis, OCS, and prove that the strategies generated by OCS reach target tissue in a safe way, whenever such strategy exists; Algorithm OCS, however, might not always terminate. Simulations of two applications of needle steering show that the OCS behavior depends on the degree of initial knowledge of the environment as well as the application- and scenario-specific layout of critical regions, but that its performance remains practical in all cases.

Applying our workflow to other (autonomous) applications is possible but requires varying degrees of adaptations or transfer. For other needle-steering applications, the adaptations are at the implementation level of the Uppaal models, since, e.g., trajectories might have to be computed differently or the tissue layout might need to be adjusted. For navigation problems with more motion primitives than PUSH, PULL, and ROTATE, one needs to extend the Uppaal models of the *Decision Maker/User* and *Controlled Device/Needle* with the accordingly labeled edges, provide C++ implementation of those motions, and adapt the FITCIRCLE routine in the OCS algorithm. For navigation problems where critical regions are detected not (only) by force sensors, the corresponding parameter and the CHECKSAFETY function in the OCS algorithm need to be adapted; and for navigation problems with goals different from reaching one target, one needs to revisit the Uppaal model *State Checker* and the CHECKTARGET function along with the termination condition of the algorithm. The fundamental requirement on all applications is that they must allow constructing detection regions around critical regions. For most mobile (autonomous) systems, however, this requirement is met since the transition from safe to unsafe states typically is not discrete.

Future work on OnSS for needle steering could extend the timed game to a game with 2 or 1 1/2 players, in which tissue is permitted uncontrollable or probabilistic movements, or to model false positives resp. false negatives in the detection of CRs. Further, one may consider the characteristics of individual motion plans, e.g., the number of required rotations, to pick the most suitable one. Currently, the existence of a motion plan suffices and the concretely executed motion plan can be chosen with uniform distribution; as soon as the problem is extended to an optimization problem, we need to rank the different offline plans so that the online game starts with the most promising one. Another new algorithmic direction would be, not to dismiss outdated strategies entirely but to try to repair them, and to identify conditions under which such repair pays off. On the modeling level, one could lift simplifying assumptions we currently make, in particular that the needle behaves the same during push and pull motions, and that previous motion paths in tissue have no influence on the needle trajectory when crossing them repeatedly after pull backs. One could also fine-tune the parameter selection regarding allowed deviation between needle and observations, force value changes, numbers of rotations, and assumed critical region sizes; and, generally, include additional assumptions on physically possible motion paths to improve the performance of strategy synthesis. For the same reason, one could experiment with different motion models for the needle. At system level, OnSS needs to be compared against other approaches for needle-steering planning, in particular POMDP-based ones, and evaluated for overhead, performance, and scalability. OnSS should also be implemented beyond the field of needle steering; many controllers for mechanical or electrical systems run in environments with effects that are hard, expensive, or impossible to formalize, and could benefit from formal guarantees and updated strategies during run time.

CRediT authorship contribution statement

Sascha Lehmann: Writing – original draft, Validation, Software, Methodology. **Antje Rogalla:** Writing – original draft, Visualization, Investigation. **Maximilian Neidhardt:** Writing – review & editing, Visualization, Investigation. **Alexander Schlaefer:** Writing – review & editing, Supervision, Funding acquisition. **Sibylle Schupp:** Writing – review & editing, Supervision, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

We thank Anton Reinecke for conducting hardware experiments and his help fixing software bugs. The anonymous reviewers provided numerous helpful hints that greatly improved the readability and accessibility of the article.

References

[1] Y. Liu, Y. Peng, B. Wang, S. Yao, Z. Liu, Review on cyber-physical systems, IEEE/CAA J. Autom. Sin. 4 (1) (2017) 27–40, https://doi.org/10.1109/JAS.2017.7510349.
[2] A. David, P. Jensen, K. Larsen, M. Mikučionis, J. Taankvist, Uppaal stratego, in: C. Baier, C. Tinelli (Eds.), Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015), in: LNCS, vol. 9035, Springer, Berlin, Heidelberg, 2015, pp. 206–211.
[3] S. Lehmann, A. Rogalla, M. Neidhardt, A. Schlaefer, S. Schupp, Online strategy synthesis for safe and optimized control of steerable needles, in: Third Workshop on Formal Methods for Autonomous Systems, (FMAS 2021)s, in: Electronic Proceedings in Theoretical Computer Science, vol. 348, Open Publishing Association, 2021, pp. 128–135.
[4] G. Hoffmann, H. Wong-Toi, The input-output control of real-time discrete event systems, in: Proceedings Real-Time Systems Symposium, 1992, pp. 256–265.
[5] O. Maler, A. Pnueli, J. Sifakis, On the synthesis of discrete controllers for timed systems, in: E.W. Mayr, C. Puech (Eds.), STACS 95: 12th Annual Symposium on Theoretical Aspects of Computer Science, in: LNCS, vol. 900, Springer, Berlin, Heidelberg, 1995, pp. 229–242.
[6] E. Asarin, O. Maler, A. Pnueli, Symbolic controller synthesis for discrete and timed systems, in: P. Antsaklis, W. Kohn, A. Nerode, S. Sastry (Eds.), Hybrid Systems II, in: LNCS, vol. 999, Springer, Berlin, Heidelberg, 1995, pp. 1–20.
[7] E. Asarin, O. Maler, A. Pnueli, J. Sifakis, Controller synthesis for timed automata, in: 5th IFAC Conference on System Structure and Control 1998 (SSC'98), Nantes, France, 8-10 July, IFAC Proc. Vol. 31 (18) (1998) 447–452, https://doi.org/10.1016/S1474-6670(17)42032-5.
[8] E. Asarin, O. Maler, As soon as possible: time optimal control for timed automata, in: F.W. Vaandrager, J.H. van Schuppen (Eds.), Hybrid Systems: Computation and Control, in: LNCS, vol. 1569, Springer, Berlin, Heidelberg, 1999, pp. 19–30.
[9] T. Brihaye, T.A. Henzinger, V.S. Prabhu, J.-F. Raskin, Minimum-time reachability in timed games, in: L. Arge, C. Cachin, T. Jurdziński, A. Tarlecki (Eds.), Automata, Languages and Programming, in: LNCS, vol. 4596, Springer, Berlin, Heidelberg, 2007, pp. 825–837.
[10] M. Jurdzinski, A. Trivedi, Reachability-time games on timed automata, in: L. Arge, C. Cachin, T. Jurdzinski, A. Tarlecki (Eds.), Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings, in: LNCS, vol. 4596, Springer, 2007, pp. 838–849.
[11] S. La Torre, S. Mukhopadhyay, A. Murano, Optimal-reachability and control for acyclic weighted timed automata, in: R. Baeza-Yates, U. Montanari, N. Santoro (Eds.), Foundations of Information Technology in the Era of Network and Mobile Computing, in: IFIP, vol. 96, Springer, 2002, pp. 485–497.
[12] R. Alur, M. Bernadsky, P. Madhusudan, Optimal reachability for weighted timed games, in: J. Díaz, J. Karhumäki, A. Lepistö, D. Sannella (Eds.), Automata, Languages and Programming, in: LNCS, vol. 3142, Springer, Berlin, Heidelberg, 2004, pp. 122–133.
[13] P. Bouyer, F. Cassez, E. Fleury, K.G. Larsen, Optimal strategies in priced timed game automata, in: K. Lodaya, M. Mahajan (Eds.), FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, in: LNCS, vol. 3328, Springer, 2004, pp. 148–160.
[14] T. Brihaye, V. Bruyère, J.-F. Raskin, On optimal timed strategies, in: P. Pettersson, W. Yi (Eds.), Formal Modeling and Analysis of Timed Systems (FORMATS 2005), in: LNCS, vol. 3829, Springer, Berlin, Heidelberg, 2005, pp. 49–64.
[15] P. Bouyer, T. Brihaye, N. Markey, Improved undecidability results on weighted timed automata, Inf. Process. Lett. 98 (5) (2006) 188–194, https://doi.org/10.1016/j.ipl.2006.01.012.
[16] P. Bouyer, K.G. Larsen, N. Markey, J.I. Rasmussen, Almost optimal strategies in one clock priced timed games, in: S. Arun-Kumar, N. Garg (Eds.), FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, in: LNCS, vol. 4337, Springer, Berlin, Heidelberg, 2006, pp. 345–356.
[17] P. Bouyer, S. Jaziri, N. Markey, On the value problem in weighted timed games, in: L. Aceto, D. de Frutos Escrig (Eds.), 26th International Conference on Concurrency Theory (CONCUR 2015), in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 42, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2015, pp. 311–324.
[18] P. Bouyer, V. Forejt, Reachability in stochastic timed games, in: S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. Nikoletseas, W. Thomas (Eds.), Automata, Languages and Programming, in: LNCS, vol. 5556, Springer, Berlin, Heidelberg, 2009, pp. 103–114.
[19] N. Bertrand, S. Schewe, Playing optimally on timed automata with random delays, in: Formal Modeling and Analysis of Timed Systems - 10th International Conference, in: LNCS, vol. 7595, Springer, Berlin, Heidelberg, 2012, pp. 43–58.
[20] M.Z. Kwiatkowska, G. Norman, A. Trivedi, Quantitative games on probabilistic timed automata, arXiv, https://doi.org/10.48550/arXiv.1001.1933, 2010.
[21] V. Forejt, M. Kwiatkowska, G. Norman, A. Trivedi, Expected reachability-time games, Theor. Comput. Sci. 631 (2016) 139–160, https://doi.org/10.1016/j.tcs.2016.04.021.
[22] M. Faella, S.L. Torre, A. Murano, Dense real-time games, in: Proceedings 17th Annual IEEE Symposium on Logic in Computer Science, IEEE, 2002, pp. 167–176.
[23] M. Faella, S. La Torre, A. Murano, Automata-theoretic decision of timed games, Theor. Comput. Sci. 515 (2014) 46–63, https://doi.org/10.1016/j.tcs.2013.08.021.
[24] G. Li, P.G. Jensen, K.G. Larsen, A. Legay, D.B. Poulsen, Practical controller synthesis for $\mathrm{MTL}_{0,\infty}$, in: H. Erdogmus, K. Havelund (Eds.), Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software (SPIN 2017), ACM, New York, NY, USA, 2017, pp. 102–111.
[25] G. Bacci, P. Bouyer, U. Fahrenberg, K.G. Larsen, N. Markey, P.-A. Reynier, Optimal and robust controller synthesis using energy timed automata with uncertainty, Form. Asp. Comput. 33 (2021) 3–25, https://doi.org/10.1007/s00165-020-00521-4.
[26] B. Finkbeiner, H.-J. Peter, Template-based controller synthesis for timed systems, in: C. Flanagan, B. König (Eds.), International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012), in: LNCS, vol. 7214, Springer, Berlin, Heidelberg, 2012, pp. 392–406.
[27] F. Cassez, A. David, K.G. Larsen, D. Lime, J.-F. Raskin, Timed control with observation based and stuttering invariant strategies, in: K.S. Namjoshi, T. Yoneda, T. Higashino, Y. Okamura (Eds.), Automated Technology for Verification and Analysis, in: LNCS, vol. 4762, Springer, Berlin, Heidelberg, 2007, pp. 192–206.
[28] J. Hoffmann, R.I. Brafman, Contingent planning via heuristic forward search with implicit belief states, in: Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling (ICAPS'05), AAAI Press, 2005, pp. 71–88.
[29] C. Muise, S.A. McIlraith, V. Belle, Non-deterministic planning with conditional effects, in: Proceedings of the Twenty-Fourth International Conferenc on International Conference on Automated Planning and Scheduling (ICAPS'14), AAAI Press, 2014, pp. 370–374.
[30] S. Yoon, A. Fern, R. Givan, FF-replan: a baseline for probabilistic planning, in: Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling (ICAPS'07), AAAI Press, 2007, pp. 352–359.
[31] G. Shani, R.I. Brafman, Replanning in domains with partial information and sensing actions, J. Artif. Intell. Res. 45 (2012) 565–600, https://doi.org/10.1613/jair.3711.

[32] M. Ghallab, D. Nau, P. Traverso, Automated Planning and Acting, 1st edition, Cambridge University Press, USA, 2016.
[33] A. Albore, H. Palacios, H. Geffner, A translation-based approach to contingent planning, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009, pp. 1623–1628.
[34] C. Daws, A. Olivero, S. Tripakis, S. Yovine, The tool Kronos, in: R. Alur, T.A. Henzinger, E.D. Sontag (Eds.), Hybrid Systems III, in: LNCS, vol. 1066, Springer, Berlin, Heidelberg, 1996, pp. 208–219.
[35] K. Altisen, S. Tripakis, Tools for controller synthesis of timed systems, in: Proc. 2nd Work. on Real-Time Tools (RT-TOOLS'02), 2002, pp. 2002–2025.
[36] S. Tripakis, S. Yovine, Analysis of timed systems based on time-abstracting bisimulations, in: R. Alur, T.A. Henzinger (Eds.), Computer Aided Verification (CAV 1996), in: LNCS, vol. 1102, Springer, Berlin, Heidelberg, 1996, pp. 232–243.
[37] S. Tripakis, K. Altisen, On-the-fly controller synthesis for discrete and dense-time systems, in: J.M. Wing, J. Woodcock, J. Davies (Eds.), FM'99 — Formal Methods, in: LNCS, vol. 1706, Springer, Berlin, Heidelberg, 1999, pp. 233–252.
[38] H.-J. Peter, R. Ehlers, R. Mattmüller, Synthia: verification and synthesis for timed automata, in: G. Gopalakrishnan, S. Qadeer (Eds.), Computer Aided Verification (CAV 2011), in: LNCS, vol. 6806, Springer, Berlin, Heidelberg, 2011, pp. 649–655.
[39] G. Behrmann, A. Cougnard, A. David, E. Fleury, K.G. Larsen, D. Lime, UPPAAL-tiga: timed games for everyone, in: Nordic Workshop on Programming Theory (NWPT'06), Reykjavik, Iceland, 2006, hal-00350470.
[40] F. Cassez, A. David, E. Fleury, K.G. Larsen, D. Lime, Efficient on-the-fly algorithms for the analysis of timed games, in: M. Abadi, L. de Alfaro (Eds.), 16th International Conference on Concurrency Theory (CONCUR 2005), in: LNCS, vol. 3653, Springer, Berlin, Heidelberg, 2005, pp. 66–80.
[41] D. Basile, M.H. ter Beek, L. Bussi, V. Ciancia, A toolchain for strategy synthesis with spatial properties, Int. J. Softw. Tools Technol. Transf. 25 (5) (2023) 641–658, https://doi.org/10.1007/s10009-023-00730-1.
[42] I. AlAttili, F. Houben, G. Igna, S. Michels, F. Zhu, F.W. Vaandrager, Adaptive scheduling of data paths using uppaal tiga, in: Proceedings First Workshop on Quantitative Formal Methods: Theory and Applications, in: Electronic Proceedings in Theoretical Computer Science, vol. 13, Open Publishing Association, 2009, pp. 1–12.
[43] G. Igna, V. Kannan, Y. Yang, T. Basten, M. Geilen, F. Vaandrager, M. Voorhoeve, S. de Smet, L. Somers, Formal modeling and scheduling of datapaths of digital document printers, in: F. Cassez, C. Jard (Eds.), Formal Modeling and Analysis of Timed Systems, in: LNCS, vol. 5215, Springer, Berlin, Heidelberg, 2008, pp. 170–187.
[44] K. Larsen, M. Mikučionis, J. Taankvist, Safe and optimal adaptive cruise control, in: R. Meyer, A. Platzer, H. Wehrheim (Eds.), Correct System Design: Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, in: LNCS, vol. 9360, Springer, Cham, 2015, pp. 260–277.
[45] A.B. Eriksen, C. Huang, J. Kildebogaard, H. Lahrmann, K.G. Larsen, M. Muniz, J.H. Taankvist, Uppaal stratego for intelligent traffic lights, in: Proceedings of the 12th ITS, European Congress, Strasbourg, France, 19-22 June, 2017.
[46] S.L. Karra, K.G. Larsen, F. Lorber, J. Srba, Safe and time-optimal control for railway games, in: S. Collart-Dutilleul, A. Romanovsky, T. Lecomte (Eds.), Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification: Third International Conference (RSSRail 2019), in: LNCS, vol. 11495, Springer, 2019, pp. 106–122.
[47] D. Basile, M.H. ter Beek, A. Legay, Strategy synthesis for autonomous driving in a moving block railway system with uppaal stratego, in: A. Gotsman, A. Sokolova (Eds.), Formal Techniques for Distributed Objects, Components, and Systems, in: LNCS, vol. 12136, Springer, Cham, 2020, pp. 3–21.
[48] T. Brihaye, S. Lasaulce, M. Jungers, N. Markey, G. Oreiby, Using model checking for analyzing distributed power control problems, EURASIP J. Wirel. Commun. Netw. (2010), https://doi.org/10.1155/2010/861472.
[49] E.R. Wognsen, B.R. Haverkort, M. Jongerden, R.R. Hansen, K.G. Larsen, A score function for optimizing the cycle-life of battery-powered embedded systems, in: S. Sankaranarayanan, E. Vicario (Eds.), Formal Modeling and Analysis of Timed Systems (FORMATS 2015), in: LNCS, vol. 9268, Springer, Cham, 2015, pp. 305–320.
[50] S. Dai, M. Hong, B. Guo, Synthesizing power management strategies for wireless sensor networks with UPPAAL-STRATEGO, Int. J. Distrib. Sens. Netw. 13 (4) (2017), https://doi.org/10.1177/1550147717700900.
[51] M. Jaeger, P.G. Jensen, K.G. Larsen, A. Legay, S. Sedwards, J.H. Taankvist, Teaching stratego to play ball: optimal synthesis for continuous space MDPs, in: Y. Chen, C. Cheng, J. Esparza (Eds.), Automated Technology for Verification and Analysis - 17th International Symposium, (ATVA 2019), in: LNCS, vol. 11781, Springer, Cham, 2019, pp. 81–97.
[52] K.G. Larsen, M. Mikučionis, M. Muñiz, J. Srba, J.H. Taankvist, Online and compositional learning of controllers with application to floor heating, in: M. Chechik, J.-F. Raskin (Eds.), Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2016), in: LNCS, vol. 9636, Springer, Berlin, Heidelberg, 2016, pp. 244–259.
[53] A. Rogalla, S. Lehmann, M. Neidhardt, J. Sprenger, M. Bengs, A. Schlaefer, S. Schupp, Synthesizing strategies for needle steering in gelatin phantoms, Electron. Proc. Theor. Comput. Sci. 316 (2020) 261–274, https://doi.org/10.4204/eptcs.316.10.
[54] R.J. Webster, J.S. Kim, N.J. Cowan, G.S. Chirikjian, A.M. Okamura, Nonholonomic modeling of needle steering, Int. J. Robot. Res. 25 (5–6) (2006) 509–525, https://doi.org/10.1177/0278364906065388.
[55] M.G. Jushiddi, J.J. Mulvihill, D. Chovan, A. Mani, C. Shanahan, C. Silien, S.A. Md Tofail, P. Tiernan, Simulation of biopsy bevel-tipped needle insertion into soft-gel, Comput. Biol. Med. 111 (2019) 103–337, https://doi.org/10.1016/j.compbiomed.2019.103337.
[56] P. Moreira, S. Misra, Biomechanics-based curvature estimation for ultrasound-guided flexible needle steering in biological tissues, Ann. Biomed. Eng. 43 (8) (2015) 1716–1726, https://doi.org/10.1007/s10439-014-1203-5.
[57] M. Narayan, M.A. Choti, A.M. Fey, Data-driven detection of needle buckling events in robotic needle steering, J. Med. Robot. Res. 04 (02) (2019) 1850005, https://doi.org/10.1142/S2424905X18500058.
[58] C. Burrows, F. Liu, F. Rodriguez y Baena, Smooth on-line path planning for needle steering with non-linear constraints, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 2653–2658.
[59] C. Rossa, T. Lehamnn, R. Sloboda, N. Usmani, M. Tavakoli, A data-driven soft sensor for needle deflection in heterogeneous tissue using just-in-time modelling, Med. Biol. Eng. Comput. 55 (8) (2017) 1401–1414, https://doi.org/10.1007/s11517-016-1599-1.
[60] T. Lehmann, R. Sloboda, N. Usmani, M. Tavakoli, Model-based needle steering in soft tissue via lateral needle actuation, IEEE Robot. Autom. Lett. 3 (4) (2018) 3930–3936, https://doi.org/10.1109/LRA.2018.2858001.
[61] W. Sun, R. Alterovitz, Motion planning under uncertainty for medical needle steering using optimization in belief space, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2014, pp. 1775–1781.
[62] R. Alur, D.L. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (1994) 183–235.
[63] W. Yi, P. Pettersson, M. Daniels, Automatic verication of real-time communicating systems by constraint-solving, in: Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques VII, Springer US, Boston, MA, 1995, pp. 243–258.
[64] K. Larsen, P. Pettersson, W. Yi, Compositional and symbolic model-checking of real-time systems, in: Proceedings 16th IEEE Real-Time Systems Symposium, IEEE, 1995, pp. 76–87.
[65] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, W. Yi, UPPAAL — a tool suite for automatic verification of real-time systems, in: R. Alur, T.A. Henzinger, E.D. Sontag (Eds.), Hybrid Systems III, in: LNCS, vol. 1066, Springer, Berlin, Heidelberg, 1996, pp. 232–243.
[66] S. Lehmann, A. Rogalla, M. Neidhardt, A. Reinecke, A. Schlaefer, S. Schupp, Modeling R^3 needle steering in uppaal, Electron. Proc. Theor. Comput. Sci. 355 (2022) 40–59, https://doi.org/10.4204/eptcs.355.4.
[67] P. Bouyer, F. Cassez, E. Fleury, K.G. Larsen, Optimal strategies in priced timed game automata, in: FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, Springer, Berlin, Heidelberg, 2005, pp. 148–160.

[68] G. Behrmann, A. David, K.G. Larsen, A tutorial on uppaal, in: M. Bernardo, F. Corradini (Eds.), Formal Methods for the Design of Real-Time Systems–Time Systems (SFM-RT 2004), in: LNCS, vol. 3185, Springer, Berlin, Heidelberg, 2004, pp. 200–236.
[69] B. Yamauchi, Frontier-based exploration using multiple robots, in: Proceedings of the Second International Conference on Autonomous Agents (AGENTS '98), Association for Computing Machinery, New York, NY, USA, 1998, pp. 47–53.