Xintao Ma

Online Checking of a Hybrid Laser Tracheotomy Model in UPPAAL-SMC


supervised by:
Prof. Dr. Sibylle Schupp
Prof. Dr. Alexander Schlaefer
Jonas Rinast

Hamburg University of Technology (TUHH)
Technische Universität Hamburg-Harburg
Institute for Software Systems
21073 Hamburg
Declaration

I, Xintao Ma, declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources. Neither this thesis nor any other similar work has been previously submitted to any examination board.

Place, Date: Hamburg, 05. December, 2013

Signature: Xintao MA
Acknowledgement

First and foremost, I would like to thank Institute of Software Technology System at Hamburg University of Technology to give me the chance to do my master thesis, which will be a great help in my future study. Besides, I would like to show my deepest gratitude to my supervisors. One is Prof. Dr. Schupp, a respectable, affable, responsible, knowledgeable and resourceful professor, who provides me with valuable guidance in every stage of the writing of this thesis. The other supervisor is the PhD student in this institute, Jonas Rinast, who guides me a lot during my student project job stage. Without his enlightening instruction, wisdom, impressive kindness and patience, I could not have completed my thesis. His keen and vigorous observation enlightens me not only in this thesis but also in my future study.

Last but not Least, I would like to thank my parents who support me throughout my whole studies and give me encouragement.
Abstract

Traditional model checking encounters a lot of problems when dealing with medical device loops: the complexity of the human body results in a state space explosion. Additionally, no models are sufficient to represent the human body completely. Therefore, online model checking verifies the system properties in a short-time future only in order to reduce these problems. The original paper [2] implements a laser tracheotomy system as a hybrid system model and evaluates the feasibility of such an online model checking approach. The result shows that online model checking makes it possible to represent the human body and also reduce the state space. This paper aims to transform the hybrid models into Uppaal-SMC models. With the help of Java, the converted laser tracheotomy hybrid models were used to verify the patient safety through online model checking in Uppaal-SMC. Real world patient traces from the PhysioNet medical database are the base to update the Uppaal-SMC models dynamically to achieve online model checking. Although our experimental results are not as good as the results from the original paper, they still show that online model checking is a practical way to verify a medical device system.
Contents
1 Introduction ................................................................................. 1
  1.1 Motivation ........................................................................... 1
  1.2 Objectives .......................................................................... 2
  1.3 Contributions ...................................................................... 3
2 Background .................................................................................. 4
  2.1 Hybrid Automata .................................................................. 4
  2.2 UPPAAL-SMC ...................................................................... 6
    2.2.1 UPPAAL ...................................................................... 6
    2.2.2 UPPAAL-SMC ............................................................. 9
  2.3 Online Model Checking ........................................................ 13
3 Transforming Hybrid Automata Models to UPPAAL-SMC Models 16
4 Case Study .................................................................................... 20
  4.1 Overview ............................................................................. 20
  4.2 Online Hybrid Models .......................................................... 22
  4.3 Offline UPPAAL-SMC Models ............................................... 29
  4.4 Online UPPAAL-SMC Overview ........................................... 36
    4.4.1 Online UPPAAL-SMC Models ....................................... 36
    4.4.2 Java Implementation ................................................... 41
5 Model Parameters ......................................................................... 43
  5.1 Parameters Overview ............................................................ 43
  5.2 Estimation ............................................................................. 45
    5.2.1 Linear Regression .......................................................... 45
    5.2.2 Sine Interpolation ........................................................... 51
6 Experimental Results ...................................................................... 56
  6.1 Property Verification .............................................................. 56
    6.1.1 Offline Models Verification .......................................... 57
    6.1.2 Online Checking of Patient Safety ............................... 62
  6.2 Parameter Estimation Result .................................................. 62
    6.2.1 Execution Time Result .................................................. 64
    6.2.2 SpO2 Estimation Result ................................................ 65
    6.2.3 O2 Estimation Result ...................................................... 67
  6.3 Evaluation Summary ............................................................. 71
7 Conclusion and Further Development ............................................ 73
Bibliography .................................................................................... 75
List of Figures .................................................................................. 78
List of Tables ................................................................................... 80
Abbreviations ................................................................................... 81
1 Introduction

1.1 Motivation

Medical devices are designed for various uses, such as pulse oximeters, which are used to measure patient’s physiology parameters, and patient controlled analgesia (PCA) infusion pumps, which are used for pain management. Recently the cooperation of medical devices has become more efficient and improves the patient safety even further. For instance, the isolated use of a PCA infusion pump involves the risk that patients overdose on the drugs, threatening the patient safety. However, when interoperating with a pulse oximeter, the patient would be much safer. Once the patient is monitored by the pulse oximeter and an unsafe situation of the patient is detected, the PCA infusion pump could possibly stop injecting drugs into the patient body [1]. Thus, cooperating medical devices can improve the patient safety and it is necessary to verify the patient safe in such systems.

Model checking is one approach to verify the safety. Traditionally model checking is commonly carried out offline, which checks the system behavior in a long-time, unbounded future. After building the offline models, it needs to prove that the models never reach unsafe states, meaning that the models should be neither changed nor updated once built. In reality offline models including medical devices are significantly hard to construct because of the specialty of the human body. Two main reasons account for this difficulty:

- There is no sufficient offline model or equation to represent the human body. Not only because there is still a lot of unknown space to explore, but also because of the complexity of the human body. To compromise typically only some vital physiology signs are modeled, yet they are insufficient to represent the exact condition of the human body.

- Furthermore, due to the complexity of the human body, the verification space may also explode. The offline model may involve a lot of physiology parameters such as numerous genes. The number for those genes is already tremendous, not even mentioning their combinations. An explosion of the verification space is the result.

Thus, Tao Li, Qixin Wang, Feng Tan, Lei Bu, Jiannong Cao, Xue Liu, Yufei Wang, and Rong Zheng recently developed an online model checking method that checks the system behavior in a short-time, bounded future [2]. The main approach is to build the system $S$ and periodically update its changing parameters every $T$ seconds. So every $kT$ seconds, the system $S$ needs to be reconstructed based on the update parameters. Then, they verify the system properties in the time interval $(kT, (k + 1)T)$. Thus, the system properties can be verified for all times.

In contrast to offline model checking, online model checking significantly reduces the verification state space. As only a short-time future needs to be considered when verifying the system properties, a lot of physiology parameters can be neglected since the influence of their changes is tiny during this short period.

For offline model checking, another disadvantage is that the human body is hard to model in a long-time run. But the behavior in a short-time run is predictable and can be modeled. For instance, after injecting morphine into a human body, the blood oxygen
level will behave differently for different people. It is possible that for some people the level will first drop and then increase but for other people it might be the other way around. This variance of human physiology results in a variance of the morphine’s absorption rate. Therefore predicting the curve of the blood oxygen level within 10 minutes is difficult. But a prediction for a few seconds is quite straightforward, because the saturation level cannot drop from 100% to an unsafe level of 59% so quickly [2].

In the original paper [2] the authors also carry out a case study on a laser tracheotomy scenario, which is modeled as a hybrid system. They then evaluate the online model checking performance based on the system, which shows that online model checking is practical. Besides, they also incorporate real world physiology traces to carry out the evaluation.

This thesis presents the hybrid system of the laser tracheotomy clinical scenario implemented in UPPAAL-SMC, a statistical model checking extension of UPPAAL, a well-known timed automata model checker. Moreover, an evaluation of this online model checking approach is carried out and compared to the original paper. Although our results show that our online model checking method is not as effective as in the original paper, it indeed succeeds in verifying the laser tracheotomy scenario.

1.2 Objectives

The main focus of this thesis is to verify the patient safety in a laser tracheotomy scenario. We apply online model checking to check the patient safety and evaluate the performance of it in this thesis.

To accomplish this goal, the following tasks need to be performed:

- The original paper uses the tool PHAVer [3] to model the laser tracheotomy hybrid system and to evaluate online model checking. However, in this thesis we focus on the tool UPPAAL-SMC, which uses statistical methods to verify the system properties. So the first task is to transform both, the offline models, and the online models of the laser tracheotomy scenario, from hybrid models to UPPAAL-SMC models.

- After the offline and online models have been built in UPPAAL-SMC, it is necessary to verify the patient safety in both variants. This is the most important prerequisite for the medical device system.

- Since in UPPAAL-SMC, the model verification and the model simulation is separate, it is not possible to automatically continue simulations after verification without reinitializing the models. However, with the help of Java, the reinitialization can be done. So, the next step is to develop a Java tool connecting to UPPAAL-SMC, such that the tool can automatically run models of UPPAAL-SMC and verify the patient safety continuously.

- In the course of continuous verification using Java, an adaptation of the models should also be carried out using real world physiology signals for reconstructing the models. The traces are extracted from PhysioNet [4], a database that offers a large collection of recorded medical signals and parameters of different patients.
In this thesis some vital physiology data traces are fed into UPPAAL-SMC models using Java.

- Lastly, a comparison and analysis between our evaluation results and the results in the original paper is carried out.

1.3 Contributions

During the project, the transformation from hybrid system models to UPPAAL-SMC models has been accomplished, and then both the offline and online laser tracheotomy models have been constructed in UPPAAL-SMC. Besides, the properties concerning the patient safety have also been expressed in UPPAAL-SMC’s query language and have been verified. In the end, the Java tool has been implemented to automatically run the models and the evaluation of online model checking has been carried out and compared to the results of the original paper. The comparison shows that online model checking succeeds in verifying this clinical scenario, although not as performant as the original paper declares.

The outline of this thesis is as follows: Chapter 2 introduces the basic definition of hybrid automata, the model checker UPPAAL-SMC, and online model checking. Besides the fundamental knowledge of UPPAAL-SMC, details on the basic UPPAAL tool are also stated in this Chapter. In Chapter 3 the process how to transform the hybrid automata models to UPPAAL-SMC models is presented step by step with specific examples. Chapter 4 presents the case study of the laser tracheotomy. First, the original hybrid models stated in the original paper are presented. Then both, the offline models, and the online models in UPPAAL-SMC, are shown in detail. Chapter 5 focuses on the parameters that need to be adjusted in the models during each reconstruction. First, a short overview on those parameters and also PhysioNet is given. Then the estimation methods to predict the parameter values, namely linear regression and sine interpolation, are presented. In Chapter 6 the experimental results are shown. Section 6.1 shows the results of the system verification. Section 6.2 shows the evaluation results of online model checking based on the different estimation methods. Section 6.3 analyzes the performance of online model checking and then summarizes it. At last, Chapter 7 concludes the thesis and suggests further research.
2 Background

2.1 Hybrid Automata

A hybrid system is a dynamic system that has both, discrete and continuous behavior [6]. For example, a multi-tank system [11], which has several tanks can be used to store water. The behavior of filling each tank is continuous, on the other hand, the switching between different tanks is a discrete behavior. As hybrid systems are rapidly employed, a reliable analysis of them is needed in the field of control systems and computer science. Therefore, hybrid automata modeling has been proposed and implemented.

A hybrid automaton is a finite state automaton extended with continuous variables that evolve over time according to the dynamical laws [7]. A finite state automaton involves locations and labeled edges. The transitions between different locations accomplish the discrete behavior. The continuous behavior of a hybrid automaton is represented using differential equation of the continuous variables [10].

There are different modeling techniques for hybrid system analysis, such as the Alur-Henzinger hybrid automata [3], which are mostly used in algorithmic analysis of hybrid systems. Another useful modeling technique is hybrid input and output automata [12], which enables compositional modeling and analysis.

**Syntax of Hybrid Automaton** A hybrid automaton is a tuple $\mathcal{H} = (L, l_0, X, E, F, I)$ [6, 9], where

- $L$ stands for a finite set of locations,
- $l_0$ is the initial location,
- $X$ is a finite set of continuous variables,
- $E$ is the finite set of edges where an edge has the form $(l, g, a, \varphi, l')$, where $l, l' \in L$, $g$ is a predicate on $\mathbb{R}^X$, $a$ represents an action, and $\varphi$ is the binary relation on $\mathbb{R}^X$,
- $F$ is the delay function for each $l \in L$,
- $I$ is the invariant mapping that assigns invariant predicates $I(l)$ to locations,

with the assumptions that:

- A variable valuation is a mapping $v$ from the continuous variables $X$ to the reals $\mathbb{R}$, $v: x \rightarrow \mathbb{R}$. So $\mathbb{R}^X$ represents a set of valuations over $X$.
- The delay function $F$ represents how the valuations on $X$ evolve with time. Thus $F(d, v)$ is the new valuation on $X$ after delay time $d$.

**Semantics of Hybrid Automata** The tuple $\mathcal{H}$ defines a time labeled transition system whose states are pairs $(l, v) \in L \times \mathbb{R}^X$, where $v \equiv I(l)$. The system exhibits continuous transitions $(l, v) \xrightarrow{d} (l, v')$, where $d$ is the delay time and $v' = F(d, v)$, meaning that the valuations evolve according to time in a location $l$. It also includes discrete transitions $(l, v) \xrightarrow{a} (l', v')$ if an edge $e \in E$ exits, which has the form $(l, g, a, \varphi, l')$, and $v \equiv g$, $\varphi(v, v')$. Therefore, a path $(l, v) \Rightarrow (l', v')$ exits if there is a finite sequence of continuous and discrete transitions from $(l, v)$ to $(l', v')$ [6].
**Example** The most commonly used example is the room thermostat [10], which can monitor the room temperature and control it depending on the monitored temperature. The room thermostat has two modes: when the thermostat is off, the room temperature will decrease. Once it drops below 4, the thermostat will be turned on and the room temperature will increase. As soon as the temperature exceeds 6, the thermostat will be turned off because the room becomes too hot. The hybrid model for this system is shown on Figure 2.1.

![Hybrid Model of Room Thermostat System](image)

**Figure 2.1:** Hybrid Model of Room Thermostat System

According to the syntax of hybrid automata, the room thermostat system can be analyzed as follows:

- Two locations exist in the hybrid model, namely *ON* and *OFF*.
- The initial location can be either *ON* or *OFF*, depending on the current room temperature.
- \( x \) is the continuous variable, which represents the room temperature that can either go up or down continuously.
- There are two edges in the model: The lower edge \( a \) transits from the location *OFF* to the location *ON* and has the guard \( x \leq 4 \). The thermostat is turned on when taking this edge. The upper edge \( b \) transits from the location *ON* to *OFF* with the guard \( x \geq 6 \) and the thermostat is turned off.
- The delay function on the location *OFF* is \( \dot{x} = -1.5 \), which represents that \( x \) changes by \(-1.5\) per time unit. And for the location *ON*, according to the delay function \( \dot{x} = 1.5 \), the evolving rate for \( x \) is \( 1.5 \) per time unit.
- The invariant predicates for locations *OFF* and *ON* are implicit. On the location *OFF*, \( x \geq 4 \) because when \( x \) reaches 4, the upper transition will be enabled. For location *ON*, \( x \leq 6 \) since the upper transition will be taken once \( x = 6 \).

Suppose the initial value for \( x \) is 7 and the thermostat is off, then the following transition sequence may occur:

\[
(x = 7, OFF) \xrightarrow{7-1.5} (x = 5.5, OFF) \xrightarrow{5.5-1.5} (x = 4, OFF) \xrightarrow{a} (x = 4, ON)
\]

The first two transitions are continuous changes inside the location *OFF* and the third transition \( a \) is a discrete transition. Thus, the system exhibits both continuous and discrete behavior in Figure 2.1.
2.2 UPPAAL-SMC

UPPAAL-SMC is an extended version of UPPAAL, which is a tool for modeling, validation, and verification of real-time systems. It was developed by Uppsala University in Sweden and Aalborg University in Denmark [8]. The core idea of UPPAAL-SMC is to first derive UPPAAL models from real world scenarios, then conduct simulations and monitor them, and afterwards use statistical methods such as Monte Carlo simulation to determine probabilities of certain system properties [6].

The UPPAAL-SMC tool consists of three parts: a graphical user interface (GUI), a verification server, and a command line. The purpose of the GUI is to model, simulate and verify the system derived from the real world. During the process of simulation and verification, UPPAAL-SMC needs to be connected to the verification server, which handles the core technology to verify the system properties. Besides, the command line tool is a stand-alone verifier [13].

The main window of the GUI is made up of an editor, a simulator and a verifier. The editor is responsible for constructing the system, which provides not only an easy graphical representation of finite state machines of the system, but also a C-like language that can be used to declare variables and functions. The simulator is designed to visualize the possible dynamic executions of a system. It shows the currently enabled transitions, current variable values, and the trace tree of each execution. The most useful function of the simulator is presenting a path that violates a system property, which helps the user to diagnose a failure of the system. However, in version 4.1.14 and below, the simulator does not support the execution of systems involving clock rates of neither 1 nor 0. The verifier is used to verify system properties using a specific query language.

Before any further introduction to UPPAAL-SMC, the basic tool UPPAAL is quickly introduced in Section 2.2.1. This introduction is divided into two parts: the syntax and the semantics of UPPAAL, and the query language. Then Section 2.2.2 presents UPPAAL-SMC, especially the modeling techniques different from UPPAAL and the SMC query methods.

2.2.1 UPPAAL

UPPAAL is based on the theory of timed automata, which is a finite state machine extended with clocks, variables, guards, and invariants. It also provides a query language using a subset of Timed Computation Tree Logic (TCTL).

• The Modeling Language

There are several formal definitions of UPPAAL’s timed automata [14,15,16]. We present UPPAAL version 4.0 [14] as it is easy to understand and covers the most useful functionality of UPPAAL. The following notations are necessary:

• C denotes a set of clocks,
• $B(C)$ denotes the set of conjunctions of simple conditions on clocks,

• Conditions have the form of either $x \preceq c$ or $x - y \preceq c$, where $x, y \in C, c \in \mathbb{N}$, and $\preceq \in \{<, \leq, =, \geq, >\}$.

**Syntax of Timed Automata** A timed automaton is a tuple $A = (L, l_0, C, A, E, I)$, where

• $L$ stands for a set of locations,

• $l_0$ is the initial location and $l_0 \in L$,

• $C$ is a set of clock variable,

• $A$ is a set of actions, co-actions and the internal $\tau$-action,

• $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of edges annotated with actions, guards, and a set of reset clocks between locations,

• $I: L \rightarrow B(C)$ assigns invariants that have the form of conjunctions over clocks to locations.

**Semantics of Timed Automata** Given a timed automaton as described above, the semantics can be expressed by a labeled transition system $\langle S, s_0, \rightarrow \rangle$, where $S \subseteq L \times \mathbb{R}^C$, $s_0$ is the initial state, and $\rightarrow \subseteq S \times (\mathbb{R}_{\geq 0} \cup A) \times S$, which means the transition relation of both, the location transitions and the clock evolving transitions, under the restriction that a clock can increase by only 1 or 0 per time unit.

The UPPAAL system is modeled as a network of timed automata that consist of one or more such automata, which communicate over channels. Each automaton in UPPAAL consists of locations, edges, clocks, and other variables. The current locations together with the current clock values represent the current state of the system. Furthermore, the transitions between the states of a system coincide with the firing of edges from locations to others [13].

**Locations** In each automaton there is one location labeled *initial* that is marked with a double circle in UPPAAL. Furthermore, the system cannot delay if a current location is labeled with *urgent* (U) or *committed* (C). For committed locations, the execution of the next outgoing transition has priority and must be executed before normal operation continues. Furthermore, locations can be labeled with invariants. An invariant is made up by a conjunction of simple conditions on clocks, differences between clocks, and boolean expressions not involving clocks. Invariants represent the conditions specifying how long the system can stay in certain locations. For instance, assume a clock increases by 1 per time unit and an invariant states an upper bound for this clock. Then, as soon as the clock reaches the upper bound, the location must be left.

**Edges** Edges connect locations and are annotated with selections, guards, synchronizations, and updates. Selections allow easy specifications of multiple edges by using an iteration variable. Guards represent required conditions to enable the edge. A guard must be a conjunction of simple conditions on clocks, differences between clocks, and boolean expressions not involving clocks. Furthermore, updates assign new values to data variables or clock variables; even user-defined functions can be executed when the transition is taken. Synchronizations are represented using channels. One or more edges will be executed synchronously labeled with a common channel.

**Channels** A channel works like a synchronized signal to perform related transitions. It can be *regular*, *urgent*, or *broadcast*. A regular channel allows delays on the transition even if the communication is possible. However, an urgent channel does not allow delay
if the communication is possible. No time is allowed to pass. Both regular and urgent channels enable only two processes at the same time. Broadcast channels are used to enable one-to-many communications.

- **The Query Language**

UPPAAL can verify system properties expressed in its query language. It includes two types of formulae: state formulae and path formulae. Path formulae can be divided into three types: reachability formulae, liveness formulae, and safety formulae. Figure 2.2 shows an overview of path formulae [14]. The meaning of each path formula is shown in Table 2.1.

![Path Formulae in UPPAAL][17]

---

[17]: # FIGURE 2.2: Path Formulae in UPPAAL [17]
Table 2.1  Relation between Path Formula and Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Formula</th>
<th>Name</th>
<th>Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>$E &lt;&gt; p$</td>
<td>Possibly</td>
<td></td>
</tr>
<tr>
<td>Liveness</td>
<td>$A &lt;&gt; p$</td>
<td>Eventually</td>
<td>not $E[]$ not $p$</td>
</tr>
<tr>
<td></td>
<td>$p \rightarrow q$</td>
<td>Leads to</td>
<td>$A[] (p \text{ imply } A &lt;&gt; q)$</td>
</tr>
<tr>
<td>Safety</td>
<td>$A[] p$</td>
<td>Invariantly</td>
<td>not $E &lt;&gt; not p$</td>
</tr>
<tr>
<td></td>
<td>$E[] p$</td>
<td>Potentially Always</td>
<td></td>
</tr>
</tbody>
</table>

**State Formula** A state formula is an expression that specifies a property of a certain state. For example, $time < 3$ describes that the variable $time$ should be smaller than 3. Besides, a state formula can also incorporate a certain location $l$ in a process $P$ by using expression $P.l$.

**Reachability Formula** A reachability formula verifies whether there is a path to a state where the property $p$ holds. A reachability formula has the form $E <> p$.

**Liveness Formula** A liveness property requires that the property $p$ eventually holds. There are two representations in UPPAAL: $A <> p$ means on all paths $p$ will hold; $p \rightarrow q$ expresses that for all paths, whenever $p$ holds, $q$ should also hold in the future.

**Safety Formula** Safety formulae are usually used to describe that a bad state cannot be reached. $A[] p$ and $E[] p$ express that either the model is always in good states or a safe path in the model exists.

### 2.2.2 UPPAAL-SMC

UPPAAL-SMC inherits all basic functionalities and techniques of UPPAAL. Statistical Model Checking (SMC) techniques are added to randomize the simulation process and to verify system properties statistically. SMC can be seen as a trade-off between hypothesis testing and formal verification. Instead of only giving a binary yes-or-no answer to the system property verification question, SMC gives a confidence probability of the truth value to the inquiry. A more detailed introduction of SMC can be found in [19]. SMC was applied in the domains of networking [20], energy aware system [21]. The following introduction to UPPAAL-SMC focuses on the differences to UPPAAL regarding the modeling language and the query language.

- **The Modeling Language**

  One of the biggest differences between UPPAAL and UPPAAL-SMC is that clock variables may advance at different speeds. Invariants may therefore set clock derivatives not only to 1 or 0 but also to different values. The rates can be integers, fractions, and can result from equations involving other variables. Thus, UPPAAL-SMC can model more complex models.
Semantics of UPPAAL-SMC. The semantics of UPPAAL-SMC extends the action transition in UPPAAL with probabilistic transitions [13]:

a) The delay time in the current location is decided by either an invariant or by an exponential distribution with a rate $\lambda$:
   - If the invariant has an upper bound, then the delay time is chosen based on a uniform distribution.
   - If not, the delay is chosen by $-\ln(u)/\lambda$, where $\lambda$ is specified by location and $u$ is a uniform random number in the interval (0,1).

b) The process with the minimum delay will be executed next.

c) The shortest delay will be executed. At the same time, the corresponding clock variables will be updated accordingly.

d) The process will take the following transition:
   - Execute all transitions that are possible.
   - The firing edge will be chosen uniformly if there are no specific weight labels on the edges.
   - If weights exist, then the probability that an edge (or branch) is fired is $w_i/W$, where $w_i$ is the weight of this edge and $W$ is the sum of all weighted edges.

Furthermore, locations and edges have also been extended compared to UPPAAL:

**Locations** The locations are not only labeled with invariants, but also with an exponential rate $\lambda$. The rate is used to decide with which probability the system leaves the location after time $t$. $\Pr(\text{leaving after } t) = 1 - e^{\lambda t}$, where $e = 2.718281828...$. The rate $\lambda$ can be expressed by a simple integer expression or two integer expressions separated by a colon like $r:q$, thus $\lambda = r/q$. The smaller the rate is the longer the delay will be.

**Edges** The edges in UPPAAL-SMC can also be annotated with a weight. Between locations, an edge can be split into several branches, which is denoted by dashed lines in UPPAAL-SMC. Each branch is labeled with weights $w$. The probability of a particular branch to be executed is $w_i/W$, where $W = \sum_j w_j$.

**The Query Language**

There are four kinds of statistical properties that can be checked in UPPAAL-SMC: quantitative, qualitative, probability comparison and probable value estimation. For qualitative, probability comparison, and value estimation, the algorithm sequential hypothesis testing [22] solves the question as stated in [18]. Quantitative properties can be checked using an estimation algorithm that resembles the Monte Carlo simulation [23, 18].

**Probability Estimation (Quantitative)** This query estimates the probability that a path formula is true under the circumstances that the bound has not been exceeded. It can be represented by $\Pr[\text{bound}] (\phi)$. There are three ways to represent $\text{bound}$: 1) Using the expression $<= \text{Integer}$ to implicitly bound by the global time. 2) Using the expression $x <= \text{Integer}$ to bound by the clock $x$. 3) Using the expression $#$
**Integer** to bound by the number of discrete steps. Here, $\phi$ is a quantified property denoted by either $<> p$ or $[] p$.

**Hypothesis Testing (Qualitative)** Hypothesis testing checks whether the probability specified in the property will be less or greater than a certain threshold. The formula can be written as $\Pr[\text{bound}] (\phi) (\text{'}>=\text{'} | \text{'}<=\text{'}) \text{prob}$, where $\text{prob} \in [0,1]$.

**Probability Comparison** Probability comparison compares two probabilities without computing them directly in UPPAAL-SMC. The following query is used: $\Pr[\text{bound}_1] (\phi_1) >= \Pr[\text{bound}_2] (\phi_2)$.

**Value Estimation** This query estimates the expected min or max values of an expression that evaluates to an integer or a clock by running a specified number of simulations. It is used by $E[\text{bound}; \text{CONST}] (\text{'}\min \text{'} | \text{'}\max \text{'} : \text{Exp})$, where $\text{CONST}$ defines the number of simulations and $\text{Exp}$ is the expression to be evaluated.

For the first three queries, UPPAAL-SMC gives results with a certain confidence, however, the last query does not support the confidence calculation yet. The results can be shown in different charts: 1) The probability density distribution chart is useful for comparisons of different distributions. 2) The probability distribution chart can be used to access a probability at a particular moment in a time interval. 3) The frequency histogram is used to calculate how many runs at a particular time satisfy the properties. 4) The cumulative probability distribution chart describes how many runs are needed to reach a given probability. 5) The confidence intervals chart are computed using the Clopper-Pearson method \[24\], which is used to express the confidence at a specific time.

UPPAAL-SMC also supports the *simulate* query, which visualizes how expressions change their values during the runs. The query was introduced in earlier versions of UPPAAL-SMC because the built-in simulator is not working when a clock rate is neither 0 nor 1. Thus, with the help of *simulate*, users can get better insight into each step that happened in the system.

**Example** Figure 2.3 illustrates modeling and property verification in UPPAAL-SMC. It shows how the clock can evolve with different rates in different locations and how the probability of reaching different location varies. There are 8 locations in the model *Foo*, and $x$ is a clock. In different locations, $x$ has different changing rates as declared by the differential equations such as $x' = 4$, which represents that $x$ is changing by 4 units per time unit when the system is in location $D22$. Additionally, every location has an exponential rate, which represents the chance of leaving the corresponding location.
A quantitative property verification result is shown in Figure 2.4. It estimates the probability that location D22 is reached within a time bound of 10 time units, written as $\Pr[<= 10] (<> Foo.D22)$. The verification result is that the probability is in the interval $[0.200678, 0.300678]$ with a confidence of 0.95.
In order to get a deeper look into how the clock $x$ changed during the run, a simulation has been done using simuate 1 [<= 5](Foo.x). The result is shown in Figure 2.5. From the changing rate, the conclusion can be drawn that the whole process is $A \rightarrow B \rightarrow C_1 \rightarrow D_{11}$.

### 2.3 Online Model Checking

As mentioned in Chapter 1, online model checking is a modern and efficient method to check system properties, like the system safety and the system liveness. Traditional model checking methods, namely offline model checking, checks the system properties in a long-time run. Given a system $S$, after building the abstract models of the system, the model checker usually explores the state space of the models to find a path where the system property does not hold in some state [25]. This procedure requires correct long-time estimates of the state space. Also, the state space explosion might occur. Furthermore, in the real world, systems may be hard to model considering the required long-time future needs to be accurate but the parameters in the system change non-deterministically. Even if the values of parameters can be determined, the equation involving them can be extremely complex.

Online model checking tries to remedy this drawback of traditional model checking. The basic idea of online model checking is to check the system properties in a short-time run and then update the system with the observed parameters. This procedure is done periodically until the system properties are not satisfied. Instead of checking the whole state space, it only checks the partial state space for the next $k$ steps as shown in Figure 2.6.
The left side represents a real trace in the real world, and it is difficult to represent using simple equations. The right side represents the procedure of online model checking. Suppose the update period is $T$. Thus each circle stands for a certain future state at time instance $kT$ and implicitly its parameters. Each arrow represents the trace in the time interval $(kT, (k+1)T]$. The dashed lines are the correct traces with correct prediction functions that lead to the correct states with correct parameter values, however, with insufficient information, the traces according to prediction functions are shown as solid lines with errors. Therefore, after each prediction, the values of the parameters should be adjusted according to the real trace. Instead of checking the system properties during the whole trace, the verification procedure has been divided into sub-checking problems. During each trace, the system properties are checked once. Therefore, it is possible to tell whether the system properties are satisfied or not.

In summary, online model checking has the following advantages:

- It reduces the state space because at first the model is build for online use and thus only few parameters need to be modeled and predicted. It is not necessary to build the whole model at a time. Instead, as time passes, the models are reconstructed and modified depending on the corrected parameters and corrected prediction traces as mentioned above. Secondly, the verification state space is further reduced since only a short-time future is taken into consideration. Thus, the state space has been reduced and only a partial state space needs to be explored \[2\].

- The real world system is normally too complex to model accurately, especially the parameters that are used to describe the system behavior. Some of the parameters may even be non-deterministic and non-predictable; Some of the pa-
Parameters may only be fully expressed by complex equations, which is a burden for both the state space and also the model checker. However, with online model checking, only a short-time future needs to be considered. Imagine a nonlinear curve that cannot be easily represented by a simple equation, however, by taking a short-time window, a linear equation may be enough to describe the curve behavior.
3 Transforming Hybrid Automata Models to UPPAAL-SMC Models

Tao Li, Qixin Wang, Feng Tan, Lei Bu, Jiannong Cao, Xue Liu, Yufei Wang, and Rong Zheng [2] have modeled a laser tracheotomy medical scenario with hybrid automata using the tool PHAVer. However, in this project this medical scenario is modeled with UPPAAL-SMC. UPPAAL-SMC depends on the statistical approaches to verify the properties of real-time systems and to estimate undecidable problems. It has an easy-use interface and visualizations for both the verification results and the procedure that the system processes. Therefore, it is necessary to implement hybrid automata using UPPAAL-SMC.

A hybrid automaton is a tuple that has the form $\mathcal{H} = (L, l_0, X, E, F, I)$, which can be implemented in UPPAAL-SMC as follows [6]:

- $L$ can be represented as locations in UPPAAL-SMC, because locations are also finite set of states and they can also be labeled with invariants.
- $l_0$ can be represented by using the initial location from which the system starts to process.
- The continuous variables $X$ can be represented by clock variables in UPPAAL-SMC, which evolve continuously with different rates.
- The edges $E$ can also be represented by the edges in UPPAAL-SMC because the edges $E$ with the form $(l, g, a, \varphi, l')$ may be matched to UPPAAL-SMC features: 1) The edges in UPPAAL-SMC also connect two locations. 2) The edges are annotated with guards, which may function as the predicate $g$. 3) In addition, they are also annotated with actions and update as $a$ and $\varphi$ require.
- In UPPAAL-SMC the delay function $F$ is implemented using invariants on clock variables with differential equations of the forms $x' = 1$ or $x' = e$, where $e$ represents the expression involved with data variables or other clock variables.
- The invariants $I$ can be expressed by the same invariants in UPPAAL-SMC, and both of them can be assigned to locations.

A summary on transforming hybrid automata models to UPPAAL-SMC models is shown in Table 3.1.

<table>
<thead>
<tr>
<th>Hybrid automata</th>
<th>UPPAAL-SMC models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locations $L$</td>
<td>Locations</td>
</tr>
<tr>
<td>$l_0$</td>
<td>Initial location with double circle</td>
</tr>
<tr>
<td>Continuous variables $X$</td>
<td>Clock variables</td>
</tr>
<tr>
<td>Edges $E$ with form $(l, g, a, \varphi, l')$</td>
<td>Edge with guards, synchronizations, updates, and selections</td>
</tr>
<tr>
<td>Delay functions $F$</td>
<td>Invariants on locations with differential equation</td>
</tr>
<tr>
<td>Invariants $I$</td>
<td>Invariants with clock variables</td>
</tr>
</tbody>
</table>
Furthermore, semantically a hybrid automaton has both discrete transitions and delay transitions as stated in Chapter 2. As long as UPPAAL-SMC can implement both transitions, the transformation from hybrid automata to UPPAAL-SMC can be done: 1) The derivative equations of clock variables in locations can represent the continuous behavior of hybrid automata. 2) The discrete transitions in hybrid automata can be expressed by the invariants in the locations and the guards on the edges exiting the locations. As soon as the bounds of the locations are reached and the guards on the edges are satisfied, the discrete jump from a location to another is accomplished.

The transformation steps are as follows:

1. Define a clock variable in UPPAAL-SMC using the keyword `clock`.
2. Add locations to the template corresponding to the locations in the hybrid automaton.
3. Add edges between locations corresponding to the transitions in the hybrid automaton.
4. Modify the invariants for the differential equations of the locations.
5. To express the continuous behavior of the system, add the differential equations like \( x' = 1 \) or \( x' = e \) to the invariants of the locations. The differential equation expresses the rate the clock changes at, which should be the same rate as stated in the hybrid automaton’s locations.
6. Depending on the conditions and predicate labels on the edges in the hybrid automaton, annotate the edges in UPPAAL-SMC with guards expressing the same requirements as in the hybrid automaton. To make sure that once the condition is satisfied the former locations are left immediately, a helper condition with the same clocks needs to be added to the invariant of the former locations because in UPPAAL-SMC, if a location has no time bound in its invariant, the delay of leaving the locations will be chosen randomly. Therefore, by specifying both the guards on edges and the invariants in locations, discrete transitions can be accomplished in UPPAAL-SMC.
7. Use other features such as synchronizations to complete the same system in UPPAAL-SMC.

**Example** In order to understand the procedure, the room thermostat example as illustrated in Chapter 2.1 is reiterated. The transformation of the room thermostat model from a hybrid automata model to a UPPAAL-SMC model is accomplished by the following steps above:

1. Define a temperature clock variable:
   
   \begin{align*}
   clock \ temp; 
   \end{align*}

2. Add the locations \textit{ON} and \textit{OFF}.
3. Add the edges from location \textit{ON} to \textit{OFF} and backwards.
4. Name both locations.
5. In location \textit{OFF} add the invariant \( \text{temp}' == -1.5 \).
   
   In location \textit{ON} add the invariant \( \text{temp}' == 1.5 \).
6. Add a guard on the edge from \textit{OFF} to \textit{ON}: \( \text{temp} <= 4 \).
Add a guard on the edge from ON to OFF: $\text{temp} \geq 6$.

Add another invariant in location OFF using a connecting &&: $\text{temp} \geq 4$.

Add another invariant in location ON using a connecting &&: $\text{temp} \geq 6$.

The result is shown in Figure 3.1.

Figure 3.1: Transformation Example: Room Thermostat in UPPAAL-SMC

To verify the functionality of the room thermostat mode, we use the simulate query to generate a simulation visualization chart with the formula: simulate 1 [$\leq 10$] {$\text{Temp, Template.ON}$}. The simulation result is shown in Figure 3.2.

Figure 3.2: Transformation Example: A Simulation Result of Temperature

The red line represents the curve of the temperature, which is continuous. Because the initial location is ON the temperature starts to increase with the rate 1.5. However, when the temperature reaches 6, the invariant specifying that the temperature should be below
6 is not satisfied anymore. As a result, the discrete transition takes place leading to a decrease in temperature. The blue curve represents whether the system is in the location \textit{ON}. A value of 1 means the system is in the location \textit{ON} and a value of 0 represents that the system is currently in the location \textit{OFF}. As soon as the temperature reaches 6, the thermostat changes to mode \textit{OFF}, thus a discrete transition has been performed by jumping from the location \textit{ON} to \textit{OFF}. The temperature then continuously drops until it reaches the bound of 4. Therefore, the hybrid automata model has been transformed into the model in UPPAAL-SMC.

Other features may also be added into the system. For instance, a synchronization channel can be added on both edges, which tells the user that the thermostat has been turned on or off. Additionally, the user may also reset the thermostat in case it is not precise, where the temperature can be set to any value manually.

As a conclusion, hybrid automata have continuous transitions as well as discrete jump transitions. It is possible to implement these behaviors in models in UPPAAL-SMC. Using differential equations of clock variables as the invariants of locations, the continuous transitions can be accomplished. Discrete jumps can be implemented using both, guards annotated on edges, and invariants in locations. According to the transformation procedures mentioned above, hybrid automata can be implemented in UPPAAL-SMC. Chapter 4 presents a detailed example considering a real world clinical scenario.
4 Case Study

In order to illustrate the transformation from hybrid automata models to UPPAAL-SMC models, and to demonstrate the online model checking efficiency, a case study is carried out. The case study is based on previous work by Tao Li, Qixin Wang, Feng Tan, Lei Bu, Jiannong Cao, Xue Liu, Yufei Wang, and Rong Zheng, which models a laser tracheotomy scenario with hybrid automata [2].

4.1 Overview

Tracheotomy is a medical surgery that creates an opening into the windpipe from the neck, and then implants a tube into the opening to help the patient breathe [27]. One of the most commonly used methods to create such an opening is to use a laser scalpel, which has been pre-programmed to control the size and the position of the opening. Using a laser scalpel is safer than a manual cut, since mistakes from humans are unpredictable and unavoidable [28]. Though, the laser tracheotomy procedure also has a risk: When the oxygen in the airway has a high concentration, the laser may burn the patient tissue. Consequently the patient safety is endangered. Additionally, during the surgery, the blood oxygen saturation level \( \text{SpO}_2 \) must not be too low. Otherwise, respiratory failure or death may result. There are three basic safety rules that should be obeyed during the procedure.

**Safety Rule One** When the laser scalpel is emitting, the windpipe oxygen level \( \text{O}_2 \) of the patient must not exceed a threshold \( \Theta_{\text{O}_2} \).

**Safety Rule Two** The blood oxygen saturation level \( \text{SpO}_2 \) must not decrease below a threshold \( \Theta_{\text{SpO}_2} \).

**Optional Safety Rule Three** Once emitting is approved by a supervisor, the laser scalpel should emit for at least \( T_{\text{approve}} \) time units unless the laser scalpel requests a stop by itself.

To increase the patient safety, the following medical devices presented in the laser tracheotomy scenario are analyzed for their safe cooperation:

- **Patient** The patient is characterized by the windpipe oxygen level \( \text{O}_2 \) and the blood oxygen level \( \text{SpO}_2 \).

- **Ventilator** The ventilator helps the patient breathe during the surgery.

- **Laser Scalpel** The laser scalpel is used to cut the windpipe of the patient.

- **Supervisor** The supervisor guarantees the patient safety by controlling the ventilator and laser scalpel according to the patient’s \( \text{O}_2 \) and \( \text{SpO}_2 \) values.

**Offline Model Checking of Laser Tracheotomy** The control loop of the offline laser tracheotomy system is shown in Figure 4.1. The patient updates his \( \text{O}_2 \) and \( \text{SpO}_2 \) levels based on the prediction equations according to the respiration rate controlled by the ventilator. The ventilator pumps out a certain amount of air to the patient body and then pumps in a certain amount of air from the patient body. However, before the laser scalpel gets the permission for emitting from the supervisor, the ventilator must be stopped.
by the supervisor. Otherwise the oxygen concentration may be too high. In addition, the ventilator may not be stopped for an extended time or the blood oxygen saturation level may drop too low. Furthermore, the supervisor may not allow the laser scalpel to emit unless O2 and SpO2 are at safe levels.

In offline model checking it is difficult to describe the long-time behavior of a patient’s SpO2 level because of the diversity of the human body. Different people have different changing blood oxygen saturation rates. There are no accurate offline models to predict the SpO2 level [2, 28, 29]. However, the short-time behavior of SpO2 is quite predictable because its value cannot jump from 99% to 59% within a short time. Its curve is smooth and can be represented with simple equations such as linear regression [2]. Therefore, online model checking is more effective in this scenario.

**Online Model Checking of Laser Tracheotomy** According to the introduction in Chapter 2, the online control loop is implemented as follows: 1) Both, the blood oxygen saturation and the windpipe oxygen, are updated periodically. The period $T$ is set to 3 seconds. 2) Suppose at time $t_0 = kT$, the SpO2 and O2 levels are updated and corrected to the real world values from the PhysioNet medical database. Their new estimation equations are calculated based on the history data and then updated. 3) Based on the up-to-date parameters, a new system is built and then in the time interval $(t_0, t_0 + T]$ the patient safety is verified. Figure 4.3 shows the procedure of online model checking of the laser tracheotomy system.
4.2 Online Hybrid Models

The hybrid models for the laser tracheotomy system have been implemented in previous work \[2\]. In order to have a better understanding of the transformation from the hybrid models to the UPPAAL-SMC models, it is essential to comprehend the online hybrid models of this clinical system first.

**Ventilator** During the surgery the patient is paralyzed and can breathe normally only with the help of a ventilator. The ventilator has three modes: pump out, pump in, and hold. It works like a reservoir: 1) The ventilator has a certain amount of air proportional to its height $0 \leq H_{\text{vent}} \leq 0.3 \text{ (m)}$. 2) When the ventilator pumps out the air to the patient, the height decreases with a rate of $-0.1 \text{ (m/s)}$. The patient inhales this air. 3) When the ventilator pumps in the air from the patient, the height increases with a rate of $0.1 \text{ (m/s)}$, and the patient exhales. 4) The supervisor normally allows the ventilator to work and then the ventilator switches between the pump-out mode and pump-in model. 5) As soon as the supervisor stops the ventilator by setting the variable $\text{VentOn}$ to false the ventilator first returns to its maximum height (Pump-in mode) and then switches to the hold mode. 6) During the hold mode the patient exhales because of its chest weight only, and no pumping occurs. 7) When a surgery step is finished the supervisor continues ventilation, represented by setting the variable $\text{VentOn}$ back to true. And then the ventilator continues looping from pumping out to pumping in.
The online hybrid automaton of the Ventilator is shown in Figure 4.3:

- Three locations exist in the hybrid model, namely PumpOut, PumpIn, and Hold.
- Initial locations can be any location depending on the previous state at time $t_0$. For example, when the system is stopped to update its parameters and the current location is Hold, then after the update, the initial location should also be Hold.
- $H_{vent}$ is a continuous variable, representing the current height of the ventilator. One thing to mention that the value of $H_{vent}$ after an update should be the same value as before the update.
- There are four edges that all have actions and guards. The action such as eventVentPumpIn is used to trigger the same event labeled in the patient model. Additionally the predicates in the brackets [] represent the conditions to trigger the edges.
- The differential equations involving $\dot{H}_{vent}$ represent the delay functions.
- In location PumpOut and Hold, both $H_{vent}$ and VentOn, are restricted by the invariants. However, in location PumpIn, only $H_{vent}$ is bounded because no matter whether the supervisor stops the ventilator or not, the location PumpIn can be entered.

**Patient** The patient breathes according to the mode of the ventilator. The model thus has three locations, Inhale, Exhale, and Hold, which correspond to the ventilator locations respectively: PumpOut, PumpIn, and Hold. The events on the edges are also triggered by the same events annotated in the ventilator model. The online hybrid automaton for the Patient is shown in Figure 4.4.
Suppose at time $t_0 = kT$ ($k \in \mathbb{N}$), when the system finished running for a period of $T = 3$ seconds, a more up-to-date windpipe oxygen value $\bar{O}_2(t_0)$ and a new blood oxygen level $\bar{SpO}_2(t_0)$ have been updated to the patient model. With the new values the system starts running from the same location where it was at time $t_0$ for the next time interval $(t_0, t_0 + T]$.

To express the short-time behaviors of the $O_2$ and $SpO_2$ values, differential equations are used to predict their values during the time interval $(t_0, t_0 + T]$.

- For the windpipe oxygen $O_2$, according to the original paper [2], there is a good model to describe the behavior of $O_2$ [30]. By setting the parameters in the differential equations, the value of $O_2$ can be predicted for a short-time frame. In this project, the parameters are extracted using linear regression based on the data from the real world $O_2$ trace in the database PhysioNet.

- For the blood oxygen saturation level $SpO_2$, the short-time behavior is also predictable. Its curve is smooth such that simple estimation methods (such as linear regression) can safely predict its values based on its past history. The estimation equation in Figure 4.4 is

$$SpO_2(t) = \bar{SpO}_2(t_0), \forall t \in (t_0, t_0 + T]$$

where $SpO_2(t)$ is the derivative of $SpO_2(t)$ at time $t$, and $\bar{SpO}_2(t_0)$ is the prediction values estimated by the linear regression based on the history values of a real world trace of $SpO_2(t)$ during $(t_0 - T_{past}, t_0)$, where $T_{past} = 3$ seconds.
In summary, the O\(_2\) and SpO\(_2\) values are predicted by estimation methods such as linear regression, whose equation parameters can be obtained from real world traces of the system.

**Laser Scalpel** The online hybrid automaton model of the Laser Scalpel is shown in Figure 4.5.

![Online Hybrid Automaton of the Laser Scalpel](image)

**Figure 4.5:** Online Hybrid Automaton of the Laser Scalpel

The main task for the laser scalpel is to request to cut the windpipe, and then to wait for the permission from the supervisor. After receiving the approval from the supervisor, the laser scalpel starts emitting. During the surgery, the emitting process can be stopped by the supervisor, by a timeout, or by itself when a cut is finished.

- The initial location can be any location as it needs to be the same location as at time \(t_0\). One important thing to mention is that in the location LaserEmitting, the initial value of \(t_{\text{emit}}\) should be the same value as at time \(t_{\text{emit}}\), meaning the value is the same as it was before the system updated. This ensures that the emitting time is not extended.
**Case Study**

- $t_{emit}$ is a continuous variable that records the laser’s emitting time. It prevents the laser scalpel from emitting too long (less than the maximum time $T_{emit}^{max}$).

- The edges are annotated with actions, guards, and updates. The actions such as $eventSupervisorStop$, $eventLaserFire$, and $eventLaserCancel$ are triggered by the supervisor. The actions $eventSurgeonCancel$, $eventSurgeonRequest$, and $eventSurgeonStop$ are triggered by the surgeon. Furthermore, the predicates in the brackets $[ ]$ must be true to enable the edges. The predicates without brackets stand for the updates that occur when the edges are fired.

- The continuous variable $t_{emit}$ increases by 1 per time unit.

- Except for the location $LaserEmitting$, which is also bounding $t_{emit}$ ($t_{emit}$ can not exceed the maximum value $T_{emit}^{max}$), all locations are restricting only the variables $LaserReq$ and $LaserApprove$ as shown in Table 4.1.

<table>
<thead>
<tr>
<th>LaserApprove</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>LaserReq</td>
<td>LaserEmitting</td>
<td>LaserCanceling</td>
</tr>
<tr>
<td>False</td>
<td>LaserRequesting</td>
<td>LaserIdle</td>
</tr>
</tbody>
</table>

The whole process of the laser scalpel approval can be summarized by Figure 4.6:

**Table 4.1 Laser Scalpel Locations by LaserApprove and LaserReq Invariants**

**Supervisor** The supervisor monitors and controls all medical devices, making decisions depending on the status of the patient. Upon receiving an emitting request from the laser scalpel, the supervisor first checks whether the patient is safe according to the safety rules mentioned above. After the approval, the supervisor is also responsible for checking the patient safety. The supervisor immediately stops the laser scalpel once the patient is in danger or the approval time has run out. The hybrid model of the Supervisor is shown in Figure 4.7. There are several points needed to understand the model:

- The supervisor mainly deals with two variables to control the medical devices: The variable $\text{VentOn}$ is used to control the mode of the ventilator. The value `true` represents that the ventilator is on and the value `false` represents the opposite. The variable $\text{LaserApprove}$ is used to determine whether the laser scalpel may emit the laser or not. Only when the value of $\text{LaserApprove}$ is `true`, the laser scalpel is allowed to operate.

- The judgment whether the patient is safe or not depends on two parameters: the windpipe oxygen $\text{O}_2$ and the blood oxygen $\text{SpO}_2$. During online model checking, the system can predict the values of $\text{O}_2(t)$ and $\text{SpO}_2(t)$ at any time $t$ in the time interval $[t_0, t_0 + T]$. We can directly use the predicted values $\text{O}_2(t)$ and $\text{SpO}_2(t)$ to make decisions on whether the supervisor should approve the laser scalpel’s request and stop the ventilator from working.

- As Figure 4.7 shows, there are two locations in the online hybrid automaton of the supervisor: $\text{LaserApproved}$ and $\text{LaserDisapproved}$. The transition from location $\text{LaserDisapproved}$ to location $\text{LaserApproved}$ is triggered by the event $\text{eventSupervisorApprove}$, which has three premises:
  
  a) The laser scalpel requests to operate indicated by the predicate $\text{LaserReq} = \text{true}$. 

---

**Figure 4.6:** The Approval Process of the Laser Scalpel
b) The windpipe oxygen $O_2$ is less than the threshold indicated by $O_2 < \theta_{O_2}$.

c) The blood oxygen saturation $SpO_2$ is greater than the threshold indicated by $SpO_2 > \theta_{SpO_2}$.

As soon as the transition is enabled, the variable $LaserApprove$ is set to $true$, upon which the laser scalpel starts to emit.

- There are two events that can trigger the transitions from $LaserApproved$ back to $LaserDisapproved$: $eventNormalDisapprove$ and $eventAbnormalDisapprove$.

  a) $eventNormalDisapprove$ This event is triggered either by a timeout or by a cancelation from the laser scalpel: The clock variable $t_{approve}$ is used to record how long ago the supervisor has approved the request from the laser scalpel, which
may not be too long ago or the patient safety is endangered. The approval time is bounded by \(0 \leq t_{approve} < t_{approve}^{\max}\). As the transition is triggered, the value of \(t_{approve}\) is set back to 0. The cancelation by the laser scalpel is indicated by \(LaserReq = \text{false}\).

b) \text{eventAbnormalDisapprove} When the patient safety is threatened, the supervisor must immediately stop the laser scalpel and turn on the ventilator. This event is triggered by the violation of either safety rule 1 or safety rule 2. Unlike \text{eventNormalDisapprove}, the variable \(t_{approve}\) is not set back to 0 because it is used to check the optional safety rule 3. In contrast to not checking the optional safety rule 3 under the event \text{eventNormalDisapprove} where the event is triggered by either \(t_{approve} = T_{approve}^{\max} > T_{approve}^{\min}\) or a request to stop itself by the laser scalpel where the safety rule 3 is trivially true, here a check is useful.

When either of the transitions is triggered, the value of \(laserApprove\) is set back to \text{false} and \(VentOn\) is set to \text{true}.

- The initial location again can be any location but it needs to be the same location as at time \(t_0\). One important thing to mention is that in the location \(LaserApproved\), the initial value of \(t_{approve}\) should be the same value as it was at time \(t_{approve}^{-}\).

### 4.3 Offline UPPAAL-SMC Models

Before transforming the online hybrid laser tracheotomy models to online UPPAAL-SMC models, it is easier to transform them to offline models first and then to add the online features.

To have a better understanding of the UPPAAL-SMC models, the events in the hybrid automata models are modeled by synchronization channels in UPPAAL-SMC. An overview of them is shown in Table \ref{table:4.2}.

\begin{itemize}
  \item Offline Ventilator According to Chapter \ref{chapter:3}, the transformation can be done step-by-step. The result of the offline UPPAAL-SMC model for Ventilator is shown in Figure \ref{figure:4.8}.
  \begin{itemize}
    \item The only continuous variable in the hybrid Ventilator model is the height of the air \(H_{vent}\). It is declared as a clock variable \(H_{vent}\) in UPPAAL-SMC.
  \end{itemize}
\end{itemize}

There are also three main locations in the model that corresponds to the same locations in the hybrid automaton model. An extra location \text{Initial} is added because the initial values of the clock variables cannot be set in the declaration in UPPAAL-SMC. To be more realistic the value of the air height in the ventilator can be set to any value at the start. Committed locations are necessary to annotate multiple synchronization channels on an edge because in UPPAAL-SMC each edge can only have one synchroniza-
### Table 4.2 Synchronization Channels

<table>
<thead>
<tr>
<th>Synchronization channels</th>
<th>Sender</th>
<th>Receiver</th>
<th>Annotated edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>VentPumpIn</td>
<td>Ventilator</td>
<td>Patient</td>
<td>PumpOut → PumpIn [Inhale → Exhale]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VentPumpOut</td>
<td>Ventilator</td>
<td>Patient</td>
<td>PumpIn → PumpOut [Exhale→Inhale]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PumpHold → PumpOut</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PatientHold → Exhale</td>
</tr>
<tr>
<td>VentPumpHold</td>
<td>Ventilator</td>
<td>Patient</td>
<td>PumpOut → PumpHold [Inhale→PatientHold]</td>
</tr>
<tr>
<td>SupervisorAppr</td>
<td>Supervisor</td>
<td>Ventilator</td>
<td>PumpOut → PumpIn [LaserDisapproved → LaserApproved]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LaserScalpel</td>
<td>LaserRequesting → LaserEmitting [LaserDisapproved \rightarrow LaserApproved]</td>
</tr>
<tr>
<td>SupervisorStop</td>
<td>Supervisor</td>
<td>Ventilator</td>
<td>PumpHold → PumpOut [LaserApproved \rightarrow LaserDisapproved]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LaserScalpel</td>
<td>LaserEmitting → LaserIdle [LaserDisapproved \rightarrow LaserApproved]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LaserScalpel</td>
<td>LaserCanceling [LaserApproved \rightarrow LaserDisapproved]</td>
</tr>
<tr>
<td>SurgeonCancel</td>
<td>LaserScalpel</td>
<td></td>
<td>LaserRequesting → LaserIdle</td>
</tr>
<tr>
<td>SurgeonReq</td>
<td>LaserScalpel</td>
<td></td>
<td>LaserIdle → LaserRequesting</td>
</tr>
<tr>
<td>SurgeonStop</td>
<td>LaserScalpel</td>
<td></td>
<td>LaserEmitting → LaserCanceling [LaserApproved \rightarrow LaserDisapproved]</td>
</tr>
<tr>
<td>Start</td>
<td>InitialStart</td>
<td>Ventilator</td>
<td>Initial → PumpOut</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Patient → Initial [Inhale]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LaserScalpel</td>
<td>LaserScalpel → Initial [Exhale]</td>
</tr>
</tbody>
</table>

In addition, however, the Ventilator communicates with both the supervisor and the patient according to Figure 4.1. On one side it synchronizes with the patient to help the patient breathe. On the other hand it synchronizes with the supervisor so that the supervisor can stop or start the ventilator.

- Only the clock $H_{vent}$ is bounded according to the invariants in the hybrid Ventilator model. However, the value of $VentOn$ is implicitly bounded in this model. In the location $PumpOut$, $H_{vent}$ must not fall below 0 and $VentOn$ should always be **true**; In the location $PumpIn$, only the variable $H_{vent}$ is bounded by an invariant: $PumpIn$ should be below the maximum height 300; In the location $PumpHold$, the variable $VentOn$ is implicitly restricted as the value should always be **false** because the ventilator must be turned off when the laser scalpel emits.

- The edges are annotated with synchronizations and guards in the same way as in the hybrid automaton. Note that the synchronization channel $Start$ synchronizes with a helper model InitialStart, which will be explained later. It is used to initialize $H_{vent}$ to any value.
In this project the value of $H_{vent}$ has been multiplied by 1000 because in earlier versions, UPPAAL-SMC did not support declarations of floating point variables. The multiplication increases the precision when rewriting the values of $H_{vent}$ into the model when the system is updated.

The hybrid automaton model has successfully been transformed into an UPPAAL-SMC model. The functionality of the ventilator has been recreated: 1) It pumps the air at the same rate as in the hybrid model, which is controlled by the continuous variable $H_{vent}$. 2) When the ventilator is turned on, the ventilator switches between pumping out and pumping in; when the ventilator is turned off, the ventilator first restores $H_{vent}$ to its maximum and then holds this position. 3) The model of the ventilator is controlled by the supervisor. In addition, the ventilator helps the patient breathe.

**Offline Patient** The offline UPPAAL-SMC model of the Patient is shown in Figure 4.9. It has the following features:

- The continuous variables for the windpipe oxygen level and the blood oxygen level are declared as clock variables $O2$ and $SpO2$ respectively.
- There are four locations in the Patient model: 1) The location Initial and the synchronization channel Start are used to initialize the clock variables and the parameters of the differential equations in each location. 2) The other three locations, Inhale, Exhale, and PatientHold, correspond to the Ventilator’s locations of PumpOut, PumpIn, and PumpHold. The discrete transitions between them are triggered by the same synchronizations as in the Ventilator model.
In this offline model the estimations of \( O_2 \) and \( \text{SpO}_2 \) values are done by first derivative equations modeled by the invariants of the locations:

a) For the windpipe oxygen \( O_2 \), the parameters \( O2PRM_b \), \( O2PRM_ai \), \( O2PRM_ae \), and \( O2PRM_ah \) are used to describe the inhale, exhale and hold rates of the patient. The values of these parameters are set by the model InitialStart.

b) For the blood oxygen saturation \( \text{SpO}_2 \), it is not possible to model its behavior. But the offline model is only used as the foundation for the online models. It is more important to check the functionality of the system rather than producing realistic values. Thus, we use a simple equation to model \( \text{SpO}_2 \). The parameters \( \text{SpO2PRM}_si \), \( \text{SpO2PRM}_se \), \( \text{SpO2PRM}_sh \) are used and set by the model InitialStart.

![Diagram of Offline UPPAAL-SMC Model of the Patient](image)

**Figure 4.9:** Offline UPPAAL-SMC Model of the Patient

Therefore, the patient is modeled by his breathing process with two physiology parameters \( O_2 \) and \( \text{SpO}_2 \). After initialization, in each location the values of \( O_2 \) and \( \text{SpO}_2 \) are predicted; the switching between the locations synchronizes with the location switching in the Ventilator model.

**Offline LaserScalpel** As shown in Figure 4.10, the model of the offline Laser Scalpel has the following features:

- Except for the location Initial, which assigns the initial values to the boolean variable LaserReq, the locations correspond to the same locations as in the hybrid LaserScalpel model.
• The only clock variable is $t_{emit}$, which is responsible for preventing the laser scalpel from emitting for too long. It is bounded in the location LaserEmitting such that it never exceeds the threshold $Temit_{max}$.

• The boolean variable $LaserReq$ is set to $\text{true}$ on the edge from the location LaserIdle to the location LaserRequesting. It is set back to $\text{false}$ on the edges that point back to the location LaserIdle.

Four traces are possible when running the model as shown in Figure 4.11: (a) The first figure shows that after the laser scalpel requests to operate, it directly cancels the request before the supervisor gives any response. (b) This trace represents that while the laser emits the laser scalpel is stopped by the supervisor. (c) This trace involving transition 4 and 5 expresses that the laser scalpel is stopped due to the timeout because the emitting operation has lasted for too long. (d) In this case the surgeon stops the laser scalpel. All the traces in the UPPAAL-SMC LaserScalpel model have the same behavior as the hybrid ones.

![Figure 4.10: Offline UPPAAL-SMC Model of the Laser Scalpel](image-url)
Figure 4.1: Traces in the Offline UPPAAL-SMC Model of the Laser Scalpel

**Offline Supervisor** The supervisor is responsible for ensuring the patient safety and controls other medical devices. It monitors the patient condition at all times and once it finds that the patient is in danger it stops the laser scalpel from emitting immediately. The offline UPPAAL-SMC model is shown in Figure 4.12.
There are two locations, namely LaserDisapproved and LaserApproved, which are the same locations as in the hybrid automaton model.

- An additional edge is added in UPPAAL-SMC from the location LaserApproved to LaserDisapproved because the guards in UPPAAL-SMC do not support disjunction. Thus, the events eventNormalDisapprove and eventAbnormalDisapprove have been separated into two edges.
- The only continuous variable is declared as the clock variable $t_{appr}$, which is used in the invariant in the location LaserApproved.
- The Supervisor controls the LaserScalpel and the Ventilator through the parameters LaserAppr and VentOn respectively, which are declared as boolean variables.

In summary, according to the values of the variables $O2$ and $SpO2$, the supervisor sets LaserAppr to true and VentOn to false when transitioning from the location LaserDisapproved to LaserApproved. Then, LaserAppr is set back to false and VentOn is set back to true on the way back.

**Offline InitialStart** The InitialStart model is an additional model that is used to initialize all the models as shown in Figure 4.13. The function SetParameter() is annotated on the edge as an update and thus is executed upon firing the edge. The following variables are initialized in the function: the clock variables $O2$, $SpO2$, and $H_{vent}$, the boolean variables LaserReq and LaserAppr, and the integer parameters in the estimation equations.
The hybrid automata models have all been transformed into models in UPPAAL-SMC. Accurately setting each variable is less important in the offline models, because they are the foundation for building the online models only. With the help of the offline models the functionality of the models can be checked in advance. The offline model checking results are shown and analyzed in Chapter 6.

4.4 Online UPPAAL-SMC Overview

Online model checking runs the system in a short-time window only. There are no good offline models for SpO2, however, the short-time behavior of SpO2 is predictable and can be expressed with simple equations. In addition, the predictions for the O2 level should be more realistic, and thus online model checking is a better option for this medical scenario. The core idea of online model checking is to first build the system, then run the system for a short time while ensuring the patient safety. Afterwards update the system with real world O2 and SpO2 values, as well as new parameters for the estimation equations that are used to describe the behaviors of O2 and SpO2. The estimation equation parameters are extracted from the history records in the real world traces and are discussed in the next chapter.

Normally UPPAAL-SMC only runs the system once and checks the system properties. Nevertheless, for online model checking running the system multiple times for several periods and repeatedly checking the safety conditions is necessary. Therefore, in order to automatically perform online model checking, a helper tool is needed to control the UPPAAL-SMC engine. In this thesis such a tool was developed in Java. In this section we first present the online models in UPPAAL-SMC and then shortly discuss the Java implementation of the helper tool.

4.4.1 Online UPPAAL-SMC Models

The biggest difference between the offline and the online models is that the initial location in the online models may be any location. As a result, the only thing that has to be changed from the offline models is adding edges from the location Initial to every other location. During the initialization, the following rules should be obeyed:
• The initial location should be the same as the location before updating the system.
• The initial values of the clock variables (except for \(O2\) and \(SpO2\)) should be identical to the values before updating the system. The clock variables are \(H_{vent}\), \(t_{emit}\), and \(t_{appr}\). The values for \(O2\) and \(SpO2\) are initialized with the real world values.

For example, consider a system has stopped at time \(t_0 = kT\) seconds and its parameters need to be updated. Before the update, the Supervisor model is in the location LaserApproved, and the clock variable \(t_{appr}\) has a value of 3. Then, after the update, when the system starts to run for another period, the initial location of the supervisor should still be LaserApproved and the clock variable \(t_{appr}\) should be equal to 3. This consistency is a key point to guarantee the patient safety. To make this clear, suppose the original location in LaserScalpel is LaserRequesting. However, after the update process, the initial location is LaserEmitting. Then the laser scalpel starts to emit regardless of the condition of the patient and also without the permission from the supervisor.

**Online Ventilator** As shown in Figure 4.14, there are three additional edges from the location Initial to the other three non-committed locations. Additionally, an integer variable is used to decide on the initial location, namely \(\text{Ventilator\_State}\): for the location PumpOut, \(\text{Ventilator\_State}\) equals 0; for the location PumpIn, \(\text{Ventilator\_State}\) equals 1; and for the location PumpHold, \(\text{Ventilator\_State}\) equals 2.

Thus, another task for the model InitialStart and the synchronization channel Start is to initialize the value of \(\text{Ventilator\_State}\), which should be the same as its previous value.
Figure 4.15: Online UPPAAL-SMC Model of the Patient with Linear Regression Estimation

Figure 4.16: Online UPPAAL-SMC Model of the Patient with Sine Interpolation Estimation
**Online Patient** The online Patient model shares the integer variable *Ventilator_State* with the Ventilator to decide on the initial location. There are two models of the online Patient in UPPAAL-SMC depending on the used estimation method. In Figure 4.15 the estimation method for the O₂ values is linear regression. The result of the estimation using linear regression for O₂ is not very good. Details are shown in Chapter 6. After analyzing the real world data, the curve of the windpipe oxygen is similar to a sine function. So another model shown in Figure 4.16 was developed where the estimation method is a sine interpolation.

The reason why the sine interpolation’s parameters in the estimation equations are divided by 100 is because first these parameters are multiplied by 100 when they are written into the initialization process in UPPAAL-SMC to increase the precision. Then the parameters are divided by 100 to be able to correctly predict the values of O₂. The scaling is necessary to obtain meaningful floating point results.

**Online LaserScalpel** Figure 4.17 shows the online UPPAAL-SMC model of LaserScalpel. An additional integer variable, namely *Laser_State* is needed to distinguish the initial locations in this model. The locations LaserIdle, LaserRequesting, LaserEmitting, and LaserCanceling correspond to the values 0, 1, 2, and 3 of the variable *Laser_State* respectively.

There is no assignment to *Laser_State* before entering a location because the locations in LaserScalpel can be distinguished from each other by the values of the boolean variables LaserReq and LaserAppr as shown in Table 4.1. Before the system runs, *Laser_State* has already been set according to these two boolean variables.

![Figure 4.17: Online UPPAAL-SMC Model of the Laser Scalpel](image-url)
Online Supervisor Similarly, an additional integer variable Supervisor State is used to decide on the initial locations of the Supervisor as shown in Figure 4.18. However, unlike other models, the decision on the initial location also depends on the clock variables SpO2 and O2. The reason can be elaborated by a counter example as follows: Suppose before an update at time $t_0$, the current location of the Supervisor model is LaserApproved and the value of O2 is 1700, which is smaller than the O2 threshold Th_O2. However, after the update, the O2 value is corrected to a value of 1800, which is greater than the threshold. The initial location should be the same as before, namely the location LaserApproved. According to the invariant of the location LaserApproved, the value of O2 should never exceed the threshold. Thus, such a state does not exist because the invariant in this location is not satisfied $[13]$. This fault is not the original model’s weakness. The original intend of the supervisor is, once the value of O2 is greater than the threshold, the supervisor immediately stops the laser scalpel and goes back to the location LaserDisapproved. But in UPPAAL-SMC, once this situation happens, it stops working because the state violating the invariant does not exist. Thus, the edges pointing back to the location LaserDisapproved are not triggered in such a case.

The solution is also taking the values of O2 and SpO2 into consideration when deciding on the initial location. If the newly updated O2 and SpO2 values violate the invariant stated in location LaserApproved, the initial location is changed to the location LaserDisapproved.

Furthermore, the value of the variable Supervisor State can be extracted from the boolean value of LaserAppr. If it equals to true, Supervisor State equals 1, otherwise it is equal to 0.
SetParameters(O2example, SpO2example, clockexample, boolexample, globaltime)

\[\text{Start!}\]

**Figure 4.19:** Offline UPPAAL-SMC Model of the InitialStart

**Online InitialStart.** Accomplishing the same functionality of the offline InitialStart model, the online InitialStart model as shown in Figure 4.19 is responsible to set the initial values and initial locations at the beginning of the system run. The function SetParameters() has several parameters, which are fed into the models:

- *O2example* The estimation equation parameters in the locations *Inhale*, *Exhale*, and *PatientHold*.
- *SpO2example* The estimation equation parameters in the locations *Inhale*, *Exhale*, and *PatientHold*.
- *clockexample* *H_vent*, *O2*, *SpO2*, *t_appr*, and *t_emit*.
- *boolexample* *Ventilator_State*, *LaserAppr*, *LaserReq*.
- *globaltime* The time how long the system has run.

**4.4.2 Java Implementation**

Because in UPPAAL-SMC the models can only be simulated once, in order to automatically run the models for a long time, a Java tool can be used as a bridge between each run of the UPPAAL-SMC models. The whole process of online model checking between UPPAAL-SMC and Java is shown in Figure 4.20.

After the system is built in UPPAAL-SMC, the system will first run for \(T = 3\) seconds and the patient safety is checked in this period. Then UPPAAL-SMC sends the current locations and current clock values to the tool at time \(t_0 = kT\). The current clock values include the air height of the ventilator, the emitting time of the laser scalpel, and the approval time of the supervisor. Upon receiving, the tool first reads the history \(O_2\) and \(SpO2\) values in the time interval \([t_0 - T_{past}, t_0]\) from the database. With the history data the tool calculates the estimation equation parameters using linear regression or sine interpolation. In addition, the real world \(O_2\) and \(SpO2\) values at the time \(t_0\) are also read from the database. Afterwards, the tool sends the locations and clock values that were received from UPPAAL-SMC back to the models. Then the models are updated with the real world \(O_2\) and \(SpO2\) values as the initial values. The newly calculated estimation parameters are also introduced in the initialization phase. Therefore, the newly updated models continue to run from the same locations and the clock values for another period of 3 seconds. The real world values correct the UPPAAL-SMC models such that they produce more accurate data traces.
The whole process runs for 600 seconds, meaning that online model checking iterates for 200 cycles. The verification result is discussed in Chapter 6 and it is compared to the results in the original paper [2].

In conclusion, with the help of Java, automatically carrying out online model checking and thus verifying the patient safety for a long time is possible in UPPAAL-SMC in this medical scenario.
5 Model Parameters

The offline and online UPPAAL-SMC models are built. However, the model parameters are still unspecified. This chapter first gives an overview on which parameters need to be considered and also introduces the database that is used to estimate the O₂ and SpO₂ values. Then the estimation methods for the O₂ and SpO₂ values are presented, namely linear regression and sine interpolation.

5.1 Parameters Overview

The parameters in the laser tracheotomy scenario are shown in the Table 5.1.

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameter</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ventilator</td>
<td>H_vent</td>
<td>The height of the ventilator</td>
<td>0 ≤ H_vent ≤ 300</td>
</tr>
<tr>
<td>Patient</td>
<td>O2PRM_inhale, O2PRM_exhale, O2PRM_hold</td>
<td>O₂ estimation equations parameters</td>
<td>Linear regression</td>
</tr>
<tr>
<td></td>
<td>a, b, d</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SpO2PRM_inhale, SpO2PRM_exhale, SpO2PRM_hold</td>
<td>SpO₂ estimation equations parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>O₂</td>
<td>O₂ level</td>
<td>Initial values are from database.</td>
</tr>
<tr>
<td></td>
<td>SpO₂</td>
<td>SpO₂ level</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Th_O₂</td>
<td>O₂ threshold</td>
<td>Unspecified</td>
</tr>
<tr>
<td></td>
<td>Th_SpO₂</td>
<td>SpO₂ threshold</td>
<td>Unspecified</td>
</tr>
<tr>
<td>LaserScalpel</td>
<td>t_emit</td>
<td>The time of the laser scalpel emitted</td>
<td>t_emit ≤ Temit_max</td>
</tr>
<tr>
<td></td>
<td>Temit_max</td>
<td>The maximum emitting time</td>
<td>Unspecified</td>
</tr>
<tr>
<td>Supervisor</td>
<td>t_appr</td>
<td>The elapsed time since the supervisor's approval</td>
<td>t_appr ≤ Tapppr_max</td>
</tr>
<tr>
<td></td>
<td>Tapppr_max</td>
<td>The maximum approval time</td>
<td>Unspecified</td>
</tr>
</tbody>
</table>

The parameters are medical thresholds, which fall into the experts’ responsibilities and are beyond this project’s range. No matter how the parameters are set, they do not com-
promise the main goal of this project. The patient safety can still be verified by online model checking regardless of the threshold values.

The real world traces for the O₂ and SpO₂ values used to calculate the parameters in the estimation equations are obtained from the medical database PhysioNet [4]. PhysioNet offers a large collection of recorded medical signals and parameters, established since 1999. It is funded by the National Institutes of Health’s NIBIB and NIGMS [32].

- **Windpipe O₂ real trace** Because there is no direct windpipe O₂ trace in the database airway CO₂ traces are used. The MGH/MF Waveform Database [31] is the only one that contains airway CO₂ traces. Other databases containing the keyword CO₂ refer to the arterial pressure, different from the airway CO₂ because the CO₂ values are affected by the lung chemical activities during absorption [33], which changes the values differently according to different conditions of patients.

<table>
<thead>
<tr>
<th></th>
<th>Inhale</th>
<th>Exhale</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>O₂</strong></td>
<td>20.96%</td>
<td>16%</td>
</tr>
<tr>
<td><strong>CO₂</strong></td>
<td>0.04%</td>
<td>5%</td>
</tr>
<tr>
<td><strong>N₂</strong></td>
<td>78%</td>
<td>78%</td>
</tr>
<tr>
<td><strong>Inert Gases</strong></td>
<td>1%</td>
<td>1%</td>
</tr>
</tbody>
</table>

Table 5.2 The Composition of Air [33]

However, the database uses the unit lpm (litre per minute) for the CO₂ traces, but the relation between CO₂ and O₂ is unknown. Therefore the first thing to do is to transform the unit lpm to a percentage. Then according to the composition of air inhaled and exhaled in Table 5.2 [33], which shows that only O₂ and CO₂ have changed during the breathing process, the relationship between O₂ and CO₂ in percentage can be calculated:

\[
O₂ = -2.289 \times CO₂ + 18.683
\]

Thus, the O₂ trace can be derived by the equation above.

Judging from the curves shown in Figure 5.1, when the patient inhales (shown as an increase in the respiration waveform [34]), the percentage of CO₂ decreases; When the patient exhales, the percentage increases, which is consistent with the change of the composition of the air during the breathing process.
Figure 5.1: Respiration Curve and CO₂ Curve from MGH/MF Waveform

- **SpO₂ real trace** The MGH/MF database does not contain traces of SpO₂. Thus, another database called MIMIC (Multiparameter Intelligent Monitoring in Intensive Care) Database is used [35]. However, this means that there is no dataset that has SpO₂ and O₂ records for the same person. But combining the data of two different patients can be used to solve the problem. This approach is consistent because such a patient with the combined records may exist, and it is also necessary to verify the safety of this kind of patient.

5.2 Estimation

Based on the real traces of O₂ and SpO₂ retrieved from PhysioNet, the estimation equations that are used to predict the O₂ and SpO₂ values during the system run can be calculated using simple methods.

For the estimation methods of O₂, not only linear regression but also sine interpolation is applied to the traces from PhysioNet. As it is shown in Chapter 6, linear regression does not give a good result when estimating O₂. For the estimation method of SpO₂, linear regression is suitable though. The following section first introduces linear regression and the actual SpO₂ and O₂ traces in Section 5.2.1. Then the sine interpolation is presented in Section 5.2.2.

5.2.1 Linear Regression

Linear regression is a statistical technique to model and interpret the relationship between variables [36]. A simple linear regression, which only has one variable to describe the data, is a straight line to interpret the behavior of the data. Linear regression is often used for describing the behavior of the data and predicting the data.

Simple linear regression can be expressed by

\[ y = \beta_0 + \beta_1 x + \epsilon \]

where \( \beta_0 \) is the intercept on the y axis and \( \beta_1 \) is the slope of the straight line. Additionally, \( \epsilon \) represents the random error, which is assumed to have a zero mean and a variance \( \sigma^2 \). \( \beta_0 \) and \( \beta_1 \) are called regression coefficients and can be estimated using least-square
estimation so that the sum of the squares of the differences between the data and the straight line is minimal. Suppose a set of data is given \((y_1, x_1), (y_2, x_2), \ldots, (y_n, x_n)\). Thus each data point can be represented using a sample regression model:

\[
y_i = \beta_0 + \beta_1 x_i + \epsilon_i, \quad i = 1, 2, \ldots, n
\]

Then, the sum of the squares of the differences between the real value of \(y_i\) and the estimated value \({\hat{y}}_i = \beta_0 + \beta_1 x_i\) can be expressed as

\[
S(\beta_0, \beta_1) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2
\]

According to the definition of least-square estimation that the difference must be a minimum, \(\hat{\beta}_0\) and \(\hat{\beta}_1\) must satisfy the following partial differential equations:

\[
\frac{\partial S}{\partial \hat{\beta}_0} = -2 \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0
\]

and

\[
\frac{\partial S}{\partial \hat{\beta}_1} = -2 \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)x_i = 0
\]

with the least-square estimators \(\hat{\beta}_0\) and \(\hat{\beta}_1\). Thus, after simplifying those two equations, we get

\[
n\hat{\beta}_0 + \hat{\beta}_1 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i
\]

\[
\hat{\beta}_0 \sum_{i=1}^{n} x_i + \hat{\beta}_1 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i x_i
\]

The solution of those two least-square normal equations is

\[
\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}
\]

\[
\hat{\beta}_1 = \frac{\sum_{i=1}^{n} y_i x_i - \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n}}{\sum_{i=1}^{n} x_i^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}}
\]

where \(\bar{y}\) and \(\bar{x}\) are average values with the formulae:

\[
\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \quad \text{and} \quad \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i
\]

We can further simplify the equation for \(\hat{\beta}_1\). The dominator in the equation represents the corrected sum of squares of \(x_i\), denoted \(S_{xx}\); and the numerator represents the corrected sum of the cross-products of \(x_i\) and \(y_i\), denoted \(S_{xy}\).

\[
S_{xx} = \sum_{i=1}^{n} x_i^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n} = \sum_{i=1}^{n} (x_i - \bar{x})^2
\]

and
\[ S_{xy} = \sum_{i=1}^{n} y_i x_i - \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n} = \sum_{i=1}^{n} y_i (x_i - \bar{x}) \]

Therefore, the formula that calculates \( \hat{\beta}_1 \) can be rewritten as

\[ \hat{\beta}_1 = \frac{S_{xy}}{S_{xx}} \]

After calculating the values of \( \hat{\beta}_0 \) and \( \hat{\beta}_1 \), the error of least-square estimation can be calculated for every value \( i \):

\[ e_i = y_i - \hat{y}_i = y_i - (\beta_0 + \beta_1 x_i), \quad i = 1, 2, ..., n \]

With the method mentioned above, the behavior of SpO\textsubscript{2} and O\textsubscript{2} can be approximated based on the data from PhysioNet.

**SpO\textsubscript{2} estimation** The behavior of SpO\textsubscript{2} in the near future can be represented using simple linear regression. For instance, one dataset retrieved from PhysioNet has the data points shown in Figure 5.2. The upper figure shows an increasing trend for the SpO\textsubscript{2} values and the lower figure shows a decreasing trend. With the data in Figure 5.2, the parameters for linear regression can be calculated and then can be fed into the UP-PAAL-SMC models.

As can been seen in Figure 5.2, \( \hat{\beta}_0 \) equals 0.1706, which cannot be written into UP-PAAL-SMC because it does not support floating point values. Thus, in order to increase the accuracy of the estimation equations, the dataset is multiplied by 100 and then rounded, resulting in the regression equations

\[ y = 17x + 8392 \]

and

\[ y = -7x + 9265 \]

Thus, the SpO\textsubscript{2} estimation equation parameters in the patient model can be assigned: *SpO2PRM_inhale* is assigned 17 and *SpO2PRM_exhale* is with -7 or the other way around because it is not clear whether during an inhale the SpO\textsubscript{2} values increase or decrease.
Figure 5.2: Sample SpO₂ Traces with Calculated Linear Regression using a 30 Seconds History
**O₂ estimation** The first attempt to describe the O₂ behavior is using linear regression. Figure 5.3 shows a trace of CO₂ retrieved from PhysioNet and its transformed O₂ data.

![CO₂ and O₂ Traces](image)

**Figure 5.3:** Real World Traces of O₂ and CO₂

After retrieving the data, the first thing to do is to classify the curve into inhale and exhale periods, because when $T_{past} = 30$ seconds, the curve contains several periods of the breathing process, including several inhale and exhale processes. Directly applying linear regression is not precise enough and results in huge errors.

Thus, the linear regression equations of O₂ should be applied to the inhale and the exhale periods separately as shown in Figure 5.4. When the patient inhales, the O₂ values increase and when the patient exhales, the O₂ values decrease. It can also be concluded that the estimation parameters are too small and cannot be written as initializations in UPPAAL-SMC directly. Again it is necessary to multiply the O₂ data by 100 for scaling. Thus, for an inhale process, a possible estimation equation could be as follows:

$$y = 230x + 1541$$

and an exhale estimation equation could be

$$y = -198x + 2099$$

Therefore, the patient model’s estimation equations can be defined by $O2PRM_{inhal} = 230$ and $O2PRM_{Exhale} = -198$. For the patient’s hold mode, the exhale rate should be smaller than in the exhale mode because the patient exhalles slowly only due to his chest weight. Thus, we assume that the hold mode has a rate that is defined by the exhale rate minus 20 for example. Then $O2PRM\_hold$ would equal -178.

However, the linear regression estimation is not precise enough to describe the behavior of the windpipe oxygen. A more detailed analysis is shown in Chapter 6.
In conclusion, linear regression may estimate change rates for O$_2$ and SpO$_2$ and the regression values are fed into the estimation equations in the Patient model in UPPAAL-SMC. Using the real world O$_2$ and SpO$_2$ values during the system run the O$_2$ and SpO$_2$ can be estimated and predicted.

![O$_2$ Inhale](image1)

\[ y = 2.3045x + 15.407 \]
\[ R^2 = 0.8651 \]

![O$_2$ Exhale](image2)

\[ y = -1.9771x + 20.99 \]
\[ R^2 = 0.785 \]

**Figure 5.4:** Sample O$_2$ Traces with Calculated Linear Regression for Inhale and Exhale
5.2.2 Sine Interpolation

Observation shows that real world traces of O₂ behave like a sine wave with a certain period. Thus, sine interpolation should be applied to describe the behavior of windpipe oxygen to improve the estimation accuracy.

Interpolation is the process of finding an appropriate equation to describe the trend of a set of data points. Sine interpolation is one instance of trigonometric interpolation, which interpolates the data points with a trigonometric polynomial. A trigonometric polynomial is a function consisting of a sum of sines and cosines with a given period, which focuses on solving periodic functions. A common solution for trigonometric interpolation is the discrete Fourier transformation.

However, in this project discrete Fourier transformation is not to be adopted to solve the sine interpolation problem. We assume that the function to describe the O₂ behavior only contains a sine function in the simplest form:

\[ y = a \sin(b \cdot x + d) + c \]

where,

- \( a \) represents the amplitude, which is expressed by the relative maximum or minimum value in the curve.
- \( b \) represents the angular frequency, which is the changing rate of the function in radians per second. The period of a sine wave can be derived from the angular frequency \( b \):
  \[ T = \frac{2\pi}{b} \]
- \( c \) represents the shift of the whole sine wave on the y-axis.
- \( d \) represents the phase, which specifies where the cycle oscillates when \( x \) equals 0. When \( b \) is not zero, the whole sine wave is shifted horizontally by \( b/d \).

A possible O₂ trace is shown in Figure 5.5, in which the values of O₂ have been multiplied by 100. In order to interpolate the data points, the following steps need to be done:
In order to calculate the period of the waveform, the first thing is to find the maximum values and their locations. There are several local maxima in the O₂ waveform. The period thus equals to the time difference between two local maximum values. Then the final value of b can be calculated by averaging all the time differences between two adjacent locations where the local maximum values occur.

\[ b = \frac{2\pi}{\sum_{i=2}^{n} (x_{i_{\text{max}}} - x_{i_{\text{max}} - 1}) / (n - 1)} \]

where \( n \) represents the number of local maximums \( l_{\text{max}} \) in the waveform to be analyzed. The averaging process eliminates the variances that occurred during the local maximum search.

After the local maxima have been found, it is easy to search for the local minimum in each period between two maxima. Thus, the amplitude \( a \) can be calculated as half the difference of a maximum and a minimum in the same period:

\[ a = \frac{1}{2n} \sum_{i=1}^{n} (x_{i_{\text{max}}} - x_{i_{\text{min}}}) \]

Because \( c \) represents the shift of the sine wave on the y-axis, it is also the shift of maximum value away from the zero point on the y-axis. Thus \( c \) can be represented as the difference between the amplitude and the maximum value.

\[ c = \frac{1}{n} \sum_{i=1}^{n} (x_{i_{\text{max}}} - \frac{1}{2}(x_{i_{\text{max}}} - x_{i_{\text{min}}})) \]
- \textbf{d} The easiest way to calculate \( d \) is to put the start value and the location where the start value occurs into the equation and solve the equation for \( d \). Thus, the value of \( d \) can be calculated by:

\[
d = \arcsin\left(\frac{y_{\text{start}} - c}{a}\right) - bx_{\text{start}}
\]

The \( O_2 \) trace can then be represented by a sine wave. The result of the interpolation of the real trace shown in Figure 5.5 is:

\[
y = 244.446 \times \sin(1.458x - 0.757) + 1851.495
\]

The calculated sine waveform is shown as the red line in Figure 5.6 and the relative errors between the real values of \( O_2 \) and the values calculated from the sine waveform are shown in Figure 5.7.

\[
\text{Relative error} = \frac{\text{Sine} - \text{Real}}{\text{Sine}} \times 100\%
\]

It can be concluded that:

- The \( O_2 \) trace and the calculated sine wave nearly coincide with each other. The numbers of their periods are the same.

- The large errors usually occur at the points where the trace plateaus, the points where the trace leaves the plateau, or the points where the sine wave rises from the local minimum points. This is because as soon as the inhale process of the patient starts, the \( O_2 \) values encounter a sharp increase.

- The relative errors vary between 0\% and 18\%. The maximum error is 17.72834\% and the minimum error is 0.000302\%. Sine interpolation has a better result than linear regression as is shown in Chapter 6.

- As can be seen in the equation even though the \( O_2 \) values have already been multiplied by 100 the parameters in the equation still have small fractional values. These values cannot be declared as floating point values in UPPAAL-SMC. If those values are rounded, the precision decreases a lot. Thus, as shown in Chapter 4, the solution is multiplying the fractional values by 100 before writing them into UPPAAL-SMC, and then dividing them by 100 again in the estimation equations in the invariants. At runtime the invariants can solve the equations involving the fractional values.
Figure 5.6: A Real World $O_2$ Trace and the Sine Interpolation Result

Figure 5.7: Relative Sine Interpolation Errors
In conclusion, the real world SpO₂ trace can be represented using linear regression based on the data from the PhysioNet using a data history of $T = 30$ seconds. The regression shows good results. However for O₂, the linear regression produces bad predictions and the estimation ability is limited when using real world traces. Thus, sine interpolation is applied to get better estimation equations for O₂. The parameters of both, the linear equations and the sine equations, are written back into the Patient model in UP-PAAL-SMC for the online model checking process.
6 Experimental Results

There are two main evaluations tasks in this project:

- The first task is to check the patient safety in the laser tracheotomy scenario. It is divided into offline model checking and online model checking. The offline models are used to verify the main functionality of each model described in Chapter 4. On the other hand, the online models are used to verify the patient safety using the real world medical parameter traces. The verification results are shown in Section 6.1.

- The second task is to analyze the feasibility of online model checking. The results are compared to the results in the original paper [2]. Our results are different from the ones in the original paper, but ours also show better state estimation than offline model checking. These results are shown in Section 6.2.

As a general result of this project, it is possible to ensure the patient safety using UPPAAL-SMC. In addition, online model checking is reliable and has a better accuracy than offline model checking when accurate long-term models are unavailable. A detailed evaluation comment can be found in Section 6.3.

6.1 Property Verification

To ensure the patient safety, there are three safety rules that may not be violated during a system run (see Chapter 4).

- The first safety rule is that the windpipe oxygen level may not exceed a threshold while the laser scalpel is emitting. The rule can be expressed in UPPAAL-SMC by:

\[ \Pr[<= 100] (<> O2 > Th_O2 && LaserScalpel.LaserEmitting) \]

The result should be tiny as such a situation should never arise, or the patient safety is in danger.

- The second safety rule is that the blood oxygen saturation level may not fall below a threshold when the laser scalpel is emitting. The rule can be expressed in UPPAAL-SMC by:

\[ \Pr[<= 100] (<> SpO2 < Th_SpO2 && LaserScalpel.LaserEmitting) \]

The result should also be tiny as such a situation should also never arise, or the patient safety is again in danger.

- The third safety rule says that the approval time of the supervisor must be longer than a threshold unless the surgeon cancels the emitting process. The rule is expressed in UPPAAL-SMC by:

\[ \Pr[<= 100] (<> (O2 > Th_O2 || SpO2 < Th_SpO2) && t_appr < T_{appr}^\text{min} && LaserAppr == \text{true}) \]

The expression describes the moment where the Supervisor must leave the location LaserApproved through the edge of the abnormal disapproval. The
approval time must be examined to be greater than the minimum threshold. The result should be nearly 0 as such a situation should also never arise, or the patient safety is again in danger.

The property verification can be divided into offline model checking and online model checking. The main functionality of each model is verified in offline model checking regardless of the parameters’ values, which are shown in Section 6.1.1. Then, the patient safety with real world O₂ and SpO₂ values is verified using online model checking in Section 6.1.2.

### 6.1.1 Offline Models Verification

This section verifies the basic functionality of each model in the laser tracheotomy scenario, ignoring the concrete parameters’ values. The result shows that the basic functions have been implemented in UPPAAL-SMC.

**Patient** The functionaility of the Patient is synchronizing with the Ventilator’s pace, updating the values of O₂ and SpO₂. Assume a simple scenario: when the ventilator pumps out, both the values of O₂ and SpO₂ increase; when the ventilator pumps in, they decrease as shown in Figure 6.1. The curves of O₂ and SpO₂ synchronize with the curve of \( H_{vent} \).

Besides, it is also shown that when the ventilator is turned off (the value of \( VentOn \) is 0) and stays in the location \( PumpHold \), the O₂ value decreases slowly and so does the SpO₂ value. When the SpO₂ value reaches the threshold, which is 90 in this case, the ventilator is turned on immediately at time 6.5.

![Figure 6.1: Verification of Patient's Functionality](image-url)

Figure 6.1: Verification of Patient's Functionality
Experimental Results

Figure 6.2:  Verification of Ventilator’s Functionality

**Ventilator** As shown in Figure 6.2(a), when the ventilator is turned on by the Supervisor, the air height increases when the Ventilator is in the location *PumpIn* and decreases when the Ventilator is in the location *PumpOut*.

As shown in Figure 6.2(b), as soon as the Ventilator is stopped by the Supervisor, namely *VentOn* equals *false*, the ventilator first increases the air height then stays at the highest as the ventilator switches to the location *PumpHold*.

When the Supervisor turns on the Ventilator again, the Ventilator starts transitions to the location *PumpOut* as the air height starts to decrease.
Figure 6.3: Verification of Laser Scalpel’s Functionality
**LaserScalpel** As shown in Figure 6.3 (a) and (b), when the values of LaserAppr and LaserReq equal **true**, the model LaserScalpel is in the location LaserEmitting; when the LaserReq changes to **false**, the model LaserScalpel changes its location to LaserCanceling. In a similar way, the relationship between the other two locations and LaserAppr and LaserReq can be verified, however this is not shown in this figure.

Another important functionality of LaserScalpel is when the laser is emitting, the emitting time may not exceed the maximum threshold of $t_{emit}$, which is 3 in this case and shown in Figure 6.3 (c). The value of $t_{emit}$ may only be reset once the location changes to LaserEmitting.

**Supervisor** When the Supervisor stays in the location LaserApproved, the value of LaserAppr should always be **true**, that is why in Figure 6.4 (a), their tracks coincide with each other. Moreover, the Supervisor may not approve the emitting surgery once the O2 and SpO2 values reach an unsafe state. When the value of SpO2 reaches 90 in the Figure 6.4 (a) at a time point around 7.9, the value of LaserAppr is set **false** immediately.

Besides, as shown in Figure 6.4 (b), when the Supervisor approves the laser scalpel to emit the light, the ventilator must immediately be turned off, as the value of VentOn must be **false**.

When the value of LaserApproved is 1 in Figure 6.4 (c), meaning that the Supervisor is in the location LaserApproved, the value of $t_{appr}$ may not reach a maximum threshold. The maximum threshold is set to 4 in this scenario. Thus, the laser scalpel cannot emit for too long and the patient safety is guaranteed.

Thus, the main functionality of every model is verified by using the simulate keyword in the offline models.
Figure 6.4: Verification of Supervisor's Functionality
6.1.2 Online Checking of Patient Safety

Since the patient safety is one of the biggest concerns nowadays in the development of medical device loops, it is necessary to check the patient safety in the laser tracheotomy scenario. Concerning the patient safety, results are more convincing using real world patient traces.

<table>
<thead>
<tr>
<th>Property</th>
<th>O₂ property</th>
<th>SpO₂ property</th>
<th>Approval time property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result with confidence 0.95</td>
<td>[0, 0.05]</td>
<td>[0, 0.05]</td>
<td>[0, 0.05]</td>
</tr>
<tr>
<td>Result with confidence 0.99</td>
<td>[0, 0.01]</td>
<td>[0, 0.01]</td>
<td>[0, 0.01]</td>
</tr>
</tbody>
</table>

The result is shown in Table 6.1. After 738 runs, all three system properties are checked in UPPAAL-SMC with a confidence of 0.95 and also with a confidence of 0.99. It is better when the confidence is greater, because in the medical area, an error with a probability of 0.001 still endangers the patient safety. However, since the laser tracheotomy scenario involves complicated equations and variables, the verification takes too much memory of the computer to reach such a confidence level. A confidence of 0.999 cannot be solved for now on our test hardware.

The O₂ property shows that O₂ cannot exceed the threshold. The SpO₂ property shows that SpO₂ cannot fall below the threshold. Furthermore, the approval time property shows that the approval time of the supervisor should be greater than a minimum value unless the surgeon cancels the emit request by itself.

From the result it can be concluded that the patient safety can be ensured in this system. The laser scalpel may not emit unless the supervisor discovers that the windpipe oxygen levels and the blood oxygen saturation levels are at the safe levels.

6.2 Parameter Estimation Result

Online model checking needs to update the system with new parameters for the estimation equations in the Patient model, and also the real world O₂ and SpO₂ values. With the help of Java we implemented a tool, which reads the real world patient O₂ and SpO₂ traces and then uses estimation methods to derive the parameters to predict O₂ and SpO₂.
Experimental Results

during the system run, such that the whole system can carry out online model checking automatically for a long time.

In order to evaluate online model checking, we execute UPPAAL-SMC together with the Java program for 600 seconds. Every 3 seconds the system queries for new O₂, SpO₂ values, and also the estimation parameters calculated from the real world patient traces. Then a new system based on the data is built and runs for another 3 seconds. This cycle continues for the whole duration. The evaluation is divided into three parts:

- The execution time of the update may not be too long, or the online model checking results have no meaning as their validity time already ran out. The update part consists of the following tasks:
  
a) The Java tool reads the real world patient traces.

b) The Java tool reads the current locations and the current clock values.

c) The Java tool calculates the estimation method parameters and then writes them back to UPPAAL-SMC.

d) The Java tool writes back the initial clock values and the initial locations into UPPAAL-SMC.

- After the newly updated estimation parameters are written back into UPPAAL-SMC, the system predicts the O₂ and SpO₂ values for a period of 3 seconds. The second evaluation is done by evaluating the errors between the prediction values and the real world values. Every \( t_0 = kT, k = 0, 1, 2, ... \) seconds, the prediction values will be compared to the real world values read from the database. Both O₂ and SpO₂ will be evaluated:

  a) For SpO₂ the only estimation method is linear regression. Suppose at time \( (t_0 + T) \), the prediction value of the blood oxygen value is \( \hat{SpO}_2(t_0 + T) \), and the real world value from PhysioNet is \( \hat{SpO}_2(t_0 + T) \). Then the relative error between the prediction value and the real world value can be represented by

  \[
  ERR_{SpO_2}(t_0 + T) = \frac{|\hat{SpO}_2(t_0 + T) - \hat{SpO}_2(t_0 + T)|}{\hat{SpO}_2(t_0 + T)}
  \]

  b) For O₂, at first linear regression is used to estimate the behavior of the windpipe oxygen. Then sine interpolation is applied. The evaluation is also accomplished by comparing the prediction values and real world values. Suppose the prediction value is \( \hat{O}_2(t_0 + T) \) and the real world value is represented by \( \hat{O}_2(t_0 + T) \). Then the relative error can be calculated by:

  \[
  ERR_{O_2}(t_0 + T) = \frac{|\hat{O}_2(t_0 + T) - \hat{O}_2(t_0 + T)|}{\hat{O}_2(t_0 + T)}
  \]

This section presents the evaluation results and compares our results to the results of the original paper [2]. Section 6.2.1 shows the execution time results and their comparison results. Section 6.2.2 presents the SpO₂ relative errors and matching comparison results.
Section 6.2.3 shows the O\textsubscript{2} relative error results but does not compare them because in the original paper there were no relative error results for O\textsubscript{2} given.

### 6.2.1 Execution Time Result

The computer platform for testing is a Macbook Pro with an 2.66 GHz Intel Core 2 Duo CPU and 4GB memory. The operating system is iOS 10.6.8.

The execution time measured in milliseconds in Java during one system run is shown in Figure 6.5. The maximum execution time reaches 300ms and the minimum time can be lower than 35ms. The comparison between the result of this project and the result of the original paper is shown in Table 6.2. Time is given in seconds.

<table>
<thead>
<tr>
<th>Execution Results Compared to the Original Paper (unit: seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td><strong>Result of this project</strong></td>
</tr>
<tr>
<td><strong>Result of the original paper</strong></td>
</tr>
</tbody>
</table>
As shown above, the execution time is much smaller than the result in the original paper, which uses PHAver, a hybrid model checker, on an Ubuntu system. The execution time is significantly smaller compared to the update period of the system, $T = 3$ seconds.

In conclusion, online model checking is practical to be carried out even with a much shorter update period.

### 6.2.2 SpO\textsubscript{2} Estimation Result

For SpO\textsubscript{2} the only applied estimation method is linear regression. The relative errors of the predicted SpO\textsubscript{2} values measured in our experiments during one system run are shown in Figure 6.6. The maximum error is approximately 6% and the minimum error is 0%.

![Relative Error of SpO\textsubscript{2}](image)

**Figure 6.6:** Relative Errors of the SpO\textsubscript{2} Measurements
The comparison to the results of the original paper is shown in Table 6.3. All values are in percentages.

Table 6.3  
<table>
<thead>
<tr>
<th>Relative errors (%)</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result of this project</td>
<td>0</td>
<td>6.01</td>
<td>1.59</td>
<td>1.15</td>
</tr>
<tr>
<td>Result of the original paper</td>
<td>0</td>
<td>3.31</td>
<td>0.58</td>
<td>0.51</td>
</tr>
</tbody>
</table>

Our results are different from the result in the original paper. This is because in the original paper, there is no specific patient trace mentioned. Our experiments used the patient trace a45463 in the MIMIC database, version 2. Different patient traces generally result in different estimation errors. Although our errors are greater than the original paper, they are reasonable. With these errors the SpO2 values still cannot drop from 99% to 59% within one simulation period, $T = 3$ seconds. The patient safety can still be ensured.

To testify the SpO2 prediction errors using linear regression, other patient traces have also been evaluated. The results are shown in Table 6.4, which presents 10 system runs, each running for 600 seconds.

Table 6.4  
<table>
<thead>
<tr>
<th>Relative errors (%)</th>
<th>a45436n</th>
<th>439n</th>
<th>n10301n</th>
<th>a45611n</th>
<th>477n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>1.62</td>
<td>0.63</td>
<td>2.99</td>
<td>0.59</td>
<td>4.18</td>
</tr>
<tr>
<td>Min</td>
<td>1.43</td>
<td>0.52</td>
<td>2.28</td>
<td>0.48</td>
<td>0.59</td>
</tr>
<tr>
<td>Mean</td>
<td>1.54</td>
<td>0.60</td>
<td>2.82</td>
<td>0.54</td>
<td>2.27</td>
</tr>
<tr>
<td>Std</td>
<td>5.58</td>
<td>1.64</td>
<td>20.36</td>
<td>3.47</td>
<td>119.21</td>
</tr>
</tbody>
</table>
It can be concluded that different patient traces result in different relative errors. If the trace of SpO2 is flat, then the errors are small; if the SpO2 trace exhibits jumps, the errors are relatively big. However, all of them show that online model checking results in small prediction errors. So linear regression is precise enough to predict a short-time future for SpO2 values.

The result shown above takes history data from $T_{\text{past}} = 30$ seconds for the linear regression into consideration. To test the efficiency of the history window size, experiments with $T_{\text{past}} = 4$ seconds were also carried out for comparison. The result is shown in Table 6.5.

### Table 6.5  SpO2 Estimation Errors with Different History Window Size

<table>
<thead>
<tr>
<th>Mean relative errors (%)</th>
<th>a45436</th>
<th>439n</th>
<th>n10301n</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{past}} = 30$</td>
<td>1.54</td>
<td>0.60</td>
<td>2.82</td>
</tr>
<tr>
<td>$T_{\text{past}} = 4$</td>
<td>2.01</td>
<td>0.41</td>
<td>1.01</td>
</tr>
</tbody>
</table>

As can be seen, with the history of 30 seconds, linear regression may not always have a better prediction performance. This is because sometimes the upcoming trends are not accommodated in data long time ago. However, a smaller window size may be thrown off by noise so it is also not reliable. A bigger window size provides a better noise resistance but has a trade-off in fast curve adaptation.

### 6.2.3 O2 Estimation Result

For O2 we first show the results with the linear regression estimation method. The original paper does not contain any results on the prediction errors for the windpipe oxygen and also does not mention which patient trace was used. The O2 prediction results are not satisfying in this project.

The relative error of the O2 prediction with the linear regression method during one system run is shown in Figure 6.7. The maximum error exceeds 60% and the minimum error is below 1%.
The result is not satisfying because with errors of more than 50% the prediction is unreliable. It means that O₂ values may drop half of its values, which may endanger the patient. The prediction is unrealistic and does not reflect the actual behavior of the O₂ values. Besides, the error variation is also too big and no pattern can be found.

One possible reason for these huge errors is that the short-time curve of windpipe oxygen cannot be expressed accurately using linear regression. There is no certain pattern to describe its behavior. Another important cause is according to the theory of the breathing process of the human, the time ratio between inhaling and exhaling is 1:2, however, the ratio in the model is 1:1, which does not correspond to the patient respiration curve. For example, in the model the patient is exhaling and the windpipe oxygen value is 1650, however the real world patient is inhaling and the windpipe oxygen value is 2000. Thus, this asynchrony between the real world and the model results in huge differences in values.

To remedy this result, sine interpolation is applied to get a better approximation. According to the description in Chapter 5, sine interpolation coincides with the general curve of the windpipe oxygen development. Thus, the errors should be smaller than the ones obtained with linear regression.

The relative errors using sine interpolation during one system run are shown in Figure 6.8. The maximum error exceeds 40% and the minimum error is below 1%. The errors variation is not as big as the variation using linear regression.
Experimental Results

Figure 6.8: Relative Errors of the $O_2$ Prediction Result using Sine Interpolation

The reason why there is still a huge prediction error is also the synchronization problem. Once the real world trace does not synchronize with the model the error could become maximal as a difference between the maximum value of the sine wave and its minimum value. However, the result has improved, especially the average error is much smaller.

The comparison of the $O_2$ prediction errors between the linear regression estimation and the sine interpolation estimation is shown in Table 6.6. All values are in percentage and the patient trace mgh077 from the MGF/MF database from PhysioNet was used. The average error has decreased by 5%. Thus sine interpolation has a better accuracy than linear regression.

Table 6.6  $O_2$ Estimation Errors Comparison between Linear regression and Sine Interpolation

<table>
<thead>
<tr>
<th>Relative errors (%)</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Regression</td>
<td>0.60</td>
<td>66.03</td>
<td>21.68</td>
<td>15.47</td>
</tr>
<tr>
<td>Sine Interpolation</td>
<td>0.12</td>
<td>40.93</td>
<td>16.32</td>
<td>11.18</td>
</tr>
</tbody>
</table>
Additional experiments using different patient traces have been carried out to further support our findings. The results of estimation errors using sine interpolation are shown in Table 6.7.

**Table 6.7**  
O2 Estimation Errors using Sine Interpolation for Five Patient Traces over 10 Runs

<table>
<thead>
<tr>
<th>Relative errors (%)</th>
<th>mgh077</th>
<th>mgh089</th>
<th>mgh057</th>
<th>mgh019</th>
<th>mgh110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>16.38</td>
<td>10.95</td>
<td>14.30</td>
<td>9.99</td>
<td>8</td>
</tr>
<tr>
<td>Min</td>
<td>16.3</td>
<td>10.93</td>
<td>14.27</td>
<td>9.94</td>
<td>7.96</td>
</tr>
<tr>
<td>Mean</td>
<td>16.34</td>
<td>10.94</td>
<td>14.28</td>
<td>9.97</td>
<td>7.98</td>
</tr>
<tr>
<td>Std</td>
<td>3.02</td>
<td>0.82</td>
<td>1.07</td>
<td>1.64</td>
<td>1.43</td>
</tr>
</tbody>
</table>

The estimation errors vary but all of them are promising. Patients with smaller prediction errors exhibit a breathing process that is closer to a sine wave. Besides, for comparison the same patient traces were analyzed using the estimation method linear regression. These results are presented in Table 6.8.

**Table 6.8**  
O2 Estimation Errors using Linear Regression for Five Patient Traces over 10 Runs

<table>
<thead>
<tr>
<th>Relative errors (%)</th>
<th>mgh077</th>
<th>mgh089</th>
<th>mgh057</th>
<th>mgh019</th>
<th>mgh110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>23.19</td>
<td>13.97</td>
<td>20.53</td>
<td>15.26</td>
<td>10.52</td>
</tr>
<tr>
<td>Min</td>
<td>17.98</td>
<td>12.26</td>
<td>16.79</td>
<td>11.32</td>
<td>8.29</td>
</tr>
<tr>
<td>Mean</td>
<td>20.80</td>
<td>13.41</td>
<td>18.85</td>
<td>12.31</td>
<td>9.2</td>
</tr>
<tr>
<td>Std</td>
<td>144.81</td>
<td>65.17</td>
<td>134.26</td>
<td>134.73</td>
<td>68.60</td>
</tr>
</tbody>
</table>
Comparing the results between the Table 6.7 and Table 6.8, it can be concluded that sine interpolation performs better than linear regression for every analyzed patient trace from the MGH/MF database. The error decreases by around 5%.

### 6.3 Evaluation Summary

In summary, comparing the result of this project and the original paper’s result, although the result is different, online model checking generally provides better results than offline model checking. There are several points our evaluation shows:

- As can be seen from the comparison results, there are differences for both the execution time and the linear regression estimation errors for SpO2. A possible reason is that they use two different patient traces from PhysioNet, because the original paper does not specify which patient trace was used for the SpO2 and O2 predictions. Despite this difference, online model checking is working well as the execution time is not big and the prediction errors of SpO2 are tolerable.

- The original paper also does not present their prediction errors for O2; however, this project evaluated the linear regression estimation method for O2 traces. The result is not satisfying and thus sine interpolation is used to predict O2. Although the errors are still relatively big, sine interpolation performs better than linear regression. A possible reason for the big errors of the O2 prediction is that maybe the real world patient traces do not synchronize well with the models in UPPAAL-SMC, i.e. the inhale-exhale processes in the model and in the real world are asynchronous.

- The patient safety is guaranteed as shown in Section 6.1.

Furthermore, in this project, online model checking is done with the cooperation of a Java tool and UPPAAL-SMC. The efficiency of using UPPAAL-SMC to implement hybrid automata is good. The advantages of using UPPAAL-SMC are shown as follows:

- UPPAAL-SMC has an easy handling user interface for the user to build the model.

- It also has an easy translation of system properties. With probabilities, it is more convincing to argue whether a system property is guaranteed or not. Especially with the charts provided by UPPAAL-SMC, it is easier to track the model progress during the system run.

- UPPAAL-SMC handles the hybrid models well. It can express the basic functionality of each model in the laser tracheotomy scenario. It also handles online model checking together with Java well. The interface with Java is usable and it is easy to get started.

However, UPPAAL-SMC has also some small problems dealing with hybrid automata and online model checking:

- One problem is in earlier versions UPPAAL-SMC does not support floating point variables, which loses a lot precision during predicting the values
for windpipe oxygen and blood oxygen levels. This problem is fixed in the version 4.1.16.

- Another problem of UPPAAL-SMC is the keyword *simulate*. Before the keyword *simulate* is used, a time bound must be declared first. However, when the user declares a time bound that the system should run for 3 time units, the simulation result of how long the system actually runs is unpredictable, sometimes the system runs for 6 time units, sometimes the system runs for only 1.4 time units. In addition, Java can only update the new parameters and real world O\textsubscript{2} and SpO\textsubscript{2} values to the new system when the simulation is over, meaning that sometimes the system has to predict a long-time future for O\textsubscript{2} and SpO\textsubscript{2} values, against the intention of online model checking. Predicting the behavior of SpO\textsubscript{2} for 6 seconds or more is quite ambitious. This is another reason why the prediction errors of both O\textsubscript{2} and SpO\textsubscript{2} can reach a high level.

- The third problem of UPPAAL-SMC implementing hybrid automata is described in Chapter 4, when introducing the online model of the Supervisor. The intended behavior of the hybrid supervisor is that, during the initialization of each system update, when the supervisor finds that the SpO\textsubscript{2} and O\textsubscript{2} values reach unsafe levels, the abnormal event should be triggered immediately. However, UPPAAL-SMC specifies that such a location with violated invariants may not exist, thus the initialization of the update fails. This can be fixed by judging the patient condition before deciding on the actual initial locations during the update process.
7 Conclusion and Further Development

Nowadays medical devices interoperate with each other to improve the patient safety by constructing a medical device loop. It is important to verify the patient safety in such loop systems. However, traditional offline model checking, which checks the system property for a long-time future, encounters a lot of problems. Especially the state explosion problem stands out remarkably in medical device loops, because the human body is complex and difficult to model.

Therefore, Tao Li, Qixin Wang, Feng Tan, Lei Bu, Jiannong Cao, Xue Liu, Yufei Wang, and Rong Zheng propose a new model checking method namely online model checking [2], which checks the patient safety in a short-time future and extends it with updates. It reduces the verification state space by ignoring unchanged parameters in the short-time future. Additionally, the behavior of the human body in a short-time future is predictable. It is even possible to use a simple method to estimate the physiology parameters to approximate the state of the human body.

In order to test the efficiency of online model checking, a case study is also carried out in the original paper. They implement a laser tracheotomy scenario modeled with hybrid automata. Then they determine the relative errors between the predicted parameter values and the real world values to show that online model checking is efficient and implementable. However, in this project, the case study is implemented in UPPAAL-SMC, a tool to model and verify the properties of real-time systems using statistical methods. To automatically carry out online model checking of the laser tracheotomy system over several update periods, a Java tool was developed and used together with UPPAAL-SMC. The results are not as accurate as the original paper states, but the general result is that online model checking is an efficient way to check medical device systems.

The main contribution of this project is at first transforming the hybrid models to UPPAAL-SMC models: differential equations involving clock variables can represent the continuous behavior of hybrid automata. The invariants together with the guards accomplish the discrete transitions of hybrid automata. Thus, according to the steps in Chapter 3, the hybrid models of the laser tracheotomy scenario can be transformed into UPPAAL-SMC models. Additionally, another contribution is the implementation of automatically carrying out online model checking using a Java tool together with UPPAAL-SMC.

In the laser tracheotomy scenario, the most difficult part is to represent the patient behavior by estimating and predicting the windpipe oxygen O2 and the blood oxygen saturation SpO2, which can only be accomplished for a short-time future because SpO2 is impossible to estimate in a long-time run. It is important to guarantee that the values of both O2 and SpO2 stay at safe levels, or the patient safety is endangered. Therefore, the laser tracheotomy system properties were verified with online model checking with the help of Java. After every period of the simulation, the Java tool stores the current locations and the current clock values from UPPAAL-SMC. Then it reads the O2 and SpO2 history data from the PhysioNet database. Afterwards it writes the new estimation parameters based on the history data and the real world values back into the UPPAAL-SMC models. In such a way UPPAAL-SMC can verify the patient safety for a long-time run.

The evaluation of online model checking in this project is done by calculating the errors between estimated parameter values and the real world values. The estimation method...
for predicting SpO$_2$ was linear regression, which shows a relative error of approximately 1%. Even though the result is different from the original paper, the errors still show that the prediction of SpO$_2$ using online model checking matches the real world trace. However, for O$_2$ the linear regression method is not accurate. In addition, sine interpolation performs better, because the real world trace of O$_2$ resembles a sine curve. The error of sine interpolation is around 10%, which is about 5% less than the linear regression method.

However, the online model checking scenario still has not reached standard method quality and improvements are still necessary to be done in the future. One possibility for improvements is instead of updating the parameters and real world values in the system every certain period, the update is initiated when the relative errors between the estimated parameter values and the real world values exceeds a threshold. This improvement will further reduce the state space by less frequent update procedures if necessary. Also, it increases the accuracy of estimation methods, as an upper bound for errors is enforced.

Another possible improvement is to verify the patient safety in the next period before the end of the current period, especially the calculation and the update progress should already be done in the period before. Currently the online model checking procedure is stopping the system simulation, calculating and updating the parameters, and then running the system again, which wastes a lot of time. By using a pipelining method the whole system can run for a long time continuously without pauses in between.

Furthermore, the exhale and the inhale process in the Patient model can be modified. In the real world scenario, the exhale time of the human is longer than the inhale time. Thus, an adapted Patient model may be more realistic and may synchronize with the real world patient trace better. As a result, the errors between the estimated O$_2$ values and the real world values may decrease.

In conclusion, UPPAAL-SMC is a powerful tool, which can model hybrid automata and then verify system properties. With the help of Java, the patient safety is verified in a laser tracheotomy surgery scenario using an online model checking approach, which offers the possibility to model the human body and reduces the verification space.
Bibliography


[10] J. P. Hespanha. Hybrid and Switched System. Lecture Notes from University of California at Santa Barbara, Fall, 2005


## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Hybrid Model for Room Thermostat System</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Path Formulae in UPPAAL</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>UPPAAL-SMC Example: Exponential Probability Model</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>UPPAAL-SMC Example: Quantitative Property Result</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>UPPAAL-SMC Example: A Simulation Run of Clock x</td>
<td>13</td>
</tr>
<tr>
<td>2.6</td>
<td>Online Model Checking State Space</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>Transformation Example: Room Thermostat in UPPAAL-SMC</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>Transformation Example: A Simulation Result of Temperature</td>
<td>18</td>
</tr>
<tr>
<td>4.1</td>
<td>Control Loop of Offline Laser Tracheotomy System</td>
<td>21</td>
</tr>
<tr>
<td>4.2</td>
<td>Online Model Checking of the Laser Tracheotomy System</td>
<td>22</td>
</tr>
<tr>
<td>4.3</td>
<td>Online Hybrid Automaton of the Ventilator</td>
<td>23</td>
</tr>
<tr>
<td>4.4</td>
<td>Online Hybrid Automaton of the Patient</td>
<td>24</td>
</tr>
<tr>
<td>4.5</td>
<td>Online Hybrid Automaton of the Laser Scalpel</td>
<td>25</td>
</tr>
<tr>
<td>4.6</td>
<td>The Approval Process of the Laser Scalpel</td>
<td>27</td>
</tr>
<tr>
<td>4.7</td>
<td>Online Hybrid Automaton of the Supervisor</td>
<td>28</td>
</tr>
<tr>
<td>4.8</td>
<td>Offline UPPAAL-SMC Model of the Ventilator</td>
<td>31</td>
</tr>
<tr>
<td>4.9</td>
<td>Offline UPPAAL-SMC Model of the Patient</td>
<td>32</td>
</tr>
<tr>
<td>4.10</td>
<td>Offline UPPAAL-SMC Model of the Laser Scalpel</td>
<td>33</td>
</tr>
<tr>
<td>4.11</td>
<td>Traces in the Offline UPPAAL-SMC Model of the Laser Scalpel</td>
<td>34</td>
</tr>
<tr>
<td>4.12</td>
<td>Offline UPPAAL-SMC Model of the Supervisor</td>
<td>35</td>
</tr>
<tr>
<td>4.13</td>
<td>Offline UPPAAL-SMC Model of the InitialStart</td>
<td>36</td>
</tr>
<tr>
<td>4.14</td>
<td>Online UPPAAL-SMC Model of the Ventilator</td>
<td>37</td>
</tr>
<tr>
<td>4.15</td>
<td>Online UPPAAL-SMC Model of the Patient with Linear Regression Estimation</td>
<td>38</td>
</tr>
<tr>
<td>4.16</td>
<td>Online UPPAAL-SMC Model of the Patient with Sine Interpolation Estimation</td>
<td>38</td>
</tr>
<tr>
<td>4.17</td>
<td>Online UPPAAL-SMC Model of the Laser Scalpel</td>
<td>39</td>
</tr>
<tr>
<td>4.18</td>
<td>Online UPPAAL-SMC Model of the Supervisor</td>
<td>40</td>
</tr>
<tr>
<td>4.19</td>
<td>Online UPPAAL-SMC Model of the InitialStart</td>
<td>41</td>
</tr>
</tbody>
</table>
Figure 4.20  Online Model Checking the Laser Tracheotomy Scenario in UPPAAL-SMC with Java ................................................................. 42

Figure 5.1  Respiration Curve and CO₂ Curve from MGH/MF Waveform .......... 45
Figure 5.2  Sample SpO₂ Traces with Calculated Linear Regression with a 30 Seconds History ........................................................................................................ 48
Figure 5.3  Real World Traces of O₂ and CO₂ .................................................. 49
Figure 5.4  Sample O₂ Traces with Calculated Linear Regression for Inhale and Exhale .............................................................................................................. 50
Figure 5.5  A Real World Trace of O₂ .................................................................. 52
Figure 5.6  A Real World O₂ Trace and the Sine Interpolation Result .................. 54
Figure 5.7  Relative Sine Interpolation Errors ...................................................... 54

Figure 6.1  Verification of Patient’s Functionality .............................................. 57
Figure 6.2  Verification of Ventilator’s Functionality ............................................ 58
Figure 6.3  Verification of Laser Scalpel’s Functionality ...................................... 59
Figure 6.4  Verification of Supervisor’s Functionality ......................................... 61
Figure 6.5  Execution Time Result ...................................................................... 64
Figure 6.6  Relative Errors of the SpO₂ Measurements ....................................... 65
Figure 6.7  Relative Errors of the O₂ Prediction Result using Linear Regression .... 68
Figure 6.8  Relative Errors of O₂ Prediction Result using Sine Interpolation .......... 69
List of Tables

Table 2.1 Relation between path formula and property ........................................ 9
Table 3.1 Summary of Transformation ................................................................... 16
Table 4.1 Laser Scalpel Locations by LaserApprove and LaserReq Invariants ....... 26
Table 4.2 Synchronization Channels ..................................................................... 30
Table 5.1 Parameters and Descriptions in UPPAAL-SMC Models ....................... 43
Table 5.2 The Composition of Air ......................................................................... 44
Table 6.1 Online Model Checking Results of the Patient Safety ......................... 62
Table 6.2 Execution Result Compared to the Original Paper (unit: seconds) ....... 64
Table 6.3 Estimation Errors of SpO₂ Compared to the Original Paper ................. 66
Table 6.4 SpO₂ Estimation Errors of Five Patient Traces over 10 Runs .............. 66
Table 6.5 SpO₂ Estimation Errors with Different History Window Size ............. 67
Table 6.6 O₂ Estimation Errors Comparison between Linear Regression and Sine Interpolation .................................................................................................... 69
Table 6.7 O₂ Estimation Errors using Sine Interpolation of Five Patient Traces over 10 Runs ........................................................................................................ 70
Table 6.8 O₂ Estimation Errors using Linear Regression of Five Patient Traces over 10 Runs ........................................................................................................ 70
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMC</td>
<td>Statistic Model Checking</td>
</tr>
<tr>
<td>PCA</td>
<td>Patient Controlled Analgesia</td>
</tr>
<tr>
<td>PHAVer</td>
<td>Polyhedral Hybrid Automaton Verifier</td>
</tr>
<tr>
<td>NIBIB</td>
<td>National Institute of Biomedical Imaging and Bioengineering</td>
</tr>
<tr>
<td>NIGMS</td>
<td>National Institute of General Medical Sciences</td>
</tr>
<tr>
<td>NIH</td>
<td>National Institute of Health</td>
</tr>
<tr>
<td>MGH/MF</td>
<td>Massachusetts General Hospital/Marquette Foundation</td>
</tr>
<tr>
<td>MIMIC</td>
<td>Multiparameter Intelligent Monitoring in Intensive Care</td>
</tr>
</tbody>
</table>