

Contributors: Benoît Barbot, Alexandru Mereacre, Nicola Paoletti and Andrea Patanè

Summary

HEARTVERIFY is a plug-and-play framework for the analysis and verification of pacemaker software and personalised heart models. Its compatibility with MATLAB Stateflow and extensibility facilitates applications in other domains.

1 Description

Purposes. Cardiac pacemakers are some of the most commonly employed medical devices, with more than one million new pacemaker implanted each year worldwide. Due to the progressive ageing of the population, their use is estimated to grow further in the next few years. To prevent software-related errors that in the past have led to patient’s death and costly device recalls, quantitative formal verification is needed to ensure the safety and reliability of pacemakers. This is challenging because of the need to capture the probabilistic and non-linear dynamics of the heart and connect it in closed-loop with the pacemaker. Moreover, we must take into account the specific physiological characteristics of the patient in order to deliver more effective personalised treatments.

Artifact and benefits. We present HEARTVERIFY, a model-based framework for the analysis and quantitative verification of heart-pacemaker systems. In HEARTVERIFY, models are specified in MATLAB Stateflow and are analysed using statistical model checking, a technique for the verification of probabilistic properties based on the sampling of system’s trajectories. This makes it suitable also for the complex cardiac dynamics we need to reproduce, for which precise (numerical) verification methods would fail.

One important feature of HEARTVERIFY is *modularity*, which allows configuring and testing different heart and pacemaker models in a plug-and-play fashion, without changing their communication interface or the verification engine. This feature makes HEARTVERIFY *easily extensible*. One example is the integration of additional analysis methods, which we showed in previous work conducted in the VERIWARE and VERIPACE projects [2, 1]: parameter synthesis [7, 9], hardware-in-the-loop simulation and energy optimisation [6], or biometric authentication [5].

The main features of the framework are described in the Methods section and implemented in a software package¹, through which the user can:

- experiment with multiple heart models, pacemaker designs and verification properties;
- estimate and analyse personalised heart models from electrocardiogram (ECG) data;
- define custom experiments and extend HEARTVERIFY with new models and algorithms.

Furthermore, the popularity of Stateflow for embedded systems and industrial applications makes HEARTVERIFY an appealing and ready-to-use solution for verification of a large variety of systems, not just from the biomedical domain.

2 Methods

Below we describe the modelling language, the main models and features of the framework, and provide instructions on how to perform the experiments shipped with the package and extend the framework with new models and experiments.

2.1 Modelling and property specification

Probabilistic Timed I/O Automata. HEARTVERIFY builds on a formal specification language called *Timed I/O Automata (TIOA)* [7, 9], which extends classical timed automata with the following features:

1. **Data Variables**, i.e. additional real-valued variables that can be updated through resets.
2. **Parameters**
3. **Non-linear guards and resets:** guard constraints (determining when an edge is enabled) and resets can involve generic and non-linear functions of the variables and parameters. This feature extremely extends the expressiveness of the language, enabling the complex cardiac dynamics we need to reproduce.

¹Available at <http://www.prismmodelchecker.org/files/HeartVerifyFramework.zip>

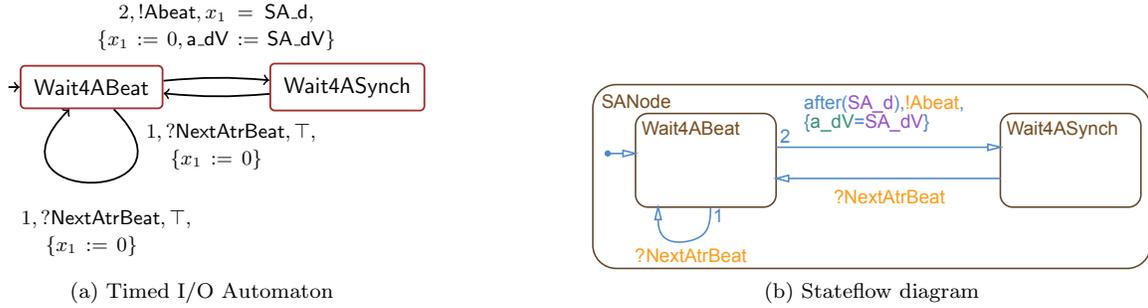


Figure 1: Automata for the SA node

4. **Probabilistic guards:** optionally, probability distributions can be used to determine the delay for an edge to become enabled. One important application of probabilistic guards is in the personalisation of the heart model, where distributions allow describing patient’s features without losing statistical information from the data.

Figure 1a depicts an automaton modelling the sino-atrial node, a component of the heart conduction system. The automaton contains one clock, x_1 , one data variable, a_dV , and two parameters SA_d and SA_dV . Locations are `Wait4ABeat` (initial) and `Wait4ASynch`. Digits 1 and 2 denote priorities, which establish an execution order among enabled edges. The language supports also input and output actions, identified by the prefixes `?` and `!`, respectively. These enable the communication between multiple TIOA components in a network.

Encoding into Stateflow. TIOA models can be seamlessly expressed as MATLAB Stateflow diagrams, which are strictly more expressive than TIOAs. In order to ensure compatibility, the following Stateflow features need to be excluded: hierarchy; state actions; junctions and multiple actions associated to a single edge. Figure 1b shows the Stateflow encoding of the previous example.

The verification engine we chose is COSMOS [3], a statistical model checker for Generalised Stochastic Petri Nets (GSPN) and HASL properties (explained below). The tool relies on the generation of optimised C++ code that implements the simulation, which, combined with the possibility to run on multiple cores, makes verification very efficient. In the framework, we also integrate a recently developed method for translating TIOAs into GSPNs [4].

HASL (Hybrid Automata Stochastic Language) properties [3] consist of two components:

1. a Linear Hybrid Automaton (LHA) that synchronises with the model and determines whether an execution is accepted or not.
2. a mathematical expression over the moments of the LHA variables. This is the actual input to the verification.

Figure 2 shows an example LHA for checking whether the number of heart beats in the first minute is in $[60, 120]$. It contains one initial location, l_0 , and one accepting location, l_1 . The three variables, t , n and y denote the time, the number of ventricular beats and the outcome of the property, respectively. n is increased whenever the LHA synchronises with event `Vbeat`. E indicates the set of all events in the GSPN ². Digits 1 and 2 denote priorities. The first expression in Fig. 2 denotes the satisfaction probability of the property, i.e. the expected value of y at l_1 . The second expression gives the variance of the number of beats per second (n/t). HASL also supports more complex expressions, e.g. with integration of variables over time [3].

2.2 Features

The heart model describes the cardiac electrical conduction system and the propagation of the action potential. Its structure is depicted in Figure 3, where each conduction node corresponds to a TIOA. Typically, impulses travel from the atria to the ventricles through the AV node (blue arrows), but can also follow the opposite path (red arrows), e.g. when the ventricle is paced. The conduction is implemented through synchronisation between the involved components. In this way, the model can be easily extended with other accessory conduction pathways. The figure also shows the connection with a dual-chamber pacemaker (grey arrows). Importantly, the model can generate *synthetic ECG signals* that reflect the events fired during the execution. The following kinds of heart models are provided in HEARTVERIFY:

²Note that the translation of TIOA actions into GSPN events produces slightly different event names.

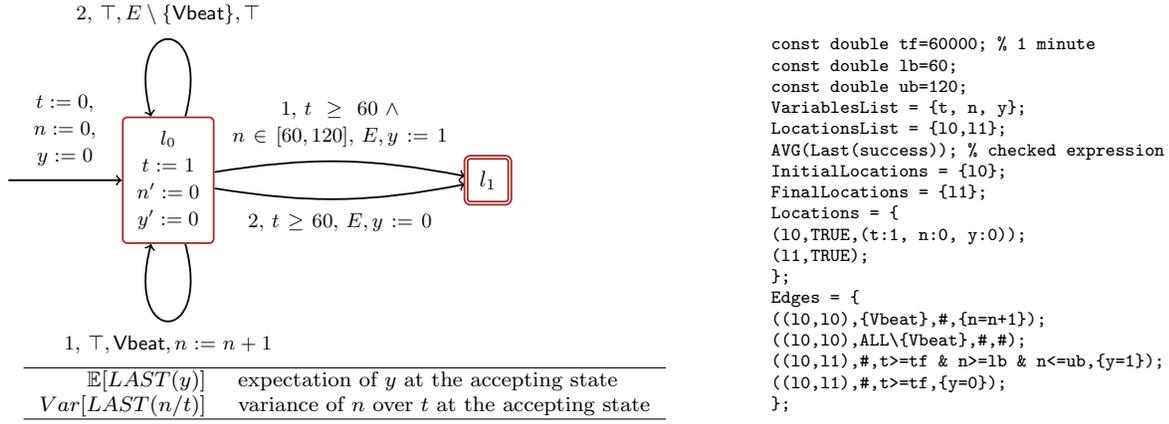


Figure 2: Example automaton and expressions for HASL property (left) and corresponding implementation (right).

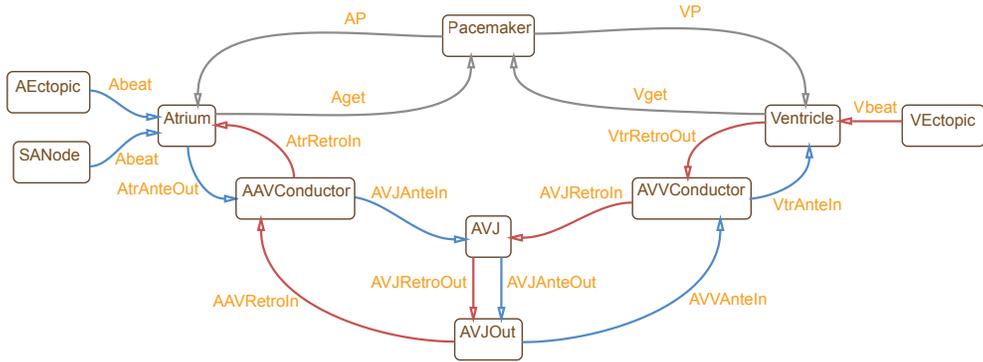


Figure 3: TIOA components in the heart model. Arrows indicate synchronisations and are labelled with the action name.

- deterministic;
- probabilistic; and
- probabilistic, with parameters estimated from data.

Personalisation. HEARTVERIFY also supports the estimation of personalised heart parameters from ECG data [5]. The method consists of the following steps:

1. **Detection of ECG features** (e.g. time between peaks and amplitudes). Some of them are directly mapped to the heart parameters as discrete probability distributions, enabling model-based generation of the synthetic signal (Figure 4).
2. **Estimation of “hidden” parameters** that cannot be directly associated with ECG features. The procedure relies on machine learning techniques (Gaussian Process optimisation) to minimise the statistical distance between the

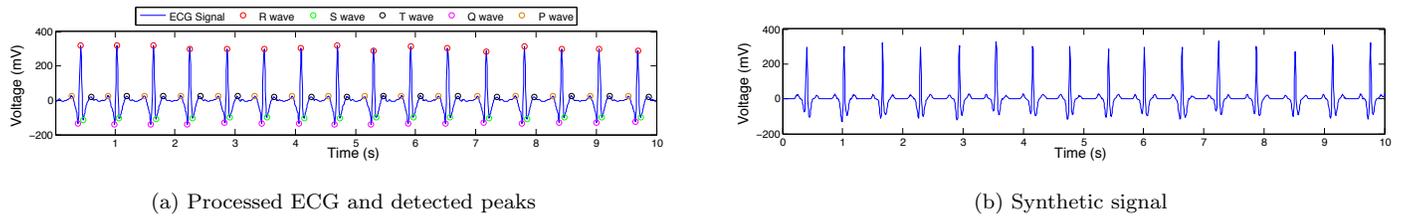


Figure 4: ECG detection and generation of synthetic signal

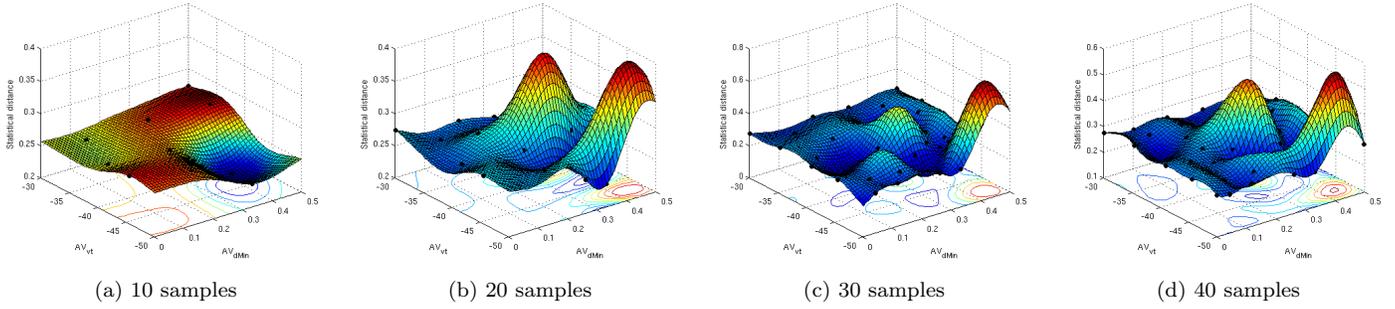


Figure 5: Optimisation of the statistical distance between data and synthetic signals w.r.t. the parameters to estimate. The accuracy of the statistical distance function improves with the number of sampled parameters.

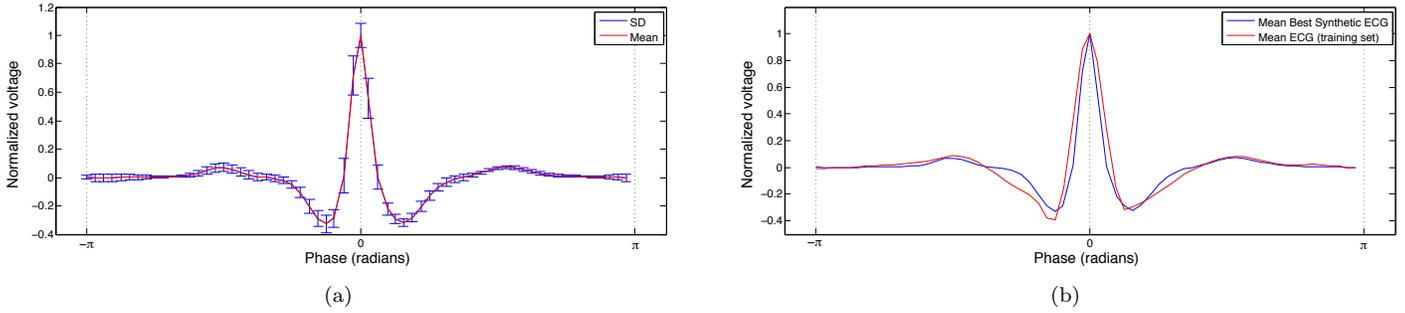


Figure 6: Mean and standard deviation of synthetic ECG (a) and comparison with input data (b).

data and the synthetic ECG produced by the model (Figure 5). Figure 6 shows the good agreement between data and the synthetic ECG associated with the optimal parameters.

The estimation can be performed using custom ECG recordings or by choosing among the patient records provided with the software.

Pacemaker models. In our framework we provide two different pacemaker designs:

- **DDD:** dual-chamber, i.e. it paces and senses both the atrium and the ventricle, and ensures they are synchronised.
- **VVIR:** rate-adaptive [8], where the pacing rate is continuously adapted according to the demand/activity of the patient, detected through the processing of the ECG and other physiological sensors. It paces only the ventricle. We include two models of rate-adaptation (see Figure 7): *open-loop*, with fixed offline ECG data; and *closed-loop*,

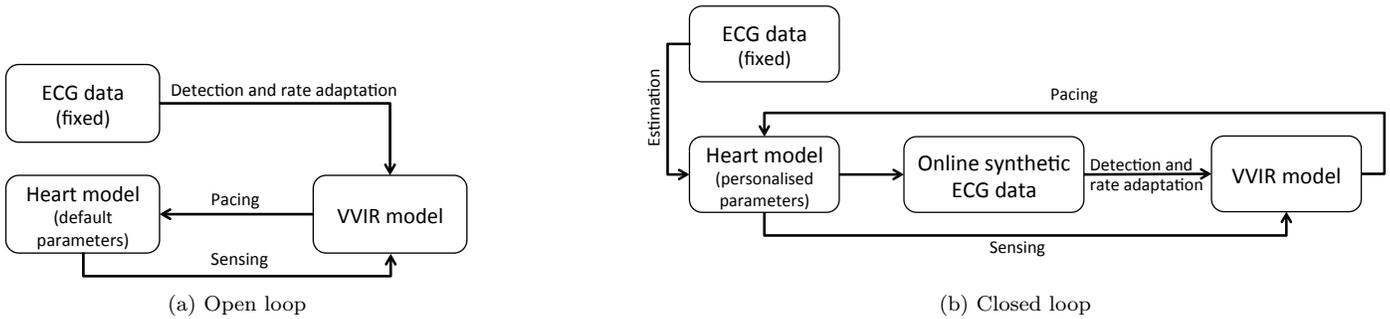
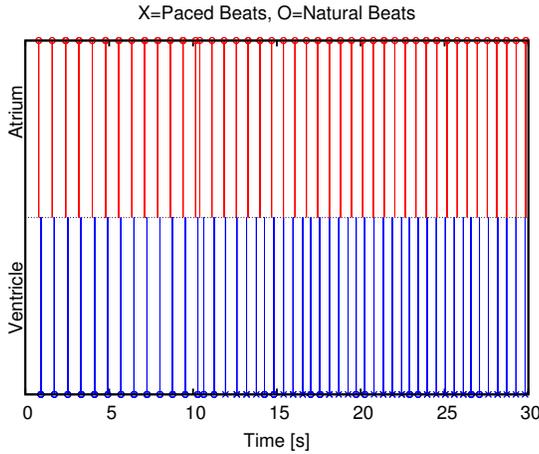
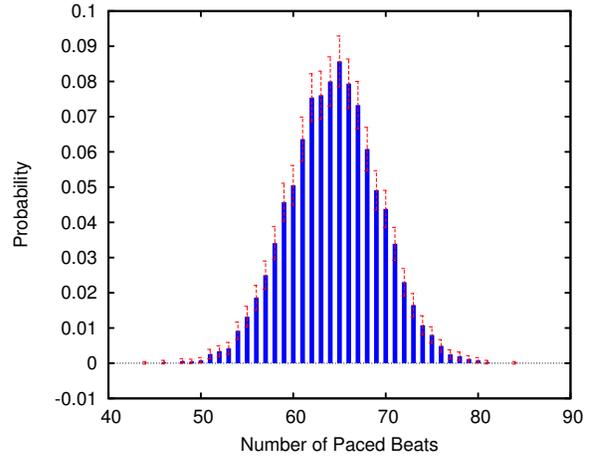


Figure 7: Rate-adaptation models



(a) Stem Plot of the heart events during one execution of the deterministic heart model and the VVIR pacemaker.



(b) Probability density function of the number of paced beats using the probabilistic heart model and the DDD pacemaker.

Figure 8: (a) was obtained with the command:

```
.\HeartVerify -m VVIR -c default -p stem Properties/wait_VVIR.lha
```

(b) was obtained with the command:

```
.\HeartVerify -m DDD -s stochastic -c default -p pdf Properties/waitPDF_DDD_stochastic.lha
```

where the pacemaker detects and adapts its rate from the online synthetic ECG signal. Closed-loop rate-adaptation is a novel feature of the framework, presented for the first time in this artifact evaluation submission.

Properties. We now illustrate some of the properties that we can analyse and verify with HEARTVERIFY:

- probability of slow heart rate or conduction defects;
- probability of sensor-induced tachycardia, occurring when sensing errors induce an excessive pacing rate;
- debug of single executions (Figure 8a);
- probability density function plots (Figure 8b).

2.3 Using and extending the framework

Package structure We include a version of the COSMOS tool³ compiled on Fedora 20 (64-bit) and of the MATLAB SUMO toolbox⁴ (used for Gaussian Process Optimisation). Binaries, header files and libraries of COSMOS are located in the directories `bin`, `include` and `lib`, respectively. The main directory of the package is `HeartVerify`, considered henceforth the working directory. Here, the source code is organized in a modular fashion, as it follows:

- **BlendingCode**: source code for the rate-adaptive routines. Note that the code is self-contained, i.e. it can be executed independently of the verification method (a `Makefile` is provided for ease of compilation), which allows the user to easily implement e.g. new patient-specific models, pacemaker sensors and ECG processing algorithms.
- **ExampleScript**: example scripts for model conversion and execution through COSMOS.
- **Models**: heart and pacemaker models in Stateflow, along with their translation in the `.grml` format required by COSMOS. All the models can be independently run in MATLAB for testing purposes.
- **Properties**: HASL formulas in `.lha` format (Figure 2).
- **SynthECG**: routines for the estimation of patient-specific parameters. Thanks to the modularity of the code, different optimisation engines and algorithm can be tested in place of SUMO.
- **exbat.cpp**: C++ code that can be referred into Stateflow models.

³<http://www.lsv.ens-cachan.fr/Software/cosmos/>

⁴<http://www.sumo.intec.ugent.be/SUMO>

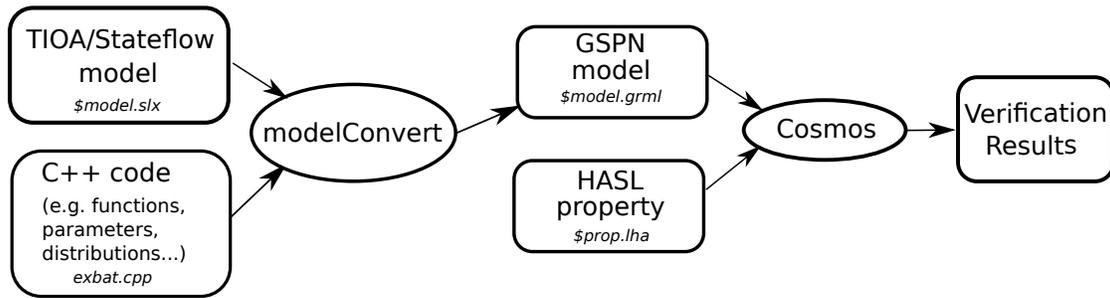


Figure 9: Compilation toolchain.

- **HeartVerify**: command-line interface. Included models, processing routines and properties can be invoked through this script, using suitable options.

For more detailed information, a **Readme** file is available in each folder.

Running experiments. We now show some command-line examples for running the experiments included in **HEARTVERIFY**:

```
./HeartVerify Properties/wait_DDD.lha
```

executes and plots a single trace using the DDD pacemaker configuration. In this case, the property counts the number of heart beats in one minute. The only mandatory input of **HeartVerify** is the HASL property. Option flags can be used to customise the experiment. For example, the command

```
./HeartVerify -le data.txt -m VVIR -s synthetic -c default -p stem -d Properties/wait_VVIR.lha
```

will first learn (-l) heart parameters from the ECG data (-e) in `data.txt`; use the VVIR pacemaker model (-m) and the personalised heart model with synthetic ECG generation capabilities (-s `synthetic`); pass the default options to **COSMOS** (-c `default`); produce a stem plot of the events during one execution (-p `stem`); and produce intermediate plots and results for debugging purposes (-d). Help on the supported flags and their usage is accessed through option -h.

Extending with new experiments. We describe the compilation toolchain of **HEARTVERIFY** (Figure 9), stressing the parts that can be modified and extended. The `modelConvert` procedure translate a Stateflow model into a GSPN model, which can be verified by **COSMOS** given an input HASL property. In the translation process, the model is merged with the external C++ source file `exbat.cpp`. This allows using custom C++ functions (defined and/or included in the source file) in the Stateflow model, which enhances the flexibility and the power of the language, as well as the modularity of the framework, effectively separating the modelling (Stateflow) and the data processing (`exbat.cpp`) aspects. For example the same heart/pacemaker model can be tested against different rate-adaptive specifications; or the pacemaker can be unplugged from the Stateflow model, and the synthetic ECG signal can be used to test hypothesis on patient’s conditions.

Thanks to its compatibility with Stateflow, **HEARTVERIFY** can be extended in countless directions without requiring any specific knowledge of verification techniques, but just some familiarity with **MATLAB/Stateflow** and **C/C++** programming. Below are some more ideas that can be easily realised:

- *Parametric analyses* can be implemented through a simple script that repeatedly executes the experiment on different parameter values. Figure 10 shows an example where we evaluate how heart dynamics is affected by the rate of sensing noise in the pacemaker.
- **COSMOS** can export GSPN models into the **PRISM** language, thus enabling *precise probabilistic model checking of Stateflow models*.
- Creation of a *MATLAB toolbox* for **HEARTVERIFY**, with a GUI that integrates Stateflow with features like graphical modelling of HASL properties, step-by-step debugging and management of verification experiments.

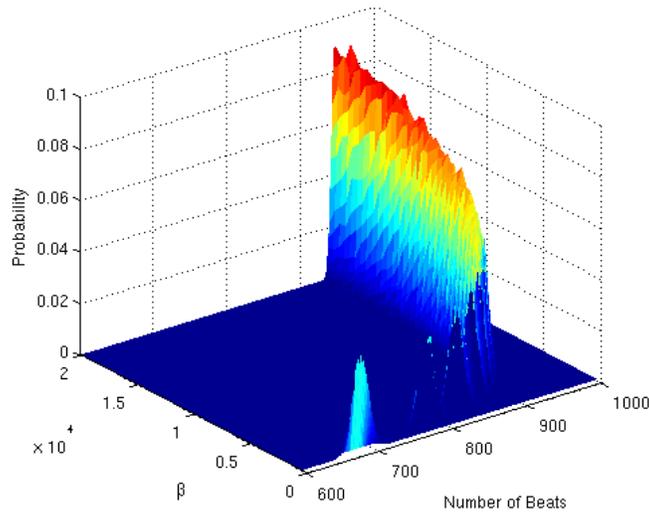


Figure 10: Probability density of the number of heart beats in 10 minutes, under different rates of sensing noise (β).

References

- [1] VERIPACE: Design, Analysis and Synthesis Tools for Cardiac Pacemaker Software. <http://qav.cs.ox.ac.uk/projects/veripace/>, 2015.
- [2] VERIWARE project. Pacemaker Verification. <http://www.veriware.org/pacemaker.php>, 2015.
- [3] P. Ballarini, B. Barbot, M. Dufлот, S. Haddad, and N. Pekergin. Hasl: A new approach for performance evaluation and model checking from concepts to experimentation. *Performance Evaluation*, 2015.
- [4] B. Barbot, M. Kwiatkowska, A. Mereacre, and N. Paoletti. Building power consumption models from executable timed i/o automata specifications. *Submitted to HSCC 2016*, 2015.
- [5] B. Barbot, M. Kwiatkowska, A. Mereacre, and N. Paoletti. Estimation and verification of hybrid heart models for personalised medical and wearable devices. Technical Report CS-RR-15-05, Department of Computer Science, University of Oxford, 2015.
- [6] C. Barker, M. Kwiatkowska, A. Mereacre, N. Paoletti, and A. Patanè. Hardware-in-the-loop simulation and energy optimization of cardiac pacemakers. In *37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, to appear, 2015.
- [7] M. Diciolla, C. H. P. Kim, M. Kwiatkowska, and A. Mereacre. Synthesising optimal timing delays for timed i/o automata. In *Proc. 14th International Conference on Embedded Software (EMSOFT'14)*. ACM, 2014.
- [8] M. Kwiatkowska, H. Lea-Banks, A. Mereacre, and N. Paoletti. Formal modelling and validation of rate-adaptive pacemakers. In *IEEE International Conference on Healthcare Informatics (ICHI)*, pages 23–32. IEEE, 2014.
- [9] M. Kwiatkowska, A. Mereacre, N. Paoletti, and A. Patanè. Synthesising robust and optimal parameters for cardiac pacemakers using symbolic and evolutionary computation techniques. In *Proceedings of the 4th International Workshop on Hybrid Systems and Biology (HSB 2015)*. To appear., 2015.