Posto: Probabilistic Safety Verification Tool for Complex Autonomous Systems

Prachi Bhattacharjee[a], Bineet Ghosh[a], Étienne André[b,c]

[a] *University of Alabama, AL, USA*
[b] *Nantes Université, Nantes, France*
[c] *Institut universitaire de France (IUF)*

Abstract

The rapid advancement of autonomy in modern systems has led to a surge in model complexity, making safety verification using traditional tools increasingly difficult. In response, we introduce **Posto**, a statistical monitoring tool that performs safety verification of complex autonomous systems, including those driven by deep neural networks (DNN). While traditional monitoring tools depend on formal models to assess safety, **Posto** only requires the system's input–output behavior (I/O execution) and the log (which may be noisy and incomplete). We provide easy-to-use scripts and comprehensive documentation to enable users to reproduce our experiments, monitor their own systems using logs, and extend the tool for their own application. We evaluate **Posto** through multiple case studies involving complex autonomous systems with DNN components.

Keywords: monitoring, neural networks, safety, monitoring tool, statistical verification

1. Motivation and significance

Over the past decade, distributed autonomous systems have been rapidly adopted across a wide range of domains. Although traditional formal verification tools have shown success in many safety-critical settings [3], they are increasingly struggling to cope with the rising complexity of these systems. In particular, these tools typically require a formal model—something that is often unavailable due to black-box components or the absence of a complete system model. To address these challenges, we introduce **Posto**[1] (see details in Table 1), a probabilistic monitoring tool that implements a statistical

[1] Tool webpage: https://autmn-lab.github.io/Posto/

C1	Current code version	v1.0
C2	Permanent link to code/repository used for this code version	https://github.com/autmn-lab/posto_tool/releases/tag/v1.0
C3	Permanent link to Reproducible Capsule	https://doi.org/10.5281/zenodo.18233568
C4	Legal Code License	*GNU General Public License v3.0* (`gpl-3.0`) [1]
C5	Code versioning system used	`git`
C6	Software code languages, tools, and services used	**`Python`**, **`numpy`**, **`scipy`**, **`mpmath`**
C7	Compilation requirements, operating environments and dependencies	See **`Posto`** Manual [2]
C8	Link to developer documentation/manual	See **`Posto`** Manual [2]
C9	Support email for questions	**autmnlab.cs@gmail.com**

Table 1: Code metadata

hypothesis testing based algorithm (see [4]) to assess system safety directly from the execution logs, even when the logs contain noisy or missing samples. Using only the system's input–output behavior (independent of internal implementations, including DNN-based components), **`Posto`** can either certify the system as safe with a high probabilistic guarantee, or generate a concrete counterexample when a violation is detected.

Contributions. Monitoring has emerged as a lightweight yet practically effective verification technique capable of handling systems of considerable complexity (e.g., [5, 6, 7, 8]). Statistical monitoring, in particular, aims to ensure that autonomous systems are trustworthy, which means that they are safe for most practical purposes with a high level of confidence. By relaxing strict formal safety guarantees in favor of statistical ones (using the method proposed in [4]), **`Posto`** provides two key advantages: *i*) it is broadly applicable and can accommodate a wide range of system types—including nonlinear and DNN-based models—by relying solely on the system's input/output execution behavior; and *ii*) it allows users to specify the desired confidence level for the analysis. For e.g., in a safety critical scenario, a high confidence level (e.g., 0.9999) may be chosen, while in less critical situations, a lower confidence level can be selected. **`Posto`** requires less time for monitoring as the desired confidence level decreases.

Related Work. Signal Temporal Logic (STL) formalizes real-valued signal properties in dense-time contexts. In addition to STL-based monitoring, recent attention has shifted towards monitoring using automata-based specifications. Techniques such as timed pattern matching on timed regular expressions [9, 10, 11] have emerged, particularly in the context of deter-

ministic offline monitoring. The concept of *model-bounded monitoring*, introduced in [12], deviates from the conventional approach of monitoring a black-box system solely against a specification. Instead, it incorporates a limited, over-approximated understanding of the system to mitigate false positives. The over-approximated knowledge takes the form of a *linear hybrid automaton* [13]. Model-bounded monitoring was extended in the framework of uncertain linear systems in [8]. Unlike all these deterministic approaches, our proposed monitoring approach does not require any formal model—and can handle complex models such as non-linear models, DNNs etc.—of the system. It trades off formal safety guarantees with high confidence probabilistic guarantees.

In addition, *statistical verification* has found extensive application across various domains [14].

Outline. We briefly recall the software in Section 2: we provide illustrative examples in Section 3; we discuss the potential impact of **`Posto`** in Section 4; we conclude in Section 5.

2. Software description

`Posto` is an open-source Python-based software designed to run on Linux platforms.

2.1. Software architecture

The architecture (and dataflow) is given in Fig. A.1. Each block corresponds to a core component of the tool's architecture, with its specific role indicated in the diagram. We now wish to discuss the core architectural components of **`Posto`**, followed by a discussion of its key software functionalities and the role of these components in supporting them.

Inputs and Outputs. The main inputs to our tool are: the I/O execution system model, the log (with noisy and missing samples), and the safety specifications against which the system is to be analyzed (with a high probabilistic guarantee). Analyzing these inputs (in **`Block 5`**), the tool produces either a safety certificate, indicating that the system satisfies the safety conditions with a high probabilistic guarantee, or a counterexample that reconstructs a concrete system behavior of the system (from the given log) that violates the safety condition. In addition, it also generates plots illustrating both safe and unsafe behaviors that provides users with deeper insight into the system's execution.

Check Safety **(*Block 5*)**. This is the main block of the tool that is responsible to perform the safety verification of the system. It takes the safety constraints specified by the user as an input, along with the computed value K from **`Block 3`**, which denotes the number of valid system trajectories that must be checked to achieve the required probabilistic guarantee. It is worth noting that these trajectories are valid reconstructions of the system behavior from the given log, where the log is either generated synthetically (in **`Block 2`**) or taken from a real system execution. Once the inputs are collected, the block verifies whether *all* K valid trajectories satisfy the safety constraints. If they do, the tool asserts the system as safe with the required probabilistic confidence. Otherwise, the trajectory violating the safety constraint is returned as a counterexample, which serves as a concrete system behavior recreated from the log that violates safety.

2.2. Software functionalities

Safety Verification. **`Posto`** enables scalable safety verification of complex autonomous systems, including systems with DNN components, using logs that may contain missing and noisy samples. The verification is performed against user-defined safety specifications with high probabilistic guarantees. The tool takes the system's I/O execution model and the log as inputs. **`Posto`** performs safety analysis by reconstructing random system trajectories from the given log. If no trajectory violating the safety specification is identified, the system is asserted to be safe with the required probabilistic guarantee. Otherwise, the trajectory violating safety is returned as a concrete (i.e., reconstructed from the log) counterexample.

Log Generation. While **`Posto`** is primarily designed for safety verification of a system using a user given log, it also supports the generation of synthetic logs that can be used for safety analysis. In addition, the tool provides visualizations of the generated logs, offering a more intuitive and physically meaningful understanding of the system's execution in real-world conditions.

Behavior Generation. **`Posto`** not only supports safety verification of complex autonomous systems, but also provides diagnostic tools to analyze system behavior and evaluate its robustness to different logging parameters, making it useful for system designers.

3. Illustrative examples

In this section, we reproduce the experimental results reported in [4, Section 6] for the Jet model, Van der Pol oscillator and the Mountain Car case studies. These experiments are designed to assess the scalability and

practical usability of the tool on complex autonomous systems, including systems with DNN-based components. In particular, the evaluation studies: *i*) the impact of varying the logging probability (i.e., the number of records in the log), *ii*) the impact of varying the amount of uncertainties in the log samples, *iii*) impact of probabilistic confidence c, *iv*) effect of various other parameters.

Note on stochasticity of the recreations. It is worth noting that the safety verification method proposed in our work is inherently statistical. Consequently, the reproduction of the plots will not yield an exact match but rather a stochastic recreation. In other words, the plots are recreated using the same set of parameters as in the draft, which define the distributions. The recreated plots represent one possible outcome from that distribution, hence the term "stochastic recreation". As a result, some figures that were initially inferred to be safe in the draft may yield unsafe results (or vice versa) during this process.

Jet Model. The main results of the Jet Model study are provided in Figs. A.2 and A.3 (as in [4, Figs. 2 and 3]). Once the tool is downloaded and set up, the experimental results can be reproduced using the script[2] provided in our GitHub repository, with detailed usage instructions available in [15] (and additional steps in Appendix A). For instance, to recreate the result shown in Fig. A.2a (similarly, Figs. A.2b and A.2c, $\cdots$, Figs. A.3c and A.3d), one can execute the following command:

```
python artEval.py --fig=A2a
```

Van der Pol Oscillator. The main results of the Van der Pol Oscillator study are provided in Fig. A.4 (as in [4, Fig. 4]). The results for the Van der Pol Oscillator can be reproduced in a manner similar to the Jet Case Study (see Appendix A for details). For example, to recreate the result shown in Fig. A.4a, one can run the following command:

```
python artEval.py --fig=A4a
```

Mountain Car. We also present additional experiments (beyond those reported in [4]) using a DNN-based controller for the mountain car benchmark [16] in Appendix B. These experiments further demonstrate the efficacy and applicability of the proposed tool.

[2] https://github.com/autmn-lab/posto_tool/blob/master/artEval.py

4. Impact

Although **Posto** is still a prototype tool based on a recent algorithm [4]—and not yet an industry-adopted software—we believe it has potential to develop a user base interested in monitoring complex autonomous systems (including those with DNN-based and black-box components) against safety properties.

Posto is capable of efficiently monitoring complex systems against safety properties very efficiently. It operates directly on noisy logs (even with missing samples) while relying only on the I/O execution model of the system to assess its safety. This makes **Posto** particularly suitable for modern autonomous systems, where perfect logs and formal models are rarely available. Given its broad applicability, minimal modeling requirements, and computational efficiency, we believe **Posto** has the potential to open up new research directions in the verification of complex autonomous systems, especially in the context of statistical and data-driven safety analysis.

Furthermore, **Posto** can serve as a valuable resource for researchers and engineers working on the design of modern autonomous systems. By integrating **Posto** into their current processes, they can evaluate the safety of their designs. They can then analyze the resulting safety guarantees or violations (counterexamples) to gain deeper insights and improve their designs.

5. Conclusions and Future Work

We introduced here **Posto**, an open-source Python-based software, aiming at performing statistical monitoring for safety verification of complex autonomous systems, including those driven by deep neural networks (DNN). While traditional monitoring tools depend on formal models to assess safety, **Posto** only requires the system's input–output behavior (I/O execution) and the log (which may be noisy and incomplete). Our experiments demonstrate that **Posto** can efficiently monitor reasonably complex systems, including those equipped with DNN-based controllers. We believe **Posto** will be valuable not only for engineers to analyze system logs to detect safety violations—such as collision risks or other undesirable behaviors in robotics and autonomous vehicles—but also for researchers developing new monitoring-based methods. In such cases, **Posto** can serve as a useful comparative baseline.

Given the scalability of our method and the observations it offer, our implementation paves the way for several future research directions. Our ultimate aim is to adopt a fully representation-free strategy, where the I/O execution representation will be co-designed with the monitoring approach.

References

[1] Free Software Foundation, Gnu general public license, version 3.
URL https://www.gnu.org/licenses/gpl-3.0.en.html

[2] A. Lab, Posto: Probabilistic Safety Monitoring – User Manual, Technical report, University of Alabama, User Manual (2024).
URL https://autmn-lab.github.io/Posto/files/Posto-manual.pdf

[3] B. Ghosh, É. André, MoULDyS: Monitoring of autonomous systems in the presence of uncertainties, Science of Computer Programming 230 (Aug. 2023). doi:10.1016/j.scico.2023.102976.

[4] B. Ghosh, É. André, Probabilistic safety verification of distributed systems: A statistical approach for monitoring, in: C. Ferreira, C. A. Mezzina (Eds.), FORTE, Vol. 15732 of Lecture Notes in Computer Science, Springer, 2025, pp. 114–133. doi:10.1007/978-3-031-95497-9_7.

[5] D. A. Basin, G. Caronni, S. Ereth, M. Harvan, F. Klaedtke, H. Mantel, Scalable offline monitoring of temporal specifications, Formal Methods in System Design 49 (1-2) (2016) 75–108. doi:10.1007/s10703-016-0242-y.

[6] E. Bartocci, J. V. Deshmukh, A. Donzé, G. E. Fainekos, O. Maler, D. Ničković, S. Sankaranarayanan, Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications, in: E. Bartocci, Y. Falcone (Eds.), Lectures on Runtime Verification – Introductory and Advanced Topics, Vol. 10457 of Lecture Notes in Computer Science, Springer, 2018, pp. 135–175. doi:10.1007/978-3-319-75632-5_5.

[7] B. Ghosh, É. André, Offline and online monitoring of scattered uncertain logs using uncertain linear dynamical systems, in: M. Mousavi, A. Philippou (Eds.), FORTE, Vol. 13273 of Lecture Notes in Computer Science, Springer, 2022, pp. 67–87. doi:10.1007/978-3-031-08679-3_5.

[8] B. Ghosh, É. André, Offline and online energy-efficient monitoring of scattered uncertain logs using a bounding model, Logical Methods in Computer Science 20 (1) (2023) 2:1–2:33. doi:10.46298/lmcs-20(1:2)2024.

[9] M. Waga, T. Akazaki, I. Hasuo, A Boyer-Moore type algorithm for timed pattern matching, in: M. Fränzle, N. Markey (Eds.), FORMATS, Vol. 9884 of Lecture Notes in Computer Science, Springer, 2016, pp. 121–139. doi:10.1007/978-3-319-44878-7_8.

[10] A. Bakhirkin, T. Ferrère, D. Ničković, O. Maler, E. Asarin, Online timed pattern matching using automata, in: D. N. Jansen, P. Pavithra (Eds.), FORMATS, Vol. 11022 of Lecture Notes in Computer Science, Springer, 2018, pp. 215–232. doi:10.1007/978-3-030-00151-3_13.

[11] M. Waga, É. André, I. Hasuo, Parametric timed pattern matching, ACM Transactions on Software Engineering and Methodology 32 (1) (2023) 10:1–10:35. doi:10.1145/3517194.

[12] M. Waga, É. André, I. Hasuo, Model-bounded monitoring of hybrid systems, ACM Transactions on Cyber-Physical Systems 6 (4) (2022) 30:1–30:26. doi:10.1145/3529095.

[13] N. Halbwachs, Y.-É. Proy, P. Raymond, Verification of linear hybrid systems by means of convex approximations, in: B. Le Charlier (Ed.), SAS, Vol. 864 of Lecture Notes in Computer Science, Springer, 1994, pp. 223–237. doi:10.1007/3-540-58485-4_43.

[14] A. Legay, B. Delahaye, S. Bensalem, Statistical model checking: An overview, in: H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. J. Pace, G. Rosu, O. Sokolsky, N. Tillmann (Eds.), RV, Vol. 6418 of Lecture Notes in Computer Science, Springer, 2010, pp. 122–135. doi:10.1007/978-3-642-16612-9_11.

[15] A. Lab, Posto: Probabilistic Safety Monitoring – Recreating Results, Technical report, University of Alabama, Recreating Results (2024). URL https://autmn-lab.github.io/Posto/files/Posto-recreation.pdf

[16] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, I. Lee, Verisig: Verifying safety properties of hybrid systems with neural network controllers, in: N. Ozay, P. Prabhakar (Eds.), HSCC, ACM, 2019, pp. 169–178. doi:10.1145/3302504.3311806.

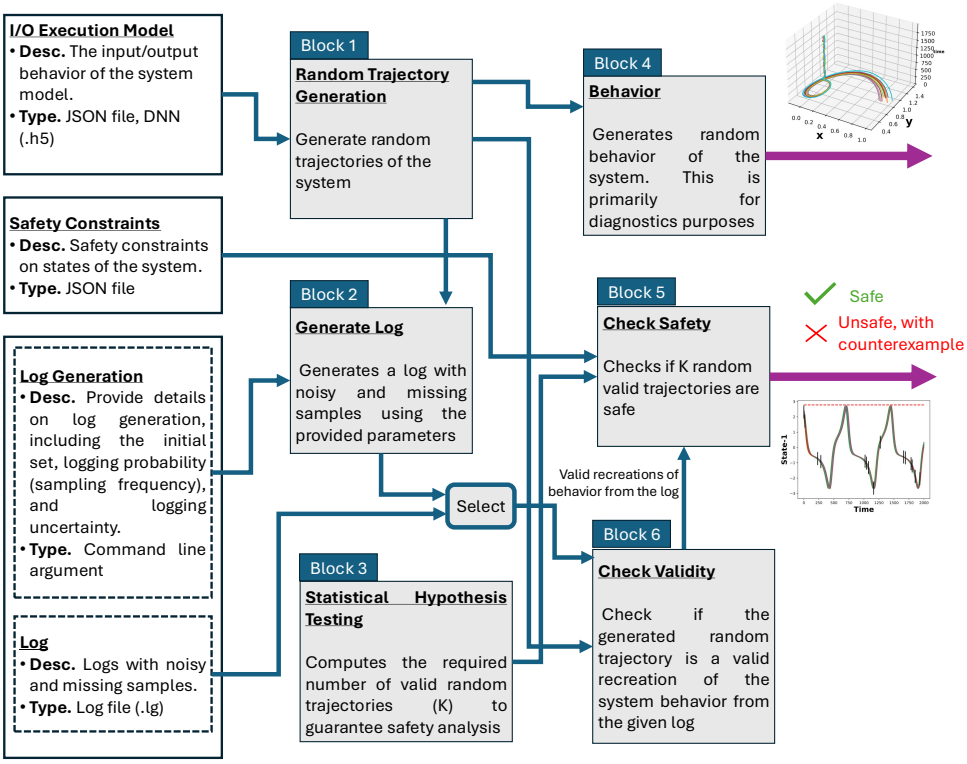


Figure A.1: **Posto Architecture and Dataflow.** Each block represents a core architectural component. Inputs are shown as hollow rectangles on the left, internal components as solid gray rectangles, and outputs as purple arrows on the right. Interactions between components are indicated by solid blue arrows.

Appendix A. Recreating Existing Results

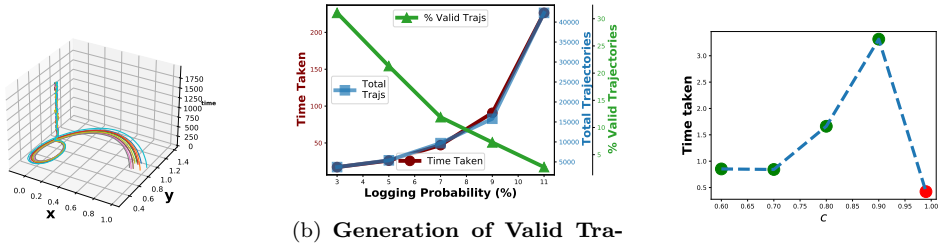
This appendix lists the exact commands required to recreate the figures reported in the paper. All commands are executed from the root directory of the **Posto** repository using the `artEval.py` script [3]. Due to the stochastic nature of the proposed monitoring approach, regenerated figures are stochastically equivalent but not necessarily identical.

Environment Setup.

```
cd /path/to/Posto
export POSTO_ROOT_DIR=/path/to/Posto
```

Figures A.2(a)-(c). To generate Fig. A.2, run:

[3] https://github.com/autmn-lab/posto_tool/blob/master/artEval.py



(a) **Random Trajectories** (b) **Generation of Valid Trajectories** (c) **Probabilistic confidence** c

Figure A.2: **Jet Model Case Study [4, Fig. 2].** *Random Trajectories (Fig. A.2a).* x- and y- axis represents state variables, and z-axis represents time step. *Impact of Logging Probability on Generation of Valid Trajectories (Fig. A.2b).* We illustrate how varying the logging probability affects the monitoring execution time and the percentage of valid trajectories out of the total number of generated trajectories. *Impact of Probabilistic confidence c (Fig. A.2c).* We illustrate the impact of logging probability on the time taken to perform monitoring and safety inferences. The red dot denotes an instance where the system was inferred to be unsafe with a counterexample, and green dots denote instances where the system was inferred to be probabilistically safe with the corresponding confidence value indicated in the x-axis.

```
python artEval.py --fig=A2a
python artEval.py --fig=A2b
python artEval.py --fig=A2c
```

Figures A.3(a)-(d). To generate Fig. A.3, run one of the following commands:

```
python artEval.py --fig=A3a
python artEval.py --fig=A3b
python artEval.py --fig=A3c
python artEval.py --fig=A3d
```

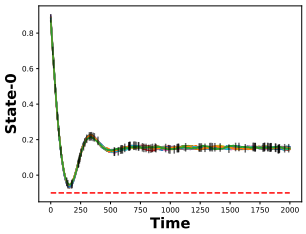
Figures A.4(a)-(d). To generate Fig. A.4, run one of the following commands:

```
python artEval.py --fig=A4a
python artEval.py --fig=A4b
python artEval.py --fig=A4c
python artEval.py --fig=A4d
```

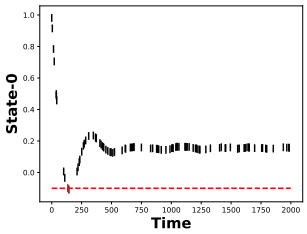
Note that for each command, the auxiliary plot is displayed first, followed by the corresponding Fig. A.4 subfigure. Closing the first window reveals the target figure.

Figure B.5. To generate Fig. B.5, run:

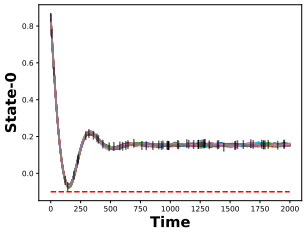
```
python artEval.py --fig=B5
```



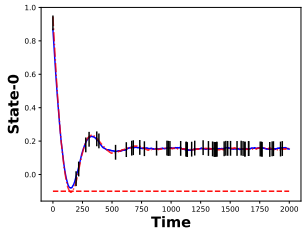
(a) Frequent records, low uncertainty



(b) Frequent records, high uncertainty



(c) Sporadic records, low uncertainty



(d) Sporadic records, high uncertainty

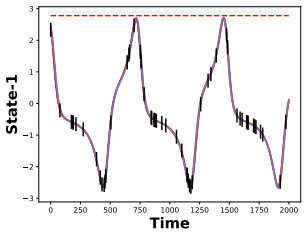
Figure A.3: **Jet Monitoring [4, Fig. 3].** Evolution of the state—x for the jet model—with time. The volume of the records increases from left to right, and the probability of logging increases from bottom to top. The colored trajectories are the generated random valid trajectories during the process of monitoring, the dotted trajectory in red is an unsafe trajectory discovered during the process of monitoring (see Fig. A.3d), the black regions are records in the log given as an input to the monitoring algorithm, and the dark-brown regions are unsafe-records in the log (Fig. A.3b). The red dotted line represents the safe range for the variable.

Appendix B. Mountain Car Case Study

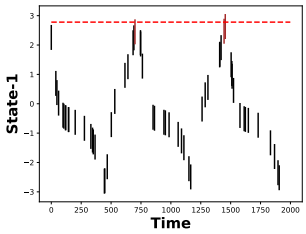
In this appendix, we report new results for a Mountain Car Controller that uses a neural-network-based policy [16]. The controller, provided as a trained `.h5` model, is executed using the `ANN` mode of `Posto` and maps the car's position and velocity to an acceleration command which is used to compute the next position and velocity of the car [16]. The discretized closed-loop dynamics are subject to injected environmental uncertainty. Safety is defined by requiring the velocity to remain below $v \leq 0.055$ at all time steps. Logs are generated under four configurations combining logging probabilities of 20% and 40% with uncertainty levels of 0.004 and 0.008. Only the configuration with 40% logging probability and uncertainty 0.008 violates the safety constraint, while the remaining cases are inferred to be safe, as shown in Fig. B.6.

Recreating Figure B.6

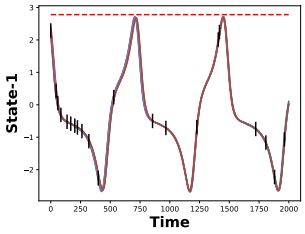
Fig. B.6 reports the monitoring results for the Mountain Car controller across all four logging and uncertainty configurations. To recreate all subfig-



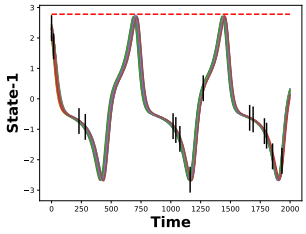
(a) Frequent records, low uncertainty



(b) Frequent records, high uncertainty



(c) Sporadic records, low uncertainty



(d) Sporadic records, high uncertainty

Figure A.4: **Van der Pol Oscillator Monitoring [4, Fig. 4].** Evolution of the state—y for the Van der Pol model—with time. The volume of the records increases from left to right, and the probability of logging increases from bottom to top. The colored trajectories are the generated random valid trajectories during the process of monitoring, the dotted trajectory in red is an unsafe trajectory discovered during the process of monitoring (see Fig. A.4d), the black regions are records in the log given as an input to the monitoring algorithm, and the dark-brown regions are unsafe-records in the log (Fig. A.4b). The red dotted line represents the safe range for the variable.

ures of Fig. B.6, run the `artEvalNN.py` script [4] using the following commands from the root directory of the **`Posto`** repository:

```
python artEvalNN.py --fig=B6a
python artEvalNN.py --fig=B6b
python artEvalNN.py --fig=B6c
python artEvalNN.py --fig=B6d
```

Each command generates the corresponding subfigure of Fig. B.6, reflecting one combination of logging probability and uncertainty level. Due to the stochastic nature of the monitoring procedure, regenerated plots may differ slightly across runs while preserving the qualitative behavior and safety outcomes reported in the paper.

[4] https://github.com/autmn-lab/posto_tool/blob/master/artEvalNN.py

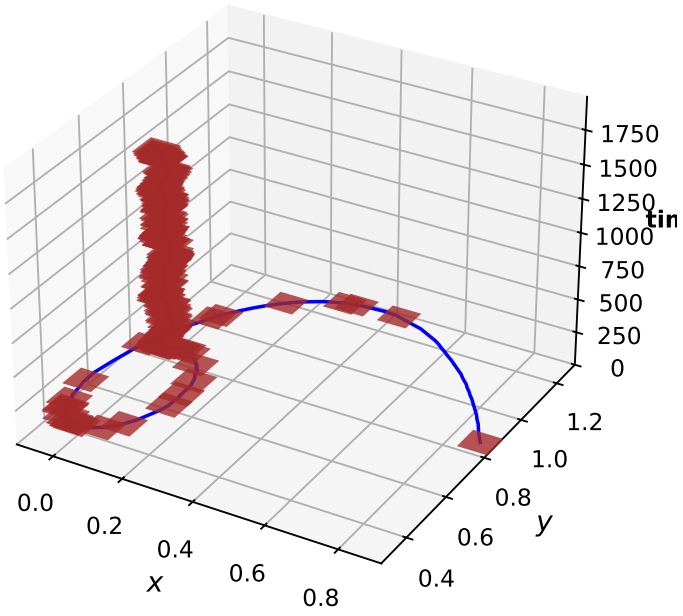
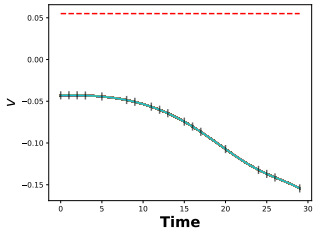
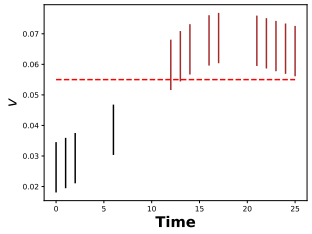


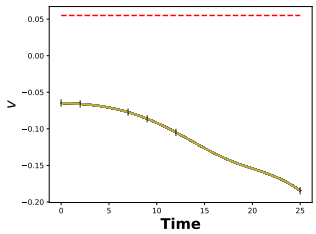
Figure B.5: **Log Generation Example.** The reference trajectory is shown in blue, and the brown boxes illustrate the corresponding noisy samples of the log obtained using with given logging probability and noise level.



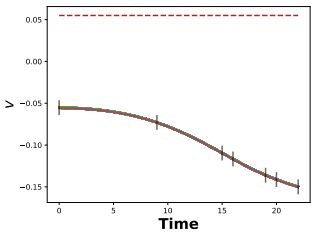
(a) Frequent records, low uncertainty



(b) Frequent records, high uncertainty



(c) Sporadic records, low uncertainty



(d) Sporadic records, high uncertainty

Figure B.6: **Mountain Car Controller Monitoring.** Evolution of the state—v for the Mountain Car Controller—with time. The volume of the records increases from left to right, and the probability of logging increases from bottom to top. The colored trajectories are the generated random valid trajectories during the process of monitoring, the black regions are records in the log given as an input to the monitoring algorithm, and the dark-brown regions are unsafe-records in the log (Fig. B.6b). The red dotted line represents the safe range for the variable.