HyMU: a software for Hybrid Systems Health Monitoring under Uncertainty

Elodie Chanthery[1,2], **Pauline Ribot** [1,3] and **Amaury Vignolles**[1,2]
[1]CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
e-mail: first.last@laas.fr
[2]Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
[3]Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France

Abstract

HyMU (Hybrid system Monitoring under Uncertainty) is a software developed in Python by the DISCO team at LAAS-CNRS. It was designed to simulate, diagnose and prognose hybrid systems using model-based methods. Its main feature is to deal with many types of uncertainties (modeling uncertainty, unreliable observations and unknown future inputs). A multimode representation of the hybrid system has to be established by specifying continuous and discrete evolutions defining its behavior and its degradation. From this representation, HyMU computes a Hybrid Particle Petri Net (HPPN) model, a diagnoser and a prognoser. The HPPN model can be simulated with an input scenario to compute the system outputs and health mode. From the HPPN model and a set of observations, the diagnoser and the prognoser compute the current and future mode beliefs, the mode trajectories and the predicted Remaining Useful Life (RUL). This paper describes HyMU, its functionality and gives some examples of results.

1 Introduction

The complexity of systems has evolved to reach a point where it is often impossible for humans to fully understand and explain their behaviors, especially with the occurrence of faults, leading to failures. This is why efficient diagnosis and prognosis techniques have to be adopted to detect, isolate and prevent faults and therefore, failures. As it becomes harder and harder to execute these techniques, tools have been developed to ease and speed up their execution, and to reduce the global costs of unavailability and repair actions. The goal of this paper is to introduce one of these tools: HyMU (Hybrid system Monitoring under Uncertainty).

HyMU is a software developed in Python by the DISCO team at LAAS-CNRS. It is available on github [1] under a LGPL license. Based upon the Hybrid Particle Petri Nets (HPPN) formalism, which is an extension of Petri Nets, it was designed to simulate, diagnose and prognose hybrid systems using model-based methods. Its two main features are (1) to deal with many types of uncertainties (modeling uncertainty, unreliable observations, unknown future inputs), (2) to focus on the degradation, considered not only as a function of the continuous state but also as a function of the set of events that have occurred on the system.

Basically, the user gives as input a multimode representation of the hybrid system and provides a scenario of inputs for the system to monitor. HyMU is then able to simulate the system model by using an implicit HPPN model of the system and to monitor the health state of the system by building a diagnoser and a prognoser. The use of the HPPN formalism is necessary to cope with uncertainties and degradation, which are not taken into account by more classical formalisms such as hybrid automata. The diagnosis results take the form of a set of health mode beliefs, while the prognosis results are represented as estimated End Of Life (EOL) values for the system or Remaining Useful Life (RUL) beliefs.

The paper is organized as follows. Section 2 explains the models used by HyMU, i-e. the multimode representation and the HPPN formalism. Section 3 presents the HPPN-based health monitoring methodology which is implemented in HyMU. Section 4 presents the HyMU software in details and explains how to model, simulate, diagnose and prognose using the HyMU tool. Section 5 shows the application of HyMU on a three-tank system, and the results obtained. Finally, section 6 and section 7 present other existing tools and introduce future work we would like to implement.

2 System modeling for HyMU

The main goal of HyMU is to follow and evaluate the health modes of the system. Health modes are defined as follows. More details are provided in [1].

Definition 1 (Health mode)**.** *A health mode of the system is a combination of one discrete state and associated continuous dynamics and degradation evolution.*

HyMU requires as inputs a multimode representation of the system and a scenario. The multimode representation is then used to build an implicit HPPN model of the system. These two kinds of models are recalled in the next subsections.

2.1 Multimode representation

To our knowledge, the multimode representation concept has been introduced in [2]. It is based on the concepts of hybrid automaton [3] and captures the system's intra mode behavior through the continuous dynamics and the continuous variables. As said before, the difference between usual continuous dynamics and degradation is that degradation is

[1]https://github.com/echanthe/HyMU

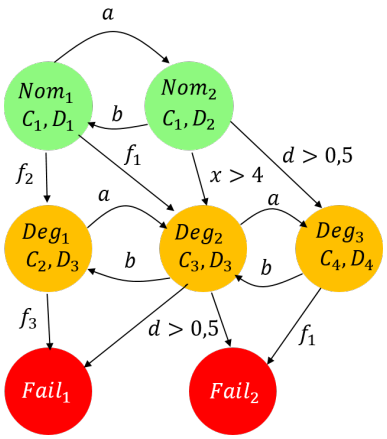

Figure 1: Example of a multimode representation for a system.

function of the continuous state and the set of events that have occurred on the system. We then adapted the concept of multimode representation and give here our own formal definition for hybrid system including degradation.

Definition 2 (Multimode representation)**.** *A multimode representation is a finite multidigraph $MR = (V_m, A_m)$ where V_m is the set of vertices representing the health modes of the system and A_m is the set of arcs representing the transitions between health modes. Each arc in A_m is associated with a set of conditions that can imply discrete events of E or continuous conditions. The continuous conditions are named guards and represent constraints on the continuous state and/or the degradation state.*

The set $E = E_o \cup E_{uo}$ is the set of event labels that is partitioned into observable event labels E_o and unobservable event labels E_{uo}. For example, an anticipated fault in the system model is represented by an unobservable event $f \in E_{uo} \subset E$. An event e is a couple $e = (v, k)$ where $v \in E$ is an event label (or type) and $k \in \mathbb{R}$ the time of occurrence of e. An event (v, k) is unobservable if for all k, v does not belong to the set of discrete observations of the system.

Figure 1 illustrates a multimode representation for a system with 7 health modes. Nominal modes are represented by green circles, degraded modes are represented by orange circles and failure modes by red circles. Each mode is associated with continuous dynamics and degradation dynamics. The system has for example 2 nominal modes Nom_1 and Nom_2 in green, observable events are $E_o = \{a, b\}$ and $E_{uo} = \{f_1, f_2, f_3\}$ are the faults that may occur on the system. x is the continuous state, and d the degradation state. Transitions between Nom_2 and Deg_2 or between Nom_2 and Deg_3 are two guards on continuous values. Figure 6 is also an example of multimode representation, obtained via HyMU, which continuous and degradation dynamics are given in the model description file on github.

2.2 Hybrid Particle Petri Nets

As argued before, the use of a formalism that copes with uncertainties and degradation for health monitoring purpose, which are not taken into account by more classical formalisms such as hybrid automata, is necessary. Our solution is the definition of the Hybrid Particle Petri Nets (HPPN), which have been presented in detail in [1]. In this section, we give the main ideas of the formalism, but we recommend reading the original paper for a better understanding.

Definition 3. *An HPPN is defined as a* 11*-tuple $\langle P, T, A, \mathcal{A}, E, X, D, \mathcal{C}, \mathcal{D}, \Omega, \mathbb{M}_0 \rangle$ which describes discrete states (with symbolic places), continuous dynamics (with numerical places) and degradation evolution (with degradation places) of a system and relations between them:*

- *P is the set of places, partitioned into numerical places P^N, symbolic places P^S and degradation places P^D, $P = P^S \cup P^N \cup P^D$,*
- *T is the set of transitions,*
- *$A \subseteq P \times T \cup T \times P$ is the set of arcs,*
- *$\mathcal{A}$ is the set of arc annotations,*
- *E is the set of event labels,*
- *$X \subseteq \mathbb{R}^{n_N}$ is the state space of the continuous state vector, with $n_N \in \mathbb{N}_+$ the number of continuous state variables,*
- *$D \subseteq \mathbb{R}^{n_D}$ is the state space of the degradation state vector, with $n_D \in \mathbb{N}_+$ the number of degradation state variables,*
- *$\mathcal{C}$ is the set of continuous dynamics equation sets associated with numerical places,*
- *$\mathcal{D}$ is the set of degradation equation sets associated with degradation places,*
- *Ω is the set of conditions associated with transitions,*
- *$\mathbb{M}_0$ is the initial marking of the Petri net.*

The places of an HPPN are marked by tokens that carry different types of information.

Symbolic places P^S model the discrete states of the system and are marked by symbolic tokens called configurations. The set of configurations at time k is denoted $\mathbb{M}_k^S$. Each configuration in an HPPN carries the trace of events that occurred on the system until time k. A configuration $\delta_k \in \mathbb{M}_k^S$ is a token at time k whose value is a set of events b_k that occurred on the system until time k: $b_k = \{(v, \kappa) \| \kappa \leq k\}$.

Numerical places P^N represent continuous dynamics of the system and related uncertainty. Each numerical place $p^N \in P^N$ is associated with a set of dynamics equations $C_{p^N} \in \mathcal{C}$ modeling system continuous dynamics and its corresponding model noise and measurement noise:

$$C_{p^N} = \left\{ \begin{array}{rcl} x_{k+1} & = & \mathbf{f}(x_k, u_k) + \mathbf{v}(x_k, u_k) \\ y_k & = & \mathbf{h}(x_k, u_k) + \mathbf{w}(x_k, u_k) \end{array} \right. , \quad (1)$$

where $x_k \in X$ is the continuous state vector, $u_k \in \mathbb{R}^{n_u}$ is the vector of n_u continuous input variables, $\mathbf{f}$ is the noise-free continuous evolution function, $\mathbf{v}$ is a noise function, $y_k \in \mathbb{R}^{n_y}$ is the vector of n_y continuous output variables, $\mathbf{h}$ is the noise-free output function and $\mathbf{w}$ is the noise function associated with observation. Functions $\mathbf{f}$, $\mathbf{v}$, $\mathbf{h}$ and $\mathbf{w}$ depend on the considered place p^N. Numerical places are marked by numerical tokens called particles. The set of particles at time k is denoted $\mathbb{M}_k^N$. More precisely, a particle $\pi_k \in \mathbb{M}_k^N$ is a token whose value represents a possible continuous state $x_k \in X$ of the system at time k.

Degradation places P^D represent degradation dynamics of the system and related uncertainty. Each degradation place $p^D \in P^D$ is associated with a set of equations

$\mathtt{D}_{p^D} \in \mathcal{D}$ modeling system degradation dynamics:

$$\mathtt{D}_{p^D} = \mathtt{d}_{k+1} = \mathbf{g}(\mathtt{d}_k, b_k, x_k, u_k) + \mathbf{z}(\mathtt{d}_k, b_k, x_k, u_k), \quad (2)$$

where $\mathtt{d}_k \in D$ is the degradation state vector, b_k is the set of events that occurred on the system until time k, $\mathbf{g}$ is the noise-free degradation evolution function and $\mathbf{z}$ is a noise function. Functions $\mathbf{g}$ and $\mathbf{z}$ depend on the considered place p^D.

Degradation places are marked by degradation tokens. The set of degradation tokens at time k is denoted $\mathbb{M}_k^D$. A degradation token $d_k \in \mathbb{M}_k^D$ links a configuration δ_k to a particle π_k and its value is a possible degradation state $\mathtt{d}_k \in D$ of the system at time k.

In the HPPN formalism, health modes are represented by combinations of three places: a symbolic place, a numerical place representing the continuous dynamics of the mode and a degradation place, representing the degradation dynamics of the mode.

The marking $\mathbb{M}_k$ of an HPPN at time k is the distribution of tokens in the different places:

$$\mathbb{M}_k \quad = \quad \mathbb{M}_k^S \cup \mathbb{M}_k^N \cup \mathbb{M}_k^D, \quad (3)$$

where $\mathbb{M}_k^S \in (2^{\mathbb{M}_k^S})^{\mathrm{s}}$, $\mathbb{M}_k^N \in (2^{\mathbb{M}_k^N})^{\mathrm{n}}$ and $\mathbb{M}_k^D \in (2^{\mathbb{M}_k^D})^{\mathrm{h}}$ are respectively symbolic, numerical and degradation markings at time k.

Initial marking $\mathbb{M}_0$ represents the initial conditions of the system (the initial continuous and degradation states and the set of events that have occurred until time 0).

Firing rules in an HPPN are different depending on the utilization: model simulation, diagnosis or prognosis. A triplet of conditions $\Omega_t =< \omega_t^S, \omega_t^N, \omega_t^D >\in \Omega$ is associated with a transition $t \in T$. The condition ω_t^S can test the occurrence on an event in the value of a configuration δ_k. The numerical condition ω_t^N and the degradation condition ω_t^D are guards on the values of the continuous state x_k and on the degradation state $\mathtt{d}_k$. If one of these three conditions is not specified, it is set to the TRUE default value. We will introduce the firing rules based on these conditions for diagnosis and prognosis processes later in this article.

3 HPPN-based health monitoring methodology

Diagnosis aims at tracking the system's current health state from discrete and continuous observations on the system. Prognosis aims at predicting the system future states and its RUL/EOL from diagnosis and future inputs available from a mission scenario for example. During an arbitrary prediction horizon τ, the goal is then to determine if and when the system will enter a failure mode and will not be operational anymore.

Our HPPN-based health monitoring methodology uses the Hybrid Particle Petri Nets (HPPN) data structure to generate three different objects: a model, a diagnoser and a prognoser from a multimode representation of the hybrid system. These three objects are generated during the first phases of the health monitoring procedure, given in Algorithm 1.

Algorithm 1 HPPN-based health monitoring procedure

1: $HPPN_\Phi \leftarrow CreateHPPNModel()$
2: $HPPN_\Delta \leftarrow GenerateHPPNDiagnoser(HPPN_\Phi)$
3: $HPPN_\Pi \leftarrow GenerateHPPNPrognoser(HPPN_\Phi)$
4: **for all** k **do**
5: $\quad O_k \leftarrow (U_k^S, u_k^N, Y_k^S, y_k^N)$
6: $\quad \Delta_k \leftarrow Update(HPPN_\Delta, k, O_k)$
7: $\quad \Pi_k \leftarrow Prognose(HPPN_\Pi, \Delta_k, \mathbb{U}_k^+)$
8: **end for**

The first offline step (line 1) is the generation of the HPPN model $HPPN_\Phi$ of the system. It can be directly built from a multimode representation or created from expert knowledge. The second offline step (line 2) is the automatic generation of an HPPN-based diagnoser $HPPN_\Delta$ from the system model $HPPN_\Phi$. The last offline step (line 3) is the automatic generation of an HPPN-based prognoser $HPPN_\Pi$ from the system model $HPPN_\Phi$.

The online process (line 4-8) uses the system consecutive observations O_k (discrete and continuous inputs and outputs) to update the diagnoser marking [4]. This marking contains all diagnosis hypotheses. The diagnosis Δ_k is given by the marking of the HPPN-based diagnoser $HPPN_\Delta$ and represents a distribution of beliefs obtained by particle filtering. To compute the system prognosis Π_k at time k (line 7), the prognoser is initialized with Δ_k and its marking evolves according to a given set of future inputs $\mathbb{U}_k^+$. The set $\mathbb{U}_k^+ = \{\mathbb{U}_\kappa | \kappa \in \{k+1, ..., k+\tau\}\}$ includes the system future inputs during the prediction horizon $\tau \in \mathbb{N}$, where $\mathbb{U}_\kappa = (U_\kappa^S, u_\kappa^N)$ is the set including the discrete and continuous input vectors at future time κ. The prognosis Π_k is defined as the marking of the prognoser at the end of this process.

Figure 2 shows the different processes and what is included in HyMU implementation.

4 The HyMU Software

Figure 3 gives an overview of the HyMU architecture. All functions are coded in Python. On the left are represented the files created by the user. As said before, the HyMU software allows to simulate and to monitor the health state of a hybrid system from a multimode representation and a scenario. Some intermediate functions are represented in the middle of the figure to better understand the command lines given in the following sections to execute the two runner files `simulator.py` and `monitor.py`. The outputs of HyMU are diagnosis and prognosis results built thanks to the runner file `monitor.py`.

4.1 Model for HyMU

As illustrated by Figure 3, the first step for the user is to define the multimode representation of the hybrid system given in the file `model.py`.

The health modes of the system are described by filling `model.py` with:

- the set of identified discrete states P^S,
- the set of continuous dynamics $\mathcal{C}$,
- the set of degradation equations $\mathcal{D}$.

The transitions between modes are also described in `model.py` by defining the set of conditions Ω. This set contains triplets associated with transitions in T that may contain: one or several symbolic conditions and/or one or several numerical conditions and a degradation condition.

To define symbolic conditions on event occurrences, the event set E has to be characterized: the fault events to be diagnosed, the unobservable events and the observable events

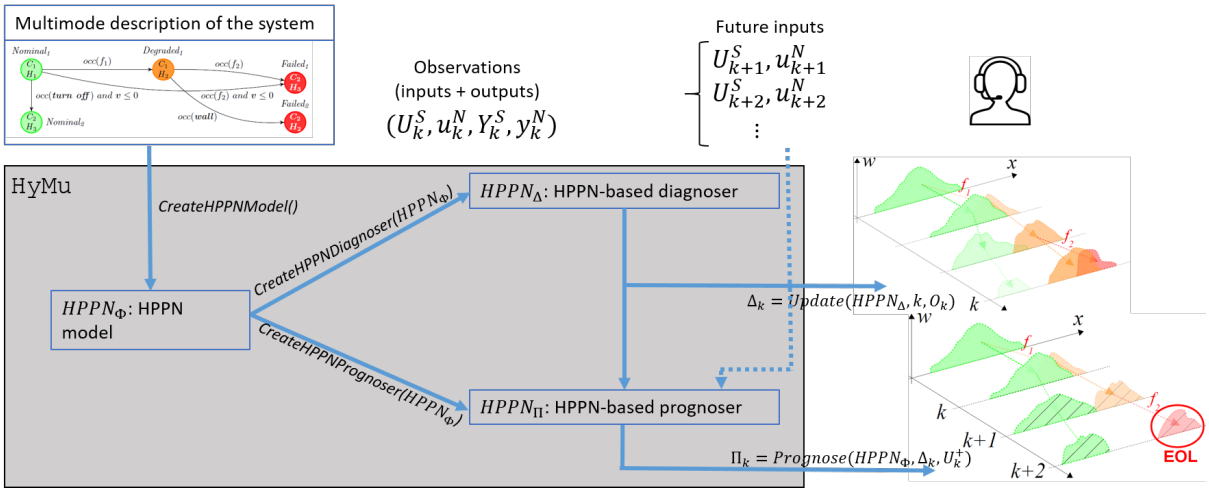

Figure 2: Overview of the health monitoring architecture.

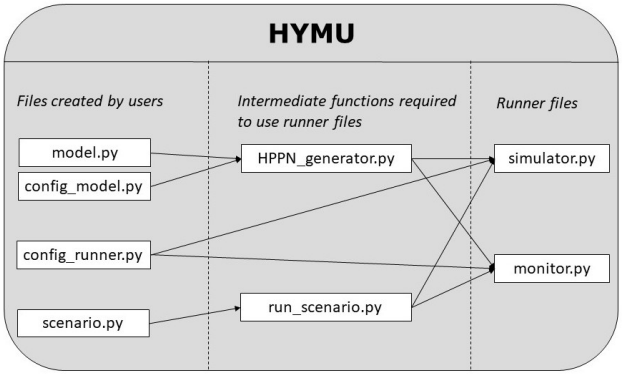

Figure 3: Overview of the HyMU Software.

have to be defined. Numerical and degradation conditions can be tests on the continuous and degradation states.

The file `config_model.py` may contain all model parameters and noise parameters that appear in Equations 1 and 2 and thresholds for fault probabilities if the degradation equations are given by probability functions. This content could be added in the model file, the model programming is actually very open and free to code.

The files `model.py` and `config_model.py` are called by `HPPN_generator.py` to create an implicit HPPN model. This HPPN model is used for simulating and for monitoring the health state of the system.

4.2 Simulate with HyMU

To simulate a hybrid model with HyMU, the second step for the user is to define a scenario as illustrated in Figure 3. A scenario has to contain a list of timed discrete events and timed continuous inputs. These inputs can be listed in a file or generated from a code specifying the observation number, sampling time, and the times of event occurrences.

As shown in Figure 4, all those information are written in a file `scenario.py` and must be converted into a table in the `csv` format by using the file `wrapper.py`.

A scenario is then simulated with the following command line:

```
» ipython run_scenario.py
Sources/simulator.py config_runner.py
model.py config_model.py
scenario_wrapped.csv
```

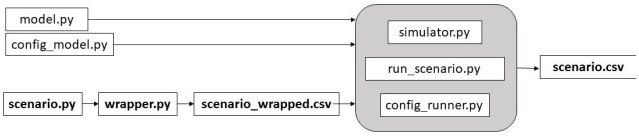

Figure 4: Simulating a scenario with HyMU.

The file `run_scenario.py` manages the execution of a runner object (simulator or monitor). The file `config_runner.py` contains the initial conditions of the model for the simulation corresponding to the initial marking $\mathbb{M}_0$ and commands of the HPPN model.

During the simulation, the values of tokens evolve according to continuous and degradation dynamics associated with the places and continuous and discrete inputs described in the scenario file.

The firing rule for the simulation part is defined as follows.

Rule 1 (Firing rule for simulation)**.** *A transition t is fired if its input symbolic, numerical and degradation places have tokens that satisfy the triplet of conditions $\Omega_t =< \omega_t^S, \omega_t^N, \omega_t^D >$. After the transition firing, the tokens that satisfied Ω_t are moved into the output places of t and evolve according to new dynamics.*

After a simulation, a repository with the name of the scenario is created. It contains the simulation results:

- a new file `scenario.csv` containing the inputs and generated system outputs,
- the plotted figures of the simulated health modes, the HPPN model and the multimode representation of the hybrid system.

4.3 Monitor with HyMU

The health state monitoring is performed with a diagnostic function and a prognostic function. Both functions are

called by the monitor. Their inputs are the files `model.py`, `config_model.py` and the scenario in the `csv` format that have already been described in previous sections.

The monitor is then executed with the following command line:

```
» ipython run_scenario.py
Sources/monitor.py config_runner.py
model.py config_model.py scenario.csv
```

The monitoring results for a given scenario are written in a repository with the name of the scenario.

Uncertainty and computational performance

Uncertainty and computational performance are two substantial challenges for diagnostics and prognostics that cannot be ignored by HyMU.

The first one is related to the result accuracy and precision, whereas the second one is related to calculation time and computational resource management. In order to cope with these two challenges, we developed and implemented the Stochastic Scaling Algorithm (SSA), that has been described in [5]. It basically determines the number of tokens in the HPPN during the system monitoring phase, knowing six parameters: ρ_{Δ}^{max} (resp. ρ_{Π}^{max}) and ρ_{Δ}^{min} (resp. ρ_{Π}^{min}), that are respectively the maximum and the minimum number of tokens of each type to represent a diagnostic (resp. prognostic) result and ρ_{tot} the total number of tokens of each type to monitor the system for diagnosis (resp. prognosis).

The user has to write the chosen values of the six scale parameters of the SSA algorithm in the file `config_runner.py`.

Diagnose with HyMU

An HPPN-based diagnoser is built from the generated HPPN model [6]. Its online process takes as inputs the set of discrete and continuous observations described in the file `scenario.csv` to monitor both the system behavior and degradation under uncertainty. We recall in this section the main ideas for the diagnosis online process, focused on the uncertainty management. The diagnoser generation and its online process are described in detail in [1].

From the initial marking $\mathbb{M}_0$ and the initial commands described in the file `config_model.py`, the HPPN-based diagnoser marking $\mathbb{M}_k$ evolves at time k according to the observations O_k and dynamics associated with numerical and degradation places. The marking evolution in the HPPN-based diagnoser is based on two steps, prediction and correction, which combine the transition pseudo-firing, particle filters and the SSA algorithm.

1- Prediction step During the diagnosis prediction step, the values of the particles and degradation tokens evolve as functions of the continuous and degradation dynamics associated with the numerical and degradation places to which the particles belong. Numerical uncertainty about imprecision on the continuous part of the model and numerical values is so dealt with particle filters to estimate the continuous state of the HPPN-based diagnoser according to model noise **v** and measurement noise **w** (see Equation 1).

The firing rule associated with the diagnosis part is the following.

Rule 2 (Firing rule for diagnosis process)**.** *A transition t is fired if the symbolic condition ω_t^S and the numerical condition ω_t^N are satisfied by the configuration δ and the particle π* **or** *if the degradation condition ω_t^D is satisfied by the degradation token d. After the transition firing, the values of tokens are updated as follows: the configuration δ is updated with the label of the event that occurred, the value x of a particle π and the value* `d` *of a degradation token are updated according to the continuous and the degradation dynamics associated with the numerical and degradation places in which π and d belong after the transition firing.*

Symbolic uncertainty corresponding to the discrete part of the model and observations (missing observation or false observation for example) is managed during the diagnosis prediction step by using pseudo-firing of transitions [7]. Transition pseudo-firing duplicates tokens: tokens in the input places of a fired transition are not moved but duplicated, and their duplicates are moved in the output places of the transition. Pseudo-firing creates new hypotheses on the system health modes.

2- Correction step The diagnosis correction step updates the predicted marking based on the computation of scores and on resampling. More details about the score computation can be found in [1]. The resampling step uses the SSA algorithm to adapt the number of particles in the HHPN-based diagnoser. It avoids the combinatory explosion by limiting the number of tokens at each step of the diagnosis algorithm.

Output of the diagnoser process The output of the diagnoser process at any time k is an estimation of the system health state that takes the form of the marking of the HPPN-based diagnoser $\Delta_k = \hat{\mathbb{M}}_k = \{\hat{\mathbb{M}}_k^S; \hat{\mathbb{M}}_k^N; \hat{\mathbb{M}}_k^D\}$, where $\hat{\mathbb{M}}_k$ represents all the possible diagnosis hypotheses on the system mode at time k as a distribution of beliefs over the current health mode and how this mode has been reached. In other words, the marking $\hat{\mathbb{M}}_k$ indicates the belief over the continuous state, the fault occurrences and the degradation state.

Prognose with HyMU

An HPPN-based prognoser is built from the generated HPPN model. The prognosis process predicts the system future states by simulating the future evolution of the diagnosis hypotheses from future inputs specified in the file `scenario.csv`. It also predicts events that will occur on the system. Particularly, from a hypothesis in Δ_k, it predicts the possible sets of events that would lead the system to failure modes. Mode switches are simulated when some conditions on the continuous state are satisfied. The time of occurrence of an event that leads to a failure mode is a possible EOL of this hypothesis. We recall in this section the main ideas for the prognosis online process. The prognoser generation and its online process are described in detail in [5].

Initialization The HPPN-based prognoser is initialized with diagnosis hypotheses in Δ_k. Without performance constraints, a diagnosis hypothesis is completely reproduced in the prognoser. To improve computational performance of the prognosis process, however, the hypotheses in Δ_k can be partially reproduced. The SSA algorithm is then used to select which hypotheses to set as initial condition for prognosis based on their belief degrees and the precision with which they will be simulated. By choosing the three scaling parameters ρ_{Π}^{min}, ρ_{Π}^{max}, and ρ_{Π}^{tot} in `config_runner.py`, the user is able to control the prognoser performance.

Prognoser online process and output The marking of the HPPN-based prognoser evolves according to future inputs by using the same firing rules as in diagnosis process (see Rule 2).

The output of the prognoser process is the marking of the HPPN-based prognoser: $\Pi_k = \hat{\mathbb{M}}_{k_{EOP}}$, where k_{EOP} is the End Of Prediction. The prognosis Π_k is a distribution of beliefs over the system future modes until time k_{EOP}. It especially contains the event occurrences that will lead to failure modes, and particularly the faults and their times of occurrences. It thus contains all necessary data to compute a belief distribution over the system RUL/EOL.

5 Application on a Water Tanks example

5.1 Water-tank benchmark

The benchmark used to illustrate the use of HyMU is a three-tank system described in Figure 5.

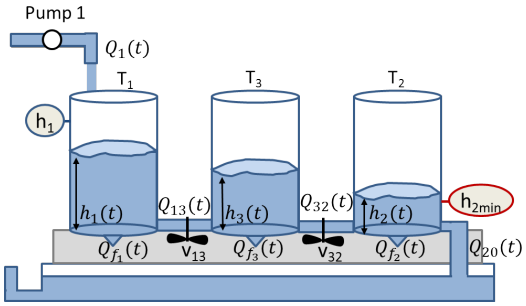

Figure 5: Three-tank system description.

The tanks are configured in a series circuit. The water comes from Pump 1 into tank T_1. Flow $Q_1(t)$ delivered by the pump is supposed to be constant. Tank T_2 empties with flow $Q_{20}(t)$. The available measurements at time t are the water level in tank T_1 denoted $h_1(t)$ and the total mass of water in the three-tank system denoted $W(t)$. Valves v_{13} and v_{32} allow the water to flow between tanks. They are controlled by discrete control inputs $open_{v_{13}}$, $close_{v_{13}}$, $open_{v_{32}}$ and $close_{v_{32}}$. Fault events f_1, f_2 and f_3 model leaks that may occur in tanks T_1, T_2 and T_3. The leak f_1 is three times bigger than f_2 and f_3. f_4 and f_5 are fault events that may occur on the valves. The valve v_{13} (resp. v_{32}) may become stuck closed: it is represented by the occurrence of fault f_4 (resp. f_5). The goal of the system is to maintain the water level in tank T_2 greater than a minimum value $h_{2_{min}}$. The leak in tank T_1 is considered too large and therefore leads to the system failure when f_1 occurs because the system is not able to achieve the goal anymore. Leaks in tanks T_2 and T_3 are not large enough to prevent the goal achievement on their own, but the occurrence of the two leaks leads to the system failure. Both f_4 and f_5 prevent the water delivered by the pump to reach the tank T_2 and then lead to the system failures. Therefore, it is supposed that the system enters a failure mode if either f_1, f_4 or f_5 occurs, or if f_2 and f_3 occur.

In order to ease the understanding and analysis of the results, we will only consider the events on valve v_{13} and faults f_1 and f_4 in this example. The three-tank system is then composed of ten different functioning modes. The initial mode is Nom_1, in which both valves are open. The associated continuous dynamics is C_1. In this mode, the tanks have a weak stress level (degradation dynamics H_1).

When the command $close_{v_{13}}$ occurs, the system enters mode Nom_2 in which v_{13} is closed (continuous dynamics C_2). In this mode, the stress level of tanks T_2 and T_3 are weak, but the one of T_1 is important (degradation dynamics H_2). The system can go back to Nom_1 when $open_{v_{13}}$ occurs.

From Nom_1 (resp. Nom_2), the system can switch to degraded mode Deg_2 (resp. Deg_3) if fault f_1 occurs. In this mode, continuous dynamics C_3 (resp. C_4) models the water levels in the tanks with T_1 leaking while valve v_13 is open (resp. closed). Both Deg_2 and Deg_3 have the same degradation dynamics H_4 in which stress levels in T_2 and T_3 are weak, and the degradation state associated with fault f_1 is no longer evaluated, as f_1 already occurred.

From Nom_1 or Nom_2, the valve v_{13} may become stuck closed (f_4): the system will enter degraded mode Deg_1, in which the continuous dynamics are the same as if the valve was closed voluntarily (C_2). In this mode, stress levels of tanks T_2 and T_3 are weak, the one of T_1 is important, and the degradation state associated with fault f_4 is no longer estimated, as f_4 already occurred (degradation dynamics H_3).

Through the same fault occurrence, modes Deg_2 and Deg_3 can lead to mode Deg_4 with continuous dynamics C_4 and degradation dynamics H_5 in which stress levels for T_2 and T_3 are weak. The degradation state associated with f_1 and f_4 is no longer estimated, as they both occurred.

Finally, from each of the degraded modes, the system may enter a failure mode if the water level in tank T_2 becomes smaller than the minimum value $h_{2_{min}}$. It is represented by a fault f_0, which is nothing but a failure indicator.

The equations corresponding to continuous and degradation dynamics can be found in the file `small_three_tanks.py`. In this water-tank example, degradation dynamics are represented by probability functions associated with each anticipated faults in the system. These probability functions depend on several qualitative stress levels (weak/high stress) identified for different system configurations (open/closed valves for example). The multimode representation of the system plotted by HyMU from the file `small_three_tanks.py` is given in Figure 6. The implicite HPPN model contains 19 places (10 symbolic places, 4 numerical places and 5 degradation places). Table 1 shows the numerical and degradation places and their dynamics. As explained in Section 2.2, each mode is represented by a symbolic place, a numerical place and a degradation place. For example, mode Nom_1 is represented by places p_1 (symbolic place), p_2 (numerical place) and p_3 (degradation place), associated with dynamics C_1 and H_1.

p_i^N	$C_{p_i^N}$	p_i^D	$\mathtt{D}_{p_i^D}$
p_2	C_1	p_3	H_1
p_5	C_2	p_6	H_2
p_{10}	C_3	p_8	H_3
p_{13}	C_4	p_{11}	H_4
		p_{15}	H_5

Table 1: Correspondence between places and dynamics for the three-tank system.

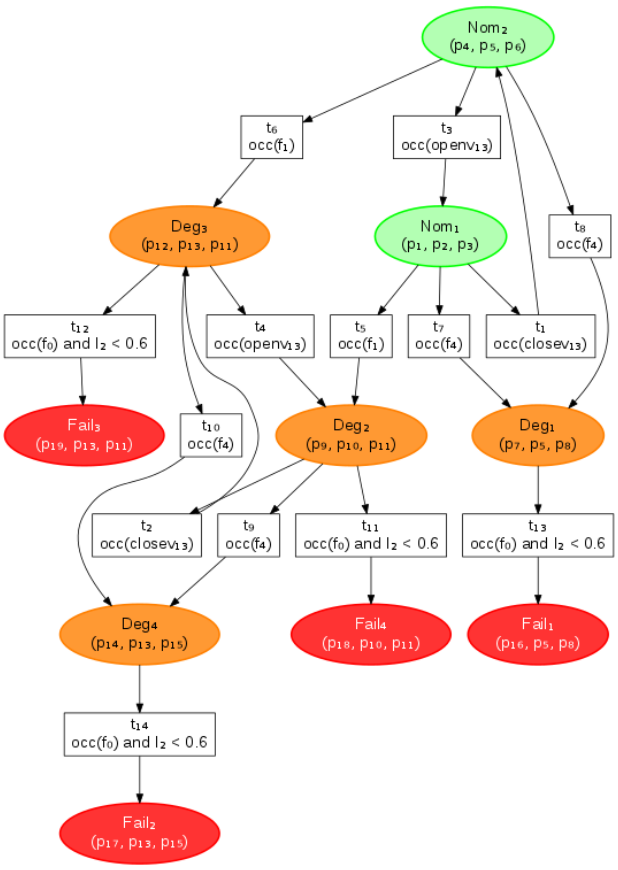

Figure 6: Multimode representation of the three-tank system

5.2 Simulation

As explained in Section 4.2, the simulation process is started with the following command line:

```
» ipython   run_scenario.py
Sources/simulator.py config_runner.py
water_tanks/small_three_tanks.py
water_tanks/config_model.py
water_tanks/scenario_1_wrapped.csv
```

In the simulation scenario, the continuous input flow $Q_1(t)$ is constant and equal to 3.5×10^{-5}. In this example, we took 3400 samples, one every 60 sec. The discrete inputs $closev_{13}$ and $openv_{13}$ are simulated as follows: starting from 310 min, the valve v_{13} is closed every hour, and re-opened 20 min after. This scenario is coded in the file `scenario_1.py` and converted by the file `wrapper.py` into the file `scenario_1_wrapped.csv`.

The simulation result is illustrated in Figure 7. As expected from the simulation scenario, regular switches from the mode Nom_1 to the mode Nom_2 can be observed starting from around 18000s. After approximately 190 000s, the tank T_1 begins to leak, and finally the water level in T_2 becomes smaller than the value $h_{2_{min}}$, resulting in a failure of the system. After this fault occurred, the system switches to mode Deg_3. The discrete inputs still going on, the system enters Mode Deg_2 when $open_{v13}$ occurs. Then, the failure appears, right after the switch from Deg_2 to Deg_3, so the system enters the failure mode $Fail3$, from Deg_3.

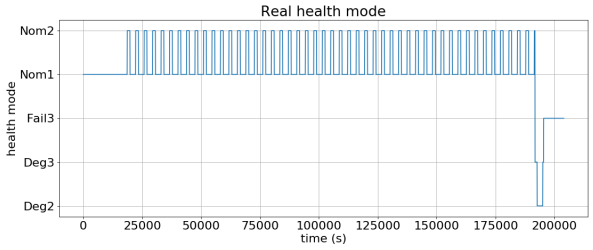

Figure 7: Three-tank system simulated health modes.

5.3 Monitoring

Diagnosis Results

Figure 8 shows the diagnosis results applied to our system. As the diagnoser takes the uncertainty into account, it associates each mode to a given probability. This probability is represented by the line width in Figure 8: the higher the probability, the wider the line. The highest belief is colored in blue. We can notice that up to 190 000s, the hypothesis having the highest belief matches with the simulation of the system. After that, the fault happened and is successfully detected even if some ambiguity remains between modes $Fail3$ and $Fail2$ at the end because they have similar dynamics. Indeed, the occurrence of a tank leak f_1 or a stuck valve f_4 have the same effect on the water level h_2.

Modes Deg_1, Deg_2 and Nom_2 all have high belief before the fault's occurrence. This can be explained by the fact that the three-tank system has a slow continuous behavior that does not allow the diagnoser to identify the relevant continuous dynamics. This belief is also represented in the prognosis results, as we will see in the upcoming section.

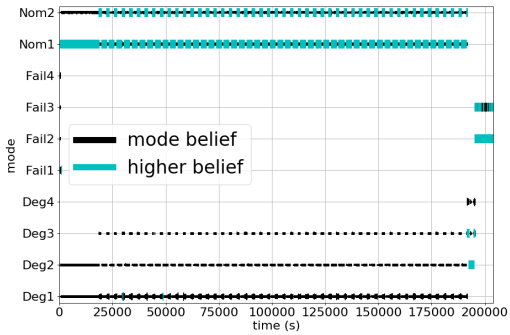

Figure 8: Three-tank system mode belief.

Prognosis Results

Figure 9 illustrates the estimated RUL for the three-tank system. The estimated EOL is around 190000 seconds after the beginning of the estimation. Different lines are plotted in Figure 9, the line in black represents the prognosis result with the higher belief. The other lines can be explained with the fact that there is a possibility that the system starts in a degraded mode which would lead to a faster failure. As it was mentioned and seen in the diagnoser results, this possibility is likely, but less than the system starting in nominal mode. However, we can notice that both lines merge and lead to the same EOL date.

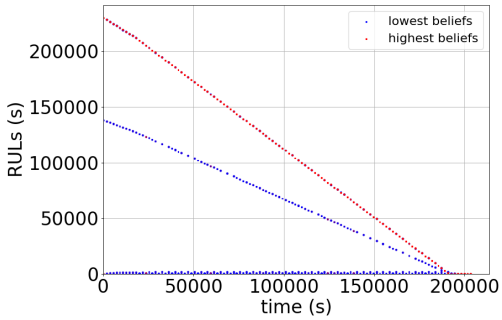

Figure 9: Three-tank system RUL.

6 Other existing tools

We discuss in this section of other softwares that have been designed to perform diagnosis and prognosis on hybrid systems.

HyDiag [8] and its extensions HyDiagPro and ActHyDiag, have been developed in Matlab by the DISCO team at LAAS-CNRS since 2009. It is designed to simulate, diagnose and prognose hybrid systems using model-based techniques. The system behavior is described by a hybrid automaton. The model, the diagnosis and the prognosis do not take into account any uncertainty.

HyDe (Hybrid Diagnosis Engine) [9] is a software developed by NASA. It is a general framework for stochastic and hybrid model-based diagnosis and offers flexibility to the diagnosis application designer. However, HyDe does not provide any information about the degradation of the system and is not used for prognosis purpose.

QED (Qualitative Event-based Diagnosis) [10] is a software that makes detection, isolation, and identification of faults. It is based on qualitative, event-based reasoning. An open-source software implementation was in development in 2017. As HyDe, it does not provide any information about the degradation of the system and is not used for prognosis purpose. A probabilistic version of QED has been proposed in [11] for robust fault isolation.

7 Conclusion and Future Work

HyMU is a software in Python distributed under LGPL license and available on github with a tutorial. The user has to provide a multimode representation of the hybrid system and a scenario. HyMU simulates the system behavior and is able to follow its evolution knowing the available set of observations (inputs and outputs) and to provide at each time diagnosis and prognosis results. The main feature of the software is that it takes into account multiple sources of uncertainty during the health monitoring process, which makes the results closer to reality.

In future works, we want to give the possibility to HyMU to manage heterogeneous systems, as it is focused on hybrid systems right now. Heterogeneous systems are systems that can have discrete parts, continuous parts, or hybrid parts. These parts can intertwine, and the tool must be able to switch from one to another easily. Moreover, as modifications of the HPPN formalism are currently taking place, HyMU will also be undergoing some modifications to match with the newly established formalism. An example of these modifications is that the three places of an HPPN will be merged into an unique place containing all the dynamics equation sets.

Acknowledgments

First, we want to thank Q. Gaudel who was the father of the first version of HyMU, for all his theoretical and implementation works. We also thank A. Agourram, O. Jorge and S. Mouline, 3^{rd} year students from INP-ENSEEIHT, Toulouse, who were the first beta testers for HyMU and wrote the first tutorial.

References

[1] Q. Gaudel, E. Chanthery, P. Ribot, and M.J. Daigle. Diagnosis of hybrid systems using HPPN: theory and application on a planetary rover. In M. Sayed-Mouchaweh, editor, *Fault Diagnosis of Hybrid Dynamic and Complex Systems*, pages 209–241. Springer Verlag, 2018.

[2] M. Bayoudh, L. Travé-Massuyes, and X. Olive. Hybrid systems diagnosis by coupling continuous and discrete event techniques. *IFAC Proceedings Volumes*, 41(2):7265–7270, 2008.

[3] T.A. Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer, 2000.

[4] Quentin Gaudel, Pauline Ribot, Elodie Chanthery, and Matthew J. Daigle. Health Monitoring of a Planetary Rover Using Hybrid Particle Petri Nets. In *37^{th} International Conference on Application and Theory of Petri Nets and Concurrency*, Poland, 2016.

[5] P. Ribot, E. Chanthery, and Q. Gaudel. HPPN-based prognosis for hybrid systems. *Annual Conf. of the PHM Society*, 2017.

[6] Q. Gaudel, E. Chanthery, and P. Ribot. Health Monitoring of Hybrid Systems Using Hybrid Particle Petri Nets. In *Annual Conf. of the PHM Society*, USA, 2014.

[7] L. Zouaghi, A. Alexopoulos, A. Wagner, and E. Badreddin. Modified particle Petri Nets for hybrid dynamical systems monitoring under environmental uncertainties. In *IEEE/SICE International Symposium on System Integration*, pages 497–502, 2011.

[8] E. Chanthery, Y. Pencolé, P. Ribot, and L. Travé-Massuyès. Hydiag : extended diagnosis and prognosis for hybrid systems. In *DX 2015*, France, 2015.

[9] S. Narasimhan and L. Browston. HyDE - a general framework for stochastic and hybrid model-based diagnosis. In *DX 2007*, pages 162–169, 2007.

[10] M.J. Daigle, I. Roychoudhury, and A. Bregon. Qualitative event-based diagnosis applied to a spacecraft electrical power distribution system. *Control Engineering Practice*, 38:75–91, 2015.

[11] M.J. Daigle, I. Roychoudhury, and A. Bregon. Qualitative event-based fault isolation under uncertain observations. Technical report, NASA Ames Research Center Moffett Field United States, 2014.