Hybrid Model Learning for System Health Monitoring

Amaury Vignolles, Elodie Chanthery, Pauline Ribot

Hybrid Model Learning for System Health Monitoring

Amaury Vignolles * **Elodie Chanthery** * **Pauline Ribot** *

* *CNRS, LAAS, UPS, INSA 7 avenue du colonel Roche, 135 Av. de Rangueil, 31400 Toulouse, France (e-mail: firstname.name@laas.fr).*

Abstract: Health monitoring approaches are usually either model-based or data-based. This article aims at using available data to learn a hybrid model to profit from both the data-based and model-based advantages. The hybrid model is represented under the Heterogeneous Petri Net formalism. The learning method is composed of two steps: the learning of the Discrete Event System (DES) structure using a clustering algorithm (DyClee) and the learning of the continuous system dynamics using two regression algorithms (Support Vector Regression or Random Forest Regression). The method is illustrated with an academic example.

Keywords: Hybrid and switched systems modeling, Model Learning, Health Monitoring, Clustering, Regression

1. INTRODUCTION

Health monitoring consists in estimating and predicting the health state of a system. It is linked to either the degradation of the system or the occurrence of a fault. A fault is a non-acceptable deviation from a characteristic property or a system parameter. System health monitoring is performed by integrating diagnosis and prognosis capabilities. Diagnosis aims at detecting a fault occurrence and precisely locating it in the system. Prognosis aims at predicting fault occurrences in order to determine the remaining useful life of the system before failure. It is possible to distinguish two main health monitoring approaches: model-based and data-based approaches. While model-based approaches are very efficient if the system model is accurate, they perform less when the model is not precise enough. On the other hand, data-based methods are helpful when the model is unavailable, but they lack explainability or expert knowledge. The solution is to take advantage of the two types of methods. In this paper, we aim at using a model-based health monitoring approach by learning a model thanks to data. The studied systems are complex and deal both with continuous and discrete data. Their behavior can be represented by a multimode representation by using, for example, hybrid automaton Henzinger (2000). For health monitoring purposes, a functioning mode of the system is the combination of one discrete state with continuous and degradation dynamics. This multimode representation can be transformed into a Heterogeneous Petri Nets (HtPN) model. The HtPN formalism is an interesting extension of the Petri Net formalism, where the places represent the different functioning modes with associated continuous and degradation dynamics. This formalism is well suited for the hybrid system health monitoring under uncertainty because of its capability to represent the evolution of different hypotheses on the health state thanks to parallelism. However, the learning process described in this article could be adapted and applied to other formalisms for hybrid systems, like hybrid automata.

This paper proposes a methodology to learn a hybrid model for system health monitoring using the formalism of HtPN. The methodology's first step consists of obtaining the Petri Net structure, represented by places and transitions, using a clustering method. Hence, for each of the modes, regression algorithms such as Support Vector Regression (SVR) and Random Forest Regression (RFR) are applied to learn the continuous dynamics of the system.

The paper is organized as follows. Section 2 gives a state of the art of existing methods used to learn models of different types (hybrid, discrete or continuous). Section 3 presents the HtPN formalism used for health monitoring purposes. Section 4 presents the proposed learning methodology to obtain a hybrid model under the HtPN formalism. Section 5 illustrates the application of the methodology and gives results for an academic example, the water tanks. Finally, Section 6 presents some conclusions and perspectives.

2. RELATED WORK

This section briefly presents learning model solutions for hybrid systems.

The first solution to learn hybrid models is to merge techniques used to learn discrete event models and techniques used to learn continuous dynamics. This idea has been applied in Niggemann et al. (2012), where hybrid automata are learned. The first step is to learn a prefix tree acceptor based on the sequence of observations: each observable sequence leads to a path from the tree's root to a leaf. Then, for each node, the continuous behavior is learned. Finally, the nodes with similar continuous behavior are merged to reduce the size of the hybrid automata. The main drawback of this approach is that it requires the events to be observable to have a correct prefix tree acceptor. Event-based methods are primarily used to learn the structure

of a Discrete Event System (DES). An example to learn a Petri Net (PN) can be found in Dotoli et al. (2008). Using the observations of the events and the available output vectors, a PN is obtained through an integer linear programming problem or an identification algorithm, depending on whether the sets of places and transitions are known beforehand. In Moreira and Lesage (2019), a Deterministic Automaton with Outputs and Conditional Transitions (DAOCT) is computed using observed fault-free paths. Another method to learn Petri Nets, introduced in Leclercq et al. (2008), is based on Neural Networks. However, these learning methods fit only for DES, where all data are discrete, but not for hybrid systems where continuous conditions may change health states.
With a clustering method, both continuous dynamics variations and discrete events can impact the creation of the DES structure. However, not all methods are usable in our case. For instance, clustering methods requiring a known number of outputted clusters, such as K-means, cannot be used. Even if the size of clusters has to be decided, a fixed number of clusters would be limiting, where size could improve the knowledge of the system. To meet this need, the DyClee algorithm, presented in Roa et al. (2019), has been proposed for tracking evolving environments with dynamic clustering. It is a two stage clustering algorithm, one based on distance and one on density. First, each sample is assigned to a micro cluster in the distance-based stage following the L1norm. Then, each micro cluster is assigned a low, medium or high density. This density is used to output the final clusters. The final clusters comprise connected high density micro clusters for the inside and either medium or high density micro clusters for the border. The low density micro clusters may be considered as outliers. From these clustering results, it has been shown in Barbosa et al. (2017) that a Timed Automaton could be retrieved to model the system, using a parallel between the clusters created and the states of an automaton. Based on this knowledge and the availability of experts to further improve the algorithm, the DyClee algorithm was chosen as a base to learn the DES structure of our hybrid model.

Concerning learning continuous dynamics, the gradient descent method is probably one of the most used methods to learn parameters to fit a set of data. The work of Tamssaouet et al. (2020) is based on an Inoperability Input-Output model, which measures the distance between the current state of the system and the failure state and aims at prognosing a system. Gradient descent is used to tune the model parameters by comparing the inoperability estimated by the model and the one measured. However, the gradient descent method presents one major problem: to tune the parameters of the function, the function by itself, or at least its shape, must be known, which is not always our case. Hence, the gradient descent method is not suitable for our needs.
Neural networks can be used to predict output to an unknown or complex function, as in Djedidi and Djeziri (2020), where a Nonlinear Autoregressive Network with eXogenous inputs (NARX) model measuring the power consumption of a phone is learned. However, Neural networks have a high need for hyper-parameter tuning, which is a significant drawback.
Support Vector Regression (SVR), detailed in Awad and Khanna (2015), is often used to learn continuous dynamics, as, for example, in Drezet and Harrison (1998) or Zhang and Xi (2004). The SVR algorithm tries to find a function $w_i x_i$ such that the highest number of data points are contained in the interval $[w_i x_i - \epsilon, w_i x_i + \epsilon]$, with w_i the model's coefficient, x_i the features of the data, and $\epsilon \in \mathbf{R}$ the desired error. The main advantage of SVR is that it has excellent generalization capability and does not need a complex tuning phase. Another solution is to use Random Forest Regression (RFR) as in Liaw et al. (2002). It consists in averaging the prediction of decision trees. The data are randomly distributed among various decisions trees, which each gives a prediction. All these predictions are then averaged and the obtained value is outputted. RFR's main advantages are that it is robust against overfitting and quite able to generalize. Moreover, RFR only has two hyper-parameters to set and the obtained results are not that sensitive to them.
In Van Overschee and De Moor (1994) two numerical algorithms for subspace state space system identification (N4SID) to identify combined deterministic stochastic linear systems are introduced. The first algorithm calculates unbiased results, while the second one trades off the precision of the results for a lower computational complexity as it is a simpler biased approximation. This method could be applied to learn continuous dynamics. However, a less complex option has been chosen for now.

The second solution to learn hybrid models is to apply techniques suited to hybrid models directly. In Lauer and Bloch (2008), switched and non linear hybrid systems are learned using the Support Vector Machine (SVM) and Support Vector Regression (SVR) frameworks. Both of these techniques are based upon kernels. A new regularized technique for identifying piecewise affine systems is proposed in Pillonetto (2016). This technique, named Hybrid Stable Spline algorithm (HSS) uses the stable spline kernel to model the impulse responses of submodels as zero-mean Gaussian processes. Feng et al. (2010) focus on identifying discrete time affine hybrid systems with measurement noise. The proposed approach is based on recasting the problem into a polynomial optimization form and exploiting its inherent sparse structure to obtain computationally tractable problems. In Vidal (2004), PieceWise Auto Regressive eXogenous (PWARX) models are identified using an algebraic geometric solution representing the number of states n as the degree of a polynomial p and encoding the orders and the model parameters as factors of p.
These learning methods could be used to learn a hybrid system. However, a transformation of obtained equations into a DES structure would be required to apply our health monitoring method based on the HtPN formalism.

Looking at the related work, we propose a new methodology mixing DES structure and continuous dynamics learning techniques to obtain hybrid system models. As underlined in the DES structure learning part, the DyClee algorithm best suits our needs as it does not need to know the number of clusters before learning and fits dynamic processes. As for continuous dynamics, both SVR and RFR methods will be implemented and tested because they both have generalization capability and do not need a complex tuning phase.

3. HETEROGENEOUS PETRI NETS FORMALISM

This section describes the Heterogeneous Petri Nets (HtPN) formalism that can be used for health monitoring purposes. This formalism is extended from the Petri Nets for their ability to represent parallelism. When applying diagnosis and prognosis algorithms for system health monitoring, following multiple tokens is necessary.

A HtPN is formally defined as a tuple $HtPN =< P, T, A, Guard, Jump, E, X, \Gamma, C, D, \mathbb{M}_0 >$ where:

- P is the set of places;
- T is the set of transitions;
- $A \subset (P \times T \cup T \times P)$ is the set of arcs;
- $Guard$ is the set of conditions associated with the incoming arcs of transitions;
- $Jump$ is the set of assignments associated with the outgoing arcs of transitions;
- E is the set of event labels: $E = E_o \cup E_{uo}$, where E_o is the set of observable event labels and E_{uo} is the set of unobservable event labels;
- $X \subset \mathbb{R}^{n_N}$ is the state space of the continuous state vector, where $n_N \in \mathbb{N}_+$ is the finite number of continuous state variables;
- $\Gamma \subset \mathbb{R}^{n_D}$ is the state space of the degradation state vector, where $n_D \in \mathbb{N}_+$ is the finite number of degradation state variables;
- C is the set of continuous system dynamics;
- D is the set of degradation system dynamics;
- $\mathbb{M}_0$ is the initial marking of the network.

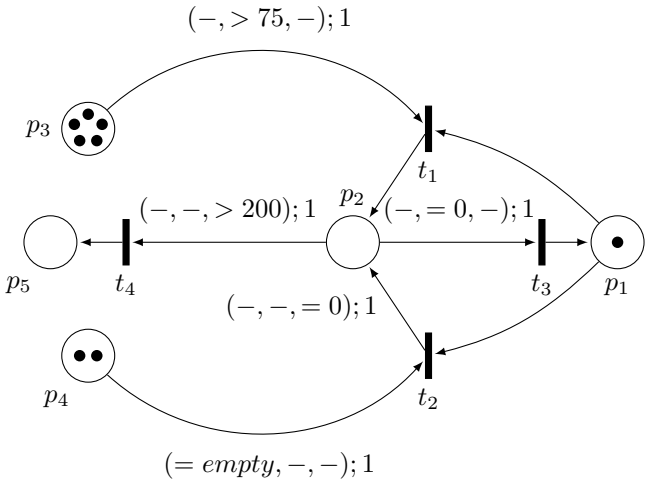
The main subsets of a HtPN are briefly summarized hereafter. An example of a HtPN is presented in Figure 1(a). The net represents the behavior of a machine that empties bottles stored in two different locations, either because the quantity of liquid in a bottle reached a given value or because the user asked for the emptying. The machine degrades with time and can enter a faulty state if it reaches a given degradation value. The behavior of this machine depends on both discrete and continuous information. The place p_1 represents the availability of the machine to empty the bottles. The place p_2 is marked when the machine is busy. Tokens in p_3 and p_4 represent bottles that are automatically filled up. A bottle in p_3 is emptied when it reaches a given quantity of liquid. A bottle in p_4 is emptied when the discrete *empty* command occurs. When the machine is working, if its degradation exceeds 200 units, the system enters a failure state represented by p_5.

3.1 Places and tokens

A place $p \in P$ in a HtPN is associated with a set of equations $C_p \in C$ modeling the system continuous dynamics and the associated noise (uncertainties on the evolution and the measurements) as well as a set of equations $D_p \in D$ modeling the system degradation dynamics:

$$p = \begin{Bmatrix} C_p \\ D_p \end{Bmatrix} \quad (1)$$

The continuous dynamics C_p represents the evolution of the continuous state vector x when the system is in place p. The system degradation dynamics D_p represents the evolution of the degradation state vector γ when the system is in the place p.



(a)

Places	Continuous dynamics	Degradation dynamics
p_1	-	-
p_2	$x_{k+1} = x_k - 1$	$\gamma_{k+1} = \gamma_k + 3$
p_3	$x_{k+1} = x_k + 3$	-
p_4	$x_{k+1} = x_k + 2$	-
p_5	-	-

(b)

Fig. 1. Example of a HtPN (a) and dynamics associated with its places (b)

Equations associated with the places of the HtPN in Figure 1(a) are presented in Figure 1(b). The symbol − means that no dynamic is associated with the place. For example, places p_1 and p_5 have neither continuous nor degradation dynamics. Places p3 and p4 only have continuous dynamics, representing a bottle's volume evolution. Place p_2, on the other hand, has both continuous and degradation dynamics: they represent the evolution of the volume in a bottle and the evolution of the degradation of the machine.

A place p contains tokens with three attributes: $< \delta_k^h, \pi_k^h, \phi_k^h >$, representing respectively discrete, continuous and degradation information for a token h at time k. These attributes evolve according to discrete events and dynamics associated with the place the token h belongs to.

Initial marking $\mathbb{M}_0$ of the HtPN is the distribution of tokens in the different places representing the nitial system conditions. Each token carries its initial discrete information (the set of events that have already occurred), its initial continuous information and its initial degradation information. For example, each token in p_3 and p_4 represents a bottle at the initialization of the process.

3.2 Arcs

The arcs of a HtPN are divided into two sets: $A = A_{\bullet t} \cup A_{t\bullet}$, where $A_{\bullet t}$ contains all the incoming arcs and $A_{t\bullet}$ contains all the outgoing arcs of transitions.

An incoming arc $a_{p,t} \in A_{\bullet t}$ going from place $p \in P$ to transition $t \in T$ is associated to a set $\Omega_{p,t} \in Guard$:

$$\Omega_{p,t} = \left\{ (\Omega_{p,t}^S, \Omega_{p,t}^N, \Omega_{p,t}^D); \rho_{p,t} \right\} \quad (2)$$

where $(\Omega^S_{p,t}, \Omega^N_{p,t}, \Omega^D_{p,t})$ represents respectively a discrete, a continuous and a degradation condition, and $\rho_{p,t} \in \mathbb{N}^+$ is the weight of the arc. For example, in Figure 1(a), the arc $a_{(p_3,t_1)}$ is associated to the set $\Omega_{p_3,t_1} = \{(\Omega^S_{p_3,t_1}, \Omega^N_{p_3,t_1}, \Omega^D_{p_3,t_1}); \rho_{p_3,t_1}\}$, with:

- $\Omega^S_{p_3,t_1} = -$, which means that no discrete condition is defined,
- $\Omega^N_{p_3,t_1} => 75$, which means that the continuous state of the token in p_3 has to be greater than 75,
- $\Omega^D_{p_3,t_1} = -$, which means that no degradation condition is defined,
- $\rho_{p_3,t_1} = 1$, which means that only one token will be consumed during the transition firing.

A transition can be fired if the set of conditions $\Omega_{p,t}$ is satisfied for each of its input arcs. If the conditions for arcs a_{p_3,t_1} and a_{p_1,t_1} are satisfied the transition t_1 can be fired by consuming one token in p_3 and one token in p_1.

An outgoing arc $a_{t,p} \in A_{t\bullet}$ going from a transition $t \in T$ to a place $p \in P$ is associated to a set $\Omega_{t,p} \in Jump$:

$$\Omega_{t,p} = \{(\Omega^S_{t,p}, \Omega^N_{t,p}, \Omega^D_{t,p}); \rho_{t,p}\} \quad (3)$$

where $(\Omega^S_{t,p}, \Omega^N_{t,p}, \Omega^D_{t,p})$ represents respectively a discrete, a continuous and a degradation assignment and $\rho_{t,p} \in \mathbb{N}^+$ is the weight of the outgoing arc.

During the transition firing, consumed tokens are moved in the output places of the fired transition t. The attributes of these tokens remain constant or are updated. This is defined by the set of assignments $\Omega_{t,p} \in Jump$ associated with the outgoing arc. For example, in Figure 1(a), the arc a_{t_2,p_2} is associated to the set $\Omega_{t_2,p_2} = \{(\Omega^S_{t_2,p_2}, \Omega^N_{t_2,p_2}, \Omega^D_{t_2,p_2}); \rho_{t_2,p_2}\}$, with:

- $\Omega^S_{t_2,p_2} = -$, which means that the discrete information of the token will not be updated,
- $\Omega^N_{t_2,p_2} = -$, which means that the continuous state of the token will not be updated,
- $\Omega^D_{t_2,p_2} = 0$, which means the degradation state of the token going through $a_{(t_2,p_2)}$ will be set to 0,
- $\rho_{t_2,p_2} = 1$, which means that only 1 token will be placed in p_2 after the transition firing.

For a more detailed description of the formalism, refer to Vignolles et al. (2021) where the HtPN formalism is introduced.

4. METHODOLOGY FOR LEARNING A HTPN

The process of HtPN learning is presented in Figure 2. It requires data and is performed offline because data normalization, the first step of the process, requires knowing the maximum and the minimum values of each data feature. As the process is offline, the number of samples used is finite. However, if the minimum and maximum values are known beforehand, the process can be done online. Data represent the observable variables of the system, usually acquired through sensors and actuators. In the case of non-faulty sensors, data truly represent the system's behavior. In a second step, these normalized data will be given to the Dynamic Clustering algorithm, DyClee. The results of DyClee are then interpreted and used to generate the structure of the HtPN. Combining these three steps

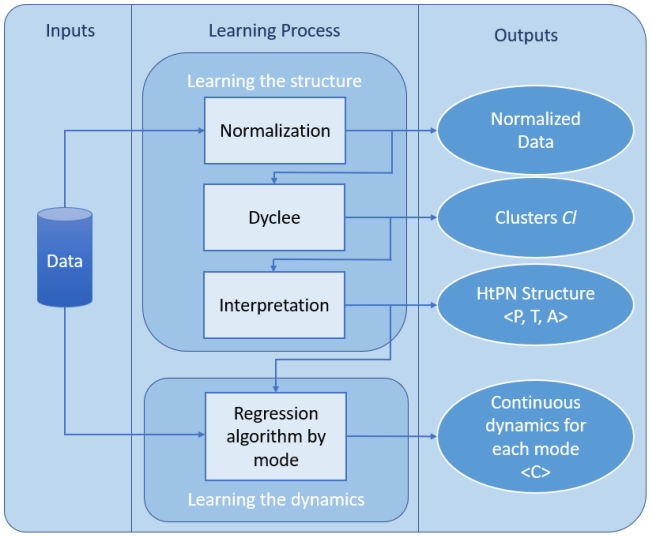


Fig. 2. Global idea for HtPN learning

determines the number of places and transitions for the structure. A fourth step aims to learn the system's continuous dynamics for each identified place of the obtained structure by applying regression algorithms using the data. This step outputs the continuous dynamics C associated with each place of the HtPN structure.

4.1 Input data

The input data used to learn the HtPN structure are the observable variables of the system. Some of these variables are sampled real values, like the water level in a tank, a pressure or a temperature. Some others are events, like the controls of a valve (open or close) or pressing an on/off button. An observable event is an event whose occurrence is detected from sensors. Input data are contained in an $i \times j$ matrix (as can be seen in Figure 6(a)). The line i corresponds to a sample of the data. Column j corresponds to an observable variable of the data. In the case of discrete events, different events might be present on the same column, the open and close controls on the same valve, for example. The set of samples is denoted S.

4.2 Learning the HtPN structure

Normalization Raw data require normalization before the clustering process for each feature to be equally important in the process. If a column of the data matrix ranges from 0 to 2 and another from 0 to 10^5, the first column might be neglected by the program as the values of the second column are more important. Data normalization modifies the input data file by:

- Creating a new column for each of the discrete events and converting the absence or presence of the event in a numerical value, respectively 0.0 and 1.0. This will ensure that each of the events has its column;
- Normalizing the values with the following equation:

$$data[i,j] = \frac{data[i,j] - min[j]}{max[j] - min[j]} \quad (4)$$

with $max[j]$ and $min[j]$ respectively the maximum and the minimum values of the j^{th} column and

$data[i, j]$ the value of the data of the j^{th} column and i^{th} line.

An example can be seen in Figure 6(b).

DyClee Once normalized, the data are given to DyClee along with a second file containing the user parameters for DyClee process, such as the desired size of the micro clusters. The size of the micro clusters needs to be chosen by an expert. DyClee outputs a set Cl of clusters, to which are assigned the data samples.

Interpretation of DyClee results Clustering results obtained with DyClee are automatically processed to output a HtPN structure. The pseudo code behind the interpretation of DyClee results is presented in Algorithm 1. For each cluster $Cl_i \in Cl$, a place p_i is created.
Then, the transitions linking these places are learned.

Algorithm 1 Obtaining the HtPN structure

Input: Cl, S
Output: $HtPN\_structure : P, T, A$

$P = \{\}, T = \{\}, A = \{\}$
for all $Cl_i \in Cl$ **do**
  $P \leftarrow P \cup Create(p_i)$
end for
$act_p \leftarrow Cl^{S[0]}$
$cnt = 0$
$created\_transitions = \{\}$
for all $s \in S$ **do**
  if $Cl^s != act_p$ **then**
    $cnt += 1$
  else
    $cnt = 0$
  end if
  if $cnt == 3$ **then**
    if $act_p\_Cl^s \notin created\_transitions$ **then**
      $[T, A] \leftarrow [T, A] \cup C\_t(act_p\_Cl^s, act_p, Cl^s)$
      $created\_transitions.add(act_p\_Cl^s)$
      $act_p \leftarrow Cl^s$
    end if
  end if
end for

Let $S[0]$ be the first sample of the sample set S. The cluster in which it is affected is noted $Cl^{S[0]}$ and is saved in the variable act_p. A counter cnt which initial value is 0 and an empty set of created transitions are defined. Then, for each sample s in S, the cluster Cl^s to which it was affected by DyClee is compared with the cluster saved in act_p. As long as $Cl^s == act_p$, the value of cnt remains 0. When a sample is assigned to another cluster (*i.e.* $Cl^s != act_p$), cnt is incremented. When this counter reaches 3, clusters in act_p and Cl^s are considered linked, and a transition is created, named $act_p\_Cl^s$. The place associated with the cluster act_p is linked as the input of this transition, and the place associated with the cluster Cl^s as the output. The created transition is saved in the set $created\_transitions$ to avoid creating multiple times the same transition if the samples oscillate between two clusters. The counter cnt was implemented to avoid creating false transitions because of outliers. We suppose that an outlier contains less than a given number of consecutive samples. As a starting point, this number is set to 3. The counter will prevent the creation of a transition linking the outlier with the rest of the structure, as it will not reach the transition creation threshold. As the learning of the DES structure is unsupervised, its quality cannot be evaluated.

Going back to the formal definition of a HtPN, after learning the HtPN structure with DyClee, the following sets describing the HtPN structure are completed:

- P, the set of places;
- T, the set of transitions;
- $A \subset (P \times T \cup T \times P)$, the set of arcs.

4.3 Learning continuous dynamics

Two regression algorithms are applied to learn the system's continuous dynamics even from noisy measurements: the Support Vector Regression (SVR) and the Random Forest Regression (RFR). Both algorithms are implemented using the toolbox Scikitlearn for Python. The advantage of using different algorithms is that the user can fully tune the learning. The user can then decide if the faster or the better fitted algorithm is chosen. To represent the fitting a coefficient of determination R^2 is used:

$$R^2 = 1 - \frac{(y_{true} - y_{pred})^2}{(y_{true} - y_{true_{mean}})^2} \quad (5)$$

where y_{true} is the true value of the data, $y_{true_{mean}}$ is the mean value of y_{true} and y_{pred} is the value predicted by the learned model. The closer the coefficient R^2 is to 1, the better the algorithm fits the data. It provides a measure of how well the learned model can replicate the observed outcomes. For this article, the better fitted algorithm has been chosen. This means that when building the final HtPN, each continuous dynamic will be represented by the learned model that better fits the data.

The continuous dynamics learning is done place by place giving the set C of continuous dynamics of the system as referenced in the formal definition of the HtPN. The set of equations C_p associated with a place p is a fitted SVR or RFR model:

$$C_p : y_k = RM(x_k), \quad (6)$$

where RM represents the obtained regressive model and $RM(x_k)$ is the prediction given by this model for a particular continuous state vector x at time k.

Although not yet implemented, a similar process is expected when learning the set D of degradation functions. However, the challenge in learning the degradation function is the time window considered, as degradation dynamics are usually much slower than continuous dynamics describing the system functioning.

5. APPLICATION AND RESULTS

5.1 System description

The benchmark used to illustrate the learning of a hybrid model is a three-tank system illustrated in Figure 3. The data are obtained from one single simulation of the system. The *a priori* knowledge of the model is used for easy

process verification. This illustration is a case study to ensure that the learning process behaves as expected.

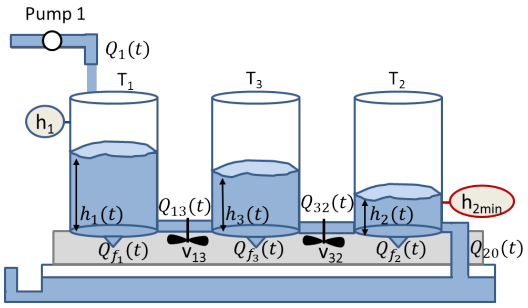


Fig. 3. Three-tank system description

The tanks are configured in a series circuit. Flow $Q_1(t)$ delivered by the pump in tank T_1 is supposed to be constant. Tank T_2 empties with flow $Q_{20}(t)$. The available measurements at time t are the water levels h_i in each tank T_i. Valves v_{13} and v_{32} allow the water to flow between tanks. Only valve v_{13} is controlled through discrete control inputs $open_{v_{13}}$ and $close_{v_{13}}$. A leak may occur in tank T_1 and is represented by an unobservable fault event f_1. 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 leads to system failure when f_1 occurs because the system cannot achieve the goal anymore. In this example, the continuous state vector represents the water level in each of the three tanks. The degradation state vector represents the probability of each fault occurrence.

The three-tank multimode representation is composed of six different modes as illustrated in Figure 4. The initial place is in nomimal mode Nom_1, in which both valves are open. When the command $close_{v_{13}}$ occurs, the system enters mode Nom_2, where v_{13} is closed. The system can go back to Nom_1 when $open_{v_{13}}$ occurs. From Nom_1 (resp. Nom_2), the system can switch to a degraded mode Deg_2 (resp. Deg_3) if fault f_1 occurs. Finally, from Deg_2 and Deg_3, the system may enter failure modes $Fail_4$ and $Fail_3$ if the water level in tank T_2 becomes smaller than the minimum value $h_{2_{min}}$. This is represented with the occurrence of a fault event f_0. The modes in the red square in Figure 4 are the *a priori* learnable modes of our system, be it directly through the occurrence of events (for $close_{v13}$ and $open_{v13}$) or a modification in dynamics (for the occurrence of f_1).

5.2 Learning the HtPN structure

The simulated scenario is shown in Figure 5. 310min after the initialization, v_{13} is closed every hour for 20min to perform a water treatment in T_1. Fault f_1 is injected at 201840s and f_0 occurs at 206040s. Neither f_1 nor f_0 are observable. Looking at this behavior, after the structural learning, five places can be expected.

A part of the data obtained with the simulated system is shown in Figure 6(a). The first column is the timestamp of the sample. The second, third and fourth columns are the water level in the first, second and third tanks. The last column indicates the occurrences of events. When an event occurs in the data, it will be memorized until

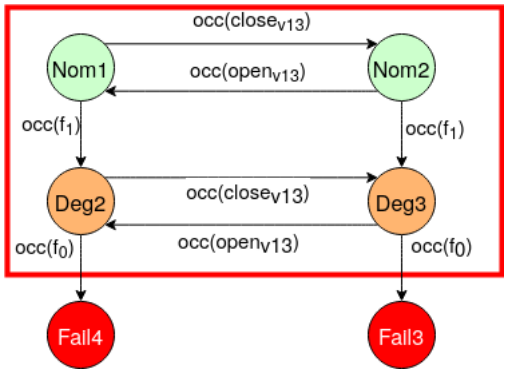


Fig. 4. Multimode representation of the three-tank system

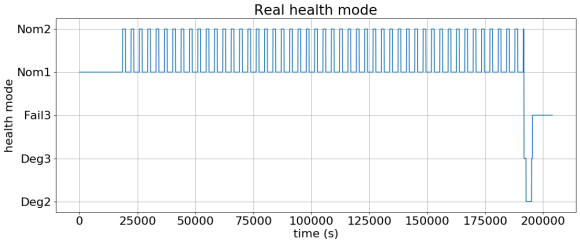


Fig. 5. The simulated scenario

the occurrence of another event. For example, the event *openv13* occurred at $t = 0$, and is memorized until the event *closev13* at $t = 309$.

```
308|2.0979920734557616|1.8823981584084746|1.98808594503705|openv13
309|2.1323807007763182|1.8827729021376856|1.954482648376446|closev13
```

(a)

```
308.0, 0.3235, 0.8272, 0.8217, 1.0, 0.0
309.0, 0.3314, 0.8274, 0.8078, 0.0, 1.0
```

(b)

Fig. 6. Example of data before (a) and after (b) normalization

For illustration, the data given in Figure 6(a) are shown post normalization in Figure 6(b). The first column remains unchanged as it corresponds to the timestamps of data. Columns 2, 3 and 4 were normalized between 0 and 1. The last column is split into two columns: one for the event *openv13* and one for *closev13*. Each of these columns takes the value 1.0 when the corresponding event has occurred and 0.0 otherwise.

The normalized data are used by DyClee, configured with a chosen size of the micro clusters equals to 0.2. This value was obtained empirically.

The results obtained by DyClee are shown in Figure 7: 4 clusters are obtained. The results from DyClee are then interpreted to learn the HtPN structure. For each of the four clusters identified by DyClee, a place p_i is created, with $i \in [1..4]$. Five transitions are created by using Algorithm 1: $p_1 \rightarrow p_2$, $p_2 \rightarrow p_1$, $p_1 \rightarrow p_4$, $p_4 \rightarrow p_3$ and $p_3 \rightarrow p_4$. Figure 8 illustrates the learned HtPN structure with places and transitions.

The clusters identified by DyClee can be compared to the true modes of the simulated system illustrated in Figure 5.

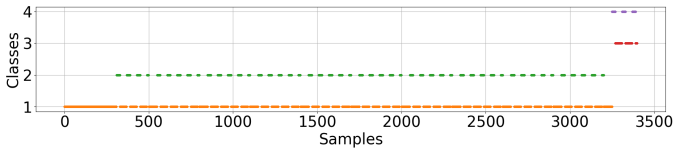


Fig. 7. Clusters identified by Dyclee

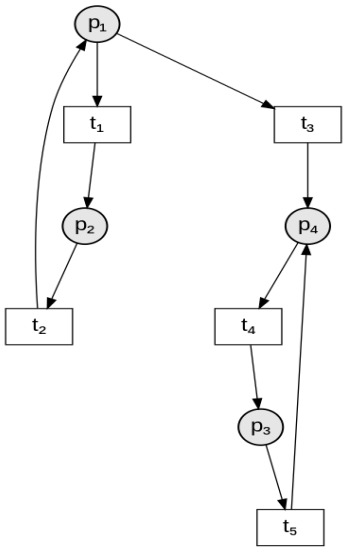


Fig. 8. Obtained HtPN structure

First, looking at both figures, mode Nom_1 can be linked to place p_1 and mode Nom_2 can be linked to place p_2. The injection of fault f_1 can be noticed in Figure 7 as the change to places p_3 and p_4. Although unobservable, fault f_1 impacts the behavior of the water level in tank T_1. However, fault f_0 goes unnoticed as fault f_0 represents the system's failure and is based on the water level in tank T_2 crossing a given threshold. Fault f_0 does not impact any dynamics of the system, thus it cannot be detected by DyClee.

The whole multimode representation could not be learned from observations in the available data. As said earlier, the learned part corresponds to the red square in Figure 4.

Considering these comments, the learning of the HtPN structure performed by Dyclee gives satisfying results. It can model the system's behavior quite precisely, as we managed to recover all the multimode representation that was retrievable with the available data.

5.3 Learning continuous dynamics

All the continuous dynamics associated with the continuous variables are learned during the complete learning process. The learning process is illustrated only on three specific sets of samples for the water level in tank T_1 by applying RFR and SVR algorithms for understandability purposes. The first dataset contains samples from 0 to 309, which corresponds to the place p_1 from the operation start until the event $close_{v13}$. The second set contains samples from 310 to 329 associated with place p_2. The last dataset contains samples from 330 to 370 for which the system returns in place p_1. The results of the dynamics learning process for the three datasets are shown respectively in Figures 9, 10 and 11. The real data are represented with red dots, the fitted SVR with the blue line and the fitted RFR with the yellow line. For each mode (*i.e.* a place), the fitted algorithms are scored using the coefficient of determination R^2 and the computational time and are compared in Table 1.

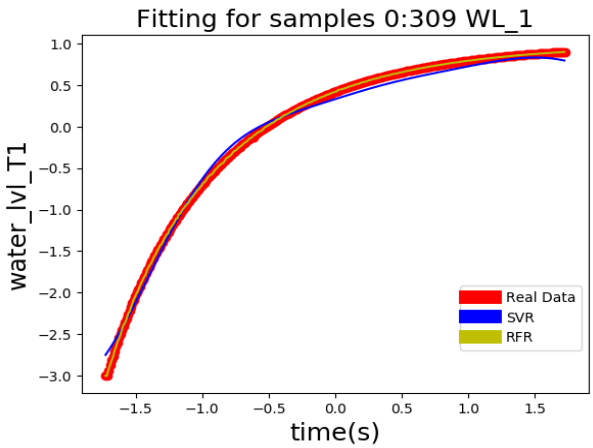


Fig. 9. Results for samples [0:309]

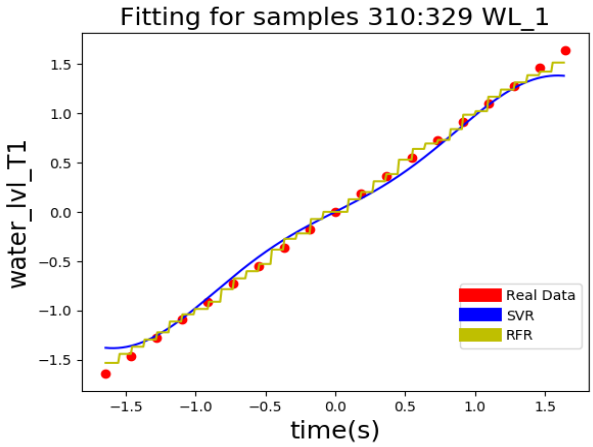


Fig. 10. Results for samples [310:329]

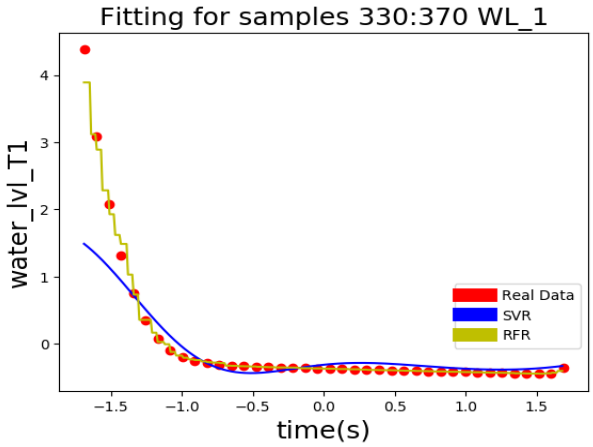


Fig. 11. Results for samples [330:370]

Table 1. Comparison of the R^2 score and fitting time

	R^2		fitting time	
samples	SVR	RFR	SVR	RFR
0 to 309	0.99356	0.99997	0.7ms	6.7ms
310 to 329	0.98908	0.99604	0.4ms	4.4ms
330 to 370	0.68164	0.99003	0.3ms	4.4ms

For this particular application, the SVR is usually faster by a factor 10 but has a very slightly inferior R^2 to the one of RFR, except for the samples 330 to 370 for which its R^2 shows that the fitted model cannot convey the dynamics of the system.

It can also be noticed that the continuous dynamic for h_1 learned from samples 0 to 309 is very different from the continuous dynamics learned from samples 330 to 370, even if it is considered as the same place p_1. This can be explained by the fact that samples 0 to 309 can be associated with an initialization phase in the system behavior. Considering this, an additional place could be added in the set P of the HtPN model and associated with the samples identified as the initialization phase in the system behavior. In this case, a place p_0 and a transition from p_0 to p_1 are created.

A similar process could have been applied to learn the set D of degradation functions based on the time of failure of the system. However, as it is not the focus of this first approach, it was left for future work.

6. CONCLUSION

This article presents a method to learn a hybrid model for system monitoring from data. The HtPN formalism has been chosen to perform hybrid system health monitoring under uncertainty because of its capability to represent the evolution of different hypotheses on the hybrid system health state thanks to parallelism. However, the learning process could be applied to other formalisms for hybrid systems. The first step of the learning method focuses on the global HtPN structure comprised of a set of places P, a set of transitions T and a set of arcs A using the DyClee clustering method. The second step aims at learning continuous dynamics associated with each place of the obtained structure. Continuous dynamics C are learned through RFR and SVR regression algorithms. The proposed learning method was applied on a three tank example. Based on simulation data, a hybrid model was learned and compared to the *a priori* known model. The results obtained are satisfying as the learned model fits what was expected given the available data.

Future work will focus on the learning of the missing parts of the model and the possibility to learn and improve a known model. Although missing, the sets E, X and $\mathbb{M}_0$ can be easily retrieved from the inputted raw data and are thus considered easy to obtain. The retrieval of the sets *Guard*, *Jump*, Γ, and D, however, will be the core of our future works as it appears to require thorough work. Moreover, the learnability of the process has to be defined and thoroughly investigated. Finally, the relationship between quantity and quality of data and the relevance of the model should be studied.

REFERENCES

Awad, M. and Khanna, R. (2015). Support vector regression. In *Efficient learning machines*, 67–80. Springer.

Barbosa, N.A., Travé-Massuyès, L., and Grisales, V.H. (2017). Diagnosability improvement of dynamic clustering through automatic learning of discrete event models. *IFAC-PapersOnLine*, 50(1), 1037–1042.

Djedidi, O. and Djeziri, M.A. (2020). Power profiling and monitoring in embedded systems: A comparative study and a novel methodology based on narx neural networks. *Journal of Systems Architecture*, 111, 101805.

Dotoli, M., Fanti, M.P., and Mangini, A.M. (2008). Real time identification of discrete event systems using petri nets. *Automatica*, 44(5), 1209–1219.

Drezet, P. and Harrison, R. (1998). Support vector machines for system identification.

Feng, C., Lagoa, C.M., and Sznaier, M. (2010). Hybrid system identification via sparse polynomial optimization. In *Proceedings of the 2010 American Control Conference*, 160–165. IEEE.

Henzinger, T. (2000). The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, 265–292. Springer.

Lauer, F. and Bloch, G. (2008). Switched and piecewise nonlinear hybrid system identification. In *HSCC 2008*, 330–343. Springer.

Leclercq, E., el Medhi, S.O., and Lefebvre, D. (2008). Petri nets design based on neural networks. In *ESANN*, volume 16, 529. Citeseer.

Liaw, A., Wiener, M., et al. (2002). Classification and regression by randomforest. *R news*, 2(3), 18–22.

Moreira, M.V. and Lesage, J.J. (2019). Discrete event system identification with the aim of fault detection. *Discrete Event Dynamic Systems*, 29(2), 191–209.

Niggemann, O., Stein, B., Vodencarevic, A., Maier, A., and Büning, H.K. (2012). Learning behavior models for hybrid timed systems. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Pillonetto, G. (2016). A new kernel-based approach to hybrid system identification. *Automatica*, 70, 21–31.

Roa, N.B., Travé-Massuyès, L., and Grisales-Palacio, V.H. (2019). Dyclee: Dynamic clustering for tracking evolving environments. *Pattern Recognition*, 94, 162–186.

Tamssaouet, F., Nguyen, K.T., Medjaher, K., and Orchard, M. (2020). A contribution to online system-level prognostics based on adaptive degradation models. In *ePHM Conf.*, volume 5.

Van Overschee, P. and De Moor, B. (1994). N4SID: Subspace algorithms for the identification of combined deterministic-stochastic systems. *Automatica*, 30(1), 75–93.

Vidal, R. (2004). Identification of PWARX hybrid models with unknown and possibly different orders. In *Proceedings of the 2004 American Control Conference*, volume 1, 547–552. IEEE.

Vignolles, A., Chanthery, E., and Ribot, P. (2021). Modeling complex systems with Heterogeneous Petri Nets (HtPN). URL `https://hal.laas.fr/hal-03137722`. Working paper or preprint.

Zhang, L. and Xi, Y. (2004). Nonlinear system identification based on an improved support vector regression estimator. In *International Symposium on Neural Networks*, 586–591. Springer.