Combining Particle Filters and Consistency-based Approaches for Monitoring and Diagnosis of Stochastic Hybrid Systems

Sriram Narasimhan[1] and **Richard Dearden** and **Emmanuel Benazera**[2]

Abstract. Fault detection and isolation are critical tasks to ensure correct operation of systems. When we consider stochastic hybrid systems, diagnosis algorithms need to track both the discrete mode and the continuous state of the system in the presence of noise. Deterministic techniques like Livingstone cannot deal with the stochasticity in the system and models. Conversely Bayesian belief update techniques such as particle filters may require many computational resources to get a good approximation of the true belief state. In this paper we propose a fault detection and isolation architecture for stochastic hybrid systems that combines look-ahead Rao-Blackwellized Particle Filters (RBPF) with the Livingstone 3 (L3) diagnosis engine. In this approach RBPF is used to track the nominal behavior, a novel n-step prediction scheme is used for fault detection and L3 is used to generate a set of candidates that are consistent with the discrepant observations which then continue to be tracked by the RBPF scheme.

1 Introduction

NASA's vision for the twenty first century includes robotic exploration of deep space and human-robotic exploration of Mars and the moon. Safety is a major priority for all these efforts, for manned as well as unmanned missions. One key component for autonomous operation of such systems while ensuring safety is fault detection and isolation. For safety-critical systems, fast and efficient fault detection and isolation techniques are necessary in order to maintain a high degree of availability, reliability, and operational safety [10]. These systems tend to be hybrid in nature (a mix of discrete and continuous dynamics), hence both the discrete mode and continuous state of the system need to be tracked. In addition the systems and any models of the system are stochastic due to operation in uncertain environments and the presence of sensor and process noise. As a result diagnosis algorithms for such systems also need to be stochastic in nature.

Bayesian belief update approaches to stochastic hybrid system diagnosis try to maintain an approximation of the true belief state either by sampling [5, 3] or by maintaining limited trajectories [4]. In order to deal with the continuous dynamics, the problem is broken up into a discrete mode estimation coupled with continuous state estimation. Typically Kalman filters are used to track continuous state and the estimated state is used as the observation function to update weights of particles or probabilities of trajectories. In these approaches the fault is diagnosed by ensuring particles enter the true fault mode and then the observation function would keep increasing the weight of these particles. If there are a large number of fault modes then this requires a lot of computational resources since particles need to be put in all fault modes to make sure no fault is missed.

On the other hand consistency-based approaches like Livingstone [12, 6] hand use structural and behavioral models (as opposed to transition models) to diagnose the faults. The system is modeled in abstracted form and the observations are also converted to this form ("monitors" for Livingstone). When the predictions from the model are not consistent with the observations, then the discrepancies are used to identify conflicts which are then used to identify possible fault candidates. These candidates can be tracked by comparing the predictions under these fault conditions against the observations. In this approach, rather than blindly guessing the faults, the constraints in the model are used to limit the candidates to be considered. However these approaches tend to be deterministic in nature (in some cases prior probabilities are used) and hence cannot deal with uncertain transitions and noise in the sensors and system.

In this paper, we combine these two approaches in an effort to reduce the computational complexity associated with probabilistic approaches while extending Livingstone-like approaches to handle stochasticity. Our approach combines the look-ahead Rao-Blackwellized Particle Filter (RBPF) [2, 5] and Livingstone 3 (L3) systems to provide a diagnosis architecture for stochastic hybrid systems. Section 2 describes the RBPF and L3 algorithms and also describes the unified modeling framework used by both diagnostic systems. In Section 3 we present the combined architecture and explain the different components of this architecture: the nominal observer, the fault detector, the fault observer and the candidate generator.

2 Preliminaries

2.1 The Look-Ahead Rao-Blackwellized Particle Filter (RBPF)

The look-ahead Rao-Blackwellized particle filter [2] is detailed in Algorithm 1. It differs from the standard particle filter in two important respects. First, it maintains sufficient statistics (in the form of Gaussian means and covariances) for the continuous part of the system state. Thus each particle (sample) can be thought of as consisting of a sampled discrete mode plus a Kalman filter that represents a distribution over the continuous value the system could have in that mode. Secondly, the algorithm employs look-ahead to ensure that low-probability states are properly accounted for. While a standard particle filter samples new states from the transition prior distribution

[1] QSS Group Inc., NASA Ames Research Center, Moffett Field, CA, USA, email:sriram@email.arc.nasa.gov

[2] RIACS, NASA Ames Research Center, Moffett Field, CA, USA, email:dearden@ptolemy.arc.nasa.gov, ebenazer@email.arc.nasa.gov

$P(Z_t|z_{t-1}^{(i)})$ (the Monte Carlo step in a standard particle filter), look-ahead allows us to incorporate the new observation to compute the *posterior* distribution $P(Z_t|z_{t-1}^{(i)}, y_t)$ over states the sample could end up in, and sample directly from that.

The look-ahead RBPF algorithm operates as follows. Assuming that we use N samples, first the discrete mode $z_0^{(i)}$ for each of the N is sampled from the given prior distribution Z_0 over the modes (Step 1 in Algorithm 1). The mean $\mu_0^{(i)}$ and covariance $\Sigma_0^{(i)}$ for the initial continuous state for each particle is also assumed to be given. At each time step three computations are performed which form the core of look-ahead RBPF. First, a look-ahead is performed for each sample. All successor modes for the sample are enumerated and the continuous state in each of these modes at the next time step is computed using a Kalman filter. The observation is then used to compute the posterior probability $\widehat{Post}(i,m)$ of the sample transitioning to the successor mode. The weight of the particle is re-calculated as the sum of these posterior probabilities. Secondly (Step 7), the particles are resampled as in the regular particle filter algorithm based on their weights. Thirdly (Steps 9 and 10), a new mode for each new particle is sampled from the posterior distribution computed in Step 5, and the mean and covariance are set to those computed for that mode by the Kalman filter in Step 4. Note that while look-ahead RBPF is limited to linear-Gaussian models by the use of a Kalman filter in Step 4, there is no reason why the Gaussian particle filter (GPF) [5] could not be used, allowing non-linear models to be tracked using the same algorithm. However, the use of an unscented Kalman filter [11] in the GPF algorithm complicates the discussion of our fault detection algorithm in Section 3.2 below.

1: Initialization: for N particles p^i, $i = 1, \cdots, N$, sample discrete modes $z_0^{(i)}$, from the prior $P(Z_0)$ and set $\mu_0^{(i)}$ and $\Sigma_0^{(i)}$ to the prior mean and covariance in state $z_0^{(i)}$. $t = 1$.
2: **for all** $p^{(i)} = (z_{t-1}^{(i)}, \mu_{t-1}^{(i)}, \Sigma_{t-1}^{(i)})$ **do**
3: **for** each possible successor mode $m \in succ(z_{t-1}^{(i)})$ **do**
4: Perform a Kalman update using parameters from mode m:

$$(\hat{y}_{t|t-1}^{(i,m)}, \hat{S}_t^{(i,m)}, \hat{\mu}_t^{(i,m)}, \hat{\Sigma}_t^{(i,m)}) \leftarrow KF(\mu_{t-1}^{(i)}, \Sigma_{t-1}^{(i)}, y_t, \theta(m))$$

5: Compute posterior probability of mode m as

$$\widehat{Post}(i,m) \leftarrow P(m \mid z_{t-1}^{(i)}, y_t) = P(m \mid z_{t-1}^{(i)}) \mathcal{N}(y_t; y_{t|t-1}^{(i,m)}, S_{t|t-1}^{(i,m)}, y_t)$$

6: Compute the weight of particle $\hat{p}^{(i)}$:

$$w_t^{(i)} \leftarrow \sum_{m \in succ(z_{t-1}^{(i)})} \widehat{Post}(i,m)$$

7: Resample N new samples $p^{(i)}$ where: $P(p^{(i)} = p^{(k)}) \propto w_t^{(k)}$.
8: **for all** $p^{(i)}$ **do**
9: Sample a new mode $m \sim P(Z_t \mid z_{t-1}^{(i)}, y_t)$
10: Set $z_t^{(i)} \leftarrow m$, $\mu_t^{(i)} \leftarrow \mu_t^{(i,m)}$ and $\Sigma_t^{(i)} \leftarrow \Sigma_t^{(i,m)}$.

Algorithm 1: The look-ahead RBPF algorithm

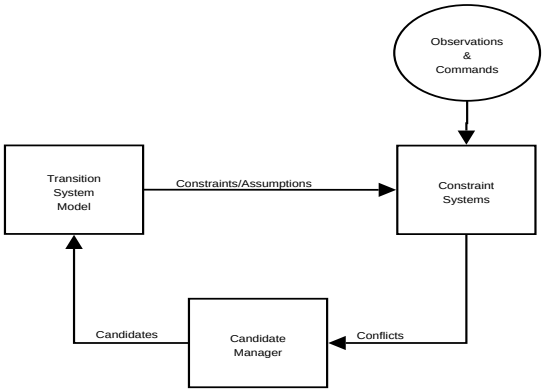

Figure 1. Livingstone 3 Diagnostic Architecture

2.2 Livingstone 3 (L3)

The L3 diagnosis architecture ([9]) is illustrated in Figure 1. It consists of three main components. The *system model* stores the model of the system and is responsible for tracking the modes of operation of the different components and determining the constraints that are valid at any point in time. The *constraint system* serves the role of tracking the overall system behavior using constraint programming techniques. It receives constraints from the System Model indicative of the current configuration of the system and propagates these constraints to try to assign consistent values to variables in the system. When inconsistencies are seen (observations are different from propagated values for corresponding components), the *candidate manager* is responsible for using the conflicts generated as a result of these inconsistencies to generate candidates that resolve all the conflicts and that can possibly explain all of the inconsistencies.

Algorithm 2 details the L3 approach. Each candidate c^i is represented as a triple (F^i,(z_t, s_t),E_t) where F^i is the set of fault transitions $((f_1, t_1), \ldots, (f_j, t_j))$ that are hypothesized to have occurred (with their time of occurrence), z_t is the discrete mode of the system and s_t is the state of the system at time t (current time) under the fault conditions and E_t is the explanation graph tracing back from values of variables at time t to values of variables at time $t - LT$ where L is some truncation limit applied to limit how far back we are willing to go in order to generate candidates. s_t includes all variables that have memory i.e., variables v whose values at time t, v_t depend on their values at time $t-1$, v_{t-1}. These may include continuous state variables x if the quantitative and continuous constraints are included in the Livingstone 3 models. Initially we only have the empty/null candidate in our candidate set C indicating our belief about no fault in the system. The initial discrete mode z_0 and the initial state s_0 are used as initial mode and state for the empty candidate. At this point the explanation graph for the empty candidate has only the variables from the current time step and no edges. At each time step t, we compute the new system discrete mode z_t and instantiate the constraints $q(z_t)$ into a constraint store Q for this mode. Note that the mode and constraints are actually composed from the modes and constraints for each component rather then pre-enumeration for all system modes. The state s_t is computed from s_{t-1} (incorporating any transition conditions imposed by transition from z_{t-1} to z_t). We add edges $s_{t-1} \rightarrow s_t$ and $z_{t-1} \rightarrow z_t$ in the explanation graph. In addition if z_{t-1} to z_t was a transition based on some decision function over internal variables (autonomous transition) we add edges from all s_{t-1} involved in the decision function to z_t. Finally we remove all

variables at time $t-LT-1$ (including s_{t-LT-1} and z_{t-LT-1}) from the explanation graph. The sensed input values $u_t = U$ are added as constraints $q(u_t)$. The resulting set of constraints (constraint store) Q are propagated to infer values for other variables including the output variables y_t. The explanation graph is also updated based on the propagation of variable values. These are compared against the observations o_t and any discrepancies are used to identify conflicts by tracing back in the explanantion graph starting from the discrepant y_t. The conflicts are then used to generate faults $F_1, \ldots, F_j$. Candidate c^i is replaced by $c^i_1, \ldots, c^i_j$ in candidate set C where c^i_j = (F_j,(z_t, s_t),E_{t-1}).

```
1:  C = c^0;c^0=(φ, (z_0, x_0), φ)
2:  for each time step t do
3:    for all c^i ∈ C do
4:      z_t = Next(z_{t-1});s_t = Next(s_{t-1})
5:      E_t = E_{t-1} ∪ (z_{t-1} → z_t, s_{t-1} → s_t) \ E_{t-KT-1}
6:      Q = q(z_t) ∪ q(u_t)
7:      Propagate Q; Update E_t
8:      if y_t ≠ o_t then
9:        Isolate faults f_1, ..., f_j
10:       C_new = c_1(f^1_t, z_t, s_t, E_t), ..., c_j(f^j_t, z_t, s_t, E_t)
11:       C = C − c^i ∪ C_new
```

Algorithm 2: The L3 algorithm

2.3 Modeling Paradigm and Assumptions

We assume that the stochastic hybrid system is modeled as a network of hybrid automata in a component connection framework. In other words, the system is modeled as a set of components and connections between them. The connections that typically connect variables across two components constrain the two variables to be equal. If causality can be established then one variable serves as an output variable and other serves as an input variable. The behavior of any component is modeled as a hybrid automaton where the states of the automaton represent discrete modes of operation of the component. Faults are modeled as instantaneous abrupt transitions to fault modes where the behavior of the component is known beforehand[3]. The behavior of the component in each mode is modeled as differential algebraic equations (for the RBPF) and as constraints from other constraint systems like Boolean and Enumeration domains (for L3). The system mode z is computed as the composition of the individual component modes $z(C^i)$, i.e., $z = z(C^1) \cup \ldots \cup z(C^n)$ where $C^1, \ldots, C^n$ are the components in the system model. The behavior of the system in any mode z^j can be expressed as a union of the set of constraints enforced by each component plus the set of global constraints (constraints that do not depend on the mode of any component), i.e., $M(z^j) = M(z^j(C^1)) \cup \ldots \cup M(z^j(C^n)) \cup M^G$.

The transitions between the discrete modes include a transition conditions and a probability (constant) value. The transition condition determines when the component may switch modes of behavior. The probability value then stochastically determines the chances of the transition actually being fired. Transitions can be of three types Commanded, Autonomous, and Faulty. Commanded transitions are changes in modes as a result of external commands while autonomous transitions are changes in component modes as a result of internal conditions. Commanded transitions are modeled with the sensing of the issuance of an external command event as the transition condition and a probability indicating the chance of the command actually being executed. Autonomous transitions are modeled by a transition conditions that is a boolean valued function of internal variables and a probability that indicates the chances of the transition being fired when the function evaluates to TRUE. However autonomous transitions may involve additional uncertainties due to the fact in some cases we can only estimate distributions for values of the variables involved in the decision function. Faulty transitions (transitions to fault modes) are modeled with unobserved events as transition conditions (we want to determine the occurrence of exactly these events) with an associated probability that indicates the prior probability of that fault.

[3] Unknown modes may be used to represent faults for which the behavior of the component is not known before hand.

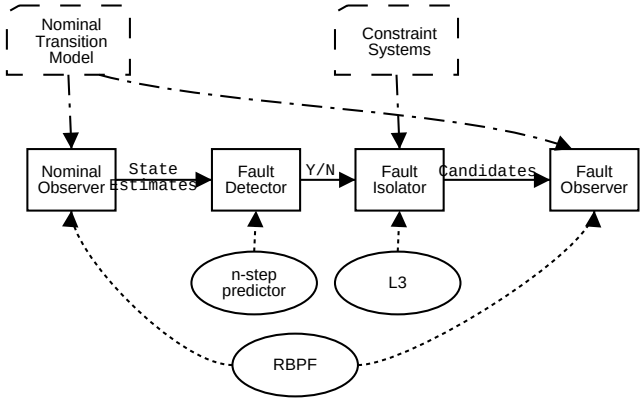

Figure 2. Combined Diagnostic Architecture

3 Combined Architecture

Our diagnosis architecture illustrated in Figure 2 consists of four main components, a nominal observer, a fault detector, a candidate generator and a fault observer. The nominal observer tracks the evolution of system behavior till a fault is detected at which point the candidate generator is used to generate a set of candidates that are then tracked by the fault observer.

Before presenting our architecture we list some of the assumptions that we need:

1. The modes of the components can be separated into a set of nominal modes and a set of fault modes. This also implies that the transitions may be divided into a set of nominal transitions (transitions into nominal modes) and faulty transitions (transitions into fault modes). In general, this assumption may be relaxed as long as we make a separation between the modes we want track with the particle filter and the modes that we want to track using the Livingstone framework. For example, the separation may be based on the probability associated with the transitions.
2. Faults can be detected within K time steps of occurrence and this K is known beforehand.
3. Multiple faults do not occur with K time steps of each other.

3.1 Nominal Observer

The nominal observer is used to track the evolution of the nominal behavior of the system. The discrete modes of the system are

tracked by sampling from a posterior probability whereas the distribution over the continuous state is estimated by an unscented Kalman filter. Obviously there is uncertainty in the state estimates which is captured as a distribution. However there is also uncertainty about the discrete modes. In the case of commanded transitions (where the commands are sensed) the actual execution of the command may not be certain and there might a small chance that the command is not executed. Also since autonomous transition conditions are based on the continuous state, the uncertainty about the estimated continuous state leads to an uncertainty in evaluation of the autonomous transition conditions.

The nominal observer uses the look-ahead RBPF algorithm (Algorithm 1) to track the discrete modes and continuous state of the system as a set of particles $p^{(1)},\ldots p^{(N)}$, where N is the number of particles used. However instead of pre-computing the state space equations of the system in each system mode, we use a lazy approach. We maintain a cache of visited modes and the state space equations in these modes $\{z^i \leftarrow f(z^i)\}$ where f is a possibly non-linear function representing the state space equations. Initially this cache is empty. When a particle changes its discrete mode from z_t to z_{t+1} during the re-sampling step (including the initial sampling step), if $z_t \neq z_{t+1}$ we look in the cache for z_{t+1}. If it is found then the corresponding model $f(z_{t+1})$ is used for the unscented Kalman filter update. If it is not found then we instantiate the constraints in the system $M(z_{t+1})$ (equation 1) and then symbolically derive $f(z_{t+1})$ from $M(z_{t+1})$ [8] and add it to the cache.

3.2 Fault Detector

Since we are not tracking the fault modes we need a fault detector to indicate that a fault has occurred in the system. In our case, a fault in the system is detected only if the fault detectors in all the particles indicate the presence of the fault. When the fault detector in a single particle raises a flag this may be because of an actual fault in the system or because the trajectory that the particle is tracking does not match the true trajectory of the system. As a result we do not start isolating the fault until all particles indicate a discrepancy in the tracking. We first describe the fault detector mechanism that is run locally in each particle. Then we describe what happens when a fault is detected locally (by some particles) but that globally the fault has not been detected (there exist some particles that are tracking ok).

Since each particle is using a Kalman filter to track the system behavior, the fault detection has to take this into consideration. A number of fault detection schemes that work in conjunction with a Kalman filter tracking approach have been proposed (see for example [1, 7]). However, these fault detection schemes use only a single step prediction for fault detection: the likelihood of the estimate is assessed through the likelihood of the observation y_t given the one-step prediction $\tilde{y}_t^1 = \mathcal{N}(\tilde{\nu}_t^1, \tilde{\Phi}_t^1)$[4] (this is the Kalman filter from step $t-1$ with the Kalman gain equation applied, but before conditioning on the observation). This can run into problems in certain situations. We know that the mean of $\hat{y}_t = \mathcal{N}(\hat{\nu}_t, \hat{\Phi}_t)$ is always between y_t and $\tilde{\nu}_t^1$. We distinguish three different cases (assuming $y \leq \tilde{\nu}_t^1$, $\ll$ indicating significant difference and $<$ indicating *close to*): (i) $y_t \ll \hat{\nu}_t < \tilde{\nu}_t^1$, (ii) $y_t \ll \hat{\nu}_t \ll \tilde{\nu}_t^1$, (iii) $y_t < \hat{\nu}_t \ll \tilde{\nu}_t^1$. The one-step prediction approach will not work in case (iii) and may not work in case (ii) because the difference between the prediction and the measurement may not be large enough given the process noise. Moreover, in a model with significant noise, the Kalman filter will tend to closely track the observation at every step due to the Kalman update, and will therefore reduce the chance of future detection of the fault.

One solution is to collect statistics over the difference between $\hat{y}$ and y over time, and to detect model bias by looking at whether that difference is consistently positive or negative (i.e. if it is random, then it may be considered as noise, if it is always positive or always negative, that is a bad model). The drawback of this approach is that it is usually not clear when to stop collecting data to take a decision.

In [7] is presented an approach based on a discrete wavelet transform in combination with a statistical decision function. Unfortunately it requires the precise setting of several parameters and thresholds either by hand or (fault) simulations, that precludes its use with complex hybrid systems with exponential (fault) mode combinations.

We overcome these problems by extending the prediction to n-steps besides the standard estimate computation, and use an approach based on decision theory. Two derivable decision functions are built, to assess for the nominal and faulty behaviors respectively whose equality at each time step defines a sharp non-faulty/faulty decision threshold. Furthermore, their variations are studied to adapt the window size n.

We write $\tilde{y}_t^n = \mathcal{N}(\tilde{\nu}_t^n, \tilde{\Phi}_t^n)$ for the n-step prediction of the observation as before, and $\tilde{x}_t^n = \mathcal{N}(\tilde{\mu}_t^n, \tilde{\Sigma}_t^n)$ for the n-step prediction of the process. The n-step predicted state $\tilde{x}_t^n$ is calculated by taking $\hat{x}_{t-n}$ and then predicting the state forward without the Kalman updates, up to current time t. Similarly, the n-steps covariance $\tilde{\Sigma}_t^n$ includes the process noise over n-steps but is not minimized. Its value is generally large. Since we will be comparing the n-step prediction with the Kalman filter, we will work in the process space rather than the observation space because it has more dimensions, so our estimates of the similarity of the two should be more accurate.

We assess the *likelihood of the filter estimate* as opposed to the likelihood of the observation:

$$\begin{aligned} p(\hat{x}_t \mid y_t, \tilde{x}_t^n) &= \frac{p(y_t, \tilde{x}_t^n) p(\hat{x}_t)}{p(y_t, \tilde{x}_t^n)} \\ &\approx p(y_t \mid \hat{x}_t^n, \hat{x}_t) p(\tilde{x}_t^n \mid \hat{x}_t) p(\hat{x}_t) \end{aligned}$$

If this probability is high then we expect the system to be nominal. We approximate it by computing a likelihood (L) of the estimate in the form of the nominal (N) indicator $L(N \mid y_t, \tilde{x}_t^n) \triangleq L(y_t \mid \hat{x}_t^n) L(\tilde{x}_t^n \mid \hat{x}_t) P_N$. The *a priori* n-step likelihood $L(y_t \mid \tilde{x}_t^n)$ is based on the distance between the observation y_t and the n-step observation prediction $\tilde{y}_t^n$. This is a natural extension to the one-step a priori likelihood. Due to the potentially large variance $\tilde{\Sigma}_t^n$, it may not be sufficient for quick detection. Then we propose to examine the Kullback-Leibler (KL) divergence between $\tilde{x}_t^n$ and $\hat{x}_t^n$, which measures how different the two distributions are. $KL(\tilde{x}_t^n, \hat{x}_t)$ can be understood as the average number of bits that are wasted by encoding events from the predicted distribution (over n-steps) with a code based on the estimated distribution. Therefore, the less bits are wasted, the more it is likely the system behavior is nominal. We thus note $L(\tilde{x}_t^n | \hat{x}_t) = KL(C - \tilde{x}_t^n, \hat{x}_t)$, where $C = 1/(2\pi^{n_x/2} |\tilde{\Sigma}_t^n|^{1/2})$. The fault ($F$) indicator follows: $L(F \mid y_t, \tilde{x}_t^n) \triangleq (C - L(y_t \mid \tilde{x}_t^n)) KL(\tilde{x}_t^n, \hat{x}_t)$. $P_N = \sum_{m \in succ_N(z_t)} P(m|z_t)$, $succ_N(z_t)$ denotes the *nominal successors* to the current mode z_t, and $P_F = 1 - P_N$.

We build a decision function g based on these indicators $g_N(y_t, \tilde{x}_t^n) = \log(L(y_t \mid \tilde{x}_t^n)) + L(\tilde{x}_t^n | \hat{x}_t) + \log(P_N)$ assesses for the nominal behavior and $g_F(y_t, \tilde{x}_t^n) = \log(C - L(y_t \mid \tilde{x}_t^n)) + KL(\tilde{x}_t^n | \hat{x}_t) + \log(P_F)$ for any faulty behavior. The sign

[4] We use a non-standard notation for the Kalman filters in this section because we want to be very clear about when we are referring to the complete distribution ($\hat{x}$, etc.) and when we are referring to the mean ($\hat{\mu}$ etc.).

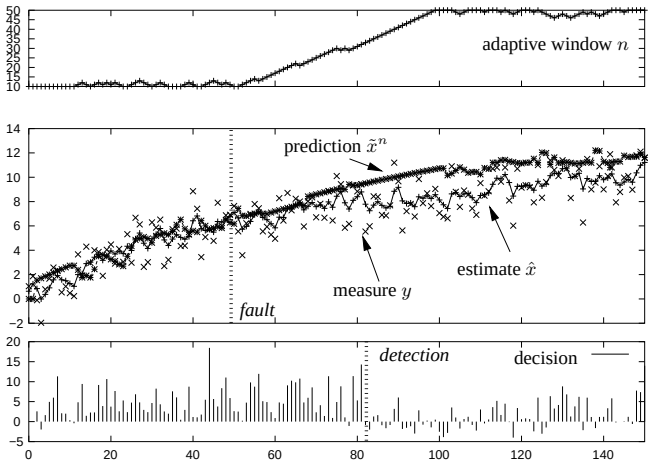

Figure 3. Filter fault detection and adaptive monitoring window.

of $g(y_t, \tilde{x}_t^n) = g_N(y_t, \tilde{x}_t^n) - g_F(y_t, \tilde{x}_t^n)$ is studied. This value is returned by the filter, besides the current estimate and serves as a fault indicator (when < 0). An extension to this filter automatically adapts n by considering the decision function g variations, so n increases when a fault decision becomes more likely, and decreases otherwise. Results are presented on figure 3. The three graphs show:

Middle graph The actual observations that are being tracked, the n-step predictor of state, and the Kalman filter estimate of state for a system in which a fault occurs.

Top Graph The number of steps for the n-step predictor growing as the likelihood of detecting a fault grows.

Bottom Graph The value at each step of the decision function. The fault is detected approximately 30 steps after it actually occurs (when the function becomes negative).

When the fault detector running inside a particle detects a fault, the particle sets a timer for KT (T is the observation sampling rate) and goes to sleep. If all other particles detect a fault before this timer runs out then the particle wakes up and starts the candidate generator algorithm. If the timer expires and there are still some particles that have not indicated a fault then the particle wakes up and sets its weight to 0 and gets re-sampled. During the time that particles are asleep we continue tracking with fewer numbers of particles (total particles less the particles that are asleep).

3.3 Candidate Generator

The candidate generator is started for each particle when the fault detectors of all particles have raised a flag. Let the time when the last particle indicates a fault be t_{fd}. Let the time of fault detection for any particle $p^{(i)}$ be $t_{(i)}$. For each particle $p^{(i)}$, we run the L3 single fault candidate generation algorithm (from assumption 3) starting at time $t_{(i)}$ and backtracking to time $t_{fd} - KT$ (from assumption 2). Note that if $t_{(i)} <$t$_{fd} - KT$, then the particle is re-sampled as a new particle.

For each particle $p^{(i)}$, we use the Livingstone 3 diagnosis engine in the following fashion. First we run L3 in a purely simulation mode to get predictions for all observed variables between time $t_{(i)}$ and current time t. Then we compare these predictions against the observations at the corresponding times to identify a set of discrepancies. These discrepancies are of the form $((v_1, t_1), \ldots, (v_k, l_k))$ where v_k is a variable and t_k is the time at which v_k was discrepant. During the simulation we build up the explanation graph from $t_{(i)}$ to time t. The explanation graph traces the justifications for assigning values to variables (for example propagating a constraint) all the way back to the reason for adding a constraint. Some constraints are added based on the assumption that components are nominal and we need to figure out which of these assumptions are necessary for derivation of discrepancies. For each discrepancy (v_k, l_k), We trace back in the explanation graph to identify a subset of assumptions that contributed to the discrepancy called a conflict c_k. We now generate one single fault candidate (f_j, t_j) that resolves all of the identified conflicts but only for $t_j \leq t_{(i)}$. We "install" this candidate and simulate as before to get predictions for all time steps between t_j and current time t (all predictions before t_j should be the same as nominal). The comparison against observations yields a new set of discrepant observations and conflicts (which in this case may include fault assumptions) which are then added to the conflict set. If there are no discrepant observations then this fault candidate is added to the possible fault candidate set. Another single fault candidate is generated and the process repeated until we cannot generate anymore single faults (alternately we may restrict ourselves to a fixed number of candidates).

3.4 Fault Observer

After the L3 candidate generator has isolated faults indicating both the fault transition and the time of transition, we need to run a fault observer to track the behavior of the system under these fault conditions. We use the RBPF to do this job also. Let us assume that for each particle $p^{(i)}$, L3 has isolated a set of fault candidates $((f_1, t_1), (f_2, t_2), \ldots, (f_j, t_j))$. We replace $p^{(i)}$ with j new particles $(p_1^{(i)}, \ldots, p_j^{(i)})$. Each new particle $p_j^{(i)}$ sets a timer to t_j indicating the time when the particle starts particpating in the RBPF. The initial continuous state for this new particle $((\mu_0^j, \Sigma_0^j))$ is set to the continuous state of the original particle at time t_j $((\mu_{t_j}^i, \Sigma_{t_j}^i))$ i.e., $\mu_0^j = \mu_{t_j}^i$ and $\Sigma_0^j = \Sigma_{t_j}^i$. This follows since the new particle follows the same trajectory as the original particle until the time of fault (t_j). Hence there is no need to track the behavior of this particle before the fault.

We now restart the RBPF from time $t_{fd} - KT$ with these newly created particles. As mentioned before each particle p^j gets added to the RBPF scheme only when the time step reaches t_j. The fault observer runs in this fashion until time t_{fd}. Note that we are still using only the nominal transition model to sample the discrete mode changes. While the fault observer is running, the fault detector is suppressed and no new faults are detected (Assumption 3). After time t_{fd} we switch to the nominal observer scheme (including the fault detector) since the fault is assumed to have occurred before t_{fd}. The candidate generator and fault observer algorithm is presented as Algorithm 3.

The complete diagnostic algorithm is illustrated in Algorithm 4. The RBPF is used in nominal observer mode to track the evolution of the nominal behavior of the system. The fault detector decision function g_t is used to determine if there is any discrepancy between the predictions and observations. If there is such a discrepancy for any particle $p^{(i)}$ then the particle goes to sleep for KT time steps at the end of which it kills itself by setting its weight to 0 ($w_{t+KT+1}^{(i)} = 0$). Once all particles have gone to sleep implying that none of the nominal trajectories are consistent with the observations we run the candidate generator and fault observer (Algorithm 3) to isolate and track the faults from time $t - KT$ to time t at which point we switch back to the nominal observer.

1: **for all** $p^{(i)}$ **do**
2: Run L3 to generate single fault candidates:

$$Z = (z^{(i)}_{t-KT}, \ldots, z^{(i)}_t)$$

$$((f_1, t_1), (f_2, t_2), \ldots, (f_j, t_j)) = DoL3(t^{(i)}_{fd}, Z)$$

3: **for** each (f_j, t_j) **do**
4: Create a new particle:

$$p^{(i)}_j(z = z_{t_j} + f_j, \mu = \mu(\hat{x_{t_j}}), \Sigma = \Sigma(\hat{x_{t_j}}), w = w^{(i)} \times P(f_j))$$

5: Sleep till time t_j
6: Run the RBPF algorithm from time t-KT to t

Algorithm 3: Candidate Generator and Fault Observer Algorithm

1: **for** each time step t **do**
2: Run RBPF to estimate $(z_t, \hat{x}_t, \tilde{x}_t)$ for each particle $p^{(i)}$
3: Compute $g^{(i)}_t$ as detailed in Section 3.2
4: **if** $g^{(i)}_t < 0$ **then**
5: Put particle $p^{(i)}$ to sleep till time $t + KT + 1$
6: Kill the particle after time $t + KT + 1$: $w^{(i)}_{t+KT+1} = 0$
7: **if** all particles are asleep **then**
8: Run candidate generator and fault observer

Algorithm 4: The Combined algorithm

4 Conclusions and Future Work

We have presented an architecture for monitoring and diagnosis of stochastic hybrid systems. Our approach combines the Rao-Blackwellised particle filter for tracking discrete mode transitions and continuous state, an n-step predictor scheme for fault detection and the Livingstone 3 algorithm for fault isolation. This is work in progress and we do not yet have results from an integrated system. However some of the pieces of the architecture have been tested with promising results. We hope to have results from the complete system in the final version of this paper.

There are several avenues for improving this architecture. One obvious thing to do is determine empirically exactly how K and n should be set in relation to one another (assuming n is not being adapted dynamically). Intuitively, we might expect them to be the same, since they are both measuring how many steps it takes before a fault can be detected. However, there is a subtle difference: K is a property of the system as a whole, the number of steps it takes to detect any fault. On the other hand, n is a parameter of the algorithm; we can vary n to trade-off between more accurate estimation of when a fault actually occurred and reliability of detection (a small n means less time in which the fault could have happened, but may miss faults that can't be distinguished in n steps). Clearly n should never be larger than K, but how close it should be may depend on the consequences of not knowing the time of the fault reliably. In Section 3.2 we discussed the fact that n can be adapted on-line to improve performance for a particular fault. This allows n to be adapted to the particular fault being tracked (n increases if the fault is very gradual and hence harder to distinguish from nominal). Again, the choice of when to increase or decrease n may be informed by how important it is to determine the actual time of the fault.

We have only discussed linear models and standard Kalman filters in this work. Since very few real-world systems are actually linear, we plan to extend the work to non-linear systems. The Gaussian particle filter [5] already uses unscented Kalman filters (UKF) to apply RBPF to non-linear systems, but the fault detector will need to be modified to cope with non-linearity. The difficulty here is that the approximation that occurs in the UKF will be magnified by the n steps of the predictor so the variance of the resulting distribution may be very large indeed, leading to increased problems in detecting faults early and reliably.

Finally, Livingstone 3 is currently being expanded by adding additional constraint systems, including the ability to use continuous constraints. As these new capabilities become available, L3 should be able to generate a richer set of candidate diagnoses, and hence the performance of the system we have described here should be able to be similarly improved.

REFERENCES

[1] Gautam Biswas, Gyula Simon, Nagabhushan Mahadevan, Sriram Narasimhan, John Ramirez, and Gabor Karsai, 'A robust method for hybrid diagnosis of complex systems', in *Fourteenth International Workshop on Principles of Diagnosis (DX '03)*, Washington D.C., USA, (2003).

[2] Nando de Freitas, Richard Dearden, Frank Hutter, Ruben Morales-Menendez, Jim Mutch, and David Poole, 'Diagnosis by a waiter and a mars explorer', *Invited paper for Proceedings of the IEEE, special issue on sequential state estimation*, (2003).

[3] Stanislav Funiak and Brian Williams, 'Multi-modal particle filtering for hybrid systems with autonomous mode transitions', in *Fourteenth International Workshop on Principles of Diagnosis (DX '03)*, Washington D.C., USA, (2003).

[4] M. Hofbaur and B.C. Williams, 'Mode estimation of probabilistic hybrid systems', *Hybrid Systems: Computation and Control, Lecture Notes in Computer Science (HSCC 2002)*, **2289**, 253–266, (2002).

[5] Frank Hutter and Richard Dearden, 'The gaussian particle filter for diagnosis of non-linear systems', in *Proceedings of the Fourteenth International Workshop on the Principles of Diagnosis*, Washington, DC, (2003).

[6] James Kurien and Pandu Nayak, 'Back to the future with consistency-based trajectory tracking', in *AAAI/IAAI*, pp. 370–377, (2000).

[7] Eric-J. Manders and Gautam Biswas, 'Fdi of abrupt faults with combined statistical detection and estimation and qualitative fault isolation', in *In Proc. of the 5th Symposium on Fault Detection, Supervision and Safety for Technical Processes, Washington DC*, pp. 347–352, (2003).

[8] Sriram Narasimhan, *Model-based Diagnosis of Hybrid Systems*, Ph.D. dissertation, Vanderbilt University, Nashville, TN, USA, August 2002.

[9] Sriram Narasimhan, Lee Brownston, and Daniel Burrows, 'Explanation constraint programming for model-based diagnosis of engineered systems', in *IEEE Aerospace Conference*, Big Sky, Montana, (2004).

[10] R. Patton, P.M. Frank, and R.N. Clark, *Issues in Fault Diagnosis for Dynamic Systems*, Springer Verlag, 2000.

[11] E. Wan and R. van der Merwe, 'The unscented kalman filter for non-linear estimation', in *Proc. of IEEE Symposium 2000*, Lake Louise, Alberta, Canada, (2000).

[12] Brian Williams and Pandu Nayak, 'A model-based approach to reactive self-configuring systems', in *AAAI*, pp. 971–978, (1996).