State-Augmented Mutating Particle Filtering for Fault Detection and Diagnosis

by

Cameron Brown

B.S. in Mechanical Engineering, University of Pittsburgh, 2016

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2019

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Cameron Brown

It was defended on

August 13, 2019

and approved by

Daniel Cole, Ph.D., P.E., Associate Professor, Department of Mechanical Engineering and Materials Science

Jeffrey Vipperman, Ph.D., Professor, Department of Mechanical Engineering and Materials Science

William Clark, Ph.D., Professor, Department of Mechanical Engineering and Materials Science

Thesis Advisor: Daniel Cole, Ph.D., P.E., Associate Professor, Department of Mechanical Engineering and Materials Science

State-Augmented Mutating Particle Filtering for Fault Detection and Diagnosis

Cameron Brown, M.S.

University of Pittsburgh, 2019

This research develops a model-based particle filter algorithm for quickly detecting sudden faults in dynamic systems. Faults are defined as the abnormal behavior or failure of the system components. This novel method avoids the numerical issues of some other model-based methods. It also allows the fault magnitudes to take on continuous values, instead of constraining them to discrete values.

The multiple-model particle filter (MMPF) and interacting multiple-model particle filter (IMMPF) techniques are tested on a nuclear reactor pressurizer system for the detection of loss-of-coolant accidents (LOCA). The drawbacks of these methods leads us to the develop the novel algorithm: the state-augmented mutating particle filter (SAMPF), which uses random walk techniques. The SAMPF detects sudden faults faster than conventional random walk techniques. Choosing the proper parameters for the algorithm is considered. The performance of the SAMPF is compared to that of the IMMPF for the pressurizer system. The SAMPF is superior to the IMMPF in fault diagnosis accuracy and consistency.

Table of Contents

List of Tables

List of Figures

1.0 Introduction

The goal of this research is to develop a new technique that can quickly detect sudden and severe faults. Faults are the abnormal behavior or failure of the components that make up a dynamic system. Some examples of system components are the LIDAR on a self-driving vehicle, the thrusters on a rocket, or the pipes in a nuclear power plant. The LIDAR may fail and read incorrect distances from its environment. The thruster may fail and output a thrust that is lower than needed for the desired trajectory. The coolant-carrying pipes of a nuclear power plant may rupture, affecting the flow that is cooling the reactor core. A component may even be non-physical, such as the software that helps to operate a system. Safe system operation can be compromised by the incidence of faults in these components. If faults are not detected and dealt with quickly, there may be damages to expensive components or risk to human life. In cyber-physical systems, confidential information may be lost.

Fault detection and diagnosis (FDD) has become an important part of system operation. Detection is the acknowledgement that a fault has occurred, and diagnosis is the act of determining the severity of the fault. In some systems, operators monitor sensor measurements and carry out FDD manually. As systems have become increasingly complex, the number of parameters has increased beyond the point of effective operator cognition. Some important parameters may not even be able to be measured directly, i.e, not able to be displayed to the operator. Therefore, there has been a need for FDD algorithms that can detect and diagnose faults automatically. Automated algorithms can generally track multiple measurements more efficiently and react much more quickly than a human can.

1.1 Overview of Fault Detection and Diagnosis Techniques

The most basic method of automated FDD is to trigger an alarm when a sensor value exceeds a threshold. The advantage of this method is its simplicity and reliability. However, the alarm only triggers when there is a large change in sensor measurement. Even severe

faults can cause subtle changes in measurements. Also, if the fault does not directly affect a measured value, this threshold method does not often allow for the diagnosis of the severity of the fault. So, more advanced analytical methods of FDD, such as data-driven and model-based methods, have been created.

Two broad categories of FDD methods are data-driven and model-based. Data-driven methods rely on large quantities of historical system data to build input/output models of a system. Model-based methods use physics and mathematics to derive models of the system. Note that these categories are rough and have much overlap.

1.1.1 Data-driven methods for FDD

Data-driven methods use historical measurements from the system to find implicit relationships between process parameters. This category is dominated by multivariate statistical analysis and machine learning tools. Again, there is much overlap between these two; methods from one are used in the other and vice-versa.

1.1.1.1 Multivariate statistical analysis Multivariate statistical methods are attractive because they efficiently handle complex system behavior and relationships between correlated process variables. Some multivariate statistical methods are principal component analysis (PCA), Fisher discriminant analysis (FDA), partial least-squares (PLS), and independent component analysis (ICA). All of these methods involve reducing a large set of system variables into a small set that still retains most of the useful information. Each method reduces the problem's dimension differently.

PCA projects the high dimensional variables onto the directions of the data that have the highest variance. This method is effective for data representation, but not data classification. The high variance directions of the data may not have any data class information. [4].

If classification is important for the given problem, then FDA may be used. FDA projects the high dimensional data onto a direction that preserves data classification information. As a result, FDA cannot reduce the dimension of the data as much as other methods [4].

PLS minimizes the distance between the high dimensional variables and their projection onto a lower dimension. PLS can handle multicolinearity among independent variables and is robust in the face of noisy or missing data. It is effective for data prediction but struggles with data interpretation [5].

ICA expresses a set of measured variables as a linear combination of statistically independent components. ICA can handle non-Gaussian data noise, unlike PCA. However, ICA cannot determine the ordering, sign, or exact amplitude of the components [6].

All of these methods use a specific metric of the lower dimensional projections to detect and diagnose the fault. Typically, the T^2 or squared prediction error (SPE) statistic is used. The fault is detected when one of the statistics pass some threshold.

When there is training data, these methods are very effective for fault detection. However, these methods require large amounts of data to train and build robust models, and gathering high quantities of experimental data may be costly. Additionally, these methods can struggle in the face of missing or anomalous data points. If a scenario occurs that is outside of the training data, a data-driven method may not be able to carry out effective FDD.

1.1.1.2 Machine learning methods Machine learning tools have come to the forefront of data-driven FDD due to the rapid increase in computational power over the last several years. Machine learning tools used in FDD include artificial neural networks (ANN), support vector machines (SVM), Gaussian mixture models (GMM), and k-nearest neighbors (KNN). Most machine learning algorithms use a combination of statistical analysis and optimization to build models that relate the inputs and outputs of the system. The models are "trained" offline, i.e., using data collected from previous system operation. Oftentimes, the data is preprocessed, then it is fed through the machine learning algorithm.

ANNs uses layers of calculation units that extract features from a data set. The networks are trained by adjusting weights associated with each calculation step. ANNs can learn complex input/output relationships and are generally regarded as easy to use. However, they may take a long time to train and can get trapped in local optimization minima [4].

SVM is a dimension reduction method that maximizes the distance between the lower dimensional projection and the closest data point to the projection. This method is very generalizable and its optimization function has no local minima. SVM tends to be slower and more computationally expensive than other methods [4].

GMM clustering data by assuming that it is sampled from multiple Gaussian distributions. Parameters of the distributions are typically found through the expectation-maximization algorithm. This method works well when the data is, in reality, sampled from multiple Gaussians [4].

KNN is similar to GMM in that it clusters data. This method does not assume that the data follows a model. It instead assumes that data points near to each other are of the same or similar classes. For each data point, k of its nearest neighbors are put into the same class. KNN is versatile and easy to implement, but becomes significantly slower as k increases [7].

During operation, the predicted outputs from the models are compared with the actual system outputs. Faults are detected when some statistic between the two, say, the error is greater than some threshold [8].

Being data-driven methods, these techniques suffer from the same drawbacks as mentioned before. Training data is necessary to build robust models and anomalous faults may not be able to be detected. This is where model-based methods become useful.

1.1.2 Model-based methods for FDD

In these methods, a mathematical system model is derived from first principles, historical data, or a combination of both. Initially, a model is derived for the unfaulted operating condition. Usually, observers are used on both the model and the system to produce an output. The output of the real system is compared to the output of the model. If the error between the outputs exceeds a threshold, then the fault is detected.

Having only an unfaulted model may not allow for the diagnosis of the severity of the fault. If possible, models are built for faulted operating conditions as well. Faulted conditions can be modelled when the model includes states or parameters that are directly affected by the fault. Faults can be modelled as additive or multiplicative. Once again, observers are

used to output values for the faulted models. The fault can then be diagnosed by evaluating the error signal between the real system and all of the models. Essentially, the model that is closest to the real system output is chosen. That model's states and parameters then become the most likely fault condition.

The major drawback of model-based methods is that an accurate system model is required. Models for complex systems can often be difficult to derive due to nonlinearities or lack of system knowledge.

However, model-based FDD can be very robust if the models are well-formulated. Good models can accurately estimate the systems states and parameters, allowing for faster and more accurate FDD [9].

1.1.3 Choosing appropriate FDD methods and hybrid FDD

Deciding which FDD method to use greatly depends on the nature of the system and the user's knowledge of the system. If a system deals directly with human life, a FDD method with poor diagnostic accuracy but quick detection time may be desirable so that safety measures can be taken as soon as possible. In a system where start-up and shut-down are expensive, a FDD method with high diagnostic accuracy may be desirable so that appropriate measures are taken to continue faulted operation. Often times it is most effective to combine several FDD methods into one, known as hybrid FDD. Hybrid FDD allows for the advantages of certain methods to alleviate the drawbacks of others.

1.1.4 Observers and state estimation in dynamic systems

As mentioned before, observers are generally used to produce outputs for systems and system models. Much research has been done in the area of building and selecting appropriate observers. A model-based FDD approach is the focus of this research, so we will now discuss this important tool.

Observers (or state estimators) are used when we would like to know the values of states that are not directly measurable. True values of the states may be obscured by sensor noise. Alternatively, there may be no sensor that can directly measure a state. Observers are given measurements as inputs and produce estimates of the states.

Kalman filters are a standard method of state estimation in linear dynamic systems with Gaussian noise. In these systems, Kalman filters provide optimal state estimates that minimize mean-squared error between the actual and estimated states. It starts by assuming that an estimate of the current state and state covariance are known. The state and state covariance of the next time step are predicted using the current state and covariance estimates, and the state model. A measurement is taken and compared to a predicted measurement which is generated with the predicted state and the measurement model. This is called the innovation. The predicted measurement covariance is calculated using the measurement model, predicted state covariance, and measurement uncertainty. The Kalman gain is calculated with the predicted state covariance, measurement model, and the predicted measurement covariance. This gain represents how much the final estimates will be adjusted by the new measurement information. The final state and state covariance estimates are found using the predicted state, Kalman gain, innovation, and predicted measurement covariance. These estimates are then used to predict values for the next step, and the process is repeated recursively.

The derivation of the standard Kalman filter assumes that the system is linear. When a system is nonlinear with Gaussian noise, extended Kalman filters (EKFs) are commonly used for generating state estimates. The EKF is identical to the Kalman filter, except when calculating the predicted state and measurement covariances. In these steps, a Taylor series expansion of the nonlinear system model about the previous state estimate is used. The linearization is necessary to ensure that the Gaussian PDFs remain Gaussian when propagating forward in time. The state estimate of the EKF is no longer optimal due to the linearization being an approximation. If the system model is outside of some bounds around the true system model, then the EKF estimates can diverge from the actual system. Divergence can also occur if the initial state estimate is far off because the linearization based on that point will not be an accurate approximation of the system model.

The unscented Kalman filter (UKF) has been developed as an improvement over the EKF that can handle nonlinear systems with non-Gaussian noise. The UKF uses the unscented transformation, by which we can calculate the statistics of the state that propagates forward in time through a nonlinear system model. The UKF samples carefully selected points from the previous state distribution, then passes them through the nonlinear model. These new points are then used to provide an approximation of the current state distribution. UKFs have become very popular for state estimating, but they can only accurately estimate the first and second order moments of a non-Guassian distribution. The UKF will introduce some error in the third-and-above order moments of a non-Guassian distribution [10].

1.1.4.1 Particle filters as observers To perform state estimation in systems with any amount of nonlinearity to any order of moment for non-Gaussian noise distribution, a method called particle filtering has been developed. Particle filtering has become popular due to many real systems having highly nonlinear models with non-Gaussian noise. Particle filtering is a Monte Carlo method, meaning it uses random sampling. It takes a number of random samples (particles) from a prior state distribution. It then propagates them forward in time, and assigns them weights depending on the new measurement. The weights represent how likely the state value of that particle is. It resamples the particles based on their weights, then propagates them forward in time, and so on.

For FDD, particle filtering provides some advantages over model-based FDD techniques that use any variant of the Kalman filter. Firstly, particle filters have been shown to provide accurate estimates for highly nonlinear and highly non-Gaussian systems.

Secondly, as discussed previously, there are model-based FDD techniques that require several models to be built for effective fault diagnosis. FDD that uses any variant of the Kalman filter falls into this category. Using certain techniques, particle filters only require one model for fault diagnosis (see Section 3.3). Using only one model reduces modeling effort on the part of the designer.

Although the system analyzed in this research is modelled as linear with Gaussian noise (see Appendix A), the particle filter was chosen because it is generalizable to nonlinear and non-Gaussian systems. Additionally, our proposed algorithm takes advantage of a particle filter's ability to diagnose faults with only one system model.

2.0 Particle Filtering

In this section, we will discuss the underlying concepts of particle filtering, such as Bayesian inference, importance sampling, and Monte Carlo. Then we will lay out the algorithm for the standard particle filtering and discuss issues in the design and implementation of particle filters.

2.1 Recursive Bayesian Inference

The goal of particle filtering is to carry out Bayesian inference. This is a method of statistical inference that uses Bayes Theorem; that is,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.1}$$

where A and B are some event. $P(A|B)$, $P(B|A)$, and $P(A)$ are called the posterior distribution, the likelihood, and the prior distribution, respectively. $P(B) = \int_A P(A)P(B|A)$ is a scaling term that ensures the expression sums to unity, i.e., it is a probability distribution. The prior is what we know about A before observing B and the likelihood is how likely we are to observe B given A. Using these two quantities, we can find the posterior which is what we know about A after observing B.

Bayesian inference is popular because it incorporates prior knowledge into its algorithm, unlike some other techniques that use only observations for inference. In many cases, the user will have some general knowledge of how a system works or where the initial conditions are. Using this knowledge to fortify the inference makes sense intuitively. Even if the user has no prior knowledge, a general guess can be used.

Bayesian inference can be used for estimating the probability density function (PDF) of the state of a dynamic system recursively. The Bayesian framework works well for this task because the state PDF at the previous time step can be used as the prior knowledge for the current time step.

To carry out Bayesian inference in a dynamic system, we consider a discrete-time system whose state vector evolves according to the model

$$\mathbf{x}_k = \mathbf{f}_{k-1}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1}) \tag{2.2}$$

where $\mathbf{f}_k$ is the state transition function and $\mathbf{w}_k$ is a zero-mean, white noise sequence that is independent and identically distributed. The PDF of $\mathbf{w}_k$ is assumed to be known. At each timestep k, measurements from the system become available. These measurements are related to the state vector by

$$\mathbf{z}_k = \mathbf{h}_k(\mathbf{x}_k, \mathbf{v}_k) \tag{2.3}$$

where $\mathbf{h}_k$ is the measurement function and $\mathbf{v}_k$ is a zero-mean, white noise sequence of known PDF. Here we assume that we know the initial PDF $p(\mathbf{x}_1)$ of the state and that we have the models $\mathbf{f}_k$ and $\mathbf{h}_k$ for $i = 1, ..., k$. The available information at time step k is the set of measurements $\mathbf{D}_k = \{\mathbf{z}_k : t = 1, ..., k\}$.

The goal of Bayesian inference in dynamic systems is to find the posterior PDF $p(\mathbf{x}_k|\mathbf{D}_k)$ of the state at the current time step k given the available information. Through Bayes Theorem, we can find the posterior PDF with the equation

$$p(\mathbf{x}_k|\mathbf{D}_k) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{D}_{k-1})}{p(\mathbf{z}_k|\mathbf{D}_{k-1})} \tag{2.4}$$

As in Bayes Theorem,

$$p(\mathbf{z}_k|\mathbf{D}_{k-1}) = \int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{D}_{k-1})d\mathbf{x}_k \tag{2.5}$$

The terms in the right-hand-side of 2.4 are found in two phases: prediction and correction. In the prediction phase, we use the Chapman-Kolmogorov identity to obtain the PDF

$$p(\mathbf{x}_k|\mathbf{D}_{k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{D}_{k-1}) \tag{2.6}$$

We assume that the prior PDF $p(\mathbf{x}_{k-1}|\mathbf{D}_{k-1})$ is known. If we are at the beginning of the simulation or operation, this distribution will be based on some initial guess of the state.

The prior PDF is distribution of the previous state given all past information. $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ is the probabilistic form of the state transition model,

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}) \equiv \mathbf{f}_{k-1}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1}) \tag{2.7}$$

so it is known. In this first phase, we are predicting where the state will be given the previous available information.

In the correction phase, a measurement $\mathbf{z}_k$ becomes available at time step k. Along with the observation function, we use the measurement to find $p(\mathbf{z}_k|\mathbf{x}_k)$, the likelihood of seeing the measurement given the state. $p(\mathbf{z}_k|\mathbf{x}_k)$ is the probabilistic version of the observation equation,

$$p(\mathbf{z}_k|\mathbf{x}_k) \equiv \mathbf{h}_k(\mathbf{x}_k, \mathbf{v}_k) \tag{2.8}$$

so it is known. In this second phase, we are updating our prediction using the measurement we just received.

In dynamic systems that have linear models with Gaussian noise, the Kalman filter is used as the way of solving 2.4. There are many systems of interest that are nonlinear with non-Gaussian noise, and the integral in 2.5 may be very complex and analytically intractable. Particle filtering can provide an approximation of 2.4 in these types of systems using two special techniques. These techniques are called importance sampling and Monte Carlo.

2.2 Importance Sampling and Monte Carlo

Monte Carlo techniques use random sampling to give approximate answers to problems that are analytically unsolvable. In this case, the goal is to approximate $p(\mathbf{x}_k|\mathbf{D}_k)$ because it involves integrals that may be intractable. The Monte Carlo principle for approximating a probability distribution is

$$p(\mathbf{x}_k|\mathbf{D}_k) \approx \sum_{i=1}^{N} w_k(i)\delta(\mathbf{x}_k - \mathbf{x}_k(i)) \tag{2.9}$$

where $\delta(\mathbf{x}_k - \mathbf{x}_k(i))$ is the Dirac delta function at $(\mathbf{x}_k - \mathbf{x}_k(i))$. $\mathbf{x}_k(i)$ is a random sample from the state space. $w_k(i)$ is a weight. Essentially, the right-hand-side is a probability mass function approximating the posterior $p(\mathbf{x}_k|\mathbf{D}_k)$.

The questions now arise: what distribution over the state space do we draw samples from, and where does the weight come from? Importance sampling gives a solution to both of these.

We introduce a probability distribution $q(\mathbf{x}_k|\mathbf{D}_k)$ from which we sample $\mathbf{x}_k(i)$. This distribution is generally called the importance or proposal distribution and is chosen by the user. The weight is calculated by

$$w_k(i) = \frac{p(\mathbf{x}_k(i)|\mathbf{D}_k)}{q(\mathbf{x}_k(i)|\mathbf{D}_k)} \tag{2.10}$$

Intuitively, this is a measure of how close q is to the posterior. However, this weight still includes the posterior. So, we substitute Equation 2.4 into 2.10 to get

$$w_k(i) = \frac{p(\mathbf{z}_k|\mathbf{x}_k(i))p(\mathbf{x}_k(i)|\mathbf{D}_{k-1})}{p(\mathbf{z}_k|\mathbf{D}_{k-1})q(\mathbf{x}_k(i)|\mathbf{D}_k)} = \frac{p(\mathbf{z}_k|\mathbf{x}_k(i))p(\mathbf{x}_k(i)|\mathbf{x}_{k-1}(i))p(\mathbf{x}_{k-1}(i)|\mathbf{D}_{k-1})}{p(\mathbf{z}_k|\mathbf{D}_{k-1})q(\mathbf{x}_k(i)|\mathbf{D}_k)} \tag{2.11}$$

All terms on the right-hand-side are known except for $p(\mathbf{z}_k|\mathbf{D}_{k-1})$, the intractable normalization term. We disregard this term because each weight can be normalized by the sum of the weights. So

$$w_k(i) \propto \tilde{w}_k(i) = \frac{p(\mathbf{z}_k|\mathbf{x}_k(i))p(\mathbf{x}_k(i)|\mathbf{x}_{k-1}(i))p(\mathbf{x}_{k-1}(i)|\mathbf{D}_{k-1})}{q(\mathbf{x}_k(i)|\mathbf{D}_k)} \tag{2.12}$$

The proposal distribution can be expanded

$$q(\mathbf{x}_k(i)|\mathbf{D}_k) = q(\mathbf{x}_{k-1}(i)|\mathbf{D}_{k-1})q(\mathbf{x}_k(i)|\mathbf{x}_{k-1}(i), \mathbf{D}_k) \tag{2.13}$$

The term $q(\mathbf{x}_k(i)|\mathbf{x}_{k-1}(i), \mathbf{D}_k)$ represents the probability of transitioning from the state $\mathbf{x}_{k-1}(i)$ to the state $\mathbf{x}_k(i)$ according to the importance distribution. Substituting Equation 2.13 into 2.12 gives

$$\tilde{w}_k(i) = \left(\frac{p(\mathbf{z}_k|\mathbf{x}_k(i))p(\mathbf{x}_k(i)|\mathbf{x}_{k-1}(i))}{q(\mathbf{x}_k(i)|\mathbf{x}_{k-1}(i), \mathbf{D}_k)}\right)\left(\frac{p(\mathbf{x}_{k-1}(i)|\mathbf{D}_{k-1})}{q(\mathbf{x}_{k-1}(i)|\mathbf{D}_{k-1})}\right) \tag{2.14}$$

The second fraction on the right-hand-side is simply the weight from the previous time step, so

$$\tilde{w}_k(i) = \frac{p(\mathbf{z}_k|\mathbf{x}_k(i))p(\mathbf{x}_k(i)|\mathbf{x}_{k-1}(i)}{q(\mathbf{x}_k(i)|\mathbf{x}_{k-1}(i), \mathbf{D}_k)} w_{k-1}(i) \tag{2.15}$$

The weights are then normalized

$$w_k(i) = \frac{\tilde{w}(i)}{\sum_{i=1}^{N} \tilde{w}(i)} \tag{2.16}$$

2.2.1 Choosing importance distribution

The final issue is to choose the distribution $q(\mathbf{x}_k(i)|\mathbf{x}_{k-1}(i), \mathbf{D}_k)$. Ideally, we want it to be as close to the posterior as possible, but also straightforward to sample from. The most common choice of importance distribution is the state transition function $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ because it easy to implement and the weight update simplifies to

$$\tilde{w}_k(i) = p(\mathbf{z}_k|\mathbf{x}_k(i))w_{k-1}(i) \tag{2.17}$$

Intuitively, the importance distribution projects the samples to the next time step, so the state transition function makes sense. The state transition function is used as the importance distribution in this research.

2.2.2 Resampling

One issue that arises when carrying out importance sampling sequentially is called sample degeneracy. After several time steps, only a few samples have significant weights, and the remaining samples do not contribute much to the estimation of the posterior. So, a resampling step is introduced to help mitigate this problem. Before progressing to the next time step, the samples are sampled again according to their weights. This ensures that samples with high weights are replicated and forces the samples into areas of high likelihood. After resampling, the particle weights are all set to $w_k(i) = \frac{1}{N}$. This is done because resampling translates particle weights into particle frequencies. Resampling makes the multiplication

by the previous weight in 2.17 unnecessary because the term $\frac{1}{N}$ will be cancelled in the normalization step. Thus the weight calculation becomes just the measurement likelihood

$$\tilde{w}_k(i) = p(\mathbf{z}_k|\mathbf{x}_k(i)) \tag{2.18}$$

Note that this is assuming resampling occurs at every time step (see Section 2.3.1).

Particle filtering is a sequential implementation of Monte Carlo and importance sampling with the goal of Bayesian inference, i.e, the estimation of the PDF of the state at the current time.

2.3 Standard Particle Filtering Algorithm

In accordance with Bayesian inference, particle filtering has a prediction and update step. To start, we randomly sample N times from the PDF $p(\mathbf{x}_{k-1}|\mathbf{D}_{k-1})$. If it is the first time step, $p(\mathbf{x}_{k-1}|\mathbf{D}_{k-1}) = p(\mathbf{x}_1)$, which is some initial guess of a distribution. If not, these samples will be the resampled particles from the previous time step. These samples are called state "particles" $\mathbf{x}_k(i)$ where $i = 1, ..., N$ is the particle index. We want to propagate the state values forward in time by sampling from the importance distribution. Recall that the importance distribution we have chosen is $q(\mathbf{x}_k(i)|\mathbf{x}_{k-1}(i), \mathbf{D}_k) = p(\mathbf{x}_k|\mathbf{x}_{k-1}) \equiv \mathbf{f}_{k-1}(\mathbf{x}_{k-1}(i), \mathbf{w}_{k-1}(i))$. So, we passed through the state transition function to produce a new set of particles

$$\mathbf{x}_k(i) = \mathbf{f}_{k-1}(\mathbf{x}_{k-1}(i), \mathbf{w}_{k-1}(i)) \tag{2.19}$$

In this case, passing the particles through the state transition function is the equivalent of sampling from the importance distribution. This is the prediction phase.

We pass the particles through the measurement function without noise to produce measurement particles

$$\mathbf{z}_k(i) = \mathbf{h}_k(\mathbf{x}_k(i)) \tag{2.20}$$

Given a particle's state value, this is where the corresponding measurement would lie, and these measurement particles are a projection into the measurement space. We do not include the measurement noise $\mathbf{v}_{k-1}$ because it is introduced in the next step.

After making a new measurement $\mathbf{z}_k$, we evaluate the likelihood of each observation particle given the measurement, $p(\mathbf{z}_k|\mathbf{z}_k(i))$. This is the distribution of the measurement noise $\mathbf{v}$, with mean $\mathbf{z}_k$ evaluated at $\mathbf{z}_k(i)$. This is the update phase. To begin building the posterior distribution of the state, we normalize the likelihood to produce a weight for each $\mathbf{x}_k(i)$ state particle

$$w_k(i) = \frac{p(\mathbf{z}_k|\mathbf{z}_k(i))}{\sum_{i=1}^{N} p(\mathbf{z}_k|\mathbf{z}_k(i))} \tag{2.21}$$

Normalization is necessary because the sum of the particle likelihoods may to sum to one. The state particles $\mathbf{x}_k(i)$ and their weights $w_k(i)$ form a discrete distribution that is an approximation of $p(\mathbf{x}_k|\mathbf{D}_k)$. Each particle now contain values for the state of the system and a single weight; this is denoted

$$\tilde{x}_k(i) = \{w_k(i), \mathbf{x}_k(i)\}. \tag{2.22}$$

and the set of N particles is denoted

$$\tilde{\mathbf{x}}_k = \{\tilde{x}_k(1), \tilde{x}_k(2), ..., \tilde{x}_k(N)\} \tag{2.23}$$

One common way of generating a state estimate from the distribution $p(\mathbf{x}_k|\mathbf{D}_k)$ is by taking its expected value

$$\hat{\mathbf{x}}_k = E\{x_k\} \approx \sum_{i=1}^{N} w_k(i)\mathbf{x}_k(i) \tag{2.24}$$

Alternatively, an estimate could be drawn from the median of the particles or from the particle with the highest weight.

The distribution $p(\mathbf{x}_k|\mathbf{D}_k)$ is sampled N times in the resampling step. The most basic way of resampling is to sample the particles with probability equal to their weight, with replacement; this is called multinomial resampling. Particles with the highest weights are most likely to be replicated. The weights of the resampled particles are set to $\frac{1}{N}$.

The system shifts forward by one time step and the resampled state particles are now considered samples from the prior state distribution. The particles are passed through the state transition function once again, and the process repeats recursively. The method described above is deemed the bootstrap particle filter. The term "bootstrap" is used because bootstrapping is any method of random sampling that includes replacement. We will herein

refer to this algorithm as the standard particle filter. Algorithm 1 shows the particle filtering algorithm in its entirety. Figure 1 shows a visual representation of the particle filtering algorithm.

2.3.1 Resampling techniques in particle filtering

Resampling provides particle filtering with long-term robustness because it keeps particles in areas of high likelihood. However, it introduces a new issue. Particles with low likelihoods are eliminated, limiting the amount of diverse information in our particle system. The particles are no longer identically and independently distributed because many of them are duplicated. This problem is not as significant as sample degeneracy, and resampling has been shown to be practically beneficial in nearly all applications.

Resampling can be performed at whatever frequency the user chooses. A popular way of determining when to resample is by evaluating the variance of the unnormalized particle weights. If the variance of the unnormalized particle weights is low, that means all of the particles are in the neighborhood of high likelihood; thus, resampling may not be necessary [11], [12].

Several different methods of resampling have been developed. The three most common are multinomial, stratified, and systematic. All methods are, in some way, based on the cumulative sum of the particle weights.

In multinomial resampling, the particles are resampled with probability equal to their weight. In other words, the cumulative sum of the weights is searched through purely randomly. With this method, the maximum number of times a particle can be resampled is N, and the minimum number of times a particle can be resampled is 0. As a result, this method is more likely to produce a set of particles with high variance. Also, this method has a comparatively high computational complexity. However, it is extremely easy to implement.

Algorithm 1 Standard particle filter

Given: $\mathbf{f}_k, \mathbf{h}_k, p(\mathbf{x}_1), p(\mathbf{z}_k)$

for $k \geq 1$ **do**

if $k = 1$ **then**

for $i = 1, ..., N$ **do**

Initialize particles:

Sample particle state values $\mathbf{x}_1(i) \sim p(\mathbf{x}_1)$

Collect measurement $\mathbf{z_1}$

Calculate measurement particles $\mathbf{z}_1(i) = \mathbf{h}_1(\mathbf{x}_1(i))$

Calculate weights $w_1(i) = \frac{p(\mathbf{z}_1|\mathbf{z}_1(i))}{\sum_{i=1}^{N} p(\mathbf{z}_1|\mathbf{z}_1(i))}$

Form particle $\tilde{x}_1(i) = \{\mathbf{x}_1(i), w_1(i)\}$

end for

Set of particles $\tilde{\mathbf{x}}_1$ form approximation of posterior state distribution $p(\mathbf{x}_1|\mathbf{D}_1)$

Calculate state estimate $\hat{\mathbf{x}}_1 \approx \sum_{i=1}^{N} w_1(i)\mathbf{x}_1(i)$

Resample state particles with probabilities = weights $w_1(i)$ to generate new set of N particles

else

for $i = 1, ..., N$ **do**

Move particles forward in time $\mathbf{x}_k(i) = \mathbf{f}_{k-1}(\mathbf{x}_{k-1}(i), \mathbf{w}_{k-1}(i))$

Collect measurement $\mathbf{z_k}$

Calculate measurement particles $\mathbf{z}_k(i) = \mathbf{h}_k(\mathbf{x}_k(i))$

Calculate weights $w_k(i) = \frac{p(\mathbf{z}_k|\mathbf{z}_k(i))}{\sum_{i=1}^{N} p(\mathbf{z}_k|\mathbf{z}_k(i))}$

Form particle $\tilde{x}_k(i) = \{\mathbf{x}_k(i), w_k(i)\}$

end for

Set of particles $\tilde{\mathbf{x}}_k$ form approximation of posterior state distribution $p(\mathbf{x}_k|\mathbf{D}_k)$

Calculate state estimate $\hat{\mathbf{x}}_k = \sum_{i=1}^{N} w_k(i)\mathbf{x}_k(i)$

Resample state particles with probabilities = weights $w_k(i)$ to generate new set of N particles

end if

end for

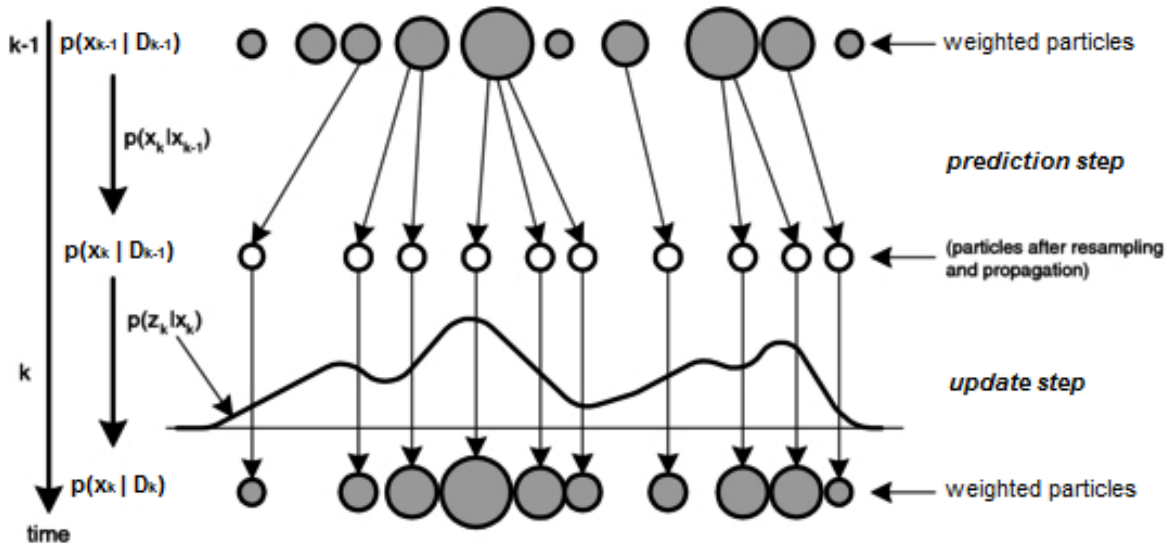

Figure 1: Diagram of the recursive particle filtering algorithm. Starting from a set of weighted particles at time $k-1$, we resample according to the weights. The particles are propagated forward in time in the prediction step. The arrows in the prediction step show how many times each particle was duplicated. In the update step, particles are given weights according to the measurement at time k. The process is then repeated recursively. **Source:** taken from [2] and modified for this paper.

In stratified resampling, the cumulative sum of the weights is divided into N subintervals. The subintervals are then each sampled once. The upper and lower limits that the ith particle can be resampled are $\text{floor}(Nw_k(i)) + 2$ and $\max(\text{floor}(Nw_k(i)) - 1, 0)$. Thus, this method produces a lower particle variance than multinomial. Its computational complexity is also lower than that of multinomial because it does not require use of the random number generator. It is slightly more difficult to implement than multinomial.

Systematic resampling initially searches the cumulative sum of the weights purely randomly in the interval $(0, \frac{1}{N}]$, then each successive sampling is a constant deterministic jump from the previous sample. The upper and lower limits for this method are $\text{floor}(Nw_k(i)) + 1$ and $\text{floor}(Nw_k(i))$, producing a lower particle variance than stratified resampling. Systematic resampling has a computational complexity equal to that of stratified resampling [13].

For this research, we are using the most basic implementation of resampling, i.e, multinomial resampling at every time step.

2.3.2 Number of particles

The number of particles N chosen by the user is a significant step in the design of a particle filter. It can be shown that as the number of particles approaches infinity, the particle filter becomes optimal in its estimation of the state PDF. In other words, the the approximation will exactly equal the real state distribution when the number of particles tends to infinity.

It has been claimed that the number of particles needs to increase exponentially with the dimension of the problem to maintain a good approximation of the state distribution [14], [15], [16]. The dimension is how many different states and measurement variables there are. Having more measurement variables will make our belief in the measurement stronger, because we are getting information from multiple different sources; thus, the measurement likelihood becomes more and more narrow. So, if the number of particles is relatively low, only a few particles will end up in areas of high likelihood. Thus, a few particles will have significant weight, and the problem becomes degenerate.

The primary tradeoff is that the computational load will become more and more significant as the number of particles increases. Generally, measurements become available at regular intervals and need to be processed before the next measurement is received. So, a logical choice would be to increase number of particles until the particle filter computation time is equal to the time between measurements. In many systems, however, computational power is shared among many subsystems. Thus, the number of particle will ultimately depend on the nature of the system and which subsystems are most important.

2.3.3 Advanced particle filtering techniques

There are several advanced particle filtering techniques that can further mitigate sample degeneracy. For the purposes of this research, we will not integrate any of these methods; however, we feel it is important to mention some of these techniques.

The auxiliary particle filter (APF) is an algorithm that has been shown to perform better than the standard particle filter. It aims to "look ahead" by trying to predict which particles will be in regions of high probability mass at the next time step. The particle weights are computed using the predictive PDF $p(z_{k+1}|x_k)$. An approximation of this PDF can be used if the analytical form is not known. The APF has shown general improvement in performance over the standard particle filter [12].

Sample degeneracy has been further addressed through the use of the resample-move method. In this straightforward method, the particles are "jittered" by random amounts after the resampling step. The jittering introduces bits of new information to the particles to aid in diversity [12].

An approach called block sampling has been introduced that further addresses this. Block sampling considers the history of particle from a fixed lag L to the current time. This collection of particles is weighted according to the measurement likelihood. Then the history of particles is resampled according to the weights to produce the next set of N particles. Sampling from the history of particles allows for more diverse information to be introduced [12].

3.0 Particle Filtering for Fault Detection and Diagnosis

There are several model-based methods that use particle filters as observers for FDD [17], [18], [19]. Here we will cover a few popular techniques including multiple-model particle filtering, interacting multiple-model particle filtering, and state-augmented random walk particle filtering.

3.1 Multiple-model Particle Filtering

A popular way to detect faults using model-based techniques is the multiple-model approach. In these methods, a model is developed for each fault condition of concern

$$\mathbf{x}_k^{(j)} = \mathbf{f}_{k-1}^{(j)}(\mathbf{x}_{k-1}^{(j)}, \theta^{(j)}, \mathbf{w}_{k-1}^{(j)}) \tag{3.1}$$

$$\mathbf{z}_k^{(j)} = \mathbf{h}_k^{(j)}(\mathbf{x}_k^{(j)}, \theta^{(j)}, \mathbf{v}_k^{(j)}) \tag{3.2}$$

where $j = 1, ..., J$ is the index of the model (or operating mode). $\theta^{(j)}$ are any parameters that are indicative of a fault. Usually, $j = 1$ is the nominal, or unfaulted, model. The rest of the models contain some variation that is indicative of a fault. The assumed fault parameters for each model $\theta^{(j)}$ are constant in time.

A particle filter with N particles is used on each model, as described in 2.3. The particle filters are run in parallel, and produce a state estimate $\hat{\mathbf{x}}_k^{(j)}$ for each of the J models. Each state estimate is an estimate of where the state value may be if the jth fault has occurred.

The likelihood of each state estimate is calculated and then used to determine the probability of each model. There are several methods for calculating model probabilities, but the most straight-forward is Bayesian hypothesis testing (BHT). BHT assumes knowledge of a prior probability for each mode and calculates the current probability of the jth mode H^j as

$$P(H^j|\mathbf{D}_k) = \frac{P(\mathbf{z}_k|\mathbf{D}_{k-1}, H^j)P(H^j|\mathbf{D}_{k-1})}{\sum_{j=1}^{J} P(\mathbf{z}_k|\mathbf{D}_{k-1}, H^j)P(H^j|\mathbf{D}_{k-1})} \tag{3.3}$$

where $P(\mathbf{z}_k|\mathbf{D}_{k-1}, H^j)$ is the likelihood of the measurement given the jth model. $P(H^j|\mathbf{D}_{k-1})$ is the prior probability of the jth model [1]. Particle filtering provides a convenient approximation for the measurement likelihood given the model

$$P(\mathbf{z}_k|\mathbf{D}_{k-1}, H^j) \approx \frac{1}{N}\sum_{i=1}^{N} p(\mathbf{z}_k|\mathbf{z}_k^{(j)}(i)) \tag{3.4}$$

where $p(\mathbf{z}_k|\mathbf{z}_k^{(j)}(i))$ is the distribution of the measurement noise, with mean $\mathbf{z}_k$ evaluated at $\mathbf{z}_k^{(j)}(i)$. Recall that $p(\mathbf{z}_k|\mathbf{z}_k^{(j)}(i))$ is the numerator of Equation 2.21 for each of the J models. This numerator is the unnormalized particle weight. So, the measurement likelihood in Equation 3.4 is approximated by the average of the unnormalized particle weights [19].

The result of BHT is a set of J model probabilities, i.e., how likely each faulted condition is. These probabilities are then used to find the expected value of the states over all J models

$$\hat{\mathbf{x}}_{k,mix} = \sum_{j=1}^{J} \hat{\mathbf{x}}_k^{(j)} P(H^j|\mathbf{D}_k) \tag{3.5}$$

$$= \sum_{j=1}^{J}\sum_{i=1}^{N} \mathbf{x}_k^{(j)}(i) w_k^{(j)}(i) P(H^j|\mathbf{D}_k) \tag{3.6}$$

$$= \sum_{j=1}^{J}\sum_{i=1}^{N} \mathbf{x}_k^{(j)}(i) P(\mathbf{x}_k(i)|H^j) P(H^j|\mathbf{D}_k) \tag{3.7}$$

The expected value of the fault parameters can be found in a similar way

$$\hat{\theta}_k = \sum_{j=1}^{J} \theta^{(j)} P(H^j|\mathbf{D}_k) \tag{3.8}$$

If the model probability $P(H^j|\mathbf{D}_k)$ is low, then the state estimate produced by that model is given low weightage in this expected value calculation. Likewise for the fault parameters. A diagram of this algorithm is shown in Figure 2.

The multiple-model fault detection method suffers from numerical issues when the magnitudes of the faults exceed a problem specific threshold. Recall that the weighting and resampling phases of the particle filter work to keep the particles of each model close to the actual sensor measurement. When the system is at steady-state, each model causes its estimation to move toward a steady state that is valid for the model and sensor measurements. In some situations, where the faulted model is sufficiently different from the true model, the

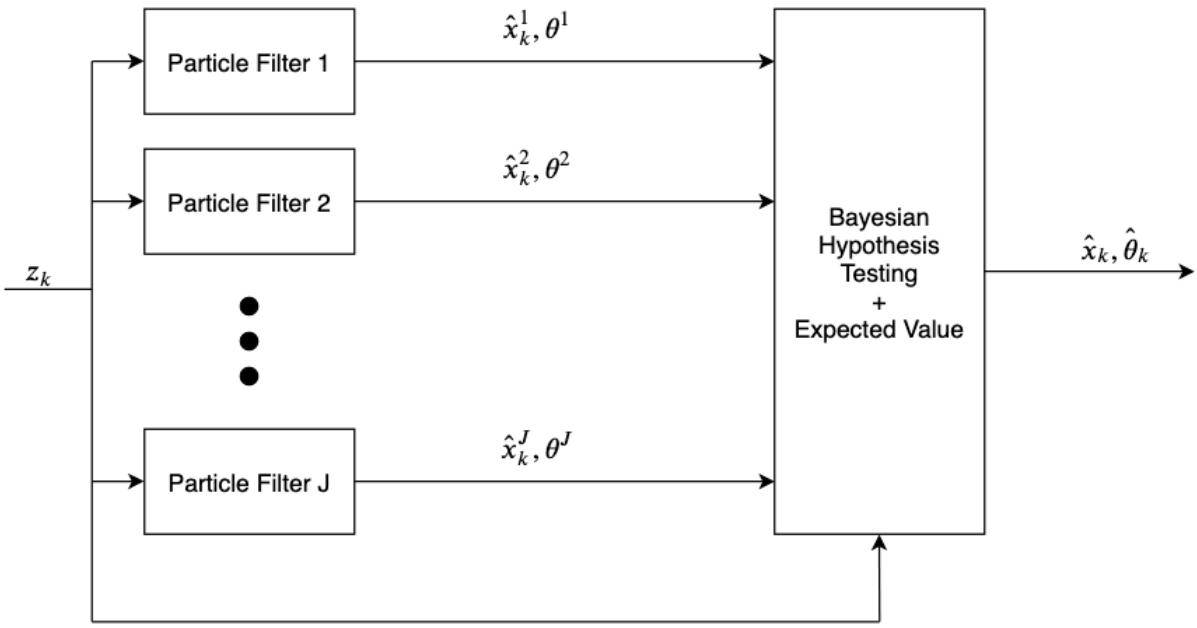

Figure 2: Diagram of the multiple-model particle filtering FDD algorithm. Each particle filter takes in the measurement and generates a state estimate and the measurement likelihood. The likelihood is sent to the BHT to calculate the model probability. The particle filters' state estimates and fault parameter values are used with the model probabilities to calculate the overall expected value of the state and fault parameters.

state value of the faulted model will drift from the true state estimate. The assumed fault may pull the estimation away from the measurement with greater "force" than the weighting and resampling. This can cause the models with large fault magnitudes to completely diverge from the actual measurement. When a fault occurs, the probabilities of those divergent models will be calculated as effectively zero because they are so far away from the actual measurement. This happens even if the real fault magnitude matches the magnitude of one of those divergent models. This drawback is demonstrated and discussed in Section 4.2.

3.2 Interacting Multiple-model Particle Filtering

One way to solve the divergence issue is through the use of the interacting multiple-model (IMM) framework. Conventional IMM methods involve treating the system as a hybrid one. A hybrid system is one whose state vector contains both continuous and discrete values. The discrete value is the system operational mode $H \in \{H^1, H^2, ..., H^J\}$. These are the hypotheses that the jth model has occured. The continuous values are those of the physical states of the system. The FDD problem then becomes that of estimating both the discrete and continuous states of the system that will result in the PDF $p(\mathbf{x}_k, H|\mathbf{D}_k)$. IMM particle filtering (IMMPF) methods have been shown to be very effective for FDD [20], [18].

However, standard IMMPF requires that a matrix of transition probabilities between pairs of modes is known [18]. This is a $J \times J$ matrix where the (m, n)th element is the probability of transitioning from fault m to fault n. The transition probability is necessary for the "interaction" phase of the IMM algorithm, when the model estimates are mixed together. Existing IMMPF research examines systems with a relatively small number of modes. For systems with a large number of modes, determining this matrix can be a difficult task.

An IMMPF method that does not require a transition matrix is presented in [17]. This method is identical to the standard multiple-model method, but introduces a re-initialization step to the multiple-model approach described previously. If the particle weights for a given model become small, then that model's probability will also become small. If the probability

$P(H^j|\mathbf{D}_k)$ falls below a certain threshold, then the particle state values of that model are all re-initialized

$$\mathbf{x}_k^{(j)}(i) = \hat{\mathbf{x}}_{k-1,mix} + \mathbf{w}_k^{(j)}(i), \quad i = 1, ..., N \tag{3.9}$$

That is, each state value of the particle is set to the expected value from Equation 3.8 plus a sample from the process noise. If the model probability is above the threshold, then the particles remain unaltered during this step. Re-initialization ensures that the state estimates for a given model will not diverge from the actual measurement. This step is executed at the beginning of every time step. The rest of the multiple-model algorithm is then carried out normally. This method is tested and discussed in Section 4.3.

One issue with multiple-model and IMMPF methods is that they assume the system will operate according to a discrete set of modes. Choosing which modes to consider and how many particles to allocate to each mode may be a cumbersome task. In reality, many systems can have faults that take on a continuous set of values over a certain range. One way to allow the fault to have continuous value is through a "random walk" procedure, which uses random sampling.

3.3 State-augmented Particle Filter with Random Walk

The state vector is augmented with any parameters θ that describe faults in the system. The parameters are then propagated through time with artificial additive noise and the state transition model becomes

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{g}(\mathbf{x}_k, \theta_{k+1}, \mathbf{w}_k) \\ \theta_k + \mathbf{w}_{rw,k} \end{bmatrix} \tag{3.10}$$

where $\mathbf{w}_{rw,k}$ is the user-chosen noise. This noise is the random walk component of the algorithm; the particles will be randomly walking through their space. A particle filter is used on this model to produce the estimate for both the states and the parameters [21]. Each particle will now have values for the states, the parameters, and a single weight.

In this framework, the parameters are propagated first, then the resulting states are calculated. This makes sense intuitively, because we want to see how the fault parameter value will affect the state. This is fundamentally the same as a multiple-model particle filtering, but now each particle is essentially its own model; each particle has a unique fault parameter value.

Since we obviously cannot measure the parameters directly, the weight of each particle is still calculated according to the measurable states as in (2.21). However, the parameter value will affect the particle's state value, thus affecting the weight indirectly. The estimate of the parameters is the weighted sum

$$\hat{\theta}_k = \sum_{i=1}^{N} w_k(i)\theta_k(i) \tag{3.11}$$

The estimated parameter values will indicate whether or not a fault has occurred and how severe the fault is.

3.3.1 Choosing random walk distribution

As discussed previously, this method allows the particles to propagate with only one system model. All that needs to be done is the selection of the noise distribution according to which the particle fault values will propagate. It could be argued that choosing a distribution requires less effort than having a distinct model for each fault value of concern. In some sense, the choosing of the distinct models has been automated by the noise distribution of the fault parameters.

The fault parameter values of each particle will explore their space depending on the choice of artificial noise distribution of $\mathbf{w}_{rw,k}$. Choosing this distribution is dependent on the nature of the system. A popular initial choice is the Gaussian distribution with zero mean. If the covariance of the distribution is large, the particles will be exploring a wider range of values and will respond faster to a fault. However, the accuracy of the detection will deteriorate. If the covariance is small, the accuracy will improve, but the response will be slow. Thus, using a Gaussian-only random walk may not be give us the best performance for FDD.

The contribution of this research in this document is to introduce a "mutation" mechanism to the random walk. With a certain mutation probability p_{mut}, a particle's fault values will be sampled from a mutation distribution, and the values of the remaining particles will be sampled from a base distribution. The mutation distribution will be chosen to explore a large portion of the fault space, and the base distribution will be chosen to explore the neighborhood of the current fault estimate.

The user-chosen p_{mut} will allow us to control how many particles will mutate. Essentially, it will be a weighting of how much we want to explore the fault parameter space. Choosing the mutation distribution will depend on system. Ideally, it would be a distribution that can span all fault parameter values of concern for the given system.

For the purposes of this research, a uniform distribution is used as a natural choice for mutation distribution. The uniform distribution will not favor any particular area of the fault space and will explore its range purely randomly. If a certain area of the fault space is important to the user or operator, then a more peaked distribution could be used in that area. For example, a Gaussian with a large covariance centered on the area of concern.

Particles that are not chosen for mutation will have their parameter values propagated according to a zero-mean Gaussian distribution, as in the standard random walk procedure. The covariance of the Gaussian will be very small compared to the range of the uniform distribution. The Gaussian distribution will allow the particles to detect smaller faults, or small changes in the current fault. We want the highest density of particles to be in the area of the current estimate

The idea is that it is likely that some particles will have parameter values in the neighborhood of the real fault value when it occurs, due to the mutation. Those particles will receive high weights, and thus the fault will be detected and diagnosed quickly. The entire proposed algorithm, deemed the state-augmented mutating particle filter, is shown in Algorithm 2.

Determining p_{mut} is dependent on the system. If the mutation probability is too large, then the accuracy of the estimation will be poor because there will be more particles away from the actual fault value. If it is too small, there may not be enough particles searching the desired fault parameter space. This trade-off is explored in Section 4.4. Considerations for when to decide that the fault has occurred is also discussed in Section 4.4.

Algorithm 2 State-augmented mutating particle filter

for $k \geq 1$ **do**

if $k = 1$ **then**

for $i = 1, ..., N$ **do**

Initialize particles:

Sample $\theta_1(i) \sim \mathcal{N}(\theta_1, \sigma^2_{\theta_1})$

Sample $\mathbf{x}_1(i) \sim \mathcal{N}(\mathbf{x}_1, \sigma^2_{x_1})$

Calculate weight $w_1(i)$

Calculate expected values $\hat{\mathbf{x}}_1$ and $\hat{\theta}_1$

end for

Resample particles according to weights to generate new set of N particles

else

for $i = 1, ..., N$ **do**

if $RAND \sim U(0, 1) \leq p_{mut}$ **then**

Sample $\theta_k(i) \sim U(0, \theta_{max})$

else

Sample $\theta_k(i) \sim \mathcal{N}(\theta_{k-1}(i), \sigma^2_{\theta})$

end if

Calculate $\mathbf{x}_k(i) = g(\mathbf{x}_{k-1}(i), \theta_k(i), \mathbf{w}_k)$

Calculate weight $w_k(i)$

Calculate expected values $\hat{\mathbf{x}}_k$ and $\hat{\theta}_k$

end for

Resample particles according to weights to generate new set of N particles

end if

end for

This method allows for simpler design of the FDD algorithm. As stated previously, the designer will not need to determine which exact models to track, and how many particles to allocate to each model. At the same time, this method introduces additional flexibility to the design. The designer can choose p_{mut} and the number of particles to best suit the system.

4.0 Results and Discussion

The system used for testing the performance of the particle filtering FDD methods is the pressurizer of a nuclear pressurized water reactor (PWR). In PWRs, the pressurizer is a component used to regulate the pressure of the coolant (water) flowing through the primary coolant loop. A diagram of the primary loop is shown in Figure 3. Water is pumped through the reactor core, cooling it and carrying its heat to the steam generator. In the steam generator, heat from the primary coolant is transferred to cooler water in the secondary loop. Primary coolant is pumped back through the core and so on. The pressurizer ensures that the water in the primary loop remains subcooled.

The fault of interest is a small-break loss of coolant accident (LOCA) in the primary loop, i.e., a water leak. A small-break LOCA affects the coolant volume of the primary loop of a nuclear reactor. Thus, the pressurizer was chosen as our system of interest because its liquid level is an indication of the coolant volume. Inside the pressurizer is a combination of liquid and steam. A liquid level sensor measures the portion that is liquid. The pressurizer level is controlled by inserting or extracting coolant through the charging and letdown flows, respectively. Pressurizer level is measured in percent of maximum capacity. Small-break LOCAs are measured in percent, where 100% is the full break of a pipe that is 4.5 inches in diameter. The derivation of the pressurizer model can be found in Appendix A.

4.1 Particle Filtering Results for Estimating Pressurizer Level

The standard particle filter was tested on pressurizer level data from a PWR simulator to demonstrate its performance. The simulator was developed by GSE Systems and models a 970 MWe, three-loop, generic pressurized water reactor (GPWR). This simulator is used to train real NPP operators.

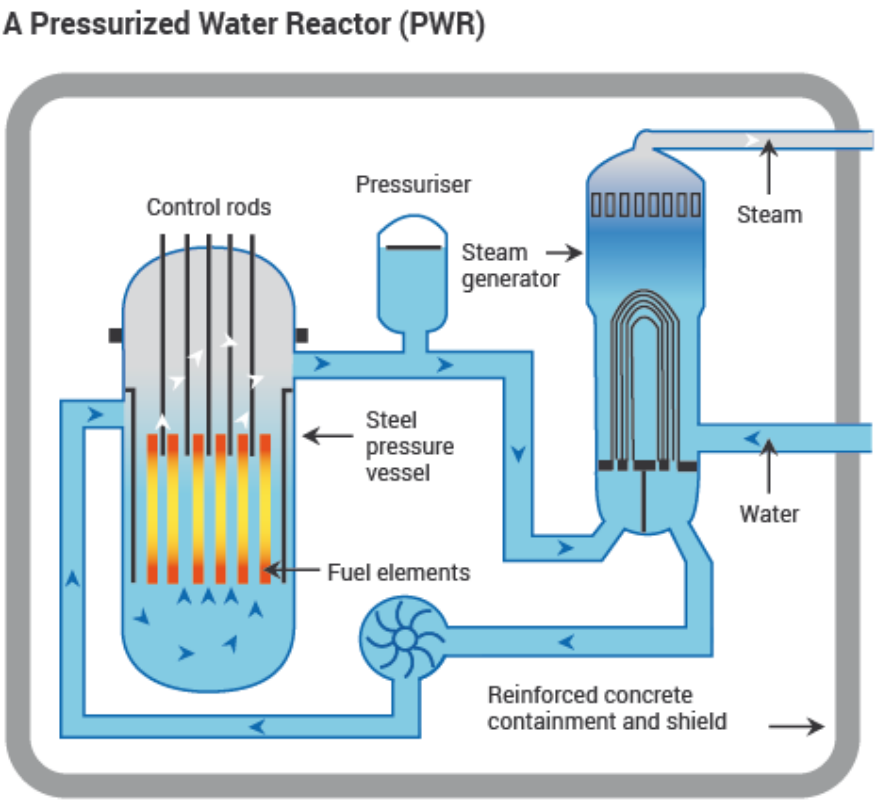

Figure 3: Diagram of the primary coolant loop of a nuclear pressurized water reactor. **Source:** taken from [3]

NPP power output is measured in percent of maximum capacity. The data was taken during a power change from 100% to 75% to 100% over a timespan of roughly 1.6 hours. This power change causes large fluctuations in pressurizer level, so it will be a good test for the particle filter algorithm. Results are shown in Figure 4.

Multinomial resampling was chosen because of its simplicity and effectiveness. In this resampling method, the particles are resampled with probability equal to their weight. Resampling was carried out at every time step because the computational power was available. The number of particles was $N = 1000$, chosen as an initial guess from prior knowledge and testing.

The root-mean-square error (RMSE) was used as a metric for the accuracy of the estimation. That is

$$\text{RMSE} = \sqrt{\frac{1}{T}\sum_{k=1}^{T}(\hat{L}_k - L_k)^2} \tag{4.1}$$

where $\hat{L}_{L,k}$ is the estimated pressurizer level from the expected values of the particles and their weights. The RMSE was 0.0571% pressurizer level. Choosing $N = 1000$ gives an approximately optimal estimate of the pressurizer level. This demonstrates that selecting a sufficient number of particles gives a satisfactory estimate of the state.

4.2 Multiple-model Particle Filtering Results for Detecting LOCA

The multiple-model approach was tested on pressurizer data from the GSE simulator. The multiple-model formulation of the pressurizer can be found in Appendix B. The data was taken during a power change from 936 MW to 1000MW at 5MW/min. Data from a power change was used because it will show how these methods handle differentiating between transients and faults. A LOCA is initiated with initial magnitude 1% (∼200 gallons per minute (gpm)) and final magnitude 5% (∼1000 gpm). The simulator stops at reactor trip.

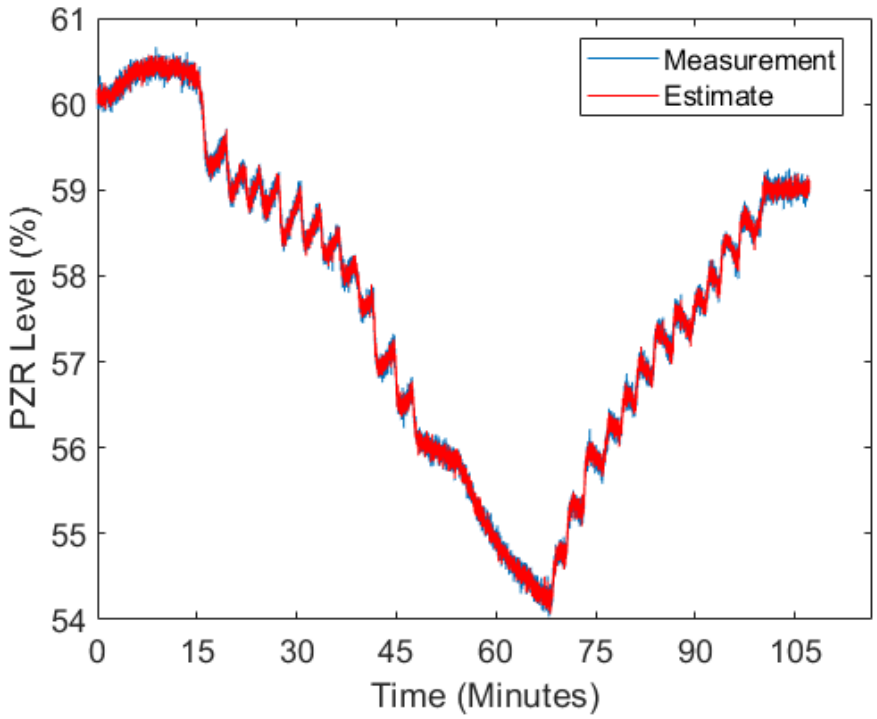

Figure 4: Particle filtering results for estimation of pressurizer level during power change from 100% to 75% to 100% over 1.6 hours.

Particle filters were used on three different models assuming $M^{(j)} = 0, 500, 1000$ gpm. These leak magnitudes were chosen to demonstrate a drawback of the multiple-model method. $N = 1000$ for each particle filter as before. Figure 5 shows results for the estimation of the leak magnitude. Figure 6 shows results for pressurizer level estimates.

To give an idea of how the particle weighting process works, Figure 7 shows how the particle weights evolve over a short time period. The particles nearest the measurement receive the highest weights.

In Figure 6, the 0 gpm and 500 gpm model estimates stay close to the actual pressurizer level for the duration of the data. However, the 1000 gpm model estimate slowly strays, then hits a breakpoint and diverges even faster. This is happening because the model is defined to have a high leak magnitude. The 1000 gpm leak magnitude is causing the pressurizerlevel estimate to drop rapidly. On the other hand, the particle filter attempts to keep the pressurizer level estimate close to the pressurizer level measurement. The high leak magnitude is pulling the pressurizer level estimate downwards at a greater rate than the particle filter's weighting and resampling algorithm pull the estimate upwards. As the 1000 gpm particles diverge from the pressurizer level measurement, their weights decrease. Eventually, all of the particle weights become machine-zero. When all of the particle weights are the same, they all have equal probability of being resampled. Therefore, resampling accomplishes nothing and is not able to keep the particles close to the level measurement. This is where the breakpoint occurs, and the 1000 gpm estimate completely diverges. This phenomenon is show in Figure 8. Here we will define the detection time as the length of time between LOCA incidence and the estimate becoming non-zero. In these results, the LOCA detection time is 21.5 seconds. The leak magnitude estimate jumps to 500 gpm and does not effectively track the rising leak magnitude. At the time of the LOCA, the 1000 gpm model estimate is extremely far from the actual pressurizer level. The probability of the 1000 gpm model is then essentially zero, giving the expected value of the leak magnitude zero weighting from the 1000 gpm model. In summary, this method can detect that there is a LOCA, but cannot estimate the correct leak magnitude if there are divergent magnitudes. Aside from this, the method correctly ignores the transients of the power change.

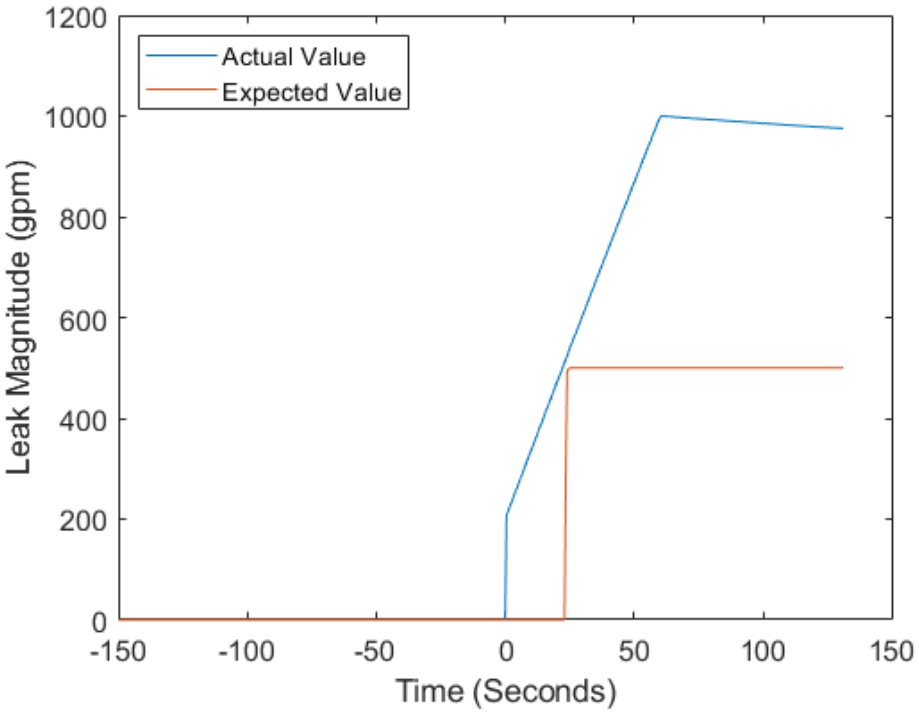

Figure 5: Leak magnitude estimation during plant power change using multiple-model particle filtering. The leak magnitude estimate shows that there is a leak, but gets "stuck" at 500 gpm due to the divergence of the 1000 gpm model

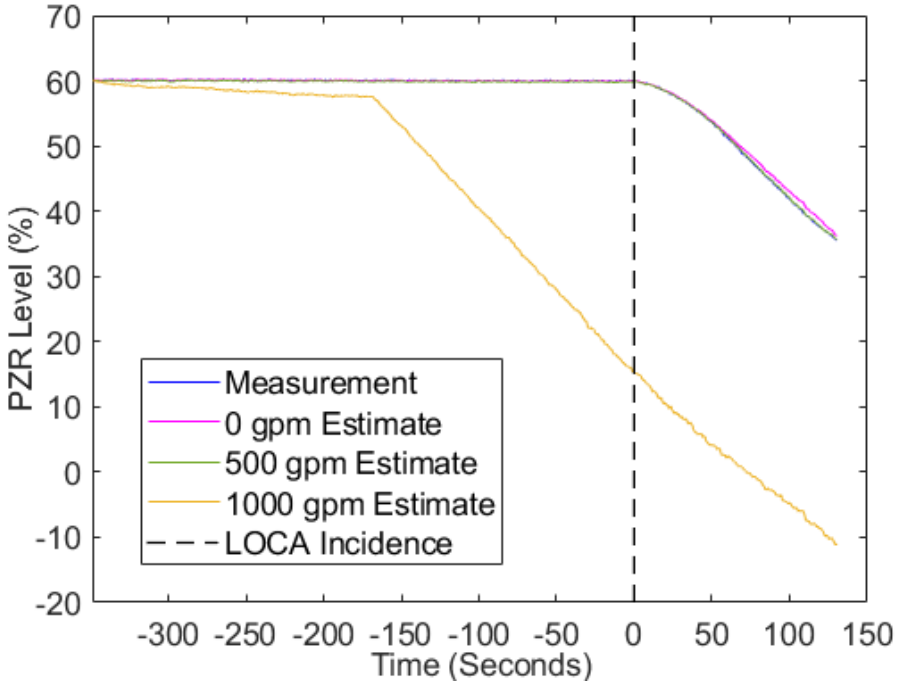

Figure 6: Pressurizer level estimates from particle filters using the standard multiple-model method. The 1000 gpm model estimate slowly strays from the measurement, then completely diverges.

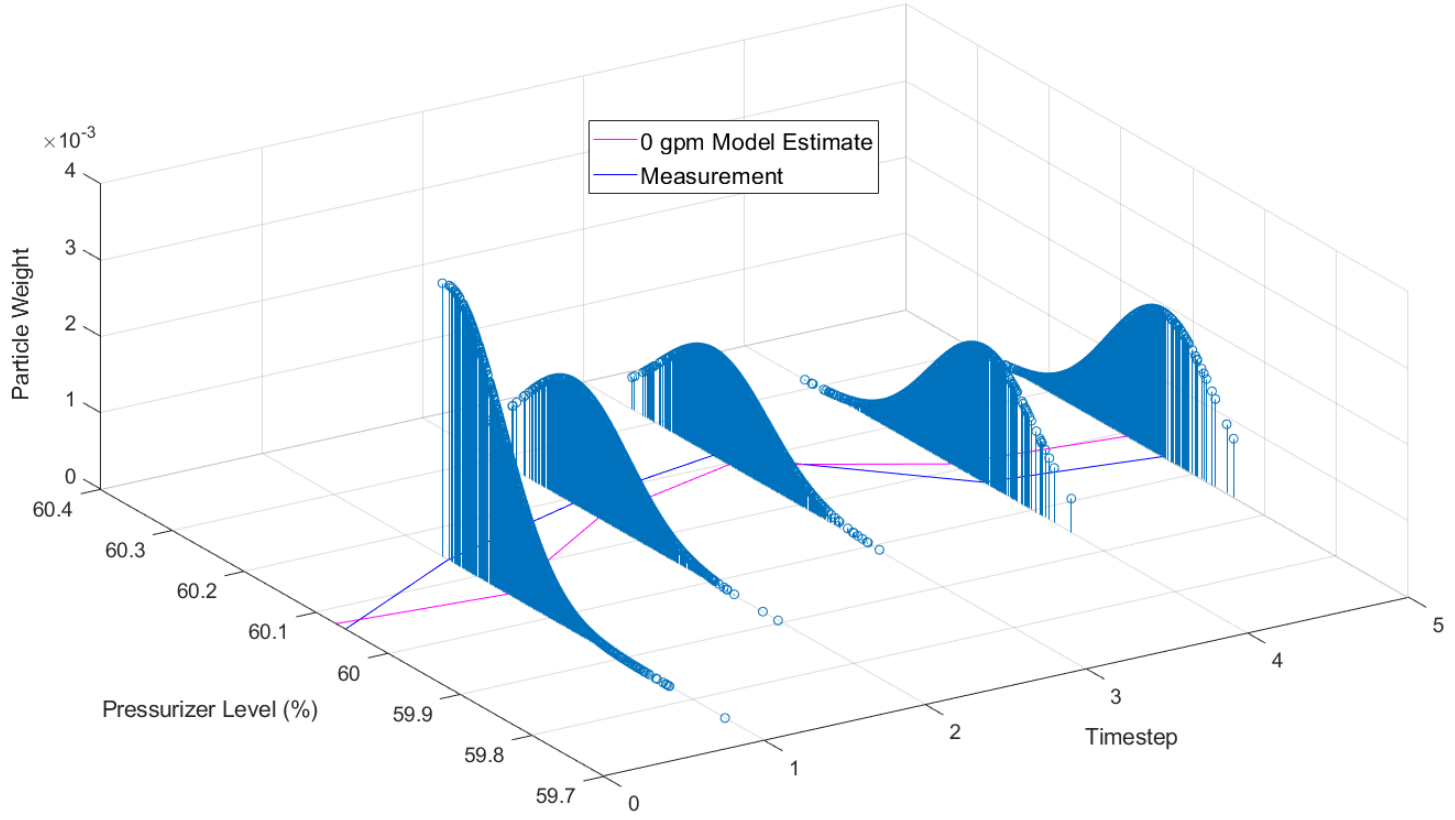

Figure 7: 3D plot of 0 gpm particle weights (shown as stem plots) over a short time period. On the X-Y plane, the blue line shows the pressurizer level measurement, and the megenta line shows the 0 gpm pressurizer level estimate.

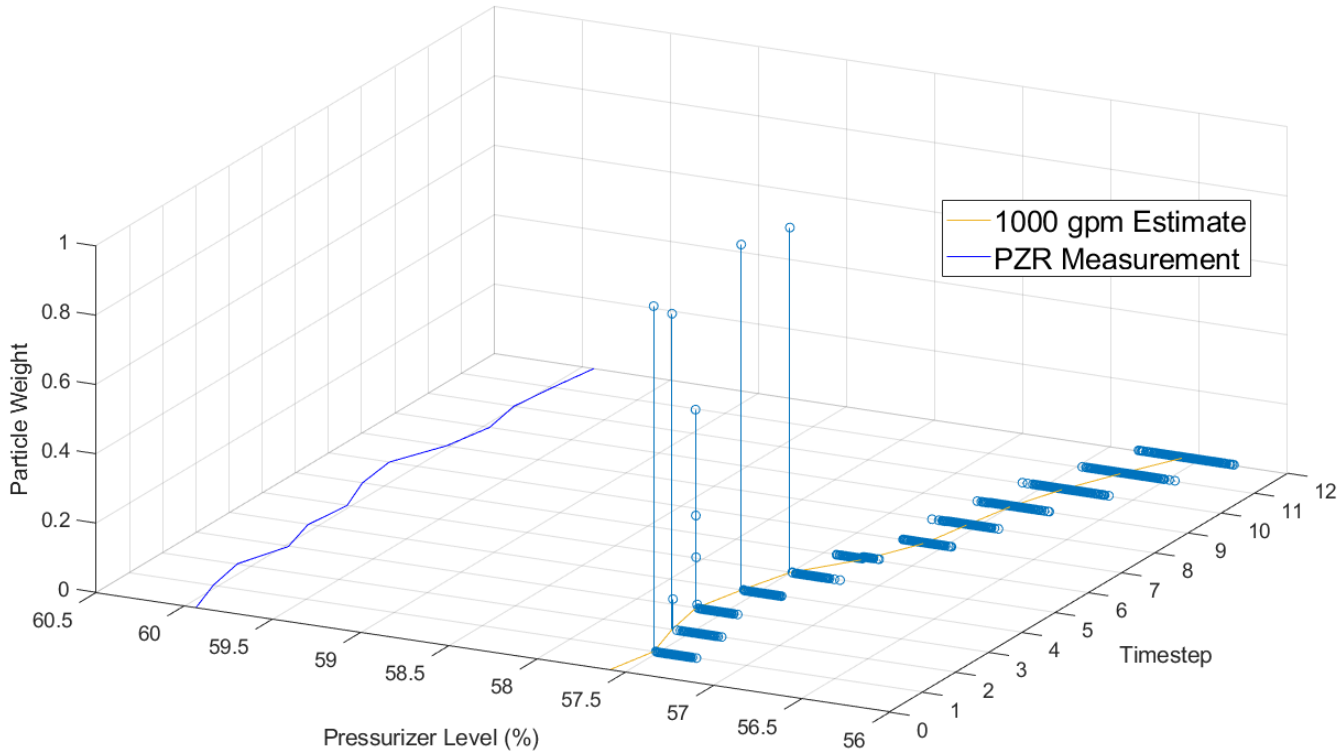

Figure 8: 3D plot of 1000 gpm particle weights (shown as stem plots) around the time of complete divergence. On the X-Y plane, the blue line shows the pressurizer level measurement and the yellow line shows the 1000 gpm model particle filter estimate. Before complete divergence, only one or a few particles have significant weight. Then, a threshold is crossed, after which the weights are so small that the computer thinks they are all effectively 0.

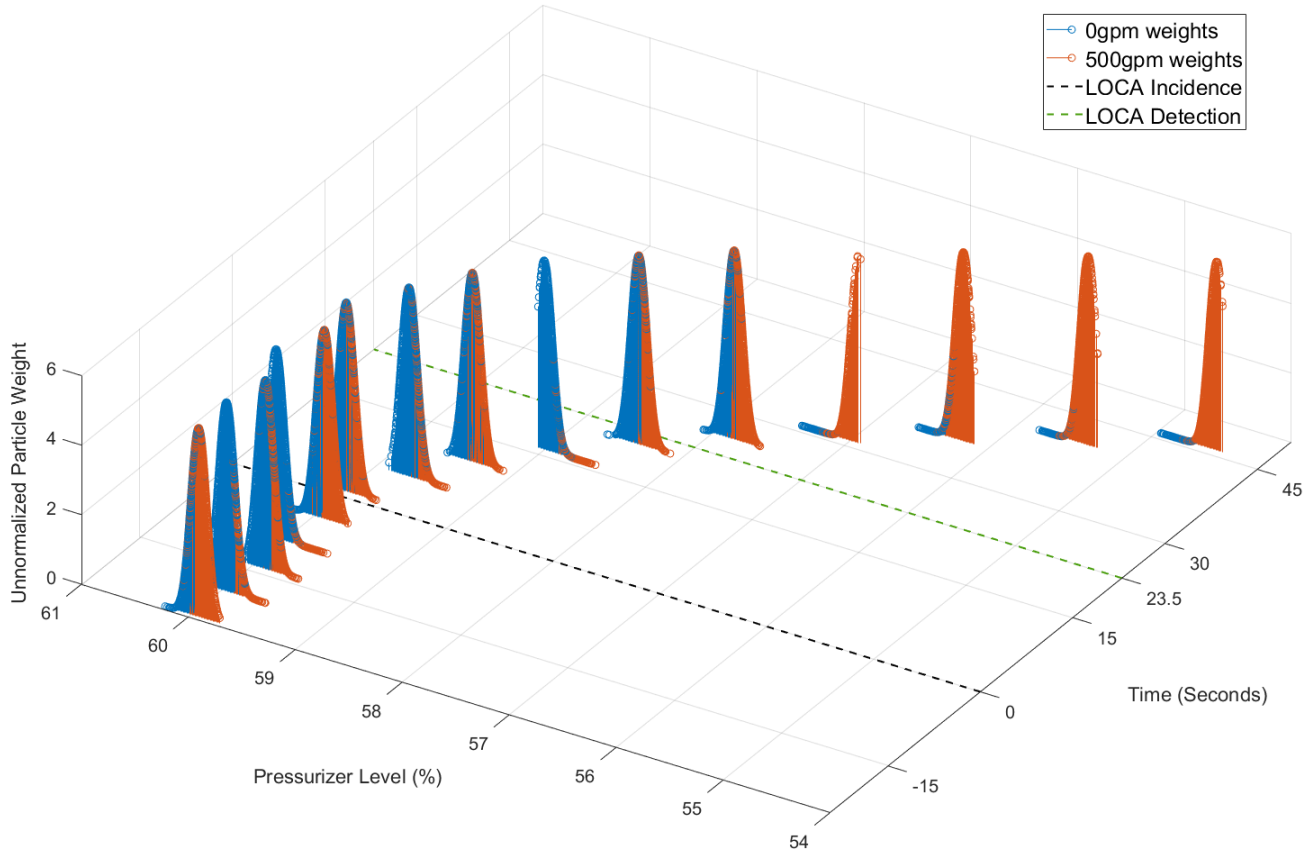

Figure 9: 3D plot of the 0 gpm and 500 gpm model particle weights. This illustrates the detection mechanism of the multiple-model particle filter. Unnormalized particle weights are shown because they provide a clearer trend. Prior to LOCA incidence, the 0 gpm weights are dominant. After incidence, the 500 gpm weights become dominant. This results in the leak magntude estimate jumping to 500 gpm at 23.5 seconds after incidence.

4.3 Interacting Multiple-model Particle Filtering Results for Detecting LOCA

The interacting multiple-model technique of [17] was tested on the same data used for the standard multiple-model test. Particle filters were used on models assuming $M^{(j)} = 0, 100, 200, ..., 2000$ gpm. We already know that this method overcomes the divergence issue, so these leak magnitudes were chosen to demonstrate the overall performance of the IMM algorithm.

As the resolution of the set of leak magnitudes increases, more particles are needed to differentiate between models. Therefore, $N = 2000$ was used and provided good performance. The probability threshold of $P(H^j|\mathbf{D}_k)$ for re-initialization was 10^{-10}. Figure 11 shows results for the pressurizer level estimates of each particle filter. Figure 10 shows the leak magnitude estimates.

In this experiment, the detection time was 22 seconds. Figure 10 shows that the estimate tracks the real leak value effectively. The estimate also levels off near the actual final leak value. Figure 11 shows that none of the particle filter estimates are able to diverge from the actual pressurizer level.

The interacting multiple model method works well but there is one drawback. Only a discrete set of leak magnitudes can be explored by the particles. This is because each of our models assumes a leak magnitude that is fixed. Also, we are giving each model a fixed number of particles. Many particles could be exploring leak magnitudes that are not relevant to what is actually occurring in the system. Then, we would be spending computational effort on particles that are not giving us valuable information.

We could remedy this by giving each model a dynamic number of particles based on where we think the actual leak magnitude lies. However, dynamically determining the number of particles needed for each model may be a cumbersome task that takes up more computational effort. Determining where we want most of the particles to be may also be complicated. A natural solution would be to remove the discretization constraints on the particles and allow them to take on any value in the leak magnitude space continuously. We would treat the leak magnitude as a state of the system, i.e., augment the state vector with leak magnitude.

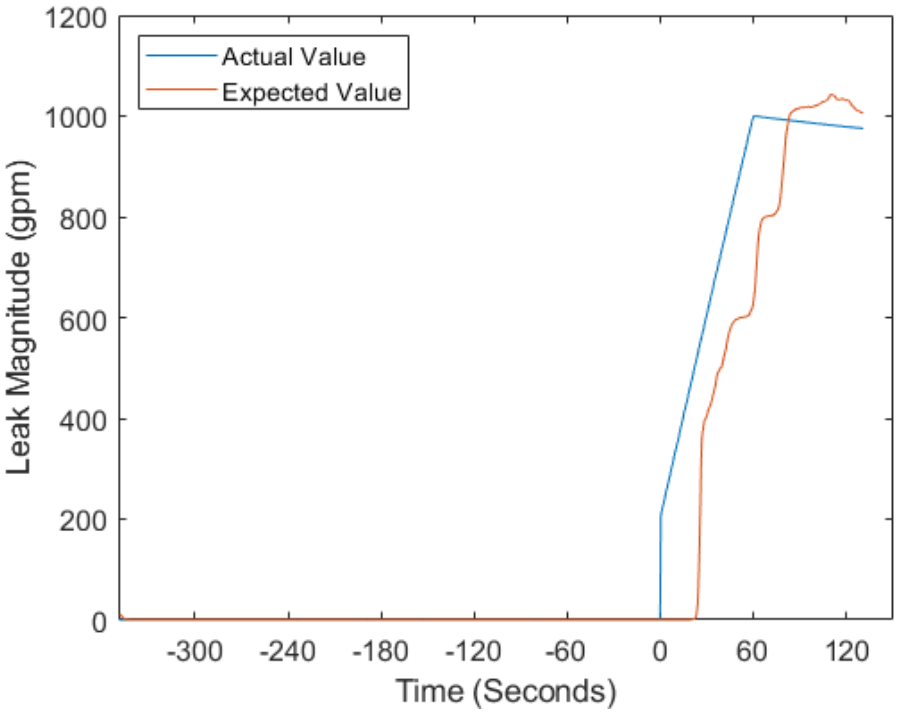

Figure 10: Leak magnitude estimation during plant power change using interacting multiple-model method. The leak magnitude estimate becomes non-zero at 21.5 seconds after the leak occurs. The estimate tracks relatively closely and settles near the real value.

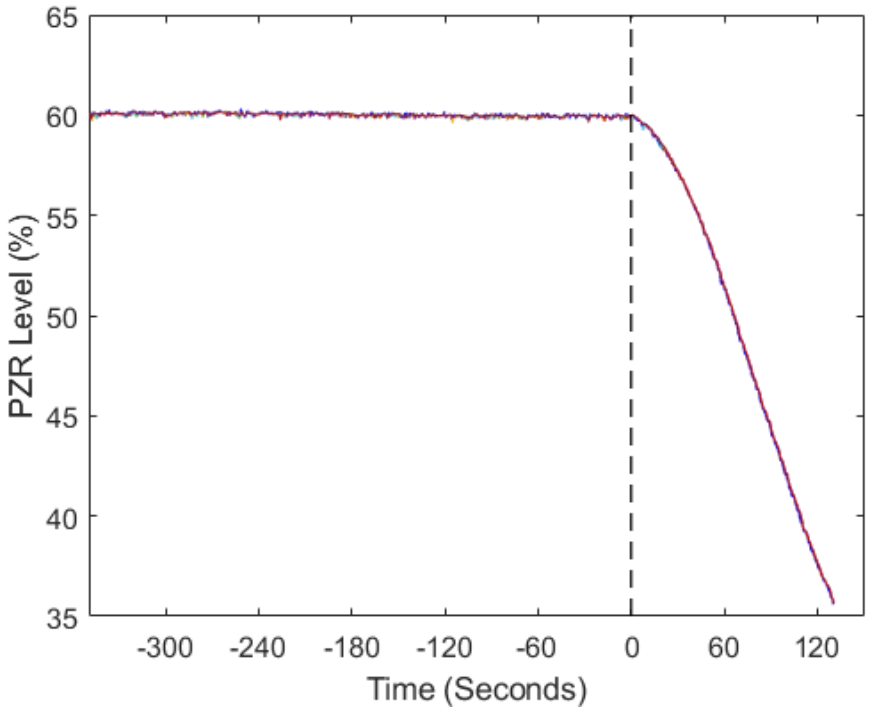

Figure 11: Pressurizer level estimates from particle filters using the interacting multiple-model method. Estimates from all particle filters are shown. Legend not shown due to high number of models. None of the models are able to diverge due to the re-initialization step.

4.4 State-augmented Mutating Particle Filter Results for Detecting LOCA

The state-augmented mutating particle filter (SAMPF) was tested on the same LOCA data as above. A state-augmented version of the pressurizer model can be found in Section C. Results for leak magnitude estimation are shown in Figure 12.

Since the dimension of the state vector has now increased to 2, so we assume that the number of particles likely needs to be increased. N was chosen to be 10000 for this test as an initial guess. Further considerations for number of particles are addressed in Section 4.5.

In a NPP, a LOCA may happen suddenly and with high initial magnitude. As such, the range of the uniform distribution of mutation was 10000 gpm, which is approximately a 50% break. This range should cover most leak magnitudes of concern. The probability of mutation was chosen to be 0.01 as an initial guess. The standard deviation of the local search Gaussian was 10 gpm, a reasonable guess to detect changes in the leak magnitude. The estimation of the pressurizer level remains largely unchanged, so it is not shown.

The most obvious issue is the spikes that occur occasionally throughout the estimation. Due to these spikes, we cannot define detection time as we previously have. We will discuss how to define detection time in a later section. By inspection, however, it can be seen that the estimation responds to the LOCA more quickly than the IMMPF.

We can examine one of the spikes to investigate what is causing this phenomenon. We will look at the first very large spike occurring at the time step 20. Recall that the estimation of the leak magnitude is carried out through the expected value of the particles' values and their weights. For a spike to occur, there must be particles that have high leak magnitude values that also have high weights. The weighting is determined only by the particles' pressurizer level values, so we will examine the pressurizer level estimation in this area.

Figure 13 shows the pressurizer level estimation in the neighborhood of time step 20. Time step 17 to 20 has been delineated to highlight the issue. It appears that the expected value of the pressurizer level is above the actual value in this window. At time step 17, some particles have high leak magnitudes due to the mutation process. This causes their pressurizer level to drop near the actual level. The particles then gain high weights, so they are resampled a number of times. Then, at time step 18, the pressurizer level measurement

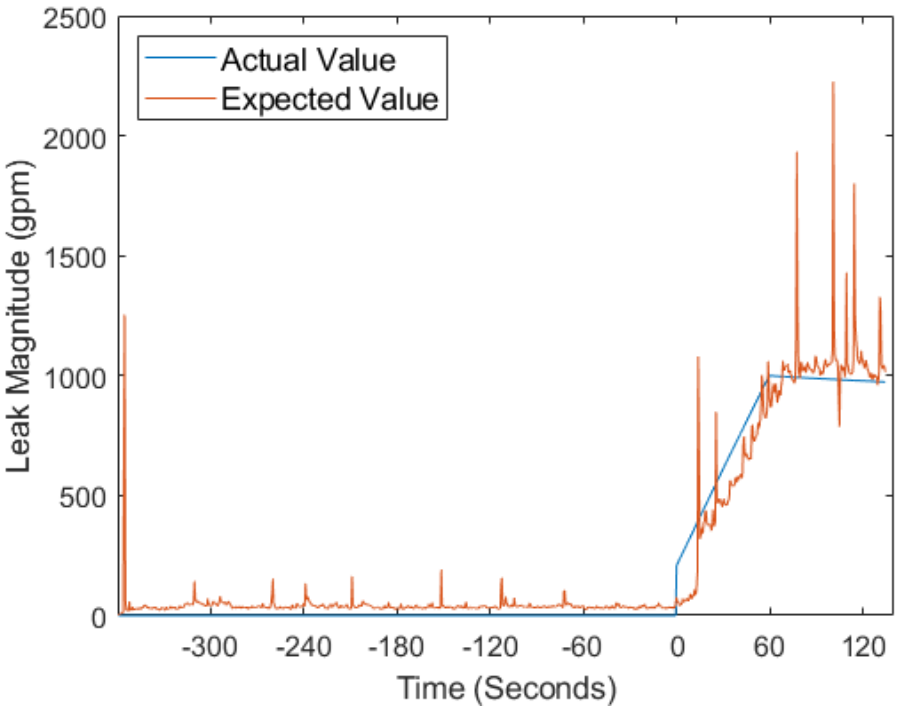

Figure 12: Leak magnitude estimates from the SAMPF with a mutation probability of 0.01. The spikiness is related to how aggressively we search the leak magnitude space with the mutation proability.

drops naturally. The particles that had high weights at time step 17 are still in the vicinity of the actual pressurizer level. Again, their weights become high and they are resampled many times. This process repeats at time steps 19 and 20 creating a "snowball" effect that results in a huge peak at 20. Further examination shows that something similar happens around each major spike.

Note also that the leak magnitude estimation is non-zero when the real value is zero, i.e., when the system is at steady-state. This means that the SAMPF will not be able to effectively detect faults below the non-zero value. This, along with the spikes, are a side effect of the mutation process.

There is no straightforward way to completely eliminate either of these phenomena, but they can be reduced by altering various algorithm parameters. The spikes are caused by particles that have been mutated to higher leak magnitudes. This leads us to believe that reducing the mutation probability can help to combat the spikes in some way. However, we are sacrificing the diversity of global information in our particle filter. Intuitively, the particle filter may not respond as quickly to a LOCA because there are fewer particles "testing" high leak magnitudes. So, there may be some trade-off between the severity of the spikes and speed of response.

We can test this by reducing the mutation probability to 0.001. Figure 14 shows the result. It can be seen that many of the spikes have been eliminated or reduced. By inspection, however, the estimation does not respond as quickly as before.

To further investigate the trade-off between spike severity and speed of response, we need to define metrics for these quantities. A reasonable choice for speed of response is time constant. The time constant is defined as the time it takes for the estimate to reach 63% of the actual leak magnitude. An increase in time constant indicates a slower speed of response. Although they are not the same, the time constant will be a sort of "stand-in" for the detection time. Later we will discuss the development of a heuristic for official leak detection.

For severity of spikes, we will use the RMSE between the estimate and actual value. To a degree, the RMSE indicates the magnitude and frequency of the spikes. If it is high, the spikes are large and occur often.

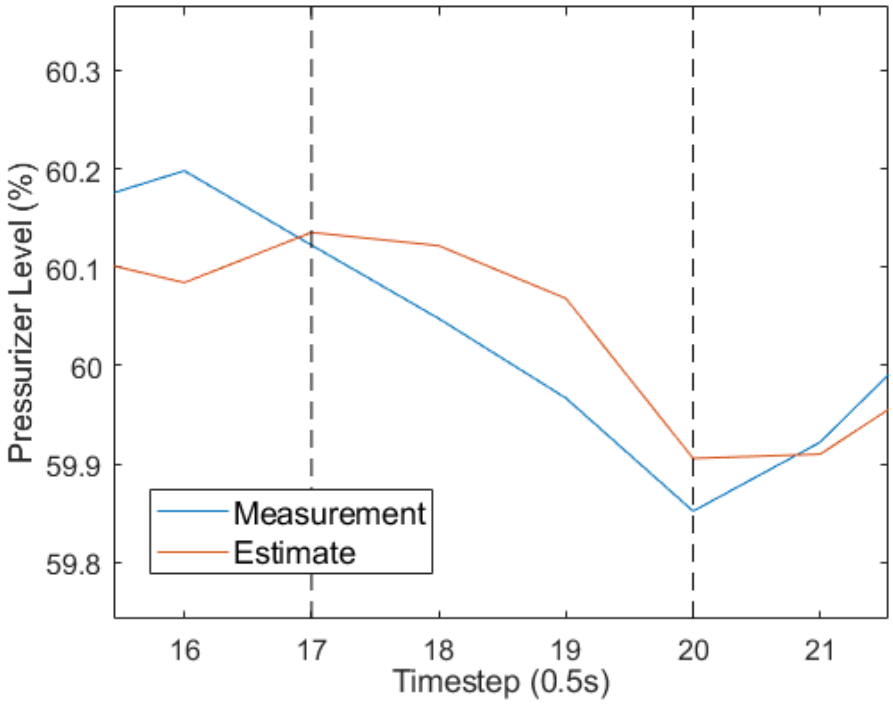

Figure 13: Pressurizer level measurement and pressurizer level estimate in the neighborhood of the 20th timestep. The measurement is consistently dropping in this region, so the estimator thinks there may be a leak.

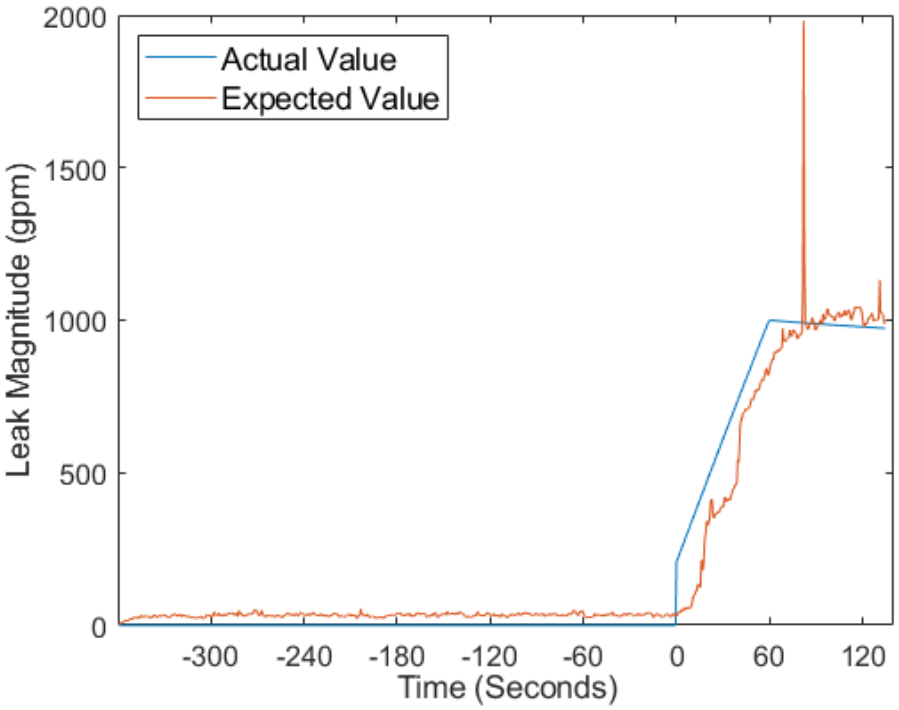

Figure 14: Leak magnitude estimate from the SAMPF with a mutation probability of 0.001. The lower mutation probability has reduced or eliminated many of the spikes.

We have already demonstrated that the SAMPF can track a changing leak magnitude that occurs during plant transients. So to further characterize the trade-off, we will now examine data from a 15% ($\sim$3000 gpm) LOCA during steady state. This will simplify the use of our metrics and will also show that the SAMPF can quickly detect LOCAs of very large magnitudes.

The SAMPF was carried out on this with various mutation probabilities between 0.0001 and ~ 0.5. This seemed like a reasonable range of mutation probabilities to test. Twenty trials for each mutation probability were run with different initialization of the random number generator. The time constant and RMSE were averaged across the trials for each mutation probability. Figures 15 and 16 show plots of the results with the mutation probability axis scaled logarithmically to show a clearer trend. The standard deviation of the RMSE and time constant results is shown by the vertical lines at every data point.

The results show that the time constant does not change much when varying mutation probability. The range of time constants is around 0.5 seconds. In a NPP, as in most systems, this small of a difference in response time will not be as significant as the accuracy of the estimate. Therefore we should use the RMSE as the deciding factor when choosing mutation probability. Our batch of results suggests that a mutation probability in the neighborhood of 0.01 achieves the lowest RMSE. The probability may be made adaptive, wherein it could depend on a number of other factors. During plant transients, times of high load, or high system stress, the mutation probability could rise to accommodate the higher likelihood of LOCA occurrence. During steady state, 0.01 should be a sufficient probability.

Note the high RMSE and standard deviation of the RMSE at the mutation probability of 0.0001. When the mutation probability is very small, there are very few particles that are exploring high leak magnitudes. When the LOCA occurs, the estimation will jump to the leak magnitude value of those few particles. By chance, those particles may or may not be close to the actual leak magnitude. Thus, the RMSE can be very high or very low for a given run, resulting in a large standard deviation.

In conclusion, this method can quickly detect a leak of any magnitude within the range of the mutation distribution. This comes at the cost of estimation accuracy, as the RMSE is somewhat poor.

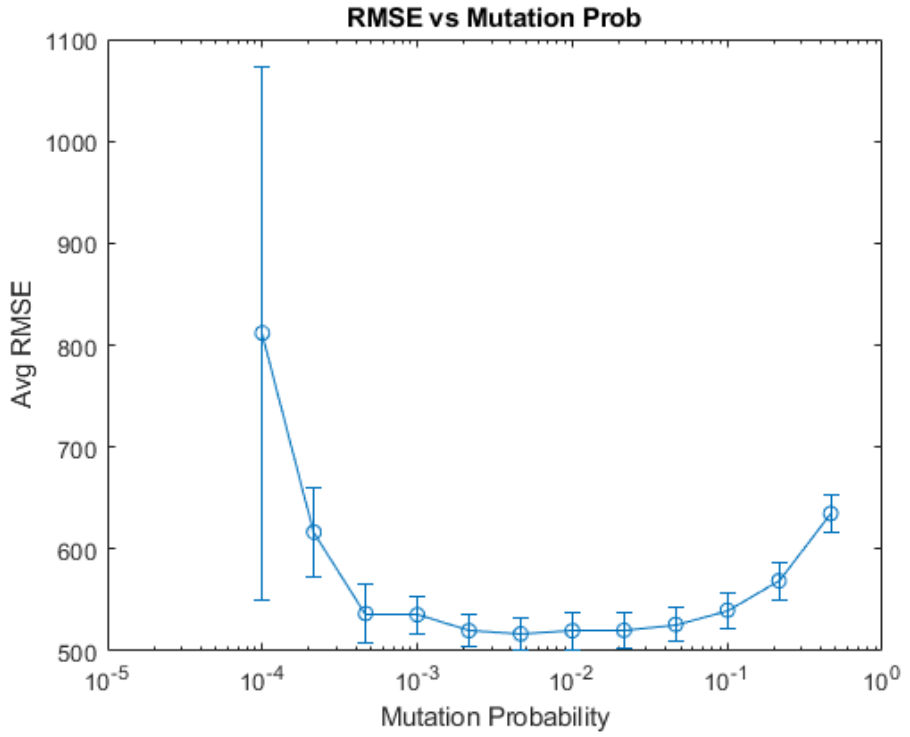

Figure 15: Plot of averaged RMSE results across 20 trials of SAMPF on 15% LOCA during steady-state with varying mutation probabilities. The RMSE has a minimum around $p_{mut} = .01$.

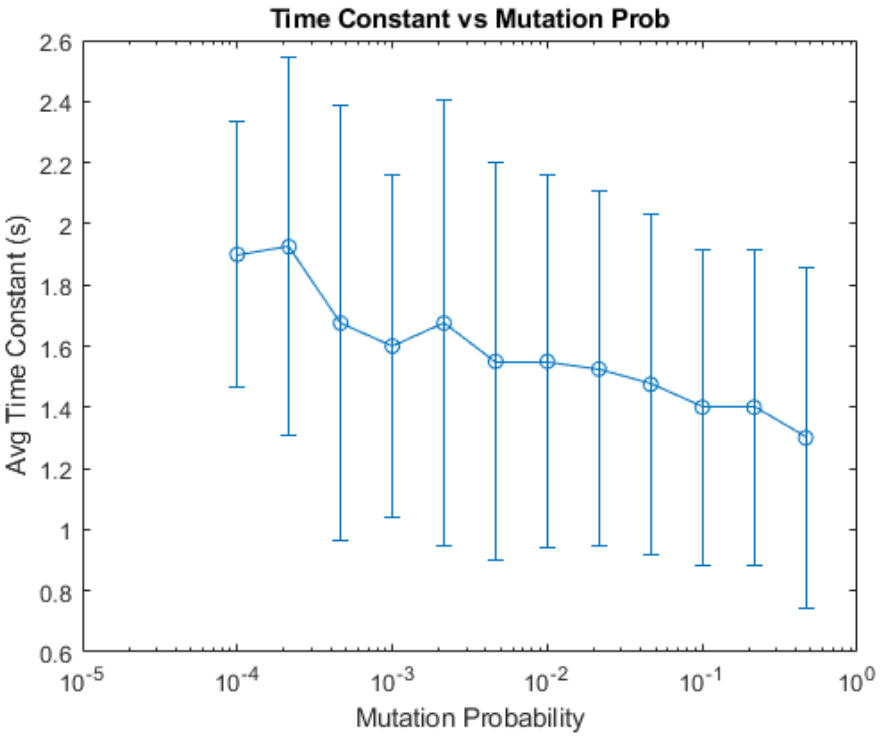

Figure 16: Plot of averaged time constant results across 20 trials of SAMPF on 15% LOCA during steady-state with varying mutation probabilities. Time constant does not decrease significantly over this interval of mutation probabilities.

If the computational power is available, a hybrid FDD system can be built that incorporates the SAMPF. The SAMPF can be used to quickly react to sudden and sharp faults. Another FDD method can be used to capture faults that are below the SAMPF's errant fault estimate at steady-state. Preferably, an FDD method with high accuracy would be selected to combat the spikiness of the SAMPF estimate. This would require analysis on how to fuse the estimates of both algorithms.

If the computational power is not available, we can develop a heuristic that will allow us to ignore the spikes but still quickly detect the LOCA. If the leak magnitude estimate is above a certain threshold for a certain number of consecutive times steps, then the alarm will trigger and an estimated leak magnitude will be displayed. Choosing the number of consecutive time steps after which the alarm is triggered should be high enough to ignore spikes, but short enough to maintain a good detection speed. In these experiments, no spike lasted more than 3 time steps. The magnitude threshold above which the alarm will trigger should be sufficiently above the average estimate before LOCA incidence.

For our results, the average estimate before LOCA changes slightly depending on the mutation probability. At a mutation probability of 0.01, the average value is approximately 14 gpm, so we would set our threshold at 40 gpm. In summary, the alarm would trigger when the leak magnitude estimate is above 40 gpm for 4 consecutive timesteps. Of course, this heuristic will be different depending on the system.

4.5 Considerations for Number of Particles in SAMPF

To make our SAMPF more efficient, we can carry out an analysis on how the number of particles affects the spikiness and speed of response. In these tests, the SAMPF was carried out on the 15% LOCA data. The number of particles was varied between 10 and 100000, and twenty trials were run for each number of particles. The mutation probability for all trials was 0.01, as found in the previous analysis. As before, the leak mangitude estimation RMSE and time constant were averaged across the trials for each number of particles. Results are show in Figures 17 and 18. The standard deviation is again shown with vertical lines.

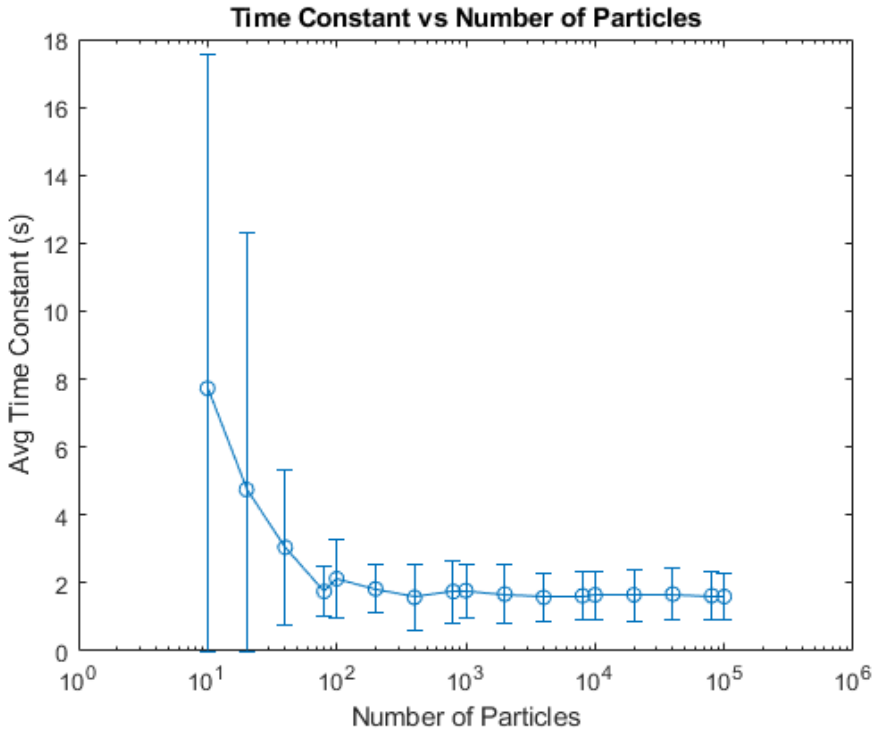

Figure 17: Plot of averaged leak magnitude estimation time constant results across 20 trials of SAMPF on 15% LOCA during steady-state with varying numbers of particles. The Time constant hits a minimum around $N = 1000$

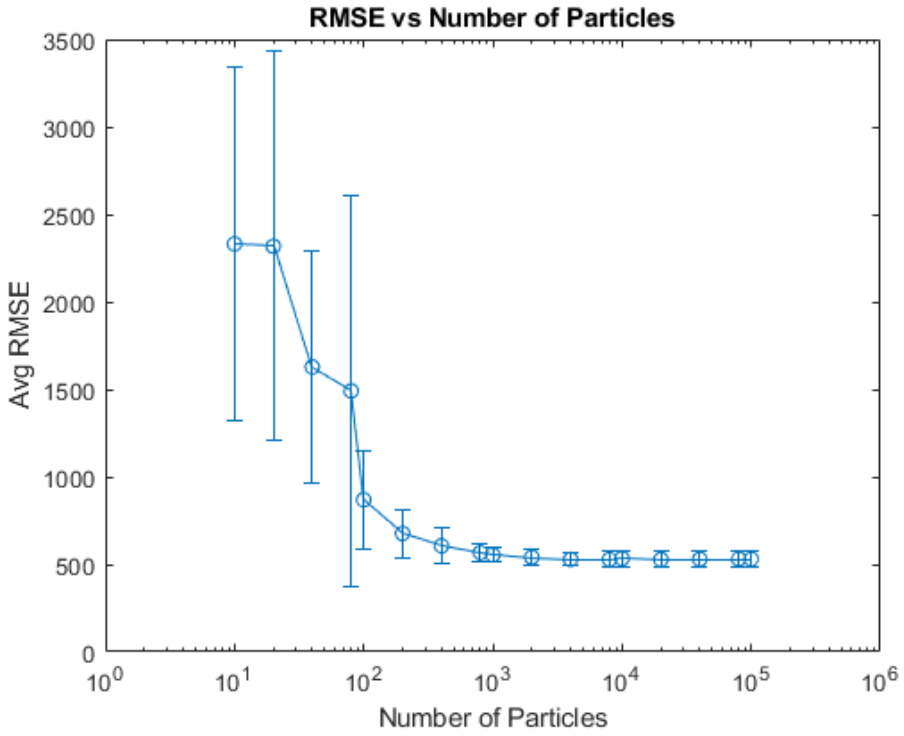

Figure 18: Plot of averaged leak magnitude estimation RMSE results across 20 trials of SAMPF on 15% LOCA during steady-state with varying numbers of particles

For both RMSE and time constant, there appears to be an exponential relationship with number of particles. Time constant levels off around 100 particles. We will again use the RMSE as the deciding factor because it levels off at a higher number of particles than time constant. At 1000 particles both RMSE and number of particles are essentially minimized, so that will be our choice of particles for this system. Note also that the standard deviation levels off for both time constant and RMSE. This indicates that increasing the number of particles increases the consistency and repeatability of the SAMPF.

Figure 19 shows an example of the leak magnitude estimation using the parameters that we found from the analysis: $N = 1000, p_{mut} = 0.01$. All other parameters are the same as before. Figure 20 shows results from a Gaussian-only random walk particle filter. The time constant for the SAMPF is 1.5 seconds, while the time constant of the Gaussian-only random walk is 11 seconds.

4.6 Comparison of IMMPF and SAMPF for LOCA Detection

Now that we have found the best SAMPF parameters for this system, we can compare its performance with the IMMPF of [17] in further depth. All tests are done with the data from the 15% LOCA at steady-state. 50 trials were done for both the SAMPF and IMMPF, with different intialization of the random number generator. The leak magnitude estimation RMSE and time constant were averaged across the 50 trials. Here we will also record the computation time per timestep, i.e., how long it takes the algorithm to finish its computation every iteration. This was also averaged across the 50 trials. For the SAMPF, $N = 1000, p_{mut} = 0.01$. All other SAMPF parameters are the same as before. For the IMMPF, $M^{(j)} = 0, 100, 200, ..., 10000$ gpm to mimic the SAMPF's mutation distribution range of 10000 gpm. The total number of particles for all models in the IMMPF was restricted to be equal to the number of particles used in the SAMPF. This was done so that computational capabilities would be comparable. Therefore, $N = \text{ceil}(\frac{1000}{J})$ for each model in the IMMPF. Table 4.6 shows the performance comparison.

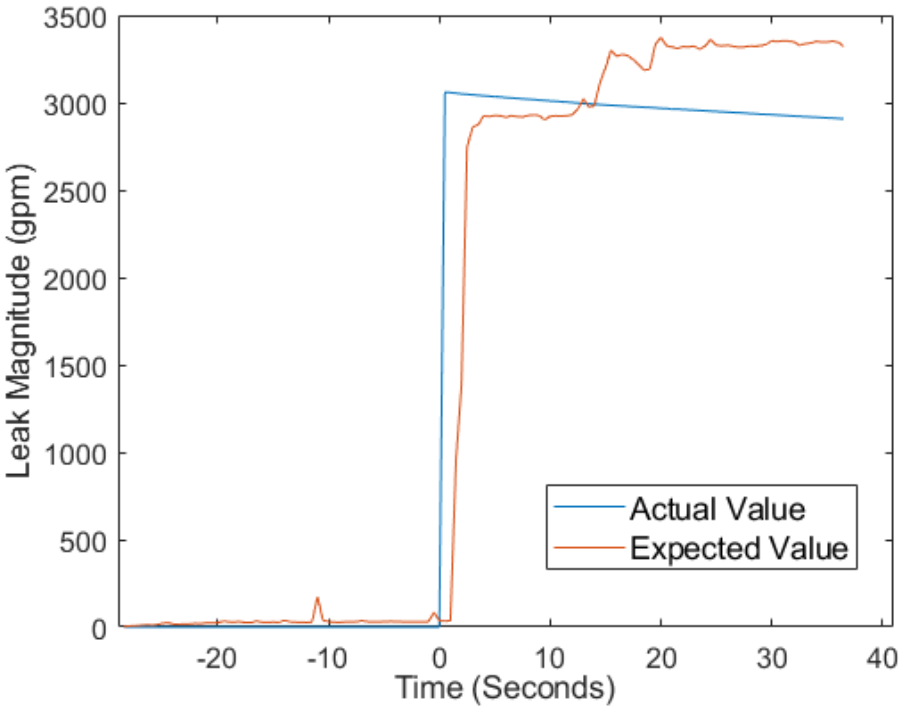

Figure 19: SAMPF leak magnitude estimation with $N = 1000, p_{mut} = 0.01$ on 15% LOCA data. The estimate responds to the leak within 2 seconds, but it struggles to settle near the final leak value before the reactor trips.

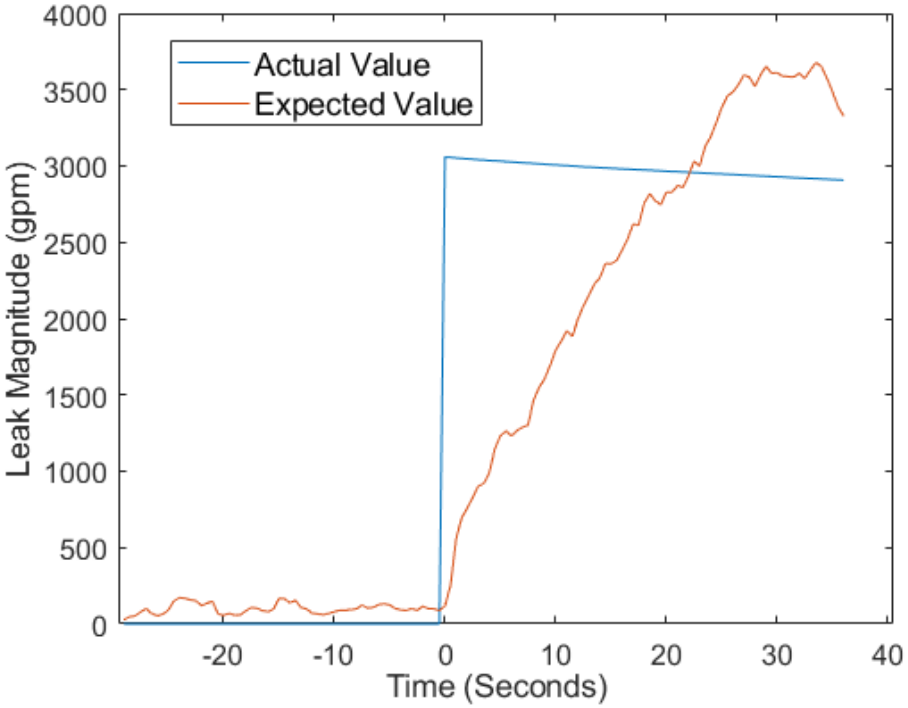

Figure 20: Gaussian-only random walk leak magnitude estimation with $N = 1000$ on 15% LOCA data. The estimate ramps up to the real value slowly.

Table 1: Leak magnitude estimation performance comparison between the SAMPF and IMMPF on 15% LOCA data with total number of particles equal to 1000 for both algorithms. Results averaged across 50 trials.

	RMSE (gpm)	Time Constant(s)	Computation Time per Iteration(ms)
IMMPF	749.1 ± 135.1	1.77 ± 0.82	7.745 ± 0.484
SAMPF	543.7 ± 49.29	1.7 ± 0.81	9.887 ± 0.656

The IMMPF is shown to be approximately 22% faster in computation time than the SAMPF. At $N = 1000$ and with a sampling time of $h = 0.5s$, this difference is negligible. This may become an issue at larger Ns and smaller sampling times. The difference in time constant is approximately 4%, which is likely negligible in all cases. The SAMPF achieves a 27% lower average RMSE and 63% lower standard deviation of the RMSE than the IMMPF. This indicates that the SAMPF is more accurate and consistent than the IMMPF algorithm of [17] when total number of particles is held constant. Accuracy and consistency will likely be more important than the slower computation time in most systems.

The standard IMMPF method that requires a transition matrix may provide better performance than the SAMPF. However, one benefit of the SAMPF over the standard IMMPF is that it does not require any knowledge of a transition matrix.

5.0 Summary & Conclusion

Fault detection and diagnosis is broken into two general categories: data-driven and model-based techniques. Data-driven techniques require large quantities of data to train the detection tools, which may not always be available for the faults of concern. A model-based approach has been chosen for this research to avoid the need for mass data collection. Model-based techniques are appealing for systems that can be effectively modelled. These methods often use observers to produce outputs for the system and system models.

For this research, the observer of choice is the particle filter for two reasons. The first is that it can produce quality estimates for systems with any nonlinearities and noise with any distribution. Kalman filter-based observers can struggle to produce quality estimates for nonlinear and non-Gaussian systems.

Second, the state-augmented mutating particle filter (SAMPF) takes advantage of the particle filter's Monte Carlo nature which provides two benefits: 1) only one model is needed to diagnose faults of any magnitude and 2) modelled faults are not constrained to discrete values. This is in contrast to standard multiple-model methods wherein several models with fixed faults are run in parallel. Choosing which fault values to model and how many particles to allocate at each value may be a cumbersome task.

The SAMPF allows for continuous particle fault values using a technique called random walk, wherein each particle randomly determines its fault value according to a distribution chosen by the user. Choosing a distribution is usually more straightforward than picking discrete fault values to monitor. Using a distribution instead of discrete values offers more flexibility to the user for which areas of the faults space to track.

In the random walk technique, particle fault values are typically distributed with a Gaussian. Depending on the system, it may go unstable only minutes after fault occurence. So, one goal of the research in this document was to develop a random walk method that could quickly detect faults of large magnitudes. The SAMPF achieves this by introducing a mutation probability that enables the particles to explore a large range of fault values.

The ideal number of particles and mutation probability for the SAMPF were found for the pressurizer system. The performance of the SAMPF with these parameters is compared with an interacting multiple-model (IMM) technique for detecting a loss-of-coolant accident in a nuclear reactor simulation. The SAMPF is superior in estimation accuracy and consistency. However, it was slightly slower in computation time. There was no significant difference in time constant between the IMMPF and SAMPF. This comparison is based on average values across 50 trials.

The SAMPF still suffers from a non-zero fault estimate during steady state, whereas the IMMPF does not. To combat this, certain heuristics could be developed for the SAMPF to determine when the fault should be considered detected. Alternatively, the SAMPF could be used in parallel with another method that does not suffer from non-zero fault estimates at steady state. At steady state, the non-zero fault estimate of the SAMPF at steady state would be considered alongside the zero fault estimate of the other method.

In conclusion, the primary benefit of the SAMPF is that it allows for continuous fault values while being able to quickly detect large fault values. The idea behind the SAMPF is a basis for multi-distribution random walk techniques. Most random walk methods use only one distribution. Different types of distributions could be useful depending on the nature of the system. Overall, we feel that this method has much potential to be refined and expanded.

5.1 Future Work

There is still much room for improvement in the SAMPF. There are several ways to build upon it, discussed in the following sections.

5.1.1 Adaptive mutation probability and number of particles

The mutation probability and number of particles could be adaptively determined depending on conditions in the system. If the system is in a period of high stress, and a fault is more likely to occur, mutation probability and number of particles could be increased to accomodate. These parameters could be based on several other considerations. Adaptively determining these parameters could increase estimation accuracy and decrease computation time.

5.1.2 Mutation distribution

Mutation distribution will likely have an effect on detection speed and diagnosis accuracy. For instance, choosing a less aggressive distribution (a high-covariance Gaussian instead of a uniform) may alleviate some spikiness in the fault estimation, perhaps at the cost of detection speed. Work can be done on determining the best general mutation distribution whose parameters could also be adaptive. Again this will depend on the nature of the system. More than two distributions could be used.

5.1.3 Hybrid FDD

This algorithm struggles somewhat with estimation accuracy, but has a very fast detection time. One or more additional FDD methods could be introduced, wherein the SAMPF would be used for detection and the other methods used for diagnosis. Hybrid FDD can likely provide high estimation accuracy and fast detection times for most systems.

5.1.4 Advanced particle filtering techniques

The particle filter used in this research is the most basic version of the algorithm. Advanced techniques mentioned in Section 2 could be employed to improve computation time, estimation accuracy, and detection time.

5.1.5 Nonlinear and non-gaussian systems

The pressurizer system used in this research was modelled as a linear system with Gaussian noise. Since it is a particle filtering method, the SAMPF can be generalized to nonlinear, non-Gaussian systems. Comparison of the SAMPF with other methods could be carried out on a nonlinear, non-Gaussian system. These systems would provide a more rigorous test of the SAMPF's performance.

Appendix A

Pressurizer Model

The model is taken from [1], wherein data was collected from a pressurized water reactor simulator to generate and identify a pressurizer model. First principles were used to derive the model form for the primary coolant loop. Least-squares system identification was used to determine the parameters of the system. The continuous-time model is

$$\dot{L} = -\frac{\dot{\rho}_L - \dot{\rho}_V}{\rho_L - \rho_V} L_L - \frac{c_5}{\rho_L - \rho_V}(\dot{m}_{\text{eff}} + \dot{m}_{\text{net}} + \dot{m}_{\text{leak}}) \tag{A.1}$$

where

$$\dot{m}_{eff} = \frac{c_1\dot{\rho}_L + c_2\dot{\rho}_V + c_3\dot{\rho}_{HL} + c_4\dot{\rho}_{CL}}{c_5} \tag{A.2}$$

$$\dot{m}_{net} = \dot{m}_{out} - \dot{m}_{in} \tag{A.3}$$

and L is the liquid level, the V subscript refers to the vapor part of the pressurizer, the L subscript refers to the liquid part of the pressurizer, the HL and CL subscripts refer to the hot and cold-leg sections of the primary loop, respectively, $\dot{m}_{\text{in}}$ is the mass flow entering the primary loop through the chargers, $\dot{m}_{\text{out}}$ is the mass flow leaving the primary loop through the letdown, and $\dot{m}_{\text{leak}}$ is any mass flow leaving the primary loop through a leak. The values of constants $c_1, ..., c_5$ are found in Table 2.

Table 2: Least-squares estimates of the system parameters for GSE Systems GPWR simulator pressurizer. Constants found in the reasearch of [1].

c_1	c_2	c_3	c_4	c_5
-6.94×10^1	-3.50×10^1	1.99×10^2	4.30×10^2	7.63×10^{-3}

Transforming the model into a continuous-time state-space form, which has the form

$$\dot{x} = A(t)x + B(t)u \tag{A.4}$$

$$z = x \tag{A.5}$$

where the state and input are, respectively,

$$x = L \tag{A.6}$$

$$u = \dot{m}_{\text{eff}} + \dot{m}_{\text{net}} + \dot{m}_{\text{leak}} \tag{A.7}$$

and the matrices are

$$A(t) = -\frac{\dot{\rho}_L - \dot{\rho}_V}{\rho_L - \rho_V} \tag{A.8}$$

$$B(t) = -\frac{c_5}{\rho_L - \rho_V} \tag{A.9}$$

$$C = 1 \tag{A.10}$$

The simulation is a digital system, so the model needs to be formulated in discrete-time. The standard form for this type of system is

$$x_{k+1} = Ax_k + Bu_k + \mathbf{w}_k \tag{A.11}$$

$$z_k = Cx_k + v_k \tag{A.12}$$

where $x_k \in \mathbb{R}^n$ is the state vector, $u_k \in \mathbb{R}^p$ is the input vector, $z_k \in \mathbb{R}^r$ is the measurement vector, A_k is the state matrix, B_k is the input matrix, C_k is the output matrix, $\mathbf{w}_k$ is process noise, and v_k is measurement noise. Both noise sources are assumed to be Gaussian white processes.

During discretization, the time-varying matrices were assumed constant over the sample time. This is because the time constant of the system is long compared to the sample time ($h = 0.5$s). Accordingly, the discrete-time matrices were calculated by

$$A_k = e^{hA(t)} = e^{hA(kh)} \tag{A.13}$$

$$B_k = \left(\int_{\tau=0}^{h} e^{hA(kh)} d\tau \right) B(kh) \tag{A.14}$$

For the pressurizer system, our state is the pressurizer level. This means that each particle will have a weight and a single state value that is a potential realization of the pressurizer level. The particle is denoted by $\tilde{x}_k(i) = \{w_k(i), L_k(i)\}$. The expected value of the pressurizer level is calculated by

$$\hat{L}_k = \sum_{i=1}^{N} w_k(i) L_k(i) \tag{A.15}$$

Appendix B

Pressurizer Multiple-Model Formulation

In this Appendix, we will formulate the pressurizer system into the multiple-model format for FDD. The different fault scenarios show up in the input of the system

$$u = \dot{m}_{\text{eff}} + \dot{m}_{\text{net}} + \dot{m}_{\text{leak}}^{(j)} \tag{B.1}$$

To simplify notation, we will define $M^{(j)} = \dot{m}_{\text{leak}}^{(j)}$. Here, $M^{(j)}$ is an additive fault parameter. We will have J pressurizer models, with each assuming the jth leak magnitude. Everything else in the model remains the same as the model derived in Appendix A.

The goal of the BHT is to find the probability of each of the J models

$$P(M^{(j)}|\mathbf{D}_k) = \frac{P(\mathbf{z}_k|\mathbf{D}_{k-1}, M^{(j)})P(M^{(j)}|\mathbf{D}_{k-1})}{\sum_{j=1}^{J} P(\mathbf{z}_k|\mathbf{D}_{k-1}, M^{(j)})P(M^{(j)}|\mathbf{D}_{k-1})} \tag{B.2}$$

This is the probability that the current leak magnitude is equal to $M^{(j)}$.

The expected value of the leak magnitude can be calculated at each time step

$$\hat{M} = \sum_{j=1}^{J} M^{(j)} P(M^{(j)}|\mathbf{D}_k) \tag{B.3}$$

The expected value of the pressurizer level is

$$\hat{L}_k = \sum_{i=1}^{N} w_k(i) L_k(i) \tag{B.4}$$

Appendix C

State-Augmented Pressurizer Model

When treating the leak magnitude as a state of the system, we assume that it has negligible physical dynamics and propagates through time with a "random walk." This means that its value will change according to a user-chosen random noise process

$$\dot{M}(t) = \mathbf{w}_{rw}(t) \tag{C.1}$$

or in discrete time

$$M_{k+1} = M_k + \mathbf{w}_{rw,k} \tag{C.2}$$

where $w_{rw}(t)$ and $w_{rw,k-1}$ are the user-chosen process noise. With addition of leak magnitude to the state vector, the continuous-time state-space pressurizer model becomes

$$\begin{bmatrix} \dot{L} \\ \dot{M} \end{bmatrix} = \begin{bmatrix} A(t) & B(t) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L \\ M \end{bmatrix} + \begin{bmatrix} B(t) \\ 0 \end{bmatrix} u + \begin{bmatrix} \mathbf{w}(t) \\ \mathbf{w}_{rw}(t) \end{bmatrix} \tag{C.3}$$

$$z = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} L \\ M \end{bmatrix} \tag{C.4}$$

where $u = \dot{m}_{\text{eff}} + \dot{m}_{\text{net}}$. M no longer shows up in the input u because it has been moved to the state vector. In discrete time the system model is

$$\begin{bmatrix} L_{k+1} \\ M_{k+1} \end{bmatrix} = \begin{bmatrix} A_k & B_k \\ 0 & 1 \end{bmatrix} \begin{bmatrix} L_k \\ M_k \end{bmatrix} + \begin{bmatrix} B_k \\ 0 \end{bmatrix} u_k + \begin{bmatrix} w_k \\ w_{2,k} \end{bmatrix} \tag{C.5}$$

$$z_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} L_k \\ M_k \end{bmatrix} \tag{C.6}$$

Each particle in our particle filter will now have a pressurizer level value, a leak magnitude value, and a weight. This is denoted by $\tilde{x}_k(i) = \{w_{i,k}, L_k(i), M_k(i)\}$.

Bibliography

[1] J. A. Farber and D. G. Cole. Using model-based fault detection to differentiate transients and loss of coolant accidents. Technical report, University of Pittsburgh, 2018.

[2] M. Brandner D. Watzenig and G. Steiner. A particle filter approach for tomographic imaging based on different state-space representations. *Measurement Science and Technology*, 18, 2007.

[3] Nuclear power reactors. `http://www.world-nuclear.org/information-library/nuclear-fuel-cycle/nuclear-power-reactors/nuclear-power-reactors.aspx`. Accessed: 2019-02-08.

[4] Olga Veksler. Cs434a/541a: Pattern recognition lecture series. `http://www.csd.uwo.ca/~olga/Courses/CS434a_541a/`, 2004.

[5] G. David Garson. Partial least squares: Regression & structural equation models. Technical report, School of Public & International Affairs, North Carolina University, 2016.

[6] Shinji Hasebe Manabu Kano, Shouhei Tanaka and Iori Hashimoto. Monitoring independent components for fault detection. *AIChE Journal*, 49(4), 2003.

[7] Onel Harrison. Machine learning basics with the k-nearest neighbors algorithm, 2018. `https://bit.ly/2OcMqHv`.

[8] Che Rosmani Che Hassan Norazwan Md Nor and Mohd Azlan Hussain. A review of data-driven fault detection and diagnosis methods: applications in chemical process systems. *Reviews in Chemical Engineering*, 2019.

[9] Rolf Isermann. Model-based fault-detection and diagnosis — status and applications. *Annual Reviews in Control*, 29, 2005.

[10] E. A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, pages 153–158, Oct 2000.

[11] Luca Martino Victor Elvira and Christian P. Robert. Rethinking the effective sample size. Technical report, IMT Lille Douai, Universidad Carlos III, Universite PAris Dauphine, 2018.

[12] Arnaud Doucet and Adam M. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. Technical report, University of Warwick, Dec 2008.

[13] Miodrag Bolic Tiacheng Li and Petar M. Djuric. Resampling methods for particle filtering: classification, implementation, and strategies. *IEEE Signal Processing Magazine*, 70, 2015.

[14] D. J. Salmond N. J. Gordon and A. F. M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEEE Proceedings*, 140(2), Apr 1993.

[15] Peter Bickel Thomas Bengtsson and Bo Li. Curse-of-diminesionality revisited: Collapse of the particle filter in very large scale systems. *IMS Collections*, 2:316–334, 2008.

[16] Peter Jan Van Leeuwen. Particle filtering in geophysical systems. In *Mathematical Advances in Data Assimilation*. 2008.

[17] Zhengjiang Zhang and Junghui Chen. Fault detection and diagnosis based on particle filters combined with interactive multiple-model estimation in dynamic process systems. *ISA Transactions*, 2018.

[18] X. Wang and V. L. Syrmos. Interacting multiple particle filters for fault diagnosis and non-linear stochastic systems. In *American Control Conference*, June 2008.

[19] Ping Li and Visakan Kadirkamanathan. Particle filtering based likelihood ratio approach to fault diagnosis in nonlinear stochastic systems. *IEEE Transaction on Systems, Man, and Cybernetics*, 31(3):337–343, Aug 2001.

[20] Xiaoyi Wang Jiping Xu Li Wang Jiabin Yu Zhiyao Zhao, Peng Yao. Reliable flight performance assessment of multirotor based on interacting multiple model particle filter and health degree. *Chinese Journal of Aeronautics*, 2019.

[21] James V. Candy. *Bayesian Signal Processing*, chapter 8. John Wiley & Sons, Incorporated, 2016.