Proceedings of the 45th IEEE International Midwest Symposium on Circuits and Systems
Tulsa, OK August 5-7, 2002

NEURAL NETWORKS APPLIED TO AUTOMATIC FAULT DETECTION

Stefan Jakubek and Thomas Strasser

Institute for Machine and Process Automation
Vienna University of Technology
Gußhausstr. 27-29, 1040 Vienna, Austria

ABSTRACT

The objective of this paper is an automatic fault detection scheme for applications in the automotive industry. In order to increase data reliability and for the purpose of monitoring and control of test equipment a fault detection system based on multivariate data analysis is being developed. The detection scheme has to process up to several hundreds of different measurements at a time and check them for consistency. The main problem lies in the fact that besides the available data no further information is provided. The chosen approach models the distribution function of available fault-free data using ellipsoidal basis function networks. An important requirement for the fault detection scheme is that it should be able to automatically adapt itself to new data. The present paper is focused on this feature. It is demonstrated how a gradient optimization with algebraic constraints can be applied to adapt a pre-existing network to new data points. Numerical examples with actual data show that the proposed method produces good results.

1. INTRODUCTION

In order to achieve maximum productivity, modern measuring systems are now highly sophisticated, characterized by complex interrelationships between sub-systems. The developers' growing need for reliable assessments gives rise to an online monitoring system that automatically checks measured data for consistency. The task is to analyze training data and to build a probability model based on these data using neural networks. However, in a high-tech environment there are often several hundreds of measurements to be processed simultaneously which entails the problem that any model has to be built in a high dimensional measurement space. The algorithm proposed in [1] uses principal component analysis to break down the high dimensionality of the problem. Since no information about functional relationships between data channels is known a-priori, a statistical approach was chosen. The distribution function of the available data-set is estimated using kernel regression techniques. Fault detection is accomplished by checking whether a new data record lies in a cluster of training data or not. Areas or clusters for valid data are determined through kernel regression. In order to reduce the computational effort the distribution functions are approximated by neural networks with ellipsoidal basis functions. In order to use as few basis functions as possible a new training algorithm for ellipsoidal basis function networks is presented. Even during normal operation of the engine test-bed new training data are provided for fault detection and it is required that the system is able to adapt itself autonomously to the new data. The present paper focuses on strategies and algorithms for recursive network training. The rest of the paper is organized as follows: Section 2 gives a brief introduction into the fault detection algorithm in [1]. In section 3 strategies and decision layers for recursive training of an existing network are presented. Section 4 explains how single neurons or groups of neurons can be updated using gradient descent techniques. It is explained how constraints to the gradient can be used to guide the direction of convergence. Finally, the paper is concluded by some numerical results.

2. NETWORK TRAINING

After transforming the measurement channels using PCA one has to find a smooth approximation for the probability density function $f(\boldsymbol{x})$ where $\boldsymbol{x}$ resembles an arbitrary point in the measurement space. A common way to accomplish this is placing a *kernel function* $K(\boldsymbol{x})$ about each data point $\boldsymbol{x}_i$ [2]. In the present application Gaussian kernels were used. With d as the dimension of the data record one gets for the approximation $\hat{f}(\boldsymbol{x})$ of the probability density function

$$\hat{f}(\boldsymbol{x}) = \frac{1}{nh_1 \dots h_d} \sum_{i=1}^{n} \left[\prod_{j=1}^{d} K\left(\frac{x_j - t_{ij}}{h_j} \right) \right]. \qquad (1)$$

Here n denotes the number of available records for training, x_j is the j -th component of $\boldsymbol{x}$, t_{ij} is the j-th component of the i-th data point and $h_1 \dots h_d$ are so-called *bandwidth* parameters. Taking into consideration the large number of data both the sum and the product in (1) make this formula time-consuming to evaluate. Moreover, it is difficult or even impossible to discover structures, like isolated clusters in the data. Both problems give rise to develop a powerful method to reduce the number of parameters in $\hat{f}(\boldsymbol{x})$ whilst keeping the degree of information almost constant. A neural network with ellipsoidal basis functions can be used to approximate the map $\boldsymbol{x} \rightarrow \hat{f}(\boldsymbol{x})$.

2.1. Neuron and network structure

In our design a neuron $n(\boldsymbol{x})$ has an ellipsoidal Gaussian basis function:

$$n(\boldsymbol{x}) = \gamma \exp\left(-\frac{1}{2} \tilde{\boldsymbol{x}}^T \boldsymbol{A} \tilde{\boldsymbol{x}} \right) \qquad (2)$$

The weight is denoted as γ and $\tilde{\boldsymbol{x}}$ denotes the vector from the neuron's center $\boldsymbol{\mu}$ to $\boldsymbol{x}$: $\tilde{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{\mu}$. The matrix $\boldsymbol{A}$ is symmetric and contains "spread parameters" which are determined using a Taylor approximation of the target function about the neuron's center. With the use of this neuron structure the network output is calculated in

the following way (where n_{neu} is the number of neurons):

$$\tilde{f}(\boldsymbol{x}) = \sum_{i=1}^{n_{neu}} \gamma_i \exp\left(-\frac{1}{2}\tilde{\boldsymbol{x}}^T \boldsymbol{A}_i \tilde{\boldsymbol{x}}\right) \tag{3}$$

Now it is possible to obtain clusters in the training data for the plausibility check. Details about this algorithm can be found in [1].

3. RECURSIVE NETWORK TRAINING STRATEGIES

If a record of new data points arrives, it is subdivided into two classes. For this purpose it is investigated, whether a data point fits into one of the existing models. Every model uses one or more principal axes, that span the *model space*. If the column vectors of the matrix $\boldsymbol{S}_i$ span the model space of the i-th model, and $\boldsymbol{x}_k$ is a new data point, the error vector $\boldsymbol{e}_{i,k}$ describes the orthogonal distance from $\boldsymbol{x}_k$ to the i-th model space [3]:

$$\boldsymbol{e}_{i,k} = \underbrace{\left[\boldsymbol{I} - \boldsymbol{S}_i(\boldsymbol{S}_i^T\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^T\right]}_{\boldsymbol{\Gamma}_i}\boldsymbol{x}_k \tag{4}$$

An error measure can thus be defined as

$$J_{i,k} = ||\boldsymbol{e}_{i,k}||_2^2 = \boldsymbol{x}_k^T\boldsymbol{\Gamma}_i^T\boldsymbol{\Gamma}_i\boldsymbol{x}_k \tag{5}$$

The first decision layer selects those data points that can be assigned to a pre-existing model i (i.e. for these data points for a certain i $J_{i,k}$ falls below a threshold). These points can be used to update existing models. All the remaining data points lie in new areas of the measurement space and are used to train new models. They usually emerge when new domains of the operating envelope are being tested. The next decision layer looks at pre-existing models and the selected data that fit into these models. Again, the data set is separated into those points that lie in the vicinity of existing neurons and those that lie in unpopulated areas. The former are used to update the neuron-parameters, the latter are taken into account by placing new neurons.

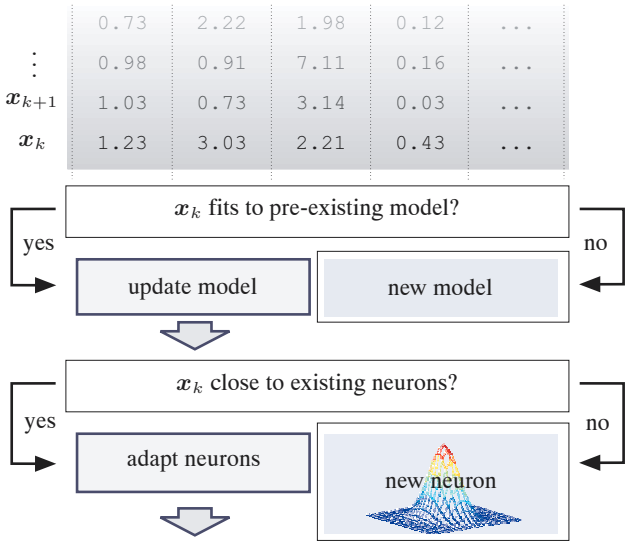

Figure 1: Decision layers for network update

4. UPDATING NEURONS

As mentioned above after new training data are available an update of the existing network is necessary. For that case we develop $n(\boldsymbol{x})$ into a Taylor series with respect to the parameters γ and $\boldsymbol{A}$:

$$n(\boldsymbol{x}) \doteq n_0(\boldsymbol{x}) + \left.\frac{\partial n(\boldsymbol{x})}{\partial \gamma}\right|_0 (\gamma - \gamma_0) + \sum_{i,j}\left.\frac{\partial n(\boldsymbol{x})}{\partial a_{i,j}}\right|_0 (a_{i,j} - a_{i,j,0}) \tag{6}$$

Explicit formulas for the derivatives are:

$$\left.\frac{\partial n(\boldsymbol{x})}{\partial \gamma}\right|_0 = \frac{n_0(\boldsymbol{x})}{\gamma_0}, \tag{7}$$

$$\left.\frac{\partial n(\boldsymbol{x})}{\partial a_{i,i}}\right|_0 = n_0(\boldsymbol{x})(-0.5\tilde{x}_i^2), \quad \left.\frac{\partial n(\boldsymbol{x})}{\partial a_{i,j}}\right|_0 = n_0(\boldsymbol{x})(-\tilde{x}_i\tilde{x}_j) \tag{8}$$

With the desired output p_k of the network for data point # k the network error becomes

$$e_k = \sum_{i=1}^{n_{neu}} n_i(\boldsymbol{x}_k) - p_k \tag{9}$$

where $n_i(\boldsymbol{x}_k)$ stands for the output of neuron i. Next, the network is subdivided into those neurons to be adapted ($^a n(\boldsymbol{x})$) and those to remain unchanged ($^u n(\boldsymbol{x})$):

$$e_k = \sum_{i=n_{a1}}^{n_{al}} {}^a n_i(\boldsymbol{x}_k) + \underbrace{\sum_{i=n_{u1}}^{n_{um}} {}^u n_i(\boldsymbol{x}_k) - p_k}_{\nu_k} = \sum {}^a n(\boldsymbol{x}_k) - \nu_k. \tag{10}$$

Here, n_{al} and n_{um} denote the number of neurons to be adapted or unchanged respectively. After a variation of neuron parameters γ and $a_{i,j}$ the new predicted network error e_{k+} at data point k becomes

$$e_{k+} = e_k + [\nabla^a n(\boldsymbol{x}_k)]^T \cdot \boldsymbol{\vartheta} \tag{11}$$

where $\boldsymbol{\vartheta}$ stands for a vector of parameter variations for one neuron, $\nabla^a n(\boldsymbol{x}_k)$ is the column vector containing all parameter derivatives of one neuron as given in (7) to (8). Forming a column vector ${}_a\triangledown\,(\boldsymbol{x}_k)$ of parameter derivatives for all neurons ${}^a n_i(\boldsymbol{x}_k)$ we get

$${}_a\triangledown\,(\boldsymbol{x}_k) = \begin{bmatrix} \nabla^a n_{a1}(\boldsymbol{x}_k) \\ \nabla^a n_{a2}(\boldsymbol{x}_k) \\ \vdots \\ \nabla^a n_{al}(\boldsymbol{x}_k) \end{bmatrix} \tag{12}$$

and with a column vector $\boldsymbol{\theta}$ of parameter variations for all neurons ${}^a n(\boldsymbol{x}_k)$ one gets

$$e_{k+} = e_k + {}_a\triangledown^T \cdot \boldsymbol{\theta} \tag{13}$$

For a gradient descent one has to choose

$$\boldsymbol{\theta} = {}_a\triangledown\rho. \tag{14}$$

with $\rho \in \mathbb{R}$. Then, e_{k+} becomes

$$e_{k+} = e_k + ({}_a\triangledown^T\,{}_a\triangledown) \cdot \rho \tag{15}$$

The goal is to reduce the network error at one specific data point k by a certain percentage:

$$e_{k+} = \lambda e_k \tag{16}$$

For $\lambda = 0$ the network is completely adapted to the data point, for $\lambda = 1$ no adaptation to the current data point is carried out. With this approach we get for ρ:

$$\rho = (\lambda - 1)e_k / ({}_a\nabla_a^T \nabla) \tag{17}$$

4.1. Neuron weighting

It turned out, that the derivatives with respect to the spread parameters $\frac{\partial n(\boldsymbol{x})}{\partial a_{i,j}}\Big|_0 = n_0(\boldsymbol{x})(-\tilde{x}_i\tilde{x}_j)$ are almost always significantly higher than $\frac{\partial n(\boldsymbol{x})}{\partial \gamma}\Big|_0 = \frac{n_0(\boldsymbol{x})}{\gamma_0}$. In the vicinity of a neuron's (say Neuron n_l) center, however, this leads to the situation that the gradient with respect to the $a_{i,j}$ of a neighbouring neurons can have more influence on the network output, than parameter changes of n_l itself. Adapting only parameters of n_l also has the advantage, that the influence of the data point is more bounded. Therefore, if a data point $\boldsymbol{x}_k$ lies in the vicinity of a neuron, instead of the actual gradient, an alternative gradient is used, that emphasizes this neuron. A data point $\boldsymbol{x}_k$ is said to lie in the vicinity of a neuron j if

$$\exp(-\frac{1}{2}\tilde{\boldsymbol{x}}^T \boldsymbol{A}_j \tilde{\boldsymbol{x}}) > \xi. \tag{18}$$

Here, ξ is a "proximity threshold", in practice $\xi = 0.8$ turned out to be a good value. If the above condition (18) is fullfilled, say for neuron 2, then instead of $\boldsymbol{\theta}$ as given in (14) one has to use

$$\boldsymbol{\theta} = \underbrace{\begin{bmatrix} \nabla^a n_{a1}(\boldsymbol{x}_k) \\ 100\nabla^a n_{a2}(\boldsymbol{x}_k) \\ \vdots \\ \nabla^a n_{al}(\boldsymbol{x}_k) \end{bmatrix}}_{{}_a\nabla_{mod}} \rho. \tag{19}$$

In (19) the weight of neuron 2 raised by a factor of 100 against the remaining neurons which has the effect that mainly the parameters of this neuron are used to adapt the network to data point $\boldsymbol{x}_k$. Using (19) the scaling factor ρ now becomes

$$\rho = (\lambda - 1)e_k / ({}_a\nabla^T \, {}_a\nabla_{mod}). \tag{20}$$

4.2. Fixed points

Changes both of the neuron weight γ and of the spread parameters $a_{i,j}$ entail a more or less global change of the neuron output. In many cases this is not desirable, since we only want to adjust the network output locally. Therefore it is desirable to adapt one or more neurons to a data point as effectively as possible (e.g. using gradient methods) and at the same time ensure that the overall change of the network output remains within some limits. This can be achieved by introducing **fixed points** $\boldsymbol{p}_f$. The concept is to ensure that the output of a neuron (say neuron 2) remains unchanged at a fixed point, and, under this constraint find an "optimal" vector $\boldsymbol{\theta}$ of parameter variations. Computing the gradient ${}_a\nabla(\boldsymbol{p}_f)$ at the fixed point $\boldsymbol{p}_f$ after adaptation to the k-th data point the neuron output at $\boldsymbol{p}_f$ changes as follows:

$$n_{k+}(\boldsymbol{p}_f) - n_k(\boldsymbol{p}_f) \doteq {}_a\nabla(\boldsymbol{p}_f) \cdot \boldsymbol{\theta} \tag{21}$$

Since we only want to affect the output of neuron 2 ${}_a\nabla(\boldsymbol{p}_f)$ becomes

$${}_a\nabla(\boldsymbol{p}_f) = \begin{bmatrix} \boldsymbol{0} \\ \nabla^a n_{a2}(\boldsymbol{p}_f) \\ \boldsymbol{0} \\ \vdots \end{bmatrix} \tag{22}$$

In order to leave the neuron output unchanged one simply has to ensure that ${}_a\nabla(\boldsymbol{p}_f) \cdot \boldsymbol{\theta} = 0$ i.e. $\boldsymbol{\theta}$ must lie in the null space of ${}_a\nabla(\boldsymbol{p}_f)$ as was suggested in [4, 5]:

$$\boldsymbol{\theta} \in \mathcal{N}({}_a\nabla(\boldsymbol{p}_f)) \tag{23}$$

Usually, there is more than one fixed point to be observed so that a matrix $\boldsymbol{\Phi}$ is constructed by stacking the ${}_a\nabla(\boldsymbol{p}_{f,i})$ of all fixed points $\boldsymbol{p}_{f,i}$ as row vectors:

$$\boldsymbol{\Phi} = \begin{bmatrix} {}_a\nabla^T(\boldsymbol{p}_{f,1}) \\ {}_a\nabla^T(\boldsymbol{p}_{f,2}) \\ {}_a\nabla^T(\boldsymbol{p}_{f,3}) \\ \vdots \end{bmatrix} \tag{24}$$

Again, a suitable choice of the variation vector $\boldsymbol{\theta}$ must lie in the right null -space of $\boldsymbol{\Phi}$. Let $\boldsymbol{N}$ denote a matrix whose columns span $\mathcal{N}(\boldsymbol{\Phi})$, e.g. $\boldsymbol{\Phi} \cdot \boldsymbol{N} = \boldsymbol{0}$. The optimal projection ${}_a\nabla_{opt}$ of ${}_a\nabla_{mod}$ on $\mathcal{N}(\boldsymbol{\Phi})$ thus yields (see [5])

$${}_a\nabla_{opt} = \boldsymbol{N}(\boldsymbol{N}^T\boldsymbol{N})^{-1}\boldsymbol{N}^T \cdot {}_a\nabla_{mod} \tag{25}$$

Using $\boldsymbol{\theta} = \rho \, {}_a\nabla_{opt}$ one gets for ρ

$$\rho = (\lambda - 1)e_k / ({}_a\nabla^T \, {}_a\nabla_{opt}) \tag{26}$$

The way ${}_a\nabla_{opt}$ is determined in (25) assures that the variation vector θ lies as closely to ${}_a\nabla_{mod}$ as possible while satisfying all constraints. It remains to determine a reasonable amount of fixed points for each neuron. The following idea produced good results: If d denotes the dimensionality of the network then d fixed points are chosen on the contour ellipse defined trough $-\frac{1}{2}\tilde{\boldsymbol{x}}^T\boldsymbol{A}\tilde{\boldsymbol{x}} = \alpha$ with $\log\alpha \in]0;1[$. Ideally, these fixed points are placed on the vertices of the contour ellipse (Fig. 2). Thus, the neuron is constrained along the whole contour line (indicated red in Fig. 2) during the parameter update. The logarithmic constant α determines the contour line to be fixed. The smaller α is chosen ($\sim \log(0.2)$), the less global are the effects on the neuron output due to parameter variations. On the other hand, lager values of α ($\alpha > \log(0.7)$) can lead to significant global changes of the neuron shape as can easily be imagined. There

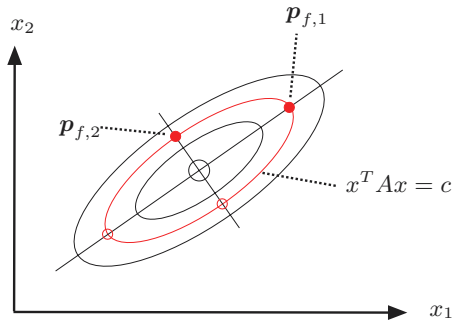

Figure 2: Choosing fixed points on contour lines

is one critical situation when a data point lies in the vicinity of a fixed contour line. Then, the denominator in (26) tends towards zero (${}_a\nabla_a^T\nabla_{opt} \to 0$) which leads to numerical difficulties and parameter divergence. In this case one either has to do without fixed points when adapting to this particular data point (which bears the risk of unreasonable parameter changes) or the network is not updated to the data point at all. For the sake of stability of the algorithm we used the latter alternative.

5. RESULTS

Figure 3 shows an example for a one-dimensional network consisting of two neurons. The data that the network should adapt to are indicated by black dots. It can be seen that the fixed points keep the effects of adaptation local to the data points. Figures 4 and 5 depict a

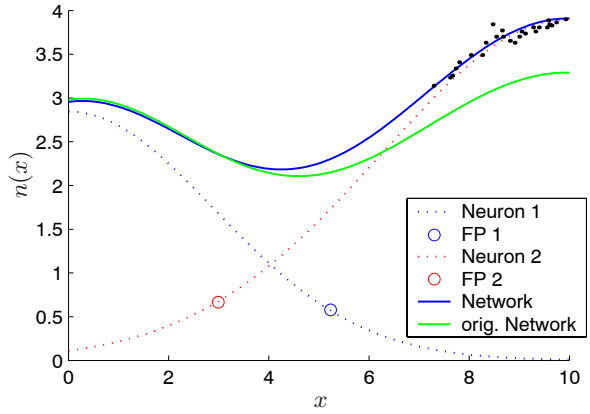

Figure 3: One-dimensional example

network trained with actual data before and after adaptation to a new data record. The model was built on PCA-channels one and two. Black dots represent the desired new network output p_k. It can be seen that the adaptation produces satisfactory results.

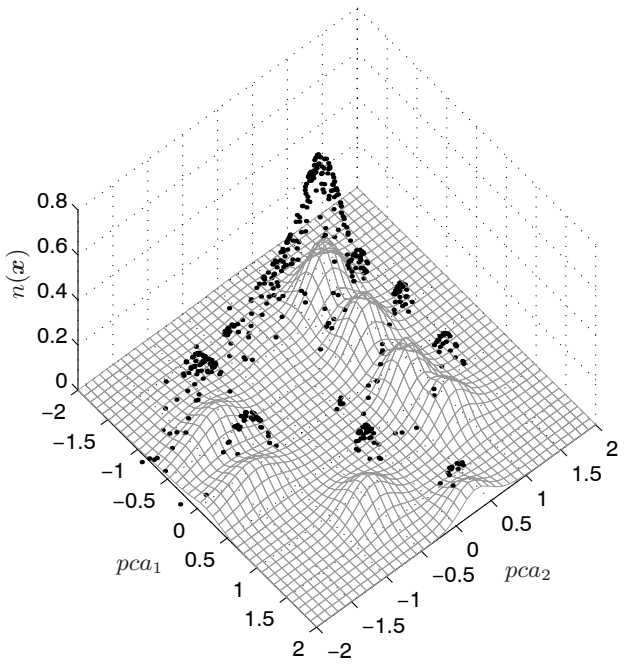

Figure 4: Two-dimensional example, prior to adapation

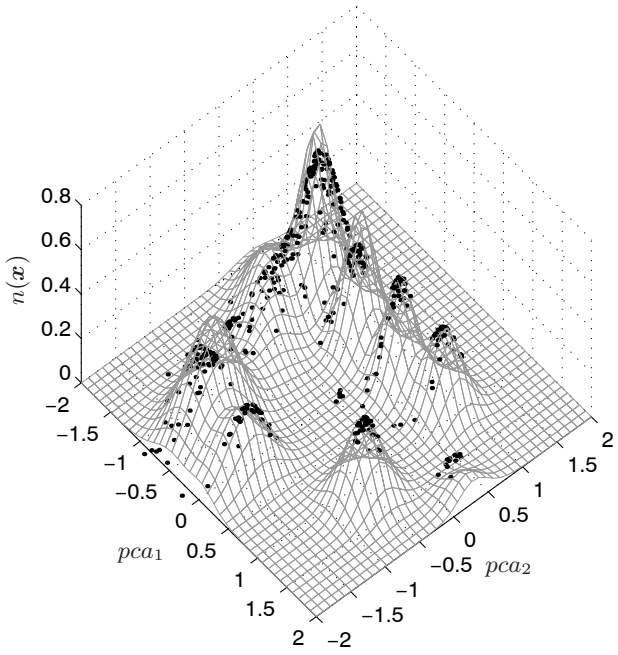

Figure 5: Two-dimensional example, after adaption

6. CONCLUSION

In this paper an approach for fault detection in automotive measuring systems was presented that automatically adapts itself if new data are available. The main problem was to develop an algorithm for adapting neurons to a new target function. For this purpose we introduced an approach for parameter adaption using gradient descent, neuron weighting and fixed points. Application to both a simulated example and to real data show that the proposed method yields good results. The next development stage features an extended gradient method that moves the centers of existing neurons in addition to the parameter changes already demonstrated in this paper. Moreover it is planned to replace the PCA in the preprocessing step by another suitable method.

7. REFERENCES

[1] S. Jakubek and T. Strasser. Fault-diagnosis using neural networks with ellipsoidal basis functions. In *Proceedings of the 20th American Control Conference (ACC 2002), Anchorage, AK*, pages 3846–3851, May 2002.

[2] Kyle S. Cranmer. Kernel estimation for parametrization of discriminant variable distributions. *http://www-wisconsin.cern.ch/∼cranmer/keys.html*, 1999.

[3] J. Chen and R.J. Patton. *Robust Model-Based Fault Diagnosis for Dynamic Systems*. Kluwer Academic Publishers, Boston/Dordrecht/Londonnn, 1999.

[4] S. Schweid and T.K. Sarkar. Projection minimization techniques for orthogonal QMF filters with vanishing moments. *IEEE Transactions on Circuits and Systems II*, 42:694–701, 1995.

[5] T.J. Hebert and Keming Lu. Expectation-maximization algorithms, null spaces, and MAP image restoration. *IEEE Transactions on Image Processing*, 4:1084–1095, 1995.