Assessing neural networks for sensor fault detection

Georg Jäger, Sebastian Zug, Tino Brade, André Dietrich, Christoph Steup
Otto-von-Guericke Universität Magdeburg
Department of Distributed Systems
Magdeburg, Germany
Email: gjaeger@st.ovgu.de,
{zug, brade, dietrich}@ovgu.de

Ana-Maria Cretu
Universite du Quebec en Outaouais
Departement of Computer Science
Canada
Email: ana-maria.cretu@uqo.ca

Christian Moewes
Otto-von-Guericke Universität Magdeburg
Institut of Knowledge and Language Engineering
Magdeburg, Germany
Email: cmoewes@ovgu.de

***Abstract*—The idea of "smart sensing" includes a permanent monitoring and evaluation of sensor data related to possible measurement faults. This concept requires a fault detection chain covering all relevant fault types of a specific sensor. Additionally, the fault detection components have to provide a high precision in order to generate a reliable quality indicator. Due to the large spectrum of sensor faults and their specific characteristics these goals are difficult to meet and error prone. The developer manually determines the specific sensor characteristics, indicates a set of detection methods, adjusts parameters and evaluates the composition.**

In this paper we exploit neural-network approaches in order to provide a general solution covering typical sensor faults and to replace complex sets of individual detection methods. For this purpose, we identify an appropriate set of fault relevant features in a first step. Secondly, we determine a generic neural-network structure and learning strategy adaptable for detecting multiple fault types. Afterwards the approach is applied on a common used sensor system and evaluated with deterministic fault injections.

I. Introduction

Future concepts for sensor actuator systems ("Internet of Things" (cf. [1]), "Cyber-Physical Systems" (cf. [2]), or "Pervasive/Ubiquitous Computing" (cf. [3])) are focused on distributed smart devices. They organize themselves based on a current task and aggregate adaptively the needed environment information from available and relevant sensors. The continuous adaptation and reconfiguration breaks with classic design patterns based on a static configuration on design-time. In order to provide the dynamic composition, we have to apply additional concepts related to loosely coupled communication, self-description (data types, physical units, uncertainty), synchronization and fault-handling [4].

The last point is particularly challenging due to the large spectrum of sensor fault types. If an application should be able to weight or validate a sensor measurement correctly, it needs additional validity information. This value indicates the possibility of a fault during the measurement process. Previous publications mapped the result of the most relevant fault detection operation on simple scalar values [5]. We enhanced the concept and developed a vectorized fault indicator for typical sensor fault types summarized in Tab. I. The vector covers all fault types relevant for a certain sensing setup and provides a fine grained abstraction of the measurement validity. Consequently, the approach requires a specific validation algorithm for each fault type that is relevant for the implemented transducer. Related to the different characteristics of the sensor faults (duration, derivation, stochastic properties) a huge amount of methods were presented for fault detection (see [6]). Hence, the developer has to choose an appropriate fault detection method and has to determine the "'magic constants"' (thresholds, weighting factors, limits) implementing a complex fault model for a sensor system.

These manual adjustments do not correspond with our idea of an effective development process for smart sensors. In previous work we describe the use of sensor description files providing an automated software-development process in Mathworks Simulink. Based on a machine readable sensor characteristics (timing, physical units, signal dynamic) and a target description (processor, communication interfaces, ADC properties, etc.) we generate source code for interfaces and a basic processing chain of a smart sensor [7]. If we want to embed the sensor fault handling in this approach, we need to define a general and simplified way to design fault detectors in order to reduce manual interventions and adjustments. At the end, we are able to evaluate sensor description files containing possible fault models and generate and configure automatically an appropriate set of detection methods. To reach this goal, we have to encapsulate the development of a suitable detection methods and their parameters in a more abstract way. Consequently we looked for an approach that

- covers a large spectrum of sensor faults
- based on a parameter set developed in an automatable process and
- can be executed on embedded devices.

In this paper we evaluate the capabilities of Neural Networks for this purpose. Neural Networks can be applied on complex classification problems. The number of parameters that have to be adjusted are limited to some basic parameters like network structure, network type, or transfer functions. Furthermore this values can be determined by cross validation an therefore are also automatable.

Hence, we summarize conventional fault detection approaches and discuss previous implementations based on neural networks in Sec. II. Starting from this point we analyze the required adjustments in the neural network design and configuration process Sec. III. Sec. IV describes a first implementation and evaluates the results. In Sec. V we provide a final conclusion and an outlook on future work.

II. State of the Art

Fault detection represents the first step in the fault diagnosis toolchain of Fault Detection and Isolation (FDI). According to the authors of [8], fault detection only indicates that something went wrong. The correct identification of a certain fault type or its origin follows in separate steps. These extensions are planned for future work, in this paper we concentrate on A general indicator for faulty measurements.

All fault detection strategies require an additional reference. By comparing it with current measurements or their features (noise, deviation, correlation) faults can be recognized. The reference is generated based on

- hardware redundancy (homogeneous or heterogeneous sensors in multi-sensor applications),

- analytical redundancy (mathematical models of the observed system with predictions) or
- signal analysis (knowledge on the measurement characteristic (one or more samples).

Comprehensive discussions on sensor fault detection for all 3 cases are given in [6], [9]. In this paper we address the detection methods for individual smart sensors. Hence, hardware redundancy is not considered in the following paragraphs.

A. Sensor fault detection

Fault detection methods vary from simple threshold checks up to complex signal filter algorithms [10]. The first variation uses an implicit knowledge of the environment model and of the integrated sensor to estimate ranges for a single/multiple set of features. If a detector should be able to recognize outliers, we will implement a gradient check for instance. Its upper limits represents the dynamic of the observed system and the known "normal" noise level of the sensor. Consequently, the developer has to define a suitable feature in a first step and to identify and to evaluate the thresholds in a second step [11]. In many cases the thresholds can not be reliably determined on design-time. Adaptive thresholds can be an appropriate method to cope with this challenge. Instead of validating a value by a predefined constant limit, the thresholds are dynamically arranged. It is common practice to define a constant and a dynamic part in this case. This method guarantees a higher flexibility but requires more effort for adjustment and calibration.

The second type of methods uses filters to generate a residual signal which can again be thresholded to detect a sensor fault. Typically spectral filters are used to detect sensor fault signals, too [12]. Digital filters such as the Savitzky-Golay filter [13] can also be used to generate a residual signal. Again expert knowledge is necessary to design such filters. Other filters are based on the physical laws of motion, e.g., the Kalman filter that can even detect faults in real time [14].

An alternative form of redundancy evaluates mathematical models. The developer describes the environment by balancing equations or rules and transforms this information into system state models, event/situation-calculus, state machines etc. The fault detection algorithm analysizes the deviations and looks for a significant deviation. A large number of modeling techniques and residual validation are available tailored for specific scenarios, sensor parameters, or communication aspects [10], [15].

The previously mentioned techniques originate from digital signal processing and/or physics. All of these methods have to be tuned manually and thus generally depend on human expert knowledge. Especially in complex sensor-based systems, the definition of analytical fault detection methods is either inconsistent or time-consuming. Short production cycles require methods that learn to detect faults solely based on observations.

Such data-driven approaches demand representative training sets that include normal states and faults, both labeled as such [16]. Then any statistical learning method can be used to learn such a model. Rule-based methods, e.g., decision trees [17], return threshold-based if-then rules but do not perform so well. Black-box models such as artificial neural networks generalize very well and therefore more appropriate.

B. Sensor fault detection with neural networks

A historical example of neural networks for pattern recognition are *Probabilistic Neural Networks (PNN)*, first introduced 1990 by Donald F. Specht[18]. The learning process of this kind of neural networks is replaced by the generation of the PNN, because every sample of the training data is transferred to a neuron and therefore saved inside the resulting neural network. New samples are classified by calculating a similarity to every class. The sample is assigned to the class with the highest similarity. They are used for sensor fault detection in a modified way by A. Jabbari et al.[19]. His PNN's were applied to monitor temperature sensors in a cold chain for food. Combining current temperature measurements and additional environment information a, PNN was able to distinguish between normal states (temperature varies due to air exchanges while the door opening) and faulty transducers.

Time-Delay Neural Networks (TDNN) are another type which were applied in common fault detection by Christensen et al. [20]. The paper describes the detection of hardware faults for autonomous robot systems. They consider that hardware faults change the flow of sensory data and also the reaction of the control program. These changes are detectable by a TDNN. The detection method is evaluated in three different tasks performed by real robots. The structure of a TDNN is basically the same as a *Multi-Layer-Perceptron (MLP)*. This means there are only forward connections. Therefore standard backpropagation can be used to learn this kind of neural network. TDNN are able to detect patterns in time series by analyzing sliding windows of a signal. The length of the window must be static as a tradeoff to the feed-forward characteristic.

To overcome the disadvantage of a static window length, *Locally Recurrent Neural Networks (LRNN)* were applied to fault detection in [21]. The authors tested their system on a model consisting of three watertanks. Informations on different sensors were available to detect faults inside the model. Locally Recurrent Neural Networks don't have recurrent connections between neurons but inside of special neurons. These neurons are called *dynamic neurons* and have an additional *linear dynamic system (LDS)* which transmits the output back to the input. With this recurrent connection, LRNN are able to deal dynamically with time series.

Another approach with recurrent connections were introduced by Hochreiter and Schmidhuber in [22]. They did not define a type of neural networks, but a module, called *memory cell* which can be applied to all neural network types. They used different neurons and recurrent connections in a way that memory cells are able to decide which information to store, at which time and how long to store this information. Furthermore they can decide when to show the stored information to the rest of the neural network. These modules could be a part of a neural network for fault detection, as it enables the network to analyze time series dynamically. An implementation of the memory cell concept on sensor fault detection tasks is not known yet. After intensive literature research, we could not find works on fault detection for single sensor systems based on neural networks. Most works on single sensor setups use additional information about their environment.

III. Our approach

The implementation of a neural network requires a number of basic steps. The left side of Tab. II summarizes the procedure from input data analysis up to training and evaluating of a neural network. We recognize the need for a subdivision while applying the concept on fault detection tasks. The third column (Index) of Tab. II assigns an index number that references the following subsections, where we discuss the intended steps:

Table I
CATEGORIES OF FAULTS IN SENSING APPLICATIONS [23]. THE DASHED LINE ILLUSTRATES THE PROGRESS OF A PHYSICAL VALUE. IN CONTRAST, THE SOLID GRAPH DEPICTS THE CORRESPONDING FAULTY MEASUREMENTS.

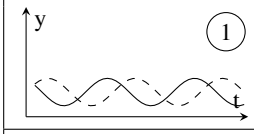
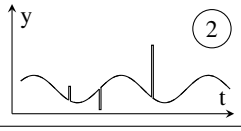
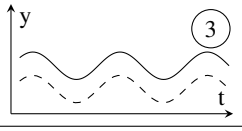
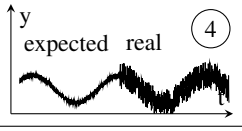
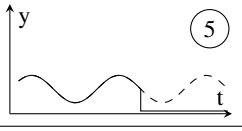
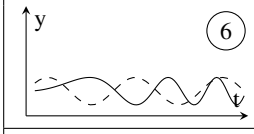
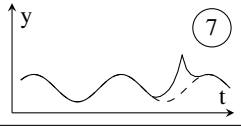
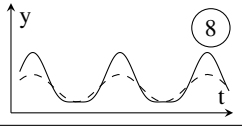
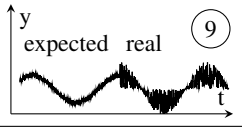
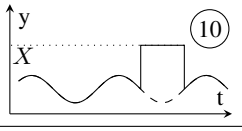
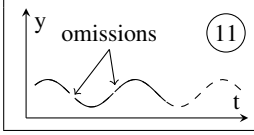
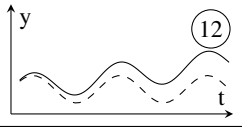
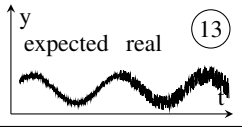
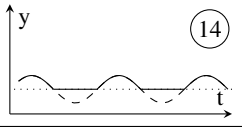
Delay	Offset			Stuck-at
	sporadic	permanent	stochastic	
constant 	outlier 	constant 	constant 	at zero 
variable 	spike 	value correlated 	value correlated 	at X 
omissions / broken link 		time correlated 	time correlated 	saturation 

Table II
ADAPTATION OF THE COMMON DEVELOPMENT PROCESS FOR NEURAL NETWORKS RELATED TO SENSOR FAULT DETECTION

Common development steps	Adaptation for sensor fault detection	Index
Input data analysis	Selection of relevant fault types	A
	Definition of an environment model	B
	Derivation of appropriate features	C
Acquisition of a sample data base	Generation of measurement samples superimposed by selected faults	D
Selection of a suitable network type	Weighting of neural approaches considering the specific setup	E
Train the network		F
Evaluation		G

A. Fault selection

For each fault detection application we need an appropriate fault model considering all possible deviations between real values and sensor measurements. A comprehensive classification of sensor data centric fault models is given in [23]. Fig. I illustrates the major types organized in relation to the correlation, duration or amplitude characteristic. We distinguish 14 different fault types such as outliers, offsets or additional noise. The mentioned sensor description involves an identification of the relevant sensor type and their mathematical parameter.

At the moment, we just consider 4 fault types : outliers (2), constant offset (3), noise (4) and Stuck-at-Zero (5). The selection covers the fault characteristic of many sensing devices. Hence, we want to evaluate our ideas based on this subset and integrate other fault types later.

B. Environment Model

Additional to the fault model, we have to consider an appropriate environment model describing the application context of the sensor. The environment model characterizes the non-faulty state and defines the ranges of the measurement value as well as the dynamics of the monitored system.

In order to use neural networks, the environment model has to be described implicitly by samples in the database/training data. Hence, creating a mathematical environment model is replaced by generating/collecting data for training a neural network.

C. Relevant features

Preprocessing is one part of input data analysis which can increase performance of neural networks dramatically. In fault detection preprocessing is also called feature extraction. Features [10] are additional information calculated from raw measurement signals. Therefore during this step the developer has to choose the relevant features as input of the neural network. One main challenge is to identify the best composition of available feature related to the faults we want to detect. To decide which feature to use, some requirements are given in [24]. They should be:.

- computable efficiently.
- uncorrelated with other features.
- independent of external influences.
- characterized by high differences between features and small internal differences.

Along with these conditions, we look for a minimal number of features so that all faults considered in the fault model are covered. The features have to decouple sensor measurements and fault detection methods. These abstraction guarantee the applicability of the approach on a wide range of sensor types. We selected the following features that will be used as an input for our neural network. The variable x_t represents measurement samples, $\overline{x}_t$ the mean value and T the sliding window length.

Mean

Calculating mean enables our system to recognize

time-correlated faults. Furthermore a mean value identifies on usual values of a signal. If the current value differs strongly from mean, a faulty measurement can be assumed. We compute the mean value over a sliding window with length *T*. Hence, mean is defined by

$$E(x_t) = \overline{x}_t = \frac{1}{T} \cdot \sum_{\tau=0}^{T-1} x_{t-\tau}$$

Standard-Deviation

The Standard Deviation quantifies the width of a probability distribution and defines the expected deviation of a measurement related to the mean. For parametric distribution functions we can calculate the probability of the current difference from mean. Standard Deviation is defined for a sliding window with length *T* by:

$$s(x_t) = \sqrt{\frac{1}{T} \cdot \sum_{\tau=0}^{T} (x_{t-\tau} - \overline{x}_t)^2}$$

Deviation

The first deviation reflects the dynamic of the observed system. The value allows a neural network to recognize outliers, spikes etc. The deviation can be calculated by:

$$\frac{dx}{dt} = \frac{x_t - x_{t-1}}{\Delta t}$$

Signal-to-Noise Ratio

The Signal-To-Noise-Ratio (*SNR*) allows to estimate the noise level of a signal. In literature it is often defined as signal power divided by noise power [25]. However, to compute an running *SNR* we apply the following definition

$$SNR(x_t) = \frac{E(x_t)}{s(x_t)}$$

Correlation-Coefficient

The Correlation-Coefficient describes the similarity of two signals. Therefore the correlation-coefficient is defined as [25]:

$$r_{xy} = \frac{E(x_t - y_t) - E(x_t) \cdot E(y_t)}{s(x_t) \cdot s(y_t)}$$

Furthermore the Correlation-Coefficient allows to derive a functional relation between to signals. As the coefficient is in the [-1,1] interval, it can be interpreted as:

- $r_{xy} > 0$: high values in x yield high values in y.
- $r_{xy} < 0$: high values in x yield low values in y.
- $r_{xy} = 0$: x and y are not correlated.
- $r_{xy} = 1$: x,y are linear correlated: $y = a \cdot x + b; a > 0$.
- $r_{xy} = -1$: x,y are linear correlated: $y = a \cdot x + b; a < 0$.

Additional transformations of signal like Fourier-Transformation and power-density spectrum are possible features that need to be investigated. Tab. III maps our fault models of Tab. I on the mentioned features. The tabular evaluates the detection capabilities of individual features regarding outliers, offsets, etc. The assignment represents a first hypothesis that has to be proven in future work.

Table III
HYPOTHETICAL RELEVANT FEATURES FOR SENSOR FAULT CATEGORIES

ID	Fault category	Mean	Variance	Deviation	Corr.-Coefficient	Sig.-to-N. Ratio
①	Constant Delay					
②	Outlier			●		●
③	Constant Offset	●				
④	Constant Noise		●	●		●
⑤	Stuck-At-Zero		●	●	●	
⑥	Variable Delay				●	
⑦	Spike			●		
⑧	Value-Correlated Offset		●			
⑨	Value-Correlated Noise		●			●
⑩	Stuck-At-X		●	●		
⑪	Omission	●	●	●	●	●
⑫	Time-Correlated Offset	●			●	
⑬	Time-Correlated Noise		●			●
⑭	Saturation			●	●	●

For this paper we chose 4 fault types which are detectable by our feature set. In further investigations we will extend this number.

D. Generation of data samples

As the neural network will implicitly generate an environment model during learning, the collected samples need to represent the data produced by the final wokring system. One possibility to generate these samples is to set up a real sensor system and collect the measurements. In a post processing step all measurements have to be (manually) classified as faulty or correct. However this approach has two major drawbacks. Firstly, some faults occur very seldom or depend on specific environment conditions. Hence, data acquisition needs to run for a long time to capture them. Secondly, the manual classification is an extensive work especially for large data sets. Alternatively, a fault injection framework handling all relevant fault types is more effective. In this case a schedule of faults to happen in the simulation is used as input. The the fault injection tool creates the data based on a simulated system behavior and a defined fault characteristic for each fault in the schedule. Another possibility is to employ the fault injection on real sensor measurements. However possible real measurements faults compicate the classification in this approach.

E. Neural network type

Feature extraction is a preprocessing step of fault detection. Hence, we have to implement an appropriate detection method, that is able to cover all relevant fault types by one approach, namely neural networks.

The type of neural network is a basic parameter. As shown in Sec. II there are many types of neural networks which could be suitable for sensor fault detection. As it is a known problem of recurrent neural networks to learn these, for this work we concentrate

on feedforward neural networks. Time-Delay Neural Networks are one possible choice. They are easy to train but can even analyze time-series in a bordered range.

Besides the type of neural network we have to define other parameters, e.g., the number of hidden layers and the number of neurons in every hidden layer. This is typically done by cross-validating different neural network structures. Choosing the best number of layers, special attention should be paid to the "curse of dimensionality" [16]. This phenomenon describes different aspects of high dimensional data. For instance, with an increasing number of dimensions, the distances of neighboring points increases too. Thus, with every additional hidden layer that maps input data in a higher-dimensional feature space, this effect makes it harder to establish suitable separating hyperplanes. Another aspect of this phenomenon, and a burden for any machine learning method, is the fact that we need an exponentially growing number of data points to estimate the parameters of additional hidden layers properly [16]. Thus a higher number of hidden layers potentially decreases the generalization performance of the network.

F. Training of the neural network

The last important parameter is the training function and its parameters. that can determined by different approaches. A number of algorithms applies the gradient descent, as the Broyden-Fletcher-Goldfarb-Shanno(BFGS) algorithm, a quasi-Newton backpropagation algorithm, which is an derivation of standard backpropagation [26]. This algorithm is similar to the Newton Method, but computes the derivations only approximately which increases the speed of training. Unfortunately it has to save the Hessian matrix which is $n \times n$ where n is the number of weights and biases used in the neural network. For large networks this matrix will increase quadratically. One benefiting parameter of this training function is the optional weight decay parameter. This parameter is used while learning to force the weights of unused neurons to zero. The number of neurons can be defined in a wider spectrum and the decision on how to choose the number of neurons per layer is easier. The value of this parameter can be examined with cross-validation again. Another important parameter of every training function is the goal of performance. The training of neural network will stop, when this value has been reached. Zero isn't a good goal for training a neural network, because of generalization problems and overfitting. To avoid overfitting, an appropriate number of epochs has to be chosen.

G. Evaluation and validating the neural network

Evaluating a neural network requires an error function. Known variants are mean squared error, sum squared error or mean absolute error. As we want to perform an classification-task, a more appropriate error function is needed. Precision and recall are commonly used in this context:

$$\text{Precision} : \frac{t_p}{t_p + f_p} \text{ and Recall} : \frac{t_p}{t_p + f_n}$$

where t_p denotes all faulty samples that were classified correctly (*true positives*). f_p references *false positives*, whose classification is wrong by assuming a fault. The counterpart is marked by f_n, faulty samples that were classified as fault free. Therefore precision is an indication of how much of the returned faults are correct classifications. Recall only indicates how many of the occured faults are detected. Both values can be combined to the so-called *F-Score*. There are different approaches for this combination, but commonly the *F1-Score* is used.

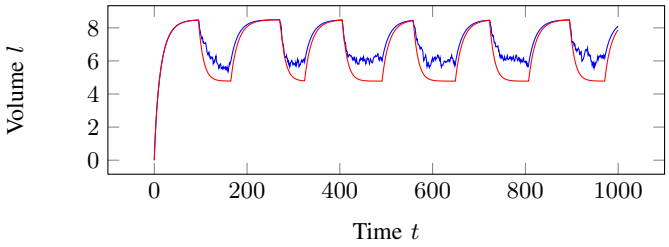


Figure 1. Example of a fault free sensor measurement; The theoretical model (red) and the real system behavior with a noisy random outflow (blue)

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

In the end, the assessment of a neural network can be quantified by a single value. This concentrated representation supports our goal of an automated configuration process of a fault detector.

IV. Evaluation

We evaluated our approach based on the commonly used water tank example. It provides an intake and a drain. The inflow is constant over time, while the outflow depends on the current height of the liquid inside the water tank. Additional we integrate a random outflow with a specific mean, variance and length. The second outflow is triggered periodically and represents the uncertainty of our environment model. Fig. 1 shows an example of a fault free sensor run.

Our water tank model and a level sensor are simulated. It is assumed that just one sensor fault may occur at one point in time. As described in section III-A we consider only 4 fault types. The fault state is calculated in a fault injection framework and applied on the simulated sensor measurements from the water tank model. This guarantees, that only one fault can occur at a time. By simulating this setup with different fault parameters we generate samples and targets for learning and validation. Beside the raw sensor measurements also the features (mean, standard-derivation, gradient, Signal-to-Noise-Ratio and Correlation-Coefficient) were saved. If a window length was needed to calculate a feature, we defined $T = 64$. We chose a Time-Delay Neural Networks and applied the BFGS algorithm for learning. Furthermore TDNN are able to analyze time series by presenting time slots. In this case, the network has to analyze six sliding windows in parallel, every window representing one feature. As the input of the TDNN will be a sliding window of a sensor measurement, its output should be a sliding window too. Inside this window every time step is marked as faulty(1) or not(0). We trained TDNN with 2 hidden Layers. In order to find the best structure 3 neural network with different numbers of neurons per layer $[25, 50, 75]$ were created. All networks were trained with the BFGS algorithm. We used the mean squared error as an error function. As cross validation give the best regularization values between 0.04 and 0.06 all network structures were trained with a regularization parameter of 0.04, 0.05 and 0.06. The transfer function of every neuron in a hidden layer is the hyperbolic tangent.

To compare the resulting neural network with common methods, we implemented a Limit Checking (LC) fault detector for every fault. Outliers were detected by checking the gradient. If the current sensor value is equal to zero, an Stuck-at-Zero fault is detected. To detect a Constant Offset, the mean was checked and Noise was recognized by evaluating the current Signal-to-Noise ratio. With precision and recall we were able to compare both methods. We used 25% of the database to estimate this values, 75% were used to train the TDNN. In Table IV

the results of comparison are shown. The first value is the precision of the specific fault detector assigned to the detected fault. The second value is the corresponding recall. As LC is implemented only for one fault type at a time, the "-"-sign indicates that no measurements of precision or recall for the other fault types were produced. As the reader can see, TDNNs were not able to detect specific faults more reliable, but are coequal in precision and recall. Faults like Stuck-at-Zero appeared to be recognizable easy, against faults like Outlier. Outlier were only detectable with a precision of 0.1 and recall of 0.02. This could be caused by the random outflow of the water tank. The blue line in Fig. 1 shows an exemplary sensor measurement with a noisy random outflow(e.g. from $t = 100$ to $t = 180$). Timesteps of this random outlow seems to be equal to Outliers. Therefore the distance between fault free timesteps and Outlier is very small(the euclidean distance is sometimes less than 0.001), so that the neural network cannot distinguish between faulty and fault free data points. LC of the gradient produces only a precision of 0.34 and a recall of 0.39 too. Another problematic fault is the Constant Noise. The TDNN reached only a precision of 0.47 and a recall of 0.4. As the LC obtains a recall of 1 but also a precision of 0.48 may the Signal-to-Noise ratio is not a appropriate feature to detect Noise. Nevertheless Stuck-at-Zero and Constant Offset could be detected reliably. Hence, we were able to create one neural network to detect different faults.

Table IV
COMPARISON OF DIFFERENT COMMON FAULT DETECTION METHODS WITH A TDNN, (PRECISION/RECALL)

	Fault Detection Method				
		Limit Checking on			
	TDNN	Gradient	Mean	Signal-to-Noise Ratio	raw signal
Outlier	0.1/0.02	0.34/0.39	-	-	-
Constant Offset	1/1	-	1/0.97	-	-
Constant Noise	0.47/0.40	-	-	0.48/1	-
Stuck-At-Zero	0.98/0.99	-	-	-	0.99/1

V. CONCLUSION

As one milestone towards an automatic configuration of a fault detector our goal was to show that neural networks provide the capabilities for multi fault detection on single sensor system. Therefore we trained a Time-Delay Neural Network to detect four different fault types. Two of them were reliably detectable, Noise was detected sporadic and Outlier could not be recognized. Nevertheless one neural network was able to detect more than one fault type. We will continue our work by testing other neural network types, such as neural networks with recurrent connections in order to improve results. One promising approach are the *Long-Short Term Memory Cells* [22], which are not applied to fault detection until now, but seems to provide promising capabilities. Furthermore work on how to choose features and which feature can be used to detect specific faults could increase success rate of fault detection too. With greater knowledge about features, selection of these will become more automatable.

ACKNOWLEDGMENT

This work was partially supported by the EU under the FP7-ICT programme, through project 288195 "Kernel-based ARchitecture for safetY-critical cONtrol" (KARYON).

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
[2] J. Shi, J. Wan, H. Yan, and H. Suo, "A Survey of Cyber-Physical Systems," in *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, pp. 1–6, IEEE, 2011.
[3] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini, "A Survey of Context Data Distribution for Mobile Ubiquitous Systems," *ACM Computing Surveys*, vol. 45, no. 1, pp. 1–49, 2013.
[4] S. Zug, M. Schulze, A. Dietrich, and J. Kaiser, "Programming abstractions and middleware for building control systems as networks of smart sensors and actuators," in *Proceedings of Emerging Technologies in Factory Automation (ETFA '10)*, (Bilbao, Spain), 9 2010.
[5] H. Piontek, *Self-description mechanisms for embedded components in cooperative systems*. PhD thesis, University of Ulm, 2007.
[6] M. van der Meulen, "On the use of smart sensors, common cause failure and the need for diversity," in *6th International Symposium of Programmable Electronic Systems in Safety Related Applications*, (Cologne, Germany), 5 2004.
[7] T. Brade, M. Schulze, S. Zug, and J. Kaiser, "Model-Driven Development of Embedded Systems," in *12th Brazilian Workshop on Real-Time and Embedded Systems (WTR)*, (Gramado, Brazil), Brazilian Computer Society, 5 2010.
[8] S. Ding, *Model-based fault diagnosis techniques: design schemes, algorithms, and tools*. Springer Verlag, 2008.
[9] G. Vachtsevanos, F. Lewis, M. Roemer, A. Hess, and B. Wu, *Intelligent fault diagnosis and prognosis for engineering systems*. Wiley, 2006.
[10] R. Isermann, "Model-based fault-detection and diagnosis–status and applications," *Annual Reviews in control*, vol. 29, no. 1, pp. 71–85, 2005.
[11] B. Hardekopf, K. Kwiat, and S. Upadhyaya, "Secure and fault-tolerant voting in distributed systems," in *Proceedings of IEEE Aerospace Conference*, vol. 3, 2001.
[12] J. E. White and J. L. Speyer, "Detection filter design: Spectral theory and algorithms," *IEEE Transactions on Automatic Control*, vol. 32, no. 7, pp. 593–603, 1987.
[13] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures.," *Analytical Chemistry*, vol. 36, pp. 1627–1639, July 1964.
[14] A. Zolghadri, "An algorithm for real-time failure detection in kalman filters," *IEEE Transactions on Automatic Control*, vol. 41, no. 10, pp. 1537–1539, 1996.
[15] R. Patton, "Fault-tolerant control systems: The 1997 situation," in *IFAC symposium on fault detection supervision and safety for technical processes*, vol. 3, pp. 1033–1054, 1997.
[16] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2003.
[17] Y. Sheng and S. Rovnyak, "Decision tree-based methodology for high impedance fault detection," *IEEE Transactions on Power Delivery*, vol. 19, no. 2, pp. 533–536, 2004.
[18] D. F. Specht, "Probabilistic neural networks," *neural networks*, vol. 3 Issue 1, pp. 109–118, 1990.
[19] A. Jabbari, R. Jedermann, and W. Lang, "Application of computational intelligence for sensor fault detection and isolation," *Proceedings Of World Academy Of Science, Engineering And Technology*, vol. 22, pp. 503–508, 2007.
[20] A. L. Christense, R. OGrady, M. Birattari, and M. Dorigo, "Fault detection in autonomous robots based on fault injection and learning," *Autonomous Robots*, vol. 24, pp. 49–67, 2008.
[21] P. Przystalka, "Model-based fault detection and isolation using locally recurrent neural networks," *Lecture Notes in Computer Science*, vol. 5097, pp. 123–134, 2008.
[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computations*, vol. 9, pp. 1735–1780, 1997.
[23] S. Zug, A. Dietrich, and J. Kaiser, *Fault Diagnosis in Robotic and Industrial Systems*, ch. Fault-Handling in Networked Sensor Systems. St. Franklin, AUS: Concept Press Ltd., 2012.
[24] J. C. da Silva, A. Saxena, E. Balaban, and K. Goebel, "A knowledge-based system approach for sensor fault modeling, detection and mitigation," *Expert Systems with Applications*, vol. 39 Issue 12, pp. 10977–10989, 2012.
[25] M. Meyer, *Signalverarbeitung: Analoge und digitale Signale, Systeme und Filter*. Springer Vieweg, 2011.
[26] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.