16th World Congress of the International Fuzzy Systems Association (IFSA)
9th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT)

Robustification of Self-Optimising Systems via Explicit Treatment of Uncertain Information

Jan H. Schoenke, Werner Brockmann

Department of Computer Science, University of Osnabrück
Email: {jschoenk, Werner.Brockmann}@uni-osnabrueck.de

Abstract

Uncertainty treatment in self-optimising systems touches two design-issues. Firstly, a valid estimation of uncertainties within the system is impossible beforehand as the uncertainties as well as the systems behaviour changes during run-time due to self-optimisation. Secondly, the design of a self-optimising system needs to mediate between the often conflicting goals of optimality and robustness. Here we present the concept for a lightweight algorithmic add-on for self-optimising function approximators that enables to reflect uncertainties related to the current state and to flexibly combine optimality and robustness in one design. Illustrating examples of TS-fuzzy systems highlight the properties of our approach.

Keywords: Self-Optimising Systems, Uncertain Information, Uncertainty Treatment, Regression

1. Introduction

One of the most important and challenging tasks in autonomous systems is to deal with uncertain information. This is especially true for self-optimising systems as they introduce dynamics at another level, which cannot be foreseen at design-time. The treatment of uncertainties consists of both a representation of the degree of uncertainty the system has to deal with and a strategy to make the system robust against this uncertainty, i.e. to reduce or even to eliminate its impact. Such a 'robustification' poses a severe design challenge.

At first sight, uncertainty arises from the measurements of the sensors and propagates throughout the modules of a multi-staged systems architecture along the data path. In general, uncertainty can arise with any signal and information that is processed within a system and is thus even related to the result of this processing. This results in a *spread of uncertainty* throughout the system. So each module may has to cope with uncertain inputs and thus produces an output that to some extent is uncertain as well.

Here it is important to note that many sources of uncertainty are not statistical in their nature and that a non-linear system behaviour leads to a varying impact of uncertainties and hence complicates their estimation or even reduction significantly.

The estimation of measurement uncertainty is addressed by the "Guide to the expression of Uncertainty in Measurement" (ISO/IEC Guide) [1] which provides a definition of uncertainty, lists possible sources of uncertainty and gives norms for its estimation for industrial automation systems as well as for scientific experiments. Basically [1] wants the designer to specify all relevant and recognised influences that can effect the uncertainty of a measurement and attribute the output of the measurement by a scalar value reflecting its degree of uncertainty. This also holds for the further processing of these (basic) measurements, e.g. by filters, so that each processing module has to reflect the uncertainty of its output by a scalar value attributed to each of its output values.

For classical, i.e. static systems this is a fundamental, yet complex part in the design of robust systems. But for self-optimising systems it is simply impossible to do at design-time as the behaviour of a self-optimising module changes during run-time. So the influence of an uncertain input on the output is changing as well and hence unknown at design-time. So for self-optimising systems an uncertainty estimation that complies with the ISO/IEC Guide [1] has to be done on-line. Each module itself thus has to reflect dynamically about the relevance of its inputs to determine the degree of uncertainty of its outputs due to uncertain inputs.

This fact does not release the designer from considering uncertainties in the design, but instead of pre-computing the uncertainty estimation and handling at design-time, the designer has to enhance each module to do this computation itself at runtime. The additional computational effort to reflect about the uncertainty in each module should be as small as possible in order to keep the overall complexity of the system and its computational demands low and not to violate real-time constraints of an application if there are any, e.g. for embedded systems.

The other design issue is the overall design goal a self-optimising systems has to achieve, namely to modify the module's behaviour in order to perform optimally on a particular task. Enabling a module to deal with uncertainties compels the designer to make the system robust against uncertainties as well. But robustness and optimality are conflicting design goals as a robust behaviour is not optimal in the general case. So there is a strong need for an integral approach that enables the designer to specify a robust system behaviour for the case of too strong uncertainties and still allows the system to be self-optimising, if the uncertainties are tolerable.

This paper presents such an integral approach in form of a lightweight algorithmic add-on for modules which act as (self-optimising) function approximators in regression tasks. The rest of the paper is therefore organised as follows. Section 2 reviews the state of the art of uncertainty representations as well as methods and architectures for uncertainty treatment. In section 3 the proposed approach is introduced and formally described. Properties of the approach are demonstrated by an illustrating example in section 4. The results are discussed in section 5 and section 6 concludes the paper.

2. Related Work

This section reviews different uncertainty representations and approaches for uncertainty treatment. The presented methods are discussed concerning their applicability in the design and operation of self-optimizing systems.

2.1. Uncertainty representations

The representations that are able to model uncertainties of any information in a gradual way can be grouped into four categories by their expressiveness.

The first category is the *scalar attribution* of measurement values as the most simple form of an uncertainty representation and also the way the ISO/IEC Guide [1] wants uncertainties to be reported. Examples for this kind of uncertainty representation are symmetric error bounds defining intervals of possible alternatives for the measurement value [2], variance and standard deviation related values from families of parametrised probability distribution functions defining how likely a measured value is [3], scaling parameters of standardised fuzzy numbers defining in a gradual way how possible a measured value is [4] and trust signals from the trust management framework defining in a more general way how trustworthy a measurement is [5].

As this category of representation has the least expressiveness along all uncertainty representations, it likewise has the least computational and design demands. E.g., the additional effort to make the designer fit prior knowledge about the uncertainties into the representation is low and the representation itself is easy to understand and to compute. Nevertheless, there are differences concerning computational and design demands between the examples listed above. E.g., interval arithmetic is easier to understand than probabilistic theory.

The second category of uncertainty representations uses (crisp) *sets* to represent the uncertainty about a measurement. A single measurement value is replaced by a set of possible alternatives that arise from the uncertainty that is modelled. This is classical possibility theory that expands the error bound related intervals from the first category to general sets [6]. As the expressiveness of this class of uncertainty representation increases compared to the first class, the computational and design demands increase as well. The computational demands of propagating sets limit the applicability of this class. The interval arithmetic [7] is a special case of the possibility theory that allows an efficient calculation of the output interval of a module under linear mappings. Determining the output interval of possible values under non-linear mappings is not trivial and increases the computational demands drastically. So for modules that only provide a linear mapping the interval arithmetic is a valuable tool but its applicability is limited for non-linear mappings. And on top of that, at least at a controller (or decision making) module there is the need to choose one value out of an interval to be the actual output of the controller as the actor can only apply one concrete value to act on the controlled process. The design demands increase here as well because the designer has to determine for every alternative to the actual measured value whether it is possible or not due to the measurement uncertainty.

The third category of uncertainty representations is *scalar weighting* which is an extension of the second category. Here again a set of possible alternatives of a measurement is considered and each alternative is weighted by a scalar value ranging from 0 to 1 representing how likely or possible in a gradual way any alternative is. Examples for this kind of uncertainty representation are the membership functions of fuzzy set theory [4] which give a scalar weight to each alternative value and can hence be interpreted in the sense of possibilistics. Probability density functions act similarly in the probability theory [3] which forces the sum or integral of all weights of all alternatives to equal one and interprets these weights as likelihood.

Looking at the design demands to specify the information needed to make use of the expressiveness of this class of uncertainty representations, the designer not only needs to determine whether a particular alternative is possible or not as for the second class, but he is also asked to give a weight to each alternative. Since the demand of prior knowledge to determine the weights that accurately represent the uncertainty is very high and the processing of arbitrary functions describing the weights of all alternatives under a non-linear mapping is very complex, this class of uncertainty representation is only applicable if the *characteristic functions* describing the weights of all alternatives are restricted to parametrised families of functions. This way the computational and design demands are limited. But these approaches then tend to fall into the first class of uncertainty representations as there are only very few parameters that define the characteristic weight-function and thus the complete uncertainty representation.

The fourth and last category of uncertainty representations contains all *higher order weighting* approaches. Higher order weighting here refers to the weights that are the heart of the third class. These weights are replaced by distributions of different weights in order to avoid the need to specify the weights precisely. Examples for this kind of uncertainty representation are Imprecise Probabilities like the Dempster Shafer Theory [8], that defines lower and upper bounds for the probability distribution of alternative measurements, and type-2-fuzzy sets that allow to express the ambiguity about the gradual possibility of the alternatives. While this class of uncertainty representations allows to express nearly any type and aspect of uncertainties, the computational and design demands are immense.

2.2. Uncertainty Treatment at Architectural Level

System architectures have been extended to cope with uncertainties in recent years. Two categories can be distinguished here depending on the way uncertainty treatment influences the system operation. On the one hand, a detailed assessment and treatment of the uncertainties takes place at the level of concrete signals and internal parameters, i.e. integral within signal processing algorithms, henceforth called *fine-grained*.

One of the most common approaches is to use the calculus of probability theory for an explicit fine-grained representation of uncertainty [9]. This modelling treats measurements as observations of random variables, and state transitions depending on actions are modelled as conditional probabilities. Consequently, assumptions about the distribution of measurements and the conditional probabilities of actions have to be made in advance and the results typically depend on the chosen frame of discernment. Besides the complexity of additional information a design engineer must give, the complexity of calculations increases significantly. And often only approximate solutions, e.g. by particle filters [9], are feasible.

On the other hand, uncertainties are treated at module level by steering the module's activity as a whole without affecting its internal operation, henceforth called *coarse-grained*. Here the functionality of the system is decomposed into several behaviours each designed for a specific subtask. These modules interact with each other by the means of some meta-signals. This is similar e.g. to the *activation-based behaviour control architecture* [10] which Albiez et al. introduced for autonomous robots. It works without an explicit representation of uncertainty but uses meta-signals called activity and target rating. Each behaviour generates a target rating, expressing how much the current state fits the behaviour's goal. In result modules are dynamically activated depending on the rating of a situation and it is possible to implicitly handle critical situations differently within this architecture without designing counteractions explicitly. The actions of the robot are hence based on the normal signal flow from sensors to actuators as well as on these meta-signals yielding an emergent behaviour. Thus, this activation mechanism allows a robustification to a certain degree by activating only a defined subset of behaviours, i.e. it deals implicitly with uncertainties at a coarse-grained level.

A fine-grained robotic approach is the notion of a *pain level* which has been introduced by Farrell in [11] to gradually model sensor failures at runtime. The fault status of each sensor is estimated gradually based on the history of its measurements resulting in an additional signal attribute, the pain level, given to an injury agent declaring that the sensor is working properly or broken somehow. In this architecture, each sensor with a pain level above some defined threshold is discarded from further processing. But it is possible for a sensor to recover and being reintegrated into processing, thus making the approach comparable to coarse-grained uncertainty treatment.

The ORCA-architecture [12] reflects this gradual nature of an uncertainty measure similar to pain levels by introducing *health signals* regarding the health status not only of the sensors but also of potentially any processing module. Here the normal signal flow is defined in basic control units (BCUs) providing the functionality of the system in a fault-free case. These are accompanied by organic control units (OCUs) which monitor and react to the health signals for supervising the BCUs and changing parameters within a BCU module as well as their interplay in case of anomalies, thus performing a coarse-grained handling of uncertainties. This influence is gradual and allows for a graceful degradation while maintaining the safety of the system.

In the ORCA-architecture the system is fully functional without the OCUs. In order to get a greater flexibility and to improve expressiveness and performance, the biological motivation of health signals is abandoned with the introduction of the Trust Management (TM) framework [13, 5]. Its key features are to represent the trustworthiness of signals or parameters by attributes, called *trust signals*, and to integrate them directly into the signal processing algorithm in a fine-grained manner. Hence, the uncertainty treatment becomes an integral part of the functional units, i.e. the BCUs. In the TM framework an information attributed with a trust signal $\vartheta = 1$ has to be processed normally, i.e. as if there was no TM or uncertainty, respectively, at all. Whereas for a trust signal $\vartheta = 0$ the corresponding information must not be used for processing anyhow. Due to a more general applicability of this underlying principle, Trust Management yields a generic uncertainty representation and treatment throughout the whole systems architecture. It thus has not to introduce additional (OCU) modules into existing architectures and therefore allows for a seamless migration from existing conventional and coarse-grained architectures to an integral fine-grained processing of uncertain information with a low additional complexity for uncertainty handling.

2.3. Uncertainty Treatment at Processing Module Level

The fine-grained treatment of uncertain information or even missing or faulty inputs in single modules within an architecture is an unavoidable and well established part of many real-world applications. A general overview of the most common methods within this field is given in [14, 15]. The field can be roughly divided into three areas.

First, with *imputation* a measured value of an uncertain input is replaced by a more certain one, thus reducing the uncertainty. The source to get the more certain value from depends on the imputation method. Several imputation methods are reviewed in [16]. Basic imputation methods use filter, regression or nearest neighbour techniques to estimate an imputation value. Advanced imputation methods restrict the imputation values to comply to a given distribution. The module consuming the inputs is unaffected by this kind of uncertainty treatment and thus the complexity overhead is low at system level. But the uncertainty about the inputs is not reflected at the output of such a module which is thus treated as completely reliable.

Multiple imputation methods generate a discrete set of imputation values resulting in a set of output values. Mean and variance of this set define the most likely output value and its quality. But the complexity overhead gets large due to multiple evaluations of the module.

The second category uses *ensemble* techniques [17] to overcome the need for imputation. A single module performing one task is replaced by an ensemble of modules all performing the same task, but each using a different subset of the complete set of inputs. To avoid propagating values of uncertain inputs, only the performing unit which does not use any of the uncertain inputs but uses all of the certain inputs is evaluated. For this kind of approach the number of modules increases exponentially with the number of uncertain inputs, and hence the engineering and computation efforts as well.

The third way to treat input uncertainty is to *model* the uncertainty directly. Therefore each input is extended by a measure of its degree of uncertainty according to one of the methods of chapter 2.1 and the module has to be able to handle these extended inputs in a fine-grained manner. The main advantage of modelling uncertainties directly is the low computational complexity compared to ensemble methods in the cases of multiple uncertain inputs. The main disadvantage is the need to correctly model and process the uncertainties to get reliable results. For several uncertainty representations special solutions have been developed. The interval representation of uncertainties is used in [18, 19], the fuzzy representation in [20] and (imprecise) probabilities in [21, 22]. These approaches are fine-grained extensions to a conventional architecture, which inherit the calculation and design issues from their respective uncertainty representation.

In general, there are many approaches to uncertainty treatment at architectural and module level, but not all of them are feasible for self-optimising system modules. An appealing feature of approaches at coarse-grained architectural level is that they allow the designer to easily blend between a self-optimising and a fixed robust behaviour. But this way the designer can only specify a behaviour of the system in the presence of uncertainties, rather than a strategy which may incorporate the self-optimised behaviour to treat and to reduce the uncertainties at hand. So it is important to give the designer the opportunity to specify how to make the self-optimised behaviour robust in addition to the specification of a special fall-back behaviour. This can only be achieved by a fine-grained uncertainty treatment at module level. And from the class of fine-grained uncertainty treatment the direct modelling is the only approach that does not hide the uncertainty, but shows it to the module. So the direct modelling of uncertainties enables a self-optimising module to reflect about its uncertainties at hand according to its own behaviour. And in addition to that, it allows the module to estimate the uncertainty of its output efficiently.

So our aim is to find a fine-grained approach that enables the designer to specify a strategy for the treatment of uncertain inputs by enabling each module to react on and reflect about and finally to reduce the uncertainties at hand, even when the modules behaviour changes at run-time due to self-optimisation.

3. A unifying approach for uncertainty treatment in self-optimising modules

Our approach for uncertainty treatment in self-optimising modules is presented in this section. Firstly, the general concept of our approach is described and formalised. Subsequently, a special case is derived from the general formal description in order to illustrate the design properties this approach provides and how it behaves formally.

3.1. General Concept

The duality of the design goals robustness and optimality is the major concern of the presented approach. Therefore a fine-grained algorithmic add-on to (existing) self-optimising modules is presented.

It is an integral solution that gives room to both of the design goals depending on the uncertainty at hand. This add-on models uncertainties by trust signals and enhances each module in two complementary ways that even allow for a system wide uncertainty treatment similar to coarse grained uncertainty treatment. Firstly, the module is enabled to blend dynamically between its self-optimising behaviour and a robust behaviour depending on the uncertainty of the inputs. Secondly, the module can reflect its own output uncertainty depending on the uncertainty of the input and its own functional behaviour.

As the behaviour of the self-optimising module changes during run-time, the uncertainty treatment of the proposed algorithmic add-on must be fully transparent to the basic self-optimising module and thus only concerns the evaluation of the module to form its output. This way the uncertainty treatment of the add-on is completely independent of the self-optimising property of the module at an algorithmic level.

If an input to a module is uncertain, it is ambiguous at which point in the input space the module has to be evaluated, i.e. what the (true) actual input is. Making a module robust against uncertain inputs means to make the output of the module unaffected by these uncertainties. Our approach here to ensure such a robust module behaviour is to limit the influence of an uncertain input on the output depending dynamically on the actual degree of uncertainty, thus following the basic principles of the TM framework. The aim is to allow the module to perform its regular self-optimising behaviour as long as the uncertainties at the input are tolerable and to gradually reduce the influence of uncertain inputs by an Uncertainty Treatment Strategy (UTS). This UTS allows the designer to specify how the module reacts on too large uncertainties in order to still ensure a robust system behaviour.

To enable the module to reflect about the actual degree of uncertainty of its output, we follow the spirit of error propagation. But instead of using all partial derivatives at the actual evaluation point, we rather use the variance of the output along an uncertain input only, since an uncertain input does not allow to exactly determine the point in the respective dimension of the input space where the partial derivatives should be calculated. This integral scheme provides properties similar to the differential one, but as the integral takes a global look at the behaviour of the module it is more robust concerning the uncertainty of an input. If the variance along an uncertain input vanishes, the input is irrelevant for the calculation of the output and its uncertainty can be ignored as well, i.e. the uncertainty vanishes. If the variance along an uncertain input is high, the input has a strong influence on the output and thus the uncertainty of the output has to be increased according to the variance of the output along the uncertain input dimension. The same is true for multiple uncertain inputs. For multiple outputs, this approach has to be calculated for each output of the module separately.

This general concept is applicable to different kinds of modules that perform tasks like process modelling, classification, control or other whether they are self-optimising (at run-time) or not. As the great variety of these tasks cannot be captured in one paper, we focus here on the class of continuous mappings between real vector spaces as they are basic to nearly any real-world application, e.g. modelling and control.

3.2. Formal Description of the Approach

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be the continuous mapping of a self-optimising module that maps an input vector $\vec{x} \in \mathbb{R}^n$ to an output vector $\vec{y} = f(\vec{x}) \in \mathbb{R}^m$. The mapping f may be non-linear and, of course, time variant due to the self-optimisation property of the module. But for any single evaluation of the module, it is sufficient to treat its processing as being static. Let further be $\vec{\vartheta}_x \in [0,1]^n$ the uncertainty representation of the input $\vec{x}$, that leads to an extended input format $(\vec{x}, \vec{\vartheta}_x)$, and $\vec{\vartheta}_y \in [0,1]^m$ the uncertainty representation of the output $\vec{y}$ that leads to an extended output format $(\vec{y}, \vec{\vartheta}_y)$. The proposed add-on extends the mapping f to a mapping $f^* : \mathbb{R}^n \times [0,1]^n \to \mathbb{R}^m$ and measures the uncertainty $\vec{\vartheta}_y$ at the output $\vec{y}$ of f^* by the variance σ^2 (defined in equation (2)).

Formalising our general concept for the fine-grained robustification then results in a system of partial differential equations presented in (1) which f^* has to comply with.

$$\frac{\partial f_j^*(x, \vartheta_x)}{\partial x_i} = V_i(\vartheta_x) \frac{\partial f_j(x)}{\partial x_i} \forall i, j \qquad (1)$$

The basic idea is to limit the influence of an uncertain input x_i by bounding its respective partial derivative in f^* at the current working point. The vector valued *blending function* $V : [0,1]^n \to \mathbb{R}^n$ uniquely defines how the behaviour of the module blends from the regular ($\vartheta_x = 1$) to the robust case ($\vartheta_x = 0$). The boundary conditions related to a solution of (1) thus uniquely define the actual behaviour of f^* in the certain as well as in the robust case.

Since equation (1) is not constructive nor allows for a direct deduction of a solution for f^*, the section 3.3 will show a practical choice for V and derive a solution for the large class of Linear In the Parameters (LIP) approximators with separating base functions. LIP-approximators are also widely used for non-linear mappings in self-optimising systems as they allow for a convex optimisation problem formulation. The class contains polynomials as well as many types of fuzzy systems and radial basis function networks which can for example serve as a controller, as a process model or as filters and the like.

To quantify the trust level ϑ_y of an output y, the dependency between an uncertain input x_i and the output y, i.e. the variance σ_i^2 along the uncertain input dimension ($[\underline{x_i}, \overline{x_i}]$) is mapped to the trust level ϑ_y. If the variance σ_i^2 along the uncertain input dimension x_i is zero, the output is independent of the uncertain input x_i. In this case the trust level ϑ_y of the output should hence be independent of ϑ_{x_i} and should not be decreased by the uncertainty of the input x_i, i.e. the respective module can recover from uncertainty. With an increasing variance σ_i^2 along an uncertain input dimension x_i, the output dependency increases too and the trust level ϑ_y of the output must decrease accordingly. This approach to one uncertain input x_i can be applied to multiple uncertain inputs x by weighting the different variances along different inputs x_i with their trust levels ϑ_{x_i} or e.g. by a t-norm operation.

The calculation of the variance of f has to be done according to the trust levels ϑ_x. This requires a ϑ_x-weighted variance σ^2 of the function f which is defined in equation (2).

$$\sigma^2 = Var(f(x), \vartheta_x) = \tag{2}$$

$$\sum_{A \in P(\{1,...,n\})} \prod_{a \in A} \vartheta_{x_a} \prod_{b \in \overline{A}} (1 - \vartheta_{x_b}) Var(f(u)|_{u_A = x_A})$$

The term $Var(f(u)|_{u_A=x_A})$ in the variance calculation in (2) refers to calculating the variance of f along the respective input dimensions that are in $\overline{A}$ which are assumed to be uncertain and setting the values of the input dimensions that are in A to the corresponding values in x, which are assumed to be certain.

In general, the ϑ_x-weighted variance calculation in equation (2) has a computational complexity of $O(2^n)$ as there is a sum over all elements of the power set $P(\{1, ..., n\})$ of the indices $\{1, ..., n\}$. This computational effort equals the demands of an ensemble approach and is thus rarely applicable. But there is a computationally cheap variant of equation (2) for separating functions, e.g. the aforementioned LIP-approximators that will be shown in the next section.

3.3. Linear Blending for LIP-Approximators

Restricting equation (1) to the special case of a linear trust-related blending between a regular and a robust case leads to the system of partial differential equations defined in equation (3).

$$\frac{\partial f^*(x, \vartheta_x)}{\partial x_i} = \vartheta_{x_i} \frac{\partial f(x)}{\partial x_i} \forall i \tag{3}$$

In this linear approach the function V is simply the identity and thus the influence of each input x_i onto the output y is linearly limited by its corresponding trust signal ϑ_{x_i}. The linear blending approach is now worked out for the class of LIP-approximators which can be evaluated via an inner product on vector spaces between a parameter vector $\alpha \in \mathbb{R}^k$ and vector of base functions $\phi : \mathbb{R}^n \to \mathbb{R}^k$ as shown in equation (4).

$$y = \alpha^T \phi(x) = \sum_{i=1}^{k} \alpha_i \phi_i(x) \tag{4}$$

While the mapping of the base functions ϕ may be non-linear, the effect of the parameters α onto the output y is strictly linear. Hence, there is an elegant solution to equation (3) as long as the base functions ϕ_i are separable, i.e. $\phi_i(x) = \prod_{j=1}^{n} \phi_{i,j}(x_j)$. Solving (3) by assuming separability and (4) leads to equation (5) with integration constants $\tilde{c}_{i,j}$.

$$f^*(x, \vartheta_x) = \sum_{i=1}^{k} \alpha_i \prod_{j=1}^{n} \left(\vartheta_{x_j} \phi_{i,j}(x_j) + \tilde{c}_{i,j}\right) \tag{5}$$

While each $\tilde{c}_{i,j}$ needs to be constant in x_j, they are allowed to be varied in ϑ_{x_j}. Choosing the integration constants according to $\tilde{c}_{i,j} = (1 - \vartheta_{x_j}) c_{i,j}$ provides a new set of design parameters $c_{i,j}$ that uniquely define the behaviour of the function f^* in the robust case, but do not affect the evaluation of f^* in the regular case, yielding $f^*(x, 1) = f(x)$. And as the blending to the design parameters $c_{i,j}$ only affects the $\phi_{i,j}$, the UTS does not affect the self-optimisation process and can still incorporate the knowledge of the (self-optimising) LIP-approximator represented by the parameter vector α but without an impact of x_j. The calculation of the output trust level ϑ_y of equation (2) can then be simplified for LIP-approximators as shown in equation (6).

$$\sigma^2 = \mathrm{Var}(f(x), \vartheta_x) = \tag{6}$$
$$\sum_{i,j=1}^{k} \alpha_i \alpha_j \left(\prod_{a=1}^{n} (\vartheta_a^x C_{i,j,a}(x) + (1 - \vartheta_{x_a}) V_{i,j,a}) - \prod_{a=1}^{n} (\vartheta_a^x C_{i,j,a}(x) + (1 - \vartheta_a^x) Q_{i,j,a}) \right)$$

with

$$C_{i,j,a}(x) = \phi_{i,a}(x_a) \phi_{j,a}(x_a) \tag{7}$$
$$V_{i,j,a} = \frac{\int_{\underline{x}_a}^{\overline{x}_a} \phi_{i,a}(x) \phi_{j,a}(x) dx}{\overline{x}_a - \underline{x}_a} \tag{8}$$
$$Q_{i,j,a} = \frac{\int_{\underline{x}_a}^{\overline{x}_a} \phi_{i,a}(x) dx}{\overline{x}_a - \underline{x}_a} \cdot \frac{\int_{\underline{x}_a}^{\overline{x}_a} \phi_{j,a}(x) dx}{\overline{x}_a - \underline{x}_a} \tag{9}$$

The complexity of the variance calculation (6) is in $\mathcal{O}(n^2)$, which is a comparatively small effort for an exact solution. And since all $\phi_{i,j}$ are fixed for a particular LIP-approximator, the corresponding integrals are constants for that approximator as well and can hence be pre-computed at design time. Thus no integration has to be performed on-line. A simple way to finally map the variance

σ^2 to an output trust level in $[0,1]$ is for example $\vartheta_y = \max(0, 1 - c \cdot \sqrt{\sigma^2})$, where $c > 0$ defines the sensitivity of the output trust level ϑ_y to the variance measure σ^2.

3.4. Design Pattern: Mean Value

The design parameters $c_{i,j}$ allow the designer to specify the UTS of the module in the robust case, i.e. for $\vartheta_{x_i} = 0$. In order to keep the overall design effort low, these parameters can be chosen according to a global design pattern. A simple, yet powerful pattern is the *mean value* strategy and it is illustrated exemplarily here and in the following example.

The basic idea of this pattern is to guide the output of the module to the mean value along the uncertain input dimensions for the robust case. Equation (10) then uniquely defines all $c_{i,j}$ to achieve this guidance. Thus, there is no extra effort for the designer to manually specify the systems behaviour.

$$c_{i,j} = \frac{\int_{\underline{x_j}}^{\overline{x_j}} \phi_{i,j}(u)\,\mathrm{d}u}{\overline{x_j} - \underline{x_j}} \qquad (10)$$

Applying this design pattern to the linear blending strategy enables a self-optimising module to smoothly blend between its current self-optimising behaviour and the mean of its output along uncertain input dimensions for the robust case. And as the presented algorithmic add-on only affects the evaluation, the mean value that is used to gain a dynamically robust behaviour independent of uncertain input always relies on the actual knowledge of the module gathered by self-optimisation in its parameter vector α.

4. Illustrating Examples

The examples presented in the following accompany the formal description of the linear blending strategy from above in order to give first insights into the basic properties of our approach at signal level. Since the proposed add-on consists of both an uncertainty treatment and an output uncertainty estimation, two different examples are presented here to highlight the properties of these two parts. The first example displays the effect of the trust levels ϑ_x on the output, i.e. the evaluation of an enhanced module, and the corresponding output uncertainty estimation. The second example illustrates the impact of the robustification on the operation of a module in an artificial test environment.

4.1. Output Uncertainty Estimation

This example shows the shaping of the enhanced output f^* and the corresponding output trust level ϑ_y of a module with two inputs for varying input trust levels ϑ_x. The module used here is a zero-order Takagi-Sugeno-Fuzzy-System (TSFS) with sum-prod inference and triangular membership functions. As this TSFS is LIP with separable base functions, we can apply the linear blending strategy with the mean value design pattern and analyse its behaviour.

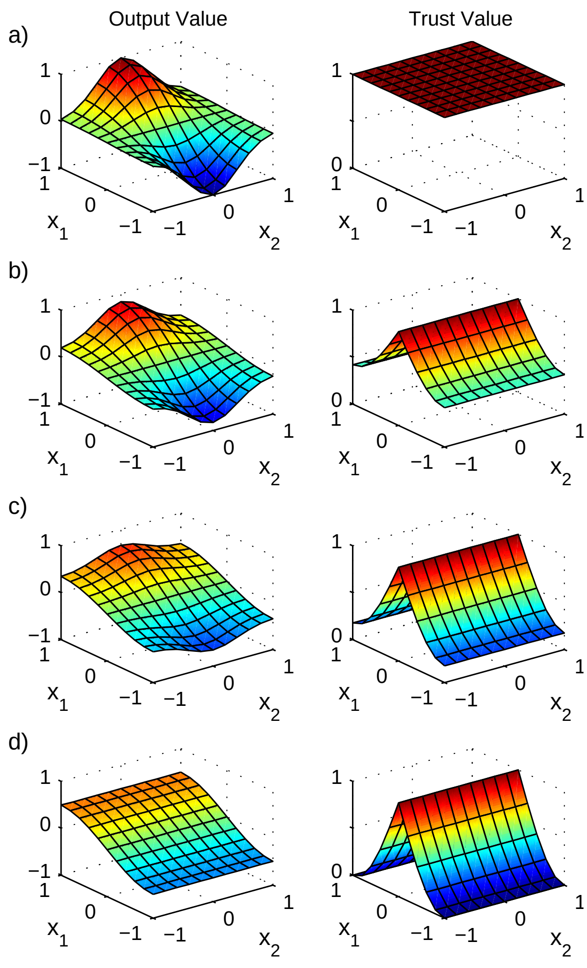

Figure 1: Functional behaviour (left) of an extended module with two inputs (x_1, x_2) for different values of the trust signal $\vartheta x_2 = \{1, 2/3, 1/3, 0\}$ (for a), b), c) and d)) of the second input x_2 and the corresponding output trust ϑ_y (right).

The TSFS consists of a 11×11 grid of equally spaced membership functions over the square $[-1,1]^2$ and its conclusions are set according to the nonlinear function

$$f(x_1, x_2) = \sin(\pi/2 x_1) \exp(-\pi x_2^2). \qquad (11)$$

For clarity reasons, only the trustworthiness of input x_2 is varied, i.e. the trust-signal ϑ_{x_2}, for otherwise fixed parameters.

4.1.1. Experimental Results

Fig. 1 shows the shape of the enhanced output f^* (left) and its respective output trust level ϑ_y (right) for different degrees of uncertainty of input x_2, i.e. for different values of the input trust level ϑ_{x_2}. From top to bottom the trust level ϑ_{x_2} drops from one to zero in steps of thirds. The sequence in the left column shows that the output f^* gets flattened along the input x_2 for a decreasing trust level ϑ_{x_2}. Thus the output f^* depends the less on the input x_2 the less the trust level ϑ_{x_2} gets and becomes completely independent of the input x_2 in the extreme case of a vanishing trust level ϑ_{x_2}. So even for garbage on this input, the output is well defined by changing the functional behaviour, see Fig. 1 d) (left).

The corresponding sequence of the output trust levels ϑ_y in the right column shows a remarkable behaviour depending on the input trust level ϑ_{x_2}. The output trust level ϑ_y only shapes along the input dimension x_1 and is flat, i.e. constant, along the input dimension x_2. Thus, it is sufficient to focus on the behaviour of the output trust level ϑ_y along the input x_1. In normal operation, i.e. $\vartheta_{x_2} = 1$, the output is always certain (Fig. 1 top). At an input value of $x_1 = 0$ the output trust level ϑ_y equals one independently of the input trust level ϑ_{x_2}. For any other input value of x_1 the output trust level ϑ_y gets the lower the lower the input trust level ϑ_{x_2} becomes. At the boundary, i.e. for input values $x_1 = -1$ or $x_1 = 1$, the output trust level even drops to a value of $\vartheta_y = 0$ for a vanishing input trust level ϑ_{x_2}.

This behaviour of the output trust level ϑ_y directly reflects the behaviour of the enhanced output f^* as it measures the impact of the flattening in f^* due to uncertain inputs. The more the output of the underlying function f is structured along an uncertain input dimension, the more uncertain is the output in such a case and the greater is the impact of the flattening along this input dimension. On the other hand, if the flattening in some case does not change the shape of the output at all, there is no reason to decrease the output trust level ϑ_y as the output is independent of the uncertain input.

4.2. Robustification by Uncertainty Treatment

This second example illustrates the robustification of a processing module due to explicit uncertainty treatment in an artificial test environment by comparing the performance of an enhanced module and a non-enhanced module. As a source of input uncertainty two different error models are evaluated separately, a local one and a global one.

The Local Error Model (LEM) refers to effects like noise. It forms the disturbed input x_{LEM} by adding bounded white noise to the true input point x, i.e.

$$x_{LEM} \sim x + U(-\eta, \eta). \tag{12}$$

The Global Error Model (GEM) refers to effects like outliers. The disturbed input x_{GEM} is formed by randomly replacing the true input point x by a point draw according to a equally distributed random variable over the domain of x, i.e.

$$x_{GEM} \sim (1-r)\cdot x + r \cdot U(-1,1), \tag{13}$$

with $r \sim B(1,\rho)$. In order to be able to gradually steer the impact of these error models the bounds $\eta \in [0,2]$ of the additive noise and the fraction of replacements $\rho \in [0,1]$ are varied for the local and the global error model, respectively.

For both error models the experiments are designed to also cover the extreme cases of maximal uncertainty about the input in order to test the limits of the proposed approach. Although these extreme cases are of little practical relevance, they essentially show the inherent properties of the proposed approach. The robustification against these uncertainties is founded on models that use only general prior knowlegde about the design parameters η and ρ to determine the trust levels for the local error model $\vartheta_{x,LEM} = 1 - 0.25\eta^2$ and the global error model $\vartheta_{x,GEM} = 1 - \rho$.

The performance is measured by the Root Mean Squared Error (RMSE) between the evaluations of the undisturbed input and disturbed input, see equation (14).

$$\text{RMSE} = \sqrt{\sum_{x\in X} \left(f(x) - f(x_{LEM,GEM})\right)^2} \tag{14}$$

The set of test inputs X for the RMSE calculation contains 1000 samples that are drawn according to a uniform distribution over the domain of x. Due to the stochastic nature of the underlying data for the RMSE calculation, these calculations are repeated 1000 times and are reported via the minimal, mean and maximal value of these 1000 repetitions.

As the robustification via the linear blending strategy treats each input dimension separately, it is sufficient to look at a 1-dimensional example to highlight the properties of the proposed approach as the effect of the enhancement works equally and independently for every input dimension. The insights of the 1-dimensional case can directly be adopted to the multi-dimensional case of arbitrary combinations of uncertain inputs. Thus, the tested module is realized using a TSFS with $N = 11$ equally space membership functions on the domain of x, i.e. $[-1, 1]$ and its conclusions are set up according to the nonlinear test function

$$y(x) = \sin(0.5\pi x - 0.25\pi). \tag{15}$$

4.2.1. Experimental Results

The results of this experiment are shown in Fig. 2 for different impact parameters η and ρ, respectively. For both error models the RMSE gets the higher the higher the impact of the error model gets despite the enhancement, but the RMSE with enhancement grows more slowly. At small impact parameters ($\eta < 0.5$, $\rho < 0.25$) no or only little performance gain is achieved due to enhancement. A

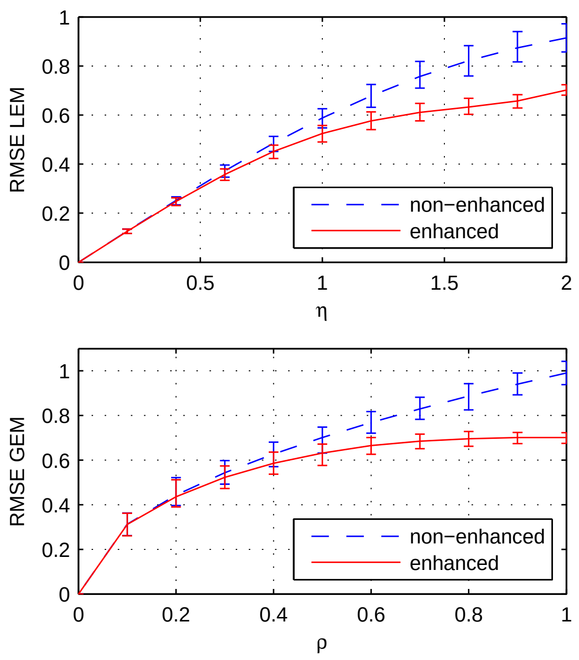

Figure 2: Performance evaluation results of a module for an uncertain input that is disturbed by a local error model (top) and global error model (bottom). For comparison the RMSE is plotted with (solid line) and without (dashed line) uncertainty treatment. The error bars indicate the spread of the performance by showing the minimal and maximal performance values.

remarkable difference in the performance shows up only at high impact parameters ($\eta > 1$, $\rho > 0.5$). Here the RMSE without enhancement grows nearly linearly with an increasing impact factor. With enhancement the performance loss is much lower and the RMSE of the global error model even clearly saturates to a worst case performance. The RMSE of this worst case performance is identical for both error models as both trust levels drop to zero in the extreme case of maximum uncertainty.

As the trust signals modeling the uncertainty here only apply coarse general prior knowledge about the strength of the uncertainty, the performance may be increased by further fine-tuning. Nevertheless, the worst case performance of the enhanced module is not worse than the performance for $\vartheta_x = 0$, whereas it may get totally random in the non-enhanced case in the sense of *garbage in - garbage out*.

5. Discussion

These first illustrative investigations on dealing with the input uncertainty show that the proposed robustification technique is able to increase the average performance of the module by using qualitative general prior knowledge only. So the module itself becomes more robust in its behaviour by modelling the uncertainty by trust signals depending on the data at hand, but also depending on the functional behaviour at the current working point as well. The latter point is especially important for self-optimizing systems in that it leads to the best possible performance at any time.

The linear blending strategy investigated here allows for a very efficient fine-grained implementation concerning the robustification for LIP-approximators. Along with the output uncertainty estimation it also produces only a moderate computational effort for separable LIP-approximators. Both is important for self-optimizing systems as the functional behaviour is dynamically changing at run-time. For other classes of approximators, especially if they are not separable, the computational demands may be higher.

The great merit of the output uncertainty estimation is the involvement of the actual behaviour of each model. Thus, every model can always reliably reflect whether an uncertain input is relevant or irrelevant for the calculation of the output and whether there is a spread of uncertainty or a reduction of uncertainty. This is especially important for self-optimising modules that change their behaviour over time, as the relevance of different inputs for the output changes dynamically. So the output uncertainty estimation really enables a reliable and system wide dynamic uncertainty treatment when uncertainty measures are propagated. In this way, the system may even recover dynamically from a low trust level in an early processing stage to a higher trust level in later processing stages as the attributed signal gets (dynamically) less relevant. This allows the subsequent modules to go for their optimal behaviour. Nevertheless, a robust system behaviour is still always ensured by the UTS that is smoothly blended to by decreasing trust levels.

Another key feature of the presented add-on is its statelessness, i.e. the absence of internal memory. Thus, only the current input values and trust signals are processed to form the output of the module and to calculate the corresponding uncertainty estimate of the output. This is very powerful since any signal can instantaneously be discarded from processing or recovered to in a gradual way without much computational and design efforts.

6. Conclusion

This paper focused on an integral, lightweight solution for the conflicting design goals of a robust treatment of uncertain inputs versus optimal and dynamic behaviour of self-optimizing systems. It presented a general framework for algorithmic add-ons to function approximators to make them robust against uncertain inputs by limiting the influence of uncertainties onto the output. It also allows to dynamically estimate the uncertainty of the outcome. By a proper selection of the blending function, different strategies to deal with and to reduce the impact of uncertainties can be established. In this paper the design pattern of linear blending to the mean value was introduced and its properties were investigated exemplarily for a TS-fuzzy system as a widely used representant for the class of LIP-approximators. The investigations showed that trust signals as a scalar attribute are sufficient to represent and tackle uncertainties. Furthermore, the robustification by our proposed strategy is able to ensure a robust module performance for different sources and degrees of uncertainties by applying only very general prior knowledge about the uncertainties at design-time. Even for a totally uncertain input, a clearly determined behaviour is achieved which adapts optimally to the current working point at run-time.

For LIP-approximators, the presented approach keeps the computational complexity in $O(n^2)$ both for incorporating input uncertainties as well as for the output uncertainty estimation. The overall additional computational demand for this add-on is thus comparatively low. It is also independent of the actual strategy the designer chooses for the uncertainty treatment strategy. This strategy is established solely by setting the parameters $c_{i,j}$ according to a global design pattern and keeps also the specific design effort low. Future work hence will address further design patterns for other uncertainty treatment strategies.

References

[1] Iso/iec guide 98-3:2008, uncertainty of measurement – part 3: Guide to the expression of uncertainty in measurement (gum:1995).

[2] Michael Grabe. *Generalized Gaussian Error Calculus.* Springer, 2010.

[3] Rick Durrett. *Probability: theory and examples.* Cambridge university press, 2010.

[4] George Klir and Bo Yuan. *Fuzzy sets and fuzzy logic*, volume 4. Prentice Hall New Jersey, 1995.

[5] W. Brockmann, A. Buschermöhle, J. Hülsmann, and N. Rosemann. Trust management — handling uncertainties in embedded systems. In *Ch. Müller-Schloer, H. Schmeck, T. Ungerer (eds.): Organic Computing - A Paradigm Shift for Complex Systems*, volume 1, pages 589–591. Springer, 2011.

[6] Kenneth Kunen. *Set theory.* College Publ., 2011.

[7] Ramon E Moore and RE Moore. *Methods and applications of interval analysis*, volume 2. SIAM, 1979.

[8] A. Dempster and G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, NJ, 1976.

[9] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, September 2005.

[10] J. Albiez, T. Luksch, K. Berns, and R. Dillmann. An activation-based behavior control architecture for walking machines. *The Int. Journal of Robotics Research*, 22(3-4):203–211, 2003.

[11] C. Ferrell. Failure recognition and fault tolerance of an autonomous robot. *Adaptive Behavior*, 2(4):375–398, 1994.

[12] W. Brockmann, E. Maehle, K. Grosspietsch, N. Rosemann, and B. Jakimovski. *ORCA: An Organic Robot Control Architecture*, volume 1, pages 385–398. Springer, 2011.

[13] W. Brockmann, A. Buschermöhle, and J. Hülsmann. A generic concept to increase the robustness of embedded systems by trust management. In *Proc. IEEE Int. Conf. on Systems Man and Cybernetics (SMC)*, pages 2037–2044. IEEE Press, 2010.

[14] P. Liu, E. El-Darzi, L. Lei, C. Vasilakis, P. Chountas, and W. Huang. An analysis of missing data treatment methods and their application to health care dataset. In *Advanced Data Mining and Applications*, pages 583–590. Springer, 2005.

[15] P. J García-Laencina, J.-L. Sancho-Gómez, and A.R. Figueiras-Vidal. Pattern classification with missing data: A review. *Neural Computing and Applications*, 19(2):263–282, 2010.

[16] X. Yan, H. Xie, and T. Wang. Using mvpca: An uncertain sensor data estimation method. *Journal of Computational Information Systems*, 8(10):4185–4192, 2012.

[17] H. Mohammed, N. Stepenosky, and R. Polikar. An ensemble technique to handle missing data from sensors. In *IEEE Sens Appl Symp, Houston, Texas, USA*, pages 101–105, 2006.

[18] Zian Wang and Fernando L Alvarado. Interval arithmetic in power flow analysis. *IEEE Trans. Power Systems*, 7(3):1341–1349, 1992.

[19] S. K. Michael Wong, LS Wang, and YY Yao. Interval structure: A framework for representing uncertain information. In *Proc. 8th Int. Conf. on Uncertainty in Artificial Intelligence*, pages 336–343. Morgan Kaufmann Publishers Inc., 1992.

[20] Ivan Hlaváček. Uncertain input data problems and the worst scenario method. *Applications of Mathematics*, 52(3):187–196, 2007.

[21] Y.-M. Wang, J.-B. Yang, D.-L. Xu, and K.-S. Chin. On the combination and normalization of interval-valued belief structures. *Information Sciences*, 177(5):1230–1247, 2007.

[22] G. Nassreddine, F. Abdallah, and T. Denoux. State estimation using interval analysis and belief-function theory: Application to dynamic vehicle localization. *IEEE Trans. Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(5):1205–1218, 2010.