Autonomous supervision and optimization of product quality in a multi-stage manufacturing process based on self-adaptive prediction models.

LUGHOFER, E., ZAVOIANU, A.-C., POLLAK, R., PRATAMA, M., MEYER-HEYE, P., ZÖRRER, H., EITZINGER, C. and RADAUER, T.

2019

Autonomous Supervision and Optimization of Product Quality in a Multi-Stage Manufacturing Process based on Self-Adaptive Prediction Models

Edwin Lughofer[a], Alexandru-Ciprian Zavoianu[a], Robert Pollak[a], Mahardhika Pratama[b], Pauline Meyer-Heye[c], Helmut Zörrer[c], Christian Eitzinger[c], Thomas Radauer[d]

[a]*Department of Knowledge-Based Mathematical Systems, Johannes Kepler University Linz, Austria*
[b]*School of Computer Science and Engineering, Nanyang Technological University, Singapore*
[c]*Profactor GmbH, Steyr-Gleink, Austria*
[d]*Stratec Consumables, Anif, Austria*

Abstract

In modern manufacturing facilities, there are basically two essential phases for assuring high production quality with low (or even zero) defects and waste in order to save costs for companies. The first phase concerns the early recognition of potentially arising problems in product quality, the second phase concerns proper reactions upon the recognition of such problems. In this paper, we address a holistic approach for handling both issues consecutively within a predictive maintenance framework at an on-line production system. Thereby, we address *multi-stage functionality* based on i) data-driven forecast models for (measure-able) product quality criteria (QCs) at a latter stage, which are established and executed through process values (and their time series trends) recorded at an early stage of production (describing its progress), and ii) process optimization cycles whose outputs are suggestions for proper reactions at an earlier stage in the case of forecasted downtrends or exceeds of allowed boundaries in product quality. The data-driven forecast models are established through a high-dimensional batch time-series modeling problem. In this, we employ a non-linear version of PLSR (partial least squares regression) by coupling PLS with generalized Takagi-Sugeno fuzzy systems (termed as *PLS-Fuzzy*). The models are able to self-adapt over time based on recursive parameters adaptation and rule evolution functionalities. Two concepts for increased flexibility during model updates are proposed, i.) a dynamic outweighing strategy of older samples with an adaptive update of the forgetting factor (steering forgetting intensity) and ii.) an incremental update of the latent variable space spanned by the directions (loading vectors) achieved through PLS; the whole model update approach is termed as *SAFM-IF (Self-Adaptive Forecast Models with Increased Flexibility)*. Process optimization is achieved through multi-objective optimization using evolutionary techniques, where the (trained and updated) forecast models serve as surrogate models to guide the optimization process to Pareto fronts (containing solution candidates) with high quality. A new influence analysis between process values and QCs is suggested based on the PLS-fuzzy forecast models in order to reduce the dimensionality of the optimization space

Email addresses: `edwin.lughofer@jku.at (Corresponding Author)` (Edwin Lughofer), (Alexandru-Ciprian Zavoianu), (Robert Pollak), (Mahardhika Pratama), (Pauline Meyer-Heye), (Helmut Zörrer), (Christian Eitzinger), (Thomas Radauer)

and thus to guarantee high(er) quality of solutions within a reasonable amount of time (→ better usage in on-line mode). The methodologies have been comprehensively evaluated on real on-line process data from a (micro-fluidic) chip production system, where the early stage comprises the injection molding process and the latter stage the bonding process. The results show remarkable performance in terms of low prediction errors of the PLS-Fuzzy forecast models (showing mostly lower errors than achieved by other model architectures) as well as in terms of Pareto fronts with individuals (solutions) whose fitness was close to the optimal values of three most important target QCs (being used for supervision): flatness, void events and RMSEs of the chips. Suggestions could thus be provided to experts/operators how to best change process values and associated machining parameters at the injection molding process in order to achieve significantly higher product quality for the final chips at the end of the bonding process.

Keywords: multi-stage production system, predictive maintenance, quality criteria (QC), PLS-fuzzy regression, self-adaptive and evolving forecast models, influence analysis, process optimization, suggestions for improved machining parameters

1. Introduction

1.1. Motivation and State-of-the-Art

Predictive maintenance relies on real-time monitoring and diagnosis of system components, process and production chains [1]. The primary strategy is to take action when items or parts show certain behaviors that usually result in machine failure, reduced performance or a downtrend in product quality. A central aspect of predictive maintenance thereby is to recognize untypical system behavior *at an early stage*, ideally as early as possible [2], [3], in order to have enough time for an appropriate reaction and to avoid bad quality, component or machine fallouts or even risks for operators in subsequent processing stages.

In case of a multi-stage production process, the problem of early maintenance becomes even more apparent and demanding whenever problems at an early stage have a significant impact on the quality of production parts at a latter stage [4]. The time span between an arising problem and its effect is thus much longer, which in turn increases additional waste and unwanted costs during production phases inbetween the early and latter stages. Furthermore, it is typically very difficult to realize for operators that quality downtrends occurring at a later stage stem from problems in former production stages, especially when the later stage is conducted significantly delayed to the early stage (e.g., in case of chip production, the delay can be up to several days or weeks, see Section 5). This means that manual supervision has a low likelihood to be successful in multi-stage maintenance cases. Hence, (semi-)automatic approaches are urgently demanded, which usually comprise two stages for achieving rich and smooth maintenance in a multi-stage process:

1. Early recognition of arising problems in form of downtrends in product quality or in form of cases where measures for product quality are expected to fall out of allowed tolerance limits in a latter/final stage of production.

2. Suggestions for proper reactions in case of predicted product quality problems.

For the first task, usually techniques from the fields of forecasting [5] and prognostics [6], [7] are applied, which can either rely on process parameter settings (static case) or process values recorded over time (dynamic case). Supervision of process parameters were proposed in [8] (there for a phone camera lens injection molding machine) and in [9] (there for plastic injection moulding). Both approaches rely on linear models and static mappings which do not integrate the time component (for respecting varying delays in input variables) and which cannot be (self-)adapted over time to address significant system dynamics properly. In [10], conventional partial least squares regression (PLSR) models are used for transforming the high-dimensional process value space in order to reliably predict product quality in steel industry. The same type of PLSR models are used in [2] for automated driving a process to a specific predefined quality. Again, these models can reflect only linear behaviors sufficiently well and cannot autonomously adapt to process changes and system dynamics.

A proper reaction to predicted arising quality downtrends can be in form of manual expert-based intervention in the process [11] or by automatized process parameter optimization cycles [12], [13], [14], [15]. The latter basically demonstrate strategies how to optimize machining (process) parameters, which steer the production process in a certain direction, for achieving better product quality. They are based on relational *static* mappings between process parameters and resulting quality measures, established through design of experiments (DoE) and data-driven (empirical) model construction either with [12], [13] or without [15] analytical background about the process. Due to their static nature, they can only predict how certain (machining) parameters will most possibly influence the final product quality and perform the optimization based on this static predictive assumption. However, they are not able to take into account any undesired, unforeseen variations during the actual on-line production process, which may arise, e.g., due to changing system dynamics (e.g., new charge types,...) or due to specific non-stationary environmental influences, which may have an effect on the final product quality as well. In this sense, they are not able to predict and balance out undesired dynamic process behaviors which may lead to worse product quality outcomes than originally expected due to particular settings of the process (control) parameters.

Finally, all of the aforementioned related works regarding both, prognostics and forecast models as well as process optimization concepts, have one in common: they are only focussed on a single-stage production process, but do not respect any multi-stage aspect, i.e. they do not predict from process parameters/values recordings to product quality criteria over more production stages. Also, we could not find any holistic framework in literature which combines early prediction of problems with proper reactions in form of process optimization cycles in a fully automatized way, and which can be easily applied at different production sites where process data is recorded (i.e., which do not use any analytical, application-dependent process models).

1.2. Our Approach

The first part of our approach focusses on the prediction of quality criteria (QC), typically representing some dimensions, shapes, outlooks of the production items/parts, from an early stage to a latter or final production stage ($\rightarrow$ multi-stage functionality). We propose a new self-adaptive version of time-series based forecast models including several extensions for increased flexibility in order to achieve a reliable long-term prediction of the product quality; in particular, our modeling approach is grounded in the following aspects:

- Time-series based modeling purely from (recorded) process data by respecting the (measured) trends of process values over a specific time-frame occurring at the first (early) stage of production in order to predict the quality criteria in a latter stage. We respect the case where the quality criteria are measured occasionally (manually), rather than permanently, which leads to a (predictive) batch process modeling task (Section 2.1).
- The batch process modeling task typically results in very high-dimensional learning problems as the trends of past consecutive time-series samples for several process values at the first stage comprise the input space $\rightarrow$ we perform time-series transformation to reduce the curse of dimensionality (Section 2.2).
- Addressing the expected non-linearity contained in the system by using generalized Takagi-Sugeno fuzzy systems for establishing time-series based forecast models which are defined and trained in the latent variable space $\rightarrow$ achieving a non-linear version of PLSR (PLS-Fuzzy), see Section 2.2.
- Addressing the expected significant system dynamics over time by performing automatic on-line updates of the forecast models using rule evolution criteria and recursive (nearly exactly converging) parameter adaptations (Section 3).
- Improving the flexibility and robustness of the dynamic model updates by a.) integrating forgetting mechanisms in form of smooth sample outweighing over time in order to react on dynamic changes/drifts more quickly (Section 3.1), and by b.) incremental updating the latent variable space (PLS loading vectors) in order to account for possible changes in the covariance structure between inputs (time-series) and targets (QC measurements) (Section 3.2).

Our whole modeling approach will thus be termed as *SAFM-IF = Self-Adaptive Forecast Models with Increased Flexibility*.

The second part of our approach proposes a concept for providing supportive information and actions for operators in case when significant, unintended downtrends in product quality are observed (predicted) by the time-series based forecast models. This should help operators and machines to find appropriate reactions to the downtrends — depending on the time gap between the production phases under consideration and on the (induced) prediction horizon of the forecast models, there can be more or less time for providing the supportive actions. Our approach thus includes

i) a strategy for reducing the dimension of the optimization space in order to assure better and faster convergence — it comprises time-series compression and influence analysis of process values on QC targets based on the inner structure of the time-series based forecast models — and ii) a constrained heuristics-based search technique, which uses time-series based forecast models as surrogate mappings for permanent fitness calculations and which suggests better process values trends for balancing out the current undesired situation (bad QC values) based on Pareto-optimal individuals. In general, the search has to be conducted within a multi-objective problem setting, as more than one (and often antithetical) QC values can become deteriorating or can fall out of allowed bounds.

Both parts of our new approach ('self-adaptive time-series based prediction models' and the 'process optimization component') can be embedded in an on-line production system as shown by the grey-shaded components in the framework shown in Figure 1 — the only assumption thereby is that process data is permanently recorded, based upon which the whole production process can be supervised and maintained. This finally leads to a holistic and application independent predictive maintenance approach for production systems, which is able to combine early prediction of problems with proper reactions in form of process optimization cycles in a fully automatized way. Note: in our application case, we tried to directly predict from an early to a late/final stage of the production without taking into account intermediate stage(s) (hence, the arrow from the intermediate stage is shown in dotted line in Figure 1). Upon success, i.e. achieving forecast models over the stages with sufficient accuracy, this then would provide the longest possible prediction horizon and thus sufficient time for optimization and proper reaction (see Section 6 for the results). In other systems, where such a direct link is not achievable with reasonable accuracy, data from intermediate stages may be included as well for establishing proper forecast models. However, the proposed methods for forecasting and optimization as discussed below remain the same (only the data for process values at different stages need to be joined to one time-series based data set (batch training) or vector (model adaptation), respectively, before the modeling can start).

Our suggested components were evaluated based on a longer data stream recorded in a *multi-stage micro-chip production system*, lasting over 8 months (from October 2016 to May 2017). Therein, the early (first) stage of the production is the injection moulding process (IE), and the latter (last, third) stage is the bonding process (BL) (Section 5.1). Thus, we built and adapted our models by using the (appropriate) time-series trends of process values permanently recorded at the IE process as inputs (the trends before chips were finished there) and the final resulting QC values of the chips at the BL process as targets (Section 5.3). We show results of our forecasts models on the flatness of the chips in six different nest positions, on the RMSE values of the chips (two different ones, 050 and 095) as well as on the void events happening on the chips (Section 6). These three types of quality criteria comprise the most important quality criteria in the production system. It is known by experts that their synchronous optimization is not trivial, thus a real challenge for our approach.

Based on the results, it turned out that 1.) flatness and void events quality criteria can be predicted with an error significantly below 10% and this mostly with quasi-linearity in the *PLS-Fuzzy* models (having 1 to 3 rules), and 2.)

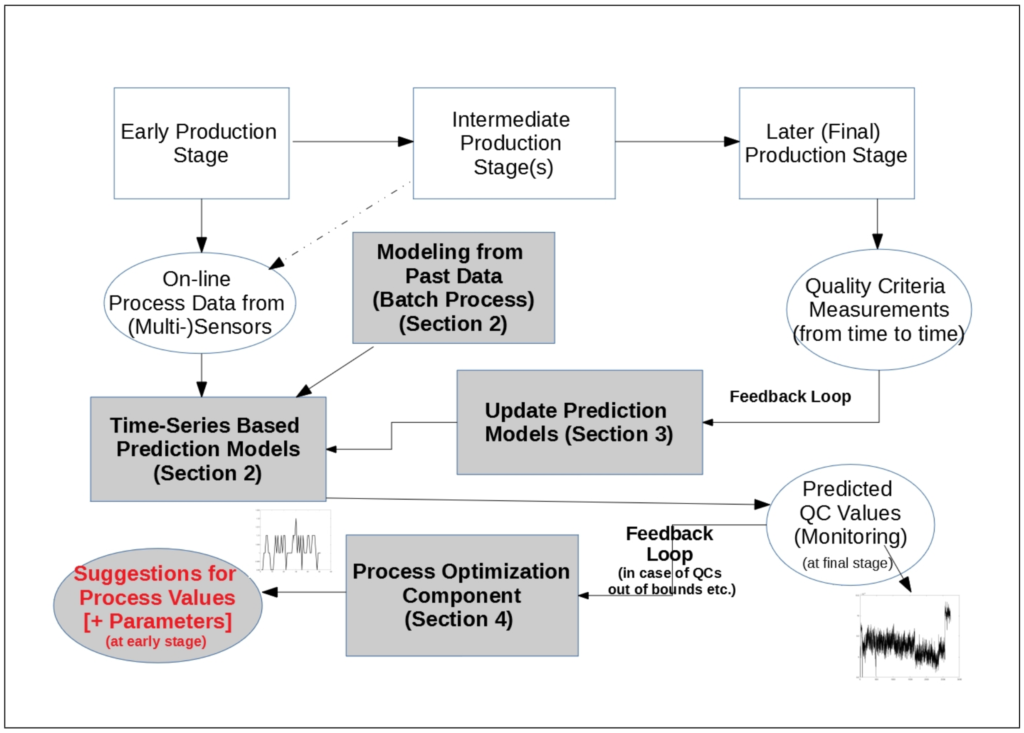

Early Production Stage
Intermediate Production Stage(s)
Later (Final) Production Stage
On-line Process Data from (Multi-)Sensors
Modeling from Past Data (Batch Process) (Section 2)
Quality Criteria Measurements (from time to time)
Feedback Loop
Time-Series Based Prediction Models (Section 2)
Update Prediction Models (Section 3)
Predicted QC Values (Monitoring) (at final stage)
Feedback Loop (in case of QCs out of bounds etc.)
Suggestions for Process Values [+ Parameters] (at early stage)
Process Optimization Component (Section 4)

Figure 1: Framework of a multi-stage production process, where QC values predicted for a latter (final) stage (based on current process data from an earlier stage) can be permanently monitored: i) time-series based forecast models, which can be updated whenever new QC measurement are available, serve as prediction engine; ii) whenever predicted QC values are observed as abnormal, fall out of bounds etc., a feedback loop to the process optimization component is triggered, which provides suggestions for improved process values trends (and associated machining parameters) to operators. These components, forming a holistic and automated predictive maintenance approach, are highlighted by grey shaded boxes and their embedded methodology (our solutions) will be described in the subsequent sections.

dynamic adaptation of the models helped to further reduce forecast errors, especially when being equipped with an incremental update of the latent variable space. The multi-objective process values optimization component was able to produce high qualitative Pareto fronts (with a fast convergence of their hyper-volumes over the populations) and thus to suggest process values trends which lead to a much better behavior of void events, flatness and RMSEs, i.e. to values which are closer to their ideal ones of 0 and 0.007, respectively. Thereby, the reduction of the process values (=optimization) space by concentrating on the most influencing process values (elicited by our new influence analysis technique) played a central role to achieve these good results, as optimization on the original process values space failed to optimize the flatness of the chips completely as not achieving any values below 50. Upon expert knowledge, the suggested (optimized) process values trends could be associated with proper machining parameters at the earlier production stage.

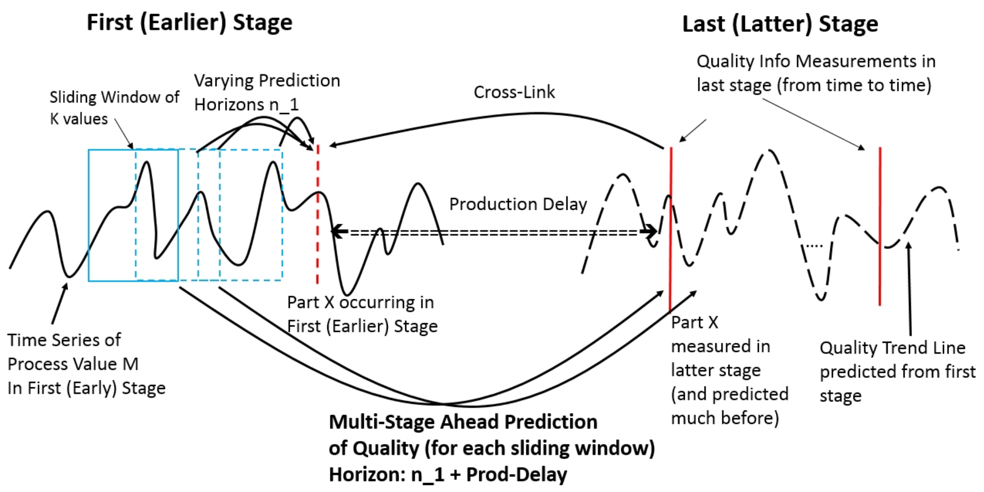

First (Earlier) Stage
Last (Latter) Stage
Quality Info Measurements in last stage (from time to time)
Varying Prediction Horizons n_1
Sliding Window of K values
Cross-Link
Production Delay
Part X occurring in First (Earlier) Stage
Time Series of Process Value M In First (Early) Stage
Part X measured in latter stage (and predicted much before)
Quality Trend Line predicted from first stage
Multi-Stage Ahead Prediction of Quality (for each sliding window) Horizon: n_1 + Prod-Delay

Figure 2: Multistage ahead prediction of quality criteria (QCs) in a latter stage (indicated by dotted black line) based on time-series data from process values at previous stages (solid dark line); here, only the univariate case is shown, but several process values are typically measured over time in the early stage and used as input for QC prediction.

2. Time-Series Based Prediction Models in a Batch Process

2.1. Problem Statement

Usually, the final quality of a product after the last stage is of utmost importance, as this may raise complaints at customer's side. Thus, predicting the final quality already at a much earlier stage (or even at the first stage), such that there is enough time for a proper reaction, would be an important key factor for a high performance predictive maintenance system with significant cost reduction. Whenever quality criteria are measured manually from time to time, a lot of man-power and costs are associated with the supervision. Thus, the measurements are typically taken occasionally from time to time, which leads to a forecast modeling problem within a batch-processing setting [2].

This leads to a signal flow of process values and measurement frequency of QCs as shown in Figure 2, where the vertical markers correspond to the time points where the QCs are measured at the last stage. The process signal (permanently measured) at the first (or earlier) stage is shown as bold solid line. We also indicate a cross-link of a measured, produced part in the latter stage to its production time at the first (earlier) stage by a vertical dotted marker. This link is required for model training and updating: the process trend before the part was produced at the earlier stage is taken as input data. The production quality forecast problem can then be tackled by establishing a prediction model, where the inputs are trends of the various process values over a particular time frame occurring at the first (earlier) stage before the same part (as measured during the last stage) has been produced, and the targets are the various quality criteria (QCs) measured for this part at the last stage. Different prediction horizons n_1 can be triggered by using different (sliding) windows of the process values occurring before the part has been produced at the first (or earlier) stage (indicated by bowed arrows). Additionally, the production delay between the first (earlier) and the last (latter) stages is an automatic given horizon to be added to n_1, which typically cannot be 'controlled' in the modeling process.

In the general case, we assume a multiple output system (i.e. Q various quality criteria), where all the targets are independent from each other and are generally numerical values reflecting the characteristics and shapes of the produced items; thus, we can treat each target separately and establish a separate forecast model for each:

$$QC_q(StageL) = f_q(\vec{x}_{StageE}(t-n_1), \vec{x}_{StageE}(t-n_1-1), ..., \vec{x}_{StageE}(t-n_2)) \tag{1}$$

$\forall q = 1, ..., Q$ with $\vec{x} = [x_1, x_2, ..., x_J]$, J the number of process values and f representing the input/output forecast mapping (the prediction model) for the qth criterion, $n_1 < n_2$, and n_1 the prediction horizon during the early stage; $n_2 = n_1 + k$ with k denoting the size of the window of past samples. Equation (1) means that Q QC criteria at Stage L are forecasted by consecutive process values ($\rightarrow$ trends) recorded n_1 up to n_2 instances before t at an earlier stage Stage E. t thereby is the point of time in the early stage when the same part is produced there for which the prediction of QC in the latter stage (StageL) is made. Whenever quality criteria are measured at the latter stage for a specific part (as indicated by vertical markers in Figure 2), they can be used for model building and updating as target values in (1). By establishing the cross-link back to the early stage, it can be recognized when the same part has been produced in the early stage (=t) and thus the trends n_1 up to n_2 instances before this point of time can be taken as input vector for the modeling process to establish f_q.

During the on-line process (or during the off-line evaluation process), the prediction of QC values at a latter Stage L can be permanently carried out by sending the latest $n_2 - n_1$ values at an early Stage E as input vector to (1) — as indicated by the two long arrows between the stages in Figure 2 (predictions carried out for two consecutive windows). Later, when the real QC info is measured at the latter stage(s), i.) it can be compared to the original predictions to check whether the model is still reliable, and ii.) it can be used to update the model in a supervised fashion (Section 3).

2.2. *Non-Linear Version of Partial Least Squares*

We are dealing with very high-dimensional data, as we have $J * k$ columns in the sample matrix $X = [\vec{x}_1(t - n_1), ..., \vec{x}_1(t-n_2), \vec{x}_2(t-n_1), ..., \vec{x}_2(t-n_2), ..., \vec{x}_J(t-n_1), ..., \vec{x}_J(t-n_2)]$, where typically J reflects the variety of sensor recordings and $k = n_1 - n_2$ the size of the window containing the past trends; $\vec{x}$ denotes the various samples from matrix X. Consecutive values in a time series can be expected to have similar information content when the dynamics between two or more measurements is not too high.

We thus develop a non-linear variant of partial least squares regression (PLSR) [16], which i) handles correlated 'neighboring' inputs in an appropriate fashion and thereby enjoyed a wide usage in data learning problems in the past where such cases occurred — see, e.g., the field of chemometrics [17] or in the field of control systems [18], [19] —, and ii) is able to resolve non-linearities in the co-variance structure between the input matrix X and the target $y = QC_q$. The latter provides us more flexibility to model possible non-linearities occurring in the relations between process values and QCs. For achieving non-linearity, there exist several non-linear variants of PLS in literature. One of the most prominent choice is kernel PLS [20]. However, they typically require an additional parameter (e.g., kernel width) to be set in advance and are exhaustive in the sense that they demand a large matrix of size $N \times N$. Furthermore, their interpretation capabilities are somewhat limited.

Therefore, our idea is to combine PLS with Takagi-Sugeno (TS) fuzzy systems, which have been successfully applied in a wide range of applications fields in the past for supervised regression and prediction problems with numerical target values [21], [22], [23]. They also have been successfully applied in data-stream processing tasks [23] and dynamic time-series based forecasting issues [22]. Furthermore, their architecture offers a partial partitioning of the input space into several regions in a natural way (sub-models represented by rules), for which partial direction vectors (latent variables) can be extracted in order to resolve locally varying co-variance structures between X and y. This sub-model structure also offers the possibility to evolve new components in new local regions on the fly in a pretty intuitive way [23] — in our case important for handling system dynamics in production sites, see also Section 3 below. We employ a specific variant of TS fuzzy systems, the *generalized* version of TS fuzzy systems, firstly introduced in [24], and which showed a better predictive performance in several past studies [25], [26], by inducing more compact rule bases with similar or even less model errors than conventional TS fuzzy systems.

The output of a (generalized) TS fuzzy system consisting of C rules is a weighted linear combination of the outputs produced by the individual rules (through the l_i's), thus:

$$\hat{f}(\vec{x}_s) = \sum_{i=1}^{C} \Psi_i(\vec{x}_s) \cdot h_i(\vec{x}_s), \quad \Psi_i(\vec{x}_s) = \frac{\mu_i(\vec{x}_s)}{\sum_{j=1}^{C} \mu_j(\vec{x}_s)} \quad , \tag{2}$$

with $\mu_i(\vec{x}_s)$ the rule firing degree obtained through

$$\mu_i(\vec{x}_s) = \exp(-\frac{1}{2}(\vec{x}_s - \vec{c}_i)^T \Sigma_i^{-1} (\vec{x}_s - \vec{c}_i)) \tag{3}$$

with $\vec{c}_i$ the center and Σ_i^{-1} the inverse covariance matrix of the ith rule, and with $h_i(\vec{x}_s) = w_{i0} + w_{i1}x_{s1} + ... + w_{ip}x_{sp}$ a linear hyper-plane describing the local regression trend of the i-th rule in the score space (with input dimensionality p). Thus, in case of generalized TS fuzzy systems, a single rule can be fully represented as a triplet of variables $(\vec{c}_i, \Sigma_i^{-1}, \vec{w}_i)$.

The connection between PLS and TS fuzzy systems can now be established in two ways:

- On global model level in the score space: therefore, PLS is performed first on the (standard normalized) data matrix X_{norm}, which is then transformed to the score space by the resulting loading vectors stored in the matrix P through multiplication, i.e. $X_S = X_{norm} * P$. The learning of and prediction with TS fuzzy systems then operate fully on the score matrix X_S including score samples $\vec{x}_s = \vec{x} * P$, which are typically of reduced space $p << J * k$. The reduced space can be elicited by using the p most important latent variables (with highest eigenvalues) explaining most of the $X - y$ co-variance structure in the data. Alternatively, it can be elicited by varying within a wrapper model training approach combined with cross-validation, see Section 5.

- On local rule-based level: therefore, $h_i(\vec{x}_s) = w_{i0} + w_{i1}x_{s1} + ... + w_{ip}x_{sp}$ is defined as a local linear PLSR function for each rule separately, where $x_{s1} = \vec{x} * p_{i1}, ..., \vec{x} * p_{ip}$ are the p most important loadings for the ith local region. Hence, the loading vectors are different for each rule, thus locally extracted with those samples 'forming' a rule (for which the rule had highest membership degree). This yields indeed more degrees of freedom in terms of representing local co-variance structures, but the antecedent parts of the rules then operate still on the full original input space.

As we do not have the number of rules available in advance and thus have to learn them from data (see subsequent section) and as our original input space is typically very high-dimensional, the only feasible choice is the global connection which reduces the input dimensionality for the TS models in advance, such that the curse of dimensionality effect can be reduced.

2.3. Robust Learning of Generalized TS Fuzzy Systems in the Latent Variable Space (PLS-Fuzzy)

As the non-linearity degree of the learning problem is not a priori known, it is wise to estimate the appropriate number of fuzzy rules during the initial stage of fuzzy model construction. Therefore, we exploit the *Gen-Smart-EFS* algorithm [25] by passing all the available samples in a full data set through its core learning engine in a single-pass manner (thus as pseudo-stream), after projecting all of these to the score space (see above). The rule evolution criterion is the central formula which steers the generation of the rules and thus is responsible for the final non-linearity degree the fuzzy model represents. This criterion incorporates the statistically motivated prediction interval, which serves as statistical tolerance region [27], whose boundary can be elicited by the χ^2 quantile with p degrees of freedom. Thus, at a default significance level of $\alpha = 0.05$ it is defined as:

$$\left(min_{i=1,...,C} \;\; mahal_i = \sqrt{(\vec{x}_s - \vec{c}_i)^T \Sigma_i^{-1} (\vec{x}_s - \vec{c}_i)}\right) > r_{win}, \quad win = argmin_{i=1,...,C}(mahal_i)$$
$$r_{win} = \text{fac} * p^{1/\sqrt{2}} \frac{1.0}{(1 - 1/(k_{win} + 1))^m} \tag{4}$$

Equation (4) thus elicits the closest rule to the current score sample $\vec{x}_s$ in terms of the Mahalanobis distance and checks whether it falls out of fac-times of the statistical tolerance region $p^{1/\sqrt{2}}$: this term approximates well the χ^2 quantile with p degrees of freedom, see also [28]; fac is a tuning factor steering rule evolution versus rule update, and the only sensitive parameter in our method (typically tuned in a batch off-line CV procedure). If (4) is fulfilled, a new rule is initialized with its center to the current sample and the inverse covariance matrix as an average of inverse covariance matrices of neighboring rules. If it is not fulfilled, the nearest rule with index win is updated by: i) its center $\vec{c}_{win}$ is moved towards the sample with a fraction according to the generalized vector quantization concept, and ii) its inverse covariance matrix Σ_{win}^{-1} is recursively (exactly) updated by a formula obtained through the usage of the Neumann series trick [28] (the index *win* omitted here for transparency reasons):

$$\vec{c}(k+1) = \vec{c}(k) + \eta_k(\vec{x}_s - \vec{c}(k)) \quad \Sigma^{-1}(k+1) = \frac{\Sigma^{-1}(k)}{1-\alpha} - \frac{\alpha}{1-\alpha} \frac{(\Sigma^{-1}(k)(\vec{x}_s - \vec{c}))(\Sigma^{-1}(k)(\vec{x}_s - \vec{c}))^T}{1 + \alpha((\vec{x}_s - \vec{c})^T \Sigma^{-1}(k)(\vec{x}_s - \vec{c}))}, \tag{5}$$

with $\eta_k = \frac{1}{k}$ and $\alpha = \frac{1}{k+1}$, and k the number of samples seen so far for which c_{win} has been the winning rule (cluster).

After the rules have been initially trained and the adequate number of rules elicited, we apply a fine tuning step of the rule antecedents. Therefore, we iterate over the whole data set a multiple times and optimize centers and inverse covariance matrices of the rules according to the expected quantization error. Therefore, we exponentially decrease the learning gain by the number of iterations in order to assure convergence, and stop the algorithm when the change in the rule centers between two consecutive cycles is smaller than ϵ.

Finally, the learning of the linear consequent parameters is robustly achieved through the concept of elastic net regularization [29] in order to omit instabilities in the matrix inversion (e.g., due to non full rank or high condition of the Hessian matrix), the latter is applied to the fuzzily weight least squares objective function $J_i = \sum_{k=1}^{N} \Psi_i(\vec{x}_s(k)) e_i^2(k)$, which yields the following optimization problem:

$$J_i = \sum_{k=1}^{N} \Psi_i(\vec{x}_s(k)) e_i^2(k) + \lambda \sum_{j=1}^{p} (\alpha w_{ij}^2 + (1-\alpha)|w_{ij}|) \longrightarrow \min_{\vec{w}_i} \tag{6}$$

where $e_i(k) = y(k) - \hat{y}_i(k)$ represents the error of the local linear model in the kth sample, λ the regularization parameter and α a parameter in $[0, 1]$, steering the degree of influence of the Lagrangian term stemming from the classical Lasso approach, i.e., $\sum_{j=1}^{p} |w_{ij}|$, versus the Lagrangian term inspired by classical ridge regression, i.e., $\sum_{j=1}^{p} w_{ij}^2$. The weighted version of least squares (first term in (6)) is used in order to trigger the local learning variant which has several important advantages over global learning — as deeply analyzed in [30]. The problem in (6) can be solved through a quadratic programming approach, termed as LARS-EN, see [29].

3. Evolving Time-Series Based Prediction Models by Dynamic Updating

Once the forecast models have been established (and fully evaluated) in batch mode, it is a challenge to keep the models up-to-date during further on-line production due to intrinsic system dynamics (as is the case in our scenario, see Section 5).

An unsupervised update of the fuzzy systems, i.e. by only updating the antecedent space [31], can be carried out for each new window of process values occurring during the production process at the first (early) stage (available as on-line stream) — a favorable property of fuzzy systems as they decompose the model structure into an unsupervised antecedent and a supervised consequent part. The update can be achieved by the same techniques (rule evolution criterion, recursive inverse covariance matrix update, center movement) as described in the previous section. However, typically an unsupervised update only can achieve a fine tuning of the models' parameters to improve their positioning towards more dense region which may turn out more and more over time (and which were not so well reflected in the initial batch data set). However, it cannot cope with real system dynamics which often makes changes in the consequents, in the rule structure and even in the covariance structure between X and y (the latter inducing changes in the latent variable space) necessary. Therefore, we use each QC measurement newly recorded at the latter (last) stage (= target values y) for updating the models. The assumption thereby is that a cross-link to the earlier stage is established (as shown in Figure 1) in order to identify when the measured part occurred in the earlier stage.

The process values trends before the production of the corresponding part in the earlier stage are used as input vector $\vec{x}$ and the currently measured QC-info at the last stage as target value y and projected to the score space: $\vec{x}_s = (\vec{x} * P, y)$. $(\vec{x}_s, y)$ is used to update the antecedent part and the rule structure by using (5) and (4) and also to recursively update the consequent parameters. The latter are linear parameters and thus updated by recursive fuzzily weighted least squares approach, converging to the real optimal solution in each update step immediately:

$$\hat{\vec{w}}_i(N+1) = \hat{\vec{w}}_i(N) + \gamma(N)(y(N+1) - \vec{r}_s^T(N+1)\hat{\vec{w}}_i(N)) \tag{7}$$

$$\gamma(N) = \frac{P_i(N)\vec{r}_s(N+1)}{\frac{\Lambda}{\Psi_i(\vec{x}_s(N+1))} + \vec{r}_s^T(N+1)P_i(N)\vec{r}_s(N+1)} \tag{8}$$

$$P_i(N+1) = \frac{1}{\Lambda}(I - \gamma(N)\vec{r}_s^T(N+1))P_i(N) \tag{9}$$

Equations (7) to (9) emphasize the local learning spirit (where each rule has its own separate update) by including Ψ_i in the denominator of the Kalman gain update (second formula). Thereby, we use the regularized inverse Hessian matrix resulting from the elastic net regularization as starting point, thus for the i-th rule $P_i = (X_s^T Q_i X_s + \alpha I)^{-1}$. $\vec{r}_s(N+1) = [x_{s1}(N+1)\ x_{s2}(N+1)\ \dots\ x_{sp}(N+1)\ 1]^T$ denotes the regressor values of the $N+1$th data sample containing p scores and including the 1 for the intercept, which are the same for all i rules. Λ denotes a forgetting factor, whose default value is equal to 1 (no forgetting).

Antecedent and consequent updating as well as evolving rules on demand in single-pass streaming manner denote the core functionality of our learning engine. However, we intend to increase the flexibility of the update process in order to account for significant system dynamics also in terms of fluctuating changes, drifts and changes in the co-variance structure. Therefore, we include two basic functionalities in the model update, which are (compactly) described in the subsequent sub-sections.

3.1. Dynamic Forgetting

Our approach is leaned on the concepts demonstrated in [32] for dynamically adjusting the forgetting factor Λ in the adaptive learning and evolution process of the fuzzy models. This factor is integrated

1. in the recursive fuzzily weighted least squares formulas for the rule consequents, see Equations (7) to (9).
2. in the resetting of the dynamic learning gain η in (5) for updating the rule centers (automatically also increasing the flexibility of the update of the inverse covariance matrix), see also [33].

The basic idea of *dynamically adjusting* Λ is that no forgetting is used as long as there is no explicit (local) drift occurring (and detected) in the system $\rightarrow$ thus, the forgetting factor is set to 1 leading to conventional life-long learning — this usually helps to improve the convergence of learning and thus the robustness of the evolving models during regular operation modes (as analyzed in [32]). On the other hand, increasing the degree of forgetting in case of (local) drifts is important to enforce a (stronger) rule movement towards the new data distribution in order to avoid a rule including a mixture of old and new local data distributions, which typically increases the model error [34].

Once a new QC measurement is available, we update the model as well as its prediction error. The updates are amalgamated and tracked over time to extract the trend of the error, which can be used to realize when the model is drifting: a statistically significant increase of the error indicates a drift. For eliciting statistical significance, we apply a modified version of the Page-Hinkley test statistics [35] by including a fading factor in it to increase its flexibility [36]. Then, the forgetting factor is decreased (thus the degree of forgetting increased) by a fraction of the intensity of the gradient of the PH statistics:

$$\Lambda_{N+1} = min(max(\Lambda_N - C_{N+1}, 0.9), 1.0), \qquad C_{N+1} = \frac{PH_{N+1} - PH_N}{rmse_{N+1}\rho} \tag{10}$$

where Λ_{N+1} is the forgetting factor after $N + 1$ samples (with maximal value = 1 $\rightarrow$ no forgetting and minimal value = 0.9 $\rightarrow$ intense forgetting), and C_{N+1} is the amount of change in accumulated drift intensity, $PH_N - PH_{N+1}$ denotes the gradient in the PH statistics over two consecutive time points and ρ a scaling factor typically set to a high value (around 1000); *rmse* is an estimator of the root mean squared model error based on past samples (e.g., used during initial batch training of the PLS-fuzzy model).

Inbetween two consecutive QC measurements (which can be significantly delayed), an unsupervised forgetting factor setting strategy based on local drifting distributions is conducted. It relies on the weighted Kullback-Leibler divergence $D_{KLW}(G_1||G_2)$ between a local data distribution in the current data chunk (G_1) and the nearest rule to it (G_2) (proof left to the reader):

$$D_{KLW}(G_1||G_2) = w_1\left(log(\frac{w_1}{w_2}) + D_{KL}(G_1G_2)\right), \quad D_{KL}(G_1||G_2) = \frac{1}{2}(trace(\Sigma_2^{-1}\Sigma_1)+(\vec{c}_2-\vec{c}_1)^T\Sigma_2^{-1}(\vec{c}_2-\vec{c}_1)-p-ln(\frac{det(\Sigma_1)}{det(\Sigma_2)})) \tag{11}$$

with w_1 the number of samples in the current chunk and w_2 the number of samples forming Rule G_2; $\vec{c}_1$ and $\vec{c}_2$ are the centers of the current chunk and of rule G_2, respectively; Σ_1 and Σ_2 are the covariance matrices of the current chunk and of rule G_2, respectively. The forgetting factor is adjusted according to the trend line of Kullback-Leibler (KL) divergence values per rule over time. The formula in (10) is applied to each rule separately (achieving different $\Lambda_{N+1}(i)$'s for the different rules) with *PH* calculated based on these KL trend lines.

3.2. Incremental Update of the Latent Variables (PLS Space)

An update of the latent variable space (loadings $p_1, ..., p_p$) is necessary if there is an intrinsic change in the covariance structure between X and y, i.e. the basic characteristics between the mapping of process values trends and QC-info changes. This can, for instance, become relevant in case of new product types and non-stationary influencing environments [31].

Updating the projection directions builds on the concepts proposed in [37] and is based on the following steps:

- Recursive (exact) update of the first projection direction p_1 by exploiting the fact that, according to the NIPALS algorithm (also termed PLS1) [38], the first *un-normalized* direction can be represented as $q_1 = X^t y = 2\lambda_1 p_1$ (with λ_1 the eigenvalue of p_1), which is due to the maximization objective based on the covariance between Xp_1 and y. Then, it is easy to see that

$$q_1(N+1) = q_1(N) - N\vec{\mu_y}(N)\Delta(N+1) + y(N+1)\vec{x}_{N+1}(N+1), \tag{12}$$

  where $\Delta(N+1) = \vec{\mu}(N+1) - \vec{\mu}(N)$, and $\vec{\mu}(N)$ is the mean of the input features estimated from the first N samples; $\vec{\mu_y}$ is the mean of the target, and $\vec{x}_{N+1}(i)$ is the ith mean-centered sample using the mean $\vec{\mu}(N+1)$ of all inputs which can be elicited incrementally from the first $N+1$ samples.

- Obtaining the remaining projection directions $p_2, p_3, ..., p_p$ via the Krylov sequence (upon applying a fast sorting algorithm [39]), whose Gram-Schmidt orthogonalized form is given by $P = [q_1, Cq_1/q_1, ..., C^{k-1}q_1/\{q_1, Cq_1, ..., C^{k-2}q_1\}]$, where C is the global covariance matrix, which can be recursively updated [40]. This leads to the remaining PLS projection directions through successive iterations from k to $k+1$, starting with $k=1$:

$$\begin{aligned} q_{k+1}(N+1) &= Cq_k(N+1) \\ q_{k+1}(N+1) &= q_{k+1}(N+1) - q_{k+1}(N+1)^T \frac{q_k(N+1)}{||q_k(N+1)||} \frac{q_k(N+1)}{||q_k(N+1)||}, \end{aligned} \tag{13}$$

- By normalizing $q_1, ..., q_p$ subject to the L^2-norm, the updated projection directions $p_1(N+1), p_2(N+1), ..., p_p(N+1)$ are obtained.

Alternatively, instead of updating the global covariance matrix (which may be large and thus time-consuming whenever the input dimensionality $J * k$ is huge), also the corresponding principal components can be updated by, e.g., using the well-known CCIPCA approach as proposed in [41]. The covariance matrix can then be approximated by the principal components direction after each update step.

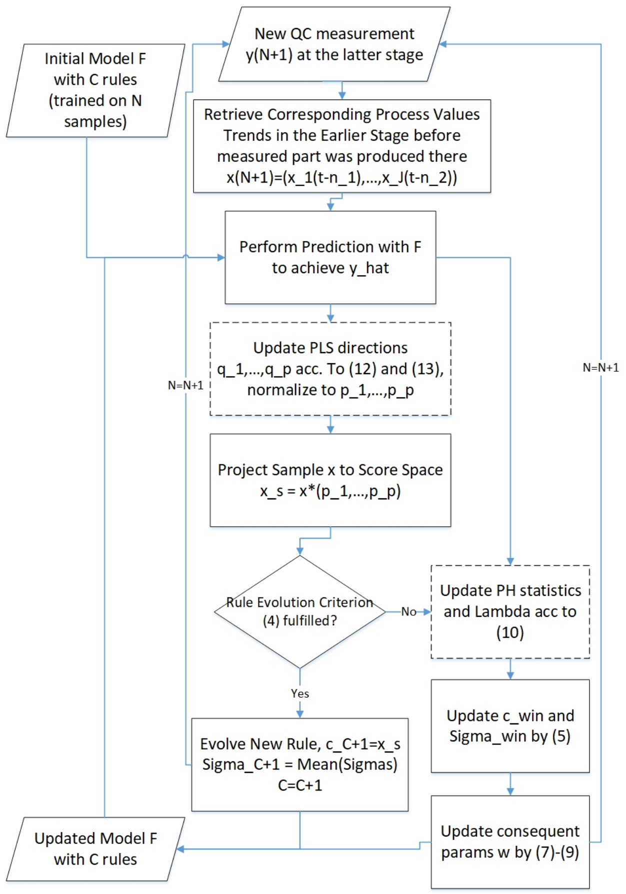

Initial Model F with C rules (trained on N samples)
New QC measurement y(N+1) at the latter stage
Retrieve Corresponding Process Values Trends in the Earlier Stage before measured part was produced there x(N+1)=(x_1(t-n_1),...,x_J(t-n_2))
Perform Prediction with F to achieve y_hat
Update PLS directions q_1,...,q_p acc. To (12) and (13), normalize to p_1,...,p_p
N=N+1
Project Sample x to Score Space x_s = x*(p_1,...,p_p)
Rule Evolution Criterion (4) fulfilled?
No
Yes
Update PH statistics and Lambda acc to (10)
Update c_win and Sigma_win by (5)
Evolve New Rule, c_C+1=x_s Sigma_C+1 = Mean(Sigmas) C=C+1
Update consequent params w by (7)-(9)
N=N+1
Updated Model F with C rules

Figure 3: Work-flow for prediction model adaptation through SAFM-IF based on a newly available QC measurement; it comprises the update concepts with increased flexibility as discussed throughout this section.

3.3. The Algorithm for Self-Adaptive Forecast Models with Increased Flexibility (SAFM-IF)

Comprising all the incremental and adaptive learning concepts from the preliminary subsections, the following algorithm as shown in the work-flow in Figure 3 is yielded. Optional components for increased flexibility are highlighted as dashed boxes: we switched these on and off in the experiments to realize their impact on the model error trends.

4. Process Optimization based on Time-Series Based Forecast Models

This section describes how the currently running production process can be optimized whenever undesired trends of QC values at the latter (final) stage of production are already observed during the earlier stage (due to the permanent

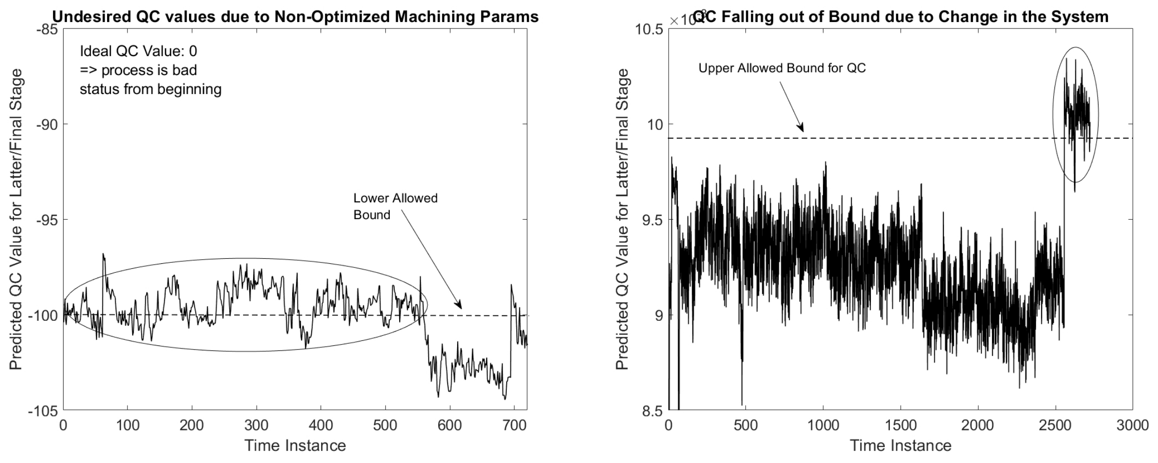


Figure 4: Left: an example of a process where QC values were predicted as undesired ones from the beginning on (close to their lower allowed bound, but far away from the ideal value of 0); this happened due to non-optimized machining parameters; right: an example of a process where QC values were predicted as normal within their bounds, but towards the end a change in the system happened which requires a proper reaction.

predictions obtained from the multi-stage forecast models). Therefore, the time-series based forecast models as established with the methodologies described in the previous sections will serve as surrogate models in a multi-objective optimization procedure. First, we provide a detailed problem statement of the process optimization procedure and then demonstrate a method how to tackle it adequately.

4.1. Problem Statement

Whenever the process values forecast problematic, undesired QC values (trends), an on-line adjustment of respective process values (and associated parameters) in the 'right direction' is demanded in order to prevent significant quality damage or even system failures that eventually translate to monetary loss for the company (compare with Figure 1, 'process optimization' component). Such undesired QC values (trends) are typically caused either due to non-optimized machining parameters for the actual product type/charge, as they are used as long-term standard values subject to operators' experience [15], or due to non-stationary environmental influences or unpredictable changing system dynamics. Real-world examples of these two main problematic scenarios are visualized in Figure 4. The right image shows the case where for a longer time frame (of around 2500 time units) the predictions of the QC values lie below the upper allowed bound (horizontal dashed line), but then suddenly exceed this bound due to an unexpected change in the production process or its environment → a proper reaction is desired to stay in-line the bounds. The left image shows the case where from the beginning on the QC values are close or even below the allowed lower bound (marked by an ellipsoid), as being caused by a non-optimized (standard) machining parameter setting → a proper reaction is desired to drive the process to better values, ideally to QC values which are close to their optimal ones.

Thus, the goal is to seek for process values trends in the early stage that lead to improved, (close-)optimal QC values in the latter stage ($\rightarrow$ process optimization). This may lead to a many-objective optimization problem, depending on the number of QC values q that are undesired, or even become out of bounds. Often, there are also several QC values which are somewhat redundant to each other, i.e. showing very similar (predicted) trends over time. In such a case, only one of a redundant group of QC values can be used a target to be optimized, which then may lead to a classical multi-objective optimization problem with 3–5 objectives (as will be the case in our case study for chip production, see the experiments section below).

Whenever the optimization is carried out on the whole process value space, the dimensionality of this space (termed as 'optimization space') can become pretty high as one needs to consider entire time-series trends with length k for a rather larger number of process values J (see Section 2.1) in order to construct meaningful prediction models. These models serve as surrogates in the optimization procedure for fitness evaluation. The predictions of these models should get close to the optimal values $opt_i, i = 1, ..., q$. In order to increase the success of convergence of optimization and this within a reasonable amount of time, we aim for a reduced dimensionality of the optimization space subject

1. to an influence analysis based on the (structure of the) PLS-fuzzy models, where the most important process values in the models for obtaining accurate predictions of QC values are elicited.
2. to a compressed information about each time-series piece with some basic statistical features.

How we have tackled both issues, will be described in detail in Section 4.2.

Therefore, when considering:

1. a total number of $q' < q$ problematic QC indicators,
2. a dimensionally reduced number of p' process values that are deemed as most influencing for the predicted QC indicators,
3. a general process values trend size of k,
4. the wish to describe trends using a compact summary description by s statistical measures $stat_1, ..., stat_s$ for each influencing process value,

the process optimization task can be formalized as:

$$\begin{aligned} f_i(\vec{x^{agg}}) &= opt_i!, \quad 1 \leq i \leq q' \qquad (14) \\ \text{subject to } x_j^{agg} &\in [l_j, u_j], \quad 1 \leq j \leq s \cdot p', \\ \text{where } \vec{x^{agg}} &= [stat_{11}, ..., stat_{1s}, stat_{21}, ..., stat_{2s}, ..., stat_{p'1}, ..., stat_{p's}] \end{aligned}$$

describes a solution in the optimization space. $f_i, 1 \leq i \leq q'$ are the forecasting models as defined in (1) and obtained in advance; they are used as surrogate models for fitness evaluation during optimization: their predictions can be directly compared with the pre-defined, known optimal values of the QC indicators, i.e. a fitness function can be easily defined, which measures the distance between a predicted $\vec{f}(\vec{x^{agg}})$ and the optimal QC indicator vector $\vec{opt}$. Typically, $s << k$ for a compact representation and $p' << p$, such that a significant reduction of the optimization space is achieved. It is worthy to mention that for an un-optimized production process, Equation (14) can be interpreted as a 'free optimization' problem as there are no secondary constraints on the statistical features extracted from the time-series (only primary constraints in form of maximal upper and lower allowed values).

In the case of pre-optimized production processes (an example is shown in the right image in Figure 4), problematic QC predictions are likely generated by i) unknown system dynamics and ii) unexpected environmental influences which arise during the production. In such a case, we wish to restrict solver solutions to a close vicinity of the current process values trends i) by reducing the lower and upper bound search intervals for each relevant x_j^{agg} from Equation (14) and ii) by introducing an extra optimization objective:

$$min_{i=1,...,T}\left(d(x_i^{agg\_old}, x^{agg\_new})\right) = \text{min!} \tag{15}$$

with d a distance function and T the number of past process values trends $x_i^{agg\_old}, i = 1, ..., T$ used (typically the most recent ones whenever the process was in 'good' state, i.e. with predicted QCs well inside their bounds). Thus, the closeness of a (new) optimized process values trend x^{agg_new} to any previously (real) occurring trends of process values is respected as additional objective in order to regularize the free optimization problem by restricting it to smaller changes in the process values. The minimum is taken in (15) in order to omit a blurring of the optimized trends in case of pretty different past trends.

4.2. Reducing the Dimensionality of the Optimization Space

A first important step before the optimization process can start concerns a reasonable dimensionality of the input space of the optimization problem. This is important to allow convergence of the solver, at least within a reasonable amount of time [42], [43]. When a significant number of original process values $x_1, ..., x_J$ is recorded during production, it soon may end up with a few hundreds of inputs for the optimization solver (also when s is small). In such a case, it is expected that the convergence and thus the speed of the optimization algorithm is slowed down drastically. Hence, we suggest to perform an influence analysis between process values (trends) and (a subset of) quality criteria to be optimized: only those process values with a significantly higher influence compared to others are used in the optimization process — a slight change in such influencing process values can already have a significant impact on the QC values, whereas a large change in non-influencing ones do contributes only slightly in the optimization process; thus, better solutions (if any) can be only achieved with much more optimization cycles and thus computational costs. This also reduces the on-line applicability of the optimization process.

In our case, the influence of a process value can be calculated through its loadings included in all the latent variables finally used in the PLS-fuzzy models (see Section 2.2). This is because the loadings denote the significance which the single process values have to explain the covariance structure between input and targets. Process values with low loadings will be masked out when calculating the scores which are the inputs to the PLS-Fuzzy forecast model. Additionally, the impact of the scores in the consequent hyper-planes of the generalized TS fuzzy systems is important, as these indicate the regression trends in the rules (which are sub-models of local regions). Higher weights $\vec{w}$ in the hyper-planes of one or more rules indicate more sensitivity and thus importance of a score to predict the target. Assuming that the fuzzy model contains C rules (evolved automatically during model training), we calculate $influ_{i;r}$, which is the influence of the ith process value onto the rth target (quality criterion) to be optimized, by:

$$influ_{i;r} = \frac{1}{k}\sum_{l=1}^{k}\sum_{j=1}^{p}\left(|p_{il,j}|(\frac{1}{C}\sum_{h=1}^{C}|w_{hj}|)\right). \tag{16}$$

p denotes the number of latent variables in the PLS-fuzzy model and $p_{il;j}$ denotes the loading of the L^2-normalized jth principal component direction in the lth time-series sample of the ith process value (k samples in sum due to the sliding window size k); thus, the influences of all time-series samples of one process value are averaged. The L^2-normalization is important in order to assure the same range of the loadings over all loading vectors, which are themselves used as inputs to the PLS-fuzzy model without any weights (so, the model has been trained by seeing all scores produced by all components as equally important). w_{hj} denotes the consequent parameter (=regression coefficient) of the score obtained through the jth component in the hth rule.

Then, the influence of a single process value can be checked versus the others, i.e. if

$$influ_{i;r} > \mu(\vec{influ_r}) + \sigma(\vec{influ_r}), \tag{17}$$

with μ the mean and σ the standard deviation of the influences of all process values onto the rth target, the ith value has a significant influence onto quality criterion r and thus should be taken into account for optimization. The joined set of influencing variables over all q' quality criteria to be optimized is then taken as input parameter space:

$$Infl\_set = \{\bigcup_{i=1}^{J}\vec{x}_i \;\mid\; \exists_{r\in\{1,\dots,q'\}}\;\text{(17) is fulfilled}\} \tag{18}$$

with $\vec{x}_i$ the ith process value and $p' = |Infl\_set|$.

After having elicited the most influencing process values, we further suggest a reduction of the optimization space by applying statistical features in order to achieve a compact representation of the time-series trends with window size k (rather than using each single raw sample as input dimension in the optimization space). It is finally a matter of the concrete application scenario at hand how these process values trends look like and thus which features are the best options to use. In Section 5.3, we will discuss appropriate features for the particular case of an injection molding process at a chip manufacturing site.

4.3. Solving the Optimization Problem and Providing Suggestions for Process Values Trends

Nearly all classical techniques of solving an optimization problem as defined in Equation (14) are based on the central idea of reducing / restating its original many/multi-objective formulation to one or several independent single-objective optimization problems. Obviously, the main advantage of this strategy is that the restated problem(s) can be tackled using well established single-objective optimization strategies. Well known techniques for transforming a many/multi-objective optimization to a single-objective one include the Tschebyscheff min-max criterion, the global criterion, the weighted sum method, and goal programming [44]. When aiming for a reduction to a set of well-spaced independent single-objective problems (each defined for a specific trade-off between the original objectives), the normal boundary intersection [45] and the normal constrained [46] methods stand out among classical techniques. The disadvantages of reducing the many/multi-objective problems to a single-objective one is that some articulation of preference (among the objectives) might be required before the start of the optimization and that the obtained single-objective optimization problems are themselves usually very hard to solve by classical gradient-based mathematical methods because of inherent problematic characteristics like non-linearity, multi-modality, non-differentiability, and small global solution attraction basins. Furthermore, classical gradient-based methods are often trapped in local optima, mostly dependent on the starting point (initial configuration) of the optimization procedure.

Derivative-free stochastic optimization techniques (i.e., meta-heuristics) are a widely-used, promising alternative to gradient-based methods in order to advance the search space [47] [48] and thus to improve (the quality of) the solution by helping non-ideally converged solutions out of their local optima. This can be achieved through varying mutation and crossover rates and intensity levels. Among various derivative-free meta-heuristics based methods, population-based approaches like evolutionary algorithms (EAs) [49] have a structural advantage as the parallel and cooperative solution space exploration paradigm they propose can easily capitalize on computational power improvements and is arguably more robust regarding parametrization than trajectory-based meta-heuristics. For example, an EA tries to continuously improve a population (i.e., set) of individuals (i.e, representations of solution candidates for the optimization problem to be solved) during a computation cycle that uses various operators inspired from natural evolution (e.g., selection, recombination, mutation, survival) to create new individuals (i.e., offspring) that (can) replace their parents inside the population.

When considering multi-objective optimization problems (MOOPs), where one usually aims to find sets of Pareto non-dominated solutions (PNs) that encompass the best trade-offs between several (i.e., 2 to 4) conflicting objectives, specialized EAs (like NSGA-II [50] and SPEA2 [51]) have turned out to be lucrative tools in various optimization applications. The main advantage of multi-objective evolutionary algorithms (MOEAs) compared to classical optimization methods, is that, by slightly adjusting the evolutionary model, these EAs are able to discover full PNs in a single run. For example, NSGA-II [50] proposes a highly elitist evolutionary model (illustrated in Figure 5). At each iteration t of the evolutionary cycle, NSGA-II generates an offspring population $O(t)$ from a parent population $P(t)$. The selection for survival mechanism constructs population $P(t+1)$ by selecting the best $|P(t)|$ individuals from the combined population $C(t) = P(t) \cup O(t)$ based on a primary non-dominated ranking strategy and a secondary (tie-breaker) objective space crowding strategy. When applying non-dominated sorting, the highest-level front $F_1(t)$ contains all the individuals that form the PN of $C(t)$. The next (lower-level) fronts $F_j(t)$, $j > 1$ are obtained by itera-

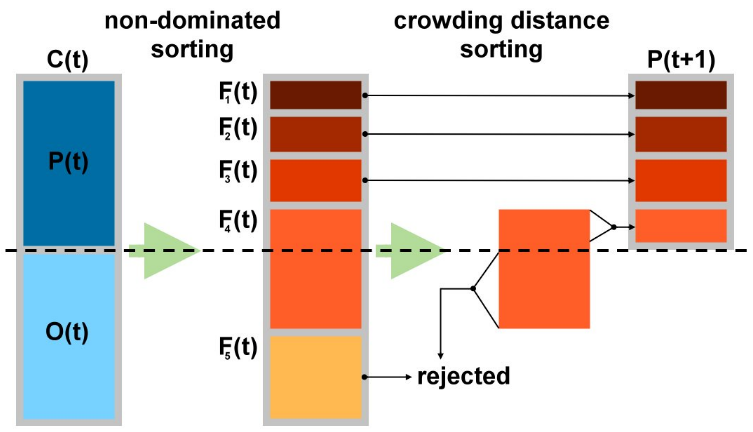


Figure 5: The NSGA-II evolutionary model

tively removing higher level Pareto fronts from $C(t)$ and extracting the Pareto optimal set from the remaining set of individuals. In order to differentiate between individuals that are part of the same front, NSGA-II (1) computes the average crowding distance in objective space between a given solution and the two closest neighbors that surround it on each objective axis and (2) opts for individuals from less densely populated areas.

In the case of our optimization scenario, individuals are encoded via a real-valued fixed-length vector that aggregates the compact representation features of the most influencing process values:

$$x^{\vec{agg}} = [stat_{11}, ..., stat_{1s}, stat_{21}, ..., stat_{2s}, ..., stat_{p'1}, ..., stat_{p's}] \tag{19}$$

where p' is the total number of influencing process values for all QC targets and s is the number of features used for the compact representation of each process value trend. Thus, each individual contains $s * p'$ allele which fully define the optimization (i.e., variable) space.

Assuming to have the ideal values of the (product) quality criteria (QCs) available (usually, these are known by experts of the concrete production process), denoted as $opt_i, i = 1, ..., q'$, the fitness for each single objective is calculated as the deviation of the actual value of the ith QC criterion (as predicted by the corresponding forecast model f_i) from its ideal value:

$$fitness(x^{\vec{agg}}) = |f_i(x^{\vec{agg}}) - opt_i| \quad \forall i = 1, ..., q' \tag{20}$$

The absolute value has to be taken in order to ensure that minimization towards 0 deviation is achieved properly. This fitness value is used by NSGA-II when performing both the non-dominated and the crowding-based sorting. Thereby, the fitness of each objective is collected for each individual, and a so-called Pareto front in the optimization space is established by those individuals which are not dominated by any other individuals with respect to *all* objectives. Based on the Pareto front and the remaining individuals in the population, the next generation is constructed by NSGA-II.

This is repeated until a fixed number of iterations is reached or until the quality of the Pareto fronts becomes saturated. The final outcome is a Pareto optimal set of solutions (see Figure 14 for an example) where that (those) one(s) closest to the origin (= those with highest fitness) can be suggested to experts/operators (see Section 6.3.2).

5. Experimental Setup

5.1. Application Scenario

In our case study, we deal with the inspection of micro-fluidic chips used for sample preparation in DNA (deoxyribonucleic acid) sequencing. On the chip, the DNA and primers are packed into aqueous droplets in oil phase. Currently, they are checked in a diagnostic instrument by means of image inspection in a closed loop. This is done in an a posteriori manner, where bad chips are sorted out once they have already been produced (in order to prevent bad parts for customers), based on machine learning classifiers as developed during a preliminary project, see [52]. This, however, typically does not prevent unnecessary waste and can even induce greater complications and risks at the production system.

Therefore, predicting downtrends in the quality of the chips at an early stage rather than in a later stage, in order to decrease or even completely avoid waste and risks, is an important challenge to be addressed. In particular, based on 63 continuously measured process values found to be useful for tracking the production behavior at the injection moulding machine (= first stage of the whole production process), time-series based forecast models should be established from data which accurately predict important quality criteria in the last stage of production (bonding liner). Intermediate stages such as the oven (for heating up the chips) may have an impact on the quality as well, but this reduces the whole predictive maintenance horizon and efficiency much. Hence, an ambitious goal was to predict from first stage directly to the quality in the last stage, also to see how much of the final quality is already determined there and whether process optimization can reliably take place (solely) in such an early stage. We concentrated on the supervision and optimization of the three most important groups of quality criteria for the company and its customers, namely *flatness* of the chips, *RMSE* (inner size of the chips) and *void events* happening on the chips. Flatness and void events are measured with respect six nest positions of the chips at the bonding liner.

5.2. Data Extraction and Preparation

The company partner is using an MS-SQL database management system to record the following categories of data of the live production system: ERP data, process parameters, process sensor measurements resulting in the process values to be supervised, and quality control (QC) data.

There are two groups of QC data available at the last stage of production (bonding liner), which we are most interested in (as representing the final product quality). The one with the *flatness* and *RMSE* values is recorded irregularly, because the corresponding QC measurements are done sparsely and manually (2–3 times per day) → batch processing case, as explained in Section 2. The flatness information consists of six different targets containing floating point flatness values, corresponding to the six possible positions on the production tray (called 'nest positions'). The ideal flatness value of a chip is 0 (= the optimal value towards which optimization should be carried out), and the theoretical allowed lower and upper ranges are -100 and 100 to go sure to prevent any customer complaints. RMSEs come with two major important recordings, termed as the RMSE050 and the RMSE095 measures, which both characterize the inner sizes of the chips. Its ideal values are 0.007 with an allowed range between 0 and 0.014.

The second group contains so-called *void events*, which characterize the appearance of several atypical events on chips: a higher number of void events indicates a higher likelihood that a chips is not functioning correctly at customers sites. Thus, the number of void events should be ideally kept close to 0, whereas there is no theoretically allowed upper bound on the number (but more than 100 is already seen as pretty high and risky by company experts). Void events also consist of six different targets containing floating point void events values, corresponding to the six possible positions on the production tray (called 'nest positions').

In order to connect all these QC readings to the respective process phases in previous production stages, the partner company's ERP system was used to identify consecutive production phases (so-called 'lot-batches') on the three stages injection molding, oven, and bonding. By this, the production path of each chip could be reconstructed via the identification numbers of the chip transport boxes. Iterating over each set of flatness, RMSE and void events readings at bonding liner (BL), the lot batch matching and box identifier could be used to find the preceding process measurement time-series of the corresponding injection molding (IE) cycle (= the point of time when the same chip for which the QC was finally available at BL has been produced there).

By this, for the two manually measured QC criteria (flatness and RMSE), finally a set of 182 samples in batch process format (Section 2.1) could be extracted that cover the time span from 2016-10-24 to 2017-04-14. After eliminating missing values in these time-series based data sets, 153 samples were left for modeling (first two thirds) and validation (last third). For the automatically measured void events (in 6 nest positions), in sum 27230 samples could be extracted, which could be down-sampled by a factor of 10 with almost no information loss to 2723 in order to reduce it to a processable size in MATLAB for modeling purposes (still leading to 1.3 Gigabytes in case of time-series lengths of 500 for 63 process values = 31500 columns) .

The process value space as recorded at the injection molding machine (first stage of production) has got 85 dimensions, however 22 of the values are constant during the examined time period. Thus, we ended up with an input space of 63 process values, which were recorded with different granularity over the time span under consideration. The image in Figure 6 gives an overall impression of how the molding process progressed along the examined QC series — thereby, for a couple of (process values based) time-series extracted over time the respective end points are shown; these have been normalized to [0, 1] due to confidentiality reasons.

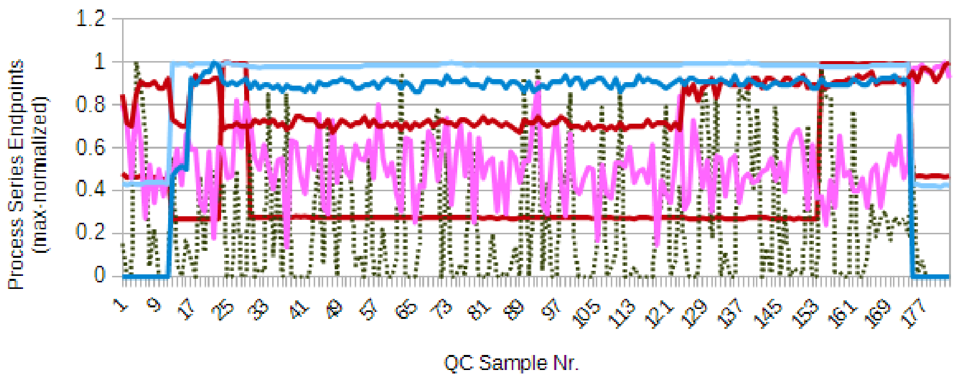


Figure 6: End points of time-series from six process values (appearing as six different lines with different colors); some showing higher fluctuations, some other lower ones indicating different types of process values which our predictive modeling should handle properly.

5.3. *Evaluation Strategies*

for batch off-line model building: on a training set collected during the period of October 2016 to January 2017, a 2-fold cross-validation process was performed for optimizing the parameters of the non-linear PLS fuzzy models. The first dimension of the parameter grid was the number of latent variables used (from 1 to maximal 25) and the second dimension the parameter fac in (4), steering rule evolution versus update (see (4)): fac was varied within 1.2 and 4.0 in steps of 0.35. We used no higher number of folds, because then several test folds would occur in-between training folds, which does not represent a realistic situation for future forecasting (especially with standard 10-fold CV, we could then observe several too optimistic results leading to over-fitting on separate test data).

Furthermore, we also applied a couple of state-of-the-art methods, which can be used for predictive modeling tasks in a time-series based regression context, for comparison purposes. We performed adequate variations of the learning parameters to make the comparison fair with the grid-search procedure used for our PLS-fuzzy modeling. We thereby used the following methods together with the following parameterizations:

- Extreme learning machines [53]: these are flat neural network type models consisting of a single layer but typically a high number of neurons, where these are combined by a linear weighted sum; we varied the number of neurons between 10 and 500 in steps of 10.

- Random forests for regression [54]: regression trees are constructed through bagging by establishing various bags of data (drawing with replacement from the whole data set), building one regression tree per bag according to the CART approach [55] and aggregating the output results of the trees by averaging; we varied the number of trees between 10 and 50 in steps of 2.

- Neural networks employing feed-forward MLPs (multi-layer perceptrons) [56]: one of the most widely used type of NNs employing sigmoids as neural activation functions; we varied the number of neurons between 5 and 50 in steps of 5 per layer and the number of layers from 1 to 4.

As model error criterion we applied the percentual mean absolute error (MAE divided by the range of the target). The final model selection was achieved by an own designed punishment criterion which multiplies the error matrix with the following term:

$$MAE^{(pen)} = MAE \cdot e^{\alpha \ n_{lvs} + \beta \ (4 - fac)} \tag{21}$$

where $\alpha = 0.5$ and $\beta = 0.05$ were determined empirically as the adequate values; the setting with its huge different impact (0.05 versus 0.5) is because we could observe that with higher number of latent variables the 2-fold CV MAE decreased significantly, whereas the non-linearity degree in terms of number of rules played a minor role in over-fitting — note that a lower number of fac leads to more rules, whereas its maximal value in the grid is 4.

A specific modeling phase with all methods was done by including the production delay of chips between the injection moulding and the bonding liner process as additional feature after the time-series data has been transformed by PLS. The is because the delay showed a high variance of values. Thus, in the i-th iteration, we used the current number of latent variables (i.e. the first i ones from the ranked list) plus the production delay, i.e. the matrix $[\vec{x}_{s1} = X * \vec{p}_1, .., \vec{x}_{si} = X * \vec{p}_i, \vec{proddelay}]$, as input for the modeling procedures, with X the regression matrix containing the (slided windowed) time series for all process values. Therefore, we could realize whether the production delay has a significant impact on the final chip quality in the last stage (bonding liner) or not (see results below).

for incremental adaptive model updating: the separate validation data is used as stream for further model updating. The evaluation strategy follows the *interleaved-test-and-then-train* procedure, leading to the *accumulated one-step ahead error/accuracy*. It is a well-known and in data stream community widely used measure based on the idea to measure model performance in one-step ahead update cycles. First, a prediction is made and this is compared with the real target value once it is available. The deviation between this is accumulated over time by simple averaging, thus directly comparable with the batch off-line MAE. We will inspect the accumulated error over time achieved by the dynamically adaptive PLS-fuzzy models (with components for increased flexibility switched on and off) compared to the case when using static models without any updates. Furthermore, we will compare these variants with incremental PLSR, which employs the pure linear PLS regression method and applies a recursive update of the linear regression coefficients (using standard recursive least squares [57]). In all model update runs, we used the optimized training parameters (number of latent variables and fac) from the initial batch off-line modeling and cross-validation phase, such that we received a fair comparison among all runs.

for process optimization: our approach was applied to a non-optimized production process phase, where QC values appeared to be close to or even out of their bounds already from the beginning of the prediction phase with the established forecast models. An example of this real-occurring case during chip production is shown in the left image in Figure 4, where the flatness value was predicted to become around its lower allowed bound of -100 from the beginning on. Similar predicted undesired values could be observed for the void events (more than 80 in most cases) and also the RMSEs were biased significantly from their ideal values (of 0.007). Three QC targets were selected, one from each of these three main groups (flatness, RMSE and void events), as the redundancy among the QC values within one group turned out to be high. The selection was based on the performance of the forecast models (see subsequent section), i.e. those ones where chosen for which the lowest prediction errors were obtained. Based on expert experience and some preliminary tests with static parameter optimization concepts [15], it was already well known that synchronous optimization of our QC targets from these three groups is a hard problem.

For the reduction of the process values to the most significant ones we applied the concepts described in Section 4.2, whose results will be reported in Section 6.3.1. According to several visualizations of time series trends of the various (most influencing) process values, it turned out that basically three features are sufficient to describe the outlook of the trends pretty well, namely mean value, standard deviation and basic slope tendency. Therefore, we used these $s = 3$ features, ending up with a optimization space dimensionality of $3 * p'$ with p' the reduced space obtained from the influence analysis. For the slope tendency, we applied a linear regression model over the whole window of time series and used the slope of this regression model as a feature.

In order to be able to calculate the fitness values for individuals according to Equation (20), i.e. to perform a prediction with the f_i's which are operating on the raw data, the time series signal needs to be reconstructed from the three features used. This is done by the following considerations:

- The mean value represents the core value of the time series trend, thus the reconstructed signal is initialized with a constant line representing the mean value.
- The standard deviation is randomly added to and/or subtracted from each constant line sample, i.e. a $N(0, \sigma)$ random value is produced k times and added to each sample.
- The time-series is "rotated" to match the actual slope over the whole window as encoded in the individual

Numerically, these reconstruction steps are achieved through the following formula:

$$recon(x) = slope * x + mean - \frac{k}{2} * slope + rand(N(0, \sigma)) \qquad \forall x = \{1, ..., k\} \tag{22}$$

with k the window length; $recon(1, ..., k)$ represents the reconstructed time series with length k. Finally, it is not so a matter how accurate this reconstruction is (because the evolutionary-based optimization cycles balance out untypical occurrences leading to low fitness values over consecutive populations, anyway), but more how well the features can be interpreted by humans in order to provide the correct actions for changing the machining parameters: according to discussions with experts of the process, it turned out that especially the values of the mean and the slope already indicate well how to change the parameters, while the whole real (raw data) trends are not so well interpretable, anyway.

Regarding the initialization of the first population, we suggest to use one half of the individuals from the running process (real occurring process values trends) and the second half as uniformly distributed values within the lower and upper bounds of the features: these are the 10% and 90% percentiles from the historic (training) data set in case of the mean, 0 and the maximal standard deviation over the historic (training) data set and the minimal and maximal slope over the (representative) historic (training) data set, which has been also used for model construction. These bounds were also inputted into the multi-objective optimization process as constraints. We used 10 times and 30 times the dimension of the optimization space for the population size and maximal number of iterations, respectively, according to our long-term expertise (and it turned out that these two numbers were not really sensitive).

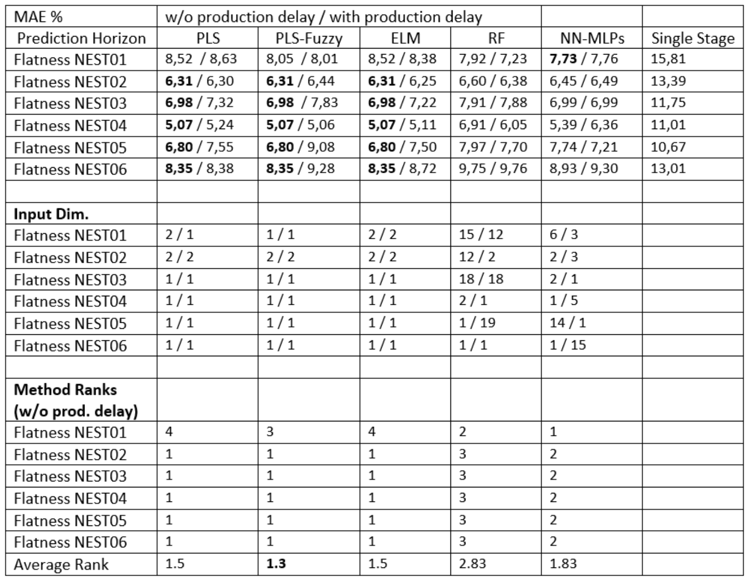
Finally, we compared the achieved final Pareto fronts and associated hyper-volumes [58] — which is a widely used measure for representing the quality of a Pareto front — between fully spanned prediction models containing all process values and reduced models comprising most influencing variables and inducing a reduced optimization space, in order to see the effect of the influence analysis and optimization space reduction on the quality of the final (Pareto) individuals.

6. Results

6.1. Performance of Batch Off-line Forecast Models

Figure 7 shows the mean absolute errors of the batch trained prognostics models with different methodologies, including partial least squares regression PLSR, PLS-Fuzzy (our approach), extreme learning machines (ELMs), random forests (RF) and feed-forward multi-layer perceptron networks (NN-MLPs) up to 4 layers. From immediate glance, it can be recognized that the extra feature in form of the production delay of a chip between IE03 and BL02 stage does not bring any performance improvement, but mostly performs slightly worse — exceptions are three cases (NEST02, NEST04 and NEST05) when applying random forests for regression, and for NEST05 when applying NN-MLPs with feed-forward sigmoid activation functions. This result is in accordance with the expectations of the process experts at the company site.

For all methods, the performance lies significantly below 10% errors, which was a strict requirement by our company partner. The performance of the methods are not deviating so much as expected, but there is tendentially a preference of PLSR, PLS-Fuzzy and ELM over the other methods (except for flatness in NEST01 position where NN-MLPs perform best), as indicated by the MAE numbers in bold font. Surprisingly, these three methods perform more or less equally, except for NEST01 position, while random forests perform significantly worse in 5 out of 6 cases (probably because of over-fitting as can be realized from the large number of inputs it uses — compare with the second part in Table 7 —, and also the optimal number of trees in the forest was pretty high as being close to 50). For the other methods, 1-2 latent variables were optimal to build a reliable prediction model. Thus, the risk of over-fitting on new on-line data is pretty low; furthermore, models with lower complexity usually have a better extrapolation behavior.

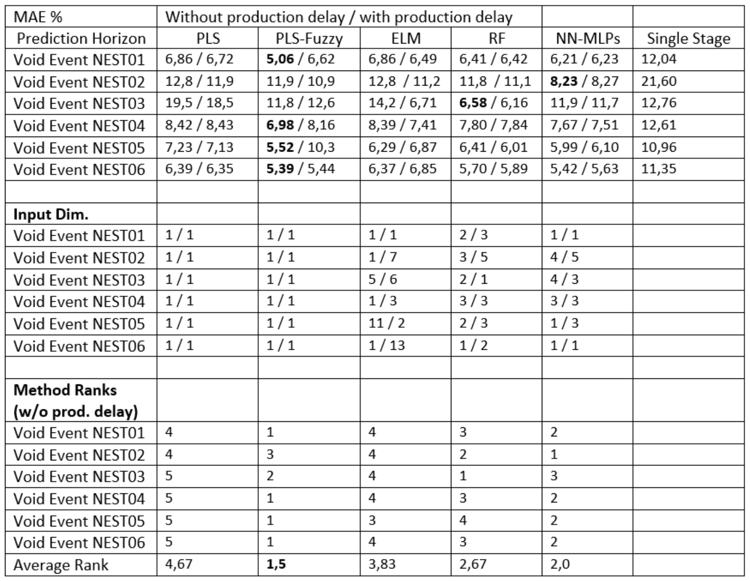


MAE %	w/o production delay / with production delay					
Prediction Horizon	PLS	PLS-Fuzzy	ELM	RF	NN-MLPs	Single Stage
Flatness NEST01	8,52 / 8,63	8,05 / 8,01	8,52 / 8,38	7,92 / 7,23	**7,73** / 7,76	15,81
Flatness NEST02	**6,31** / 6,30	**6,31** / 6,44	**6,31** / 6,25	6,60 / 6,38	6,45 / 6,49	13,39
Flatness NEST03	**6,98** / 7,32	**6,98** / 7,83	**6,98** / 7,22	7,91 / 7,88	6,99 / 6,99	11,75
Flatness NEST04	**5,07** / 5,24	**5,07** / 5,06	**5,07** / 5,11	6,91 / 6,05	5,39 / 6,36	11,01
Flatness NEST05	**6,80** / 7,55	**6,80** / 9,08	**6,80** / 7,50	7,97 / 7,70	7,74 / 7,21	10,67
Flatness NEST06	**8,35** / 8,38	**8,35** / 9,28	**8,35** / 8,72	9,75 / 9,76	8,93 / 9,30	13,01
Input Dim.						
Flatness NEST01	2 / 1	1 / 1	2 / 2	15 / 12	6 / 3	
Flatness NEST02	2 / 2	2 / 2	2 / 2	12 / 2	2 / 3	
Flatness NEST03	1 / 1	1 / 1	1 / 1	18 / 18	2 / 1	
Flatness NEST04	1 / 1	1 / 1	1 / 1	2 / 1	1 / 5	
Flatness NEST05	1 / 1	1 / 1	1 / 1	1 / 19	14 / 1	
Flatness NEST06	1 / 1	1 / 1	1 / 1	1 / 1	1 / 15	
Method Ranks (w/o prod. delay)						
Flatness NEST01	4	3	4	2	1	
Flatness NEST02	1	1	1	3	2	
Flatness NEST03	1	1	1	3	2	
Flatness NEST04	1	1	1	3	2	
Flatness NEST05	1	1	1	3	2	
Flatness NEST06	1	1	1	3	2	
Average Rank	1.5	**1.3**	1.5	2.83	1.83	

Figure 7: Performance in terms of percentage mean absolute error of various prediction modeling methods on separate validation data for the **batch off-line** case **for flatness**, after having optimized the learning parameters and input dimensionality within a cross-validation procedure on the training set; the average method ranks over all 6 QC targets are shown in the lowest row; values before the slashes denote the results when including no production delay, values after the slashes the results when including it.

The fact that PLSR and PLS-Fuzzy perform equally also means that the non-linearity degree contained in the system to be modelled is pretty low. And in fact, as can be seen in Figure 9 below (next section), the optimal number of rules in PLS-Fuzzy (static) case (elicited during the CV phase) was 1 for the NEST positions 03 to 06.

A similar picture could be observed for the RMSEs and the void events (especially regarding the range of the errors and the influence of the production delay). The results of the latter are shown in detail in Figure 8. Here, the error lies again between 5% and 7% for four nest positions, while for NEST02 and NEST03 the error is slightly above 10%. Thus, incrementally updating the models become even more important for void events and we will study in more detail how the error curves behave over time during model updates in the subsequent section. In order to provide a more summarized comparison over the methods, we also show the methods' ranks for each nest position (from 1 to 5, with 1 the best performing model with lowest error, and 5 the worst one with highest error) and the average ranks over all 6 nest positions. Obviously, PLS-fuzzy can outperform the other methods for four nest positions and also on the average rank, and it is never the worst ranked method; this can be achieved with only one latent variable as input, which usually leads to a very robust model with low likelihood of over-fitting on new on-line (test) data. NN-MLPs follow on the second place, but require a much more complex structure than fuzzy models: 3 layers (out of 4) and 40 neurons (out of 50) in each layer was found as the best choice. The PLS-fuzzy systems only required 3 to 10 rules,



MAE %	Without production delay / with production delay					
Prediction Horizon	PLS	PLS-Fuzzy	ELM	RF	NN-MLPs	Single Stage
Void Event NEST01	6,86 / 6,72	**5,06** / 6,62	6,86 / 6,49	6,41 / 6,42	6,21 / 6,23	12,04
Void Event NEST02	12,8 / 11,9	11,9 / 10,9	12,8 / 11,2	11,8 / 11,1	**8,23** / 8,27	21,60
Void Event NEST03	19,5 / 18,5	11,8 / 12,6	14,2 / 6,71	**6,58** / 6,16	11,9 / 11,7	12,76
Void Event NEST04	8,42 / 8,43	**6,98** / 8,16	8,39 / 7,41	7,80 / 7,84	7,67 / 7,51	12,61
Void Event NEST05	7,23 / 7,13	**5,52** / 10,3	6,29 / 6,87	6,41 / 6,01	5,99 / 6,10	10,96
Void Event NEST06	6,39 / 6,35	**5,39** / 5,44	6,37 / 6,85	5,70 / 5,89	5,42 / 5,63	11,35
Input Dim.						
Void Event NEST01	1 / 1	1 / 1	1 / 1	2 / 3	1 / 1	
Void Event NEST02	1 / 1	1 / 1	1 / 7	3 / 5	4 / 5	
Void Event NEST03	1 / 1	1 / 1	5 / 6	2 / 1	4 / 3	
Void Event NEST04	1 / 1	1 / 1	1 / 3	3 / 3	3 / 3	
Void Event NEST05	1 / 1	1 / 1	11 / 2	2 / 3	1 / 3	
Void Event NEST06	1 / 1	1 / 1	1 / 13	1 / 2	1 / 1	
Method Ranks (w/o prod. delay)						
Void Event NEST01	4	1	4	3	2	
Void Event NEST02	4	3	4	2	1	
Void Event NEST03	5	2	4	1	3	
Void Event NEST04	5	1	4	3	2	
Void Event NEST05	5	1	3	4	2	
Void Event NEST06	5	1	4	3	2	
Average Rank	4,67	**1,5**	3,83	2,67	2,0	

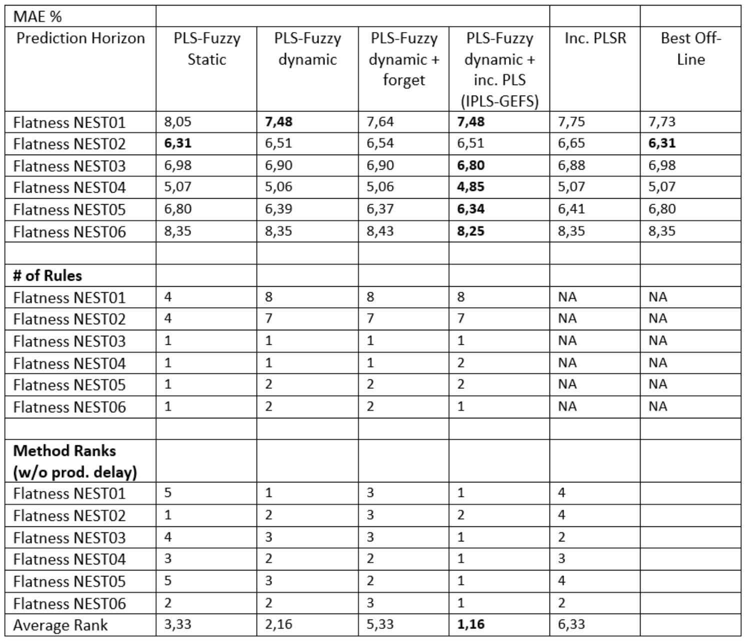
Figure 8: Performance in terms of percentage mean absolute error of various prediction modeling methods on separate validation data for the **batch off-line** case for **void events**, after having optimized the learning parameters and input dimensionality within a cross-validation procedure on the training set; the average method ranks over all 6 QC targets are shown in the lowest row; values before the slashes denote the results when including no production delay, values after the slashes the results when including it.

which still can be seen as a compact and easily interpretable model having a low likelihood to over-fit on new data.

A final note goes to the last column in both tables above, which shows the (optimized and best achievable) prediction errors within a single stage process, i.e. by using the process values trends directly at the bonding liner to predict the final quality after the bonding process. The prediction horizon there reduces to a few hours, but the error is even significantly higher than in the case of multi-stage forecasting, lying above 10% for all nest positions in case of both, flatness and void events of the chips. Upon feedback with the company experts, they judged this as a consistent result, especially the flatness of the chips is usually already determined during/after the injection molding process, whereas the bonding process should have (physically) no influence on it.

6.2. Performance of Self-Adaptive Forecast Models

In a next step, we tried to further improve the accuracies of the fuzzy and PLSR forecast models by evolving them on the separate validation data with the usage of the methodologies described throughout Section 3. We therefore used the same learning parameters as elicited during the CV process on the training set. Figure 9 shows the mean absolute errors achieved by always predicting the next QC value and comparing it with the observed one (stored in the data streaming set) before using the observed one for model adaptation. From immediate glance, it is obvious that the dynamic update of the PLS-Fuzzy models with conventional evolution (Column #3, omitting the dashed components



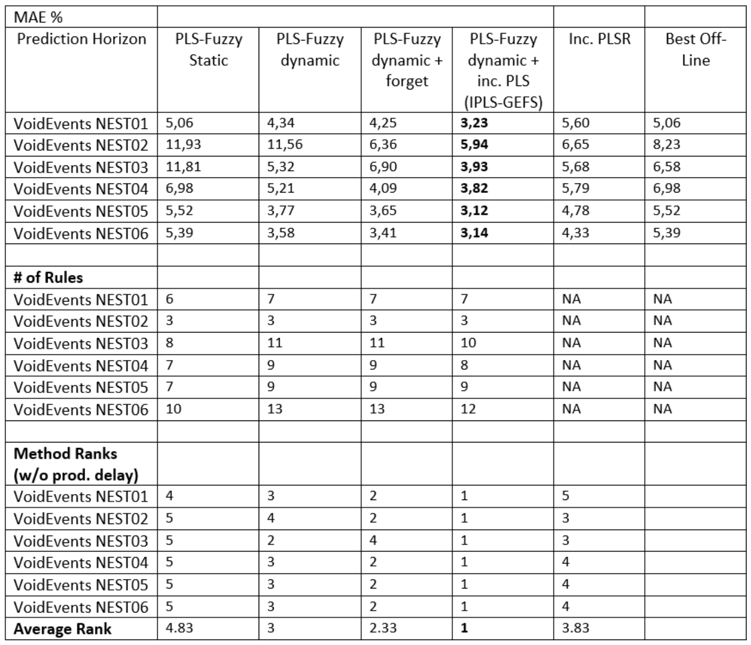
MAE %						
Prediction Horizon	PLS-Fuzzy Static	PLS-Fuzzy dynamic	PLS-Fuzzy dynamic + forget	PLS-Fuzzy dynamic + inc. PLS (IPLS-GEFS)	Inc. PLSR	Best Off-Line
Flatness NEST01	8,05	**7,48**	7,64	**7,48**	7,75	7,73
Flatness NEST02	**6,31**	6,51	6,54	6,51	6,65	**6,31**
Flatness NEST03	6,98	6,90	6,90	**6,80**	6,88	6,98
Flatness NEST04	5,07	5,06	5,06	**4,85**	5,07	5,07
Flatness NEST05	6,80	6,39	6,37	**6,34**	6,41	6,80
Flatness NEST06	8,35	8,35	8,43	**8,25**	8,35	8,35
# of Rules						
Flatness NEST01	4	8	8	8	NA	NA
Flatness NEST02	4	7	7	7	NA	NA
Flatness NEST03	1	1	1	1	NA	NA
Flatness NEST04	1	1	1	2	NA	NA
Flatness NEST05	1	2	2	2	NA	NA
Flatness NEST06	1	2	2	1	NA	NA
Method Ranks (w/o prod. delay)						
Flatness NEST01	5	1	3	1	4	
Flatness NEST02	1	2	3	2	4	
Flatness NEST03	4	3	3	1	2	
Flatness NEST04	3	2	2	1	3	
Flatness NEST05	5	3	2	1	4	
Flatness NEST06	2	2	3	1	2	
Average Rank	3,33	2,16	5,33	**1,16**	6,33	

Figure 9: Performance in terms of percentage mean absolute error of various prediction modeling methods on separate validation data for the **adaptive on-line** case for **flatness**, after having optimized the learning parameters and input dimensionality within a cross-validation procedure on the initial training set; the average method ranks over all 6 QC targets are shown in the lowest row, the best result(s) for each quality criterion in bold font; NA denotes "not available" (occurring in the case when no rules were generated by the method).

for increased flexibility in SAFM-IF according to Figure 3) can decrease the errors in 5 out of 6 cases (except for NEST02 position) and that an additional update of the latent variable space (Column #5) can further decrease the errors, finally ending up with best errors in 5 out of 6 cases. Also a surprise was that, by increasing the flexibility of the rule base update through forgetting, the errors tendentially got worse — an explanation could be that the forgetting factor was set too pessimistically by our automatic setting approach, such that a too strong forgetting was triggered, which finally lead to a loss of the convergence of the parameters.

The decrease of errors sometimes goes hand in hand with the evolution of additional rules as can be seen in the lower part of the table in Figure 9. This shows the number of rules of the fuzzy systems at the end of the stream adaptation phase. For instance, it is interesting to see that in the case of NEST04 the usage of incremental PLS Space update in combination with PLS-Fuzzy (Column #5) triggers the evolution/splitting of a rule (2 rules at the end), which is not the case when using the conventional adaptation (Column #3, only 1 rule at the end). Obviously, some rotational effect due to the update of the PLS space induced the evolution of an additional rule, which in turn had a positive effect on the error (as it decreased from 5.06% to 4.85%).

For the void events we achieved a similar picture as for the flatness — as shown in Figure 10: the most dynamic



MAE %						
Prediction Horizon	PLS-Fuzzy Static	PLS-Fuzzy dynamic	PLS-Fuzzy dynamic + forget	PLS-Fuzzy dynamic + inc. PLS (IPLS-GEFS)	Inc. PLSR	Best Off-Line
VoidEvents NEST01	5,06	4,34	4,25	**3,23**	5,60	5,06
VoidEvents NEST02	11,93	11,56	6,36	**5,94**	6,65	8,23
VoidEvents NEST03	11,81	5,32	6,90	**3,93**	5,68	6,58
VoidEvents NEST04	6,98	5,21	4,09	**3,82**	5,79	6,98
VoidEvents NEST05	5,52	3,77	3,65	**3,12**	4,78	5,52
VoidEvents NEST06	5,39	3,58	3,41	**3,14**	4,33	5,39
# of Rules						
VoidEvents NEST01	6	7	7	7	NA	NA
VoidEvents NEST02	3	3	3	3	NA	NA
VoidEvents NEST03	8	11	11	10	NA	NA
VoidEvents NEST04	7	9	9	8	NA	NA
VoidEvents NEST05	7	9	9	9	NA	NA
VoidEvents NEST06	10	13	13	12	NA	NA
Method Ranks (w/o prod. delay)						
VoidEvents NEST01	4	3	2	1	5	
VoidEvents NEST02	5	4	2	1	3	
VoidEvents NEST03	5	2	4	1	3	
VoidEvents NEST04	5	3	2	1	4	
VoidEvents NEST05	5	3	2	1	4	
VoidEvents NEST06	5	3	2	1	4	
Average Rank	4.83	3	2.33	**1**	3.83	

Figure 10: Performance in terms of percentage mean absolute error of various prediction modeling methods on separate validation data for the **adaptive on-line** case for **void events**, after having optimized the learning parameters and input dimensionality (from a ranked list of inputs) within a cross-validation procedure on the initial training set; the average method ranks over all 6 QC targets are shown in the lowest row, the best result(s) for each quality criterion in bold font; NA denotes "not available" (occurring in the case when no rules were generated by the method).

update variant including incremental PLS space updating (termed as IPLS-GEFS = full SAFM-IF approach using all components as shown in Figure 3) achieved the best performance for all nest positions (thus its average rank equal to 1, as shown in the last row), whereas the sole forgetting option (without incremental PLS space) achieved always the second best result, except in one case. Here, the error could be much more reduced than in case of flatness: especially, for the two problematic cases (NEST02 ad NEST03), the error could be significantly decreased down to 5% and even lower, to become lying well below the upper bounds of 10%. Rule evolution took place in most of the cases, where, due to incremental PLS space updating, the number of rules evolved could be a bit reduced.

Furthermore, we provided a closer look on the performance behavior of the methods over time during the adaptation mode, i.e. the error trend lines on a sample per sample basis. For the six nest positions of void events (we have chosen this quality criterion because we had the longest stream available for it), this is visualized in separate subfigures in Figure 11, (a) to (f). From these images, it can be clearly seen that the IPLS-GEFS approach with flexible latent variable space (= full SAFM-IF approach using all components as shown in Figure 3) can outperform the other methods in all six nest positions, as producing significantly lower error trends (shown as dark solid lines) appearing below all the other trend lines (shown as lighter dashed lines). The adaptation variant with forgetting (but without incremental PLS) comes close to IPLS-GEFS for some nest positions, whereas static models produce high errors and even completely fail to produce meaningful predictions towards the end of the stream for three nest positions: a significantly rising error curve can be observed. This negative effect could be compensated by the adaptive variants due to the evolution of a new rule, as could be observed based on the rule number trend line over time. Incremental PLSR, the linear version of IPLS-GEFS, performed mostly slightly worse than the conventional adaptation of the PLS-fuzzy models (without forgetting and IPLS).

Finally, we had a closer look at the observed versus predicted QC measurements, in order to see how well (the predictions from the) static and dynamic models can 'track' the dynamics of the process, which is also reflected in pretty high dynamic void events values. Figure 12 compares exemplarily the predicted trends (dark dashed lines) with the real observed ones (grey dashed lines), left for static models, right for dynamic models. It is interesting to see that in both cases the high fluctuations with peaking values are ignored, but the basic upwards and downwards trends of the void events are pretty nicely tracked by the predictions over the whole time frame. This means that our prediction model also smoothes the void event values to some extent and concentrates to follow the basic trends over time. Thus, it is also able to correctly recognize drifts or significant downtrends; this turned out to be sufficient for the experts/operators at the production site to realize problems at an early stage. It is also interesting to see that the dynamic adaptive models (right image) achieve a better tracking of the real trend than the static models, especially between Samples 300 and 500, but also towards the end of the stream, where static models become significantly biased. Finally, someone can observe that void event values appeared to lie above 45-50 over the complete time frame of the on-line prediction and test phase, whereas the ideal value is 0. This means that a non-ideal process phase could be recognized by our prediction models; such an occurrence within the same time frame was also recognized due to the predicted flatness values trends with values lying inbetween -70 and -120 for all nest positions, far from being optimal at 0. This observation was the trigger and starting point for a process optimization run, whose results will be discussed in the subsequent section.

6.3. *Performance of Process Optimization*

6.3.1. *Influence Analysis on Process Values for QC Criteria*

Before starting the real optimization process, we performed an influence analysis in order to reduce the very high-dimensional raw data feature space ($63 * 50 = 3150$) to a realistic number. By extracting the three features as discussed in Section 5.3 (mean, standard deviation and slope), the feature space is indeed reduced to $63 * 3 = 189$, but still pretty high for being able to assure convergence within a reasonable amount of time (and the operator should not wait too long for proper reactions when an undesired production phase is present), see also the computation times below. According to a correlation analysis, where we found out high correlations among the nest positions within void events and flatness groups, we therefore concentrated on one prediction model for each group of quality criteria (flatness, void events and RMSEs), namely on that one with the lowest prediction error achieved — as demonstrated in the previous section —, and applied the criterion in (17) over the three selected models to find out the most influencing, most important process values for predicting well the corresponding targets (quality criteria).

An example of an influence plot for flatness is provided in Figure 13, where the different colors (gray levels) represent the influence weights of 63 different process values at the injection molding machine for flatness of the chips at the bonding liner. Each process value is represented by 50 consecutive points (representing the time-series samples over a window with size 50), which could have a bit varying influence in some cases — however, mostly these stay in close local range (especially for the higher influencing ones), such that a cut-off for gathering the most important process values is possible. This cut-off denoted as *significance threshold* is given by Equation (17) and shown as horizontal line in Figure 13. This leaves 7 significantly influencing process values for flatness (5 on the right side in red/pink/violet colors, two in the middle with turquoise and blue colors). By also conducting this for the prediction models for RMSE and void events, we finally ended up with 13 essentially influencing process values. Thus, our optimization space remarkably reduced from 189 to $13 * 3 = 39$ dimensions. We also performed an influence analysis over all 14 models (6 for flatness, 6 for void events, 2 for RMSEs) and then only 4 additional process values turned out to be significant. This also confirmed the high redundancy among the QCs within each group.

6.3.2. Performance of Multi-Objective Evolutionary Algorithms for Finding Optimized Process Values Trends

We used the parameter settings and initialized the population as discussed in Section 5.3. We performed two different runs, i) one with the reduced space based on the most influencing process values (and features extracted from these) and ii) one with the full space based on all process values in order to observe the effect of the reduction onto convergence, Pareto front quality and computation speed of NSGA-II. The resulting Pareto fronts after the termination of NSGA-II according to the maximal number of populations ($30 * dimspace$) are visualized in Figure 14, the upper row represents the full space, the lower row the reduced space. In both rows, the left image shows the two-dimensional Pareto front on flatness versus RMSE, the right image shows the three-dimensional Pareto front on all three criteria together. The pentagrams are the final Pareto front individuals, the dark dots represent the fitness of the individuals in the initial population.

It becomes immediately clear that the reduced space leads to significantly improved results, as the full space completely fails to optimize flatness and RMSE synchronously, achieving a really bad flatness fitness on flatness of a bit above 50 (where 0 is the optimum!). For the reduced space, the individuals converge much closer to the origin, achieving individuals with ideal flatness values (0 fitness) and almost ideal RMSE values (0.001); the best individual lying closest to the origin is marked as such in the three-dimensional plots. Its distance to the origin marked by a line is around 0.2 in the reduced space, but around 70 in the full space, a remarkable difference. Furthermore, the computation time in MATLAB on a single PC for the whole optimization process run was **23 hours** for the full space, while it was only about **50 minutes** for the reduced space (where a close-optimal convergence already resulted after 15 minutes, see Figure 15). Thus, only in the latter case an answer in form of improved suggested process values can be given to an operator within a reasonable amount of time.

In both cases, the multi-objective optimization process (with all the genetic and selection operators embedded) seems to work pretty fine, as the initial population is largely improved, as it is largely shifted towards the origin of the Pareto space. As the time-series based forecast models established on the full space (and used as surrogates in the evolutionary optimization process) showed an accurate prediction behavior on new on-line process data for all QCs (as verified in the previous subsections), the only explanation is that the optimization process itself is severely affected by the curse of dimensionality the full space induces to find proper solutions (189 parameters to tune!). Figure 15 underlines this analysis by showing nicely converging hyper-volumes as quality indicators of the Pareto fronts [58] for both scenarios (thus, the optimization process is working), however with significantly higher converging values (mind the y-axis scale) in the case of the reduced space (right image). Also, it is easy to see that in the case of the reduced space the hyper-volume is already 'saturated' after 300, 400 iterations, which could be used for terminating the process earlier and thus to even improve the computation speed and reaction time for operators (by more than two thirds in this case, so from 50 minutes down to around 15 minutes).

Finally, the individual which is closest to the origin in the three-dimensional Pareto space can be used for suggesting improved process values (trends) at the injection molding machine to operators and/or to a parameter space mapping engine for interpretation and for conducting appropriate changes in the machining parameters. As an example, in the case of the reduced space, the following individual turned out to be the closest to the origin:

means for 13 process values: $[247.07, 27.29, 26.04, 0.46, 2.51, 925.11, 7.51, 5.00, 79.98, 64.92, 64.71, 68.94, 68.98]$
standard deviations for 13 process values: $[1.68, 0.39, 1.01, 0.05, 0.17, 0.29, 0.12, 0.87, 0.36, 0.10, 0.49, 0.61, 0.71]$
slope trends for 13 process values: $[-0.06, 0.05, -0.29, 0.08, -0.26, 0.21, 0.12, 0.43, 0.14, -0.38, -0.14, -0.32, 0.29]$

This can now be interpreted that the slopes in absolute values are pretty low for all process values. This means that a nearly constantly trending behavior of the (most influencing) process values at injection molding is desirable in order to obtain good QC values at the end of the bonding lining process. The standard deviations are also pretty small (< 1 in all but one case), thus the process values should underlie small fluctuations around their ideal mean values as listed above. Thus, the mean values finally count as most significant to achieve good QC values. For achieving these values (in average), either the operators/experts of the production site know how to set the corresponding machining parameters (as in our application case), achieving an adequate reaction manually to improve the QC values, or a back-mapping model between (re-constructed) process values trends and process parameters could be established before based on historic data.

7. Conclusion and Future Work

We suggested new methodologies for building time-series based forecast models in a multi-stage production process, i.e. predicting final production quality at an early stage of production. They include i) an appropriate data pre-processing step to resolve batch process data, ii) feature space transformation to reduce curse of dimensionality of the very high-dimensional data usually containing only a moderate number of samples, iii) non-linear (fuzzy) model training in an iterative manner with error convergence and iv) an automatic self-adaptive algorithm for model updating to increase flexibility and robustness of the prediction models. The latter could be verified based on real-world data from an on-line chip production process, where significantly decreasing error trends over time could be observed. Finally, these models serve i) as engine for early problem recognition as they can 'foresee' possible downtrends in (final, latter) product quality already at an early stage and ii) as surrogate models for optimizing process values trends and associated (machining) parameters. Convergence and quality of the Pareto fronts showed remarkable results coming close to their ideal values (according to pre-defined quality standards), especially in case when the curse of dimensionality in the optimization space is significantly reduced in advance by a model-based influence analysis concept. In this sense, our holistic approach as visualized in Figure 1, which combines early prediction of problems with proper reactions, was successfully evaluated and thus can be seen as a promising methodological framework for successfully conducting (early) predictive maintenance in production and manufacturing systems.

Future work include i) the direct optimization of process parameters based on static mappings between these parameters and quality criteria, which also requires the usage of a design of experiments strategy to perform machining tests for specific parameter settings with high efficiency and low costs (work in progress) $\rightarrow$ this should achieve pre-optimized settings in advance, and ii) a back-mapping model from process values trends to process parameters, whose challenge is to solve a multiple regression/prediction problem based on time-series data.

Acknowledgements

The authors acknowledge the Austrian research funding association (FFG) within the scope of the 'IKT of the future' programme, project 'Generating process feedback from heterogeneous data sources in quality control' (contract # 849962). The first author acknowledges the support by the "LCM — K2 Center for Symbiotic Mechatronics" within the framework of the Austrian COMET-K2 program.

References

[1] J. Levitt, *Complete Guide to Preventive and Predictive Maintenance*. New York: Industrial Press Inc., 2011.

[2] S. Aumi, B. Corbett, and P. Mhaskary, "Model predictive quality control of batch processes," in *2012 American Control Conference*, Fairmont Queen Elizabeth, Montral, Canada, 2012, pp. 5646–5651.

[3] R. Mobley, *An Introduction to Predictive Maintenance — Second Edition*. Woburn, Massachussetts, U.S.A.: Elsevier Science, 2002.

[4] P. Renna, “Influence of maintenance policies on multi-stage manufacturing systems in dynamic conditions,” *International Journal of Production Research*, vol. 50, no. 2, pp. 345–357, 2011.

[5] G. Box, G. Jenkins, and G. Reinsel, *Time Series Analysis, Forecasting and Control*. Engelwood Cliffs, New Jersey: Prentice Hall, 1994.

[6] S. Ekwaro-Osire, A. Gonalves, and F. Alemayehu, *Probabilistic Prognostics and Health Management of Energy Systems*. New York: Springer, 2017.

[7] D. Acuna, M. Orchard, and R. Saona, “Conditional predictive bayesian cramer-rao lower bounds for prognostic algorithms design,” *Applied Soft Computing*, vol. https://doi.org/10.1016/j.asoc.2018.01.033, 2018.

[8] C. Park, D. Moon, N. Do, and S. Bae, “A predictive maintenance approach based on real-time internal parameter monitoring,” *The International Journal of Advanced Manufacturing Technology*, vol. 85, no. 1, pp. 623–632, 2016.

[9] S. Zhang, R. Dubay, and M. Charest, “A principal component analysis model-based predictive controller for controlling part warpage in plastic injection molding,” *Expert Systems with Applications*, vol. 42, no. 6, pp. 2919–2927, 2015.

[10] M. Kano and Y. Nakagawa, “Data-based process monitoring, process control, and quality improvement: Recent developments and applications in steel industry,” *Computers and Chemical Engineering*, vol. 32, pp. 12–24, 2008.

[11] L. Wang and R. Gao, *Condition Monitoring and Control for Intelligent Manufacturing*. London, UK: Springer Verlag, 2006.

[12] K. Chockalingam, N. Jawahar, K. Ramanathan, and P. Banerjee, “Optimization of stereolithography process parameters for part strength using design of experiments,” *The International Journal of Advanced Manufacturing Technology*, vol. 29, no. 1, pp. 79–88, 2006.

[13] S. Gu, J. Ren, and G. Vancso, “Process optimization and empirical modeling for electrospun polyacrylonitrile (pan) nanofiber precursor of carbon nanofibers,” *European polymer journal*, vol. 41, no. 11, pp. 2559–2568, 2005.

[14] E. Permin, F. Bertelsmeier, M. Blum, J. Bützler, S. Haag, S. Kuz, D. Özdemir, S. Stemmler, U. Thombansen, R. Schmitt, C. Brecher, C. Schlick, D. Abel, R. Popraw, P. Loosen, W. Schulz, and G. Schuh, “Self-optimizing production systems,” *Procedia CIRP*, vol. 41, pp. 417–422, 2016.

[15] A.-C. Zavoianu, E. Lughofer, R. Pollak, P. Meyer-Heye, C. Eitzinger, and T. Radauer, "Multi-objective knowledge-based strategy for process parameter optimization in micro-fluidic chip production," in *Proceedings of the SSCI 2017 Conference (CIES Workshop)*, Honolulu, Hawaii, 2017, pp. 1927–1934.

[16] M. Haenlein and A. Kaplan, "A beginner's guide to partial least squares (PLS) analysis," *Understanding Statistics*, vol. 3, no. 4, pp. 283–297, 2004.

[17] K. Varmuza and P. Filzmoser, *Introduction to Multivariate Statistical Analysis in Chemometrics*. Boca Raton: CRC Press, 2009.

[18] J. J. Rubio, E. Garcia, G. Aquino, C. A. Ibanez, J. Pacheco, and A. Zacarias, "Learning of operator hand movements via least angle regression to be teached in a manipulator," *Evolving Systems*, vol. DOI: 10.1007/s12530-018-9224-1, 2018.

[19] J. J. Rubio, E. Garcia, G. Aquino, C. A. Ibanez, J. Pacheco, and J. A. Meda-Campana, "Recursive least squares for a manipulator which learns by demonstration," *Revista Iberoamericana de Automtica e Informtica Industrial*, vol. https://doi.org/10.4995/riai.2018.8899, 2018.

[20] R. Rosipal, "Kernel partial least squares for nonlinear regression and discrimination," *Neural Network World*, vol. 13, no. 3, pp. 291–300, 2003.

[21] W. Pedrycz and F. Gomide, *Fuzzy Systems Engineering: Toward Human-Centric Computing*. Hoboken, New Jersey: John Wiley & Sons, 2007.

[22] J. Abonyi, *Fuzzy Model Identification for Control*. Boston, U.S.A.: Birkhäuser, 2003.

[23] E. Lughofer, *Evolving Fuzzy Systems — Methodologies, Advanced Concepts and Applications*. Berlin Heidelberg: Springer, 2011.

[24] A. Lemos, W. Caminhas, and F. Gomide, "Multivariable gaussian evolving fuzzy modeling system," *IEEE Transactions on Fuzzy Systems*, vol. 19, no. 1, pp. 91–104, 2011.

[25] E. Lughofer, C. Cernuda, S. Kindermann, and M. Pratama, "Generalized smart evolving fuzzy systems," *Evolving Systems*, vol. 6, no. 4, pp. 269–292, 2015.

[26] D. Dovzan, V. Logar, and I. Skrjanc, "Implementation of an evolving fuzzy model (eFuMo) in a monitoring system for a waste-water treatment process," *IEEE Transactions on Fuzzy Systems*, vol. 23, no. 5, pp. 1761–1776, 2015.

[27] K. Tabata and M. S. M. Kudo, "Data compression by volume prototypes for streaming data," *Pattern Recognition*, vol. 43, no. 9, pp. 3162–3176, 2010.

[28] E. Lughofer and M. Sayed-Mouchaweh, “Autonomous data stream clustering implementing incremental split-and-merge techniques — towards a plug-and-play approach,” *Information Sciences*, vol. 204, pp. 54–79, 2015.

[29] H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” *Journal of the Royal Statistical Society, Series B*, pp. 301–320, 2005.

[30] E. Lughofer, “On-line assurance of interpretability criteria in evolving fuzzy systems — achievements, new concepts and open issues,” *Information Sciences*, vol. 251, pp. 22–46, 2013.

[31] R. Nikzad-Langerodi, E. Lughofer, C. Cernuda, T. Reischer, W. Kantner, M. Pawliczek, and M. Brandstetter, “Calibration model maintenance in melamine resin production: Integrating drift detection, smart sample selection and model adaptation,” *Analytica Chimica Acta*, vol. 1013, pp. 1–12 (featured article), 2018.

[32] A. Shaker and E. Lughofer, “Self-adaptive and local strategies for a smooth treatment of drifts in data streams,” *Evolving Systems*, vol. 5, no. 4, pp. 239–257, 2014.

[33] E. Lughofer and P. Angelov, “Handling drifts and shifts in on-line data streams with evolving fuzzy systems,” *Applied Soft Computing*, vol. 11, no. 2, pp. 2057–2068, 2011.

[34] E. Lughofer, M. Pratama, and I. Skrjanc, “Incremental rule splitting in generalized evolving fuzzy systems for autonomous drift compensation,” *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 4, pp. 1854–1865, 2018.

[35] H. Mouss, D. Mouss, N. Mouss, and L. Sefouhi, “Test of Page-Hinkley, an approach for fault detection in an agro-alimentary production system,” in *Proceedings of the Asian Control Conference, Volume 2*, 2004, pp. 815–818.

[36] E. Lughofer, E. Weigl, W. Heidl, C. Eitzinger, and T. Radauer, “Recognizing input space and target concept drifts with scarcely labelled and unlabelled instances,” *Information Sciences*, vol. 355–356, pp. 127–151, 2016.

[37] X.-Q. Zeng and G.-Z. Li, “Incremental partial least squares analysis of big streaming data,” *Pattern Recognition*, vol. 47, pp. 3726–3735, 2014.

[38] S. Wold, M. Sjöström, and L. Eriksson, “PLS-regression: a basic tool of chemometrics,” *Chemometrics and Intelligent Laboratory Systems*, vol. 58, pp. 109–130, 2001.

[39] I. Dramnesc and T. Jebelean, “Synthesis of list algorithms by mechanical proving,” *Journal of Symbolic Computation*, vol. 69, pp. 61–92, 2015.

[40] A. Bouchachia and R. Mittermeir, “Towards incremental fuzzy classifiers,” *Soft Computing*, vol. 11, no. 2, pp. 193–207, 2006.

[41] J. Weng, Y. Zhang, and W.-S. Hwang, "Candid covariance-free incremental principal component analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 1034–1040, 2003.

[42] M. Affenzeller, S. Winkler, S. Wagner, and A. Beham, *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. Boca Raton, Florida: Chapman & Hall, 2009.

[43] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin Heidelberg New York: Springer Verlag, 2003.

[44] K. Miettinen, *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, 1999.

[45] I. Das and J. Dennis, "Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems," *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 631–657, 1998.

[46] A. Messac, A. Ismail-Yahaya, and C. Mattson, "The normalized normal constraint method for generating the Pareto frontier," *Structural and Multidisciplinary Optimization*, vol. 25, no. 2, pp. 86–98, 2003.

[47] D. Dasgupta and Z. Michalewicz, *Evolutionary algorithms in engineering applications*. Heidelberg Berlin: Springer, 1997.

[48] C. C. Coello and G. Lamont, *Applications of multi-objective evolutionary algorithms*. Singapore: World Scientific, 2004.

[49] K. D. Jong, *Evolutionary computation: a unified approach*. New York: MIT Press, 2006.

[50] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[51] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization," in *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)*. International Center for Numerical Methods in Engineering (CIMNE), 2002, pp. 95–100.

[52] E. Lughofer, E. Weigl, W. Heidl, C. Eitzinger, and T. Radauer, "Integrating new classes on the fly in evolving fuzzy classifier designs and its application in visual inspection," *Applied Soft Computing*, vol. 35, pp. 558–582, 2015.

[53] F. Sun, K.-A. Toh, M. Romay, and K. Mao, *Extreme Learning Machines 2013: Algorithms and Applications (Adaptation, Learning, and Optimization)*. Heidelberg, New York: Springer, 2013.

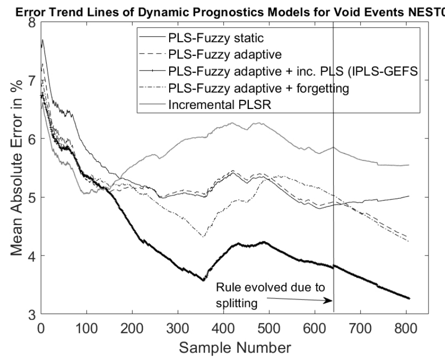
[54] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[55] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees*. Boca Raton: Chapman and Hall, 1993.

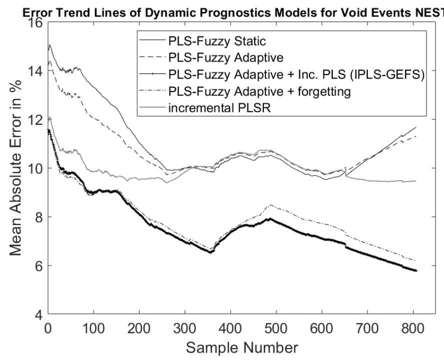
[56] S. Haykin, *Neural Networks: A Comprehensive Foundation (2nd Edition)*. Upper Saddle River, New Jersey: Prentice Hall Inc., 1999.

[57] L. Ljung, *System Identification: Theory for the User*. Upper Saddle River, New Jersey: Prentice Hall PTR, Prentic Hall Inc., 1999.

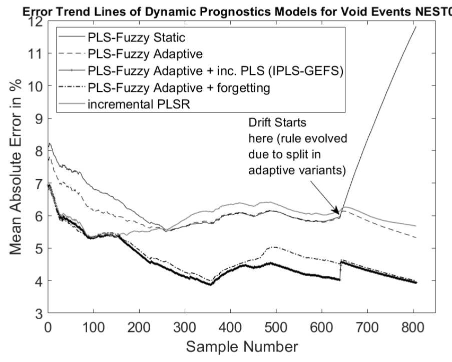
[58] E. Zitzler, "Evolutionary algorithms for multiobjective optimization: Methods and applications," Ph.D. dissertation, Swiss Federal Institute of Technology, 1999, bern, Switzerland.



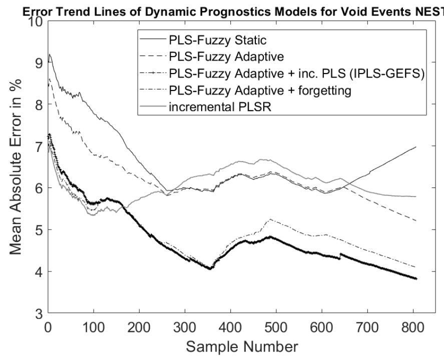
(a)



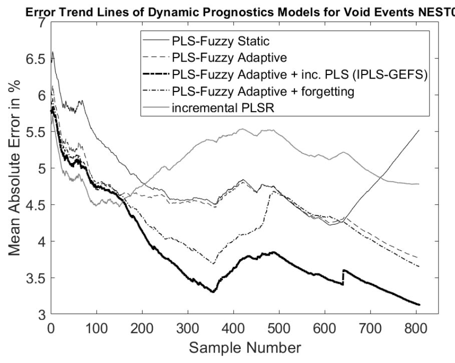
(b)



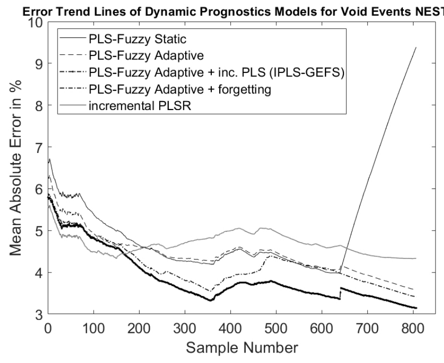
(c)



(d)



(e)



(f)

Figure 11: Prediction error trend lines over time for void events when applying static and different variants of dynamically adaptive models as indicated in the legend of the figures and represented by different line styles and colors; (a) to (f) show the trend lines for the six different nest positions (1 to 6).

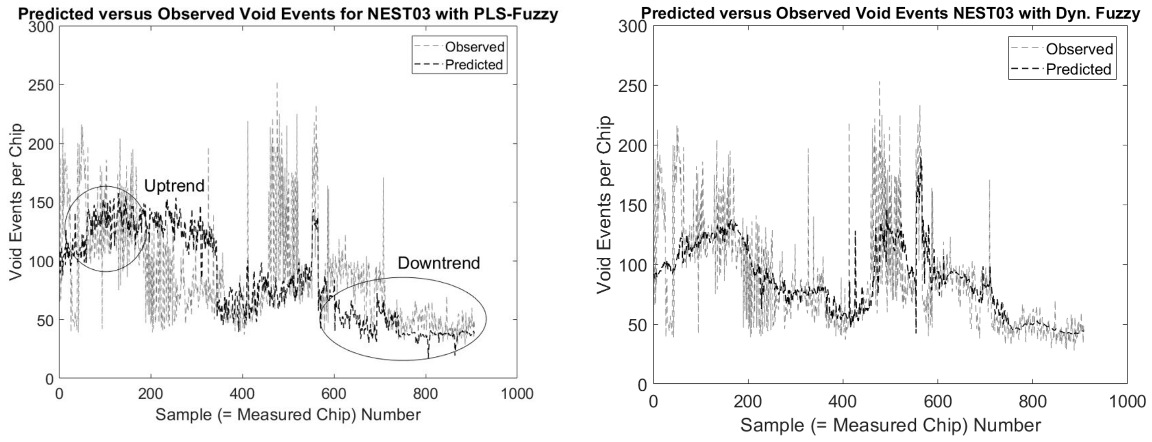


Figure 12: Comparison between at injection molding predicted (dark dashed lines) and at bonding liner de facto observed (grey dashed lines) void events values for nest position 03; the basic trends are nicely tracked, whereas the dynamic models (right image) achieve an even better tracking at around Samples 300 and 500.

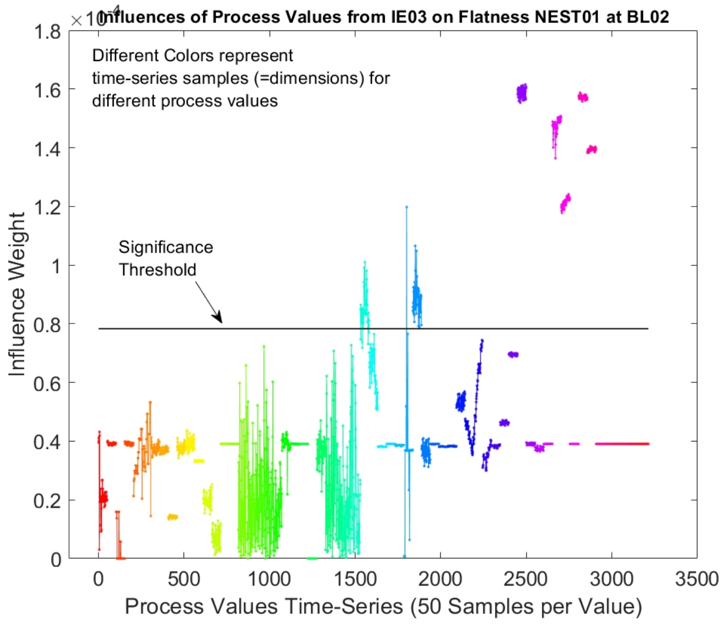


Figure 13: Influence weights of 63 process values (each containing 50 points for the 50 time-series samples) at injection molding on flatness criterion at the bonding liner, the horizontal line represent the significance threshold: all process values appearing largely above this line are seen as significantly influencing ones and are used in the process optimization run.

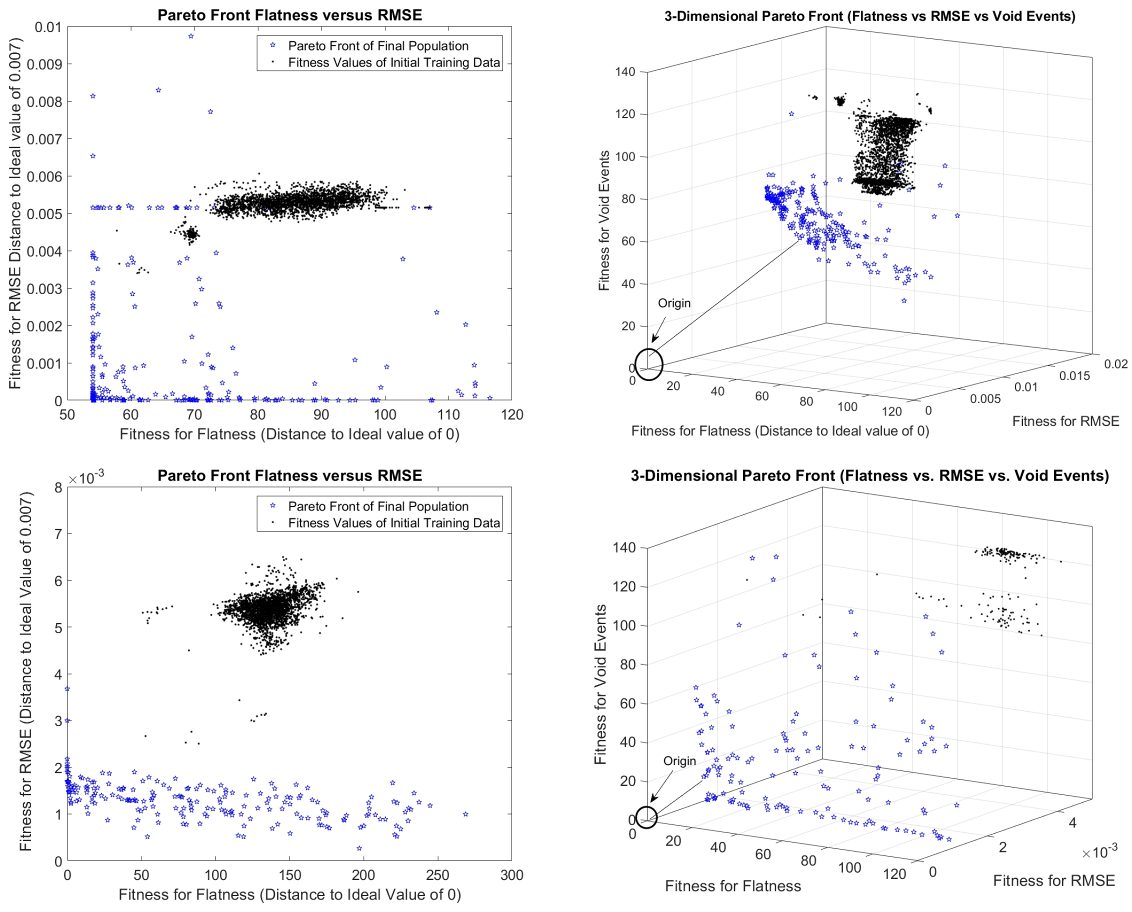


Figure 14: Pareto fronts of individuals at the end of each process optimization run: upper row for the full space (63 process values, 189 dimensions), lower row for the reduced space (13 process values, 39 dimensions); left: two-dimensional Pareto between flatness and RMSE, right: three-dimensional Pareto between flatness, RMSE and void events; please compare upper with lower row to see the improved Pareto fronts when applying the reduced space.

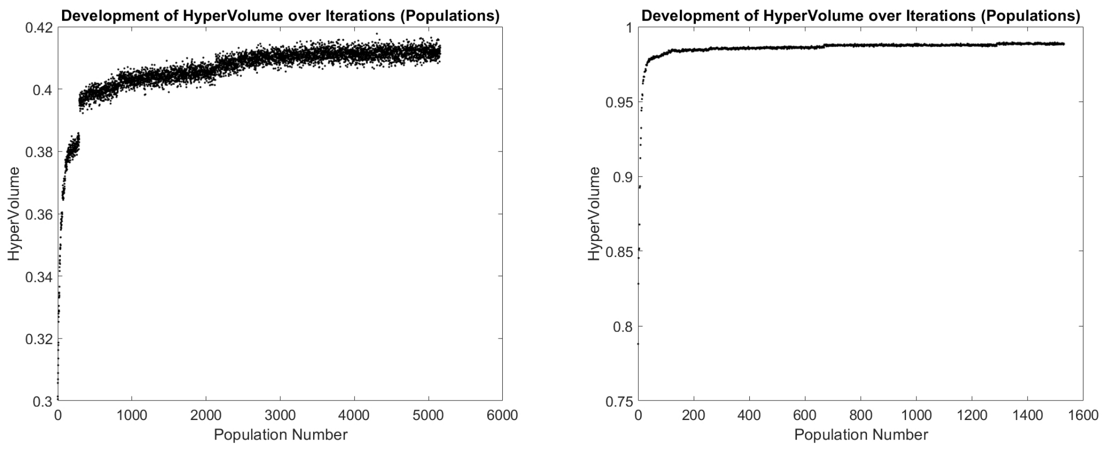


Figure 15: Hyper-volume development of Pareto fronts over the number of populations for the full space (left) and the reduced space (right); good and smooth convergence in both, but much better values for the reduced space (mind the y-axis scale), and this with lower number of populations (iterations) (mind the x-scale).