Trustworthy AI for Process Automation on a Chylla-Haase Polymerization Reactor

Daniel Hein
Siemens AG, Technology,
Munich, Germany
hein.daniel@siemens.com

Daniel Labisch
Siemens AG, Digital Industries,
Karlsruhe, Germany
daniel.labisch@siemens.com

ABSTRACT

In this paper, genetic programming reinforcement learning (GPRL) is utilized to generate human-interpretable control policies for a Chylla-Haase polymerization reactor. Such continuously stirred tank reactors (CSTRs) with jacket cooling are widely used in the chemical industry, in the production of fine chemicals, pigments, polymers, and medical products. Despite appearing rather simple, controlling CSTRs in real-world applications is quite a challenging problem to tackle. GPRL utilizes already existing data from the reactor and generates fully automatically a set of optimized simplistic control strategies, so-called policies, the domain expert can choose from. Note that these policies are white-box models of low complexity, which makes them easy to validate and implement in the target control system, e.g., SIMATIC PCS 7. However, despite its low complexity the automatically-generated policy yields a high performance in terms of reactor temperature control deviation, which we empirically evaluate on the original reactor template.

CCS CONCEPTS

• **Theory of computation → Reinforcement learning**; • **Computing methodologies → Genetic programming**; • **Applied computing → Industry and manufacturing**.

KEYWORDS

Interpretable reinforcement learning, genetic programming, process automation, real-world application

ACM Reference Format:
Daniel Hein and Daniel Labisch. 2021. Trustworthy AI for Process Automation on a Chylla-Haase Polymerization Reactor. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449726.3463131

1 INTRODUCTION

Utilizing AI-based methods in real-world industrial applications requires a certain amount of trust in the automatically-generated solution. One way to build trust, is to have machine learning (ML) methods present the reasons for their decisions in a human-understandable way [11]. On the one hand, one can attempt to interpret any useful system, even if it inherently is a black-box system, e.g., by explanation techniques for classifiers [24] or visualization methods for neural networks (NNs) [4]. On the other hand, interpretability can be evaluated via a quantifiable proxy, where at first, a model class, e.g., algebraic equations, is claimed to be interpretable and then algorithms are developed in order to optimize within this class [6, 23, 27].

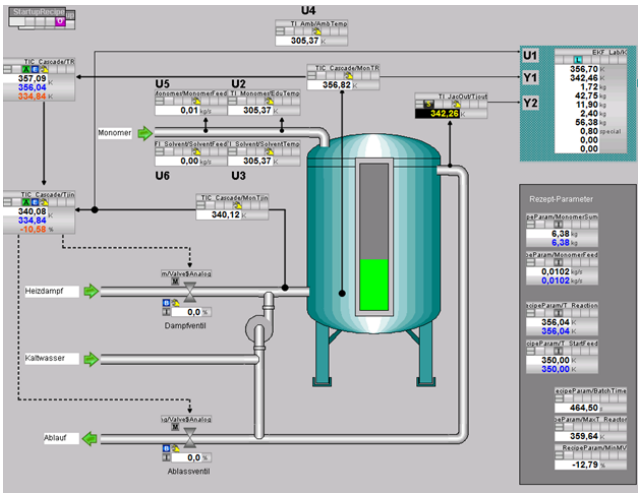

Figure 1: SIMATIC PCS 7 Chylla-Haase polymerization reactor template [1]

For the latter, recently a new method called *genetic programming reinforcement learning* (GPRL) has been proposed [18] and evaluated on challenging benchmark problems, like a hardware cart-pole demonstrator [17] or a real-world inspired industrial benchmark [16]. In the present paper, we apply GPRL for the first time on a SIMATIC PCS 7 Chylla-Haase polymerization reactor template (see Fig. 1) [1], to improve the process automation controller while keeping the policy solution interpretable and thus trustworthy. The template exemplifies the conventional control strategy using a realistic but simple simulation model directly built in SIMATIC PCS 7. It aims as a template for real applications.

If a process plant runs safely and reliably during operation, the operator always strives to maximize the economic yield. Existing degrees of freedom are used to design the operation as optimally as possible without violating existing limitations and boundary conditions. This can be done through manual intervention by the operator based on experience, or through the use of various optimization methods.

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463131

Optimization methods can basically be divided into two classes, simulation-based and data-based tools, although mixed forms are also possible. Strictly speaking, the optimization works with a simulation model in the first case and with a data-based model in the second case. An optimization based on a *general PROcess Modelling System* (gPROMS) is an example for the first case. Most data-based approaches create black-box results like neural networks. These models can show impressive results, but most of them are too complex to be interpreted or validated.

Hence, available optimization methods for process industries are either interpretable expert-generated process controllers or non-interpretable AI-generated process controllers. Herein, we show how the ML method genetic programming can be utilized together with supervised AI-based regression models to automatically generate interpretable process controllers. By doing so, the manual effort to generate interpretable process controllers can be reduced significantly. Furthermore, novel unknown control strategies can be found, which improve the quality key performance indicators (KPIs) of the controller.

In contrast to simulation-based approaches, this method significantly reduces the modeling efforts. Manually creating a simulation model is very costly in general. Even if a model already exists, e.g., from the planning phase, the adaption and parametrization of the model to represent the real behavior is still very time-consuming. Furthermore, many simulation models from the planning phase are just static ones which couldn't be easily extended to a dynamic version. The data-based approach GPRL does not require all these efforts.

Furthermore, generating black-box controllers by ML can be prone to overfit to some process model inaccuracies. Black-box controllers normally have hundreds of parameters which are automatically tuned to yield a high performance with respect to the predicted control quality. However, in many cases model inaccuracies and loopholes can be exploited by the ML process which result in controllers that only performing well on the imperfect data-based models but not on the real process. Interpretable controllers by genetic programming, formulated using the established controller building blocks are more robust to process model inaccuracies. Furthermore, experts of the respective process are able to identify contradictions and implausibilities in interpretable controllers.

The resulting process controllers can either directly replace existing control structures or can also be used as a superimposed optimization strategy. This flexibility combined with the interpretable controller function enables the process owner to balance the conflicting goals of optimizing performance and guaranteeing safety and reliability of the process.

In Sections 2 and 3, the interface of the reactor template and the GPRL method are introduced, respectively. In Section 4, we show how GPRL uses available simulation data from a Chylla-Haase reactor to generate interpretable policies. In the evaluation phase of the experiment, we show that the newly found policies reduce the control error by up to 37 %.

2 CHYLLA-HAASE POLYMERIZATION REACTOR

Continuously stirred tank reactors (CSTRs) with jacket cooling are widely used in the chemical industry, in the production of fine chemicals, pigments, polymers, and medical products. CSTRs can be run in batch, semi-batch or continuous operation. [1]

To perform an accurate control, online optimization and monitoring of processes, precise knowledge of the status of the respective process is required at all times. However, only in rare cases it is possible to measure all the states of the reactor using real sensor equipment. Often, some process variables can not be determined at all or only with great technical effort.

Simulating the dynamic process model in a soft sensor in parallel to the real process, allows the automation system to observe all modeled inner states of the reactor even if they cannot be registered by measurement equipment. The Chylla-Haase reactor, considered in our experiments, contains a universal stirred tank reactor that is employed for the production of diverse polymers with different recipes. Here, an extended Kalman filter serves as a soft sensor for monitoring the chemical reaction.

The values calculated online by the soft sensor can be used in various ways [1]:

- Calculation of the current speed of reaction and the heat released by the exothermic reaction,
- Calculation of the monomer mass remaining in the reactor, and
- Calculation of heat transfer from reactor to cooling jacket in order to detect the build-up of deposition.

A frequently considered control problem statement for Chylla-Haase reactors, is to maintain reaction temperature of a semi-batch polymerization reactor at a setpoint in spite of a variety of conditions ranging from raw material impurity, product variety, repeated batching and varying ambient temperature. Although the reactor configuration and control requirements appear rather simple, in real-world applications it is quite a challenging problem to tackle. [5]

The rector template (SIMATIC PCS 7 project and documentation) we used in our experiments is available online[1]. The production of polymer is maintained by an *sequential function chart* (SFC) recipe, where the simulation of one batch takes between 10 to 40 minutes, depending on the process parameters. The operator screen as user interface of the template is depicted in Fig. 1.

The reactor template can be influenced by two variables either given by the operator or using a superimposed optimization strategy during the batch:

- Reactor temperature setpoint $\hat{T}$, and
- Monomer feed setpoint $\hat{M}$.

These variables are called decision variables or actions.

The task is to process a certain amount of monomer into polymer, given a certain reactor setpoint. According to the recipe provided, the reactor is heated up to a certain start temperature. If this temperature is reached, the monomer is added to the stirred tank reactor at a certain monomer feed rate $\hat{M}$. As soon as a pre-defined amount of monomer is processed , the batch is finished and the tank gets

[1] https://support.industry.siemens.com/cs/de/de/view/109756215

	Variable	Description	Unit	Range
a	$\hat{T}$	Reactor temperature setpoint	K	352-365
	$\hat{M}$	Monomer feed setpoint	kg/s	0.005-0.015
s	S	Intended setpoint	K	352-365
	T	Reactor temperature	K	-
	M	Monomer mass	kg	-
	P	Polymer mass	kg	-
	UA	Heat loss coefficient	kW/K	0-1
	Q	Reaction heat flow	kW	-

Table 1: Reactor variables of action a and state s

drained. Note that the reactor temperature has to be controlled constantly by heating and cooling, since the reaction temperature is highly dependent on the amount of polymer inside the tank.

The default method to control the temperature is to use a proportional–integral–derivative (PID) controller. Today PID control is still one of the most common control strategies, since it is very often applied at the lowest level of the control hierarchy in industrial systems [3]. In 2002, Desborough and Miller conducted a survey of more than 11,000 controllers in the refining, chemicals, and pulp and paper industries which revealed that 97 % of regulatory controllers had a PID structure [10]. However, PID controllers can have severe drawbacks. They use only limited process information and the design criterion sometimes yields closed loop systems with poor robustness [2].

Hence, it is expected that an RL-based controller, which is potentially non-linear and can utilize all available process variables, can outperform standard PID regulation on many real-world use cases. The action and state variables of the reactor template, which are available for an RL controller, are given in Table 1. The action variables in $\boldsymbol{a}$ are the decision variables of the reactor template, whereas the state variables in $\boldsymbol{s}$ contain the intended setpoint S as well as sensor measurements from the reactor. Note that S is not a variable modeled by the reactor simulation, but it represents the true intended temperature setpoint of the operator. The default controller sets $\hat{T} = S$, i.e., the operator's intended setpoint is directly used as input for the underlying PID regulator.

3 GENETIC PROGRAMMING REINFORCEMENT LEARNING

The GPRL approach learns policy representations which are basic algebraic equations of low complexity [18]. Given that GPRL can find rather short (non-complex) equations, it is expected to reveal substantial knowledge about underlying coherencies between available state variables and well-performing control policies for a certain RL problem. By the term complexity of a policy, we refer to its human-interpretable form, e.g., a simple algebraic equation can be easily understood by a domain expert, hence it is of low complexity, whereas a non-linear system of equations, like a neural network, cannot be validated by a human expert, yielding a high complexity.

Note that GPRL is not only optimizing a certain trade-off between policy complexity and performance, but rather it evolves a whole optimized Pareto front from which the domain expert can choose a trustworthy and well-performing solution.

3.1 Population-based reinforcement learning

Generally, reinforcement learning (RL) is distinguished from other computational approaches by its emphasis on an agent learning from direct interaction with its environment. This approach, which does not rely on exemplary supervision or complete models of the environment, is referred to as *online learning*. However, for many real-world problems online learning is prohibited for safety reasons. For example, it is not advisable to deploy an online RL agent, who starts by applying an arbitrary initial policy, which is subsequently improved by exploitation and exploration, on safety-critical domains like process industries or power plants. For this reason, *offline learning* is a more suitable approach for applying RL methods on already collected training data and system models, to yield RL policies.

Offline RL is often referred to as *batch learning* because it is based solely on a previously generated batch of transition samples from the environment. The batch data set $\mathcal{D}$ contains transition tuples of the form $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, r_{t+1})$, where the application of action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$ resulted in a transition to state $\boldsymbol{s}_{t+1}$ and yielded a reward r_{t+1}, where t denotes a discrete time step.

Herein, we use *model-based value estimation* to estimate the performance of a policy. In the first step, supervised ML is applied to learn approximate models of the underlying environment from transition tuples $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, r_{t+1})$ as follows:

$$\tilde{g}(\boldsymbol{s}_t, \boldsymbol{a}_t) \leftarrow \boldsymbol{s}_{t+1}, \quad \tilde{r}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \leftarrow r_{t+1}. \tag{1}$$

Using models $\tilde{g}$ and $\tilde{r}$, the value for policy π of each state $\boldsymbol{s}$ in the data batch can be estimated by computing the value function $\tilde{v}_\pi(\boldsymbol{s})$. Hence, using model-based value estimation means performing trajectory rollouts on system models for different starting states:

$$\tilde{v}_\pi(\boldsymbol{s}_t) = \sum_{k=0}^{\infty} \gamma^k \tilde{r}(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}), \tag{2}$$

$$\text{with} \quad \boldsymbol{s}_{t+k+1} = \tilde{g}(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}), \quad \boldsymbol{a}_{t+k} = \pi(\boldsymbol{s}_{t+k}), \tag{3}$$

and discount factor $0 \leq \gamma \leq 1$.

Since we are searching for interpretable solutions, the resulting policies have to be represented in an explicit form. *Policy search* yields explicit policies which can be enforced to be of an interpretable form. Furthermore, policy search is inherently well-suited for being used with population-based optimization techniques, like genetic programming (GP).

The goal of using policy search for learning interpretable RL policies is to find the best policy among a set of policies that is spanned by a parameter vector $\boldsymbol{x} \in \mathcal{X}$, where $\mathcal{X}$ is the space of all interpretable policy parameters for this RL task. Herein, a policy corresponding to a particular parameter value $\boldsymbol{x}$ is denoted by $\pi[\boldsymbol{x}]$. The policy's performance, when starting from $\boldsymbol{s}_t$ is measured by its value function given in Eq. (2). Furthermore, including only a finite number of $T > 1$ future rewards for the model-based value

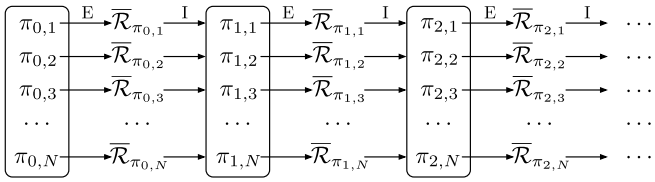

Figure 2: Population-based policy search iterating policy evaluation (E) and policy improvement (I)

estimation from Eq. (2) yields

$$\tilde{v}_{\pi[\boldsymbol{x}]}(\boldsymbol{s}_t) = \sum_{k=0}^{T-1} \gamma^k \tilde{r}(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}), \quad \text{with} \quad \boldsymbol{s}_{t+k+1} = \tilde{g}(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}) \quad \text{and} \quad \boldsymbol{a}_{t+k} = \pi[\boldsymbol{x}](\boldsymbol{s}_{t+k}). \tag{4}$$

To rate the performance of policy $\pi[\boldsymbol{x}]$, the value function is used as follows:

$$\overline{\mathcal{R}}_{\pi[\boldsymbol{x}]} = \frac{1}{|\mathcal{S}|} \sum_{\boldsymbol{s} \in \mathcal{S}} \tilde{v}_{\pi[\boldsymbol{x}]}(\boldsymbol{s}), \tag{5}$$

with $\overline{\mathcal{R}}_{\pi[\boldsymbol{x}]}$ being the average discounted return of $\pi[\boldsymbol{x}]$ on a representative set of test states $\mathcal{S}$.

In population-based RL, the interest lies in populations of policies, which means that multiple different policies exist at the same time in one policy iteration step. Fig. 2 depicts how the return values of all population members drive the improvement step (I) of the whole population. The evaluation step (E) is performed individually for every member. Applying population-based evolutionary methods to policy search has already been successfully achieved in the past [8, 9, 15, 28].

3.2 Genetic programming

GP has been utilized for creating system controllers since its introduction [21]. Since then, the field of GP has grown significantly and has produced numerous results that can compete with human-produced solutions [22], including system controllers [20, 26], game playing [14, 28], and robotics [12, 19].

GP encodes computer programs as sets of genes and then modifies (evolves) them using a so-called genetic algorithm (GA) to drive the optimization of the population, by applying selection and reproduction to the population. The basis for both concepts is a fitness value, which represents the quality of performing the predefined task for each individual. Selection means, that only the best portion of the current generation will survive each iteration and continue existing in the next generation. Analogous to biological sexual breeding, two individuals are selected for reproduction based on their fitness, and two offspring individuals are created by crossing their chromosomes. Technically, this is realized by selecting compatible cutting points in the function trees and subsequently interchanging the subtrees beneath these cuts. The two resulting individuals are introduced to the population of the next generation. Herein, we applied tournament selection [7] for selecting the individuals to be crossed.

To rate the quality of each policy candidate, a fitness value has to be provided in order for the GP algorithm to advance. For GPRL, the fitness of each individual is calculated by generating trajectories using the model-based return estimation from Eq. (5).

The overall GA used in the experiments is given as follows:

(1) Randomly **initialize** the population of size N
(2) **Determine fitness** value of each individual using Eq. (5) (in parallel)
(3) **Evolve** next generation
  (a) **Crossover** (depending on crossover ratio r_c)
    (i) Select individuals by **tournament selection**
    (ii) Cross two tournament winners
    (iii) Add resulting individuals to new population
  (b) **Reproduction** (depending on reproduction ratio r_r)
    (i) Select individuals by **tournament selection**
    (ii) Add tournament winner to new population
  (c) Automatic **cancelation** and terminal **mutation** (depending on auto cancel ratio r_a and terminal mutation ration r_m)
    (i) Apply automatic **cancelation** on all individuals
    (ii) Add canceled individuals according to r_a
    (iii) Select best individuals for each complexity level of old population
    (iv) Randomly **mutate** float terminals using normal distribution $\mathcal{N}$: $z' \sim z + 0.1z \cdot \mathcal{N}(0, 1)$, where z and z' are the original and the mutated float terminals, respectively; and create $N \cdot r_\mathrm{a}$ adjusted individuals from each best
    (v) Determine fitness value of each individual (in parallel)
    (vi) Add best adjusted individuals to new population according to r_m
  (d) **Fill** new population with new randomly generated individuals (new individuals ratio r_n)
  (e) **Determine fitness** value of each individual using Eq. (5) (in parallel)
  (f) If none of the stopping criteria is met
    (i) Go back to 3.
(4) **Return** best individual found so far for each complexity level

For the experiments described in Section 4, the following new population ratios have been used: $r_\mathrm{c} = 0.45$, $r_\mathrm{r} = 0.05$, $r_\mathrm{m} = 0.1$, $r_\mathrm{a} = 0.1$, $r_\mathrm{n} = 0.3$. Stopping criteria can be a maximum number of iterations, a certain algorithm runtime, a pre-defined performance value, etc. In our experiments, we stopped the GA after 100 iterations.

GPRL has been implemented using the open source evolutionary computation framework DEAP[2] which provides an excellent and widely-used Python implementation of genetic programing using prefix trees [13]. A detailed explaination of the GA utilized in GPRL, including a description of automatic cancelation and parameter motivation, can be found in [16].

4 INTERPRETABLE PROCESS CONTROLLERS FOR A CHYLLA-HAASE POLYMERIZATION REACTOR

The framework of learning interpretable process controllers contains two ML steps. Firstly, available data from the process is used

[2] https://github.com/DEAP/deap

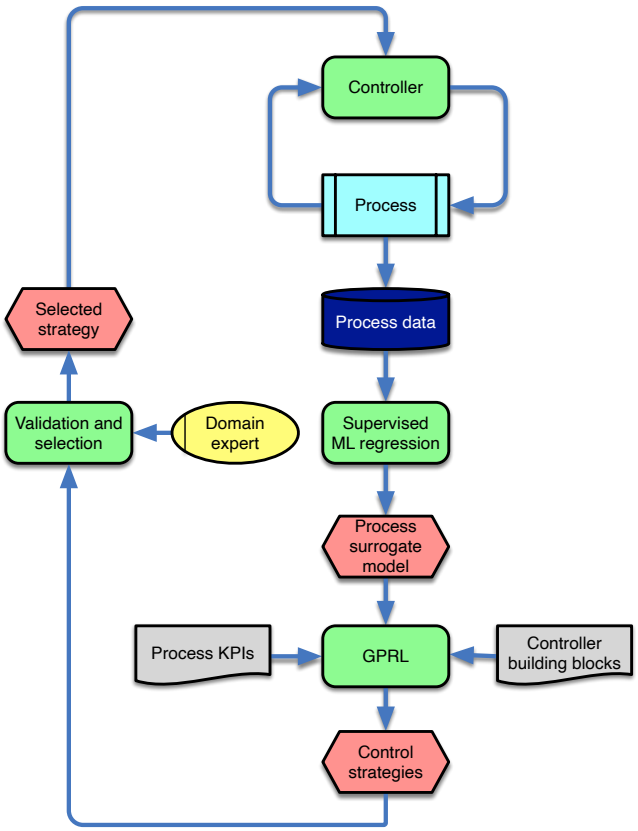

Figure 3: EPC of generating interpretable process controllers using GPRL. Note that the Domain expert is integrated into the optimization loop. However, generating interpretable control strategies is fully automatized.

to generate a process model. Here, we used supervised ML to train weights of a recurrent neural network as system identification. Secondly, GPRL is performed on this recurrent model to produce several Pareto-optimized process controllers. The controllers are optimized towards the process KPIs utilizing the platform-specific process control building blocks. An event-driven process chain (EPC) flowchart of applying GPRL for process optimization is depicted in Fig. 3.

4.1 Recurrent system identification

To generate a batch data set $\mathcal{D}$ with sufficient experience, 100 different exploration process recipes have been used. In the classic control strategy, using a PID temperature controller, the controller receives a constant setpoint. As the controller needs to handle both, heating and cooling during the production, the performance is not optimal and significant control deviations could be observed. This will be shown in Section 4.4 together with the results. For safety and acceptance reasons, GPRL is not intended to replace the PID controller but to be superimposed on it. Therefore, the setpoint temperature $\hat{T}$ of the PID controller becomes the action of GPRL and a new overall setpoint S is introduced (see Table 1). Modifying $\hat{T}$ reveals new degrees of freedom not used before in the operation of the reactor. Therefore, new training data needs to be created wherein $\hat{T}$ is changed. For our experiments, the setpoint has been changed once per recipe between 352 and 365 K at a random batch runtime between 100 and 600 s. In process optimization, such step attempts are a common method of safely exploring in a bounded subspace of the control problem.

The response of the process on these changes is then recorded and exported to the data batch. In the next step, the data is subsampled on a 10 s grid and then normalized, i.e., average 0 and standard deviation 1.

The system model $\tilde{g}$ is a recurrent neural network (RNN) with the task to predict the values of $\boldsymbol{s}_{t+1} = (T_{t+1}, M_{t+1}, P_{t+1}, UA_{t+1}, Q_{t+1})$, $\boldsymbol{s}_{t+2}, \ldots, \boldsymbol{s}_{t+F}$ from the inputs $\boldsymbol{s}_t = (T_t, M_t, P_t, UA_t, Q_t), \boldsymbol{s}_{t-1}, \ldots, \boldsymbol{s}_{t-H+1}$ and $\boldsymbol{a}_t = (\hat{T}_t, \hat{M}_t), \boldsymbol{a}_{t-1}, \ldots, \boldsymbol{a}_{t-H+1}$. The RNN is unfolded $H = 10$ time steps into the past and $F = 10$ time steps into the future. In each time step, the observable variables of the past and present are inputs. Whereas, in the future branch of the RNN only the control variables $\hat{T}$ and $\hat{M}$ are used as input. The topology of the RNN is described in [25] as *Markov decision process extraction network*. The model was implemented as a tensorflow graph using RNN cells with two hidden layers and 20 tanh activation neurons on each layer.

Note that an $F = 10$ overshooting is used, which means that the regression error used to optimize the network weights, is not only computed on the next time step, but as an average error over the next 10 time steps. In our experience, this generally helps to yield more robust predictions for models which are used in closed loop simulations, where predictions have to be made on the models own, possibly inaccurate, previous predictions.

After 10,000 training episodes, the quadratic loss of the prediction has fallen to 0.0017 on the trainings set, 0.0022 on the validations set, and 0.0021 on the generalization set (see Fig. 4). The sets contained 70, 20, and 10 step attempt time series for training, validation, and generalization, respectively. Fig. 5 depicts the average absolute loss for the predicted reactor temperature T over the 10 overshooting time steps. It is shown, how the prediction quality is decreasing for more distant time steps. However, a validation error increase from 0.019 to 0.026 on a prediction horizon of 100 seconds (10 time steps) still yields an adequate system model.

4.2 Learning an interpretable process controller

The RNN, trained in the previous step, can now be used as model $\tilde{g}$ together with the reward function $\tilde{r}$ to rate the performance of policy candidates. In case of the temperature control task we consider here, the reward is computed by $\tilde{r}(\boldsymbol{s}_t) = r_t = -(S - T_t)^2$, where S_t is the desired setpoint and T_t is the predicted reactor temperature at time step t. According to Eq. (4), the controller parameters $\boldsymbol{x}$ are tested on 100 starting states $\mathcal{S}$ drawn from the data batch $\mathcal{D}$. The trajectories produced by closed loop evaluation of policy π with parameters $\boldsymbol{x}$ and system model $\tilde{g}$, have a length of $T = 10$. Hence, the performance of $\boldsymbol{x}$ can be estimated by Eq. (5).

The building blocks of the genetic programming algorithm are basic algebraic functions $\{+, -, *, /\}$, state and action variables $\{\boldsymbol{s}, \boldsymbol{a}\}$, and dead time blocks storing past values of the variables up to 50

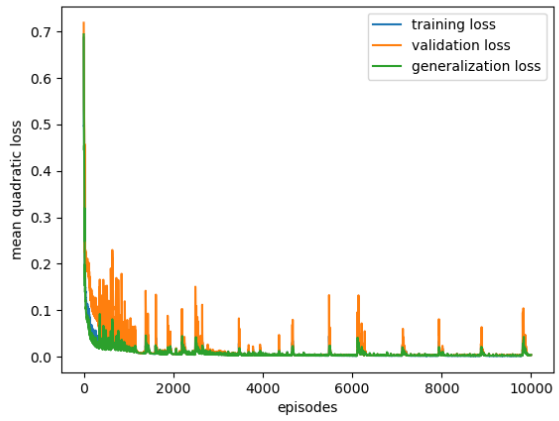

Figure 4: Learning curve of the RNN surrogate model training

Figure 5: Prediction error of the trained model RNN surrogate model over ten future time steps

seconds $\{\square_t, \square_{t-10}, \ldots, \square_{t-50}\}$, with $\square$ an arbitrary variable from $\boldsymbol{s}$ or $\boldsymbol{a}$. Here, we are searching for a policy that serves as a setpoint process controller which dynamically changes setpoint $\hat{T}$ to minimize the distance between the desired setpoint S and the actual reactor temperature T. The monomer setpoint $\hat{M}$ has been fixed to the value 0.015. Note that the policy can incorporate state and action values from the past and is evaluated in a closed loop manor for ten time steps during training. Consequently, it will not greedily try to reach the desired setpoint in the next step, but in the long run try to avoid overshooting and maximize the reward instead.

After 100 iterations of GPRL with 500 individuals in each generation, the performance of the best policies found are visualized in Fig. 6. It is easy to see that individuals of lower complexity have a higher penalty (lower reward) compared to individuals of higher

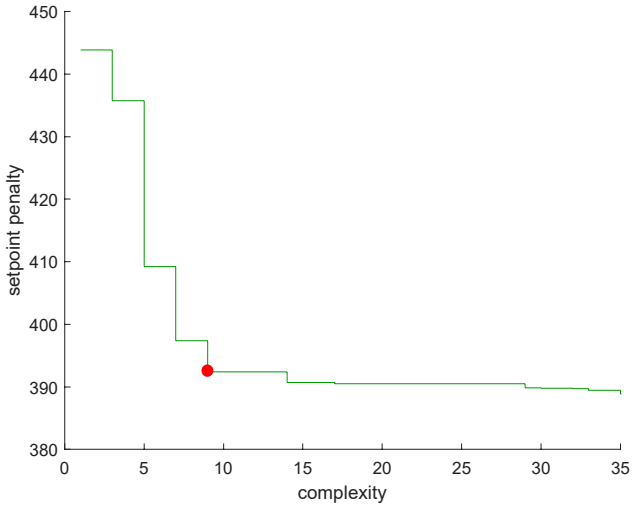

Figure 6: Pareto front generated by GPRL. According to the domain expert, the selected policy (red dot) is a good compromise between setpoint penalty (392) and complexity (9).

complexity. However, the estimated performance seems to not improve significantly for individuals of complexity 10 or higher. After discussion with domain experts, an individual of complexity 9 and an estimated penalty of around 392 has been selected (highlighted in red). The selected individual can be represented as the following equation:

$$\hat{T}_t = T_{t-30} - 2T_t + 2S - 1. \tag{6}$$

This equation can easily be interpreted thanks to its simple structure. The reactor temperature setpoint is dynamically changed with respect to the reactor temperature 30 s ago, the current temperature, and the intended setpoint. Note that all variables (T, $\hat{T}$, and S) in the equation have been normalized by average 359.12 and standard deviation 6.47.

4.3 SIMATIC PCS 7 integration

The policy found by GPRL and represented by Eq. (6), can easily be implemented in SIMATIC PCS 7, since GPRL utilized only components from the pre-defined building blocks which are available in the target system. Fig. 7 shows a screenshot of the respective *structured control language* (SCL) code. Note, that since this is the native language of the system at hand, the new policy is easy to integrate and fast in execution. Domain experts are able to fully understand the code and can easily make adjustments. All of this would be impossible if we were to use a black-box policy.

After generating the SCL code, the new control strategy can be loaded into a standard *continuous function chart* (CFC) building block. Fig. 8 shows the CFC diagram with our new policy block (FB10) and the additional dead time block (DeadTime) with 30 s below. The policy is now fully integrated into the automation system and can be evaluated.

Note that for applying the function to a real production plant, some safety logic and switching between different modes like manual and automatic would be required. Integration of this is straight

```
// Constants
// ***************************************************************************
CONST
    T_R_setpoint_Avg    := 359.12;
    T_R_setpoint_Std    := 6.47;
END_CONST


// Function
// ***************************************************************************
BEGIN
    IF BatchTime > 0 AND BatchTime < 2 THEN
        Counter_Out := Counter_In + 1;
    END_IF;

    T_R_0_Norm := (T_R_0 - T_R_setpoint_Avg) / T_R_setpoint_Std;
    T_R_30_Norm := (T_R_30 - T_R_setpoint_Avg) / T_R_setpoint_Std;
    my_sp_Norm := (my_sp - T_R_setpoint_Avg) / T_R_setpoint_Std;

    Set_T_Reaction := my_sp_Norm;

    IF BatchTime > 30 THEN
        Set_T_Reaction_Policy := T_R_30_Norm - 2.0*(T_R_0_Norm - my_sp_Norm) - 1.0;
        Set_T_Reaction := Set_T_Reaction_Policy;
    END_IF;

    Set_T_Reaction := Set_T_Reaction * T_R_setpoint_Std + T_R_setpoint_Avg;

    IF Set_T_Reaction < 352.0 THEN
        Set_T_Reaction := 352.0;
    ELSIF Set_T_Reaction > 365.0 THEN
        Set_T_Reaction := 365.0;
    END_IF;

    Set_MonomerFeed := 0.015;

END_FUNCTION_BLOCK
```

Figure 7: SCL script with the implemented GPRL control strategy

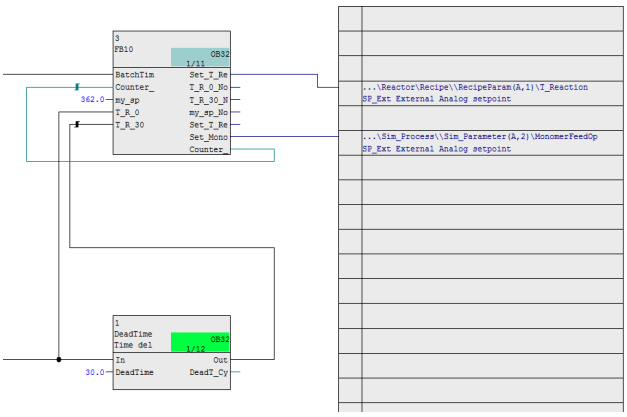

Figure 8: CFC diagram with the building block (FB10) containing the GPRL policy

forward and neglected for the following evaluation using the simulation model.

4.4 Evaluation

To evaluate and visualize the performance of the policy, we tested it on four exemplary intended setpoints, i.e., 352, 358, 362, 365 K. The intended setpoint S is plotted in red. The action of the policy $\hat{T}$ is plotted in blue. The resulting reactor temperature T, using the default PID regulator $\hat{T} = S$, is plotted in orange. The resulting reactor temperature T, using the new RL policy given in Eq. (6), is plotted in green.

Fig. 9 depicts the results for $S = 352$ K. In the beginning the RL policy changes the temperature setpoint to the maximum value of 365 K, to speed up the heating process of the reactor. After approximately 80 s, it reduces the setpoint to the minimum value

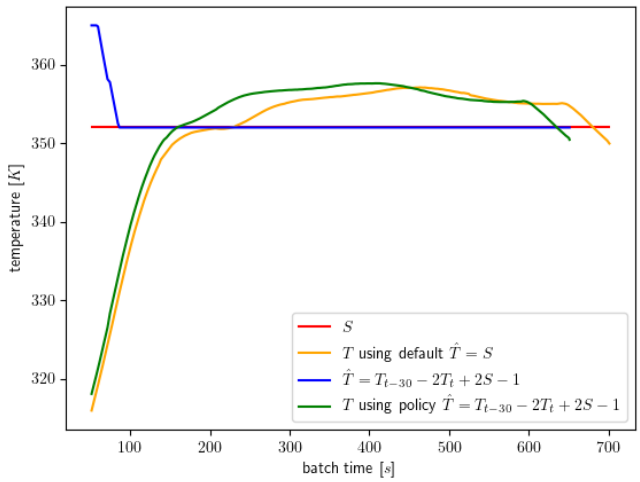

Figure 9: Setpoint controls and resulting trajectories for default and GPRL policies. Results are shown for an intended reactor setpoint of 352 K.

of 352 K, to keep the reactor temperature as close as possible to the intended setpoint. However, since the setpoint can not be lowered below the minimum of 352 K, the reactor is constantly above the intended setpoint. In this example, the new RL policy could slightly reduce control deviation by 0.3 %.

A real improvement can be seen for intended setpoint $S = 358$ K. Fig. 10 shows that the default controller produces huge overshooting, whereas the new RL policy stays close to the intended setpoint over the whole batch time. Again, the RL policy is increasing the setpoint to the maximum at the beginning just to reduce it to the minimum after 100 s. After 400 s we can see that the RL policy is starting to slightly increase the setpoint again to counteract the decrease in reactor temperature at the end of the batch. The control deviation can be reduced by 33.5 %.

In Fig. 11, we see a similar behavior for intended setpoint $S = 362$ K. The new RL policy changes the setpoint dynamically over the whole batch time to reduce control deviation by 37.7 %.

The final example is intended setpoint $S = 365$ K in Fig. 12. Here, the new RL policy reduces control deviation by 30.1 %. Note that in this example, it is very important not to overshoot too much, since depending on the product in the reactor, too high temperatures can spoil the process and ruin the quality of the final product.

To conclude, we can say that the control deviation has been reduced for all intended setpoints tested. The policy was automatically learned from safely generated batch data, easy to implement, and reduced control deviation by up to 37 %.

5 CONCLUSION

The experiments using GPRL on a SIMATIC PCS 7 polymerization reactor template show that it is possible to generate human-interpretable policies from existing process data fully automatically. In contrast to widely-known RL methods, which produce black-box value functions and/or policies, with GPRL, the domain experts can stay in the optimization loop, since the results are convenient to validate and implement. Utilizing such trustworthy AI methods will

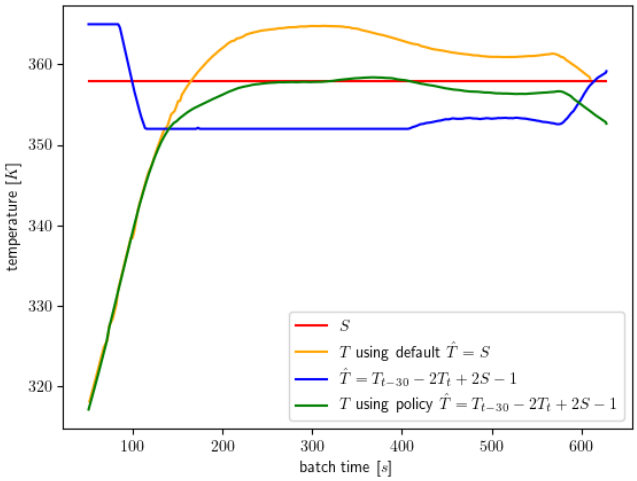

Figure 10: Setpoint controls and resulting trajectories for default and GPRL policies. Results are shown for an intended reactor setpoint of 358 K.

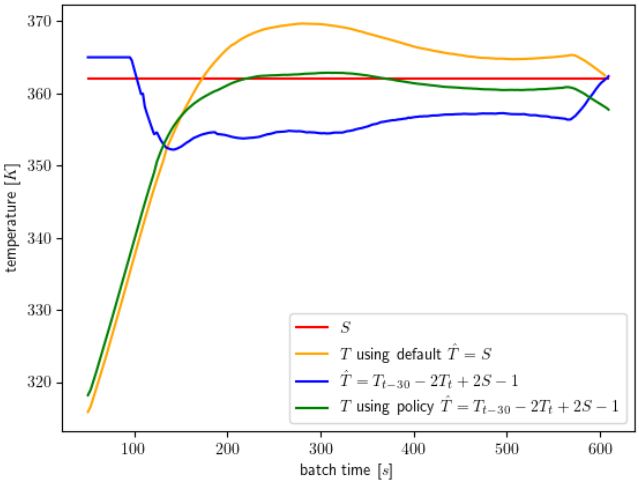

Figure 11: Setpoint controls and resulting trajectories for default and GPRL policies. Results are shown for an intended reactor setpoint of 362 K.

help bring state-of-the-art ML algorithms into real-world industry applications, to leverage the optimization potentials which are to be expected in many domains.

ACKNOWLEDGMENTS

The project this report is based on was supported with funds from the German Federal Ministry of Education and Research under project number 01IS18049A. The sole responsibility for the report's contents lies with the authors.

REFERENCES

[1] Siemens AG. 2018. *PCS 7 Unit Template "Stirred tank reactor with Kalman filter" using the example of the Chemical Industry.* https://support.industry.siemens.com/cs/de/de/view/109756215

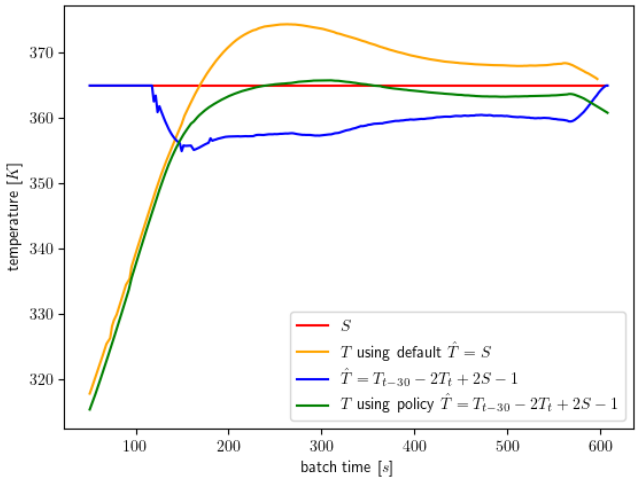

Figure 12: Setpoint controls and resulting trajectories for default and GPRL policies. Results are shown for an intended reactor setpoint of 365 K.

[2] K.J. Åström and T. Hägglund. 2001. The future of PID control. *Control engineering practice* 9, 11 (2001), 1163–1175.
[3] K.J. Åström and T. Hägglund. 2004. Revisiting the Ziegler–Nichols step response method for PID control. *Journal of process control* 14, 6 (2004), 635–650.
[4] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one* 10, 7 (2015), e0130140.
[5] A. Bhat and R.N. Banavar. 1998. The Chylla-Haase problem: a neural network controller. In *Proceedings of the 1998 IEEE International Conference on Control Applications (Cat. No. 98CH36104)*, Vol. 1. IEEE, 192–196.
[6] B. Bischoff, D. Nguyen-Tuong, T. Koller, H. Markert, and A. Knoll. 2013. Learning throttle valve control using policy search. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 49–64.
[7] T. Blickle and L. Thiele. 1995. A Mathematical Analysis of Tournament Selection. In *ICGA*. 9–16.
[8] H.S. Chang, J. Hu, M.C. Fu, and S.I. Marcus. 2007. Population-Based Evolutionary Approaches. In *Simulation-Based Algorithms for Markov Decision Processes*. Springer, Chapter 3, 61–87.
[9] H.H. Chin and A.A. Jafari. 1998. Genetic algorithm methods for solving the best stationary policy of finite Markov decision processes. In *System Theory, 1998. Proceedings of the Thirtieth Southeastern Symposium on*. IEEE, 538–543.
[10] L. Desborough and R. Miller. 2002. Increasing customer value of industrial control performance monitoring - Honeywell's experience. In *AIChE symposium series*. New York; American Institute of Chemical Engineers; 1998, 169–189.
[11] F. Doshi-Velez and B. Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608* (2017).
[12] K.L. Downing. 2001. Adaptive Genetic Programs via Reinforcement Learning. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation* (San Francisco, California) *(GECCO'01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 19–26.
[13] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. 2012. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research* 13, 1 (2012), 2171–2175.
[14] C. Gearhart. 2003. Genetic Programming as Policy Search in Markov Decision Processes. In *Genetic Algorithms and Genetic Programming at Stanford*, J. R. Koza (Ed.). Stanford Bookstore, Stanford, USA, 61–67.
[15] F. Gomez, J. Schmidhuber, and R. Miikkulainen. 2006. Efficient non-linear control through neuroevolution. In *European Conference on Machine Learning*. Springer, 654–662.
[16] D. Hein. 2019. *Interpretable Reinforcement Learning Policies by Evolutionary Computation*. Ph.D. Dissertation. Technische Universität München.
[17] D. Hein, S. Limmer, and T.A. Runkler. 2020. Interpretable Control by Reinforcement Learning. *IFAC-PapersOnLine* 53, 2 (2020), 8082–8089. 21th IFAC World Congress.
[18] D. Hein, S. Udluft, and T.A. Runkler. 2018. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence* 76 (2018), 158–169.

[19] S. Kamio and H. Iba. 2005. Adaptation Technique for Integrating Genetic Programming and Reinforcement Learning for Real Robots. *Trans. Evol. Comp* 9, 3 (June 2005), 318–333.

[20] M.A. Keane, J.R. Koza, and M.J. Streeter. 2002. Automatic Synthesis Using Genetic Programming of an Improved General-Purpose Controller for Industrially Representative Plants. In *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware (EH'02) (EH '02)*. IEEE, Washington, DC, USA, 113–123.

[21] J.R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[22] J.R. Koza. 2010. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines* 11, 3 (2010), 251–284.

[23] F. Maes, R. Fonteneau, L. Wehenkel, and D. Ernst. 2012. Policy search in a space of simple closed-form formulas: Towards interpretability of reinforcement learning. *Discovery Science* (2012), 37–50.

[24] M.T. Ribeiro, S. Singh, and C. Guestrin. 2016. Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1135–1144.

[25] A.M. Schäfer and S. Udluft. 2005. Solving partially observable reinforcement learning problems with recurrent neural networks. In *Workshop Proc. of the European Conf. on Machine Learning*. 71–81.

[26] H. Shimooka and Y. Fujimoto. 1999. Generating Equations with Genetic Programming for Control of a Movable Inverted Pendulum. In *Selected Papers from the Second Asia-Pacific Conference on Simulated Evolution and Learning on Simulated Evolution and Learning (SEAL'98)*. Springer-Verlag, London, UK, 179–186.

[27] T. Wang, C. Rudin, F. Velez-Doshi, Y. Liu, E. Klampfl, and P. MacNeille. 2016. Bayesian rule sets for interpretable classification. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 1269–1274.

[28] D.G. Wilson, S. Cussat-Blanc, H. Luga, and J.F. Miller. 2018. Evolving simple programs for playing Atari games. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Kyoto, Japan) *(GECCO '18)*. ACM, New York, NY, USA, 229–236.