DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Interpretable Reinforcement Learning Policies by Evolutionary Computation

Daniel Hein

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Stephan Günnemann
Prüfer der Dissertation:	1. Hon.-Prof. Dr.-Ing. Thomas A. Runkler
	2. Prof. Dr.-Ing. habil. Alois Christian Knoll

Die Dissertation wurde am 14.12.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 10.05.2019 angenommen.

Acknowledgments

I want to thank my supervisor Prof. Dr. Thomas Runkler for his excellent supervision. Thomas' valuable guidance regarding the focus of my research, preparation of publications, and structuring the existing material into a monolithic thesis made a huge difference in finalizing this work. I would also like to thank my advisors from Siemens Dr. Alexander Hentschel and Dr. Steffen Udluft, for directing me to promising areas of study, participating in numerous productive meetings, giving valuable advice, and finally, reviewing publications and this thesis. While Alex helped me significantly to start working on the topics reinforcement learning and genetic programming in the first years of my doctoral studies, Steffen undertook Alex's responsibilities after his leave. I wish to thank Steffen not only for his valuable subject-specific advice, but beyond that for broadening my horizon during coffee breaks with book reviews and philosophical discussions. I am very thankful to the head of my research group "Learning Systems", Volkmar Sterzing, who highly values the research conducted by us PhD candidates and has always been dedicated to providing us with the required elements for a successful industry-sponsored doctoral study, like computational resources, business contacts, travel budgets, and, most important of all, the freedom to focus on our research tasks. Further thanks go to my fellow PhD candidates at Siemens, Markus Kaiser and Stefan Depeweg, for sharing many interesting discussions and working together on experiments and publications. I am glad to say that all of my colleagues at Siemens created a productive and friendly environment, which helped me throughout my research.

I particularly wish to thank my parents, Kerstin and Mario Hein, for their continuous support throughout my whole educational and academic career. You always encouraged me to believe in my skills and to do what I am most interested in. Last, but not least, I would like to express my gratitude to my wife Asnāte Streļča for thorough proof-reading and valuable comments on several drafts of this thesis as well as for her unconditional support during stressful and sometimes frustrating times.

The project this thesis is based on was supported with funds from the German Federal Ministry of Education and Research under project number 01IB15001.

Daniel Hein
Munich, October 2019

Abstract

In this thesis, three novel algorithms for generating interpretable reinforcement learning policies from batches of previously generated system transitions, are proposed and evaluated. On challenging benchmarks, it is empirically shown that the algorithms generate human-interpretable control strategies of competitive performance and superior generalization capabilities by using evolutionary computation methods.

In the context of machine learning, the need for interpretability stems from an incompleteness in the problem formalization. Since complex real-world tasks in industry are almost never completely testable, enumerating all possible outputs given all possible inputs is infeasible. Hence, we usually are unable to flag all undesirable outputs. Especially for industrial systems, domain experts are more likely to deploy automatically learned controllers if they are understandable and convenient to assess. Moreover, novel legal frameworks such as the European Union's General Data Protection Regulation enforce interpretability of personal data processing systems.

Two of the three novel reinforcement learning methods of this thesis learn policies represented as fuzzy rule-based controllers since fuzzy controllers have proven to serve as interpretable and efficient system controllers in industry for decades. The first method, called *fuzzy particle swarm reinforcement learning* (FPSRL), utilizes swarm intelligence to optimize parameters of a fixed fuzzy rule set, whereas the second method, called *fuzzy genetic programming reinforcement learning* (FGPRL), applies genetic programming to generate a new fuzzy set, including the optimization of all parameters, from available building blocks. Empirical studies on benchmark problems show that FPSRL has advantages regarding computational costs on rather simple problems, where prior expert knowledge about informative state features and rule numbers is available. However, experiments using an industrial benchmark show that FGPRL can automatically select the most informative state features as well as the most compact fuzzy rule representation for a certain level of performance. The third interpretable approach, called *genetic programming reinforcement learning* (GPRL), finally drops the constraint on learning rule-based policies by representing the policies as basic algebraic equations of low complexity. Experimental results show that the GPRL policies yield human-understandable and well-performing control results. Moreover, both FGPRL and GPRL return not just one solution to the problem but a whole Pareto front containing the best-performing solutions for many different levels of complexity.

Comparing the results from experiments of all three interpretable reinforcement learning approaches with the performance of standard neural fitted Q iteration, a novel model predictive control approach, and a non-interpretable neural network policy method gives a comprehensive overview on the performance of the methods as well as the interpretability of the produced policies. However, choosing the most interpretable form of presentation is highly subjective and depends on many prerequisites, like the application domain, the ability to visualize solutions, or successive processing steps, for example. Therefore, it is all the more important to have methods at hand which can search domain-specific policy representation spaces automatically. The empirical studies in this thesis show that, combining model-based reinforcement learning with genetic programming, is a very promising approach to achieve this goal.

Zusammenfassung

In dieser Dissertation werden drei neuartige Algorithmen zur Erzeugung von interpretierbaren Aktionsauswahlregeln durch bestärkendes Lernen aus zuvor aufgezeichneten Zustandsübergängen vorgeschlagen und evaluiert. Durch empirische Versuche an herausfordernden Testproblemen wird gezeigt, dass die erzeugten, für Menschen interpretierbaren Regelungsstrategien durch den Einsatz evolutionärer Algorithmen eine konkurrenzfähige Regelungsgüte und darüber hinaus, im Vergleich mit herkömmlichen Lernmethoden, bessere Generalisierungseigenschaften besitzen.

Beim maschinellen Lernen ergibt sich der Bedarf nach Interpretierbarkeit aus der Unvollständigkeit der Problemformulierung. Da Lösungen für komplexe Problemstellungen aus der Industrie so gut wie nie vollständig überprüfbar sind, ist es unmöglich, alle möglichen Systemausgaben zu allen möglichen Systemeingaben aufzulisten. Daraus folgt, dass man genauso wenig in der Lage ist, alle denkbaren ungewünschten Systemausgaben zu benennen. Besonders bei industriellen Anwendungen ist davon auszugehen, dass Fachleute durch maschinelles Lernen erzeugte Steuerungen eher akzeptieren, wenn diese verstehbar und einfach auswertbar sind. Darüber hinaus fordern heutzutage auch gesetzliche Rahmen wie die EU-Datenschutz-Grundverordnung, dass Systeme, die persönliche Daten verarbeiten, interpretierbar sein müssen.

Zwei der drei neuen Methoden des bestärkenden Lernens aus dieser Dissertation lernen Aktionsauswahlregeln, die durch Fuzzy-Regler dargestellt werden, da Fuzzy-Regler bereits seit Jahrzehnten als interpretierbare und effiziente Regelungsstrategie bekannt sind. Die erste Methode heißt *Fuzzy Particle Swarm Reinforcement Learning* (FPSRL) und setzt Schwarmintelligenz ein, um die Parameter einer vordefinierten Fuzzy-Regel-Menge zu optimieren. Die zweite Methode heißt *Fuzzy Genetic Programming Reinforcement Learning* (FGPRL) und benutzt genetische Programmierung um sowohl neue Fuzzy-Regel-Mengen aus verfügbaren Regelbausteinen zu erzeugen, als auch direkt deren freie Parameter optimal einzustellen. Empirische Untersuchungen an Testproblemen zeigen, dass FPSRL in Bezug auf die benötigte Rechenzeit im Vorteil ist, wenn es darum geht Lösungen für eher einfache Probleme, für welche ausreichend Vorwissen über wichtige Zustandsvariablen und einer geschätzten Anzahl an Regeln verfügbar ist, zu finden. Auf der anderen Seite zeigen die Experimente mit einer industriellen Testumgebung, dass FGPRL sowohl die informativsten Zustandsvariablen als auch die kompakteste Regelrepräsentation für eine gegebene Regelungsgüte automatisch finden kann. Die dritte vorgeschlagene interpretierbare Methode heißt *Genetic Programming Reinforcement Learning* (GPRL) und ist schließlich nicht auf die Darstellungsform von Fuzzy-Regel-basierten Aktionsauswahlregeln beschränkt, da sie die Lösungen als einfache algebraische Gleichungen von geringer Komplexität darstellt. Die Experimente mit GPRL zeigen, dass diese Darstellungsart der Aktionsauswahlregeln sowohl menschenverstehbar ist, als auch eine sehr gute Regelungsgüte erzielt. Darüber hinaus erzeugen FGPRL und GPRL nicht nur jeweils eine einzige Lösung pro Durchlauf, sondern eine ganze Pareto-Front über die besten Lösungen für viele verschiedene Komplexitäten.

Die Resultate aller drei interpretierbaren Methoden des bestärkenden Lernens werden verglichen mit den erzielten Gütezahlen eines iterativen neuronalen Q-Lernalgorithmus, einer neuartigen modellprädiktiven Regelung und einer nicht interpretierbaren Aktionsauswahlregel basierend auf künstlichen neuronalen Netzwerken. Dieser umfangreiche Vergleich ermöglicht es, einen Überblick über die Regelungsgüte sowie die Interpretierbarkeit der gelernten Aktionsauswahlregeln zu erhalten. Nichtsdestotrotz bleibt die Entscheidung für die gewünschte Form der interpretierbaren Lösung höchst subjektiv, da sie von vielen Voraussetzungen wie Anwendungsgebiet, der Notwendigkeit zur Visualisierung oder nachgeschalteten Verarbeitungsschritten abhängt. Aus diesem Grund ist es essenziell über Methoden zu verfügen, die automatisch den Suchraum von fachgebietsspezifischen Lösungsrepräsentationen durchsuchen können. Die in dieser Dissertation durchgeführten empirischen Studien zeigen, dass modellbasiertes bestärkendes Lernen in Verbindung mit genetischer Programmierung ein äußerst vielversprechender Ansatz ist, dieses Ziel zu erreichen.

Contents

List of Abbreviations

ALICE	Autonomous learning in complex environments
CPB	Cart-pole balancing
CPSU	Cart-pole swing-up
FGPRL	Fuzzy genetic programming reinforcement learning
FPSRL	Fuzzy particle swarm reinforcement learning
FQI	Fitted Q iteration
GA	Genetic algorithm
GP	Genetic programming
GPRL	Genetic programming reinforcement learning
IB	Industrial benchmark
MC	Mountain car
MDP	Markov decision process
ML	Machine learning
MLP	Multi-layer perceptron
MPC	Model predictive control
NFQ	Neural fitted Q iteration
NN	Neural network
PID	Proportional-integral-derivative
PSO	Particle swarm optimization
PSONN	Particle swarm optimization neural network
PSO-P	Particle swarm optimization policy
RL	Reinforcement learning
RNN	Recurrent neural network

CHAPTER 1

Introduction

In this thesis, three different ways of generating interpretable reinforcement learning (RL) policies are proposed and evaluated. In the scientific area of RL, we are interested in learning a control strategy, called policy, solely based on interaction samples with a dynamical system, called environment. RL as a method from the field of machine learning (ML) uses these interaction samples to automatically learn an optimal policy which maximizes a certain numeric reward signal. The form of representation of the policy can be manifold. Implicit policy representations, for instance, learn a value function or a surrogate model which is exploited during runtime for the best action to be used for the current state. Explicit representations, on the other hand, are calculation procedures that directly compute the next action, based on current state features. The methods proposed in this thesis belong to the explicit representation category, as this kind of policy is expected to produce solutions that are more conveniently interpretable.

The search for interpretable RL policies is of high academic and industrial interest. Interpretability is defined as the ability to explain or to present in understandable terms to a human. In the context of ML, Doshi-Velez and Kim argue that the need for interpretability stems from an incompleteness in the problem formalization, creating a fundamental barrier to optimization and evaluation [30]. Since complex real-world tasks in industry are almost never completely testable, enumerating all possible outputs given all possible inputs is infeasible. Hence, we are usually unable to flag all undesirable outputs. Especially for industrial systems, domain experts are more likely to deploy autonomously learned controllers if they are understandable and convenient to assess. Moreover, legal frameworks such as the European Union's General Data Protection Regulation [114] enforce interpretability of data processing systems by stating in Article 13:

"[...] the controller shall, at the time when personal data are obtained, provide the data subject with the following [...]: the existence of automated decision-making, [...] and [...] meaningful information about the logic involved [...]."

Selbst and Powles state that the *right to explanation* debate has drawn keen interest from the technical community because it is knotty, complex, and a non-trivial technical challenge to harness the full power of ML systems while operating with logic interpretable to humans [135]. In Section 1.1, interpretable ML is formalized and a taxonomy of evaluation approaches for interpretability is presented.

To evaluate the interpretable RL approaches of this thesis, tests on four control benchmarks have been conducted. Since we are specifically interested in interpretable solutions, not only the control performances are investigated, but also the representations' ability to explain or to present in understandable terms. The first three benchmarks are the mountain car (MC) problem, the cart-pole balancing (CPB) task, and the so-called cart-pole swing-up (CPSU) problem. All of them are widely known RL benchmarks, selected to demonstrate the methods' performances on tasks where they could be easily compared with other RL methods. However, all of these problems have a low-dimensional state space, and their dynamics are deterministic and smooth. To investigate the methods' performances on real-world industry applications, a new RL testbed, the industrial benchmark (IB), has been developed and published as a part of the doctoral research. This benchmark combines a high-dimensional state space with stochastic dynamics, thus making its results increasingly meaningful for industrial applications, such as controlling wind and gas turbines.

At first, a controller class from classical control theory, namely proportional–integral–derivative (PID) controllers, is applied to yield interpretable controllers for CPB. Especially in industry settings, PID control is still a very important and common strategy to realize stable, efficient, and understandable control strategies. However, PID control is quite limited in its applicability to more complex high-level control problems, which makes it rather unsuitable for the rest of the benchmark problems. RL on the other hand, is theoretically capable of solving any Markov decision process (MDP) which makes it applicable to all of the considered tasks.

Experiments with a standard RL technique, called neural fitted Q iteration (NFQ), show that the model-free approach has problems learning well-performing policies solely from previously generated data batches. However, using surrogate models to test and select a policy candidate during the NFQ training dramatically improves the performance of this approach. Nevertheless, especially for the CPSU and the IB, the control performance is still not satisfactory. To evaluate the quality of the surrogate models, another RL approach which has been developed as part of the doctoral research, called particle swarm optimization policy (PSO-P), is applied. PSO-P performs a population-based stochastic optimization on action trajectories on a system model with the goal of finding optimal control strategies. Like with NFQ, the policy representation is implicit, because for every given state a new trajectory optimization is required to evaluate the best action. This action is applied to the environment, and the optimization is repeated for the following system state. Performing both, the trajectory optimization as well as the application of the best action on surrogate models, can be used to exploit the models for trajectories yielding the highest accumulated reward. By investigating the trajectories created in this way, one can check whether (i) the surrogate models show the desired realistic behavior in mimicking the system dynamics and (ii) the reward definition encourages the learning process to produce reasonable trajectories. As the first explicit policy representation, a neural network (NN) policy with weights trained by particle swarm optimization (PSO), called particle swarm optimization neural network (PSONN), is conducted. The results verify that it is possible to find explicit policies of high control performance by conducting model-based batch RL. Since NNs are used, PSONN does not yield interpretable policies, because it consists of thousands of joined and cascaded non-linear equations. However, the experiments show that if sufficient degrees of freedom are available for the policy representation, good policies are achievable for the benchmarks, surrogate models, and RL parameters used.

To achieve interpretable RL policy learning, this thesis contributes three novel model-based RL approaches utilizing two optimization methods from the research area of evolutionary computation, namely PSO and genetic programming (GP).

The first interpretable policy RL method presented in this thesis is called fuzzy particle swarm reinforcement learning (FPSRL). In FPSRL the parameters of fuzzy rule-based controllers are optimized by PSO using surrogate models and model-based RL to compute fitness values for particles. The rule-based representation is used because fuzzy controllers are known to serve as interpretable and efficient system controllers for continuous state and action spaces in industry for decades. Experiments on the benchmarks show that FPSRL can learn policies of high performance, provided that expert knowledge about the required rule number and essential state features is available or can be determined by additional heuristics.

However, performing initially a heuristic feature selection and subsequently creating policy structures manually, is a feasible but limited approach; in high-dimensional state and action spaces, the effort involved grows exponentially. This procedure leads to the second interpretable RL approach called fuzzy genetic programming reinforcement learning (FGPRL). By creating fuzzy rules using GP rather than tuning the fuzzy rule parameters via PSO, FGPRL eliminates the manual feature selection process. This GP technique can automatically select the most informative features as well as the most compact fuzzy rule representation for a certain level of performance. Moreover, it returns not just one solution to the problem but a whole Pareto front containing the best-performing solutions for many different levels of complexity. Experiments show that, given sufficient runtime, FGPRL offers considerable advantages over FPSRL in finding compact fuzzy controllers of high performance. Especially for problems with a high-dimensional state space, like IB, the results are superior.

Furthermore, GP is not limited to creating fuzzy rule-based solutions, in addition to that, it is a technique, which can generally encode computer programs. Since for some problems the policy representation as a set of fuzzy rules might be unfavorable by some domain experts, the third interpretable RL approach, called genetic programming reinforcement learning (GPRL), learns policy representations which are represented by basic algebraic equations of low complexity. This model-based approach is compared to a straight-forward method which utilizes GP for symbolic regression, yielding policies imitating an existing well-performing yet non-interpretable policy. The experiments clearly show that while for more simple benchmarks like MC or CPB the regression approach performs reasonably, this approach fails for CPSU and IB problems.

Comparing the results from experiments with all three interpretable RL approaches, FPSRL, FGPRL, and GPRL, to PID, NFQ, PSO-P, and PSONN on four different RL benchmarks, MC, CPB, CPSU, and IB, gives a comprehensive overview of the performance of the methods as well as the interpretability of the policies produced. The results show that choosing the most interpretable form of presentation is highly subjective and depends on many prerequisites, such as the application domain, the ability to visualize solutions, or successive processing steps. However, in all our benchmark experiments, the interpretable approaches yielded performance values on the same level as the non-interpretable methods.

1.1 Interpretable Machine Learning

Recently, Doshi-Velez and Kim [30] published a first approach on how a rigorous science of interpretable ML can be structured. They discuss what interpretability is in contrast to other criteria, consider scenarios in which interpretability is needed, and eventually propose a taxonomy for the evaluation of interpretability. In this section, the findings of their work are presented, and the interpretable RL methods developed in this thesis are positioned in their taxonomy of interpretability.

Merriam-Webster's dictionary[1] describes the verb *to interpret* as *to explain or tell the meaning of* and *to present in understandable terms*. According to cognitive psychologist Tania Lombrozo from Berkeley University of California explanations are "... more than a human preoccupation – they are central to our sense of understanding, and the

[1]Merriam-Webster dictionary, accessed 2018-06-30

currency in which we exchange beliefs" [88]. Based on these statements we can infer that interpretable ML methods have the task of presenting the reasons for their decisions in a human-understandable way.

Despite the high academic and industrial interest in interpretable ML, only a few attempts on thoroughly structuring the different meanings and perspectives on the term *interpretability* exist. However, a structured approach seems to be a key element in the scientification of interpretable ML.

The evaluation of interpretability usually can be assigned to one of two categories. The first category represents the attempt of interpreting any useful system, even if it inherently is a black box system. Examples which fall in this category are the use of explanation techniques for classifiers [115] or visualization methodologies for NNs [5]. However, the interpretable RL approaches of this thesis fall into the second category, in which interpretability is evaluated via a quantifiable proxy. Here, at first, a model class, e.g., fuzzy rules, algebraic equations, or high-level task descriptions, are claimed of being interpretable and then algorithms are developed in order to optimize within this class. Examples for this category are Bayesian rule sets for interpretable classification [156], the search for RL policies in the space of simple closed-form formulas [91], or specifying a user-centric interpretable robot programming paradigm [106]. For both classes, a fundamental problem remains: it is hard to compare methods based on evidence if the notion for interpretability is "you'll know it when you see it."

Doshi-Velez and Kim [30] argue that the need for interpretability stems from an incompleteness in the problem formalization, which creates a fundamental barrier to optimization and evaluation. A prominent example of such incompleteness concerning this thesis is safety. That is because for real-world industry problems, dynamical systems are almost never completely testable. Hence, we cannot flag all undesirable outputs, and thus we cannot completely test all possible scenarios. Another example for incompleteness in industry settings is the problem of stating the correct optimization objective. Experience shows that formally successfully optimized solutions sometimes show unexpected or even unwanted behavior. The reason for that can be an unforeseen loophole in the objective function formulation. The ML method found this loophole and exploited it to maximize its objective even further. It is expected to be easier to identify such unwanted solutions if the solution's representation is interpretable.

In [30] a taxonomy of evaluation approaches for interpretability is proposed:

- Application-grounded: Real humans, real tasks
  Application-grounded evaluation is about testing the interpretability of a system by performing evaluations in cooperation with domain experts on real tasks for which the system has been designed for. Note that the methods from the scientific field of human-computer interaction and visualization already provide a wide range of methods and approaches which can be of high utility for this kind of evaluation.

- Human-grounded: Real humans, simplified tasks
  Human-grounded evaluation should be applied if experiments with the target

community are too expensive. By performing the experiments with non-experts, more general notions of the quality of interpretability can be tested. The tasks are usually more abstract and designed to make more general statements, like reasoning about the overall task complexity.

- Functionally-grounded: No humans, proxy tasks
  Functionally-grounded evaluation uses some formal definition of interpretability. Once a class of models or regularizers has been evaluated, e.g., via human-grounded experiments, the performance can be improved, or the sparsity can be increased.

Since application- and human-grounded evaluations both involve real humans, methods from these categories are costly in terms of preparation, time effort, and financial compensation. Moreover, assessing the evaluated results from such methods is prone to biases and misunderstandings. The scientific field of human-computer interaction provides numerous principles and methodologies to conduct research with respect to interpretable human-computer interfaces [29]. The focus of the proposed methods of this thesis lies on providing algorithms that are able to create preexisting representations, namely rule-based and equation-based RL policies, without reevaluating their general interpretability by experiments with real humans. Therefore, the proposed methods can be categorized as candidates for functionally-grounded evaluation. Both, rule-based as well as algebraic-equation-based representations, have been evaluated by application- and human-grounded evaluations in previous publications and real-world applications. The experiments performed to evaluate the performance of the novel methods of this thesis are conducted on proxy tasks, i.e., benchmark dynamics, and their quality is compared performance- and sparsity-wise, in terms of fitness and complexity, respectively.

1.2 Contributions

Three novel algorithms for generating interpretable policies in model-based batch RL using PSO and GP have been developed during this research:

- FPSRL [A] and FGPRL [B] to generate rule-based policies,
- GPRL [C] to generate equation-based policies.

A novel feature selection approach called PSO-P [D,E] is proposed, which uses swarm intelligence to identify the most informative state features with respect to the optimal actions for a specific benchmark, without the need to specify the type of policy.

An RL benchmark called IB [F,G] is introduced, that possesses typical properties found in real-world applications such as gas turbines and wind turbines: high-dimensional continuous state space, multi-dimensional actions, non-linear stochastic dynamics, heteroscedastic noise, partial observability, delayed reward, and multiple contradictory objectives.

The interpretable policies from FPSRL, FGPRL, and GPRL show performance comparable to, or even surpassing that of NFQ generated non-interpretable policies for a variety of different reinforcement learning benchmark problems (MC, CPB, CPSU, IB).

[A] D. Hein, A. Hentschel, T. A. Runkler, and S. Udluft. “Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies.” In: Engineering Applications of Artificial Intelligence 65 (2017), pp. 87–98. [62]

[B] D. Hein, S. Udluft, and T. A. Runkler. 2018. “Generating interpretable fuzzy controllers using particle swarm optimization and genetic programming.” In GECCO ’18 Companion: Proceedings of the 2018 Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan. ACM, New York, NY, USA, pp. 1268-1275. [66]

[C] D. Hein, T. A. Runkler, and S. Udluft. “Interpretable policies for reinforcement learning by genetic programming.” In: Engineering Applications of Artificial Intelligence 76 (2018), pp. 158-169. [67]

[D] D. Hein, A. Hentschel, T. A. Runkler, and S. Udluft. “Reinforcement learning with particle swarm optimization policy (PSO-P) in continuous state and action spaces.” In: International Journal of Swarm Intelligence Research (IJSIR) 7.3 (2016), pp. 23–42. [64]

[E] D. Hein, A. Hentschel, T. A. Runkler, and S. Udluft. “Particle swarm optimization for model predictive control in reinforcement learning environments.” In: Critical Developments and Applications of Swarm Intelligence. Ed. by Y. Shi. Hershey, PA, USA: IGI Global, 2018. Chap. 16, pp. 401–427. [63]

[F] D. Hein, S. Udluft, M. Tokic, A. Hentschel, T. A. Runkler, and V. Sterzing. “Batch reinforcement learning on the industrial benchmark: First experiences.” In: 2017 International Joint Conference on Neural Networks (IJCNN). 2017, pp. 4214–4221. [65]

[G] D. Hein, S. Depeweg, M. Tokic, S. Udluft, A. Hentschel, T. A. Runkler, and V. Sterzing. “A benchmark environment motivated by industrial control problems.” In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI). 2017, pp. 1–8. [61]

1.3 Outline of this Thesis

The content of this thesis is structured as follows: (i) Chapter 2 introduces the theoretical concepts of RL, PSO, and GP which are later applied to yield interpretable RL policies. Specifically, the use of model-based policy search as the main RL approach is motivated and formalized. (ii) In Chapter 3, the benchmarks of this work are explained, including the IB which is a novel testbed for industrial applications. Furthermore, it is described how the building process of adequate surrogate models for model-based RL is conducted. (iii) In Chapters 4-5, the surrogate models are used to learn control policies. In a first

	Model-free	Model-based	Explicit policy	Interpr. rules	Interpr. equations	Chapter
NFQ	✓					5.1
PSO-P		✓				5.2
PSONN		✓	✓			5.3
FPSRL		✓	✓	✓		6
FGPRL		✓	✓	✓		7
GPRL		✓	✓		✓	8

Table 1.1: The six investigated RL methods and their embedding into this thesis

step, gains for an interpretable PID controller are derived from empirical studies using the system models. Subsequently, non-interpretable RL policies are learned by applying NFQ, PSO-P, and PSONN. PSO-P is the second contribution of this study, and it is both an MPC-like RL policy as well as a tool for exploiting surrogate models for optimal trajectories without the requirement of predefining any specific policy form. (iv) Chapters 6-9 contain the main contributions of this thesis, which are three methods for learning interpretable RL policies by using evolutionary computation. The three methods, FPSRL, FGPRL, and GPRL, are evaluated on the same benchmarks and compared by means of setup requirements, learning behavior, performance results, and interpretability of the resulting policies. (v) Chapter 10 concludes this work by wrapping up the achieved objectives and giving an outlook on how to forward the application of interpretable RL.

Fig. 1.1 shows an overview of the compared RL approaches of this thesis. The six different methods are depicted by compact diagrams which focus on their core methods. Note that the diagrams are ordered by the form of their policy representation from implicit & non-interpretable to explicit & interpretable. NFQ learns a value function for each problem and PSO-P optimizes action trajectories for a given world model, whereas PSONN, FPSRL, FGPRL, and GPRL perform model-based policy search with PSO or GP in order to find optimal policy parameters. The structural similarity between the training process of the latter four methods is easily recognizable in Fig. 1.1. However, only FPSRL, FGPRL, and GPRL are specifically designed to search the spaces of interpretable policies. It is important to notice that all six methods utilize the very same transition sample data batches. Table. 1.1 outlines important properties of the six RL methods and shows how they are embedded into this thesis.

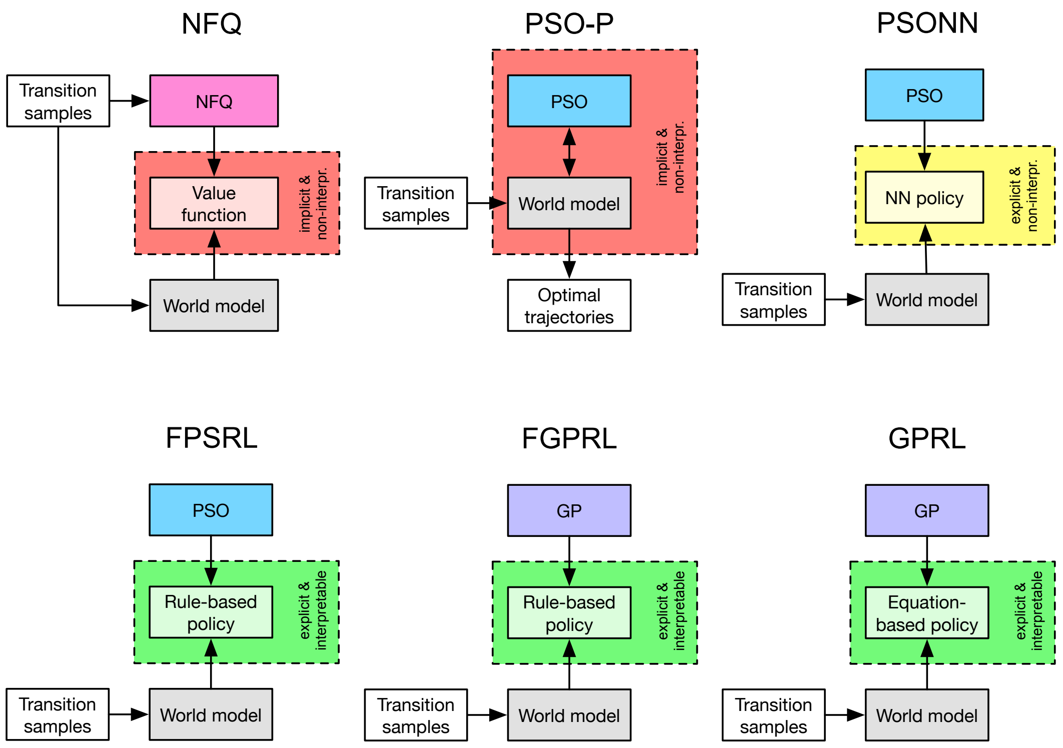


Figure 1.1: Overview of the compared RL methods. The novel interpretable approaches FPSRL, FGPRL, and GPRL are compared to standard NFQ, the on-demand model-exploiting PSO-P approach, and a model-based version of PSONN.

CHAPTER 2

Outline of Applied Methodologies

In this chapter, the methodologies applied in the course of this thesis are motivated and established. First, in Section 2.1 reinforcement learning (RL) and its variants are introduced. Model-based batch RL, policy search, and approximate RL as core concepts of the novel RL methods of this doctoral study are presented in detail. Second, in Section 2.2 particle swarm optimization (PSO) as one of two methods from evolutionary computation is outlined. Subsequently, genetic programming (GP), the second population-based optimization technique, is introduced in Section 2.3

2.1 Reinforcement Learning

In biological learning, an animal interacts with its environment and attempts to find action strategies to maximize its perceived accumulated reward. This notion is formalized in RL, an area of machine learning (ML) where the acting agent is not explicitly told which actions to implement. Instead, the agent must learn the best action strategy from the observed environment's responses to the agent's actions. For the most common (and most challenging) RL problems, an action affects both the next reward and subsequent rewards. Examples for such delayed effects are non-linear changes in position when a force is applied to a body with mass or delayed heating in a combustion engine. Both the trial-and-error search and the delayed reward signal are the most distinguishing properties with respect to other optimal control approaches. [147]

The problem of RL is formalized by using ideas from the theory of optimal control of incompletely-known *Markov decision processes* (MDPs). This means that to solve a real problem successfully, the learning agent must be able to sense the state of its environment and must be able to take actions in order to affect subsequent states. In addition to sensing and action, the agent must have a goal relating to the state of the environment. Any method that can solve such MDPs is considered an RL method.

RL is a method from the field of ML. Generally, ML methods can be assigned to the two major subcategories *supervised ML* and *unsupervised ML*. Supervised learning is learning from a training set of labeled examples, whereas unsupervised learning is about finding structure hidden in collections of unlabeled data. RL does not rely on correctly labeled training data which represents the correct behavior the agent has to learn. Consequently, it is not a method of supervised learning. However, RL also is not about finding hidden structures in data, which distinguishes it from unsupervised learning methods. Therefore, RL is considered to be a third ML paradigm.

The four subelements of RL are a policy π, a reward signal r, a value function v, and, optionally, a model $\tilde{g}$ of the environment g. A policy is a mapping from perceived states of the environment to actions to be taken when in those states, i.e., $\pi(\boldsymbol{s}) = \boldsymbol{a}$. Possible policy representations range from lookup tables over simple functions to extensive search processes on system models. The goal in RL is defined by a reward signal, which defines what good and bad events for the agent are. The *reward*, represented by a single number, is sent from the environment to the agent on each time step. Maximizing the total reward the agent receives over the long run is its sole objective. What is good in the long run is specified by a value function. A value of a state is defined as the total amount of reward an agent can expect to achieve over the future, starting from that state. Consequently, values indicate the long-term desirability of states, thus it is values which we are most concerned with when making and evaluating decisions. Rewards are given directly by the environment, whereas values must be estimated and re-estimated from sequences of observations. Hence, an essential component of RL algorithms is a method for efficiently estimating values. The fourth sub-element of RL is a model of the environment. Such a model allows inferences to be made about how the environment will behave by mimicking its behavior. RL methods that use a model for planning are called *model-based* methods.

Learning from interactions to achieve a goal can be depicted by the so-called *agent-environment interface* (Fig. 2.1). In this framework, the learner and decision maker is called the *agent* and the object of interaction is called the *environment*. Note that the term environment denotes everything outside of the agent that cannot be changed arbitrarily by the agent. Agent and environment interact continually by selecting actions (agent) and responding with new situations and rewards (environment). If in an MPD, a state includes all information about all aspects of the past action-environment interaction that make a difference for the future, then that state is said to have the *Markov property*.

The agent interacts with the environment in discrete time steps, $t = 0, 1, 2, \ldots$. At each time step, the agent observes the system's state $\boldsymbol{s}_t \in \mathcal{S}$ and applies an action $\boldsymbol{a}_t \in \mathcal{A}$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. Depending on $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$, the system transitions to a new state and the agent receives a real-value reward $r_{t+1} \in \mathbb{R}$. Herein, we focus on deterministic systems where state transition g and reward r can be expressed as functions $g : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ with $g(\boldsymbol{s}_t, \boldsymbol{a}_t) = \boldsymbol{s}_{t+1}$ and $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ with $r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) = r_{t+1}$, respectively. The continual interaction of agent and environment

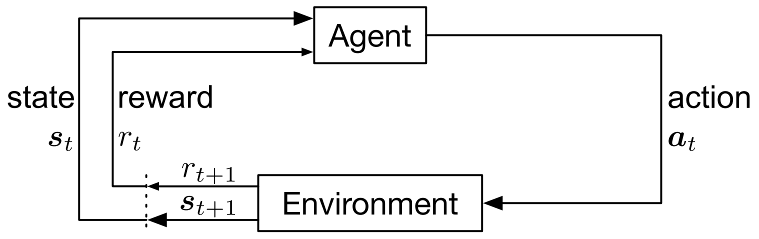


Figure 2.1: The agent-environment interface

give rise to a trajectory $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, \ldots$.

The agent's goal is to maximize the cumulative reward it receives in the long run starting from the current time step t. In the simplest case the return $\mathcal{R}_t$ we seek to maximize is the sum of the rewards:

$$\mathcal{R}_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T, \tag{2.1}$$

where T is the final time step. For problems which do not have a final time step, i.e., $T = \infty$, the expected *discounted return* is maximized:

$$\mathcal{R}_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.2}$$

with discount factor $0 \leq \gamma \leq 1$. Note that if $\gamma < 1$, the infinite sum in (2.2) has a finite value (as long as the rewards r_k are bounded). For example, if the reward is bounded by $0 \leq r_k \leq \alpha$, then the return has the upper bound

$$\mathcal{R}_t \leq \sum_{k=0}^{\infty} \gamma^k \alpha = \frac{\alpha}{1-\gamma}. \tag{2.3}$$

A very important property of the relation between returns at successive time steps is that $\mathcal{R}_t$ can be expressed in terms of $\mathcal{R}_{t+1}$ as follows:

$$\begin{aligned} \mathcal{R}_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots \\ &= r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \ldots) \\ &= r_{t+1} + \gamma \mathcal{R}_{t+1}. \end{aligned} \tag{2.4}$$

A central point in RL involves estimating *value functions*. These are functions that estimate *how good* it is to be in a given state, in terms of expected return, with respect to particular ways of acting. Such ways of acting are called *policies* in RL. A deterministic policy π is a mapping from states to actions, i.e., $\pi(s) = a$. Hence, the value function of state s under policy π is denoted $v_\pi(s)$ and calculated as

$$\begin{aligned} v_\pi(s_t) &= \sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, \pi(s_{t+k}), s_{t+k+1}), \\ \text{with} \quad s_{t+k+1} &= g(s_{t+k}, a_{t+k}). \end{aligned} \tag{2.5}$$

Analogously, the *action-value function q* is defined for π:

$$q_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_{t+k}, \boldsymbol{a}_t, \boldsymbol{s}_{t+k+1}) + \sum_{k=1}^{\infty} \gamma^k r(\boldsymbol{s}_{t+k}, \pi(\boldsymbol{s}_{t+k}), \boldsymbol{s}_{t+k+1}). \tag{2.6}$$

Note that the action-value function estimates the return of applying action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$ and following policy π afterwards. Both, v_π and q_π, can be estimated from experience with the environment and stored in tables or represented by parameterized functions. Since the value and action-value functions from (2.5) and (2.6) satisfy the same recursive relationship as the return from (2.4), we can reformulate them as follows:

$$v_\pi(\boldsymbol{s}_t) = r(\boldsymbol{s}_t, \pi(\boldsymbol{s}_t), \boldsymbol{s}_{t+1}) + \gamma v_\pi(\boldsymbol{s}_{t+1}), \tag{2.7}$$
$$q_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) + \gamma q_\pi(\boldsymbol{s}_{t+1}, \pi(\boldsymbol{s}_{t+k})). \tag{2.8}$$

Equation (2.7) is known as the *Bellman equation* for deterministic MDPs.

An *optimal policy* is defined as a policy which yields a higher or equal return for all states compared to all other policies, i.e., $\pi \geq \pi'$ if and only if $v_\pi(\boldsymbol{s}) \geq v_{\pi'}(\boldsymbol{s})$ for all $\boldsymbol{s} \in \mathcal{S}$. All optimal policies π^* share the same *optimal state-value function v^**, which is defined as

$$v^*(\boldsymbol{s}_t) = \max_\pi v_\pi(\boldsymbol{s}), \quad \text{for all } \boldsymbol{s} \in \mathcal{S}. \tag{2.9}$$

The *Bellman optimality equation* expresses the value of state $\boldsymbol{s}_t$ yielded by the optimal policy in terms of the action-value function as follows:

$$v^*(\boldsymbol{s}_t) = \max_{\boldsymbol{a}_t \in \mathcal{A}} q_{\pi^*}(\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{2.10}$$

Similarly, the *optimal action-value function* is defined as

$$q^*(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) + \gamma \max_{\boldsymbol{a}_{t+1} \in \mathcal{A}} q^*(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}). \tag{2.11}$$

Note that if an optimal action-value function is available, a greedy one-step search would be sufficient to find the best action with respect to the long term return. In this case, absolutely no knowledge about the environment's dynamics is needed to select the best actions for any given state.

2.1.1 Model-based and Model-free Policy Evaluation in Batch Reinforcement Learning

Generally, 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, the cost of interacting with the environment on a trial-and-error basis is prohibitive. 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 systems like power plants or vehicles. The potentially caused damage in the first learning phase certainly would nullify all possible future improvements. For this reason, *offline learning* is a more suitable approach for applying RL methods on already collected training data and, if proving useful, existing models, to yield RL policies. By using offline learning, the environment exploration, the process of learning a policy, as well as the evaluation of a solution can be conducted without any risk for the real environment. Hence, offline RL methods are favored by industry.

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. Hence, this data has to include an adequate amount of exploration since during training no additional system exploration can be conducted. The batch data set contains transition tuples of the form $(\mathfrak{s}_t, \mathfrak{a}_t, \mathfrak{s}_{t+1}, \mathfrak{r}_{t+1})$, where the application of action $\mathfrak{a}_t$ in state $\mathfrak{s}_t$ resulted in a transition to state $\mathfrak{s}_{t+1}$ and yielded a reward $\mathfrak{r}_{t+1}$. By applying a batch-mode RL algorithm called *fitted Q iteration* (FQI), which is based on *fitted value iteration* [53], an action-value function q_π can be estimated from the available data batch. FQI is an algorithm which uses the fixed point property of (2.8) to perform an *iterative policy evaluation*. Let's denote the initial action-value function of policy π with q_0 and choose its initial approximation arbitrarily. We now can compute each successive approximation by using the Bellman equation for (2.8) as an update rule:

$$q_{k+1}(\mathfrak{s}_t, \mathfrak{a}_t) \leftarrow \mathfrak{r}_{t+1} + \gamma q_k(\mathfrak{s}_{t+1}, \pi(\mathfrak{s}_{t+1})), \tag{2.12}$$

for all transition tuples in the data batch. Depending on the applied function approximation for q, this algorithm is expected to converge to q_π as $k \to \infty$. Generally, convergence can be proven under certain conditions for finite MPDs [147]. To rate the performance of policy π, the final action-value function is used as follows:

$$v_\pi(\boldsymbol{s}) = q_\pi(\boldsymbol{s}, \pi(\boldsymbol{s})), \tag{2.13}$$

$$\overline{\mathcal{R}}_\pi = \frac{1}{|\mathcal{S}|} \sum_{\boldsymbol{s} \in \mathcal{S}} v_\pi(\boldsymbol{s}), \tag{2.14}$$

with $\overline{\mathcal{R}}_\pi$ the average discounted return of π.

Another way of evaluating a policy offline from a batch of transition samples is referred to as *model-based value estimation*. In the first step, supervised ML is applied to learn approximate models of the underlying environment from transition tuples $(\mathfrak{s}_t, \mathfrak{a}_t, \mathfrak{s}_{t+1}, \mathfrak{r}_{t+1})$ as follows:

$$\tilde{g}(\mathfrak{s}_t, \mathfrak{a}_t) \leftarrow \mathfrak{s}_{t+1}, \quad \tilde{r}(\mathfrak{s}_t, \mathfrak{a}_t, \mathfrak{s}_{t+1}) \leftarrow \mathfrak{r}_{t+1}. \tag{2.15}$$

Using models $\tilde{g}$ and $\tilde{r}$, the value for policy π of each state $\mathfrak{s}$ in the data batch can be estimated using (2.5) to yield $\tilde{v}_\pi(\boldsymbol{s})$. Hence, using model-based value estimation means

$$\begin{tikzcd}\pi_0 \arrow[r, "\mathrm{E}"] & v_{\pi_0} \arrow[r, "\mathrm{I}"] & \pi_1 \arrow[r, "\mathrm{E}"] & v_{\pi_1} \arrow[r, "\mathrm{I}"] & \pi_2 \arrow[r, "\mathrm{E}"] & v_{\pi_2} \arrow[r, "\mathrm{I}"] & \cdots\end{tikzcd}$$

Figure 2.2: Standard policy iteration

$$\begin{tikzcd}\pi_0 \arrow[r, "\mathrm{E}"] & \overline{\mathcal{R}}_{\pi_0} \arrow[r, "\mathrm{I}"] & \pi_1 \arrow[r, "\mathrm{E}"] & \overline{\mathcal{R}}_{\pi_1} \arrow[r, "\mathrm{I}"] & \pi_2 \arrow[r, "\mathrm{E}"] & \overline{\mathcal{R}}_{\pi_2} \arrow[r, "\mathrm{I}"] & \cdots\end{tikzcd}$$

Figure 2.3: Standard policy search

performing trajectory rollouts on system models for different starting states:

$$\begin{aligned}\tilde{v}_\pi(\boldsymbol{s}_t) &= \sum_{k=0}^{\infty} \gamma^k \tilde{r}(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}),\\ \text{with} \quad \boldsymbol{s}_{t+k+1} &= \tilde{g}(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}) \quad \text{and} \quad \boldsymbol{a}_{t+k} = \pi(\boldsymbol{s}_{t+k}).\end{aligned} \tag{2.16}$$

Equivalent to the model-free value estimation the performance of a policy π is rated by summing over the estimated values for states $\boldsymbol{s}$:

$$\overline{\mathcal{R}}_\pi = \frac{1}{|\mathcal{S}|} \sum_{\boldsymbol{s} \in \mathcal{S}} \tilde{v}_\pi(\boldsymbol{s}). \tag{2.17}$$

Both, model-free and model-based value estimation, have their advantages and disadvantages. For problems with rather smooth and easy to approximate system dynamics the model-based approach has shown to be superior in terms of run time and stability. On the other hand, if the information contained in the available data batch is not sufficient to learn adequate models of the environment, the model-free approach is expected to yield better results.

2.1.2 Policy Iteration and Policy Search

Since the purpose of policy evaluation is to find better policies, the estimated value (by model-free or model-based approaches) is used in a subsequent *policy improvement* step. Once the improved policy π_{l+1} has been learned from v_{π_l}, the new value function $v_{\pi_{l+1}}$ can be computed and used to yield an even better policy π_{l+2}. This sequence of improving policies and computing their respective value functions is called *policy iteration*. Fig. 2.2 depicts this iterative process where $\xrightarrow{E}$ denotes a policy evaluation and $\xrightarrow{I}$ denotes a policy improvement step.

Policy search is very similar to policy iteration. The main difference is that in policy iteration, the improved policy fully maximizes its value function (v_π) over the action variables, while policy search updates the policy parameters in order to increase the expected average return $\left(\overline{\mathcal{R}}_\pi\right)$. Fig. 2.3 depicts this iterative process of policy search. In this thesis, the interest lies in populations of policies, which means there exist multiple different policies at the same time in one policy iteration step (Fig. 2.4). Note that despite every policy yielding its own average return value, each individual in the population

$$\begin{tikzcd}
\pi_{0,1} \arrow[r, "E"] & \overline{\mathcal{R}}_{\pi_{0,1}} \arrow[r, "I"] & \pi_{1,1} \arrow[r, "E"] & \overline{\mathcal{R}}_{\pi_{1,1}} \arrow[r, "I"] & \pi_{2,1} \arrow[r, "E"] & \overline{\mathcal{R}}_{\pi_{2,1}} \arrow[r, "I"] & \cdots \\
\pi_{0,2} \arrow[r] & \overline{\mathcal{R}}_{\pi_{0,2}} \arrow[r] & \pi_{1,2} \arrow[r] & \overline{\mathcal{R}}_{\pi_{1,2}} \arrow[r] & \pi_{2,2} \arrow[r] & \overline{\mathcal{R}}_{\pi_{2,2}} \arrow[r] & \cdots \\
\pi_{0,3} \arrow[r] & \overline{\mathcal{R}}_{\pi_{0,3}} \arrow[r] & \pi_{1,3} \arrow[r] & \overline{\mathcal{R}}_{\pi_{1,3}} \arrow[r] & \pi_{2,3} \arrow[r] & \overline{\mathcal{R}}_{\pi_{2,3}} \arrow[r] & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\pi_{0,N} \arrow[r] & \overline{\mathcal{R}}_{\pi_{0,N}} \arrow[r] & \pi_{1,N} \arrow[r] & \overline{\mathcal{R}}_{\pi_{1,N}} \arrow[r] & \pi_{2,N} \arrow[r] & \overline{\mathcal{R}}_{\pi_{2,N}} \arrow[r] & \cdots
\end{tikzcd}$$

Figure 2.4: Population-based policy search

can transfer knowledge from every other policy evaluation. Applying population-based evolutionary methods to policy search has already been successfully achieved in the past. Chin and A. A. Jafari applied genetic algorithms for generating real source code for solving a standard finite MDP [20]. Barash compared a genetic-algorithms-based methodology with policy iteration for solving MDPs on a model problem with small state and action spaces [7]. In [52], a novel neuroevolution method called CoSyNE that evolves networks at the level of weights has been introduced. CoSyNE has been tested and compared against other RL methods on difficult versions of the pole-balancing problem that involve large state spaces and hidden state. Chang, Hu, Fu, and Marcus introduced evolutionary policy iteration and evolutionary random policy search, and used it in a model-based setting by approximating the infinite horizon value function (model-free) by a finite horizon value function (model-based) [17]. Recently, Wilson, Cussat-Blanc, Luga, and J. F. Miller applied mixed type cartesian GP to Atari playing [157]. They showed that by evaluating the programs of the most evolved individuals, simple but effective strategies could be found.

2.1.3 Value Iteration

Applying policy iteration as well as policy search in batch-based RL settings can be computationally expensive since every policy evaluation step requires multiple sweeps through the data batch until it converges. In the algorithm called *value iteration*, the policy evaluation step is stopped after just one update from each transition sample. This allows us to rewrite the update step from (2.12) to an operation that combines the policy improvement and the policy evaluation in one step:

$$q_{k+1}(\mathfrak{s}_t, \mathfrak{a}_t) \leftarrow \mathfrak{r}_{t+1} + \gamma \max_{\boldsymbol{a}_{t+1}} q_k(\mathfrak{s}_{t+1}, \boldsymbol{a}_{t+1}). \tag{2.18}$$

Note that no explicit policy is applied, instead the maximum over all actions is computed. Consequently, value iteration does not aim to converge to a specific v_π but rather learns v^*, the value function of the optimal policy, defined by

$$v^*(\boldsymbol{s}) = \max_{\boldsymbol{a}} q(\boldsymbol{s}, \boldsymbol{a}). \tag{2.19}$$

Like policy iteration, value iteration can also be shown to converge to an optimal policy for discounted finite MDPs [147]. In Section 5.1, the value iteration method *neural fitted Q iteration* (NFQ) is introduced and evaluated on four benchmarks.

2.1.4 Approximate Reinforcement Learning

Classical RL requires exact representations of value functions and policies, i.e., distinct estimates of the return for every state-action pair or distinct actions for every state have to be stored. Such exact representations are no longer possible if the state or action spaces are large or even infinite in the continuous case. For this reason, applying approximate representations for value functions or policies is essential. Busoniu, Babuska, De Shutter, and Ernst propose the following taxonomy of approximate RL approaches [14]:

- Approximate value iteration: The value function is approximated and evaluated on demand during runtime to yield optimal actions. NFQ is an example of this class where the value function is approximated by a neural network (NN) (Section 5.1).
- Approximate policy iteration: For the policy evaluation step, an approximation of the policy-specific value function has to be found. Alternatively, model-based approximate policy evaluation with rollouts avoids the often difficult task of finding an appropriate value function approximation. With this approach, the value function is evaluated on demand, by Monte Carlo simulations on an approximate system model. Moreover, if the policy is to be represented explicitly, policy approximation is required, too. Note that it is possible to avoid policy approximation by computing improved actions on demand from the current value function.
- Approximate policy search: The policy is represented approximately. In each search step, the policy is updated in a direction that increases the received average return. Gradient-free policy search performs a global search avoiding getting stuck in local optima even on non-differentiable optimization criterions.

2.1.5 Population-based Policy Search with Model-based Batch Reinforcement Learning

In this section, the model-based RL approach, which is applied by the novel interpretable methods of this thesis, is motivated and introduced. Fig. 2.5 depicts this approach including the main reasons for individual design decisions.

The first requirement the proposed setup has to fulfill, is that only batch RL approaches can be applied (Section 2.1.1). In batch RL, we consider applications where online learning approaches, such as classical temporal-difference learning [146], are prohibited for safety reasons since these require exploration of system dynamics during runtime. In contrast, batch RL algorithms generate a policy based on existing data and deploy this policy to the system after training. In this setting, either the value function or the system dynamics are trained using historic operational data. Research from the past two

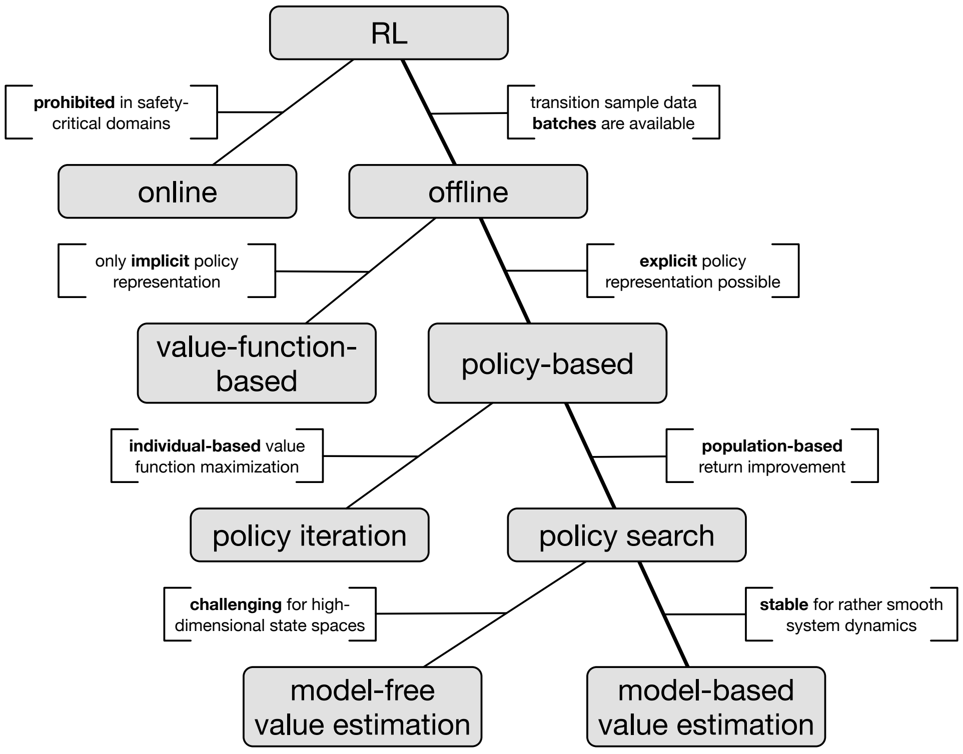

RL
prohibited in safety-critical domains
transition sample data **batches** are available
online
offline
only **implicit** policy representation
explicit policy representation possible
value-function-based
policy-based
individual-based value function maximization
population-based return improvement
policy iteration
policy search
challenging for high-dimensional state spaces
stable for rather smooth system dynamics
model-free value estimation
model-based value estimation

Figure 2.5: The RL approach used in this thesis including the main reasons for individual design decisions

decades [55, 104, 84, 39] suggests that such batch RL algorithms satisfy real-world system requirements, particularly when involving NNs modeling either the state-action value function [116, 117, 131, 130, 120] or the system dynamics [6, 125, 27]. Moreover, batch RL algorithms are data-efficient [116, 128] because batch data is utilized repeatedly during the training phase. Here, model-based value estimation using an approximate surrogate model over model-free value function approximation was chosen since high-dimensional continuous state spaces in combination with multi-dimensional continuous action spaces are expected to impose severe difficulties for approximate value function methods [8, 14].

Generating a world model from real system data in advance and training a policy offline using this model afterwards has several advantages. (i) In many real-world scenarios, data describing system dynamics is available in advance or is easily collected. (ii) Policies are not evaluated on the real system, thereby avoiding the detrimental effects of executing a bad policy. (iii) Expert-driven reward function engineering yielding a closed-form differentiable equation utilized during policy training is not required, i.e., it is sufficient to sample from the system's reward function and model the underlying dependencies using supervised ML.

Since we are searching for interpretable solutions, the resulting policies have to be represented in an explicit form. Consequently, value iteration is unsuitable for achieving this goal. Policy search yields explicit policies which can be enforced to be of an interpretable form (Section 2.1.4). Furthermore, policy search is inherently well suited for being used with population-based optimization techniques, like PSO and GP (Section 2.1.2).

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 $\boldsymbol{x} \in \mathcal{X}$. Herein, a policy corresponding to a particular parameter value $\boldsymbol{x}$ is denoted by $\pi[\boldsymbol{x}]$. For state $\boldsymbol{s}_t$, the policy outputs action $\pi[\boldsymbol{x}](\boldsymbol{s}_t) = \boldsymbol{a}_t$. The policy's performance when starting from $\boldsymbol{s}_t$ is measured by the value function $v_{\pi[\boldsymbol{x}]}(\boldsymbol{s}_t)$. Furthermore, including only a finite number of $T > 1$ future rewards for the model-based value estimation from Eq. 2.16 yields

$$\begin{aligned} v_{\pi[\boldsymbol{x}]}(\boldsymbol{s}_t) &= \sum_{k=0}^{T-1} \gamma^k r(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}), \\ \text{with} \quad \boldsymbol{s}_{t+k+1} &= g(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}) \quad \text{and} \quad \boldsymbol{a}_{t+k} = \pi[\boldsymbol{x}](\boldsymbol{s}_{t+k}). \end{aligned} \tag{2.20}$$

The discount factor γ is selected such that, at the end of the time horizon T, the last reward accounted for is weighted by $q \in [0, 1]$, yielding $\gamma = q^{1/(T-1)}$. The overall state-independent policy performance is obtained by calculating the estimated discounted average return $\overline{\mathcal{R}}_{\pi[\boldsymbol{x}]}$. Thus, optimal solutions to the RL problem are $\pi[\hat{\boldsymbol{x}}]$ with

$$\hat{\boldsymbol{x}} \in \arg\max_{\boldsymbol{x} \in \mathcal{X}} \overline{\mathcal{R}}_{\pi[\boldsymbol{x}]}, \quad \text{with} \quad \overline{\mathcal{R}}_{\pi[\boldsymbol{x}]} = \frac{1}{|\mathcal{S}|} \sum_{\boldsymbol{s}_t \in \mathcal{S}} v_{\pi[\boldsymbol{x}]}(\boldsymbol{s}_t). \tag{2.21}$$

In optimization terminology, the policy performance function $\overline{\mathcal{R}}_{\pi[\boldsymbol{x}]}$ is referred to as the fitness function $\mathcal{F}(\boldsymbol{x}) = \overline{\mathcal{R}}_{\pi[\boldsymbol{x}]}$.

In this thesis, two different types of interpretable policy representations are examined, rule-based and equation-based policies, represented by a real-valued parameter vector or by a vector containing a number of building blocks of different types which have to be compiled first to yield their respective tree-like GP individual shape. However, both kinds require parameterizing their policies only once at the beginning. During runtime, only the used state features from $\boldsymbol{s}$ are extracted and placed in the membership functions or function trees in $\pi[\boldsymbol{x}](\boldsymbol{s})$, respectively.

2.2 Particle Swarm Optimization

The PSO algorithm is a population-based, non-convex, stochastic optimization heuristic. Generally, PSO can operate on any search space that is a bounded sub-space of a finite-dimensional vector space. PSO is utilized several times in this thesis because it does not require any gradient information about its fitness landscape. Instead, PSO utilizes neighborhood information to systematically search for valuable regions in the search space. Moreover, swarm optimization makes few assumptions about the problem being optimized and can search very large spaces of candidate solutions. [77]

These properties make PSO a well-suited optimization tool to search the space of proportional-integral-derivative (PID) controller gains (Chapter 4), action trajectories (Section 5.2), neural network (NN) policy weights (Section 5.3), and rule-based controller parameters (Section 6). Alternative optimization techniques are gradient-descent based methods or evolutionary algorithms.

The position of each particle in the swarm represents a potential solution of the given problem. The particles fly iteratively through the multi-dimensional search space, which is referred to as the fitness landscape. After each movement, each particle receives a fitness value for its new position. This fitness value is used to update a particle's own velocity vector and the velocity vectors of all particles in a certain neighborhood.

At each iteration p, particle i remembers the best local position $\boldsymbol{y}_i(p)$ it has visited so far (including its current position). Furthermore, particle i knows the neighborhood's best position

$$\hat{\boldsymbol{y}}_i(p) \in \underset{\boldsymbol{z} \in \{\boldsymbol{y}_j(p) | j \in \mathcal{N}_i\}}{\arg\max} \; \mathcal{F}(\boldsymbol{z}), \tag{2.22}$$

found so far by any one particle in its neighborhood $\mathcal{N}_i$ (including itself). Particle i's neighborhood is defined as

$$\mathcal{N}_i = \{i\} \cup \{k \mid \text{node } i \text{ and node } k \text{ are connected in the swarm topology graph}\}. \tag{2.23}$$

The neighborhood relations between particles are determined by the swarm's population topology and are generally fixed, irrespective of the particles' positions. Fig. 2.6 shows examples of three different topologies for the swarm size $N = 7$. It is important to note from Fig. 2.6 that neighborhoods overlap, to ensure that the swarm converges to a single solution. The differences between swarms with highly and sparsely connected

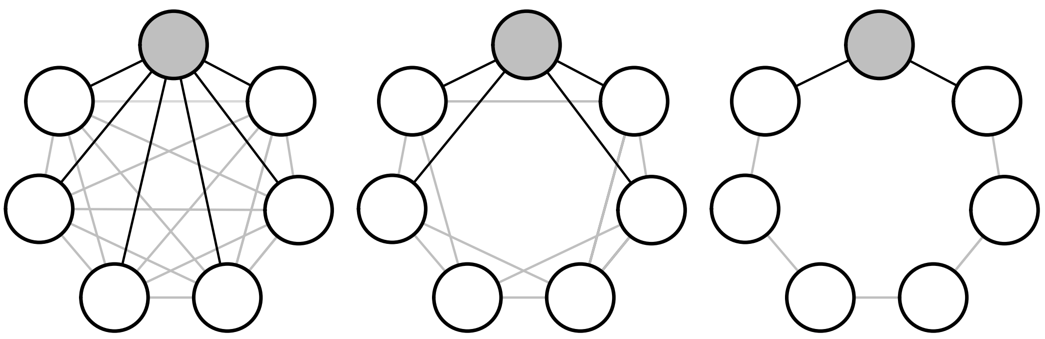

(a) Star topology
(b) Ring with four neighbors per particle
(c) Ring with two neighbors per particle

Figure 2.6: Examples of swarm topology graphs. Each particle is represented by a node in the graph, and two communicating particles are connected with an edge. Subfigure (a) shows a star topology. Each particle is directly connected to all particles of the entire swarm. In (b) every particle has four neighbors, whereas (c) shows a topology with particles with only two neighbors. In each of the diagrams one particle and its neighborhood is highlighted.

topologies lie in their convergence characteristics. The bigger the neighborhood of each particle is, the faster the swarm converges to a local maximum found early during the search process. So the fully connected PSO version converges faster and performs the least exploitation of the search space. However, this comes at the cost of less diversity. More complex topologies and their effects on the PSO algorithm are described in [37].

Let $\boldsymbol{x}_i(p)$ denote the position of particle i at iteration p. Changing the position of a particle in each iteration is achieved by adding the velocity vector $\boldsymbol{v}_i(p+1)$ to the particles position vector

$$\boldsymbol{x}_i(p+1) = \boldsymbol{x}_i(p) + \boldsymbol{v}_i(p+1), \tag{2.24}$$

where $\boldsymbol{x}_i(0) \sim U(\boldsymbol{x}_{\min}, \boldsymbol{x}_{\max})$ is distributed uniformly and the initial velocity is $\boldsymbol{v}_i(1) = 0$ as recommended in [38].

The velocity vector contains both a cognitive component and a social component that represent the attraction to the given particle's best position and the neighborhood's best position, respectively. The velocity vector is calculated as follows:

$$\begin{aligned} v_{ij}(p+1) =& v_{ij}(p) + \underbrace{c_1 r_{1j}(p)[y_{ij}(p) - x_{ij}(p)]}_{\text{cognitive component}} \\ &+ \underbrace{c_2 r_{2j}(p)[\hat{y}_{ij}(p) - x_{ij}(p)]}_{\text{social component}}, \end{aligned} \tag{2.25}$$

where $v_{ij}(p)$ and $x_{ij}(p)$ are the velocity and position of particle i in dimension j, and c_1 and c_2 are positive acceleration constants used to scale the contribution of the cognitive and social components $y_{ij}(p)$ and $\hat{y}_{ij}(p)$, respectively. The factors $r_{1j}(p)$ and $r_{2j}(p) \sim U(0,1)$ are random values sampled from a uniform distribution to introduce a stochastic element to the algorithm.

The best position of a particle for a maximization problem at iteration p is calculated as follows:

$$\boldsymbol{y}_i(p) = \begin{cases} \boldsymbol{x}_i(p), & \text{if } \mathcal{F}(\boldsymbol{x}_i(p)) > \mathcal{F}(\boldsymbol{y}_i(p-1)) \\ \boldsymbol{y}_i(p-1), & \text{else,} \end{cases} \tag{2.26}$$

where in our framework $\mathcal{F}$ is the fitness function and the particle positions represent the policy's parameters $\boldsymbol{x}$.

2.2.1 Inertia Weight and Velocity Clamping

Due to PSO being a stochastic optimization heuristic, it generally converges to different solutions when run multiple times with identical starting conditions. Two basic modifications have been developed to improve the speed of convergence and the quality of solutions found by PSO.

One of the biggest problems occurs if a particle is far from its own and the global best position. Such a particle will get large velocities, sometimes resulting in a divergent movement. To stop this harmful acceleration, two basic modifications have been proposed.

The first improvement is called velocity clamping [34]. The idea is to cut off the velocity at a specific maximum value $v_{\max j}$ in each dimension $j = 1, \ldots, n_\text{d}$. Before the position is updated, the particle's velocity is adjusted as follows:

$$v'_{ij}(p+1) = \begin{cases} v_{\max j}, & \text{if } v_{ij}(p+1) > v_{\max j} \\ v_{ij}(p+1), & \text{if } -v_{\max j} \leq v_{ij}(p+1) \leq v_{\max j} \\ -v_{\max j}, & \text{if } v_{ij}(p+1) < -v_{\max j}, \end{cases} \tag{2.27}$$

where v_{ij} is the j-th component of the original velocity vector of Eq. (2.25). Finding the best values for $\boldsymbol{v}_{\max}$ is non-trivial because it is problem-dependent.

To further increase the ability of the PSO to find optimal solutions, Shi and R. C. Eberhart introduced the inertia weight modification to control the exploration and exploitation abilities of the swarm [137]. The inertia weight w controls how the velocity from the previous iteration is carried over to the new velocity. The position update is then calculated by

$$\boldsymbol{x}_i(p+1) = \boldsymbol{x}_i(p) + \boldsymbol{v}'_i(p+1), \quad \text{with} \tag{2.28}$$

$$v'_{ij}(p+1) = wv'_{ij}(p) + c_1 r_{1j}(p)[y_{ij}(p) - x_{ij}(p)] + c_2 r_{2j}(p)[\hat{y}_j(p) - x_{ij}(p)]. \tag{2.29}$$

Similarly to the maximum velocity, the optimal value for w is problem-dependent. Furthermore, the choice of w has to be made in conjunction with the selection of c_1 and c_2. R. Eberhart and Shi proposed setting the values to $w = 0.7298$ and $c_1 = c_2 = 1.49618$ [35].

PSO, including velocity clamping and inertia weight, is summarized in Algorithm 1.

Algorithm 1: The PSO algorithm [77] including velocity clamping [34] and inertia weight [137]. Particle i is represented by object p_i with position $p_i.\boldsymbol{x}$, personal best position $p_i.\boldsymbol{y}$, and neighborhood best position $p_i.\hat{\boldsymbol{y}}$.

```
Data:
  • $N$ randomly initialized particle positions with $p_i.\boldsymbol{x} = p_i.\boldsymbol{y}$ and zero initial
    velocities $p_i.\boldsymbol{v}$ of particle $i$, with $i = 1, \dots, N$
  • Fitness function $f : \mathbb{R}^{n_\mathrm{d}} \to \mathbb{R}$
  • Acceleration constants $c_1$ and $c_2$ and inertia weight $w$
  • Search space boundaries $\boldsymbol{x}_{\min}$ and $\boldsymbol{x}_{\max}$ and velocity boundaries $\boldsymbol{v}_{\max}$
  • Swarm topology graph defining neighborhood $\mathcal{N}_i$
Result:
  • Global best position $\hat{\boldsymbol{y}}$
1  repeat
2      ▶ Local best positions
3      foreach Particle i do
4          ▶ Determine neighborhood best position of particle i
5          $p_i.\hat{\boldsymbol{y}} \leftarrow \arg\max_{z \in \{p_j.\boldsymbol{y} \,|\, j \in \mathcal{N}_i\}} f(z)$;
6      ▶ Position updates
7      foreach Particle i do
8          ▶ Determine new velocity of particle i
9          for $j = 1, \dots, n_d$ do
10             $p_i.v_j \leftarrow$
                 $p_i.v_j + c_1 \cdot \mathrm{rand}() \cdot [p_i.y_j - p_i.x_j] + c_2 \cdot \mathrm{rand}() \cdot [p_i.\hat{y}_j - p_i.x_j]$;
11             if $p_i.v_j > v_{max\,j}$ then
12                 $p_i.v_j \leftarrow v_{\max j}$;
13             else if $p_i.v_j < -v_{max\,j}$ then
14                 $p_i.v_j \leftarrow -v_{\max j}$;
15         ▶ Compute new position of particle i
16         $p_i.\boldsymbol{x} \leftarrow p_i.\boldsymbol{x} + p_i.\boldsymbol{v}$;
17         ▶ Truncate particle i's position
18         for $j = 1, \dots, n_d$ do
19             $p_i.x_j \leftarrow \min(x_{\max_j}, \max(x_{\min_j}, p_i.x_j))$
20     ▶ Personal best positions
21     foreach Particle i do
22         if $f(p_i.\boldsymbol{x}) > f(p_i.\boldsymbol{y})$ then
23             ▶ Set new personal best position of particle i
24             $p_i.\boldsymbol{y} \leftarrow p_i.\boldsymbol{x}$;
25 until Stopping criterion is met;
26 ▶ Determine the global best position
27     $\hat{\boldsymbol{y}} \leftarrow \arg\max_{z \in \{p_1.\boldsymbol{y}, \dots, p_N.\boldsymbol{y}\}} \{f(z)\}$;
28 return $\hat{\boldsymbol{y}}$
```

2.3 Genetic Programming

GP is a technique, which encodes computer programs as a set of genes. Applying a so-called genetic algorithm (GA) on these genes to modify (evolve) them, drives the optimization of the population. Generally, the space of solutions consists of computer programs, which perform well on predefined tasks [81]. Since we are interested in using interpretable fuzzy rules and equations as RL policies, the genes in our setting include membership functions, basic algebraic functions, as well as constant floating-point numbers and state variables. Note that GP individuals can be depicted as function trees and stored efficiently in memory arrays (Fig. 2.7).

The GA drives the optimization by applying selection and reproduction on the population. The basis for both concepts is a fitness value $\mathcal{F}$ 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 (Fig. 2.7). Herein, we applied tournament selection [11] to select the individuals to be crossed.

To store the individuals efficiently in the computer's memory it is convenient to represent their function trees as arrays. By traversing the tree starting from the root node in a depth-first manner, the nodes of connected subtrees are stored side by side in the array. This property allows to easily identify all array cells that have to be interchanged during crossover. Fig. 2.7 depicts an example of the crossover process on the memory array level.

Experience has shown that it can be advantageous to apply automatic equation cancellation to a certain number (defined by auto cancellation ratio r_a) of the best-performing individuals of one generation. For canceling, an algorithm is applied, which searches the chromosomes for easy-to-cancel subtree structures, calculates the result of the subtree and replaces it by this result. For example, if a subtree with a root node $+$ is found, whose children are two floating-point terminal nodes a and b, the subtree can be replaced by one floating-point terminal node c, computed by $c = a + b$. Similar cancellation rules can be derived for several functions in the function set. Since the tree depth is usually limited in GP for memory reasons, this algorithm can reduce the complexity of substructures and even the whole individual, as well as generate space for potentially more relevant subtrees.

As a mutation operator the so-called Gaussian mutator for floating-point terminals is adopted, which is common in evolutionary algorithms [133, 132]. In each generation, a certain portion (according to terminal mutation ratio r_m) of the best-performing individuals for each complexity is selected. Subsequently, these individuals are copied, and their original floating-point terminals z are mutated by drawing replacement

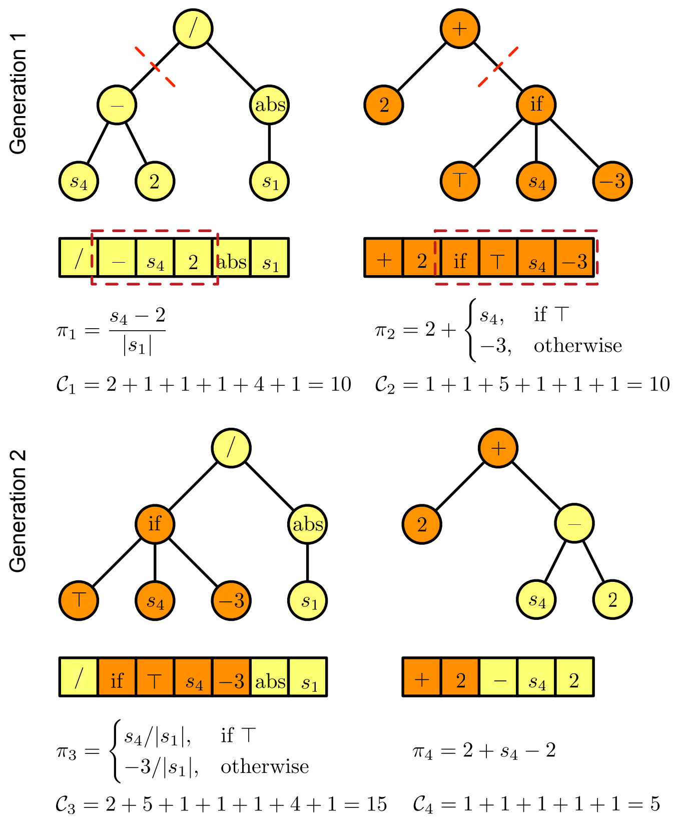

Generation 1
Generation 2
/
−
abs
s_4
2
s_1
+
2
if
⊤
s_4
−3
$\pi_1 = \frac{s_4 - 2}{|s_1|}$
$\mathcal{C}_1 = 2+1+1+1+4+1 = 10$
$\pi_2 = 2 + \begin{cases} s_4, & \text{if } \top \\ -3, & \text{otherwise} \end{cases}$
$\mathcal{C}_2 = 1+1+5+1+1+1 = 10$
$\pi_3 = \begin{cases} s_4/|s_1|, & \text{if } \top \\ -3/|s_1|, & \text{otherwise} \end{cases}$
$\mathcal{C}_3 = 2+5+1+1+1+4+1 = 15$
$\pi_4 = 2 + s_4 - 2$
$\mathcal{C}_4 = 1+1+1+1+1 = 5$

Figure 2.7: GP individuals as function trees and memory arrays. Depicted are four exemplary GP individuals π_i from two consecutive generations with their respective complexity measures $\mathcal{C}_i$. Crossover cutting points are marked in the tree diagrams of policies π_1 and π_2.

terminals z' from a normal distribution $\mathcal{N}(z, 0.1|z|)$. If the performance of the best copy is superior to that of the original individual, it is added to the new population. This strategy provides an option for conducting a local search in the policy search space because the basic structure of the individual's genotype remains untouched.

Initially, the population is generated randomly, as well as a certain portion of each population in every new generation (according to a new random individual ratio r_n). A common strategy to generate valid individuals randomly is to apply the so-called grow method [81]. Growing a chromosome is shown in detail in Algorithm 2. The overall GA used in the experiments is given in Algorithm 3.

Since this work focuses on searching for interpretable solutions, a measure of complexity has to be established. Measuring the complexity of an individual can generally be stated with respect to both its genotype (structural) or its phenotype (functional) [85]. Here, a simple node-counting measuring strategy is used where different types of functions, variables, and terminals are counted with different weightings. Hence, domain experts, who would try interpreting the RL policies delivered, could pre-define which types of genes they would be more likely to accept compared to others. For the conducted experiments complexity weightings from Eureqa[1], a commercially available software for symbolic regression [32], have been adopted. Table 2.1 lists some of these weightings and Fig. 2.7 gives four examples on how to calculate the respective complexities.

[1]`https://www.nutonian.com/products/eureqa`

Algorithm 2: The GP growth algorithm [81]. This algorithm enforces a broad variety of individuals since it randomizes the length of each subtree (lines 1 and 16) as well as the position of the biggest subtree (line 13). Both properties save the GA from generating only small initial chromosomes, which eventually would result in an early convergence of the population.

```
Data:
    • Minimum and maximum tree depths $d_{\min}$ and $d_{\max}$
    • Set of terminals and variables $\mathcal{G}_T$
    • Set of functions $\mathcal{G}_F$
Result:
    • Chromosome $c$
1  ▶ Randomly draw tree depth $d$ from a uniform distribution
2    $d \sim U(d_{\min}, d_{\max})$;
3  $c \leftarrow$ select_next_gene($d$);
4  return $c$;
5  Function select_next_gene($d$)
6      if $d < 1$ then
7          ▶ Randomly draw gene $g$ from the set of terminals and variables
8            $g \in \mathcal{G}_T$;
9      else
10         ▶ Randomly draw gene $g$ from the set of functions
11           $g \in \mathcal{G}_F$;
12     ▶ Add $g$ to subchromosome $c'$
13     ▶ Randomly select leaf $i$ of $g$
14     ▶ Build chromosome $c'_i \leftarrow$ select_next_gene($d-1$)
15     foreach Leaf $j$ of $g$ with $j \neq i$ do
16         ▶ Randomly draw subtree depth $d_j$ from a uniform distribution
17           $d_j \sim U(0, d-1)$;
18         ▶ Build chromosome $c'_j \leftarrow$ select_next_gene($d_j$)
19     return $c'$;
```

Algorithm 3: The GA used for the GP experiments. The basic algorithm from [81] is enhanced by tournament selection [11], Gaussian mutation [133, 132], and auto cancellation. This algorithm evolves a population of possible solutions and returns individuals forming a Pareto front after completion.

```
Data:
    • Population size $N$
    • Ratios $r_c$, $r_r$, $r_a$, $r_m$
Result:
    • Pareto front $F_P$
▶ Randomly initialize population $P$ of size $N$
▶ Determine fitness value of each individual (in parallel)
▶ Evolve next generation
 repeat
     ▶ Initialize a new empty population $P'$ with capacity for $N$ individuals
     ▶ Add new individuals by crossover
     repeat
         ▶ Select individuals $i$ and $j$ from $P$ by tournament selection
         ▶ Cross tournament winners $i$ and $j$ to yield offspring $i'$ and $j'$
         ▶ Add offspring $i'$ and $j'$ to $P'$
     until $N \cdot r_c$ individuals have been added to $P'$;
     ▶ Add new individuals by reproduction
     repeat
         ▶ Select individual $i$ from $P$ by tournament selection
         ▶ Add $i$ to $P'$
     until $N \cdot r_r$ individuals have been added to $P'$;
     ▶ Automatic cancelation and floating-point mutation
     repeat
         foreach Complexity level $l$ do
             ▶ Select individual $i$ from $P$ with the next highest fitness for $l$ which
               has not been selected before
             if Less than $N \cdot r_a$ individuals have been added after cancelation then
                 ▶ Apply automatic cancelation on $i$ to yield $i'$
                 if $i'$ is more compact compared to $i$ then
                     ▶ Add $i'$ to $P'$
             if Less than $N \cdot r_m$ individuals have been added after mutation then
                 ▶ Apply Gaussian mutator on $i$ to yield $i''$
                 if Fitness of $i''$ is higher compared to $i$ then
                     ▶ Add $i''$ to $P'$
     until All individuals in $P$ have been considered;
     ▶ Fill $P'$ with randomly generated individuals
     ▶ Determine fitness value of each individual (in parallel)
     ▶ Set $P \leftarrow P'$
 until Stopping criterion is met;
▶ Extract best individuals for each complexity level yielding Pareto front $F_P$
return Pareto front $F_P$
```

Variables	1
Terminals	1
$+, -, \cdot$	1
$/$	2
$\wedge, \vee, >, <$	4
tanh, abs	4
if	5

Table 2.1: GP complexities

CHAPTER 3

Benchmark Dynamics & Surrogate Models

In this chapter, the benchmarks which have been used to evaluate the reinforcement learning (RL) methods of this thesis are introduced (Section 3.1). Further, in Section 3.2 the different types of surrogate models are formally introduced and described for each benchmark. These models are employed by every following reinforcement learning (RL) technique, even by the model-free neural fitted Q iteration (NFQ) approach, where using surrogate models for policy selection will show to improve the methods outcome significantly.

3.1 Benchmark Dynamics

In this section, four RL benchmark problems are introduced. The first three problems are the well-known RL toy benchmarks mountain car (MC), cart-pole balancing (CPB), and cart-pole swing-up (CPSU) (Sections 3.1.1-3.1.3). For these tasks, numerous publications with performance values and exemplary control strategies exist. Nevertheless, although these benchmarks are toy problems, the included system dynamics are not trivial and finding adequate control strategies from batch data is a challenging task. However, to evaluate the proposed RL methods in a more realistic industry setting, the fourth benchmark is the industrial benchmark (IB). In Section 3.1.4 the dynamics of this highly stochastic and multi-dimensional testbed are introduced and motivated.

Table 3.1 gives an overview on important properties of the four benchmarks.

3.1.1 Mountain Car

In the MC benchmark, an underpowered car must be driven to the top of a hill (Fig. 3.1) [101]. This goal is achieved by building sufficient potential energy by first driving in the direction opposite to the final direction. The system is fully described by

	MC	CPB	CPSU	IB
Observation dimensionality	2	4		6
Action dimensionality	1			3
Reward	discrete			continuous
End of episode	goal state	fail state		-
Stochasticity	-			state-dependent
Observability of Markov state	fully observable			partially observable

Table 3.1: Benchmark properties

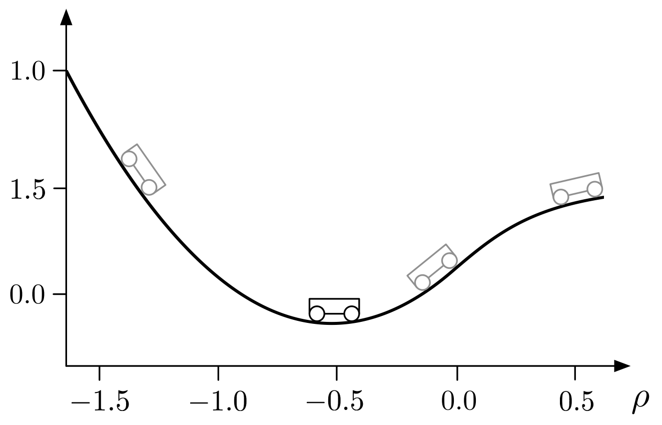


Figure 3.1: MC benchmark. The task is to first build up momentum by driving to the left in order to subsequently reach the top of the hill on the right at $\rho = 0.6$.

the two-dimensional state space $\boldsymbol{s} = (\rho, \dot{\rho})$ representing the car's position ρ and velocity $\dot{\rho}$.

The MC experiments are conducted using the freely available CLS^2 software ('clsquare')[1], which is an RL benchmark system that applies the Runge-Kutta fourth-order method to approximate closed-loop dynamics. The task for the RL agent is to find a sequence of force actions $a_t, a_{t+1}, a_{t+2}, \ldots \in [-1, 1]$ that drive the car up the hill, which is achieved when reaching position $\rho \geq 0.6$.

At the start of each episode, the car's position is initialized in the interval $[-1.2, 0.6]$. The agent receives a reward of

$$r(\boldsymbol{s}_{t+1}) = \begin{cases} 0, & \text{if } \rho_{t+1} \geq 0.6, \\ -1, & \text{otherwise,} \end{cases} \tag{3.1}$$

subsequent to each state transition $\boldsymbol{s}_{t+1} = g(\boldsymbol{s}_t, a_t)$. This reward function solely encodes the benchmark goal, which is defined by reaching the hill to the right as quickly as possible. Note that, by design, no hint to the goal heuristic is encoded into the reward

[1]http://ml.informatik.uni-freiburg.de/research/clsquare.

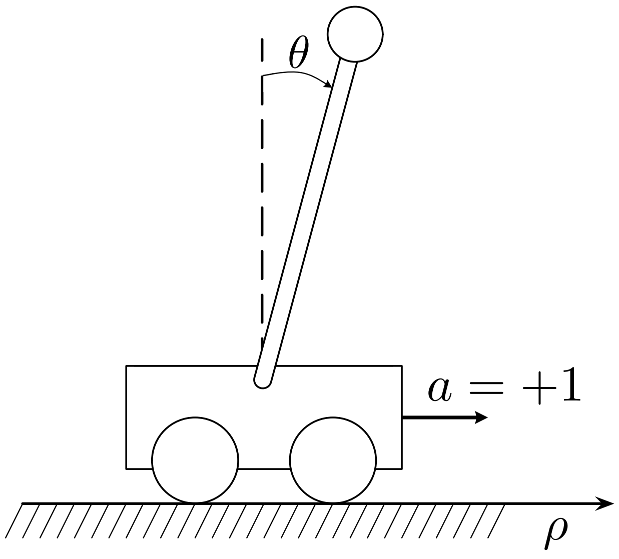


Figure 3.2: Cart-pole benchmark. The task is to balance the pole around $\theta = 0$ while moving the cart to position $\rho = 0$ by applying positive or negative force to the cart.

function, to avoid introducing any bias to the tested RL methods. When the car reaches the goal position, i.e., $\rho_{t+1} \geq 0.6$, its position becomes fixed, the velocity becomes zero, and the agent perceives the maximum reward in each following time step regardless of the applied actions.

3.1.2 Cart-pole Balancing

The cart-pole experiments described in the following two sections were also conducted using the CLS^2 software. The objective of the CPB benchmark is to apply forces to a cart moving on a one-dimensional track to keep a pole hinged to the cart in an upright position (Fig. 3.2). Here, the four Markov state variables are the pole angle θ, the pole angular velocity $\dot{\theta}$, the cart position ρ, and the cart velocity $\dot{\rho}$. These variables describe the Markov state completely, i.e., no additional information about the system's past behavior is required. The task for the RL agent is to find a sequence of force actions $a_t, a_{t+1}, a_{t+2}, \ldots$ that prevent the pole from falling over. [41]

In the CPB task, the angle of the pole and the cart's position are restricted to intervals of $[-0.7, 0.7]$ and $[-2.4, 2.4]$ respectively. Once the cart has left the restricted area, the episode is considered a failure, i.e., velocities become zero, the cart's position and pole's angle become fixed, and the system remains in the failure state for the rest of the episode. The RL policy can apply force actions on the cart from -10 N to $+10$ N in time intervals of 0.025 s.

The reward function for the balancing problem is given as follows:

$$r(\mathbf{s}_{t+1}) = \begin{cases} 0.0, & \text{if } |\theta_{t+1}| < 0.25 \text{ and } |\rho_{t+1}| < 0.5, \\ -1.0, & \text{if } |\theta_{t+1}| > 0.7 \text{ or } |\rho_{t+1}| > 2.4, \\ -0.1, & \text{otherwise.} \end{cases} \tag{3.2}$$

Based on this reward function, the primary goal of the policy is to avoid reaching the failure state. The secondary goal is to drive the system to the goal state region where $r = 0$ and balance it there for the rest of the episode. As for the MC, the reward function solely represents the benchmark goals avoiding any hint to the goal heuristic.

3.1.3 Cart-pole Swing-up

The CPSU benchmark is based on the same system dynamics as the CPB benchmark. In contrast to CPB, the position of the cart and the angle of the pole are not restricted. Consequently, the pole can swing through, which is an essential property of CPSU. Since the pole's angle is initialized in the full interval of $[-\pi, \pi]$, it is often necessary for the policy to swing the pole several times from side to side to gain sufficient energy to erect the pole and receive the highest reward.

In the CPSU setting, the policy can apply actions from -30 N to $+30$ N on the cart. The reward function for the problem is given as follows:

$$r(\mathbf{s}_{t+1}) = \begin{cases} 0, & \text{if } |\theta_{t+1}| < 0.5 \text{ and } |\rho_{t+1}| < 0.5, \\ -1, & \text{otherwise.} \end{cases} \tag{3.3}$$

This is similar to the reward function for the CPB benchmark but does not contain any penalty for failure states.

3.1.4 Industrial Benchmark

The IB[2] was designed to emulate several challenging aspects existing in many industrial applications [61, 65, 63]. It is not designed to be an approximation of any specific real-world system, but to pose a comparable hardness and complexity found in many industrial applications. The content of this section is based on the following publications as part of the doctoral research:

- D. Hein, S. Udluft, M. Tokic, A. Hentschel, T. A. Runkler, and V. Sterzing. "Batch reinforcement learning on the industrial benchmark: First experiences." In: 2017 International Joint Conference on Neural Networks (IJCNN). 2017, pp. 4214–4221. [65]

- D. Hein, S. Depeweg, M. Tokic, S. Udluft, A. Hentschel, T. A. Runkler, and V. Sterzing. "A benchmark environment motivated by industrial control problems."

[2] `http://github.com/siemens/industrialbenchmark`

In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI). 2017, pp. 1–8. [61]

In the IB, state and action spaces are continuous. The state space is high-dimensional and only partially observable. The actions consist of three continuous components and affect three control inputs. Moreover, the IB includes stochastic and delayed effects. The optimization task is multi-criterial in the sense that two reward components show opposite dependencies on the actions. The dynamical behavior is heteroscedastic with state-dependent observation noise and state-dependent probability distributions, based on latent variables. Furthermore, it depends on an external driver that cannot be influenced by the actions.

At any time step t the RL agent can influence the IB via actions $\boldsymbol{a}_t$ that are three-dimensional vectors in $[-1,1]^3$. Each action can be interpreted as three proposed changes to three observable state control variables. Those variables are: velocity v, gain g, and shift h. Each variable is limited to $[0,100]$ and calculated as follows:

$$\boldsymbol{a}_t = (\Delta v_t, \Delta g_t, \Delta h_t)\,, \tag{3.4}$$

$$v_{t+1} = \max\left(0, \min\left(100, v_t + d^{\text{v}}\Delta v_t\right)\right), \tag{3.5}$$

$$g_{t+1} = \max\left(0, \min\left(100, g_t + d^{\text{g}}\Delta g_t\right)\right), \tag{3.6}$$

$$h_{t+1} = \max\Big(0, \min\Big(100, h_t + d^{\text{h}}\Delta h_t\Big)\Big)\,, \tag{3.7}$$

with scaling factors $d^{\text{v}} = 1$, $d^{\text{g}} = 10$, and $d^{\text{h}} = 5.75$.

After applying the action $\boldsymbol{a}_t$, the environment transitions to the next time step $t+1$, yielding the internal state $\boldsymbol{s}_{t+1}$. State $\boldsymbol{s}_t$ and successor state $\boldsymbol{s}_{t+1}$ are Markovian states of the environment, which are only partially observable by the agent. In addition to the three control variables velocity v, gain g, and shift h, a setpoint p is applied to the system. Setpoint p simulates an external force like the demanded load in a power plant or the wind speed actuating a wind turbine, which cannot be controlled by the agent, but still has a significant influence on the system dynamics. Depending on the setpoint p_t and the choice of control values $\boldsymbol{a}_t$, the system suffers from detrimental fatigue f_t and consumes resources such as power, fuel, etc., represented by consumption c_t. In each time step, the IB generates output values for c_{t+1} and f_{t+1}, which are part of the internal state $\boldsymbol{s}_{t+1}$. The reward is solely determined by $\boldsymbol{s}_{t+1}$ as follows:

$$r_{t+1} = -c_{t+1} - 3f_{t+1}. \tag{3.8}$$

The dynamics can be decomposed into three different sub-dynamics named operational cost, mis-calibration, and fatigue. Consumption $c_t = c(\theta_t^c, m_t)$ is a function of the convoluted operational cost θ_t^c and mis-calibration m_t.

Dynamics of operational cost

The sub-dynamics of operational cost are influenced by the external driver setpoint p and two of the three steerings, velocity v and gain g. The current operational cost θ_t is calculated as

$$\theta_t = \exp\left(\frac{2p_t + 4v_t + 2.5g_t}{100}\right). \tag{3.9}$$

The observation of θ_t is delayed and blurred by the following convolution:

$$\theta^c_t = \frac{1}{9}\theta_{t-5} + \frac{2}{9}\theta_{t-6} + \frac{3}{9}\theta_{t-7} + \frac{2}{9}\theta_{t-8} + \frac{1}{9}\theta_{t-9}. \tag{3.10}$$

The convoluted operational cost θ^c_t cannot be observed directly, instead it is modified by the second sub-dynamic, called mis-calibration, and finally subject to observation noise. The motivation for this dynamical behavior is that it is non-linear, it depends on more than one influence, and it is delayed and blurred. All those effects have been observed in industrial applications, like the heating process observable during combustion.

Dynamics of mis-calibration

The sub-dynamics of mis-calibration m are influenced by external driver setpoint p and steering shift h. The goal is to reward an agent to oscillate in h in a pre-defined frequency around a specific operation point determined by setpoint p. Thereby, the reward topology is inspired by an example from quantum physics, namely Goldstone's "Mexican hat" potential.

In the first step, setpoint p and shift h are combined to an effective shift h^e calculated by:

$$h^\mathrm{e} = \max\left(-1.5, \min\left(1.5, \frac{h}{20} - \frac{p}{50} - 1.5\right)\right). \tag{3.11}$$

Effective shift influences three latent variables, which are domain δ, response ψ, and direction ϕ. Domain δ can enter two discrete states, which are *negative* and *positive*, represented by integer values -1 and $+1$, respectively. Response ψ can enter two discrete states, which are *disadvantageous* and *advantageous*, represented by integer values -1, and $+1$, respectively. Direction $\phi \in \{-6, -5, \dots, 6\}$ is a discrete index variable, yielding the position of the current optima in the mis-calibration penalty space. Figure 3.3 is a visualization of the mis-calibration dynamics.

Note that the following sine function represents the optimal policy for the mis-calibration dynamics:

$$h^\mathrm{e} = \sin\left(\frac{\pi}{12}\phi\right). \tag{3.12}$$

Exemplary policy trajectories through the penalty landscape of mis-calibration are depicted and described in Figure 3.4.

Dynamics of fatigue

The sub-dynamics of fatigue are influenced by the same variables as the sub-dynamics of operational cost, i.e., setpoint p, velocity v, and gain g. The IB is designed in such a way that, when changing velocity v and gain g as to reduce the operational cost, fatigue will

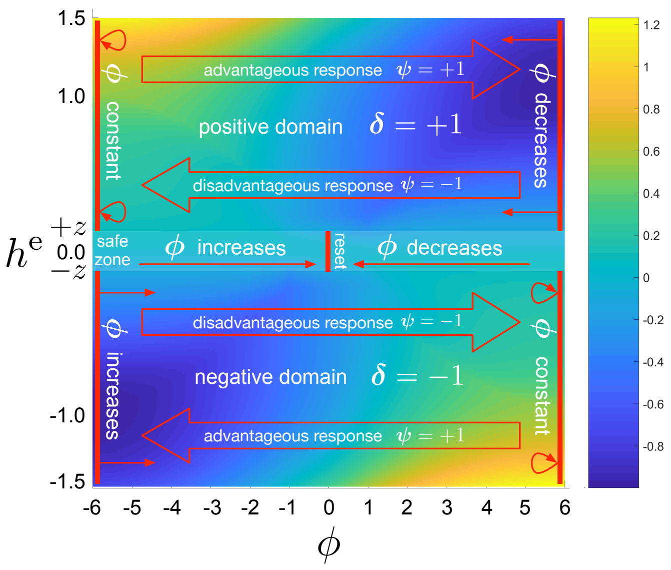
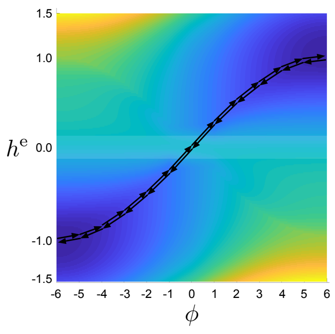
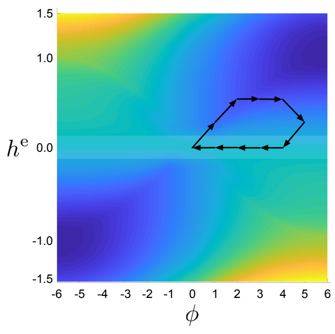


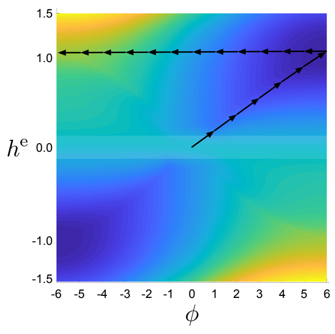
Figure 3.3: A visual description of the mis-calibration dynamics. Blue color represents areas of low penalty (-1.00), while the yellow color represents areas of high penalty (1.23). If the policy keeps h^e in the so-called safe zone, ϕ is driven towards 0 stepwise. When $\phi = 0$ is reached the mis-calibration dynamics are reset, i.e., domain $\delta = 1$ and response $\psi = 1$. The policy is allowed to start the rotation cycle at any time by leaving the safe zone and entering the *positive* or the *negative* domain. Consider *positive* domain $\delta = 1$: After initially leaving the safe zone, response is in the state *advantageous*, i.e., ϕ is increased stepwise. The upper right area is a reversal point for ϕ. As soon as $\phi = 6$, response switches from *advantageous* $\psi = 1$ to *disadvantageous* $\psi = -1$. In the subsequent time steps, ϕ is decreased until either the policy brings h^e back to the safe zone or ϕ reaches the left boundary at -6. If the latter occurs, phi is kept constant at -6, i.e., the policy yields a high penalty in each time step. Since the mis-calibration dynamics are symmetric around $(\phi, h^e) = (0, 0)$, opposite dynamics are applied in the *negative* domain at the lower part of the plot.



(a) Optimal policy



(b) Suboptimal policy



(c) Bad policy

Figure 3.4: Comparison of three mis-calibration policies. Depicted is a visual representation of the Goldstone potential based function $m(\phi, h^{\mathrm{e}})$. Areas yielding high penalty are colored yellow, while areas yielding low penalty are colored blue. The highlighted area in the center depicts the safe zone from $-z$ to z. (a) A policy which maintains h^{e} such that a sine-shaped trajectory is generated yields lowest penalty. Note that the policy itself starts the rotation cycle at any time by leaving the safe zone. After returning to the safe zone, while at the same time $\phi = 0$, the dynamics are reset, and a new cycle can be initiated at any following time step in positive or negative direction. (b) The depicted policy starts initiating the rotation cycle by leaving the safe zone, but returns after six steps. After this return, ϕ is decreased in four steps back to 0. Subsequently, the dynamics are reset. This policy yields a lower penalty compared to a constant policy that remains in the safe zone the whole time. (c) The depicted policy approaches one of the global optima of $m(\phi, h^{\mathrm{e}})$ by directly leaving the safe zone z by constantly increasing h^{e}. Subsequently, it remains at this point. However, the rotation dynamic yields a steady decrease in ϕ after reaching the right boundary at $\phi = 6$. This decrease "pushes" the agent to the left, i.e., the penalties received are increased from step to step. After reaching the left boundary at $\phi = -6$, the dynamics remain in this area of high penalty. Note that the policy could bring the dynamics back to the initial state $\phi = 0$ by returning to $h^{\mathrm{e}} < z$. This benchmark property ensures that the best constant policies are the ones who remain in the safe zone.

be increased, leading to the desired multi-criterial task, with two reward-components showing opposite dependencies on the actions. The basic fatigue f^{b} is computed as

$$f^{\mathrm{b}} = \max\left(0, \frac{30000}{5\,v + 100} - 0.01\,g^2\right). \tag{3.13}$$

From basic fatigue f^{b}, fatigue f is calculated by

$$f = f^{\mathrm{b}} \cdot (1 + 2\alpha)\,, \tag{3.14}$$

where α is an amplification. This amplification depends on two latent variables and is affected by noise. α can undergo a bifurcation if one of the latent variables reaches a certain value. In that case, α will increase and lead to higher fatigue, affecting the reward negatively.

Approximation of the Markovian state

The complete Markov state $\boldsymbol{s}$ of the IB remains unobservable. Only an observation vector $\boldsymbol{o} \subset \boldsymbol{s}$ consisting of:

- the current control variables velocity v_t, gain g_t, and shift h_t,
- the external driver setpoint p_t, and
- the reward relevant variables consumption c_t and fatigue f_t,

can be observed externally.

In Section 2.1 the optimization task in model-based RL is described as working on the Markovian state $\boldsymbol{s}$ of the system dynamics. Since this state is not observable in the IB environment, $\boldsymbol{s}_t$ is approximated by a sufficient amount of historic observations $(\boldsymbol{o}_{t-H}, \boldsymbol{o}_{t-H+1}, \ldots, \boldsymbol{o}_t)$ with time horizon H. Given a system model $\tilde{g}(\boldsymbol{o}_{t-H}, \boldsymbol{o}_{t-H+1}, \ldots, \boldsymbol{o}_t, \boldsymbol{a}_t) = (\boldsymbol{o}_{t+1}, r_{t+1})$ with $H = 30$ an adequate prediction performance could be achieved during IB experiments. Note that observation size $|\boldsymbol{o}| = 6$ in combination with time horizon $H = 30$ results in a 180-dimensional approximation vector of the Markovian state.

3.2 Neural Network Surrogate Models

In the considered problem domain, i.e., continuous, smooth, and deterministic system dynamics, neural networks (NNs) are known to serve as well-suited world models. Given a batch of previously generated transition samples, the NN training process is data-efficient, and training errors are excellent indicators of how well the model will perform in model-based RL training. In this section, the NN-based concepts of the multi-layer perceptron (MLP) and the recurrent neural network (RNN) which have been utilized in the following experiments are introduced.

However, for different problem domains, alternative types of world models might be preferable. For example, Gaussian processes [111] also provide a good approximation of the mean of the target value. In addition to that, this technique indicates the level of confidence about this prediction. This feature may be of value for stochastic system dynamics. Moreover, Bayesian NNs include stochastic input variables to capture complex statistical patterns in the transition dynamics [27]. A third alternative modeling technique is the use of regression trees [13]. While typically lacking data efficiency, regression tree predictions are less affected by non-linearities perceived by system dynamics because they do not rely on a parametric function approximation.

3.2.1 Multi-layer Perceptron

An NN is a universal approximator; hence it can approximate any given functional relationship to an arbitrary fixed precision, given a sufficient number of neurons [69, 24, 70, 60]. The most common kind of NNs is the MLP introduced by Rosenblatt in 1958 [122, 60].

An MLP is a directed graph consisting of nodes and edges. A three-layer MLP consists of one input layer and two functional layers, i.e., one hidden and one output layer. Each layer consists of a fixed number of nodes, which are hierarchically connected to the nodes on the previous and the next layer. Each edge has a real-valued weight assigned. The input vector **s** is propagated to the next layer, that is the hidden layer, to compute

$$\mathrm{in}_{(2,j)} = \sum_{i=1}^{N_1} s_i w_{(1,i),(2,j)} + b_{(2,j)}, \tag{3.15}$$

where N_1 are the number of input nodes and the index variables i and j are assigned to the input and the hidden layer nodes, respectively. The values $w_{(1,i),(2,j)}$ are the weights at the edge between the output of node i and the input of node j, and the value $b_{(2,j)}$ is a so-called bias on node j.

On the level of the hidden layer, an activation function f is applied to the input of node j:

$$\mathrm{out}_{(2,j)} = f(\mathrm{in}_{(2,j)}). \tag{3.16}$$

In an MLP, multiple hidden layers can be stacked on each other where the output of node j of layer l is computed by

$$\mathrm{out}_{(l,j)} = f\left(\sum_{i=1}^{N_{l-1}} s_i w_{(l-1,i),(l,j)} + b_{(l,j)}\right). \tag{3.17}$$

The outputs y_j of the network are similarly computed on the output layer M:

$$y_j = f\left(\sum_{i=1}^{N_{M-1}} s_i w_{(M-1,i),(M,j)} + b_{(M,j)}\right). \tag{3.18}$$

Fig. 3.5 shows a graph-based visualization of the presented MLP.

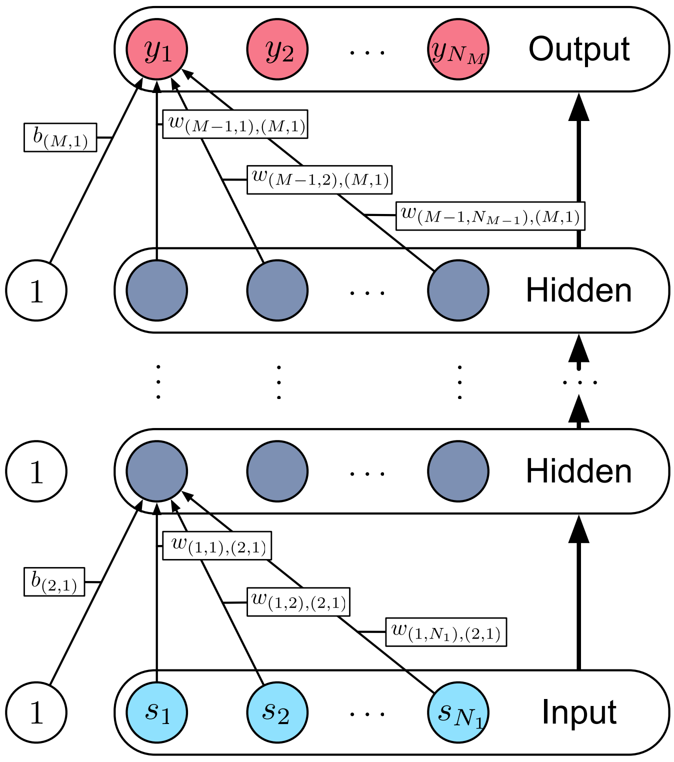


Figure 3.5: The MLP. Depicted are the connections from the input nodes to one of the hidden nodes of the first hidden layer as well as the connections from the hidden nodes of the last hidden layer to one of the network's outputs. Note that the big arrows between the layer boxes represent that all nodes between these layers are fully connected.

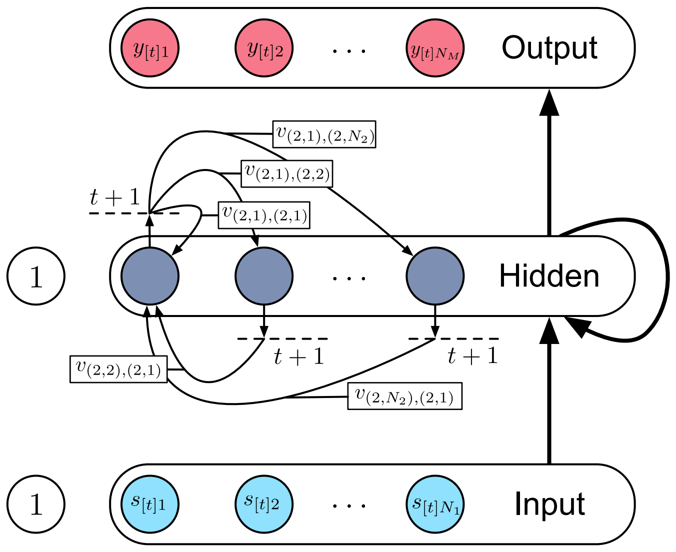


Figure 3.6: The RNN. Note that only the additional recurrent connections for the first hidden node are depicted. At the dashed lines the outputs of the nodes are memorized for the next time step.

3.2.2 Recurrent Neural Networks

RNNs are an augmented form of MLPs. By adding recurrent connections with temporal delays on the nodes of the hidden layers, RNNs can model sequential data [36]. This property makes RNNs exceptionally useful for modeling sequential tasks like speech recognition or partially observable systems [98, 56, 94, 127].

The input vectors $(\mathbf{s}_{[1]}, \mathbf{s}_{[2]}, \ldots, \mathbf{s}_{[T]})$ of the RNN are processed sequentially and yield the hidden state sequence $(\text{out}_{[1]}, \text{out}_{[2]}, \ldots, \text{out}_{[T]})$. In the final layer the output sequence $(y_{[1]}, y_{[2]}, \ldots, y_{[T]})$ is computed. In a simple RNN, node j of the hidden layer is not only provided with the input vector $\mathbf{s}_{[t]}$, but also with an additional term computed from $\text{out}_{[t-1](2,h)}$ weighted by $v_{(2,h),(2,j)}$ with h as index variable for the memory layer. Eq. (3.15) then changes to

$$\text{in}_{[t](2,j)} = \sum_{i=1}^{N_1} s_{[t]i} w_{(1,i),(2,j)} + \sum_{h=1}^{N_2} \text{out}_{[t-1](2,h)} v_{(2,h),(2,j)} + b_{(2,j)}. \tag{3.19}$$

Fig. 3.6 illustrates the recurrent connections in the hidden layer for an exemplary three-layer RNN. More detailed information on how to set up and train NNs can be found in [121] and [33].

3.2.3 Models for Mountain Car, Cart-pole Balancing & Cart-pole Swing-up

The benchmark environments mountain car (MC), cart-pole balancing (CPB), and cart-pole swing-up (CPSU), have fully observable Markov states, which allows us to model

Reward		b_1	b_2	$\dots$	b_N	
r_1	[	1	0		0	]
r_2	[	0	1		0	]
$\dots$				$\ddots$		
r_N	[	0	0		1	]

Table 3.2: One-hot encoding

their system dynamics with simple MLPs. The basis for the modeling process is a data set $\mathcal{D}$ that contains state transition samples gathered from the real system dynamics. These samples are represented by tuples $(\mathfrak{s}, \mathfrak{a}, \mathfrak{s}', \mathfrak{r})$, where, in state $\mathfrak{s}$, action $\mathfrak{a}$ was applied and resulted in a state transition to $\mathfrak{s}'$ and yielded reward $\mathfrak{r}$. Note that $\mathcal{D}$ can be generated using any (even a random) policy before policy training as long as sufficient exploration has been applied.

We generate world models $\tilde{g}$ with inputs $(\boldsymbol{s}, \boldsymbol{a})$ to predict $\boldsymbol{s}'$, using data set $\mathcal{D}$. For many applications it is advantageous to learn the deltas of the state variables and train a single model per state variable separately to yield better approximative quality:

$$\begin{aligned}
\Delta s_1' &= \tilde{g}_{s_1}(s_1, s_2, \dots, s_m, \boldsymbol{a}),\\
\Delta s_2' &= \tilde{g}_{s_2}(s_1, s_2, \dots, s_m, \boldsymbol{a}),\\
\dots&\\
\Delta s_m' &= \tilde{g}_{s_m}(s_1, s_2, \dots, s_m, \boldsymbol{a}).
\end{aligned}$$

Then, the resulting state is calculated according to $\boldsymbol{s}' = (s_1 + \Delta s_1', s_2 + \Delta s_2', \dots, s_m + \Delta s_m')$.

The reward is also given in $\mathcal{D}$; thus, the reward function can be approximated using $r = \tilde{r}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$. However, since MC, CPB, and CPSU yield discrete rewards a more stable and precise reward approximation can be derived from the available data batch. Herein one-hot encoding, which transforms the categorical rewards $(r_1, r_2, \dots, r_N)$ to a more conveniently predictable problem representation, is used. Table. 3.2 shows how the rewards are encoded as binary vectors with bits $(b_1, b_2, \dots, b_n)$ where all bits are set to zero with one exception, which is $b_i = 1$ in the bit vector of r_i. Note that such a representation can be interpreted as a probability vector for each observed reward with respect to one of the reward classes. Consequently, the NN no longer predicts reward r directly but rather it is optimized to predict the probabilities for each reward class by means of the one-hot vector, i.e., $(b_1, b_2, \dots, b_N) = \tilde{r}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$. Bit b_i with the highest prediction value is determined and subsequently the associated reward r_i is assigned to the respective state-action transition.

Before training, the respective data sets were split into blocks of 80%, 10%, and 10% which are training, validation, and generalization sets, respectively. While the weight updates during training were computed by utilizing the training sets, the weights that performed best given the validation sets were used as training results. Finally, those weights were evaluated using the generalization sets to rate the overall approximation quality on unseen data.

For the MC benchmark the following two models predicting the next state are learned:

$$\begin{aligned}\Delta\rho' &= \tilde{g}_{\rho}(\rho, \dot{\rho}, a),\\ \Delta\dot{\rho}' &= \tilde{g}_{\dot{\rho}}(\rho, \dot{\rho}, a).\end{aligned}$$

The MC reward is predicted by

$$r = \begin{cases} 0, & \text{if } b_1 > b_2,\\ -1, & \text{otherwise,}\end{cases} \tag{3.20}$$

with $(b_1, b_2) = r(\rho, \dot{\rho}, a, \rho', \dot{\rho}')$, where b_1 represents the probability of having reached the goal area. As soon as $b_1 > b_2$ the MC episode is finished and declared as successful. The MC NNs were trained with data set $\mathcal{D}_{\text{MC}}$ containing state-action transitions from trajectories generated by applying random actions on the benchmark dynamics. The start states for these trajectories were uniformly sampled as $\boldsymbol{s} = (\rho, \dot{\rho}) \in [-1.2, 0.6] \times \{0\}$, i.e., at a random position on the track with zero velocity. $\mathcal{D}_{\text{MC}}$ contains 10 000 transition samples. The surrogate model consists of two MLPs of the form 3-10-10-10-1, i.e., three input neurons for the state features ρ and $\dot{\rho}$ and one for the action a, three hidden layers containing ten non-linearly activated neurons, and one neuron as output predicting $\Delta\rho'$ and $\Delta\dot{\rho}'$ respectively. While input and output layers apply linear activation functions the neurons on the hidden layers are activated by a rectifier function, i.e., $f(x) = \max(0, x)$. Since the task is to predict probabilities with the reward network, sigmoid activation functions have been used on the output neurons and the values have been normalized to sum up to one.

For the CPB benchmark the following four models of form 5-10-10-10-1 predicting the next state are learned:

$$\begin{aligned}\Delta\theta' &= \tilde{g}_{\theta}(\theta, \dot{\theta}, \rho, \dot{\rho}, a),\\ \Delta\dot{\theta}' &= \tilde{g}_{\dot{\theta}}(\theta, \dot{\theta}, \rho, \dot{\rho}, a),\\ \Delta\rho' &= \tilde{g}_{\rho}(\theta, \dot{\theta}, \rho, \dot{\rho}, a),\\ \Delta\dot{\rho}' &= \tilde{g}_{\dot{\rho}}(\theta, \dot{\theta}, \rho, \dot{\rho}, a).\end{aligned}$$

The CPB reward is predicted by

$$r = \begin{cases} 0.0, & \text{if } b_1 > b_2 \text{ and } b_1 > b_3,\\ -1.0, & \text{if } b_2 > b_1 \text{ and } b_2 > b_3,\\ -0.1, & \text{otherwise,}\end{cases} \tag{3.21}$$

with $(b_1, b_2, b_3) = r(\theta, \dot{\theta}, \rho, \dot{\rho}, a, \theta', \dot{\theta}', \rho', \dot{\rho}')$, where b_1 and b_2 represent the probabilities for the goal area and a failed episode respectively. As soon as $r = -1$ the CPB episode is finished and declared as failed. For the training set $\mathcal{D}_{\text{CPB}}$, the samples originate from trajectories of 100 state transitions generated by a random walk on the benchmark dynamics. The 100 start states $(\theta, \dot{\theta}, \rho, \dot{\rho})$ for these trajectories were sampled uniformly from $[-0.7, 0.7] \times \{0\} \times [-2.4, 2.4] \times \{0\}$. $\mathcal{D}_{\text{CPB}}$ contains 10 000 transition samples.

The reward prediction for the CPSU benchmark is computed as follows:

$$r = \begin{cases} 0, & \text{if } b_1 > b_2, \\ -1, & \text{otherwise,} \end{cases} \tag{3.22}$$

with $(b_1, b_2) = r(\theta, \dot{\theta}, \rho, \dot{\rho}, a, \theta', \dot{\theta}', \rho', \dot{\rho}')$, where b_1 represents the probability for the goal area. The training set $\mathcal{D}_{\text{CPSU}}$ consists of 10 000 transition samples originating from 100 random walk trajectories of length 100. The start states $(\theta, \dot{\theta}, \rho, \dot{\rho})$ for these trajectories were sampled uniformly from $[-\pi, \pi] \times \{0\} \times \{0\} \times \{0\}$.

Further details about NN performance concerning data batch sizes and the number of hidden layers can be found in [62], a publication which contains a comprehensive parameter study for MC, CPB, and CPSU in a model-based RL setting.

3.2.4 Recurrent Models for Industrial Benchmark

In the IB experiments, an RNN system model predicts consumption c and fatigue f for each step of the rollout. To generate an adequate training data set $\mathcal{D}_{\text{IB}}$, the IB has been initialized ten times for each setpoint $p = \{10, 20, \ldots, 100\}$ and produced random trajectories of lengths 1 000, resulting in $|\mathcal{D}_{\text{IB}}| = 100\,000$.

The system model $\tilde{g}$ was chosen to consist of two RNNs $\tilde{g}_c$ and $\tilde{g}_f$, to predict consumption c and fatigue f, respectively. Both models are unfolded a sufficient number of steps into the past ($H_c = 30$ time steps for $\tilde{g}_c$, $H_f = 10$ time steps for $\tilde{g}_f$) and 50 time steps into the future. In each time step, they take the observable variables of the past and present as input. Whereas, in the future branch of the RNNs only the steerings (velocity, gain, and shift) are used as input. The detailed topology of these RNNs is described in [33] as *Markov decision process extraction network*.

Both models have been trained with the Rprop learning algorithm [119] on the data set $\mathcal{D}_{\text{IB}}$, with 70% training and 30% validation data for early stopping. The training process was repeated eight times, and the networks with the lowest validation errors were chosen. Empirically it has been validated that the training process of these RNNs is stable and the results depend little on the chosen learning algorithm.

Two different types of neural approximation models have been trained, $\tilde{g}_c$ and $\tilde{g}_f$, for each of the reward relevant variables, consumption $c_{t+1} = \tilde{g}_c(o_{ct})$ and fatigue $f_{t+1} = \tilde{g}_f(o_{f_t})$. In the first type, called *no self input* setting, the models had to predict their respective variable without getting that variable as input in the networks, i.e. $o_{ct} = o_{f_t} = o_t \backslash \{c_t, f_t\}$, with o_t the full IB observation. Experiences with real industrial applications have shown, that under some circumstances negative feedback effects can occur and corrupt the long-term predictions in rollout settings. In contrast, for other learning tasks the prediction quality could be dramatically increased, if the model received a history of the variable it had to forecast, called the *self input* setting, i.e. $o_{ct} = o_t \backslash \{f_t\}$

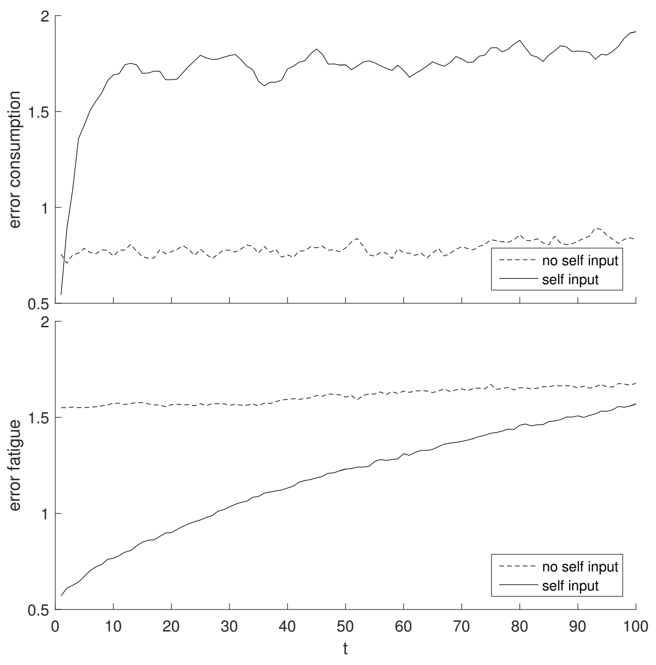


Figure 3.7: Error comparison of IB system models $\tilde{g}_c$ and $\tilde{g}_f$

and $o_{f_t} = o_t \backslash \{c_t\}$. For this experiment, both types of RNNs have been compared by rolling out randomly drawn trajectories included in $\mathcal{D}_{\text{IB}}$ on $\tilde{g}_c$ and $\tilde{g}_f$, and calculating the average absolute errors in each step t with respect to the true variables' future values given by the observations in $\mathcal{D}_{\text{IB}}$. While the network's prediction accuracy of consumption dramatically collapses after a few time steps when self input is applied, the prediction error remains almost at the same level if no self input is given to $\tilde{g}_c$. In contrast to that, the prediction of fatigue benefits from seeing historical fatigue values, at least in the evaluated period of $t < 100$. As a result, consumption is forecasted without self input and fatigue with self input. Fig. 3.7 depicts the squared error of the two selected RNNs with respect to the true IB values and the *no self input* and *self input* design of both RNNs.

CHAPTER 4

Interpretable Controllers by Classical Control Theory

In this chapter, a controller class from classical control theory, namely proportional–integral–derivative (PID) controllers, is introduced and applied to one of the benchmarks of this thesis. Since PID controllers traditionally have been parameterized by iterating response observation and adapting the respective parameters, they are generally designed to be interpretable for humans. Especially in industry settings, PID control is still a very important and common strategy in order to realize stable, efficient, and understandable control strategies.

First theoretical analysis and practical application of PID control date back to 1922 [99]. However, even 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 R. 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 [28].

A PID controller is a control loop feedback mechanism which continuously calculates an error value as the difference between a setpoint and a measured process variable. Based on proportional, integral, and derivative terms of this error value a corrective action is computed and applied to the system [87]. Each of the three PID terms contributes to the action based on its individual gain values k_P, k_I, and k_D for proportional, integral, and derivative terms, respectively. The tuning of these terms is the central task in PID controller design. Hence, a huge number of literature exists on how to tune PID controllers for various application domains [143, 134, 97, 139, 95, 23, 142, 51, 96, 141].

One of the most common prescriptive rules used in manual PID tuning are the Ziegler-Nichols response methods from 1942 [163]. Ziegler and Nichols simulated a large number of different processes and correlated the control actions with the features of the system response, to achieve an amplitude ratio of 25%. While the first method is applied to plants with step responses, the second method targets unstable plants in a closed loop system.

Manually tuning a PID controller using Ziegler-Nichols second method can be achieved by applying the following steps:

1. Set $k_{\mathrm{I}} = 0$ and $k_{\mathrm{D}} = 0$;
2. Increase k_{P} from 0 until the measured process variable is oscillating;
3. Set the critical value k_{C} to the current value of k_{P};
4. Set the critical period length p_{C} to the observed period length of the oscillation;
5. Calculate: $k_{\mathrm{P}} = 0.6k_{\mathrm{C}}$, $k_{\mathrm{I}} = 2k_{\mathrm{P}}/p_{\mathrm{C}}$, $k_{\mathrm{D}} = k_{\mathrm{P}}p_{\mathrm{C}}/8$.

Note that this method is known to have severe drawbacks. It uses only insufficient process information and the design criterion sometimes yields closed loop systems with poor robustness [4].

The three different terms of a PID controller can be interpreted as follows:

- Proportional: A high proportional gain k_{P} results in a large change in the output for a given change in the error. While setting the proportional gain too high may result in an unstable system, setting the gain too small may result in a less responsive controller.
- Integral: The integral term is applied to introduce knowledge from previous observations by accumulating the offset that should have been corrected previously, and multiplying it with the integral gain k_{I}. This term is used to accelerate the movement of the process towards its setpoint by eliminating the residual steady-state error that stems from the proportional controller.
- Derivative: The derivative term is applied to introduce a system prediction for the next time step. Computing (or estimating) the derivative of the current process response and multiplying it with the derivative gain k_{D} yields a control term which can improve both settling time and stability of the system.

In the following section, experiments have only been conducted with the cart-pole balancing (CPB) problem. The reason for that is that inherent limitations of standard PID control hinder the application on the other three benchmark problems: mountain car (MC), cart-pole swing-up (CPSU), and industrial benchmark (IB).

PID controllers are not only linear, but in addition to that, they are symmetric around the setpoint. The setpoint of the MC problem is defined as the top of the mountain at $\rho = 0.6$, whereas the valley at $\rho = -0.5$ is the system's equilibrium point. Hence, oscillating symmetrically around the setpoint of MC is not possible. Moreover, to successfully solve the MC benchmark, it is necessary to initially conduct actions which drive the car further away from its goal, in order to gain sufficient potential energy. Planning policies that can realize such a behavior cannot be represented as regulating PID controllers.

For the CPSU, Raiko and Tornio showed that a globally linear controller is generally incapable to swing the pendulum up and balance it in the inverted position [109]. In classical control theory, the CPSU problem is solved by using two separate controllers: one for the swing-up and the second one for the balancing task. However, this approach requires an intricate understanding of the dynamics of the system (parametric system identification) and of how to solve the task (switching controllers) [26].

Since PID control is a reactive feedback control system with constant parameters and no direct knowledge of the process, e.g., from additional system state observations, it is not applicable to the IB problem. Moreover, changing process behavior, caused by delayed effects included in the IB, can only be tackled by choosing a trade-off between regulation and response time with PID control. Furthermore, the presence of high noise in the IB's measurable process variables, consumption and fatigue, can be amplified by the derivative term and cause huge changes in the output.

4.1 Cart-pole Balancing Experiments with PID Control

In the CPB problem, the two process variables θ and ρ have to be regulated to their respective setpoints $\theta = 0$ and $\rho = 0$. To achieve this goal using PID control theory, a controller layout consisting of two independent PID controllers can be utilized [154]. Fig. 4.1 depicts the controller diagram. The error values $e_\theta(t)$ and $e_\rho(t)$ are determined separately before they are passed further to the respective PID terms. The terms for the angular controller are computed as follows:

$$\mathrm{P1} = k_{\mathrm{P1}} e_\theta(t), \tag{4.1}$$

$$\mathrm{I1} = k_{\mathrm{I1}} \int_{t-T1}^{t} e_\theta(t) dt, \tag{4.2}$$

$$\mathrm{D1} = k_{\mathrm{D1}} \frac{de_\theta(t)}{dt}. \tag{4.3}$$

Similarly, the terms for the position controller are computed by:

$$\mathrm{P2} = k_{\mathrm{P2}} e_\rho(t), \tag{4.4}$$

$$\mathrm{I2} = k_{\mathrm{I2}} \int_{t-T2}^{t} e_\rho(t) dt, \tag{4.5}$$

$$\mathrm{D2} = k_{\mathrm{D2}} \frac{de_\rho(t)}{dt}. \tag{4.6}$$

Hence, the final output of the controller is determined by

$$a(t) = k_1(\mathrm{P1} + \mathrm{I1} + \mathrm{D1}) + k_2(\mathrm{P2} + \mathrm{I2} + \mathrm{D2}), \tag{4.7}$$

where k_1 and k_2 are gains to balance the contributions from θ and ρ, respectively.

Tuning this PID controller for CPB using the manual method from Ziegler and Nichols is a feasible yet laborious process. It requires multiple simulation runs using the

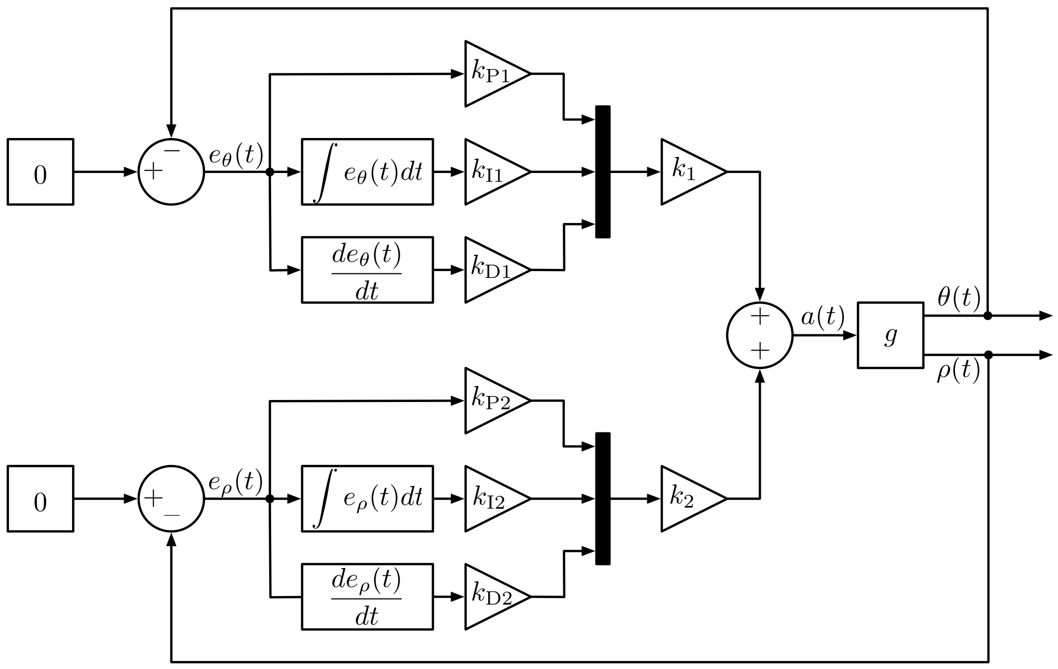
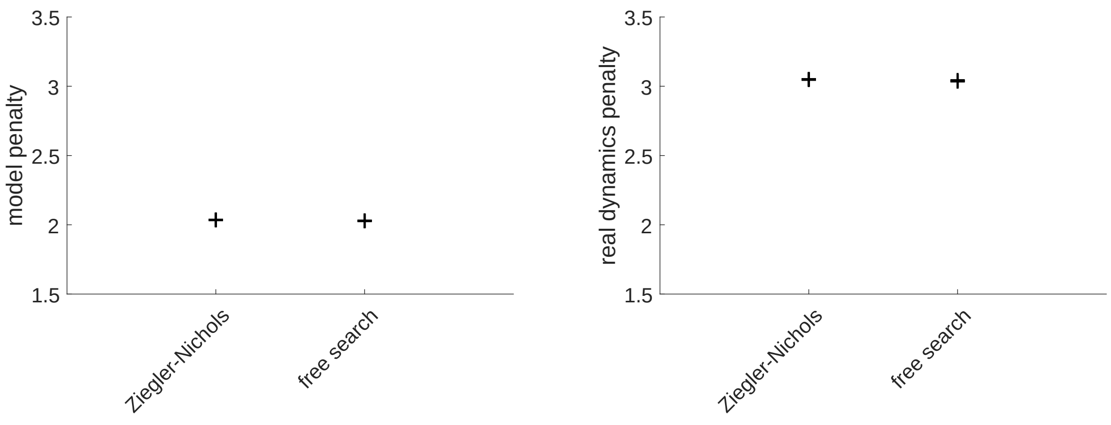


Figure 4.1: A PID controller for CPB [154]

surrogate model of the CPB and subsequent investigations of the produced trajectories in order to converge to an adequate control performance. For this reason, the process of searching for optimal Ziegler-Nichols parameters $(k_{\mathrm{C1}}, p_{\mathrm{C1}}, T_1, k_1, k_{\mathrm{C2}}, p_{\mathrm{C2}}, T_2, k_2)$ is performed by particle swarm optimization (PSO) (Section 2.2). In a second experimental setup, PSO is used to search for the PID gains $(k_{\mathrm{P1}}, k_{\mathrm{I1}}, k_{\mathrm{D1}}, T1, k_1, k_{\mathrm{P2}}, k_{\mathrm{I2}}, k_{\mathrm{D2}}, T2, k_2)$ directly. For both experimental setups, a PSO swarm of size 1 000 particles was allowed to search for 1 000 iterations. Both experiments have been repeated ten times. Each training employed 100 start states $s = (\theta, \dot{\theta}, \rho, \dot{\rho})$ that were uniformly sampled from $[-0.5, 0.5] \times \{0\} \times [-2.0, 2.0] \times \{0\}$.

Fig. 4.2 shows clearly that searching for PID controllers using Ziegler-Nichols configuration yields results of the same penalty values as searching the full space of possible PID gains freely. Applying the resulting PID controllers to the real system dynamics during evaluation shows a good performance, although other methods presented in this work achieved even lower penalty values. A potential reason for this suboptimality can be observed in Fig. 4.3. Even though the controller stabilizes the system for the majority of the states in less than 100 time steps, it tends to overshoot. This results in a damped oscillation which requires significantly longer to reach the goal area stably.



(a) Model (b) Real system dynamics

Figure 4.2: Results of PID experiments for the CPB benchmark

The best controller found by the Ziegler-Nichols tuning is determined by:

$$\begin{aligned} a(t) =& 0.95 \left(0.6k_{C1} \cdot e_\theta(t) + 1.2k_{C1}/p_{C1} \cdot \int_{t-40}^{t} e_\theta(t)dt + 0.6k_{C1}p_{C1}/8 \cdot \frac{de_\theta(t)}{dt} \right) \\ &+ 0.05 \left(0.6k_{C2} \cdot e_\rho(t) + 1.2k_{C2}/p_{C2} \cdot \int_{t-28}^{t} e_\rho(t)dt + 0.6k_{C2}p_{C2}/8 \cdot \frac{de_\rho(t)}{dt} \right), \end{aligned} \tag{4.8}$$

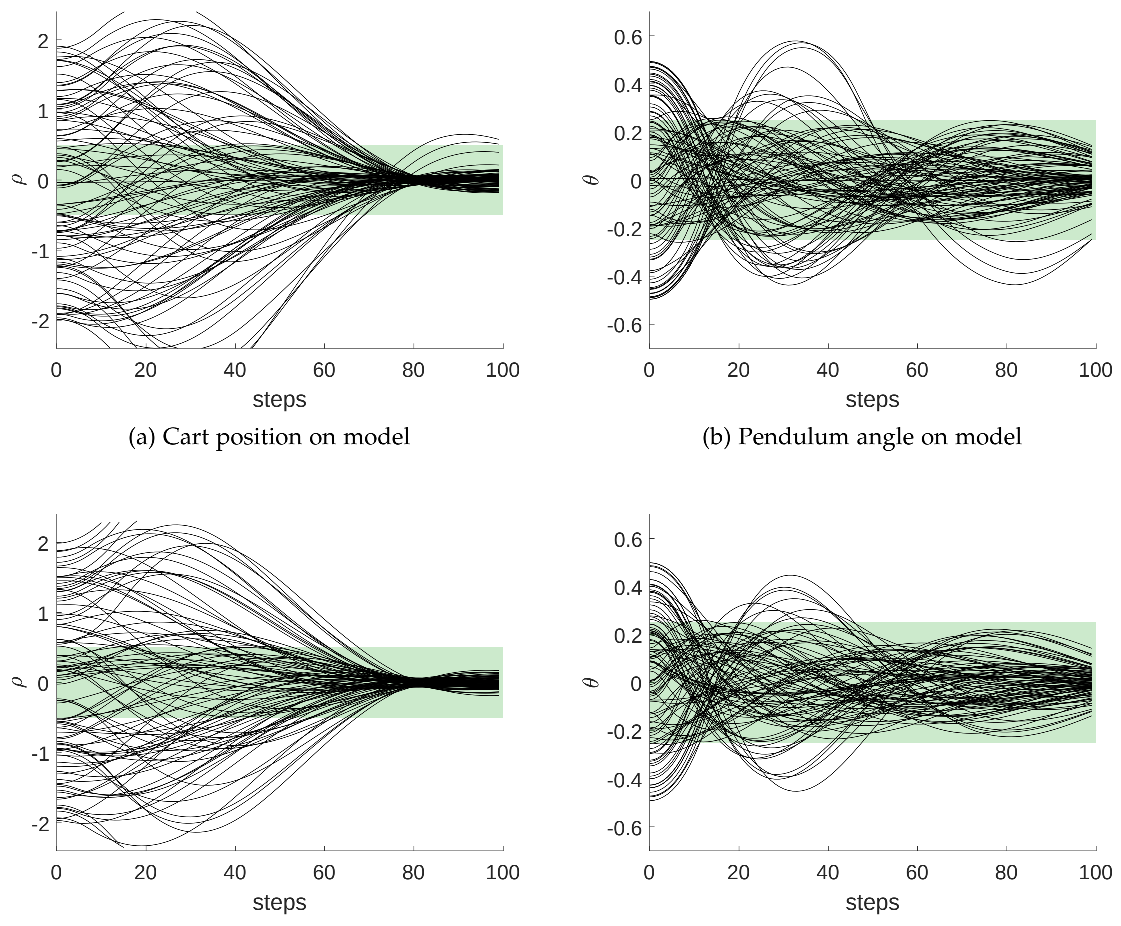
with critical values $k_{C1} = -14.3$ and $k_{C2} = -42.7$ and critical periods $p_{C1} = 113$ and $p_{C2} = 331$. This yields the following PID controller:

$$\begin{aligned} a(t) =& 0.95 \left(-8.6 \cdot e_\theta(t) - 0.15 \cdot \int_{t-40}^{t} e_\theta(t)dt - 120.9 \cdot \frac{de_\theta(t)}{dt} \right) \\ &+ 0.05 \left(-25.6 \cdot e_\rho(t) - 0.15 \cdot \int_{t-28}^{t} e_\rho(t)dt - 1061.4 \cdot \frac{de_\rho(t)}{dt} \right). \end{aligned} \tag{4.9}$$

The PSO search for the best PID controller in the full space of possible tuning gains resulted in the following best controller:

$$\begin{aligned} a(t) =& 0.94 \left(-10.1 \cdot e_\theta(t) - 2.1 \cdot \int_{t-31}^{t} e_\theta(t)dt - 85.8 \cdot \frac{de_\theta(t)}{dt} \right) \\ &+ 0.06 \left(-60.8 \cdot e_\rho(t) + 40.6 \cdot \int_{t-30}^{t} e_\rho(t)dt - 113.6 \cdot \frac{de_\rho(t)}{dt} \right). \end{aligned} \tag{4.10}$$

Note that both methods learned very similar contribution gains k_1 and k_2 while the gains of the PID terms differ substantially. However, the resulting median penalties are on the same level, which implies that it is sufficient to use the Ziegler-Nichols manual tuning method to achieve CPB controllers of high performance.



(a) Cart position on model

(b) Pendulum angle on model

(c) Cart position on real system dynamics

(d) Pendulum angle on real system dynamics

Figure 4.3: State trajectories for CPB produced by the best PID controller

4.2 Discussion

The experiments with CPB show that for certain problems it is possible to create classical PID controllers of good performance. Such PID controllers are designed to be of a human-interpretable form because classically its different gains are tuned manually by using theoretical knowledge about the specific plant and empirical parameter studies incorporating response observation and stepwise parameter tuning. Tuning heuristics, like Ziegler-Nichols methods, aim at making the elaborate process of PID controller design more convenient and straightforward.

Control theory experts are able to interpret the setup of a specific PID controller by analyzing the three different terms. i.e., proportional, integral, and derivative gains. This property yields stable, efficient, and understandable control strategies.

However, the field of application of regulating PID controllers is rather limited, since they are only linear, symmetric around the setpoint, do not incorporate planning, and cannot utilize direct knowledge of the process, for example. Therefore, only one of four benchmark problems of this thesis could be solved using PID control theory.

Reinforcement learning (RL) (Section 2.1) on the other hand, is theoretically capable of solving any Markov decision process (MDP) which makes it applicable to a dramatically broader field of application compared to classical PID control. Moreover, since RL is not limited to linear control, can incorporate the full observation of the system, and is able to solve planning tasks, it is capable of outperforming PID controllers even in classical industrial applications [10]. The following chapters are concerned with producing RL policies, starting from non-interpretable implicit policies (Section 5.1) through to interpretable explicit policies represented by basic algebraic equations (Chapter 8).

CHAPTER 5

From Model-free Implicit to Model-based Explicit Policy Learning

In this chapter, the transition from model-free implicit to model-based explicit policy representations is motivated and evaluated. Sections 5.1-5.3 introduce the non-interpretable RL methods neural fitted Q iteration (NFQ), particle swarm optimization policy (PSO-P), and particle swarm optimization neural network (PSONN). The conducted experiments with each method are discussed in each section, respectively.

5.1 Q Iteration Based Model-free Policy Learning

In this section, the results of applying NFQ to the same problems using the same data sets and approximative models as the novel interpretable RL approaches, presented later in the thesis, are discussed. NFQ was chosen because it is a well-established, widely applied, and well-documented RL methodology. In this thesis, the aim is not to claim the proposed interpretable approaches are superior to the best RL algorithms regarding performance; thus, NFQ was selected to show both the degree of difficulty of the selected benchmarks and the advantages and limitations of the proposed methods. However, recent developments in Q-based deep RL have produced remarkable results with image-based online RL benchmarks [140, 152], and future studies may reveal that their performance with batch-based offline problems is superior to that of NFQ.

Fig. 5.1 depicts the basic concept of the NFQ framework. Transition samples are utilized by the algorithm to yield the value function of the optimal policy. Note that despite NFQ is considered a model-free RL approach, using a model for policy selection has proven to improve the performance of the resulting policies significantly.

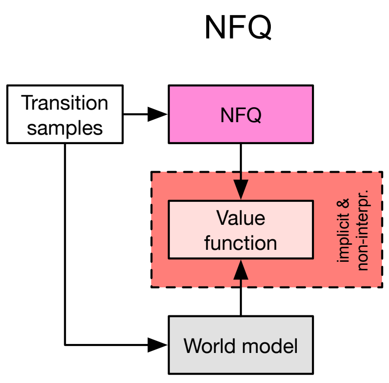


Figure 5.1: The NFQ framework

5.1.1 Neural Fitted Q Iteration (NFQ)

NFQ is an algorithm for the efficient and effective training of an action-value function, called Q function. NFQ uses the ability of neural networks (NNs) to approximate non-linear functions to represent a value function [116]. A huge advantage of using a global function representation, like with NNs (Section 3.2.1), is that it can exploit generalization effects by assigning similar values to related areas. However, since changing the weights of NNs has a global effect on the value function output, many difficulties have been reported [12]. In classical online RL, the value function is updated as soon as a new state-action experience has been made. However, adopting the weights of the NN according to this recent sensation does usually not only change the value of this state-action pair and its related area, but also has an unpredictable influence on values of unrelated state-action examples. With NFQ the malicious influence of a new update is constraint by offering previous knowledge explicitly.

NFQ belongs to the family of fitted value iteration algorithms, and is a special realization of the *fitted Q iteration* (FQI) approach proposed by Ernst, Geurts, and Wehenkel [39]. In its original form, NFQ is a model-free RL approach. This means no explicit model of the environment has to be learned, which makes the method for many applications data-efficient since transition tuples from the plant are directly used to find well-performing control policies. Generally, the method is also applicable for batch or offline RL problems, where policies cannot be evaluated during the training and consequently cannot generate new transition samples. However, the following experiments show that despite the training process of NFQ is referred to as model-free, using an environment model to evaluate and select adequate policies during training is very useful to yield adequate policy performance.

From classical Q-learning as introduced in Section 2.1.1 the update rule for the action-value function is given by:

$$q_{k+1}(\boldsymbol{s}_t, \boldsymbol{a}_t) \leftarrow r_{t+1} + \gamma \max_{\boldsymbol{a}_{t+1}} q_k(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}), \tag{5.1}$$

where after every new state-action transition perceived in tuple form $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, r_{t+1})$, the Q function is immediately updated. Under mild assumptions Q-learning using averagers as approximators converges for finite state and action spaces, as long as every state action pair is updated infinitely often. To adapt Q-learning for NNs an error function can be derived from (5.1) to measure the difference between $q_k(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and $q_{k+1}(\boldsymbol{s}_t, \boldsymbol{a}_t)$:

$$e_{k+1} = \left(q_{k+1}(\boldsymbol{s}_t, \boldsymbol{a}_t) - \left(\boldsymbol{r}_{t+1} + \gamma \max_{\boldsymbol{a}_{t+1}} q_k(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) \right) \right)^2 . \tag{5.2}$$

Note that minimizing this mean squared error by common gradient descent techniques for adjusting the weights after each new sample, can cause unpredictable effects on other areas in the state-action space. Therefore, Riedmiller proposed to update the neural function offline on the entire set of previous transitions. By doing so, more advanced and reliable learning methods like Rprop [118] could be employed.

5.1.2 Experiments with Neural Fitted Q Iteration

In its classical form, NFQ is designed for continuous state spaces, while action set $\mathcal{A}$ is assumed to be finite. The reason for that is that determining the optimal action $\boldsymbol{a}$ to be taken in state $\boldsymbol{s}$ requires the evaluation of $q(\boldsymbol{s}, \boldsymbol{a})$ for all $\boldsymbol{a} \in \mathcal{A}$ to decide which action yields the maximum value in $\boldsymbol{s}$. Although all four of our benchmarks are configured to accept continuous actions, it is known from the literature that MC, CPB, as well as CPSU benchmarks, can be successfully controlled with discrete actions. However, it is possible that applying only discrete actions might be a disadvantage compared to policies using continuous actions in terms of maximum reward. The IB has been specifically designed to require continuous actions to yield optimal control trajectories. For this reason, particle swarm optimization (PSO) is used to search the continuous three-dimensional action space for actions yielding maximum value for every state, in every NFQ iteration. However, this procedure comes at the cost of high computational effort and still does not guarantee to find the globally optimal actions, since the search space represented by the neural Q function is expected to be highly non-linear and non-convex.

For the following experiments, the NFQ implementation from the RL *teachingbox*[1] toolbox has been applied [40]. For training the NN weights, the Rprop optimization heuristic has been utilized in a full batch training setting. Table 5.1 lists the relevant NFQ and benchmark configuration parameters for the following experiments. The performance in this section is presented as penalty values computed by multiplying the achieved fitness values by -1.

Mountain Car

Ten independent NFQ trainings have been conducted for the MC benchmark. The topology of the NN which approximates the Q function was defined as 3-20-20-1, i.e.,

[1]Freely available at https://sourceforge.net/projects/teachingbox.

	MC	CPB	CPSU	IB
NN Q function	3-20-20-1	5-20-20-1		183-20-20-1
Activation function	$f(x) = \max(0, x)$			
NFQ iterations	100			
Transition samples	10 000			100 000
Time horizon T	200	100	500	100
Discount factor γ	0.985	0.97	0.994	0.97
Model usage	for policy selection			

Table 5.1: NFQ experiments settings and parameters

three neurons as input $(\rho, \dot{\rho}, a)$, 20 neurons on each hidden layer, and one neuron as output predicting the respective value. While input and output layers applied linear activation functions, the neurons on the hidden layers got activated by a rectifier function, i.e., $f(x) = \max(0, x)$. For the MC benchmark 100 NFQ iterations using $\gamma = 0.985$ have been employed on a previously generated transition data batch $\mathcal{D}_{\text{MC}}$ of size 10 000. The discrete actions for the NFQ training have been set to $\{-1, 0, 1\}$.

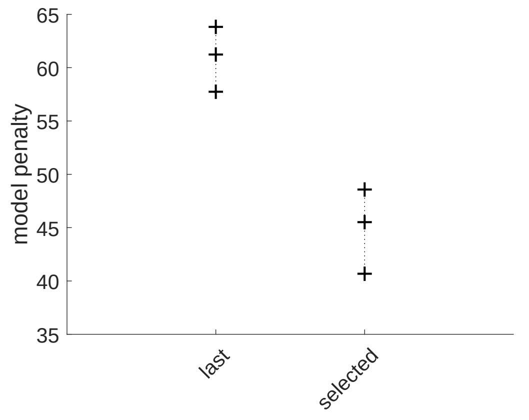
Although NFQ is designed to be a model-free method, the experiments in this section show that using a system model for policy selection can improve the method's performance significantly. Fig. 5.2 clearly shows that selecting the most promising policy using a performance prediction of some model yields significantly better control performance on the real system dynamics. After each NFQ iteration, the latest policy was tested on the world model described in Section 3.2 to compute an approximation of the real performance. The policy yielding the best fitness value so far was saved as an intermediate solution. Note that here the very same surrogate models have been used as for the model-based RL methods later in the thesis.

Comparing the state trajectories produced by applying the best policy from ten NFQ trainings on the model as well as on the real dynamics shows a very similar behavior (Fig. 5.3). The presented policy is able to drive the car up the mountain from each given initial state in less than 200 time steps.

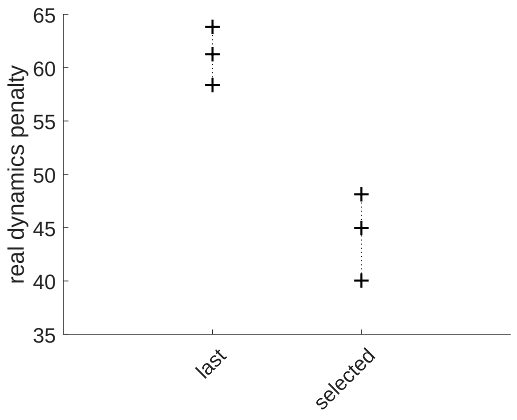
Cart-pole Balancing

Ten independent NFQ trainings have been conducted for the CPB benchmark. The topology of the NN which approximates the Q function was defined as 5-20-20-1, i.e., five neurons as input $(\rho, \dot{\rho}, \theta, \dot{\theta}, a)$, 20 neurons on each hidden layer, and one neuron as output predicting the respective value. For the CPB benchmark 100 NFQ iterations using $\gamma = 0.97$ have been employed on a previously generated transition data batch $\mathcal{D}_{\text{CPB}}$ of size 10 000. The discrete actions for the NFQ training have been set to $\{-10, 0, 10\}$.

Similarly to the MC experiments, the best policies have been stored during the NFQ training based on their predicted performance calculated on world models. As for

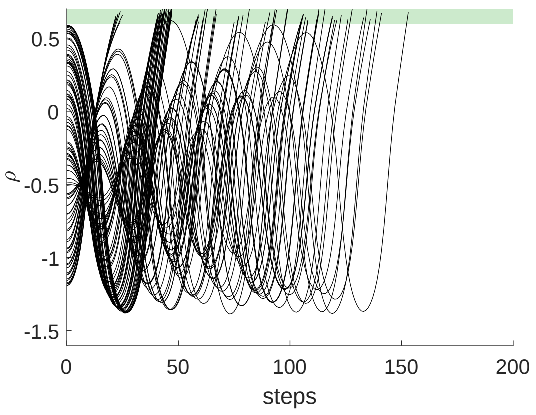


(a) Model

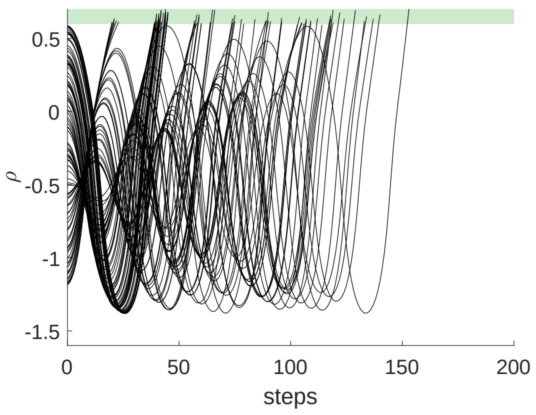


(b) Real system dynamics

Figure 5.2: Results of NFQ experiments for the MC benchmark. The markers represent (from top to bottom) the worst, median, and best NFQ result after ten independent training runs. Using a world model to select the best produced policies during the NFQ trainings, instead of just using the last results, significantly increases the performance applying NFQ solutions on the real system dynamics afterward.

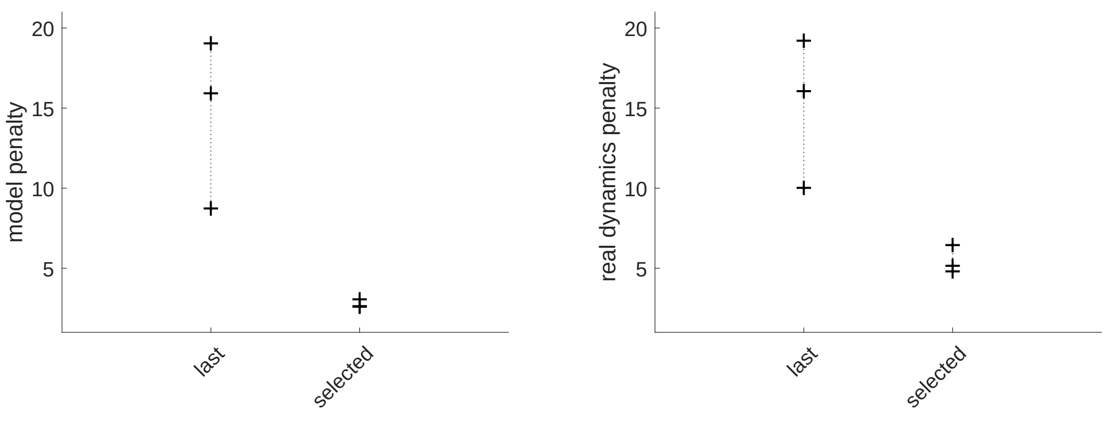


(a) Model



(b) Real system dynamics

Figure 5.3: State trajectories for MC produced by the best NFQ policy. The green area represents the goal area of the MC benchmark in which the episode is declared successful. The predicted trajectories from the world model are very similar to the trajectories produced on the real system dynamics.



(a) Model (b) Real system dynamics

Figure 5.4: Results of NFQ experiments for the CPB benchmark. Using a world model to select the best produced policies during the NFQ trainings, instead of just using the last results, significantly increases the performance applying NFQ solutions on the real system dynamics afterward. The higher penalties for the selected policies on the real dynamics in comparison to the world model indicate that the model makes small inaccuracies during its predictions which lead to an overestimation of the policies' performance.

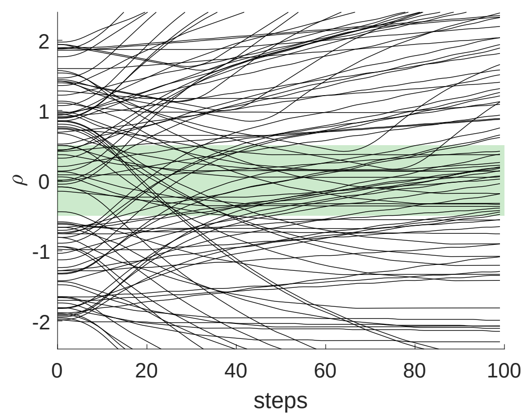
the MC benchmark, policy selection improved the performance on the real system dynamics for the NFQ results significantly (Fig. 5.4). However, the models slightly tend to overestimate the performance of the selected policies. The reason for this might stem from model inaccuracies.

In Fig. 5.5 the state trajectories for 100 initial states produced on the world model (first row) are compared to the state trajectories on the real dynamics (second row). The prediction of the model is very close to the real dynamics. However, another important observation can be made. The policies have only learned to balance the pole in the goal area (the green area in Fig. 5.5b and 5.5d). They completely ignore the cart's position for most of the states, which yields a relatively high penalty since the maximum reward is only achieved if both position and angle are in their respective goal areas. Note that for many states the policy even drives the cart over the state boundaries ($-2.4 \leq \rho \leq 2.4$), which produces failed episodes.

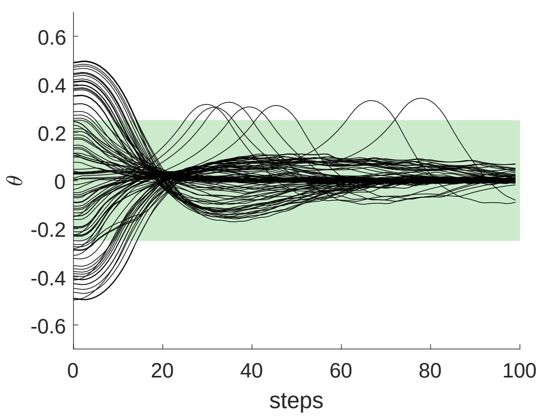
The experiments show that even the best policies trained by NFQ and selected by an adequate world model are not capable of controlling the CPB benchmark acceptably.

Cart-pole Swing-up

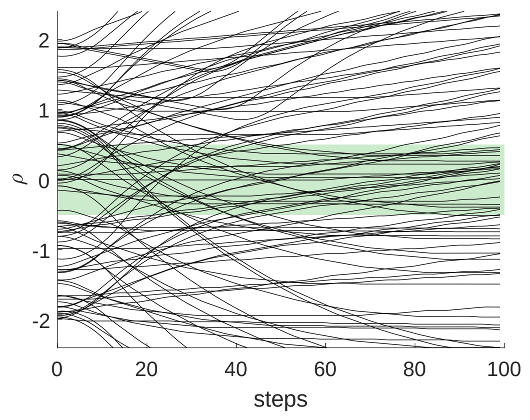
Ten independent NFQ trainings have been conducted for the CPSU benchmark. The topology of the NN which approximates the Q function was defined as 5-20-20-1, i.e., five



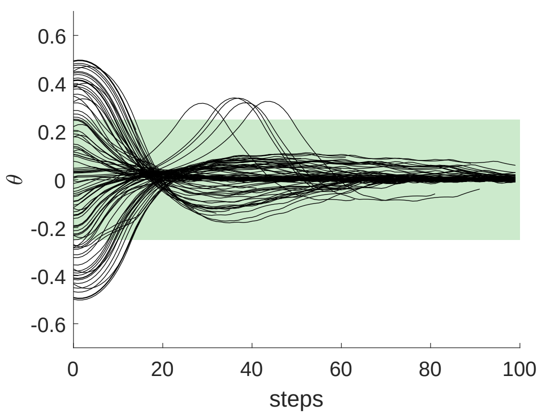
(a) Cart position on model



(b) Pendulum angle on model

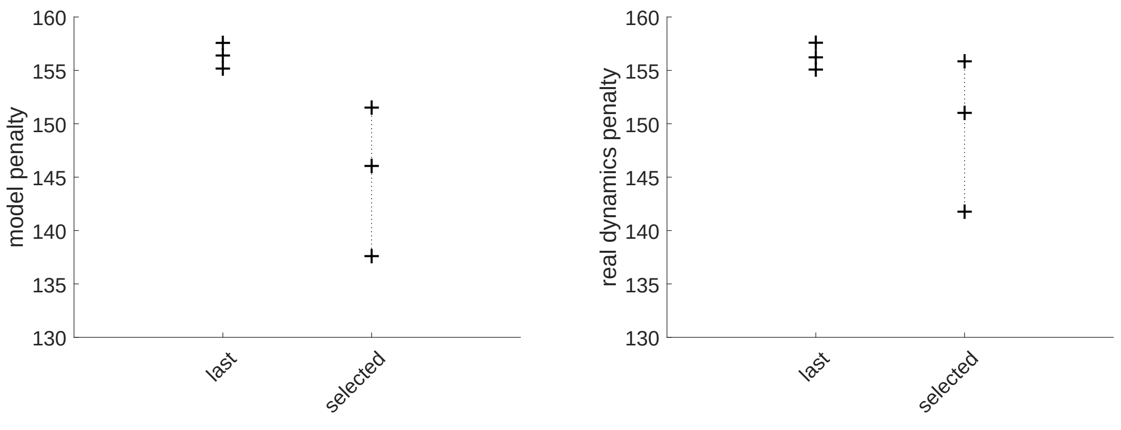


(c) Cart position on real system dynamics



(d) Pendulum angle on real system dynamics

Figure 5.5: State trajectories for CPB produced by the best NFQ policy. The policy cannot successfully control the CPB system since it completely ignores the cart's position. Not only, does it not head for the goal region (green area), but on top of that it crosses the position boundaries and produces failed episodes. Stabilizing the pendulum angle on the other hand, has been learned for all given initial states.



(a) Model (b) Real system dynamics

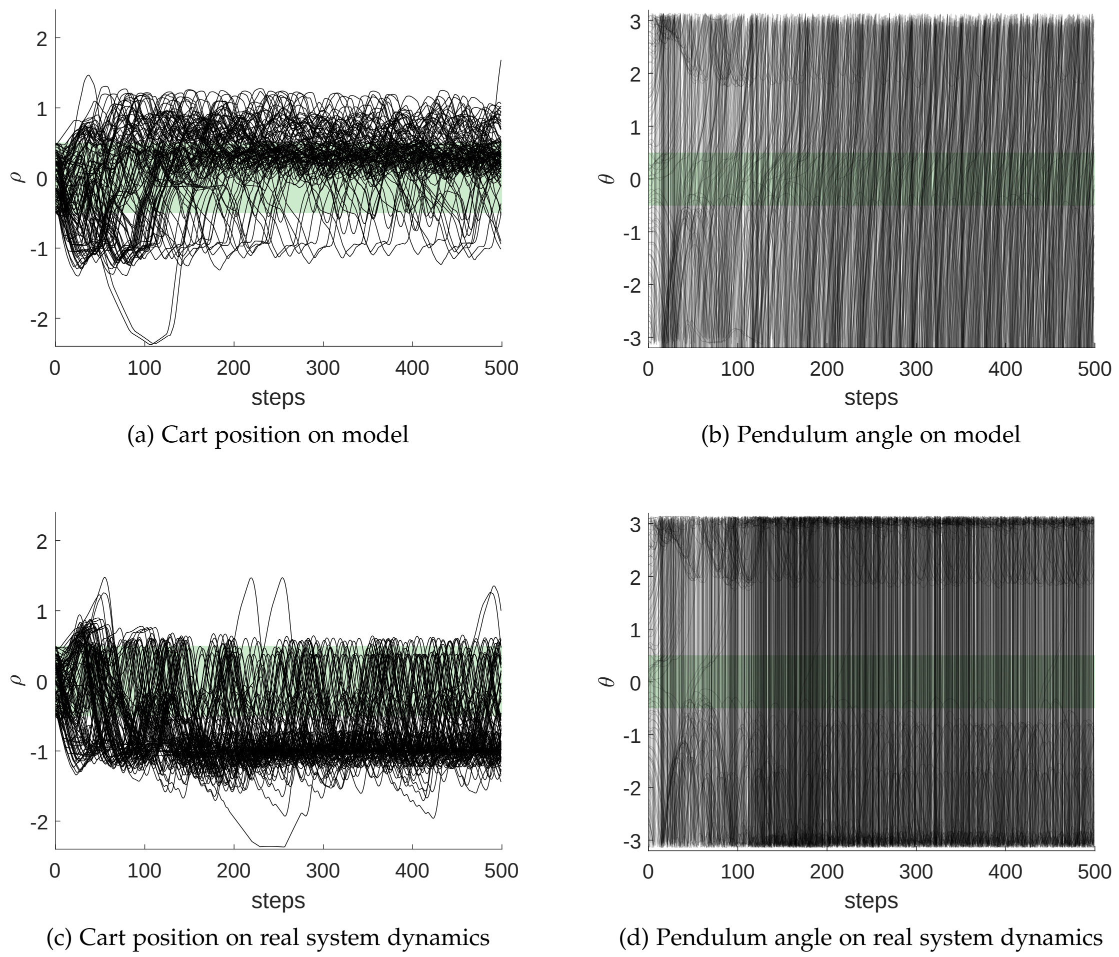
Figure 5.6: Results of NFQ experiments for the CPSU benchmark. Although the performances of the model-selected policies are better compared to the policies of the last NFQ iterations, penalties of 135 and above are still not good results for CPSU. None of the produced NFQ policies was able to swing-up the pole and balance it in the state's goal region.

neurons as input $(\rho, \dot{\rho}, \theta, \dot{\theta}, a)$, 20 neurons on each hidden layer, and one neuron as output predicting the respective value. For the CPSU benchmark 100 NFQ iterations using $\gamma = 0.994$ have been employed on a previously generated transition data batch $\mathcal{D}_{\text{CPSU}}$ of size 10 000. The discrete actions for the NFQ training have been set to $\{-30, 0, 30\}$.

Like for the MC and CPB benchmarks before, using a world model to select the best policies during an NFQ run produced the best results during the evaluation on the real dynamics. However, for the CPSU no well-performing policy could be learned using NFQ. Fig. 5.6 shows that only penalties of 135 or above could be achieved, which is considered a bad performance for CPSU. Good policies that can swing-up the pole and keep it balanced in the goal area are expected to produce a penalty of around 50 or below.

The state trajectory plots of the best NFQ policy depicted in Fig. 5.7, show how unsatisfying the performance of the produced NFQ policies is. The strategy found by the depicted policy can be described as "rotate the pole as quickly as possible and keep the cart around the position's goal region." This strategy is better than doing nothing, but of course, it is obviously suboptimal. A reward is only received in the few time steps when the pole is rotating through the angle's goal region while the cart is coincidentally located in the position's goal region.

In summary, it can be said, therefore, that NFQ was not able to produce a well-performing policy. Even using a model for policy selection did only yield suboptimal policies as learning results.



(a) Cart position on model

(b) Pendulum angle on model

(c) Cart position on real system dynamics

(d) Pendulum angle on real system dynamics

Figure 5.7: State trajectories for CPSU produced by the best NFQ policy. The policy that produced the depicted trajectories found a suboptimal solution for the CPSU problem, which keeps the cart close to the goal region and rotates the pole as fast as possible. By doing so it is able to collect a few rewards during its runtime.

Industrial Benchmark

Ten independent NFQ trainings have been conducted for the IB. The topology of the NN which approximates the Q function was defined as 183-20-20-1, i.e., 183 neurons as input $(\boldsymbol{o}_{t-29}, \boldsymbol{o}_{t-27}, \ldots, \boldsymbol{o}_t, \boldsymbol{a}_t)$, 20 neurons on each hidden layer, and one neuron as output predicting the respective value. For the IB 100 NFQ iterations using $\gamma = 0.97$ have been employed on a previously generated transition data batch $\mathcal{D}_{\text{IB}}$ of size 100 000.

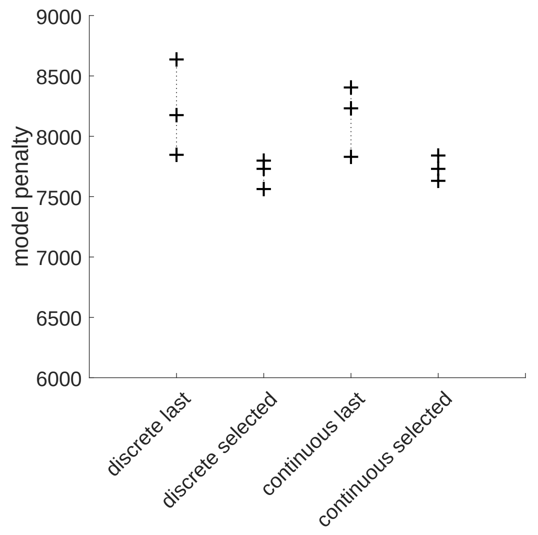
The action space of the IB is defined as $\boldsymbol{a}_t = (\Delta v_t, \Delta g_t, \Delta h_t) = [-1, 1]^3$ which describes a three-dimensional continuous action space. To find the action which maximizes the Q function of Eq. 5.2, a PSO search has been applied. A swarm of 100 particles performed 100 iterations for each training sample. Note that this optimization has to be performed during training as well as during runtime. In real-time scenarios where the next action has to be instantly available as soon as a new state is perceived, this procedure might be infeasible since the search takes too long. However, like for the toy benchmarks before, a discretization of the action space is a reasonable approach to eliminate time-consuming optimization runs. Here, we used 27 discrete actions represented by $\boldsymbol{a}_t = \{-1, 0, 1\}^3$. A suboptimal performance is expected by using discrete actions since the IB is specifically designed to run best with continuous actions. However, the results of the experiments presented in Fig. 5.8 show that the use of continuous actions did not yield any significant performance gain for the NFQ policies. Like for the toy benchmarks before, the best results could be achieved by employing a model for policy selection.

Fig. 5.9 depicts the resulting reward trajectories of the best NFQ policy. It is interesting that the policy performs visibly worse on the world model compared to the real system dynamics. An explanation might be that because this policy did not see the model during training and consequently has not been optimized to perform particularly well on it. However, since the model seems to predict slightly different trajectories than the real dynamics, the policy does not yield an optimal reward. Nevertheless, NFQ used the real transition samples from the real dynamics to train its policies, which is the reason why the performance is still good during evaluation.

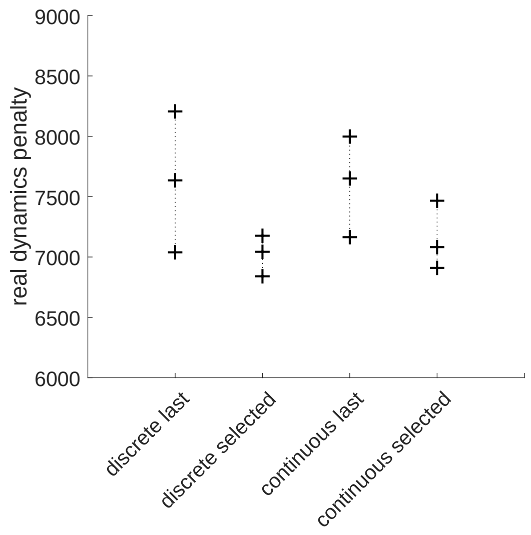
5.1.3 Discussion of the NFQ Results

The experiments with NFQ have shown that model-free RL can face severe difficulties in batch-based learning settings. Only for the simple MC benchmark well-performing policies could be learned. However, these good results were only achievable by applying a world model for policy selection. This observation could be made for all benchmarks. Running an NFQ training for a certain number of iterations and expecting it to converge to an optimal solution seems to be an unrealistic presumption.

Numerous publications exist which investigate the reasons for these often experienced difficulties. In practice, function approximation in RL is not known to converge to a point [54] and it is prone to overestimation of utility values [149]. Since NNs are not averagers, additional problems are likely to occur. To cover these problems, some improvements on Q iteration algorithms with NNs have been published in past years.

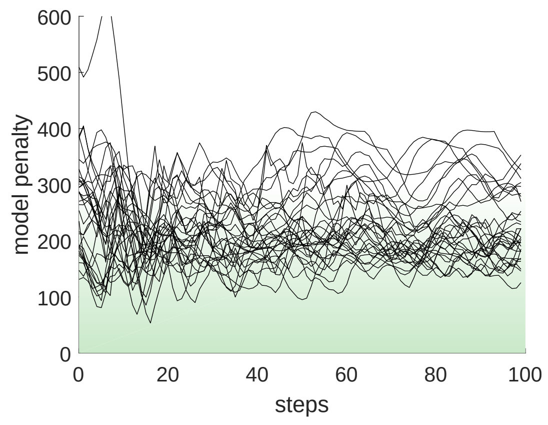


(a) Model

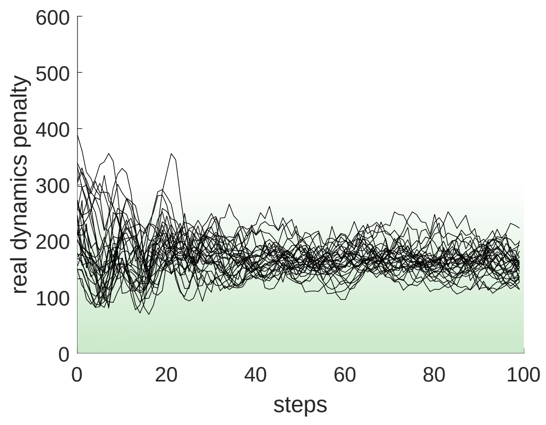


(b) Real system dynamics

Figure 5.8: Results of NFQ experiments for the IB. Using continuous actions did not improve the performance of the NFQ results. On the other hand, policy selection by a world model has proven again to be very important to yield the best NFQ results.



(a) Model



(b) Real system dynamics

Figure 5.9: Reward trajectories for IB produced by the best NFQ policy. No specific goal region is defined for the IB, so the green area illustrates the goal to reduce the respective penalties. For reasons of clarity and comprehensibility only states with setpoints between 40 and 60 are included in these plots. Note the difference between the model predicted reward trajectory and the respective trajectory from the real dynamics.

In [48] an RL method that monitors the learning process is presented. Furthermore, Hans and Udluft [59] and Faußer and Schwenker [42] applied ensembles of NNs to form a committee of multiple agents and showed that this committee benefits from the diversity on the state-action value estimations.

However, policy degradation over time is not the only problem which occurred during the presented experiments. For the CPB and CPSU benchmarks, NFQ never produced satisfying policies. For CPB the algorithm seems to overestimate the importance of balancing the pole compared to completely disregarding the cart's position. For the CPSU, NFQ only produced sub-optimal policies which try to rotate the pole as fast as possible to yield few good reward values from time to time.

The experiments in the following sections and chapters will show that model-based RL approaches perform significantly better than NFQ. In addition to that, policies created by model-based policy search can be represented in an explicit form, which forms the basis for interpretable RL policies as intended in this thesis.

5.2 Direct Generation of Optimal Trajectories

The experiments in Section 5.1 have shown that in batch RL settings even model-free learning approaches like NFQ benefit from utilizing a system model for policy selection. The performance of NFQ policies in a purely model-free training setup yielded unsatisfactory results for all of the investigated benchmark problems. From this observation, the question arises if it is better to use available system models not only for policy selection after the training but, on top of this, to use them already during the training itself. In this section, we investigate how exploiting such surrogate models for optimal actions and subsequently applying these actions to the real system performs using a method called PSO-P.

Fig. 5.10 depicts the framework of PSO-P in comparison to that of NFQ. In contrast to NFQ, PSO-P utilizes the available transition samples only for building a world model. This model is exploited for optimal action trajectories by applying PSO. Note that the policy consists of both, the model as well as the optimizer since even during runtime each action determination requires searching for optimal trajectories.

In RL, tuning the policy learning process is generally challenging for industrial problems. Specifically, it is hard to assess whether a trained policy has unsatisfactory performance due to inadequate training data, unsuitable policy representation, or an unfitting training algorithm. Determining the best problem-specific RL approach often requires time-intensive trials with various policy configurations and training algorithms. In contrast, it is often significantly easier to train a well-performing system model from observational data, compared to directly learning a policy and assessing its performance.

PSO-P is a heuristic for solving RL problems by employing numerical online optimization of control action sequences. As an initial step, a system model is trained from observational data with standard methods. The presented method works with NNs as well as with many other model types, e.g., Gaussian process or first principal

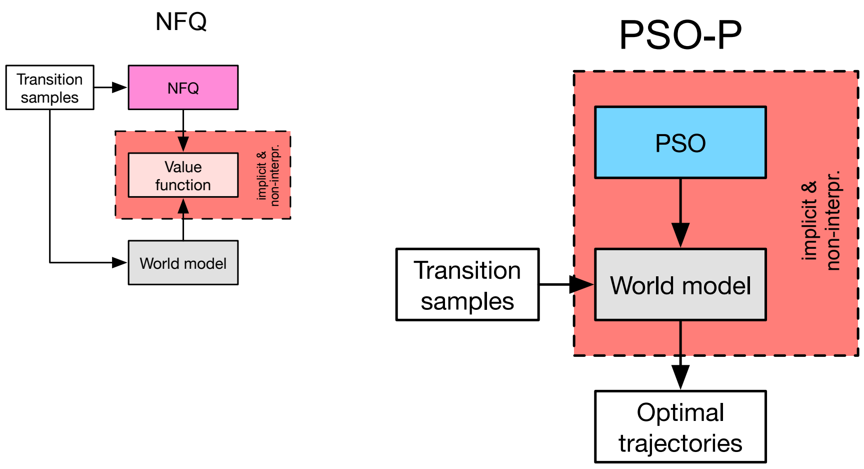


Figure 5.10: The PSO-P framework compared to NFQ

models. The resulting problem of finding optimal control action sequences based on model predictions is solved with PSO because PSO is an established algorithm for non-convex optimization which does not require any gradient information. Specifically, the presented heuristic iterates over the following steps. (i) PSO is employed to search for an action sequence that maximizes the expected return when applied to the current system state by simulating its effects using the system model. (ii) The first action of the sequence with the highest expected return is applied to the real-world system. (iii) The system transitions to the subsequent state and the optimization process is repeated based on the new state (go to step (i)).

As this approach can generate control actions for any system state, it formally constitutes an RL policy. Furthermore, PSO-P can not only be employed as RL policy, but also as a model investigation tool. By applying the computed model-optimal actions directly onto the model itself (step (ii) from above), model-optimal trajectories of arbitrary length can be generated. This kind of model exploitation can help to answer several questions regarding the world model:

- What do model-optimal trajectories look like?
- Is the reward function adequate to yield the desired control behavior?
- Given a certain set of initial states, what level of return can be expected for well-performing policies?
- What is the influence of various RL parameters (e.g., event horizon T, discount factor γ) on the optimal control?
- Does the model have exploitable inaccuracies or loopholes?

Answering such questions before conduction model-based RL with explicit policy representations is of high interest to assess the learning outcome, as we will see in the course of this thesis.

The most significant advantages of PSO-P are the following. (i) Closed-form policy learners generally select a policy from a user-parameterized (potentially infinite) set of candidate policies. For example, when learning an RL policy based on tile coding [145], the user must specify partitions of the state space. The partition's characteristics directly influence how well the resulting policy can differentiate the effect of different actions. For complex RL problems, policy performances usually vary drastically depending on the chosen partitions. In contrast, PSO-P does not require a priori assumptions about problem-specific policy representations, because it directly optimizes action sequences. (ii) Closed-form RL policies operate on the state space and are generally affected by the curse of dimensionality [8]. Simply put, the number of data points required for a representative coverage of the state space grows exponentially with the state space's dimensionality. Common RL methods, such as tile coding, quickly become computationally intractable with increasing dimensionality. Moreover, for industrial RL problems, it is often costly to obtain adequate training data which prohibits data-intensive RL methods. In comparison, PSO-P is not affected by the state space dimensionality because it operates in the space of action sequences. On the one hand, PSO-P generally requires significantly more computation time to determine an action for a given system state compared to closed-form RL policies. On the other hand, experiments show that PSO-P is particularly useful for determining the optimization potential of various industrial control optimization problems and for benchmarking other RL methods.

PSO-P adapts an approach from model predictive control (MPC) [113, 15]. The general idea behind MPC is deceptively simple: given a reliable system model, one can predict the future evolution of the system and determine a control strategy that results in the desired system behavior. However, complex industry systems and plants commonly exhibit non-linear system dynamics [126, 107]. In such cases, closed-form solutions to the optimal control problem often do not exist or are computationally hard to find [46, 92]. Therefore, MPC tasks for non-linear systems are typically solved by numerical online optimization of sequences of control actions [57]. Unfortunately, the resulting optimization problems are generally non-convex [72] and no universal method for tackling non-linear MPC tasks has yet been found [47, 112]. Moreover, one might argue, based on theoretical considerations, that such a universal optimization algorithm does not exist [158].

PSO and evolutionary algorithms are established heuristics for solving non-convex optimization problems. Both have been applied in the context of RL, however, almost exclusively to optimize policies directly. Moriarty, Schultz, and Grefenstette give a comprehensive overview of the various approaches, using evolutionary algorithms to tackle RL problems [102]. Methods, which apply PSO to generate policies for specific system control problems, were studied in [43], [144], and [100].

Recently, several combinations of swarm optimization and MPC have been proposed in the literature. In [153] the non-linear and underactuated Acrobot problem was solved by adapting PSO to run in parallel on graphics hardware, yielding a real-time MPC controller. Ou, Kang, Jun, and Julius investigated the use of a single control signal and a PSO-MPC algorithm for controlling the movement of multiple magnetized cells while avoiding obstacles [105]. In [159] the authors tackled the problem of real-time application of non-linear MPC by implementing it on a field-programmable gate array that employs a PSO algorithm. By using a parallelized PSO implementation, good computational performance and satisfactory control performance were achieved. Lee and Myung significantly reduced the computational cost of collision avoidance for a class of mobile robots [86]. By applying PSO instead of traditional optimization techniques, such as sequential quadratic programming, they achieved a significant speedup during the optimization phase. They also verified the effectiveness of the proposed RHPSO-based formation control utilizing numerical simulations.

However, none of the reviewed approaches generalizes to RL, as expert-designed objective functions that already contain detailed knowledge about the optimal solution to the respective control problem, are used. In contrast, in the present chapter, the general RL problem is reformulated as an optimization problem. This representation allows searching for optimal action sequences on a system model, even if no expert knowledge about the underlying problem dynamics is available.

This chapter is based on the following publications as part of the doctoral research:

- D. Hein, A. Hentschel, T. A. Runkler, and S. Udluft. "Reinforcement learning with particle swarm optimization policy (PSO-P) in continuous state and action spaces." In: International Journal of Swarm Intelligence Research (IJSIR) 7.3 (2016), pp. 23–42. [64]

- D. Hein, A. Hentschel, T. A. Runkler, and S. Udluft. "Particle swarm optimization for model predictive control in reinforcement learning environments." In: Critical Developments and Applications of Swarm Intelligence. Ed. by Y. Shi. Hershey, PA, USA: IGI Global, 2018. Chap. 16, pp. 401–427. [63]

5.2.1 Particle Swarm Optimization Policy (PSO-P)

In this section, the problem of optimizing the action trajectory for a physical system, that is observed in discrete, equally spaced time steps $t \in \mathbb{Z}$ is considered. At each time step t, the system is described by its Markovian state $\boldsymbol{s}_t \in \mathcal{S}$, from the state space $\mathcal{S}$. The agent's action $\boldsymbol{a}_t$ is represented by a vector of I different control parameters, i.e., $\boldsymbol{a}_t \in \mathcal{A} \subset \mathbb{R}^I$. Based on the system's state and the applied action, the system transitions into the state $\boldsymbol{s}_{t+1}$ and the agent receives the reward r_t.

In the following, deterministic systems, which are described by a state transition function $g : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ with $g(\boldsymbol{s}_t, \boldsymbol{a}_t) = (\boldsymbol{s}_{t+1})$, are considered.

The goal is to find an action sequence $\boldsymbol{x} = (\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, \ldots, \boldsymbol{a}_{t+T-1})$ that maximizes the

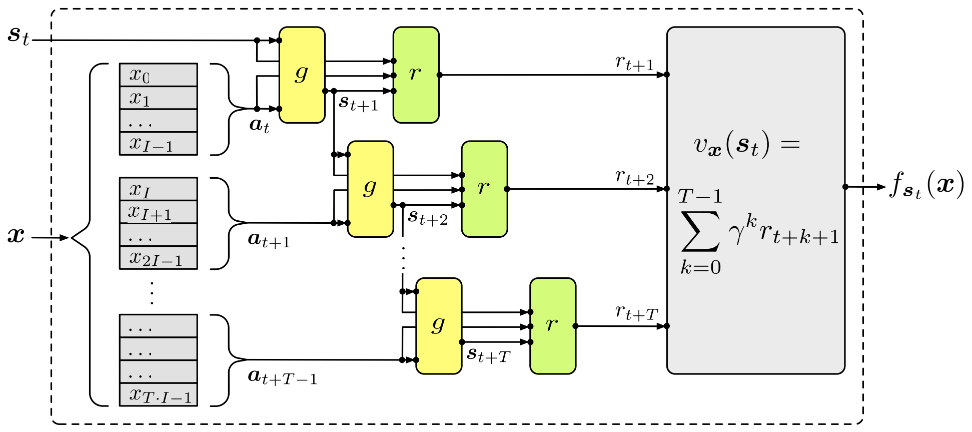


Figure 5.11: Schematic visualization of the PSO-P approach. Model-based computation of the PSO fitness function from the system's current state s_t and an action sequence x. The accumulated rewards yield the fitness value f_{s_t}, which is then used to drive the optimization.

expected return $\mathcal{R}$. The search space is bounded by $x_{\min}$ and $x_{\max}$ which are defined as:

$$x_{\min j} = a_{\min (j \bmod I)} \forall j = 0, \dots, I \cdot T - 1 \quad \text{and} \tag{5.3}$$

$$x_{\max j} = a_{\max (j \bmod I)} \forall j = 0, \dots, I \cdot T - 1, \tag{5.4}$$

where $a_{\min}$ and $a_{\max}$ are the lower and upper bounds of the control parameters respectively.

Adopting the model-based value estimation given in Eq. (2.16), the expected return $\mathcal{R}_x$ of action sequence x starting from state s_t is computed by the following value function:

$$v_x(s_t) = \sum_{k=0}^{T-1} \gamma^k r(s_{t+k}, a_{t+k}, s_{t+k+1}), \quad \text{with } s_{t+k+1} = g(s_{t+k}, a_{t+k}). \tag{5.5}$$

Solving the RL problem corresponds to finding the optimal action sequence $\hat{x}$ by maximizing

$$\hat{x} \in \arg\max_{x \in \mathcal{A}^T} f_{s_t}(x), \tag{5.6}$$

with respect to the fitness function $f_{s_t} : \mathbb{R}^{I \cdot T} \to \mathbb{R}$ with $f_{s_t}(x) = v_x(s_t)$. Fig. 5.11 illustrates the process of computing $f_{s_t}(x)$.

In this chapter, PSO is used to solve (5.6), i.e., the particles move through the search space of action sequences $\mathcal{A}^T$. Consequently, a particle's position represents a candidate action sequence $x = (a_t, a_{t+1}, \dots, a_{t+T-1})$, which is initially chosen at random.

Even though a sequence of T actions is optimized, only the first action is applied to the real-world system, and optimization of a new action sequence is performed for the subsequent system state s_{t+1}. This approach follows the widely applied control theory methods known as MPC, receding horizon control, or moving horizon method [83, 113, 15].

	MC	CPB	CPSU	IB
Particles	100			
Iterations	100			
Swarm topology	ring with four neighbors			
Parameters (w, c_1, c_1)	$(0.72981, 1.49618, 1.49618)$			
Transition samples	10 000			100 000
Time horizon T	200	100	500	100
Discount factor γ	0.985	0.97	0.994	0.97
Model usage	on demand during runtime			

Table 5.2: PSO-P experiments settings and parameters

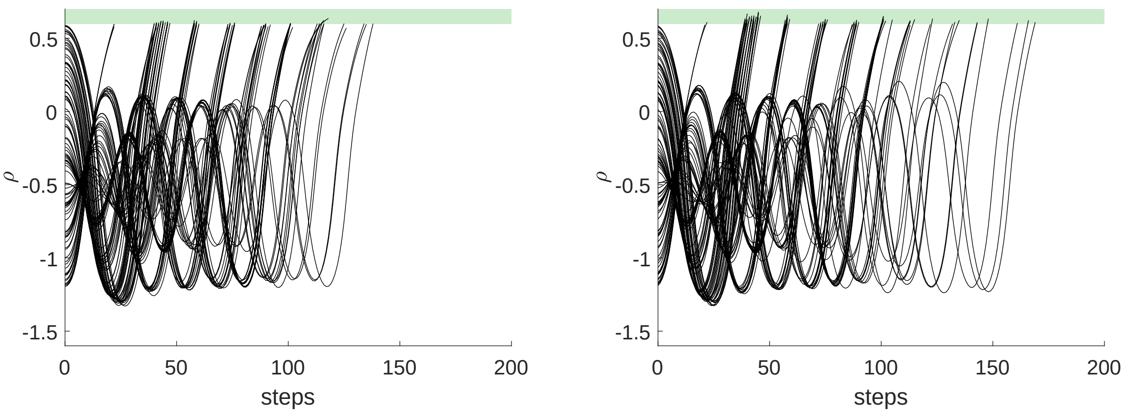
In PSO-P the state transition and reward models, referred to as $\tilde{g}$ and $\tilde{r}$ respectively, are realized by empirical approximations obtained by system identification. Thereby, models are learned by measured data from the real system dynamics g and r. Despite the fact that empirical models are likely to be inaccurate in their predictions, i.e., $\tilde{g}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \tilde{\boldsymbol{s}}_{t+1} \neq \boldsymbol{s}_{t+1} = g(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and $\tilde{r}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) = \tilde{r}_{t+1} \neq r_{t+1} = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ in (5.5), the experiments presented below verify that very stable control results can still be achieved. The reason for this advantageous behavior lies in the fact that applying only the first action of the optimized action trajectory to the system, and subsequently initializing PSO-P with the resulting real system state $\boldsymbol{s}_{t+1}$, resets the agent to the underlying true environmental conditions after each time step. Subsequently, the optimization starts with the correct initialization from scratch.

5.2.2 Experiments with PSO-P

To evaluate the performance of PSO-P, experiments on four benchmarks have been conducted. Here the two proposed settings have been evaluated: (i) applying the policy's action on the real system dynamics as runtime policy as well as (ii) applying the actions on the surrogate model for model investigation. Table 5.2 lists the relevant PSO-P and benchmark configuration parameters for the following experiments. The performance in this section is presented as penalty values computed by multiplying the achieved fitness values by -1.

Mountain Car

In the MC benchmark, the task for the PSO-P agent is to find a sequence of force actions $a_t, a_{t+1}, a_{t+2}, \ldots \in [-1, 1]$ that drive the car up the hill, which is achieved when reaching a position $\rho \geq 0.6$. At the start of each episode the car's state $\boldsymbol{s} = (\rho, \dot{\rho})$ is uniformly sampled from $[-1.2, 0.6] \times \{0\}$. Using 100 particles and 100 PSO iterations for each state to search for an optimal action trajectory, PSO-P is able to solve the MC problem given $T = 200$, $q = 0.05$, and $\gamma = 0.9851$.



(a) Model (b) Real system dynamics

Figure 5.12: State trajectories for MC produced by PSO-P. The green area represents the goal area of the MC benchmark in which the episode is declared as successful. The predicted trajectories from the world model are very similar to the trajectories produced on the real system dynamics.

Fig. 5.12 depicts the generated position trajectories for 100 training and 100 test states on the system model and the real system dynamics, respectively. The plots show that on both the model and the real dynamics PSO-P is able to successfully drive the car up the hill for every evaluated initial state in less than 200 time steps. Note that the surrogate model's predictions are very similar to the trajectories produced on the real system dynamics. Fig. 5.13 shows that the achieved performances on the model and on the real dynamics in terms of penalty values are very similar.

Cart-pole Balancing

While utilizing a world model for policy selection enabled NFQ to yield successful policies for the MC benchmark, even policy selection did not help to generate adequate policies for the CPB problem. The PSO-P experiments presented in this section, show that the reason for this failure does not stem from bad model prediction quality.

Using 100 particles and 100 PSO iterations for each state to search for an optimal action trajectory, PSO-P is able to solve the CPB problem given $T = 100$, $q = 0.05$, and $\gamma = 0.97$.

The plots depicted in Fig. 5.14 show the resulting state trajectories generated by optimal action sequences computed by PSO-P. The trajectories show that PSO-P is able to successfully drive the cart to the position goal area as well as stabilizing the pole around $\theta = 0$. This control behavior yields high reward during many time steps for the vast majority of the tested start states. However, for some of the start states PSO-P was not able to find action sequences that prevent the system from running into a failed state.

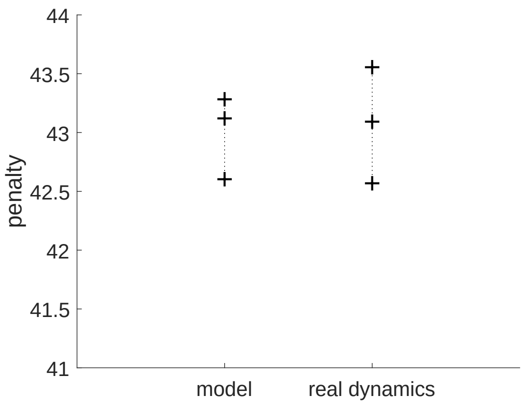


Figure 5.13: Results of PSO-P experiments for the MC benchmark. The markers represent (from top to bottom) the worst, median, and best NFQ result after ten independent training runs.

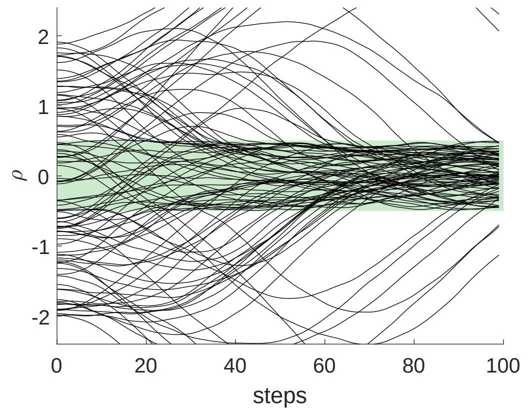
Two possible reasons are (i) some initial states are simply irretrievable using the available actions and (ii) the stochastic PSO heuristic was not able to find the optimal action trajectory given the number of particles and iterations. Another interesting observation can be made by comparing the position trajectories produced on the model with the ones produced on the real dynamics. Apparently, the model does not correctly classify all states beyond the state boundaries as failed states. This gives PSO-P the opportunity to further apply actions on the system and bring the cart back to the valid state space. However, this *rescue* strategy of course only works on the imperfect world model and fails on the real system dynamics.

The higher number of failed episodes on the real system dynamics results in a higher penalty depicted in Fig. 5.15. Despite PSO-P produced higher penalty values for the real dynamics, the overall control performance is still highly satisfactory for the CPB benchmark.

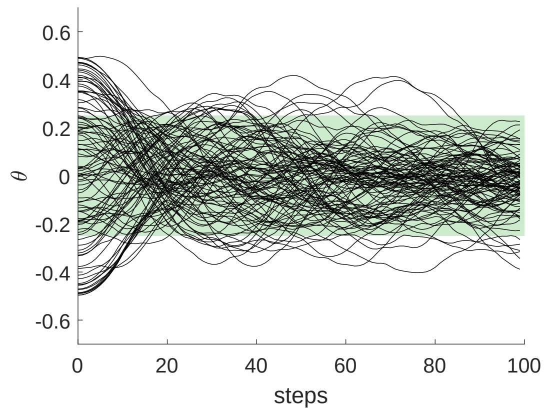
Cart-pole Swing-up

For the CPSU benchmark the goal is to find force actions $a_t, a_{t+1}, a_{t+2}, \ldots \in [-30, 30]$, that swing the pole up and subsequently prevent the pole from falling over while keeping the cart close to $\rho = 0$ for a possibly infinite period of time. Using 100 particles and 100 PSO iterations for each state to search for an optimal action trajectory, PSO-P is able to solve the CPSU problem given $T = 500$, $q = 0.05$, and $\gamma = 0.994$.

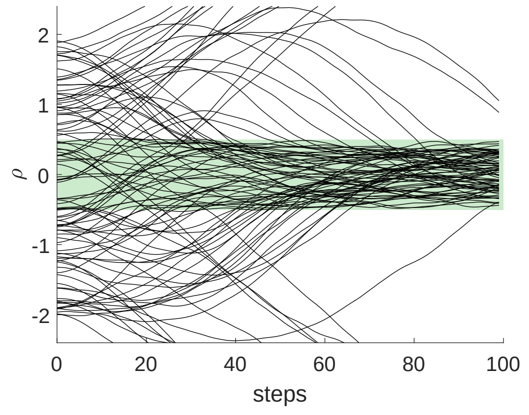
Fig. 5.16 depicts the state trajectory plots which result from applying PSO-P on CPSU. The cart position and the pendulum angle trajectories produced by applying the actions on the model (first row) show a very good control performance. The cart is quickly moved to the left or right to swing up the pole and subsequently balance it around $\theta = 0$ while the cart is kept in the dedicated goal area. For all of the 100 tested initial



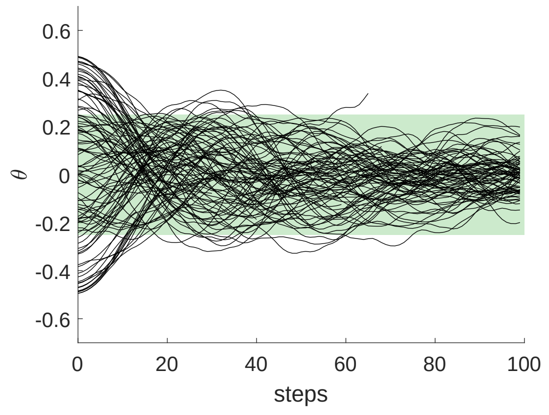
(a) Cart position on model



(b) Pendulum angle on model



(c) Cart position on real system dynamics



(d) Pendulum angle on real system dynamics

Figure 5.14: State trajectories for CPB produced by PSO-P. Figures in the upper row depict trajectories produced on the world model, trajectories depicted in the lower row are produced on different test states using the real system dynamics. The green area represents the goal area of the CPB benchmark. Leaving the restricted area of $-2.4 \leq \rho \leq 2.4$ and $-0.5 \leq \theta \leq 0.5$ results in a failed episode.

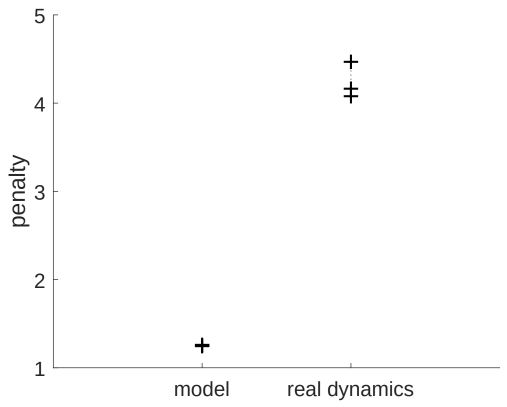


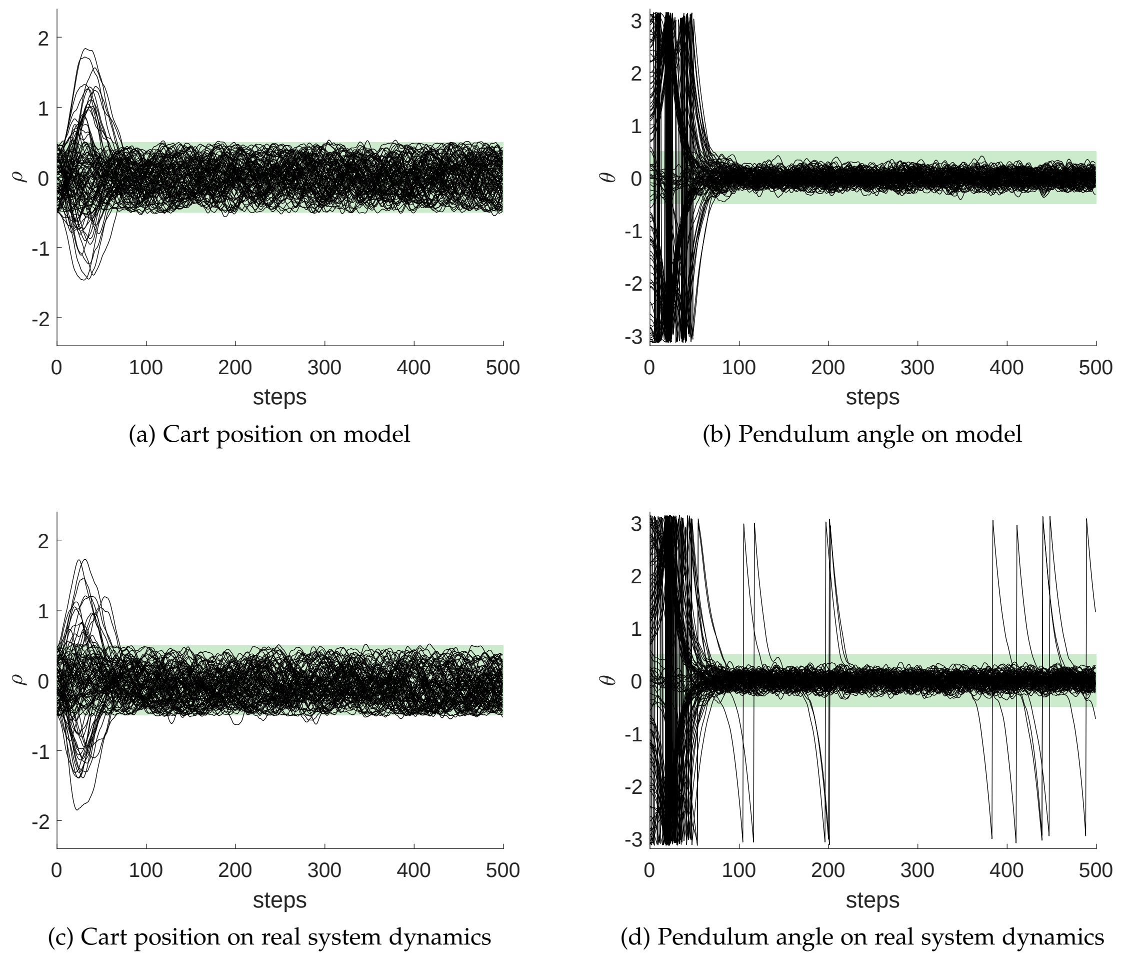
Figure 5.15: Results of PSO-P experiments for the CPB benchmark. The significant difference in penalty values between model and real dynamics arises from inaccuracies of the model prediction, which erroneously does not declare all states leaving the restricted area as fail states.

states, PSO-P never fails balancing in the goal area during the whole run time of 500 time steps. Computing the optimal next action on the model and applying it on the real system (second row of Fig. 5.16) shows a slightly different resulting control performance. Despite the vast majority of trajectories is located inside the goal area, sometimes the pendulum′ angle becomes too big and thus the pendulum can no longer be balanced. Therefore, PSO-P has to swing through the pole again and try to balance it in future time steps. One reason for that could stem from small inaccuracies of the surrogate model. The PSO gets stuck in local optimal action trajectories for which the model predicts that the system can be stabilized, but applied to the real dynamics these actions push the pendulum a bit too far from the stable area. Nevertheless, PSO-P performs significantly better compared to NFQ with or without model-based policy selection.

The differences between model and real system dynamics evaluation which we observed in the produced trajectories, results in a slightly worse average penalty value for the real dynamics (Fig. 5.17). However, penalty values between 36 and 37 are excellent results for CPSU.

Industrial Benchmark

For the IB, the task is to find a sequence of actions $\boldsymbol{x} = (\Delta v_t, \Delta g_t, \Delta h_t, \Delta v_{t+1}, \Delta g_{t+1}, \Delta h_{t+1}, \ldots, \Delta v_{t+T-1}, \Delta g_{t+T-1}, \Delta h_{t+T-1})$ which changes the control variables in a way that return $\mathcal{R}$ is as high as possible for a given time horizon T. Using 100 particles and 100 PSO iterations for each state to search for an optimal action trajectory, PSO-P is able to solve the IB problem given $T = 100$, $q = 0.05$, and $\gamma = 0.97$. Moreover, the experimental results show that the policy performance is robust even against highly stochastic benchmark dynamics as present in the IB.



(a) Cart position on model

(b) Pendulum angle on model

(c) Cart position on real system dynamics

(d) Pendulum angle on real system dynamics

Figure 5.16: State trajectories for CPSU produced by PSO-P. Figures in the upper row depict trajectories produced on the world model, trajectories depicted in the lower row are produced on different test states using the real system dynamics. The green area represents the goal area of the CPSU benchmark.

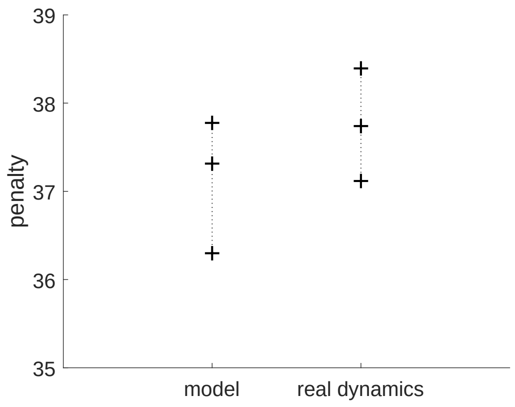


Figure 5.17: Results of PSO-P experiments for the CPSU benchmark.

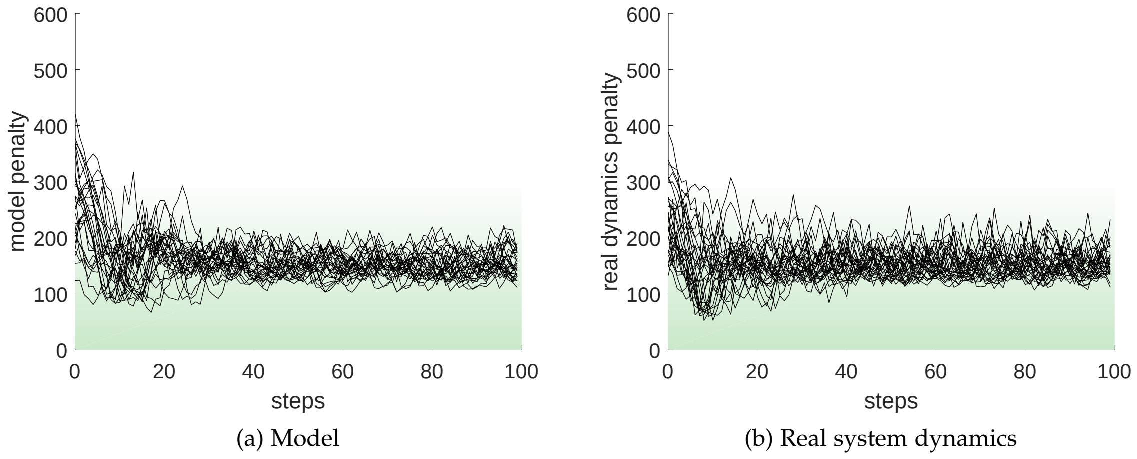


Figure 5.18: Reward trajectories for IB produced by PSO-P

The penalty trajectory plots from Fig. 5.18 show that PSO-P is able to lower the penalty in less than 40 time steps from any given initial state. Subsequently, the penalty remains at a low level, which indicates that PSO-P finds control strategies which are aware of the latent stochastic and delayed effects of the IB. Note that the employed RNN surrogate model is only a deterministic approximation of the highly non-deterministic real system dynamics. Therefore, it seems to be even more impressive that the PSO-P planning does not only result in optimal actions for the model itself (Fig. 5.18a), but also yields a very good control strategy for the real system dynamics (Fig. 5.18b).

Comparing the average penalty between evaluation on the model and on a different start state set on the real system dynamics (Fig. 5.19) shows that the model is prone to underestimate the penalty which is accumulated over time. One of the reasons for this effect is certainly the fact that the search for optimal actions is conducted

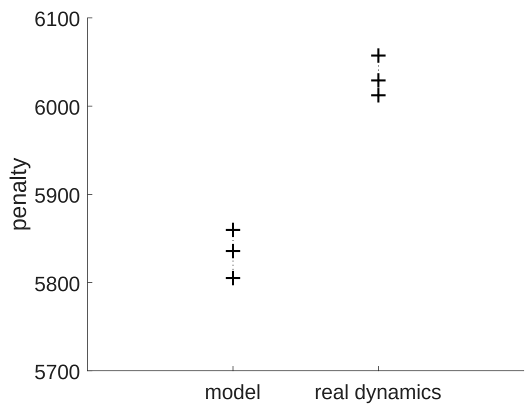


Figure 5.19: Results of PSO-P experiments for the IB

on a deterministic model, whereas the real dynamics are highly non-deterministic. Nevertheless, the produced results on the real system dynamics are still of high quality compared to results with NFQ (Section 5.1) and other methods like recurrent control NN [65].

5.2.3 Discussion

The presented results show that PSO-P is capable of providing RL policies with high-quality state-to-action mappings. In essence, PSO-P performs an online optimization of an action sequence, each time an action for a given system state is requested. Compared to learning a functional policy representation, whose actions are recalled later on, PSO-P has the following advantages:

- PSO-P does not require a priori assumptions about adequate policy representations. Hence, no bias with respect to a specific policy behavior is introduced.
- PSO-P is effective for high-dimensional state spaces, as the optimization runs in the space of action sequences, which are independent of the state space's dimensionality.
- The reward function can be changed after each system transition, as the optimization process starts from scratch for each new system state.

The drawback compared to closed-form policies is the significantly higher computational load for computing actions using PSO-P. Implementing parallelized PSO on hardware or using cloud-based computational resources will enable MPC policy solutions like PSO-P to become feasible for more and more applications. Furthermore, in many real-world industrial applications high-level system control is implemented by changing control parameters in terms of seconds or minutes which, in many cases, is a sufficient amount of time to compute the next action using PSO-P.

PSO-P is a complementary approach for solving RL problems because it searches in the action space, while established RL methods generally work in the value function space or the policy space [147]. Therefore, a promising application is to use PSO-P for benchmarking other RL methods. Moreover, PSO-P can be used for reward function design or tuning, i.e., for the process of designing a reward function that induces a desired policy behavior.

5.3 Gradient-free Neural Network Policy Optimization

After successfully applying model-based policy selection for NFQ and model exploitation to yield optimal trajectories using PSO-P, in this section we investigate whether the same models can be used to train explicit RL policies. For now the condition of learning interpretable policies is relaxed, since at first it is verified whether a model-based batch RL training can be successfully applied on the posed benchmark problems at all.

For this empirical study, the policy is represented by an NN whose weights are optimized by PSO. Fig. 5.20 depicts the framework of PSONN in comparison to that of PSO-P. The inputs of the NN policy are the state features and the output is the respective action. Using PSO to optimize the weights of NNs has been applied in several works. Zhang, H. Shao, and Li [162] evolved standard MLPs, while Juang [73] used a hybrid of genetic algorithm and PSO for RNN design. X. Chen and Li [18] proposed the use of stochastic PSO for NN training and give several reasons for using PSO instead of back propagation:

- The PSO algorithm can solve optimization problems with non-continuous solution domains;
- There is no constraint for specific transfer functions, so that more transfer functions, even non-differentiable ones, can be selected to fulfill different requirements;
- Comparing with backpropagation, the exploration ability embedded in PSO enables NN training to be more efficient to escape from local minima.

For these reasons, researchers started using PSO-trained NNs in RL [21] and real world control tasks [160].

5.3.1 Particle Swarm Optimization Neural Network (PSONN) Policy

As introduced in Section 2.1.5, in model-based RL we are searching for the optimal policy $\pi[\hat{x}]$ parameterized by an optimal parameter vector $\hat{x}$:

$$\hat{x} \in \arg\max_{x \in \mathcal{X}} \overline{\mathcal{R}}_{\pi[x]}, \quad \text{with} \quad \overline{\mathcal{R}}_{\pi[x]} = \frac{1}{|\mathcal{S}|} \sum_{s_t \in \mathcal{S}} v_{\pi[x]}(s_t), \tag{5.7}$$

where $\overline{\mathcal{R}}_{\pi[x]}$ represents the PSO's fitness function, which is defined by maximizing the expected RL return $\mathcal{R}$ for all states in $\mathcal{S}$.

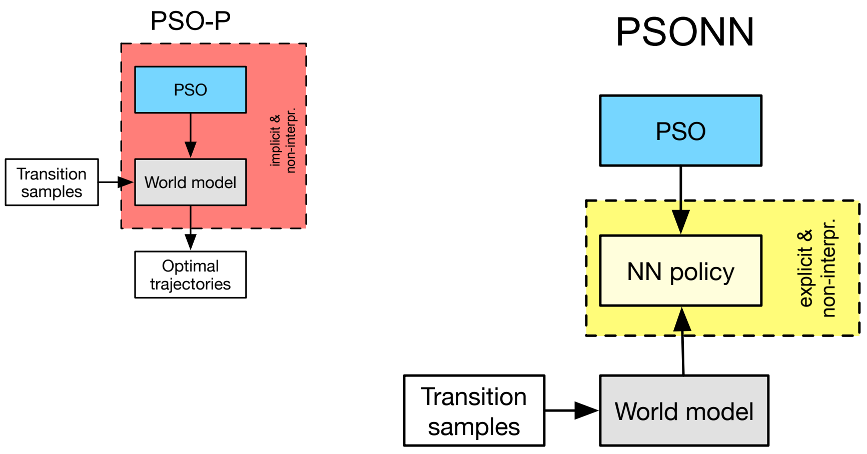


Figure 5.20: The PSONN framework compared to PSO-P

Herein, the networks' topologies, as well as the neurons' transfer functions, are fixed and not part of the swarm optimization. Parameter vector $\boldsymbol{x}$ defines the weights of the network graph edges which interconnect each neuron of one layer with all of the neurons of the next layer:

$$\begin{aligned} \boldsymbol{x} = (&w_{(1,1),(2,1)}, w_{(1,1),(2,2)}, \ldots, w_{(1,N_1),(2,N_2)}, b_{(2,1)}, \ldots, b_{(2,N_2)}, \\ &w_{(2,1),(3,1)}, w_{(2,1),(3,2)}, \ldots, w_{(2,N_2),(3,N_3)}, b_{(3,1)}, \ldots, b_{(3,N_3)}, \ldots, \\ &w_{(M-1,1),(M,1)}, w_{(M-1,1),(M,2)}, \ldots, w_{(M-1,N_{M-1}),(M,N_M)}, b_{(M,1)}, \ldots, b_{(M,N_M)}), \end{aligned} \tag{5.8}$$

where M is the number of layers, N_i is the number of neurons of the i-th layer, $w_{(i-1,k),(i,l)}$ is the weight from neuron k of the $(i-1)$-th layer to neuron l of the i-th layer, and $b_{(i,k)}$ is the bias applied to neuron k of the i-th layer (Fig. 3.5 depicts a graphical visualization of the described MLP). Each particle of the swarm represents one solution within the parameter space. The initial positions of the parameters are sampled from a uniform distribution bounded in each dimension within $[-10, 10]$.

The PSONN policy $\pi[\boldsymbol{x}](\boldsymbol{s}) = \boldsymbol{a}$ receives a state $\boldsymbol{s}$ as input, transforms the state features' values according to the pre-defined activation functions and weights in $\boldsymbol{x}$, and passes the neurons' outputs further to the next layer in the network. The signals produced by the final layer are interpreted as action $\boldsymbol{a}$, which is subsequently applied to the system. These actions are continuous values per default. Note that this explicit policy representation does not require to be evaluated several times to find optimal actions, like with NFQ. In addition to that, during policy runtime, no expensive trajectory search is needed, as with PSO-P. With PSONN, the computational effort has only to be applied during policy training, which makes it well suited for real-time applications.

	MC	CPB	CPSU	IB
NN policy	2-20-1	4-20-1		180-20-1 (3x)
Particles	1 000			
Iterations	1 000			10 000
Swarm topology	ring with four neighbors			
Parameters (w, c_1, c_1)	$(0.72981, 1.49618, 1.49618)$			
Transition samples	10 000			100 000
Time horizon T	200	100	500	100
Discount factor γ	0.985	0.97	0.994	0.97
Model usage	for policy training			

Table 5.3: PSONN experiments settings and parameters

5.3.2 Experiments with PSONN

For the experiments described below, the very same NN surrogate models have been used as for the PSO-P experiments discussed in Section 5.2. In contrast to PSO-P, with PSONN all information about the weights of an optimal policy has to be extracted from the world models during training without evaluation on the real system dynamics. Hence, the knowledge about how optimal control is achieved is contained solely in the policy after training has finished, which means that the model is no longer needed during the evaluation phase on the real system dynamics afterward.

Table 5.3 lists the relevant PSONN and benchmark configuration parameters for the following experiments. The performance in this section is presented as penalty values computed by multiplying the achieved fitness values by -1.

Mountain Car

For the MC experiments a PSONN policy of the form 2-20-1 has been predefined, i.e., two input neurons for the state features ρ and $\dot{\rho}$, 20 non-linearly activated neurons on the hidden layer, and one neuron as output predicting the optimal action a for the current state. While input and output layers apply linear activation functions, the neurons on the hidden layer are activated by a hyperbolic tangent function, i.e., $f(x) = \tanh(x)$. For the MC benchmark, a PSO swarm of size 1 000 particles was allowed to search for 1 000 iterations. The experiments have been repeated ten times. Each training employed 100 start states $s = (\rho, \dot{\rho})$ that were uniformly sampled from $[-1.2, 0.6] \times \{0\}$. As for previous MC experiments with NFQ and PSO-P, time horizon $T = 200$ and $q = 0.05$ are used which yields a discount factor $\gamma = 0.9851$.

Fig. 5.21 shows trajectories for the position variable ρ generated by applying the best resulting policy for the surrogate models. It is easy to see that from each of the 100 training states a successful trajectory to the goal area in less than 150 steps on the model (Fig. 5.21a) could be generated. Evaluating the very same policy on 100 different test

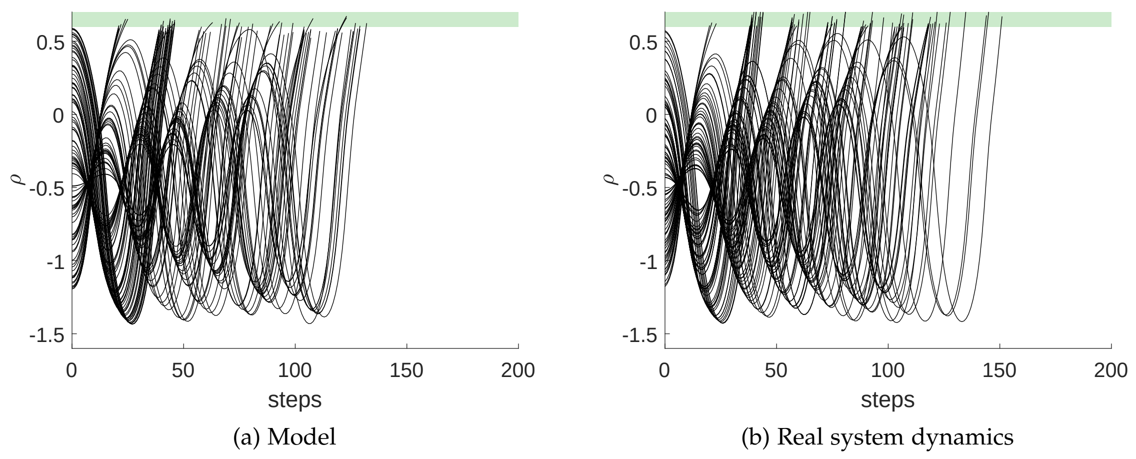


Figure 5.21: State trajectories for MC produced by the best PSONN policy. The green area represents the goal area of the MC benchmark in which the episode is declared as successful.

states using the real system dynamics generates quite similar trajectories (Fig. 5.21b).

Comparing the average model penalty with the average real dynamics penalty (Fig. 5.22) verifies the observation from the trajectory comparison. Both penalty value ranges represent successful policy trainings. The observable drift of slightly higher penalties on the real system dynamics most likely arises from differences in the randomly drawn 100 test states.

Cart-pole Balancing

For the CPB experiments a PSONN policy of the form 4-20-1 has been predefined, i.e., four input neurons for the state features θ, $\dot{\theta}$, ρ, and $\dot{\rho}$, 20 non-linearly activated neurons on the hidden layer, and one neuron as output predicting the optimal action a with respect to the current state. A PSO swarm of size 1 000 particles was allowed to search for 1 000 iterations. The experiments have been repeated ten times. Each training employed 100 start states $\boldsymbol{s} = (\theta, \dot{\theta}, \rho, \dot{\rho})$ that were uniformly sampled from $[-0.5, 0.5] \times \{0\} \times [-2.0, 2.0] \times \{0\}$.

The state trajectory plots depicted in Fig. 5.23 show clearly that the best resulting policy performs well on the CPB task. Both the cart position as well as the pendulum angle are stabilized in the goal area in less than 100 time steps for the vast majority of the initial start states. Note that for the policy training on the surrogate model (upper row) all of the start states are successfully stabilized while executing the same policy on different test states on the real system dynamics occasionally fails (lower row). The reason for that lies in the inaccuracy of the world model used for training. As visible in Fig. 5.23a, for some initial states the cart leaves the restricted area of $-2.4 \leq \rho \leq 2.4$

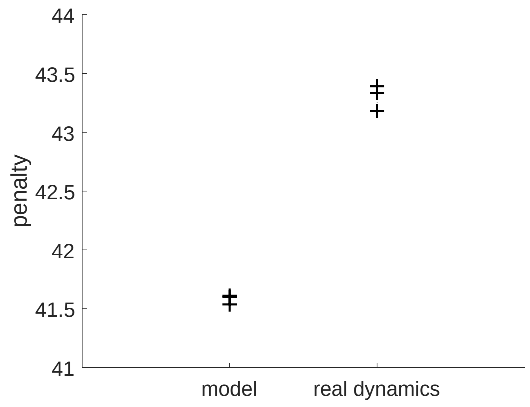


Figure 5.22: Results of PSONN experiments for the MC benchmark. The markers represent (from top to bottom) the worst, median, and best PSONN result after ten independent training runs. Both results for average model penalty as well as average real dynamics penalty are very good.

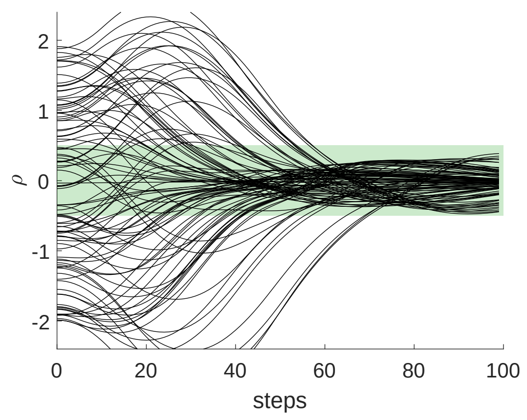
which actually should result in a failed episode. As visible in the plot, the world model does not always correctly classify states outside the boundary as failed states, which opens an opportunity for the policy learner to bring back the cart into the desired position area and continue the process of cart and pendulum stabilization. However, during evaluation on the real system dynamics, it is no longer possible to continue applying actions on the cart after transition into a fail state.

The fact that it is possible to *rescue* some states by exploiting inaccuracies of the world model, results in a significantly lower average penalty rating for the training phase on the one side and the evaluation phase on the other side. Fig. 5.24 shows that the best PSONN policies can generate policies with penalty values as low as 1.8 while evaluating the very same policies on the real system dynamics results in penalty values of 2.8.

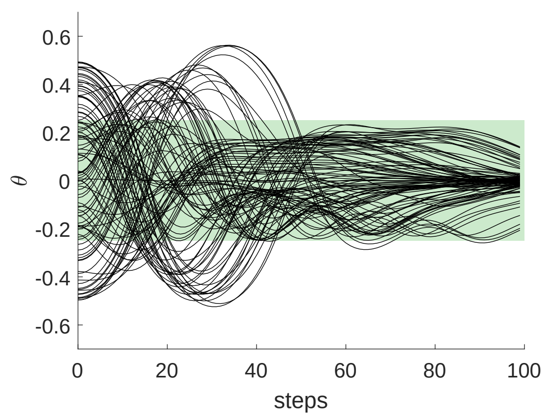
Cart-pole Swing-up

For the CPSU experiments a PSONN policy of the form 4-20-1 has been predefined, i.e., four input neurons for the state features θ, $\dot{\theta}$, ρ, and $\dot{\rho}$, 20 non-linearly activated neurons on the hidden layer, and one neuron as output predicting the optimal action a with respect to the current state. A PSO swarm with 1 000 particles was allowed to search for 10 000 iterations. The experiments have been repeated ten times. Each training employed 100 start states $\boldsymbol{s} = (\theta, \dot{\theta}, \rho, \dot{\rho})$ that were uniformly sampled from $[-\pi, \pi] \times \{0\} \times [-0.5, 0.5] \times \{0\}$.

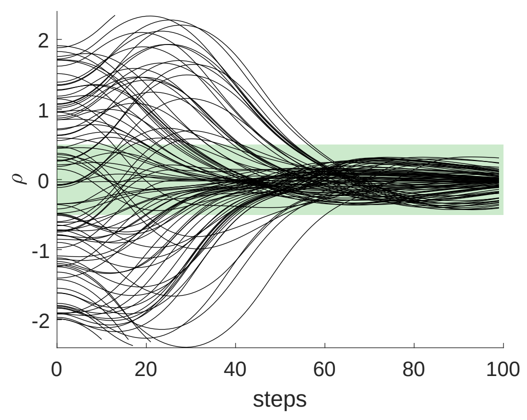
The trajectory plots generated by applying the best resulting policy are depicted in Fig. 5.25. Observe that the performance on the model (upper row) is very high. All of the 100 training states can be successfully swung-up in less than 100 time steps and subsequently kept stable in the respective goal areas. Using the same policy for trajectory



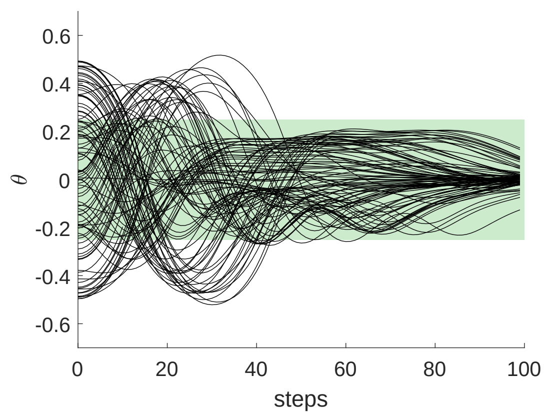
(a) Cart position on model



(b) Pendulum angle on model



(c) Cart position on real system dynamics



(d) Pendulum angle on real system dynamics

Figure 5.23: State trajectories for CPB produced by the best PSONN policy. Figures in the upper row depict trajectories produced on the world model, trajectories depicted in the lower row are produced on different test states using the real system dynamics. The green area represents the goal area of the CPB benchmark. Leaving the restricted area of $-2.4 \leq \rho \leq 2.4$ and $-0.5 \leq \theta \leq 0.5$ results in a failed episode.

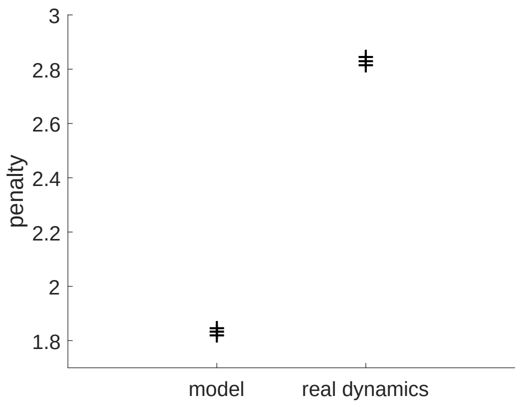


Figure 5.24: Results of PSONN experiments for the CPB benchmark. The significant difference in penalty values between model and real dynamics arises from inaccuracies of the model prediction, which erroneously does not declare all states which are leaving the restricted area as fail states.

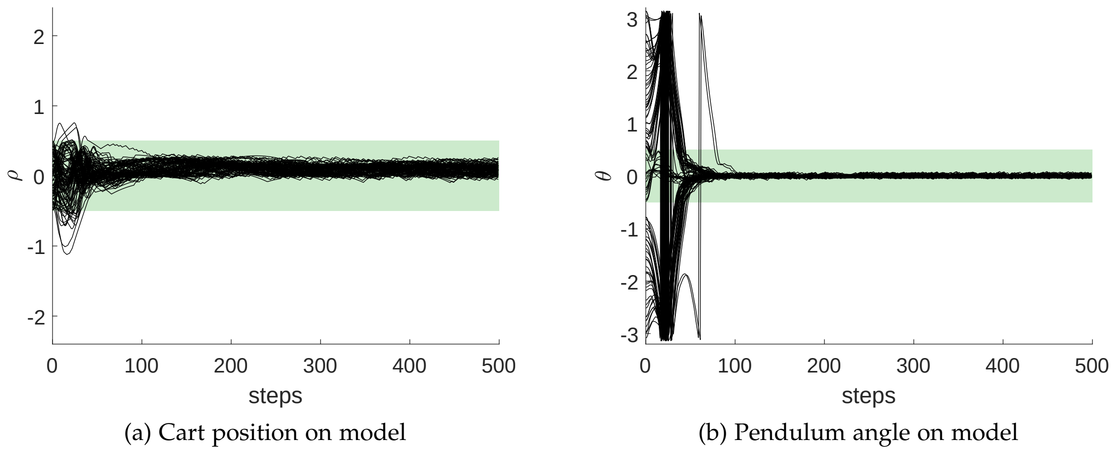
generation on a different set of test states and the real system dynamics produces similar trajectories (lower row). However, it can be observed that for some of the start states the process of swinging up takes slightly longer or needs more than one attempt.

Fig. 5.26 compares the penalties yielded by the best resulting policies of the ten independent training runs. Despite the high variance of the average penalty values, all of the policies yield adequate system controllers for CPSU regarding their capability of swinging up the pendulum and subsequently stabilize the system in its goal area. The differences in penalty values arise from the fact that some policies achieve the benchmark goal sooner than other policies.

Industrial Benchmark

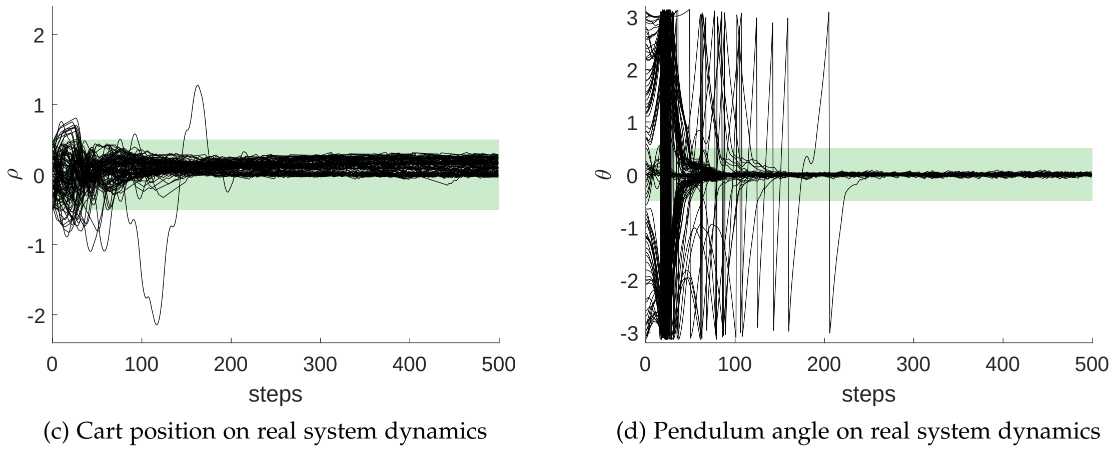
For the IB experiments a PSONN policy consisting of three independent NNs, one for each action dimension, of the form 180-20-1 has been predefined, i.e., 180 input neurons for the historical observation features (v, g, h, p, c, f) of the past 30 time steps, 20 non-linearly activated neurons on the hidden layer, and one neuron as output predicting the optimal action a with respect to the current state. For the IB, a PSO swarm of size 1 000 particles was allowed to search for 10 000 iterations. The experiments have been repeated ten times. Each training employed 100 start states randomly drawn from the initial transition sample data set $\mathcal{D}_{\text{IB}}$.

Fig. 5.27 depicts the resulting reward trajectories of the best trained PSONN policy. It can be seen that this policy successfully reduces the penalty on the surrogate model as well as during the evaluation phase on the real system dynamics. For none of the states of the randomly selected test set did the system fall into a bad state with high penalties of more than 500 as observed in [65] for other RL methods.



(a) Cart position on model

(b) Pendulum angle on model



(c) Cart position on real system dynamics

(d) Pendulum angle on real system dynamics

Figure 5.25: State trajectories for CPSU produced by the best PSONN policy. Figures in the upper row depict trajectories produced on the world model, trajectories depicted in the lower row are produced on different test states using the real system dynamics. The green area represents the goal area of the CPSU benchmark. Note that it takes slightly longer to swing up the pendulum for the same policy applied to the real system dynamics.

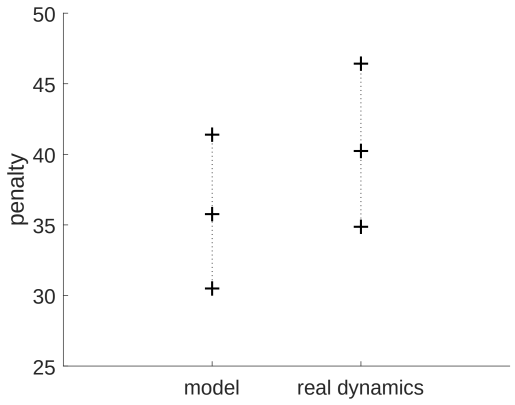


Figure 5.26: Results of PSONN experiments for the CPSU benchmark. Penalty values below 50 represent adequate control performance for the benchmark.

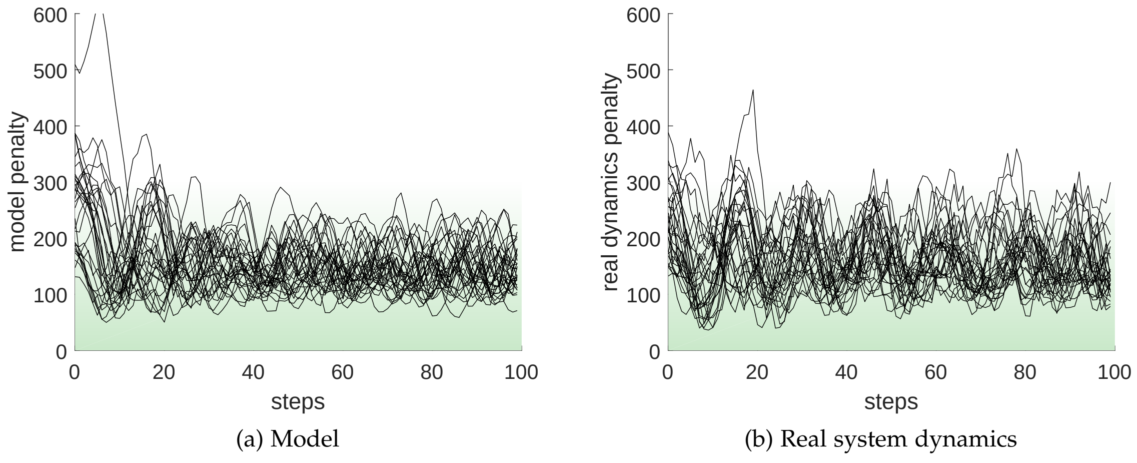


Figure 5.27: Reward trajectories for IB produced by the best PSONN policy. The green area represents the goal area of the IB. The gradient of this area represents the goal to lower the penalty as much as possible.

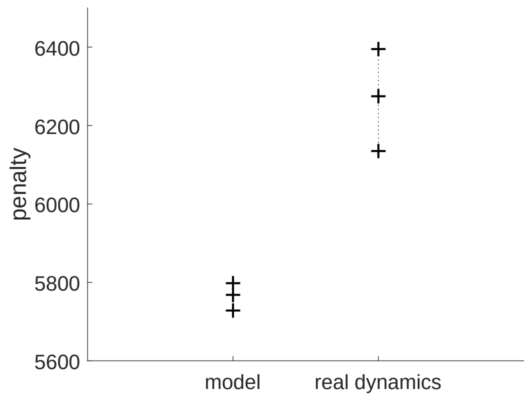


Figure 5.28: Results of PSONN experiments for the IB

Comparing the average penalties between model and real dynamics (Fig. 5.28) indicates that the surrogate model slightly overestimates the policies' performances. One reason for this could be the fact that the deterministic RNN surrogate model has only partial knowledge about the complex dynamics of the true underlying system due to the fact that it has been trained with limited data. However, the achieved penalty values between 6100 and 6400 on the given test states represent excellent control behaviors.

5.3.3 Discussion

In this section, PSONN has been successfully applied as RL policy learning approach. The existing idea of learning optimal NN weights by PSO has been combined with the model-based batch RL technique introduced in Section 2.1.5. By combining these two methods, the following is achieved:

- Using RL for learning optimal policies from previously generated transition samples;
- Validating the feasibility of performing population-based policy search on surrogate models for all four benchmarks;
- Yielding an explicit policy representation by using an NN which receives the current state and generates the action without the need to maximize over all possible actions or performing any trajectory search, as necessary for NFQ or PSO-P, respectively;
- Showing that well-performing explicit policies do exist, given a sufficient number of degrees of freedom, as multi-layer NNs with non-linear activation functions are expected to have.

However, PSONN policies do not aim at being interpretable at all. Three-layer networks with 20 hidden neurons with non-linear activation functions consist of hundreds of terms written down as closed-form algebraic expressions. Policies of such form are unlikely to be denoted as interpretable by domain experts.

Therefore, in the following chapters, three novel interpretable model-based batch RL methods are proposed and evaluated. These approaches are based on the same idea of applying population-based policy search using surrogate models built from already existing transition data. We will see that despite the resulting policies are of significantly more compact forms compared to PSONN solutions, the performance during the training phase is comparable, and the generalization quality on different test states evaluated on the real system dynamics is often even higher.

Nevertheless, for this study and during application for real-world settings, it has proven to be very useful to learn PSONN policies prior to learning interpretable solutions. This way it can be verified that the surrogate models are of adequate prediction quality, well behaving (possibly non-interpretable) policies exist at all, and that population-based RL can find such policies.

CHAPTER 6

Rule-based Policies by Particle Swarm Optimization

In this chapter, the first of three novel interpretable reinforcement learning (RL) methods is introduced and evaluated. *Fuzzy particle swarm reinforcement learning* (FPSRL) utilizes a similar model-based RL approach as particle swarm optimization neural network (PSONN) from Section 5.3, but instead of optimizing the weights of a neural network (NN) policy it utilizes particle swarm optimization (PSO) to search for optimal rule-based policy parameters. Fig. 6.1 highlights the differences of FPSRL in comparison with PSONN.

Self-organizing fuzzy rule controllers are considered to be efficient and interpretable [16] system controllers in control theory for decades [108, 129, 136]. Several approaches for autonomous training of fuzzy controllers have proven to produce remarkable results on a wide range of problems. Jang introduced a method called ANFIS, which is a fuzzy inference system implemented using an adaptive network framework [71]. This approach has been frequently utilized to develop fuzzy controllers. For example, ANFIS has been successfully applied to the cart-pole problem [123, 58, 78]. During the ANFIS training process, training data must represent the desired controller behavior, which makes this process a supervised machine learning (ML) approach. However, the optimal controller trajectories are unknown in many industry applications.

Feng applied PSO to generate fuzzy systems to balance the cart-pole system and approximate a non-linear function [44, 45]. Debnath, Shill, and Murase optimized Gaussian membership function parameters for non-linear problems and showed that parameter tuning is much easier with PSO than with conventional methods, like utilizing imprecise heuristic knowledge of experienced control engineers, because knowledge about the derivative and complex mathematical equations is not required [25]. Kothandaraman and Ponnusamy applied PSO to tune adaptive neuro-fuzzy controllers for a vehicle suspension system [80]. Experience with learning fuzzy parameters shows that the error landscape in which the optimal parameter set has to be found, very often is uneven

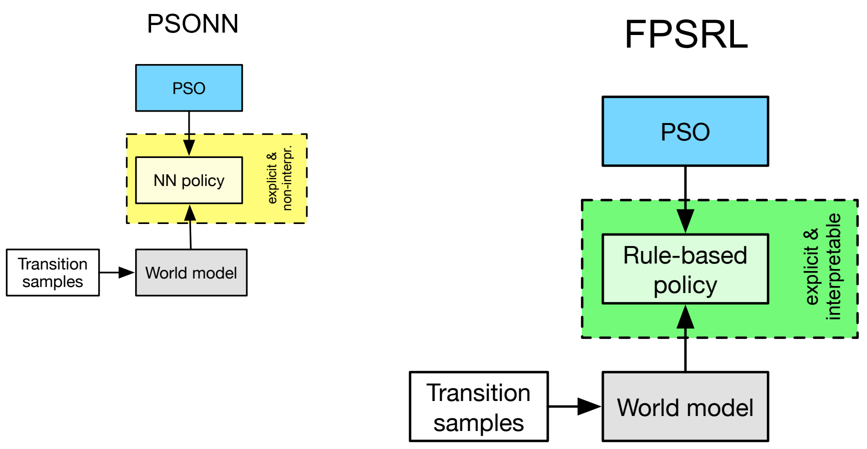


Figure 6.1: The FPSRL framework compared to PSONN

and ragged. Hence, using gradient-based optimization heuristics for such problems as used with ANFIS seems to be generally unfavorable. However, similar to ANFIS, the PSO fitness functions in all these contributions have been dedicated expert formulas or mean-square error functions that depend on correctly classified samples.

In the proposed FPSRL approach, different fuzzy policy parameterizations are evaluated by testing the resulting policy on a world model using a Monte Carlo method [147]. The combined return value of a number of action sequences is the fitness value that is maximized iteratively by the optimizer.

This chapter is based on the following publication as part of the doctoral research:

- D. Hein, A. Hentschel, T. A. Runkler, and S. Udluft. "Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies." In: Engineering Applications of Artificial Intelligence 65 (2017), pp. 87–98. [62]

6.1 Rule-based Controllers

Fuzzy set theory was introduced by Zadeh in 1965 [161]. Based on this theory, Mamdani and Assilian introduced a so-called fuzzy controller specified by a set of linguistic if-then rules whose membership functions can be activated independently and produce a combined output computed by a suitable defuzzification function [93].

In a D-inputs-single-output system with C rules, a fuzzy rule $R^{(i)}$ can be expressed as follows:

$$R^{(i)} : \text{ IF } \mathbf{s} \text{ is } m^{(i)} \text{ THEN } o^{(i)}, \quad \text{with } i \in \{1, \dots, C\}, \tag{6.1}$$

where $\mathbf{s} \in \mathbb{R}^D$ denotes the input vector (the environment state in our setting), $m^{(i)}$ is the membership of a fuzzy set of the input vector in the premise part, and $o^{(i)}$ is a real number in the consequent part.

For the FPSRL experiments in this thesis, Gaussian membership functions [155] have been applied. This very popular type of membership function yields smooth outputs, is local but never produces zero activation, and forms a multivariate Gaussian function by applying the product over all membership dimensions. The non-zero property is important because it guarantees that for every point in the input space every rule is at least somewhat activated which ensures to avoid a division by zero later in (6.4). The membership function of each rule is defined as follows:

$$m^{(i)}(\boldsymbol{s}) = \mathrm{m}[\boldsymbol{c}^{(i)}, \boldsymbol{\sigma}^{(i)}](\boldsymbol{s}) = \prod_{j=1}^{D} \exp\left\{ -\frac{(c_j^{(i)} - s_j)^2}{2{\sigma_j^{(i)}}^2} \right\}, \tag{6.2}$$

where $m^{(i)}$ is the i-th parameterized Gaussian $\mathrm{m}[\boldsymbol{c}, \boldsymbol{\sigma}]$ with its center at $\boldsymbol{c}^{(i)}$ and width $\boldsymbol{\sigma}^{(i)}$.

The parameter vector $\boldsymbol{x} \in \mathcal{X}$, where $\mathcal{X}$ is the set of valid Gaussian fuzzy parameterizations, is of size $d = (2D+1) \cdot C + 1$ and contains

$$\begin{aligned} \boldsymbol{x} = (&c_1^{(1)}, c_2^{(1)}, \ldots, c_D^{(1)}, \sigma_1^{(1)}, \sigma_2^{(1)}, \ldots, \sigma_D^{(1)}, o^{(1)}, \\ &c_1^{(2)}, c_2^{(2)}, \ldots, c_D^{(2)}, \sigma_1^{(2)}, \sigma_2^{(2)}, \ldots, \sigma_D^{(2)}, o^{(2)}, \ldots, \\ &c_1^{(C)}, c_2^{(C)}, \ldots, c_D^{(C)}, \sigma_1^{(C)}, \sigma_2^{(C)}, \ldots, \sigma_D^{(C)}, o^{(C)}, \alpha). \end{aligned} \tag{6.3}$$

The output is determined using the following formula:

$$\pi[\boldsymbol{x}](\boldsymbol{s}) = \tanh\left(\alpha \cdot \frac{\sum_{i=1}^{C} m^{(i)}(\boldsymbol{s}) \cdot o^{(i)}}{\sum_{i=1}^{C} m^{(i)}(\boldsymbol{s})} \right), \tag{6.4}$$

where the hyperbolic tangent limits the output to between -1 and 1, and parameter α can be used to change the slope of the function. The normalization applied in (6.4) in combination with the Gaussian membership function ensures that even a relatively small number of membership functions can produce policy outputs for all possible inputs.

6.2 Fuzzy Particle Swarm Reinforcement Learning (FPSRL)

In the first FPSRL step, an assumption about the number of rules per policy is required. For the experiments presented in Section 6.3, the number of rules has been iteratively increased and the performances were compared to those of the policies with fewer rules. This process was repeated until performance with respect to the world models was satisfactory. An intuitive representation of the maximal achievable policy performance given a specific discount factor for a certain model can be computed by adopting a trajectory optimization technique, such as particle swarm optimization policy (PSO-P) from Section 5.2, before FPSRL training.

A schematic representation of the proposed FPSRL framework is given in Fig. 6.2. PSO (Section 2.2) evaluates parameter vectors $\boldsymbol{x}$ of a predefined fuzzy rule representation $\pi[\boldsymbol{x}]$. For each given set of parameters $\boldsymbol{x_a} = (\boldsymbol{x}_{a_1}, \boldsymbol{x}_{a_2}, \ldots)$, a model-based RL evaluation is performed by first computing action vector $\boldsymbol{a}_t = (a_{t,1}, a_{t,2}, \ldots) = (\pi[\boldsymbol{x}_{a_1}](\boldsymbol{s}_t), \pi[\boldsymbol{x}_{a_2}](\boldsymbol{s}_t), \ldots)$

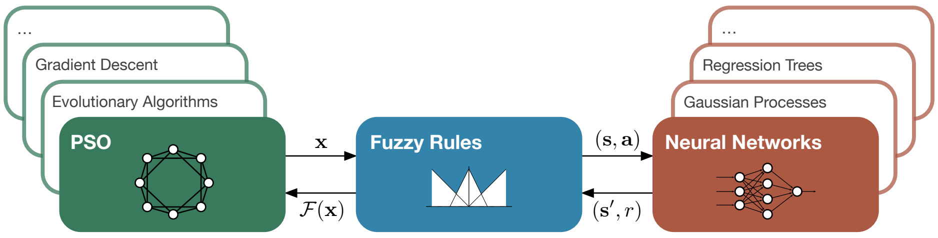


Figure 6.2: Schematic visualization of the proposed FPSRL approach. Alternative techniques that could replace PSO and NNs are presented in the background.

for state $\boldsymbol{s}_t$ (Eq. (6.4)). Then, the approximative performance of this tuple is computed by predicting both the resulting state $\boldsymbol{s}_{t+1}$ and the transition's reward r using surrogate models. Repeating this procedure for state $\boldsymbol{s}_{t+1}$ and its successor states generates an approximative trajectory through the state space. Accumulating the rewards using Eq. (2.20), the value $v_{\pi[\boldsymbol{x}]}(\boldsymbol{s}_t)$ is computed for each state, which is eventually used to compute the average return $\overline{\mathcal{R}}_{\pi[\boldsymbol{x}]}$, which drives the swarm to high performance policy parameterizations (Eq. (2.21)).

Feature Selection

Industrial applications usually have dozens or even hundreds of possible state features, collected by different plant sensors. However, practical experience shows that very often only a small subset of these sensors is required to create a policy that performs well. Unfortunately, determining the most important features is an expensive and ambiguous process, but it is nonetheless essential if we are to apply promising techniques such as FPSRL to industrial applications. Therefore, a two-step approach is proposed that yields lists of features for each action, which are ordered by their estimated relevance to an optimal policy.

First, an optimal trajectory is generated by applying PSO-P from Section 5.2 to the system model $\tilde{g}$. Here, PSO-P uses the model $\tilde{g}$ to determine action sequences $(\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, \dots, \boldsymbol{a}_T)$ starting from the state $\boldsymbol{s}_t$. The first action $\boldsymbol{a}_t$ in the optimal sequence is stored in the tuple $(\boldsymbol{s}_t, \boldsymbol{a}_t)$, and this process is repeated for all possible states in the data set $\mathcal{D}$. Note that no explicit policy representation is required using PSO-P with a surrogate model to generate optimal actions.

Second, a feature selection heuristic called AMIFS [148] is used to order the possible features $(s_1, s_2, \dots) = \boldsymbol{s}$ of state $\boldsymbol{s}$ in terms of their relevance to the individual action dimensions $(a_1, a_2, \dots) = \boldsymbol{a}$. AMIFS uses mutual information as a measure of feature relevance and redundancy to reveal non-linear feature-action relations.

Finally, the resulting ordered feature lists, e.g., $\{s_3, s_2, s_1\}_{a_1}$ and $\{s_2, s_1, s_3\}_{a_2}$ for the action dimensions a_1 and a_2, are used to construct compact and interpretable fuzzy rule representations, whose parameters will then be optimized by FPSRL.

	MC	CPB	CPSU	IB
Policy rule sizes	2/4/6		2/4/6/8	2 (3x)
Policy state features	2	4		1/2/3/4 (3x)
Particles	1 000			
Iterations	1 000		10 000	
Swarm topology	ring with four neighbors			
Parameters (w, c_1, c_1)	$(0.72981, 1.49618, 1.49618)$			
Transition samples	10 000			100 000
Time horizon T	200	100	500	100
Discount factor γ	0.985	0.97	0.994	0.97
Model usage	for policy training			

Table 6.1: FPSRL experiments settings and parameters

6.3 Experiments with Interpretable Rule-based Policies by PSO

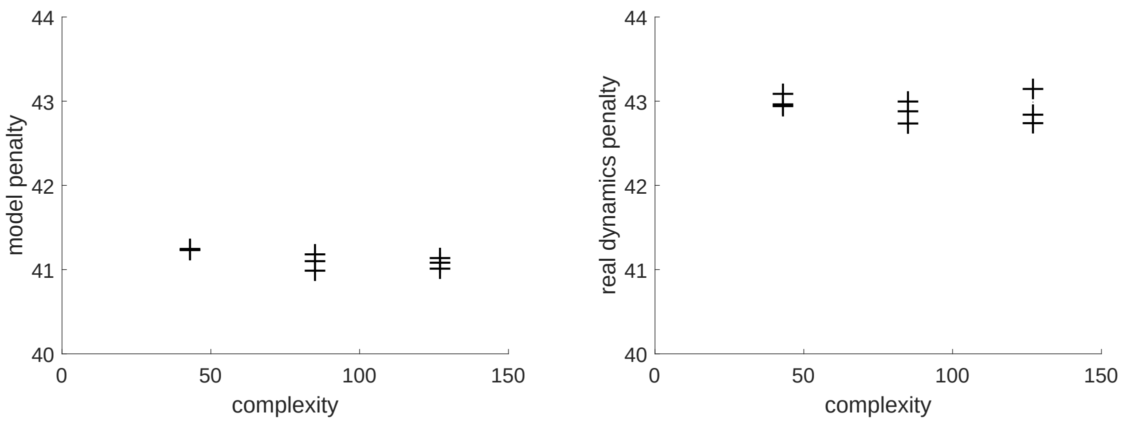
In this section, the performance of FPSRL policies with different complexities is evaluated on the four benchmarks mountain car (MC), cart-pole balancing (CPB), cart-pole swing-up (CPSU), and industrial benchmark (IB). To compare the rule-based policy results of FPSRL with that of fuzzy genetic programming reinforcement learning (FGPRL) from the next chapter, a complexity measure is introduced. The complexities of the policies are computed by using the weights from Table 7.2.

Table 6.1 lists important FPSRL and benchmark parameters which have been used in the following experiments. The performance in this section is presented as penalty values computed by multiplying the achieved fitness values by -1.

Mountain Car

For the FPSRL experiments on the MC benchmark, three different policy configurations are tested. Policies with two, four, and six rules resulting in complexity values of 43, 85, and 127, respectively. For each complexity ten independent FPSRL trainings have been conducted using 1 000 particles and 1 000 PSO iterations for each experiment.

For the MC benchmark and a discount factor $\gamma = 0.9851$ (resulting from $q = 0.05$), a policy with a penalty of 43 or lower is considered a successful solution to the benchmark problem. A policy with such performance can drive the car up the hill from any of the 100 initial states of the test sets in less than 200 time steps. Fig. 6.3 compares the performance of different rule complexities evaluated on the model and on the real system dynamics. Generally, it can be stated that all learned FPSRL policies yielded very good MC controllers with low penalty values. Not one of the 30 resulting policies has failed on the task to drive the car up the hill from every start state in less than 200 time steps. Policies of higher complexities only yielded slightly lower penalties. The differences with respect to the average penalty values between model and real dynamics



(a) Model (b) Real system dynamics

Figure 6.3: Results of FPSRL experiments for the MC benchmark. The markers represent (from top to bottom) the worst, median, and best FPSRL result after ten independent training runs. Both results for average model penalty as well as average real dynamics penalty are very good. Increasing the complexity of the fuzzy policies increased the performance only marginally, while bigger rule bases are expected to decrease the interpretability significantly.

stem from different start states in the evaluation set.

One way to visualize fuzzy policies is to plot the respective membership functions and analyze the produced output for the sample states. A graphical representation of a policy for the MC benchmark is given in Fig. 6.4.

Cart-pole Balancing

The task for FPSRL on the CPB benchmark was to find parameterizations for policies with two, four, and six fuzzy rules resulting in complexities 63, 125, and 187, respectively. Here, 1 000 particles with 1 000 PSO iterations have been used.

Since the CPB, as well as the CPSU, problem is symmetric around $\boldsymbol{s} = (\theta, \dot{\theta}, \rho, \dot{\rho}) = (0, 0, 0, 0)$, an optimal action a for state $(\theta, \dot{\theta}, \rho, \dot{\rho})$ corresponds to an optimal action $-a$ for state $(-\theta, -\dot{\theta}, -\rho, -\dot{\rho})$. Thus, the parameter search process can be simplified. It is only necessary to search for optimal parameters for one half of the policy's rules. The other half of the parameter sets can be constructed by negating the membership functions' mean parameters $c_j^{(i)}$ and the respective output values $o^{(i)}$ of the policy's components. Note that the membership function span width of the fuzzy rules (parameter $\sigma_j^{(i)}$ in Eq. (6.2)) is not negated because the membership functions must preserve their shapes.

The training employed 100 start states that were uniformly sampled from $[-0.5, 0.5] \times \{0\} \times [-2.0, 2.0] \times \{0\}$. With the discount factor $\gamma = 0.97$ ($q = 0.05$), policies that yield a penalty of 5.0 or lower are considered successful. The average penalty results depicted

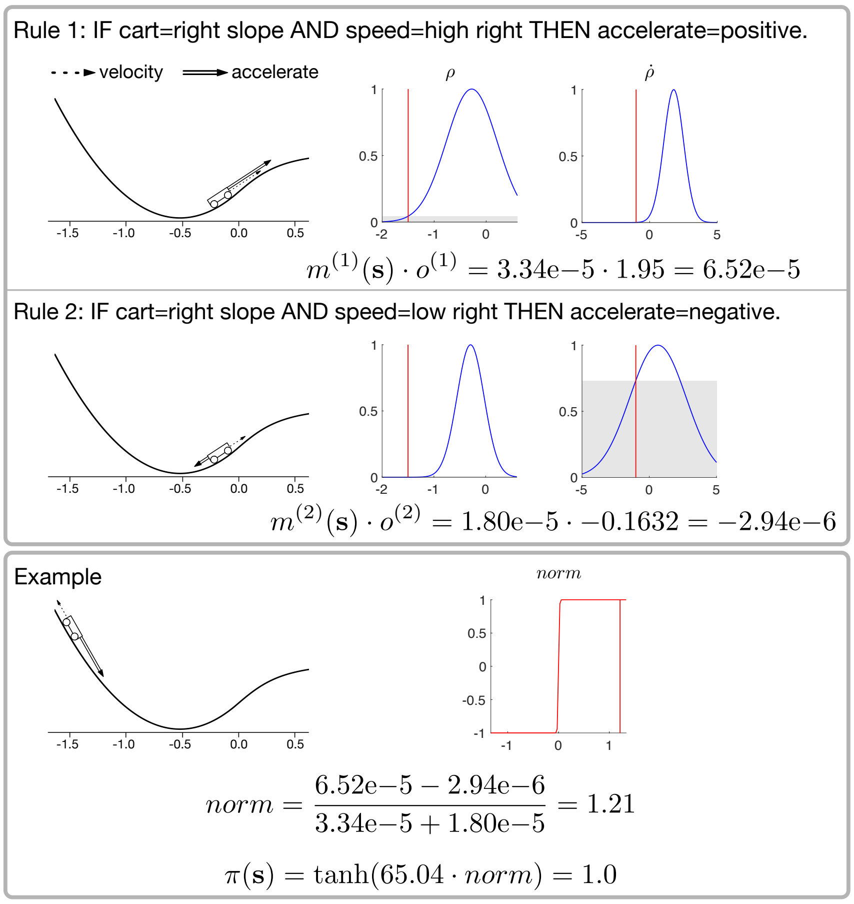
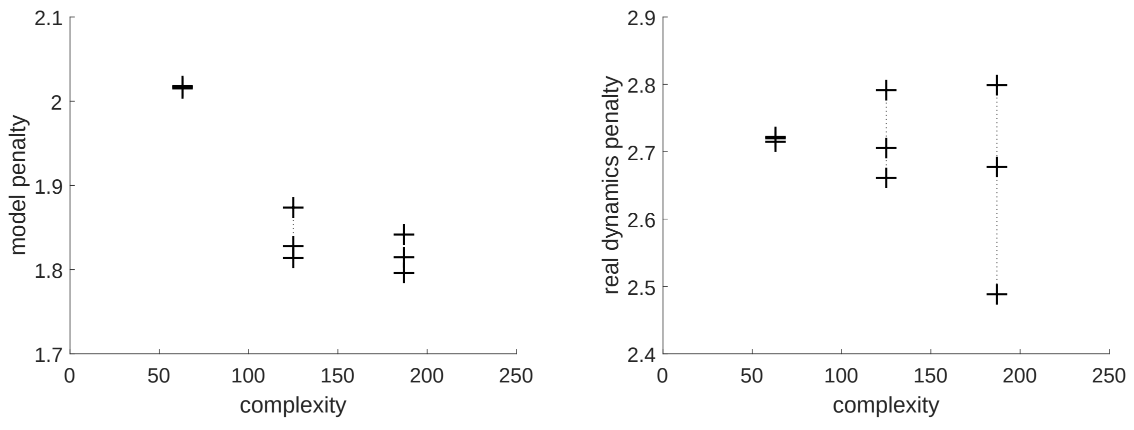


Figure 6.4: Fuzzy rules for the MC benchmark (membership functions plotted in blue, example state position plotted in red with a gray area for the respective activation grade). Both rules' activations are maximal at nearly the same position ρ, which implies that the ρ-dimension has minor influence on the policy's output. This observation fits the fact that, for the MC benchmark, a simplistic but successful policy exists, i.e., accelerate the car in the direction of its current velocity, regardless of its current position. Although this trivial policy yields good performance for the MC problem, better solutions exist. For example, stop driving to the left earlier at a certain position, results in reaching the goal in fewer time steps, which yields a higher average return. The depicted policy implements this advantageous solution as shown in the example section for state $\boldsymbol{s} = (-1.5, -1.0)$.



(a) Model (b) Real system dynamics

Figure 6.5: Results of FPSRL experiments for the CPB benchmark. Both results for average model penalty as well as average real dynamics penalty are very good. Increasing the complexity of the fuzzy policies reliably increased the performance only on the surrogate model and not on the real system dynamics, which is a strong hint that model inaccuracies were exploited during the training phase.

in Fig. 6.5 show that all three policy parameterizations yielded excellent performance results on the model as well as on the real system dynamics during evaluation. Note that despite policies with higher complexities performed better during training on the world model, their performance during evaluation on the real dynamics did not always improve accordingly. The reason for this might be that FPSRL policies with more rules have more degrees of freedom in terms of policy parameters. These additional parameters are used to overfit the policies to exploit model inaccuracies. However, during the evaluation phase policies can no longer exploit these artificial effects and sometimes perform even worse compared to policies with simpler structures.

A visual representation of one of the resulting fuzzy policies with two rules is given in Fig. 6.6.

Cart-pole Swing-up

The task for FPSRL on the CPSU benchmark was to find parameterizations for policies with two, four, six, and eight fuzzy rules resulting in complexities 63, 125, 187, and 249, respectively. Here, 1 000 particles with 10 000 iterations are used for each configuration. For this benchmark with $\gamma = 0.994$ ($q = 0.05$), solutions with a penalty of 50 or lower for a set of 100 benchmark states uniformly sampled from $[-\pi, \pi] \times \{0\} \times [-0.5, 0.5] \times \{0\}$ were considered successful. The policies exhibiting such performance can swing-up more than 99% of the given test states. Fig. 6.7 shows that no successful policy could be

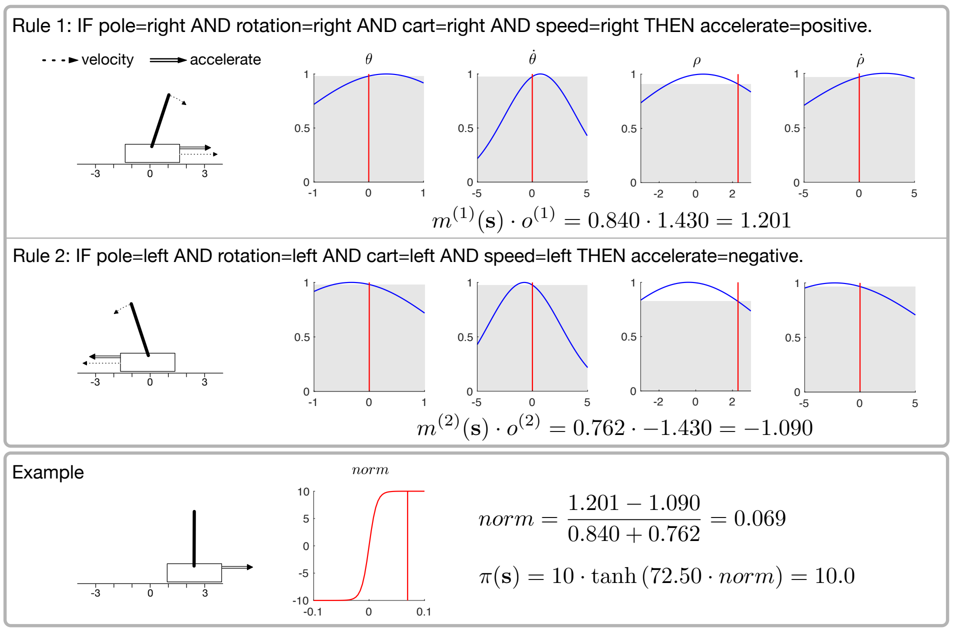

Rule 1: IF pole=right AND rotation=right AND cart=right AND speed=right THEN accelerate=positive.
velocity
accelerate
θ
$\dot{\theta}$
ρ
$\dot{\rho}$
$m^{(1)}(\mathbf{s}) \cdot o^{(1)} = 0.840 \cdot 1.430 = 1.201$
Rule 2: IF pole=left AND rotation=left AND cart=left AND speed=left THEN accelerate=negative.
$m^{(2)}(\mathbf{s}) \cdot o^{(2)} = 0.762 \cdot -1.430 = -1.090$
Example
$norm$
$norm = \frac{1.201 - 1.090}{0.840 + 0.762} = 0.069$
$\pi(\mathbf{s}) = 10 \cdot \tanh(72.50 \cdot norm) = 10.0$

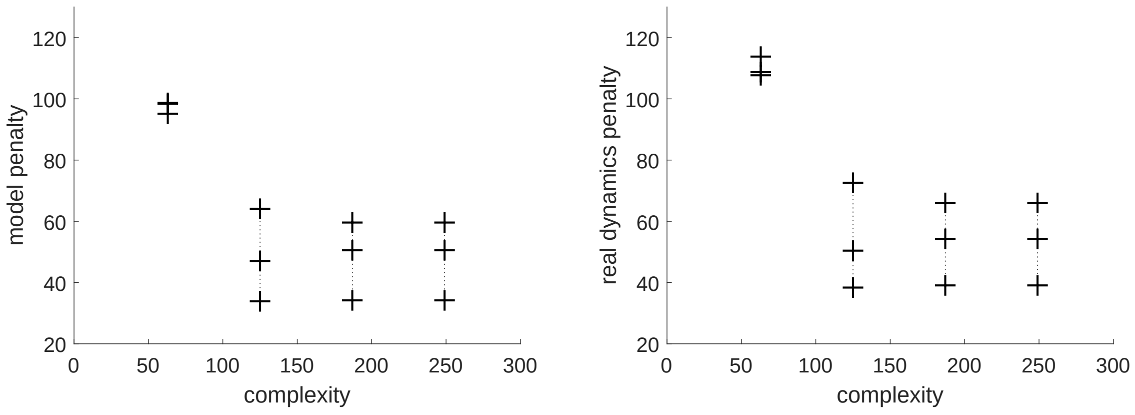
Figure 6.6: Fuzzy rule policy for the CPB benchmark. Even very compact and thus conveniently interpretable rule-base policies like the depicted one, yield high performance values and represent good CPB controllers.

found for complexity 63 (2 rules). Policies utilizing 4, 6, or 8 rules were able to produce good results for the CPSU task. However, none of the different complexity settings always produced good results. The average penalty values range from very good 38.4 up to bad 72.6. Nevertheless, good and bad policies can be identified as such by comparing them on their model penalty values. Repeating the FPSRL training several times and subsequently selecting the resulting policy with the lowest model penalty ensures to select a policy with very low penalty on the real system during evaluation.

Figure 6.8 shows how even a more complex fuzzy policy with four rules can be visualized and help make RL policies interpretable.

Industrial Benchmark

The IB's Markovian state approximation consists of 180 past observation features. Each of these features alone, or any combination of them can potentially contain crucial information required by the optimal policy. To construct interpretable fuzzy rules using FPSRL, suitable features have to be selected before conducting swarm optimization of the fuzzy parameters. The proposed feature selection method from Section 6.2 identified the following state variables as being the most important for each action dimension: Δv: $\{f_0, v_7, c_{13}, h_0\}$, Δg: $\{c_0, f_1, c_{15}, p\}$, and Δh: $\{h_4, p, h_{10}, h_0\}$. Here, the variables are listed in descending order of importance and the indices represent the time elapsed since the observation. Four different fuzzy rule structures were constructed based on



(a) Model (b) Real system dynamics

Figure 6.7: Results of FPSRL experiments for the CPSU benchmark. Only policies with four or more rules (complexity 125 and above) were able to produce good policies.

these variables. The first policy with a complexity of 99, incorporated only the first variable in each list into two rules per action dimension, while the other policies, with complexities of 129, 159, and 189, incorporated the first two, the first three, or all four variables, respectively.

Using the proposed feature selection heuristic (Section 6.2), FPSRL was able to generate policies with good performance (a model penalty of 5 700) for complexities of 129 or higher (Fig. 6.9). 1 000 particles have been used for 10 000 iterations to optimize the rule parameters of each of the 40 individual experiments, i.e., ten repeated FPSRL trainings for four different feature numbers. Like for PSONN, FPSRL training and evaluation were conducted using two different sets of 100 randomly drawn initial start states from the existing trajectory data.

Fig. 6.10 depicts the best resulting FPSRL policy which uses two state features per action dimension. Note how structured and compact this representation visualizes a well performing policy for a highly complex problem with 180 available state features.

6.4 Discussion

The traditional way to create self-organizing fuzzy controllers either requires an expert-designed fitness function according to which the optimizer finds the optimal controller parameters or relies on detailed knowledge regarding the optimal control policy. Either requirement is difficult to satisfy when dealing with real-world industrial problems. However, data gathered from the system to be controlled using some default policy is available in many cases.

The FPSRL approach can use such data to produce high-performing and interpretable

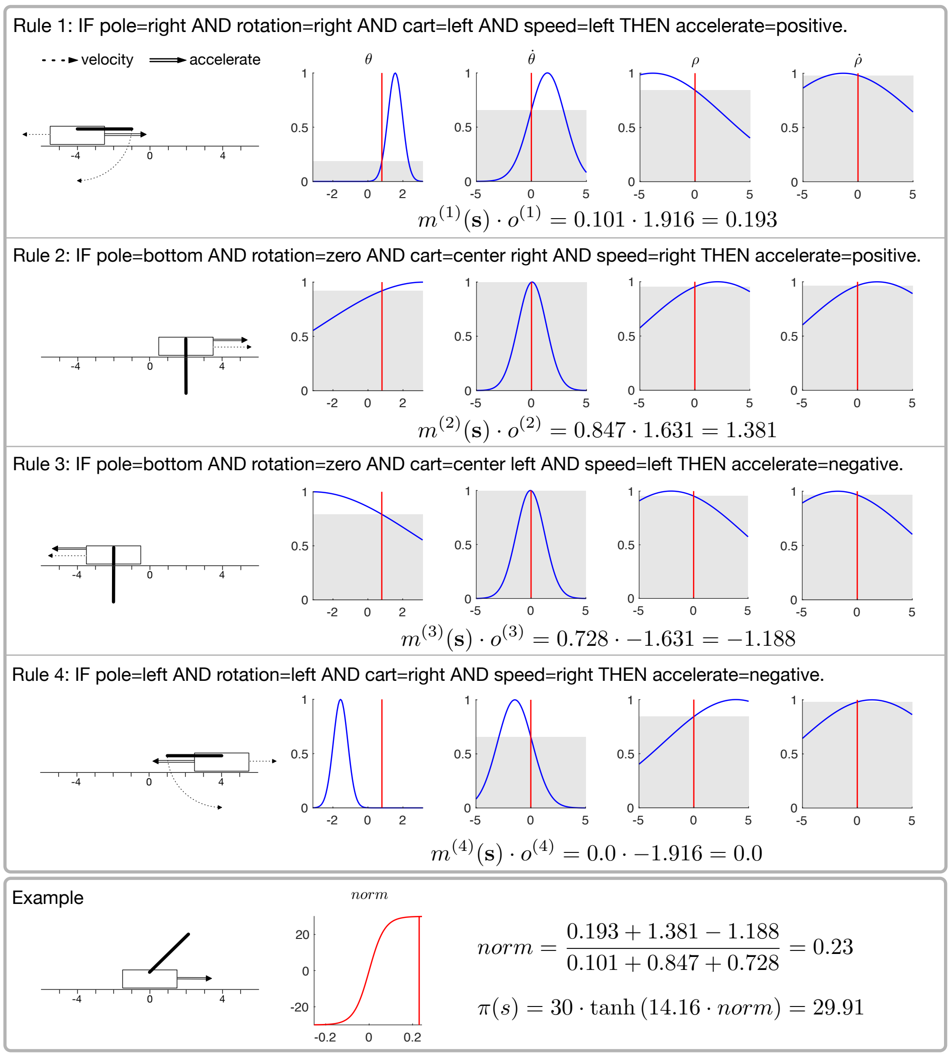
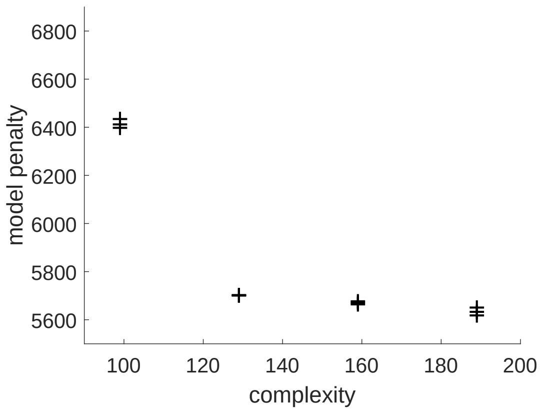
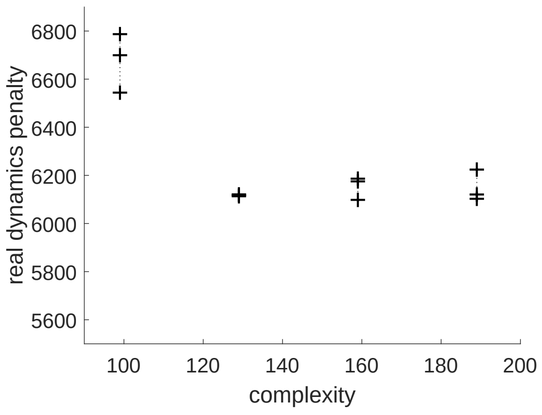


Figure 6.8: Fuzzy rules for the CPSU benchmark. Even with four rules, fuzzy policies can be visualized in an easily interpretable way. By inspecting the prototype cart-pole diagrams for each rule, two basic concepts can be identified for accelerating in each direction. First (rule 1): the cart's position is on the left and moving further to the left, while the pole is simultaneously falling on the right. Then the cart is accelerated towards the right. Second (rule 2): the cart is between the center and the right and the pole is hanging down. Then the cart is accelerated towards the right. Note that rules 3 and 4 are copies of rules 1 and 2 with opposite signs on their membership centers and outputs. The prototypes are utilized to realize the complex task of swinging the pole up. Balancing of the pole while the cart is centered around $\rho = 0$ is realized via fuzzy interaction of these prototype rules, as shown in the example in the last row.



(a) Model



(b) Real system dynamics

Figure 6.9: Results of FPSRL experiments for the IB. Only policies with two or more integrated state features (complexity $\geq$ 129) were able to produce good policies.

fuzzy policies for RL problems. Particularly for problems where system dynamics are rather easy to model from an adequate amount of data and where the resulting RL policy can be expected to be compact and interpretable, the proposed FPSRL approach might be of interest to industry domain experts.

The experimental results obtained with three standard RL toy benchmarks have demonstrated the advantages and limitations of the proposed model-based method. In addition to that, FPSRL has been applied on a highly stochastic high-dimensional problem with multi-dimensional action space, the IB. This benchmark requires an additional feature selection step to yield interpretable FPSRL policies. We have seen that applying a combination of PSO-P search for optimal action trajectories on the world model first, and subsequently exploiting the so-created state-action tuples using the feature selection heuristic AMIFS yields very useful importance ratings for the available state features. Systematically integrating only the features with the highest importance rating yields compact and well performing rule-based policies for the IB.

The performances of the FPSRL results for all four benchmarks were as good as the PSONN results. However, one of the most significant advantages of FPSRL over other RL methods is the fact that fuzzy rules can be easily and conveniently visualized and interpreted. Furthermore, a compact and informative approach to present fuzzy rule policies that can serve as a basis for discussion with domain experts has been proposed in the course of this chapter.

The application of the proposed FPSRL approach in industry settings could prove to be of significant interest because, in many cases, data from systems are readily available and interpretable fuzzy policies are favored over black-box RL solutions.

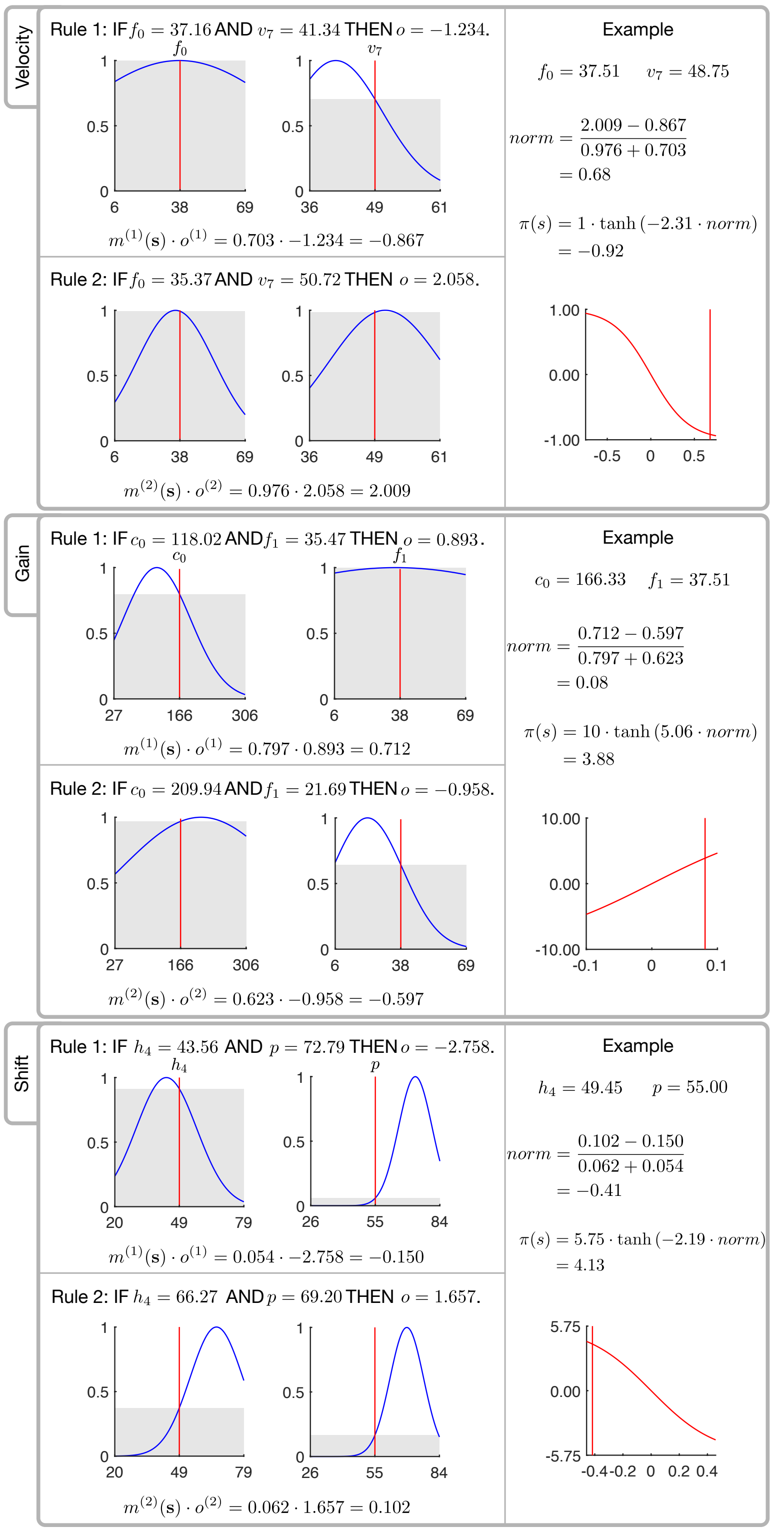


Figure 6.10: Fuzzy rules trained by FPSRL for the IB

CHAPTER 7

Rule-based Policies by Genetic Programming

In this chapter, a new interpretable reinforcement learning (RL) approach called *fuzzy genetic programming reinforcement learning* (FGPRL) is introduced and evaluated. The method fuzzy particle swarm reinforcement learning (FPSRL) from Chapter 6, has shown that an evolutionary computation method, namely particle swarm optimization (PSO), can be successfully combined with fuzzy rule-based systems to generate interpretable RL policies. This combination can be achieved by first training a model on a batch of pre-existing state-action trajectory samples and subsequently conducting model-based RL. In contrast to PSO in FPSRL, FGPRL utilizes genetic programming (GP) to parametrize rule-based policies (Fig. 7.1).

Applying FPSRL to systems with many state features is prone to yield non-interpretable fuzzy systems since every rule contains all the state dimensions, including redundant or irrelevant state dimensions, in its membership function by default. To overcome this limitation, an approach to efficiently determine the most important state features with respect to the optimal policy has been proposed (Section 6.2). Selecting only the most informative features before policy parameter training makes the production of interpretable fuzzy policies using FPSRL possible again. However, performing a heuristic feature selection initially and subsequently creating policy structures manually, is a feasible but limited approach; in high-dimensional state and action spaces, the effort involved grows exponentially.

By creating fuzzy rules using GP rather than tuning the fuzzy rule parameters via PSO, FGPRL eliminates the manual feature selection process. GP is able to automatically select the most informative features as well as the most compact fuzzy rule representation for a certain level of performance. Moreover, it returns not just one solution to the problem but a whole Pareto front containing the best-performing solutions for many different levels of complexity.

Although genetic fuzzy systems have demonstrated their ability to learn and adapt to

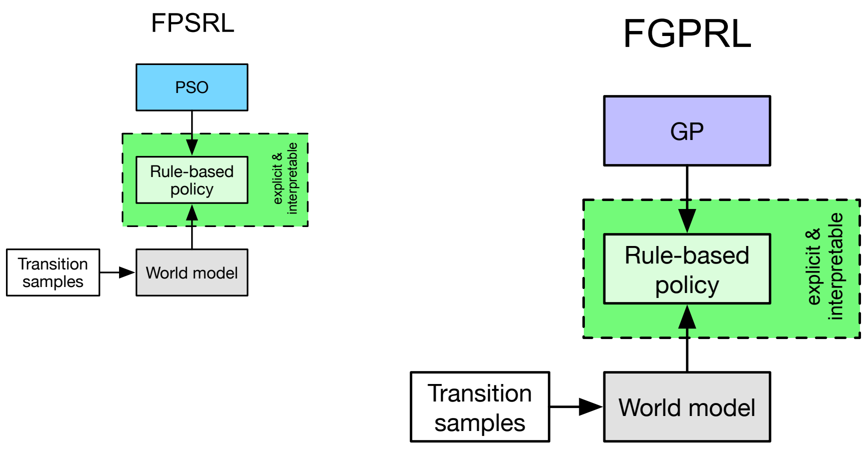


Figure 7.1: The FGPRL framework compared to FPSRL

solve different types of problems in various application domains, GP-generated fuzzy logic controllers have not been combined with a model-based batch RL approach so far.

Combining a fuzzy system's approximate reasoning with an evolutionary algorithm's ability to learn allows the proposed method to learn human-interpretable soft-computing solutions autonomously. Cordón, Gomide, Herrera, et al. provide an extensive overview of previous genetic fuzzy rule-based systems [22]. While most of the existing research in this area has focused on the genetic tuning of scaling and membership functions as well as genetic learning of rule and knowledge bases, less attention has been paid on using GP to design fuzzy rule-based systems. Since GP is concerned with automatically generating computer programs [81], it should theoretically be able to achieve both - learn rule and knowledge bases, as well as tune scale and membership functions, simultaneously [50].

Fuzzy rule-based systems have been combined with GP for modeling [68] and classification [110, 124, 19, 9] tasks. In the field of optimal system control, early applications that combine GP and fuzzy rule-based systems for mobile robot path tracking have been demonstrated [151]. Type-constraint GP has also been used to define fuzzy logic controller rule-bases for the cart-centering problem [2, 1]. Memetic GP, which combines local and global optimization, has been used to train Takagi-Sugeno fuzzy controllers to solve the cart-pole problem [150]. Recently, based on the GP fuzzy inference system (GPFIS), GPFIS-control has been proposed [79]. They used multi-gene GP to automatically train a fuzzy logic controller and tested its performance on the cart-centering and inverted pendulum problems.

In this chapter, FGPRL is applied to the same four different benchmarks like in the experiments before and the policies' performances and the interpretability of its resulting fuzzy system controllers are compared with the results of other RL methods.

This chapter is based on the following publication as part of the doctoral research:

- D. Hein, S. Udluft, and T. A. Runkler. 2018. "Generating interpretable fuzzy con-

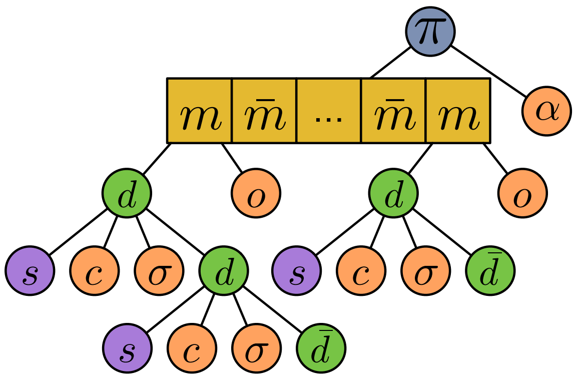


Figure 7.2: A fuzzy GP individual

trollers using particle swarm optimization and genetic programming." In GECCO '18 Companion: Proceedings of the 2018 Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan. ACM, New York, NY, USA, pp. 1268-1275. [66]

7.1 Fuzzy Genetic Programming Reinforcement Learning (FGPRL)

In this section, a GP-based approach is introduced that can generate interpretable fuzzy rule policies automatically using model-based batch RL. Analogously to FPSRL, FGPRL uses an approximate system model to predict the performance of policy candidates and subsequently uses this knowledge to generate high-performing policies iteratively. Unlike FPSRL, FGPRL not only optimizes the predefined policy parameters but also automatically selects the relevant state features, finds the necessary number of rules, and returns a Pareto front of policy candidates with different levels of complexity.

FGPRL is based on GP (Section 2.3), which 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. Since we are interested in using interpretable fuzzy controllers as RL policies, the genes include membership and defuzzification functions, as well as constant floating-point numbers and state variables. These fuzzy policies can be represented as function trees (Fig. 7.2) and stored efficiently in arrays.

Here, strongly-typed GP is used for FGPRL to avoid constructing ill-defined rules [1]. This means that each building block is assigned a type (Table 7.1). The different colors in Fig. 7.2 highlight the example individual's type structure. During the crossover process, only cutting points of equal type (color) are selected to ensure that only legal offspring are created.

To yield fuzzy rules with the structure described in Section 6.1, an additional tree correction must be applied. Generally, the GA can construct rules where two or more activation functions act on the same state variable, such as in Fig. 7.2 where the same s

Type	
Variable	s
Floating-point number	c, σ, o, α
Dimension	$d, \bar{d}$
Rule	$m, \bar{m}$
Policy	π

Table 7.1: Types of the GP building blocks

Complexity	
$\pi, \bar{m}, \bar{d}$	0
s	1
c, σ, o, α	1
d	2
m	10

Table 7.2: Complexities of the GP building blocks. The terminal blocks $\bar{m}$ and $\bar{d}$ have complexity value 0.

appears twice below the same rule m. Such rules are expected to be difficult to interpret since their shape does not conform to the standard Takagi-Sugeno fuzzy inference model as introduced in Section 6.1. Therefore, for FGPRL, every tree is checked before evaluating its fitness by looking for recurring state variables and cutting out their corresponding activation functions. Note that the structures of the subsequent activation functions are not affected.

For FGPRL, a simple node counting measurement strategy is used as complexity measure where different types of functions, variables, and terminals were weighted differently. Table 7.2 lists the weights (complexities) which are used in the experiments presented in Section 7.2. The weights yield a problem-specific balance between learning controllers consisting of more rules with fewer dimensions and vice versa.

Local Search

Since FGPRL's GP process searches the entire fuzzy policy structure space, it is prone to underestimate the importance of local fuzzy parameter tuning [103, 150]. To counteract this problem an additional parameter tuning step to all the terminals (c, σ, o, α) after optimization is complete (Fig. 7.3) is proposed. Applying PSO to every individual in the final Pareto front yields an updated front comprising at most the same number of individuals with equal or higher fitness values.

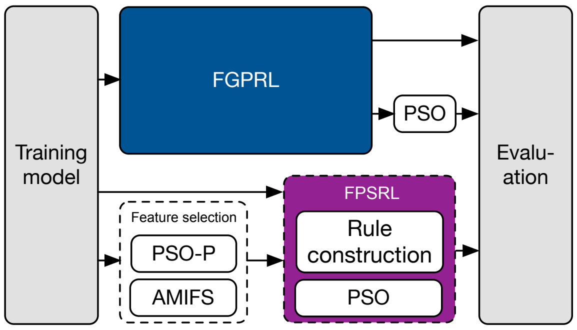


Figure 7.3: Comparing FPSRL with FGPRL. Note that FGPRL does not require a preceding feature selection step. However, applying a local parameter tuning step subsequent to FGPRL using PSO has shown to improve the performance of the Pareto optimal individuals.

7.2 Experiments with Interpretable Rule-based Policies by Genetic Programming

In this section, the results of the FGPRL experiments are discussed and compared with the results using FPSRL from Chapter 6. To yield a fair comparison regarding computational effort, the parameters have been chosen in such a way that both methods were able to employ a similar number of fitness value calculations. Table 7.3 contains the parameters and calculation steps for the benchmarks cart-pole swing-up (CPSU) and industrial benchmark (IB). Table 7.4 lists important FGPRL and benchmark parameters which have been used in the following experiments. The performance in this section is presented as penalty values computed by multiplying the achieved fitness values by -1.

Mountain Car

Fig. 7.4 compares the results of mountain car (MC) experiments with FPSRL and FGPRL in which the benchmark has been used ten times for each method. FPSRL produced 30 policies for three complexity levels, while FGPRL produced 95 policies for 25 complexity levels.

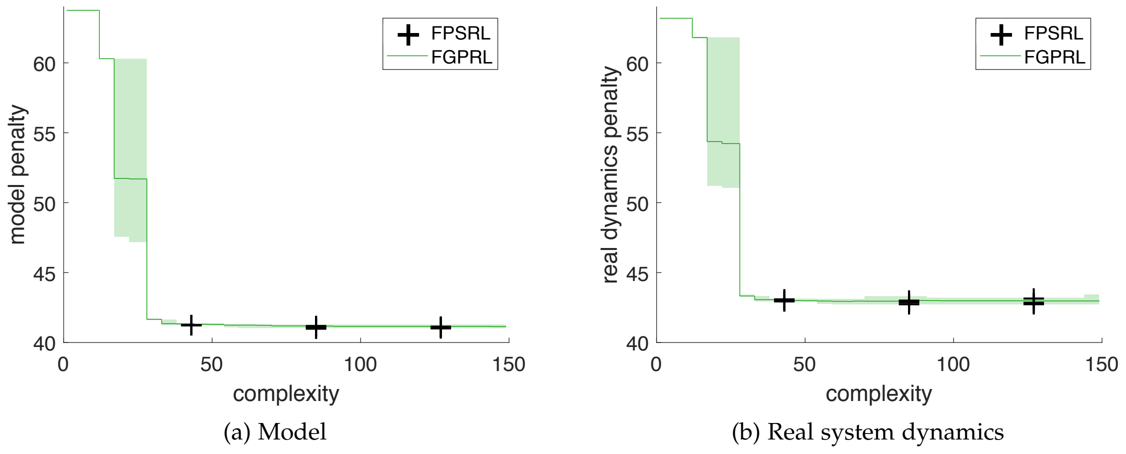
The most compact rule-based policy for FPSRL consisted of only two rules; one rule for driving left and one rule for driving right. Both rules applied membership functions on both state features, namely position ρ and cart velocity $\dot{\rho}$. In Fig. 6.4 one of these policies is visualized, to show how interpretable such simple rule-based policies can be. However, FGPRL found even simpler policies that perform just slightly worse. These policies with complexity value 28 also consist of two rules for driving left and right. However, only one of these rules has a dependency to a state feature. The other rule is activated per default and serves as some bias rule, i.e., the bias rule drives the cart to the left as long as the other rule which drives the cart to the right is not activated over a particular value. Note that FGPRL has also found more complex solutions which

	FPSRL	FGPRL
Particles/Individuals	1,000	1,000
Iterations/Generations	× 10,000	× 10,000
Fitness value calculations (A)	1e+7	1e+7
Optimization result	Single policy	Pareto front of policies
	1	22 to 41
Additional local search	× 0	× 1e+6
Additional local fitness value calc. (B)	0	2.2e+7 to 4.1e+7
Runs for multiple complexities	4	1
(A)	× 1e+7	× 1e+7
Fitness value calc. for multiple compl. (C)	4e+7	1e+7
Total fitness value calc. (B)+(C)	4e+7	3.2e+7 to 5.1e+7

Table 7.3: Computational costs for FPSRL and FGPRL

	MC	CPB	CPSU	IB
Policy max complexity	150	200	300	250
Individuals	1 000			
Generations	1 000		10 000	
New population ratios	$r_\mathrm{c} = 0.45, r_\mathrm{r} = 0.05,$ $r_\mathrm{m} = 0.1, r_\mathrm{a} = 0.1$			
Transition samples	10 000			100 000
Time horizon T	200	100	500	100
Discount factor γ	0.985	0.97	0.994	0.97
Model usage	for policy training			

Table 7.4: FGPRL experiments settings and parameters



(a) Model

(b) Real system dynamics

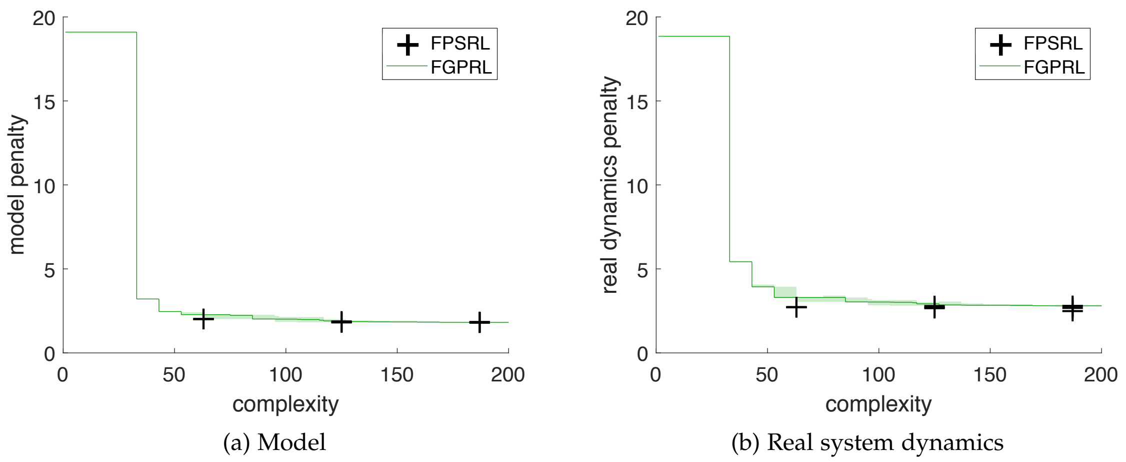
Figure 7.4: Comparison of FPSRL and FGPRL performance for the MC, for both the model and the real system. The green lines represent the median Pareto fronts over the FGPRL experiments. The semi-transparent green areas show the minimum and maximum penalties obtained from the experiments, while the black markers indicate the FPSRL results. The markers show, from top to bottom, the maximum, median, and minimum penalties for each complexity level. The Pareto fronts resulting from training on the system model are on the left, while the right-hand plot shows the manner in which testing the same policies against the real system dynamics *altered* the fronts.

perform slightly better.

Cart-pole Balancing

Fig. 7.5 compares the results of cart-pole balancing (CPB) experiments with FPSRL and FGPRL in which the benchmark has been used ten times for each method. FPSRL produced 30 policies for three complexity levels, while FGPRL produced 126 policies for 33 complexity levels.

Similarly to the MC experiments, FGPRL found less complex rule-based policies which only perform slightly worse in comparison with the FPSRL results. Comparing the results of both methods for the complexity level 63 shows that FPSRL yielded in all of the ten individual experiments a model penalty of 2.0. In contrast to that, the median penalty of the FGPRL results is at 2.28, and only one of the resulting policies achieved 2.0. Thus, comparing the performance produced on the real system dynamics shows that the FPSRL results for this complexity are better compared to the FGPRL results.



(a) Model (b) Real system dynamics

Figure 7.5: Comparison of FPSRL and FGPRL performance for the CPB, for both the model and the real system

Cart-pole Swing-up

Fig. 7.6 depicts the results of CPSU experiments in which the benchmark has been used ten times for each method. FPSRL produced 40 policies for four complexity levels, while FGPRL produced 278 policies for 96 complexity levels. Both methods involved a similar number of fitness value calculations (Table 7.3). For problems such as the CPSU, which have low-dimensional state spaces and no irrelevant or redundant state variables, applying prior knowledge to the rule construction step yields a rule structure that can be easily tuned to produce high-performance interpretable fuzzy policies. FPSRL can utilize all the available computational resources to tune a fixed set of parameters.

However, FGPRL has to employ the same resources to search a significantly larger space of possible solutions. Note that although FGPRL is theoretically able to produce exactly the same fuzzy policies for complexity 125 as FPSRL, it was unable to find a comparable solution for the system models in any of the experiments (Fig. 7.6). The best individual it produced with a complexity of 125 or less had a penalty value of 48.88, which was even above the median FPSRL penalty value of 47.05.

Comparing the model penalties with the real dynamics penalties yields another interesting observation. Even though the best FGPRL policies never surpassed FPSRL's performance for complexities of 300 and below on the system model, when they were evaluated using the real dynamics, some FGPRL policies actually performed better than the FPSRL policies. This is possible because the swarm optimization had already started to overfit the fuzzy parameters with respect to the system model; i.e., FPSRL was exploiting model inaccuracies, thus reducing its performance on the real dynamics.

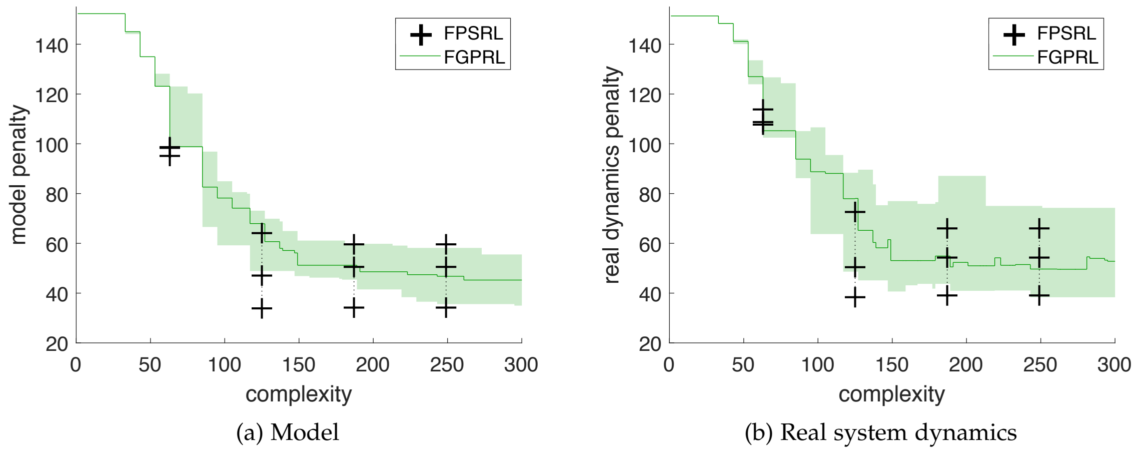


Figure 7.6: Comparison of FPSRL and FGPRL performance for the CPSU, for both the model and the real system

Industrial Benchmark

As with the toy benchmark experiments presented before, the IB has been utilized ten times for each method. FPSRL produced 40 policies for four complexity levels, while FGPRL produced 368 policies for 86 complexity levels.

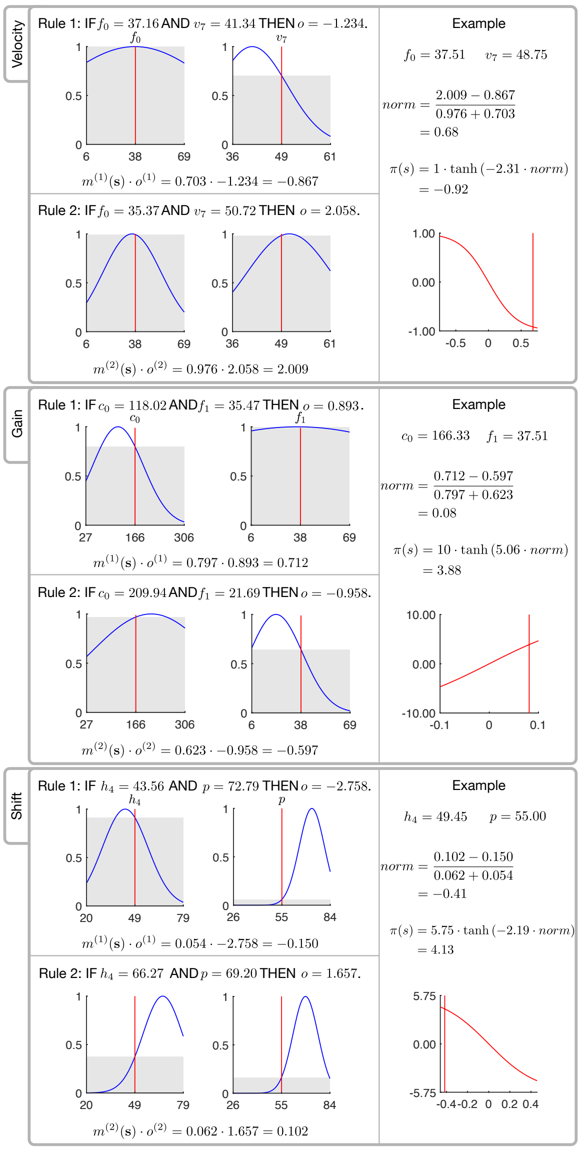
FGPRL was able to generate policies of a significantly low complexity with a higher performance, achieving a penalty of 5 635 at a complexity of 94 (Fig. 7.7). Moreover, the FGPRL search space covers all possible combinations of state dimensions and numbers of rules for each action dimension. For industrial problems where the state-to-action dependencies are not known a priori, this ability to search automatically is expected to be highly valuable to control system designers and domain experts.

Comparing the policies' performance on the approximate IB model with their performance on the real IB dynamics shows that the results of training can be transferred to the real system as long as the model's regression and generalization quality is adequate (Fig. 7.8).

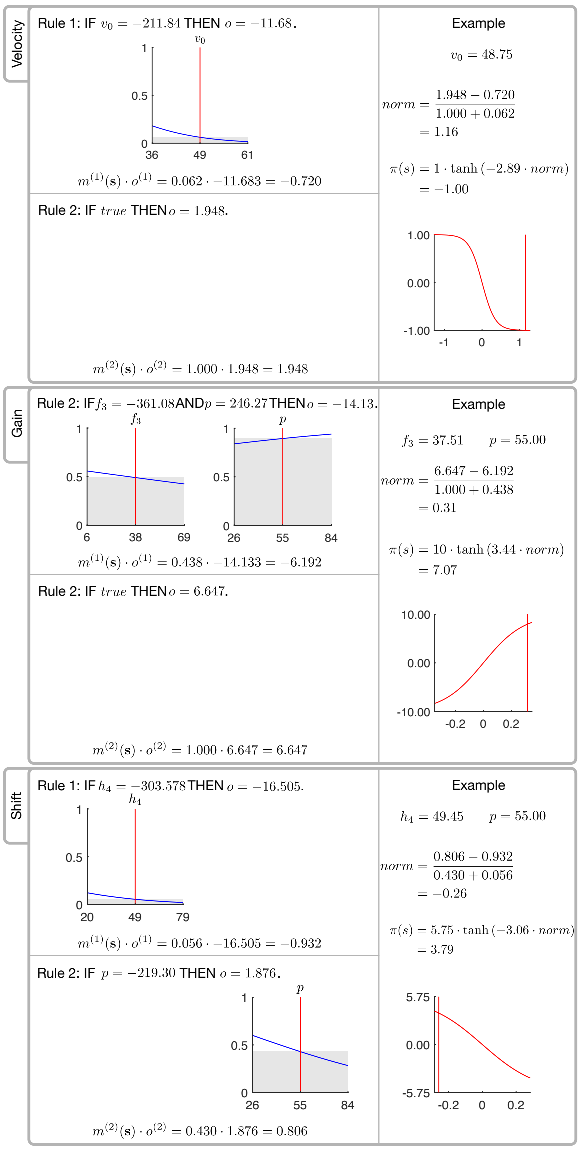
7.3 Discussion

In this chapter, a GP-based fuzzy policy learning approach, called FGPRL, has been introduced, that utilizes the same model-based batch RL technique as PSONN and FPSRL. However, instead of only tuning the parameters of NNs or fixed fuzzy policy structures, FGPRL searches the full space of all possible fuzzy controllers of the representational form described in Section 6.1, by determining the important state variables and the number of rules required and subsequently tunes all the rule parameters.

Experiments using the standard RL toy benchmarks MC, CPB, and CPSU show that

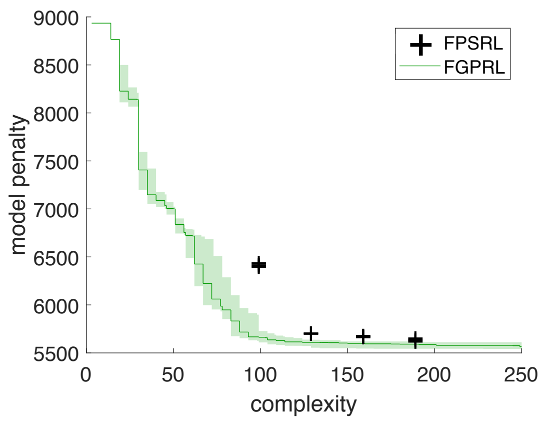


(a) FPSRL

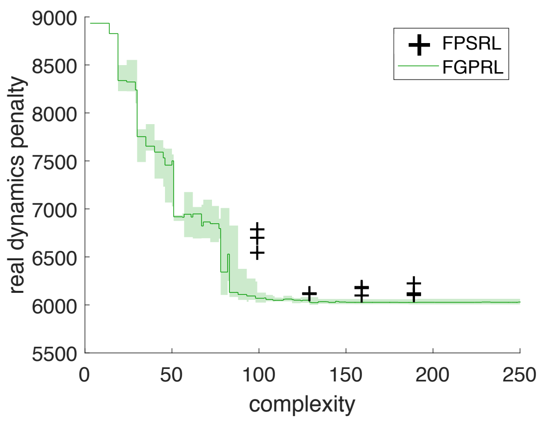


(b) FGPRL

Figure 7.7: Comparison of the fuzzy policies produced by FPSRL and FGPRL for the IB. Although both policies have similar penalty values (7.7a: 5700, 7.7b: 5635), their complexity is very different. The FGPRL policy only involves the minimum number of state variables required to yield adequate performance, meaning that the fuzzy controller is substantially more interpretable.



(a) Model



(b) Real system dynamics

Figure 7.8: Comparison of FPSRL and FGPRL performance for the IB, for both the model and the real system

FPSRL has advantages when no feature selection is necessary and the number of rules required can be readily determined by only testing a few different options. In this case, FGPRL's vast search space is a drawback, and it was far less likely to converge to a solution with similar performance to that produced by FPSRL given a similar number of fitness value calculations. However, FGPRL was occasionally able to produce high-performance solutions for these benchmarks, too.

Experiments using the IB, a benchmark that mimics real industrial systems like gas or wind turbines, yielded significant advantage for FGPRL over FPSRL. This benchmark has a high-dimensional state space, a multidimensional action space, stochastic and delayed effects, and a reward function with multiple criteria. Applying feature selection to FPSRL and manually testing different fuzzy policy structures did not yield satisfactory performance for low complexity solutions. In contrast, FGPRL was able to find high-quality interpretable solutions of low complexity with a similar number of fitness value calculations.

CHAPTER 8

Equation-based Policies by Genetic Programming

This chapter introduces a genetic programming (GP) approach for autonomously learning interpretable reinforcement learning (RL) policies from previously recorded state transitions. In the previous chapters, the interpretable methods fuzzy particle swarm reinforcement learning (FPSRL) and fuzzy genetic programming reinforcement learning (FGPRL) have shown that using fuzzy rules in batch RL settings can be considered an adequate solution to the task of learning human-interpretable and understandable policies trained by data-driven learning methods. However, in many cases the successful use of fuzzy rules requires prior knowledge about the shape of the membership functions, the number of fuzzy rules, the relevant state features, etc. Moreover, for some problems, the policy representation as a set of fuzzy rules might be unfavorable by some domain experts. The proposed *genetic programming reinforcement learning* (GPRL) approach learns policy representations which are basic algebraic equations of low complexity. Fig. 8.1 compares the GPRL framework with the previously proposed GP-based method FGPRL.

GP has been utilized for creating system controllers since its introduction by Koza [81]. Since then, the field of GP has grown significantly and has produced numerous results that can compete with human-produced results, including controllers, game playing, and robotics [82]. Keane, Koza, and Streeter automatically synthesized a controller by using GP, outperforming conventional PID controllers for an industrially representative set of plants. Shimooka and Fujimoto applied GP to generate equations for calculating the control force for a movable inverted pendulum by evaluating the individuals' performances on predefined fitness functions.

A fundamental drawback of these methods is that in many real-world scenarios such dedicated expert generated fitness functions do not exist. In RL, the goal is to derive well-performing policies only by (i) interacting with the environment, or by (ii) extracting knowledge out of pre-generated data, running the system with an arbitrary

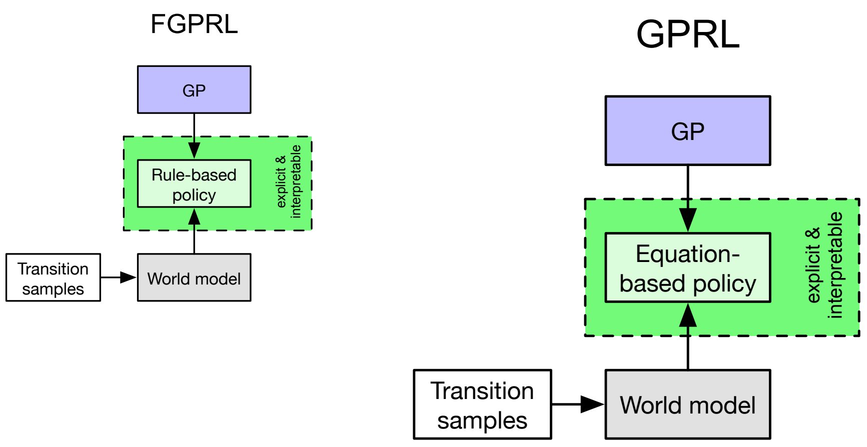


Figure 8.1: The GPRL framework compared to FGPRL

policy [147]. (i) is referred to as the online RL problem, for which Q-learning methods are known to produce excellent results. For (ii), the offline RL problem, model-based algorithms are usually more stable and yield better-performing policies [62].

In the proposed GPRL approach, the performance of a population of basic algebraic equations is evaluated by testing the individuals on a world model using the Monte Carlo method [147]. As introduced in Section 2.1, the combined return value of a number of action sequences is the fitness value that is maximized iteratively from GP generation to generation.

GP in conjunction with online RL Q-learning has been used in [31] on standard maze search problems and in [74] to enable a real robot to adapt its action to a real environment. Katagiri, Hirasawa, Hu, et al. introduced genetic network programming (GNP), which has been applied to online RL in [90] and improved by Q-tables in [89]. In these publications, the efficiency of GNP for generating RL policies has been discussed. This performance gain, in comparison to standard GP, comes at the cost of interpretability, since complex network graphs have to be traversed to compute the policy outputs. Gearhart examined GP as a policy search technique for Markov decision processes (MDPs). Given a simulation of the Freecraft tactical problem, he performed Monte Carlo simulations to evaluate the fitness of each individual [49]. Note that such exact simulations are usually not available in industry. Recently, Wilson, Cussat-Blanc, Luga, and J. F. Miller applied mixed type cartesian GP to Atari playing [157]. They show that by evaluating the programs of the best evolved individuals, simple but effective strategies can be found. Similarly, in [91] Monte Carlo simulations have been drawn in order to identify the best policies. However, the policy search itself has been performed by formalizing a search over a space of simple closed-form formulas as a multi-armed bandit problem. This means that all policy candidates have to be created in an initial step at once and are subsequently evaluated. The computational effort to follow this approach combinatorially explodes as soon as more complex solutions are required to solve more complicated control problems. However, GPRL is the first method which

Variables	1
Terminals	1
$+, -, \cdot$	1
$/$	2
$\wedge, \vee, >, <$	4
tanh, abs	4
if	5

Table 8.1: GPRL complexities

Individual representation	tree-based
Initialization method	grow method
Selection method	tournament selection (size=3)
Terminal set	state variables, random float numbers $z \sim [-20.0, 20.0]$, $\top, \bot$
Function set	$+, -, \cdot, /, \wedge, \vee, \text{if}, >, <,$ tanh, abs

Table 8.2: GP parameters

combines GP-generated policies with a model-based batch RL approach.

This chapter is based on the following publication as part of the doctoral research:

- D. Hein, T. A. Runkler, and S. Udluft. "Interpretable policies for reinforcement learning by genetic programming." In: Engineering Applications of Artificial Intelligence 76 (2018), pp. 158-169. [67]

8.1 Genetic Programming Reinforcement Learning (GPRL)

The interpretable policies which are generated by applying the GPRL approach are basic algebraic equations. 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. To rate the quality of each policy candidate, a fitness value has to be provided for the GP algorithm to advance (Section 2.3). For GPRL, the fitness $\tilde{\mathcal{F}}$ of each individual is calculated by generating trajectories using the world model $\tilde{g}$ starting from a fixed set of initial benchmark states. Table 8.1 lists the weightings applied in the GPRL experiments presented in Section 8.2. Table 8.2 gives an overview of the GP parameters and methods used in GPRL.

The performance of GPRL is compared to a rather straightforward approach, which utilizes GP to conduct symbolic regression on a data set $\hat{\mathcal{D}}$ generated by a well-performing

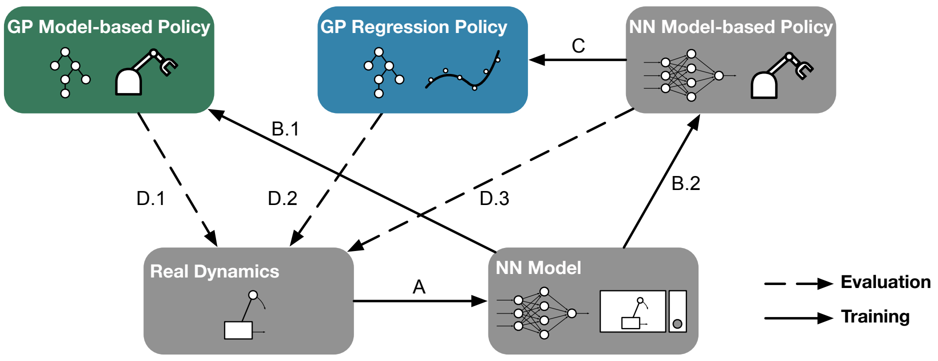


Figure 8.2: The proposed GPRL approach and its integration in the experimental setup. The world model is the result of supervised ML regression on data originated from the real dynamics (A). GPRL generates a GP model-based policy by training on this world model (B.1), which is an NN model in our experimental setup. Similarly, the PSONN policy is trained in a model-based manner by utilizing the same NN model (B.2). In contrast to both other policy training approaches, the GP regression policy mimics the already existing PSONN policy by learning to minimize an error with respect to the existing policy's action (C). All of the policies are finally evaluated by comparing their performance on the real dynamics (D.1-D.3).

but non-interpretable RL policy $\hat{\pi}$. $\hat{\mathcal{D}}$ contains tuples $(\boldsymbol{s}, \hat{\boldsymbol{a}})$, where $\hat{\boldsymbol{a}}$ are the generated actions of policy $\hat{\pi}$ on state $\boldsymbol{s}$. The states originate from trajectories created by policy $\hat{\pi}$ on world model $\tilde{g}$. One might think that given an adequate policy of any form and using GP to mimic this policy by means of some regression error with respect to $\hat{\boldsymbol{a}}$, could also yield successful interpretable RL policies. However, the experimental results indicate that this strategy is only successful for rather simple problems and produces highly non-stable and unsatisfactory results for more complex tasks.

In the experiments, the well-performing but non-interpretable RL policy $\hat{\pi}$ used to generate data set $\hat{\mathcal{D}}$ is a particle swarm optimization neural network (PSONN) policy from Chapter 5.3. The weights of this policy have been trained by model-based RL on the very same world models as applied for GPRL. Usually, $\hat{\pi}$ is expected to yield higher fitness values during training, since it can utilize significantly more degrees of freedom to compute an optimal state action mapping than basic algebraic equations found by GPRL.

Figure 8.2 gives an overview of the relationships between the different policy implementations, i.e., GPRL, GP for symbolic regression, and PSONN policy, and the environment instances, i.e., the neural network (NN) system model and the real dynamics, used for training and evaluation, respectively.

	MC	CPB	CPSU	IB
Policy max complexity	100			
Individuals	1 000			
Generations	1 000		10 000	
New population ratios	$r_\mathrm{c} = 0.45, r_\mathrm{r} = 0.05,$ $r_\mathrm{m} = 0.1, r_\mathrm{a} = 0.1$			
Transition samples	10 000			100 000
Time horizon T	200	100	500	100
Discount factor γ	0.985	0.97	0.994	0.97
Model usage	for policy training			

Table 8.3: GPRL experiments settings and parameters

8.2 Experiments with Interpretable Equation-based Policies

In this section, the performance of GPRL is evaluated empirically on four RL benchmarks. Note that compared to other publications [79], a rather big population size of 1 000 in combination with a higher new individuals ratio ($r_\mathrm{n} = 0.3$) has been used. By doing so, it has been empirically observed that GPRL converges to better solutions faster, compared to a setting with a smaller population and more generations. The latter setting very often converges to suboptimal solutions too early. One reason for that is the lack of diversity observable in smaller populations with a limited number of new individuals in each generation. Furthermore, with the parallel computing options of today's computational resources, big populations can be evaluated in parallel, while many consecutive generations have to be evaluated sequentially.

The GPRL experiments have been conducted on the mountain car (MC), cart-pole balancing (CPB), cart-pole swing-up (CPSU), and industrial benchmark (IB) problems. Table. 8.3 list relevant GPRL and benchmark parameters for the following experiments. The performance in this section is presented as penalty values computed by multiplying the achieved fitness values by -1.

Mountain Car

The MC experiments have been conducted using a time horizon T of 200 and a discount factor γ of 0.985 ($q = 0.05$). For the MC experiments, a non-interpretable PSONN policy with a penalty value of 41.5 has been trained prior to the GP experiments. A policy with this penalty value is capable of driving the car to the top of the hill from every state in the test set. The PSONN policy has one hidden layer with tanh activation function and 20 hidden neurons on each layer. Note that recreating such a policy with function trees as considered for our GPRL approach would result in a complexity value of 321.

The ten GPRL runs learned interpretable policies with median model penalties of 41.6 or lower for complexities five and above (Fig. 8.3a). However, even the policies of complexity value one managed to drive the car to the top of the hill, by merely applying the force along the direction of the car's velocity, i.e., $\pi(\rho, \dot{\rho}) = \dot{\rho}$. Though, policies with lower penalty managed to reach the top of the hill in fewer time steps. The resulting GPRL Pareto front individuals from complexity one to nine are shown in Figure 8.4.

Performing ten symbolic regression runs on the non-interpretable PSONN policy yielded interpretable policies with a median of the regression errors of 0.0067 at best (Fig. 8.3b). This rather low regression error suggests a good performance on imitating the NN policy.

In Fig. 8.3c the performances of GPRL and the GP regression approach are evaluated by testing the policies of their Pareto fronts with different start states on the real MC dynamics. On average GPRL produced the best interpretable policies for all complexities. However, the performance of the symbolic regression approach is quite similar, which suggests that for the MC benchmark such a procedure of creating interpretable RL policies is feasible.

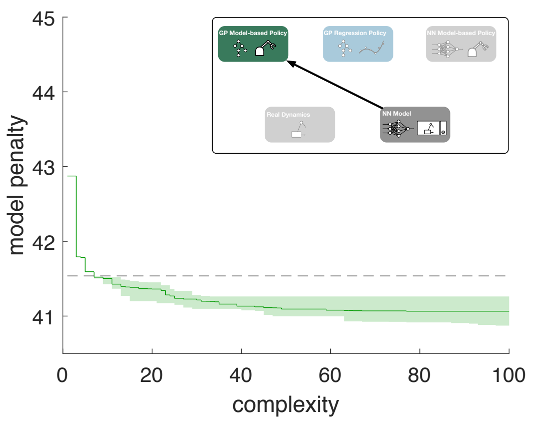
Cart-pole Balancing

The CPB experiments have been conducted using a time horizon T of 100 and a discount factor γ of 0.97 ($q = 0.05$). For the CPB experiments, a non-interpretable PSONN policy with a penalty value of 1.8 has been trained prior to the GP experiments. Based on experience from previous experiments, this performance value represents a successful CPB policy. The PSONN policy has one hidden layer with tanh activation function and 20 hidden neurons on each layer, i.e., complexity 481.

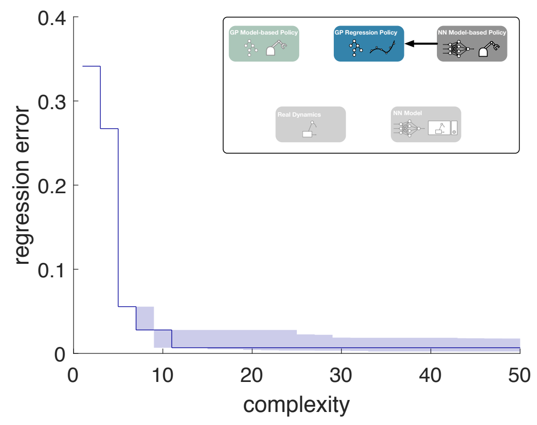
In Fig. 8.5a the results of the GP model-based trainings are depicted. All ten independent GP runs produced Pareto fronts of very similar performance. In comparison to the PSONN policy, individuals with complexity below five performed significantly worse concerning the model penalty. Individuals with complexities eleven and above, on the other hand, yielded a median penalty of 2.1 or below, which corresponds to an excellent CPB policy suitability.

Fig. 8.6 depicts all individuals of the Pareto fronts of the ten experiment runs from complexity 1 to 13. Note how the solutions agree not only on the utilized state variables but also on the floating-point values of the respective factors. Differences often only arise due to the multiplication of the whole terms with different factors, i.e., the ratios between the important state variables remain very similar. Provided with such a policy Pareto chart, experts are more likely to succeed in selecting interpretable policies, since common policy concepts are conveniently identified with respect to both, their complexity as well as their model-based performance.

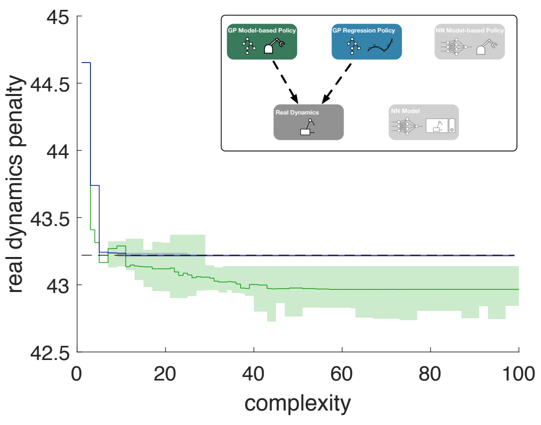
The Pareto front results of the GP regression experiments are presented in Fig. 8.5b. Here, the fitness value driving the GP optimization was the regression error with respect to the PSONN policy. As expected, the individuals of higher complexity achieve lower errors. Note that compared to GP model-based training, the Pareto front results of the ten experiments are spread throughout a bigger area. This fact suggests that the NN policy might be highly non-linear in its outputs, which makes it harder for the GP to



(a) GPRL training



(b) GP regression training



(c) Real system dynamics

Figure 8.3: Pareto fronts from GPRL and GP regression experiments for MC. Depicted are the median (green or blue lines) together with the minimum and maximum (semi-transparent green or blue areas) Pareto fronts penalty from all experiments. The dashed line depicts the performance baseline of the PSONN policy.

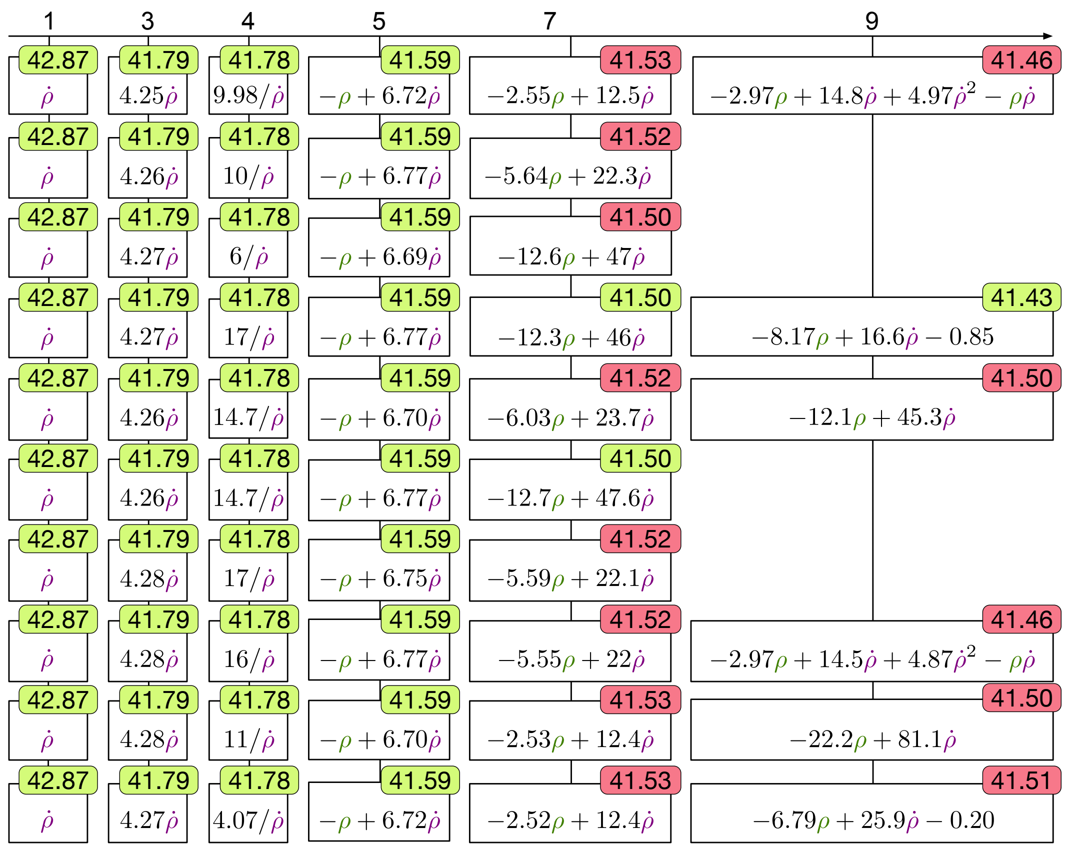
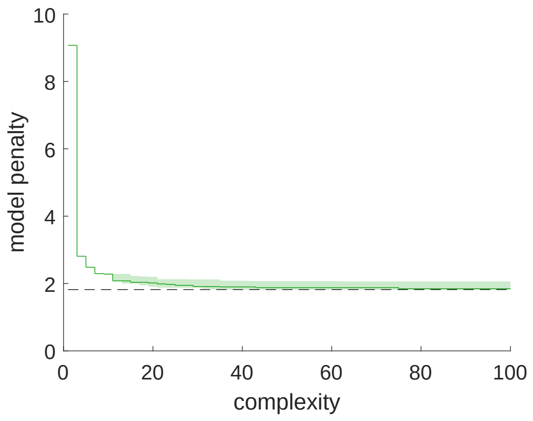
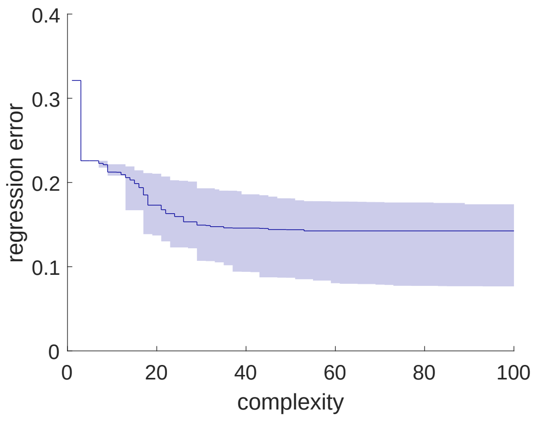

1
3
4
5
7
9
42.87 $\dot\rho$
41.79 $4.25\dot\rho$
41.78 $9.98/\dot\rho$
41.59 $-\rho + 6.72\dot\rho$
41.53 $-2.55\rho + 12.5\dot\rho$
41.46 $-2.97\rho + 14.8\dot\rho + 4.97\dot\rho^2 - \rho\dot\rho$
42.87 $\dot\rho$
41.79 $4.26\dot\rho$
41.78 $10/\dot\rho$
41.59 $-\rho + 6.77\dot\rho$
41.52 $-5.64\rho + 22.3\dot\rho$
42.87 $\dot\rho$
41.79 $4.27\dot\rho$
41.78 $6/\dot\rho$
41.59 $-\rho + 6.69\dot\rho$
41.50 $-12.6\rho + 47\dot\rho$
42.87 $\dot\rho$
41.79 $4.27\dot\rho$
41.78 $17/\dot\rho$
41.59 $-\rho + 6.77\dot\rho$
41.50 $-12.3\rho + 46\dot\rho$
41.43 $-8.17\rho + 16.6\dot\rho - 0.85$
42.87 $\dot\rho$
41.79 $4.26\dot\rho$
41.78 $14.7/\dot\rho$
41.59 $-\rho + 6.70\dot\rho$
41.52 $-6.03\rho + 23.7\dot\rho$
41.50 $-12.1\rho + 45.3\dot\rho$
42.87 $\dot\rho$
41.79 $4.26\dot\rho$
41.78 $14.7/\dot\rho$
41.59 $-\rho + 6.77\dot\rho$
41.50 $-12.7\rho + 47.6\dot\rho$
42.87 $\dot\rho$
41.79 $4.28\dot\rho$
41.78 $17/\dot\rho$
41.59 $-\rho + 6.75\dot\rho$
41.52 $-5.59\rho + 22.1\dot\rho$
42.87 $\dot\rho$
41.79 $4.28\dot\rho$
41.78 $16/\dot\rho$
41.59 $-\rho + 6.77\dot\rho$
41.52 $-5.55\rho + 22\dot\rho$
41.46 $-2.97\rho + 14.5\dot\rho + 4.87\dot\rho^2 - \rho\dot\rho$
42.87 $\dot\rho$
41.79 $4.28\dot\rho$
41.78 $11/\dot\rho$
41.59 $-\rho + 6.70\dot\rho$
41.53 $-2.53\rho + 12.4\dot\rho$
41.50 $-22.2\rho + 81.1\dot\rho$
42.87 $\dot\rho$
41.79 $4.27\dot\rho$
41.78 $4.07/\dot\rho$
41.59 $-\rho + 6.72\dot\rho$
41.53 $-2.52\rho + 12.4\dot\rho$
41.51 $-6.79\rho + 25.9\dot\rho - 0.20$

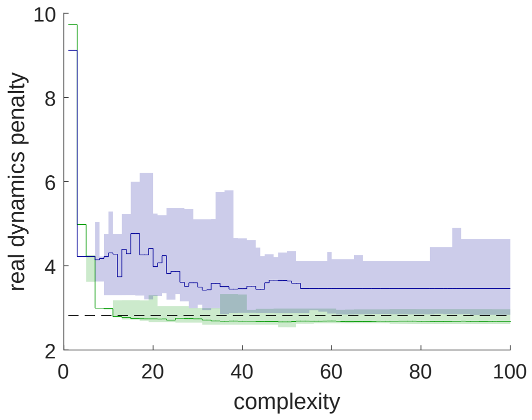
Figure 8.4: Interpretable MC policy results by GPRL for complexities 1-9. Green penalty tags highlight the best solutions for a certain complexity value.



(a) GPRL training

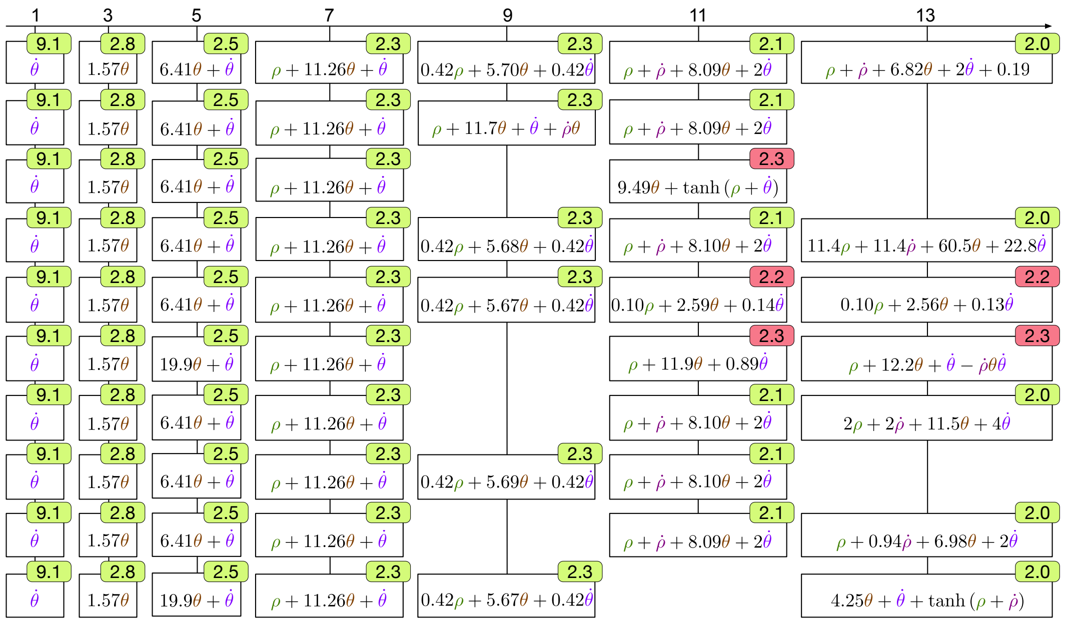


(b) GP regression training



(c) Real system dynamics

Figure 8.5: Pareto fronts from GPRL and GP regression experiments for CPB



1
3
5
7
9
11
13
9.1 $\dot{\theta}$
2.8 1.57θ
2.5 $6.41\theta + \dot{\theta}$
2.5 $19.9\theta + \dot{\theta}$
2.3 $\rho + 11.26\theta + \dot{\theta}$
2.3 $0.42\rho + 5.70\theta + 0.42\dot{\theta}$
2.3 $\rho + 11.7\theta + \dot{\theta} + \dot{\rho}\theta$
2.3 $0.42\rho + 5.68\theta + 0.42\dot{\theta}$
2.3 $0.42\rho + 5.67\theta + 0.42\dot{\theta}$
2.3 $0.42\rho + 5.69\theta + 0.42\dot{\theta}$
2.3 $0.42\rho + 5.67\theta + 0.42\dot{\theta}$
2.1 $\rho + \dot{\rho} + 8.09\theta + 2\dot{\theta}$
2.1 $\rho + \dot{\rho} + 8.09\theta + 2\dot{\theta}$
2.3 $9.49\theta + \tanh(\rho + \dot{\theta})$
2.1 $\rho + \dot{\rho} + 8.10\theta + 2\dot{\theta}$
2.2 $0.10\rho + 2.59\theta + 0.14\dot{\theta}$
2.3 $\rho + 11.9\theta + 0.89\dot{\theta}$
2.1 $\rho + \dot{\rho} + 8.10\theta + 2\dot{\theta}$
2.1 $\rho + \dot{\rho} + 8.10\theta + 2\dot{\theta}$
2.1 $\rho + \dot{\rho} + 8.09\theta + 2\dot{\theta}$
2.0 $\rho + \dot{\rho} + 6.82\theta + 2\dot{\theta} + 0.19$
2.0 $11.4\rho + 11.4\dot{\rho} + 60.5\theta + 22.8\dot{\theta}$
2.2 $0.10\rho + 2.56\theta + 0.13\dot{\theta}$
2.3 $\rho + 12.2\theta + \dot{\theta} - \dot{\rho}\theta\dot{\theta}$
2.0 $2\rho + 2\dot{\rho} + 11.5\theta + 4\dot{\theta}$
2.0 $\rho + 0.94\dot{\rho} + 6.98\theta + 2\dot{\theta}$
2.0 $4.25\theta + \dot{\theta} + \tanh(\rho + \dot{\rho})$

Figure 8.6: Interpretable CPB policy results by GPRL for complexities 1-13

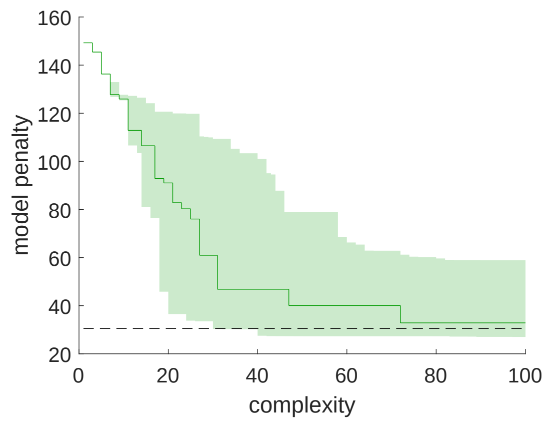
find solutions in the vast search space of algebraic equations.

To evaluate the true performance of the two approaches GP model-based training and GP regression training, the individuals of both sets of Pareto fronts have been tested on the true CPB dynamics. Fig. 8.5c shows the resulting *altered* Pareto fronts compared to the performance of the PSONN policy. It is evident that almost for every complexity the GP model-based approach GPRL is superior to the GP regression idea. Not only are the median results of significantly lower penalty, but the variance of the GPRL solution is also much lower compared to the GP regression result. Interestingly, the median GP results for complexity 11 and above even outperformed the PSONN policy result. This indicates that the NN policy already over-fitted the surrogate model and exploited its inaccuracies, while the simple GP policy equations generalize better because of their somewhat restricted structure.

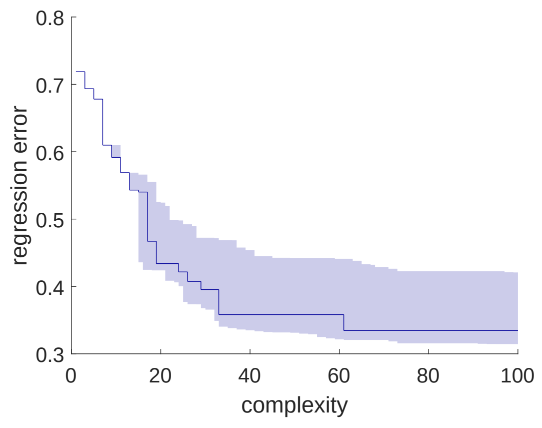
Cart-pole Swing-up

The CPSU experiments have been conducted using a time horizon T of 500 and a discount factor γ of 0.994 ($q = 0.05$). For the CPSU experiments, a non-interpretable PSONN policy with a penalty value of 30.5 has been trained prior to the GP experiments. This performance value represents a very successful CPSU policy. However, even policies yielding a penalty of 50 are considered to be successful CPSU controllers. The PSONN policy has one hidden layer with tanh activation function and 20 hidden neurons on each layer, i.e., complexity 481.

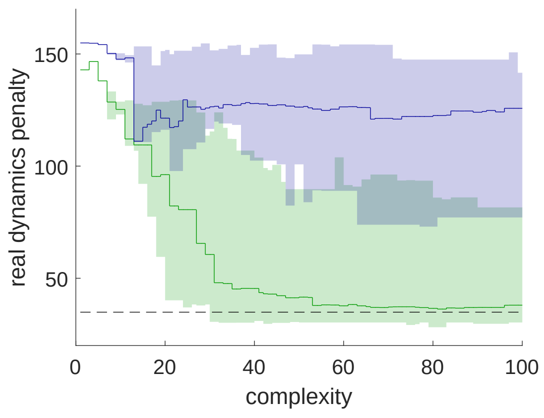
Fig. 8.7 compares the learned GPRL policies by their performance on model and real



(a) GPRL training



(b) GP regression training



(c) Real system dynamics

Figure 8.7: Pareto fronts from GPRL and GP regression experiments for CPSU

system dynamics. In comparison with results for MC and CPB, the generated Pareto fronts are widely spread for the majority of complexity levels. For instance, after 10 000 GP iterations one GPRL run found a policy with complexity 20 and the excellent penalty value of 36.5 whereas another GPRL run failed on finding a solution with penalty below 60 even for complicated policies of complexity level 100. Nevertheless, the median resulting Pareto front contains acceptable policies with penalty values below 50 for complexity 31 or more. In addition to that, we can see that from complexity level 30 on solutions could be found which outperform the best PSONN policy. This result is somewhat surprising since the PSONN policy with complexity level 481 is expected to apply significantly more degrees of freedom which can be tuned to perform best for the given task.

Fig. 8.8 visualizes all policies from the ten Pareto fronts between complexities 16 and 20. In contrast to the results of MC and CPB, the depicted complexity levels are higher;

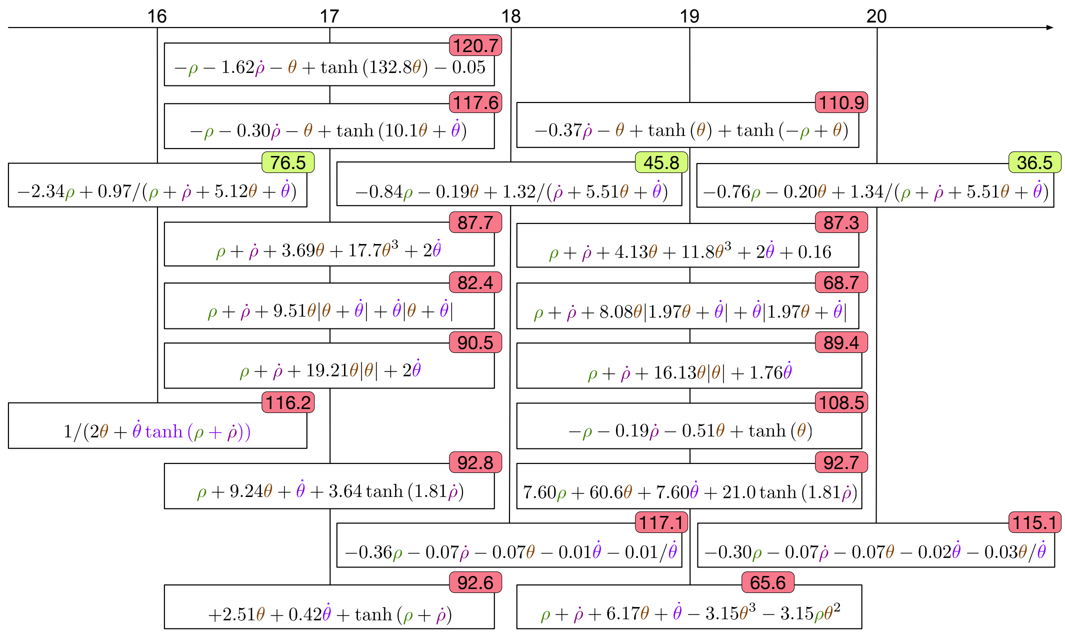

16
17
18
19
20
120.7
$-\rho - 1.62\dot{\rho} - \theta + \tanh(132.8\theta) - 0.05$
117.6
$-\rho - 0.30\dot{\rho} - \theta + \tanh(10.1\theta + \dot{\theta})$
110.9
$-0.37\dot{\rho} - \theta + \tanh(\theta) + \tanh(-\rho + \theta)$
76.5
$-2.34\rho + 0.97/(\rho + \dot{\rho} + 5.12\theta + \dot{\theta})$
45.8
$-0.84\rho - 0.19\theta + 1.32/(\dot{\rho} + 5.51\theta + \dot{\theta})$
36.5
$-0.76\rho - 0.20\theta + 1.34/(\rho + \dot{\rho} + 5.51\theta + \dot{\theta})$
87.7
$\rho + \dot{\rho} + 3.69\theta + 17.7\theta^3 + 2\dot{\theta}$
87.3
$\rho + \dot{\rho} + 4.13\theta + 11.8\theta^3 + 2\dot{\theta} + 0.16$
82.4
$\rho + \dot{\rho} + 9.51\theta|\theta + \dot{\theta}| + \dot{\theta}|\theta + \dot{\theta}|$
68.7
$\rho + \dot{\rho} + 8.08\theta|1.97\theta + \dot{\theta}| + \dot{\theta}|1.97\theta + \dot{\theta}|$
90.5
$\rho + \dot{\rho} + 19.21\theta|\theta| + 2\dot{\theta}$
89.4
$\rho + \dot{\rho} + 16.13\theta|\theta| + 1.76\dot{\theta}$
116.2
$1/(2\theta + \dot{\theta}\tanh(\rho + \dot{\rho}))$
108.5
$-\rho - 0.19\dot{\rho} - 0.51\theta + \tanh(\theta)$
92.8
$\rho + 9.24\theta + \dot{\theta} + 3.64\tanh(1.81\dot{\rho})$
92.7
$7.60\rho + 60.6\theta + 7.60\dot{\theta} + 21.0\tanh(1.81\dot{\rho})$
117.1
$-0.36\rho - 0.07\dot{\rho} - 0.07\theta - 0.01\dot{\theta} - 0.01/\dot{\theta}$
115.1
$-0.30\rho - 0.07\dot{\rho} - 0.07\theta - 0.02\dot{\theta} - 0.03\theta/\dot{\theta}$
92.6
$+2.51\theta + 0.42\dot{\theta} + \tanh(\rho + \dot{\rho})$
65.6
$\rho + \dot{\rho} + 6.17\theta + \dot{\theta} - 3.15\theta^3 - 3.15\rho\theta^2$

Figure 8.8: Interpretable CPSU policy results by our GPRL approach for complexities 16-20

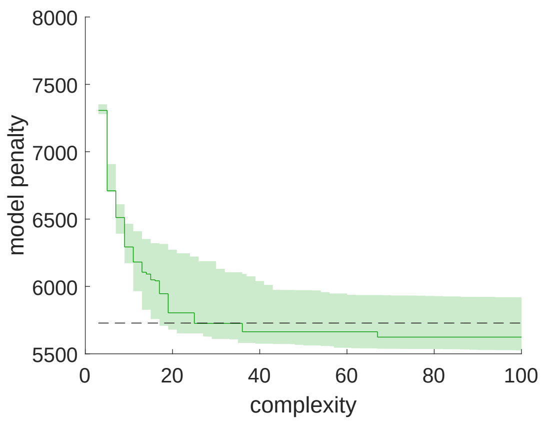
hence the resulting equations are less homogeneous. For this complexity range, only one out of ten individual GPRL runs found well-performing policies. However, this result shows that even for a challenging problem like CPSU interpretable policies exist and, given sufficient computational resources, these solutions can be found by GPRL.

The results for the GP regression training are not as promising as the GPRL approach. Fig. 8.7b shows that despite the resulting solutions yield lower regression errors with higher complexity, no successful policy could be found. Whereas policies of lower complexity seem to produce similar penalties as the GPRL results, from complexity 15 on the resulting policies start to perform significantly worse (Fig. 8.7c). Reducing the regression error seems not to improve the resulting performance during the evaluation phase.

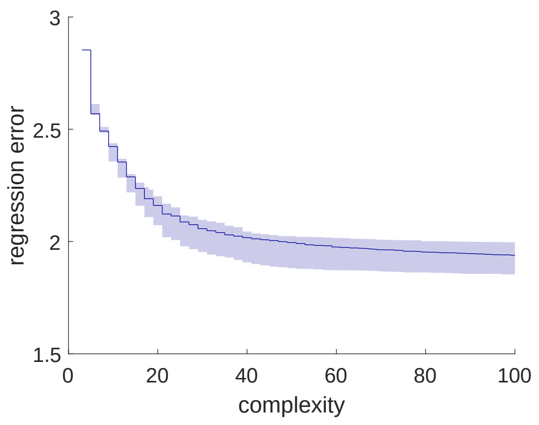
Industrial Benchmark

The IB experiments have been conducted using a time horizon T of 100 and a discount factor γ of 0.97 ($q = 0.05$). For the IB experiments, a non-interpretable PSONN policy with a penalty value of 5 728 has been trained prior to the GP experiments. The PSONN policy consists of three separate NNs with one hidden layer, tanh activation functions, and 20 hidden neurons on each layer, i.e., complexity value 31 803.

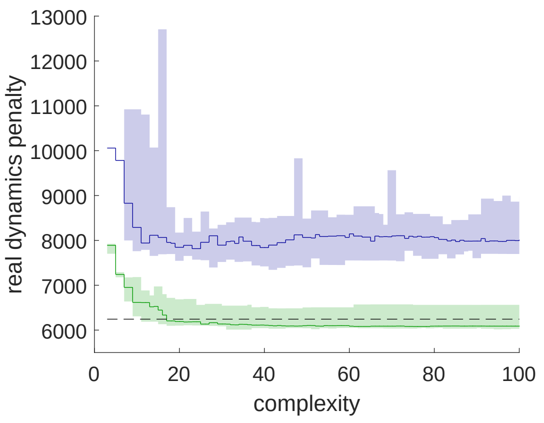
The results of ten individual GPRL runs are depicted in Fig. 8.9a. Note that the median GPRL Pareto front results outperformed the non-interpretable PSONN policy from complexity 36 on. This suggests that even in the best out of ten PSONN trainings,



(a) GPRL training



(b) GP regression training



(c) Real system dynamics

Figure 8.9: Pareto fronts from GPRL and GP regression experiments for IB

the PSO swarm got stuck in a suboptimal solution, which is likely to happen given that with PSONN thousands of network weights have to be optimized with respect to each other.

The resulting GPRL policies between complexities 21 and 29 are presented in Fig. 8.10. Note that independently learned policies share similar concepts of how policy actions are computed from related state variables with similar time lags. For example, the equations for Δh always use a linear combination of shift value h from time lags -2, -3, or -4 and the constant setpoint value p. Another example is the computation of Δv, for which a velocity value v with time lags of ten and above is always used. Moreover, it is possible to reveal common concepts and relevant differences between a rich set of possible solutions. This presentation could provide domain experts with important insights on how well-performing policies for the system at hand look like, on which state variables they react, and how they generalize in state space areas where currently insufficient training data is available.

The results of applying GP symbolic regression on a non-interpretable PSONN policy are shown in Figure 8.9b. For each policy action, an independent GP run has been conducted. After the training, multi-dimensional policies have been created in such a way that the accumulated regression errors of Δv, Δg, and Δs are as low as possible for every possible complexity value. This procedure has been repeated ten times to yield ten independent IB policies.

In the final step of the experiment, the GPRL and the GP regression solutions have been evaluated on the real IB dynamics. Comparing the two approaches in Fig. 8.9c reveals the strengths of the GPRL approach. First, the model-based GP training performs significantly better than the symbolic regression training. Despite the fact that even in the MC, CPB, and CPSU experiments, GPRL outperformed the regression approach, the experiments with IB illustrate the complete performance breakdown of the latter. Second, the excellent generalization properties of GPRL led to interpretable policies which even outperformed a non-interpretable PSONN policy from complexity 16 on.

8.3 Discussion

The introduced GPRL approach conducts model-based batch RL to learn interpretable policies for control problems from already existing system trajectories. The policies can be represented by compact algebraic equations or Boolean logic terms. Autonomous learning of such interpretable policies is of high interest for industry domain experts. Presented with a number of GPRL results for a preferred range of complexity, new concepts for controlling an industrial plant can be revealed. Moreover, safety concerns can be addressed more easily, if the policy at hand itself, as well as its generalization to certain state space areas, are entirely understandable.

The complete GPRL procedure of (i) training a model from existing system trajectories, (ii) learning interpretable policies by GP, (iii) selecting a favorable solution candidate from a Pareto front result has been evaluated on four RL benchmarks, i.e., MC, CPSU, CPB, and IB. First, the control performance was compared to a non-interpretable PSONN policy. This comparison showed that the GPRL performance on the approximation model could be slightly worse compared to the PSONN policy result. However, when evaluated on the real system dynamics, even interpretable policies of rather low complexity could outperform the non-interpretable approach in many occasions. This suggests that simple algebraic equations used as policies generalize better on new system states. In a second evaluation, the GPRL approach has been compared to a straightforward GP utilization as a symbolic regression tool, i.e., fitting the existing non-interpretable PSONN policy by GP to yield interpretable policies of similar control performance. All of our experiments showed that this strategy is significantly less suitable to produce policies of adequate performance.

Especially the experiments with the IB indicated that the application of the proposed GPRL approach in industry settings could prove to be of significant interest. In many

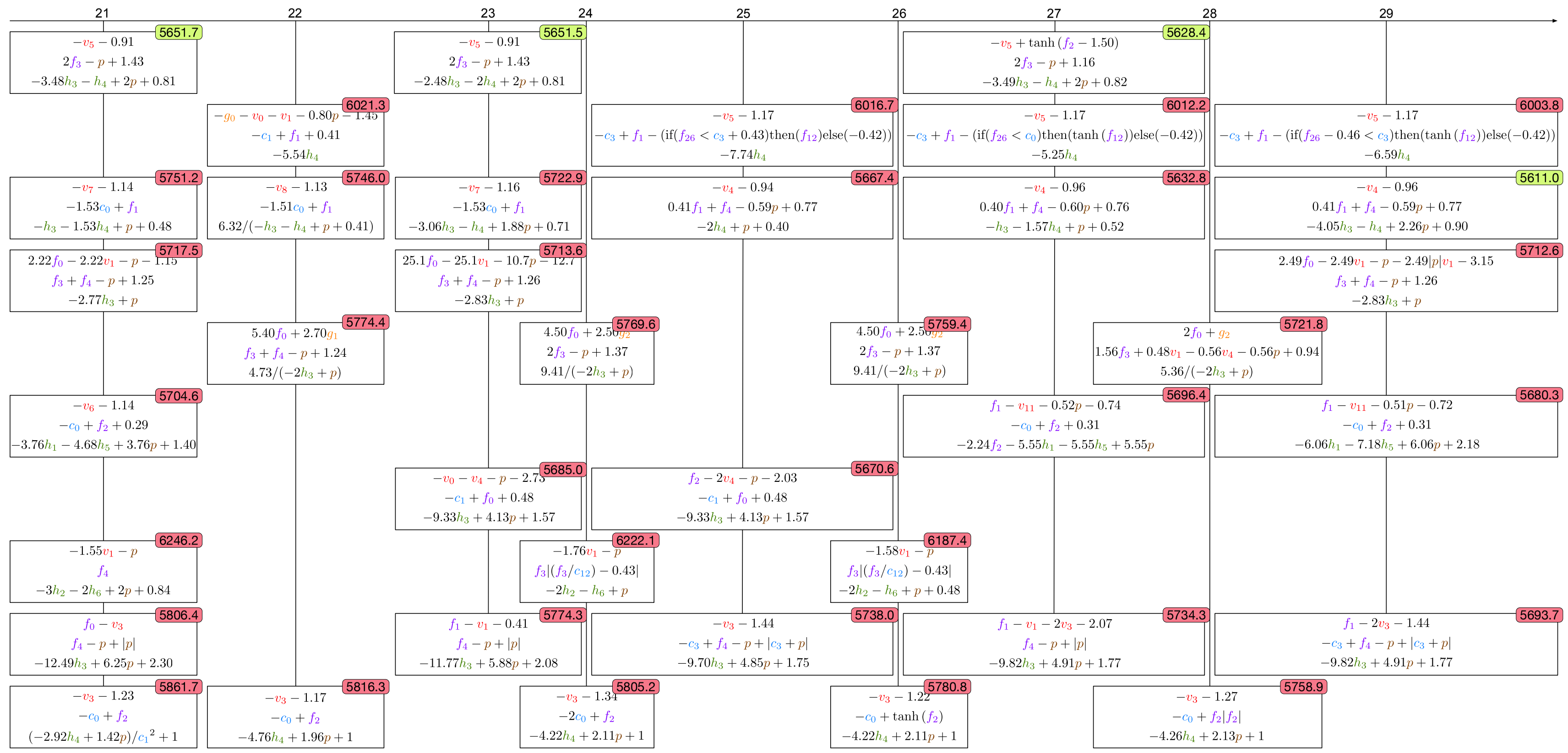

21
22
23
24
25
26
27
28
29
5651.7 $-v_5 - 0.91$; $2f_3 - p + 1.43$; $-3.48h_3 - h_4 + 2p + 0.81$
5751.2 $-v_7 - 1.14$; $-1.53c_0 + f_1$; $-h_3 - 1.53h_4 + p + 0.48$
5717.5 $2.22f_0 - 2.22v_1 - p - 1.15$; $f_3 + f_4 - p + 1.25$; $-2.77h_3 + p$
5704.6 $-v_6 - 1.14$; $-c_0 + f_2 + 0.29$; $-3.76h_1 - 4.68h_5 + 3.76p + 1.40$
6246.2 $-1.55v_1 - p$; f_4; $-3h_2 - 2h_6 + 2p + 0.84$
5806.4 $f_0 - v_3$; $f_4 - p + |p|$; $-12.49h_3 + 6.25p + 2.30$
5861.7 $-v_3 - 1.23$; $-c_0 + f_2$; $(-2.92h_4 + 1.42p)/c_1{}^2 + 1$
6021.3 $-g_0 - v_0 - v_1 - 0.80p - 1.45$; $-c_1 + f_1 + 0.41$; $-5.54h_4$
5746.0 $-v_8 - 1.13$; $-1.51c_0 + f_1$; $6.32/(-h_3 - h_4 + p + 0.41)$
5774.4 $5.40f_0 + 2.70g_1$; $f_3 + f_4 - p + 1.24$; $4.73/(-2h_3 + p)$
5816.3 $-v_3 - 1.17$; $-c_0 + f_2$; $-4.76h_4 + 1.96p + 1$
5651.5 $-v_5 - 0.91$; $2f_3 - p + 1.43$; $-2.48h_3 - 2h_4 + 2p + 0.81$
5722.9 $-v_7 - 1.16$; $-1.53c_0 + f_1$; $-3.06h_3 - h_4 + 1.88p + 0.71$
5713.6 $25.1f_0 - 25.1v_1 - 10.7p - 12.7$; $f_3 + f_4 - p + 1.26$; $-2.83h_3 + p$
5769.6 $4.50f_0 + 2.50g_2$; $2f_3 - p + 1.37$; $9.41/(-2h_3 + p)$
5685.0 $-v_0 - v_4 - p - 2.73$; $-c_1 + f_0 + 0.48$; $-9.33h_3 + 4.13p + 1.57$
6222.1 $-1.76v_1 - p$; $f_3|(f_3/c_{12}) - 0.43|$; $-2h_2 - h_6 + p$
5774.3 $f_1 - v_1 - 0.41$; $f_4 - p + |p|$; $-11.77h_3 + 5.88p + 2.08$
5805.2 $-v_3 - 1.34$; $-2c_0 + f_2$; $-4.22h_4 + 2.11p + 1$
6016.7 $-v_5 - 1.17$; $-c_3 + f_1 - (\text{if}(f_{26} < c_3 + 0.43)\text{then}(f_{12})\text{else}(-0.42))$; $-7.74h_4$
5667.4 $-v_4 - 0.94$; $0.41f_1 + f_4 - 0.59p + 0.77$; $-2h_4 + p + 0.40$
5670.6 $f_2 - 2v_4 - p - 2.03$; $-c_1 + f_0 + 0.48$; $-9.33h_3 + 4.13p + 1.57$
5738.0 $-v_3 - 1.44$; $-c_3 + f_4 - p + |c_3 + p|$; $-9.70h_3 + 4.85p + 1.75$
5759.4 $4.50f_0 + 2.50g_2$; $2f_3 - p + 1.37$; $9.41/(-2h_3 + p)$
6187.4 $-1.58v_1 - p$; $f_3|(f_3/c_{12}) - 0.43|$; $-2h_2 - h_6 + p + 0.48$
5780.8 $-v_3 - 1.22$; $-c_0 + \tanh(f_2)$; $-4.22h_4 + 2.11p + 1$
5628.4 $-v_5 + \tanh(f_2 - 1.50)$; $2f_3 - p + 1.16$; $-3.49h_3 - h_4 + 2p + 0.82$
6012.2 $-v_5 - 1.17$; $-c_3 + f_1 - (\text{if}(f_{26} < c_0)\text{then}(\tanh(f_{12}))\text{else}(-0.42))$; $-5.25h_4$
5632.8 $-v_4 - 0.96$; $0.40f_1 + f_4 - 0.60p + 0.76$; $-h_3 - 1.57h_4 + p + 0.52$
5696.4 $f_1 - v_{11} - 0.52p - 0.74$; $-c_0 + f_2 + 0.31$; $-2.24f_2 - 5.55h_1 - 5.55h_5 + 5.55p$
5734.3 $f_1 - v_1 - 2v_3 - 2.07$; $f_4 - p + |p|$; $-9.82h_3 + 4.91p + 1.77$
5721.8 $2f_0 + g_2$; $1.56f_3 + 0.48v_1 - 0.56v_4 - 0.56p + 0.94$; $5.36/(-2h_3 + p)$
5758.9 $-v_3 - 1.27$; $-c_0 + f_2|f_2|$; $-4.26h_4 + 2.13p + 1$
6003.8 $-v_5 - 1.17$; $-c_3 + f_1 - (\text{if}(f_{26} - 0.46 < c_3)\text{then}(\tanh(f_{12}))\text{else}(-0.42))$; $-6.59h_4$
5611.0 $-v_4 - 0.96$; $0.41f_1 + f_4 - 0.59p + 0.77$; $-4.05h_3 - h_4 + 2.26p + 0.90$
5712.6 $2.49f_0 - 2.49v_1 - p - 2.49|p|v_1 - 3.15$; $f_3 + f_4 - p + 1.26$; $-2.83h_3 + p$
5680.3 $f_1 - v_{11} - 0.51p - 0.72$; $-c_0 + f_2 + 0.31$; $-6.06h_1 - 7.18h_5 + 6.06p + 2.18$
5693.7 $f_1 - 2v_3 - 1.44$; $-c_3 + f_4 - p + |c_3 + p|$; $-9.82h_3 + 4.91p + 1.77$

Figure 8.10: Interpretable IB policies for complexities 21-29. The presented 3-dimensional policies are the result of ten independent GPRL trainings. Each box contains three policy equations for Δv_0,Δg_0, and Δh_0 (from top to bottom) to calculate actions for time step $t = 0$. The input variables' indices represent the respective negative time lag in which they have been recorded, e.g., h_3 represents the value of shift three time steps ago at $t = -3$. The actions are limited to -1 and +1 before they are applied to the system dynamics. The input variables (p, v, g, h, f, c) are normalized by subtracting their respective mean $(55.0, 48.75, 50.53, 49.45, 37.51, 166.33)$ and dividing by their respective standard deviation $(28.72, 12.31, 29.91, 29.22, 31.17, 139.44)$. These values can easily be calculated from the training data set.

cases, data from systems are readily available and interpretable simple algebraic policies are favored over black-box RL solutions, such as non-interpretable PSONN policies.

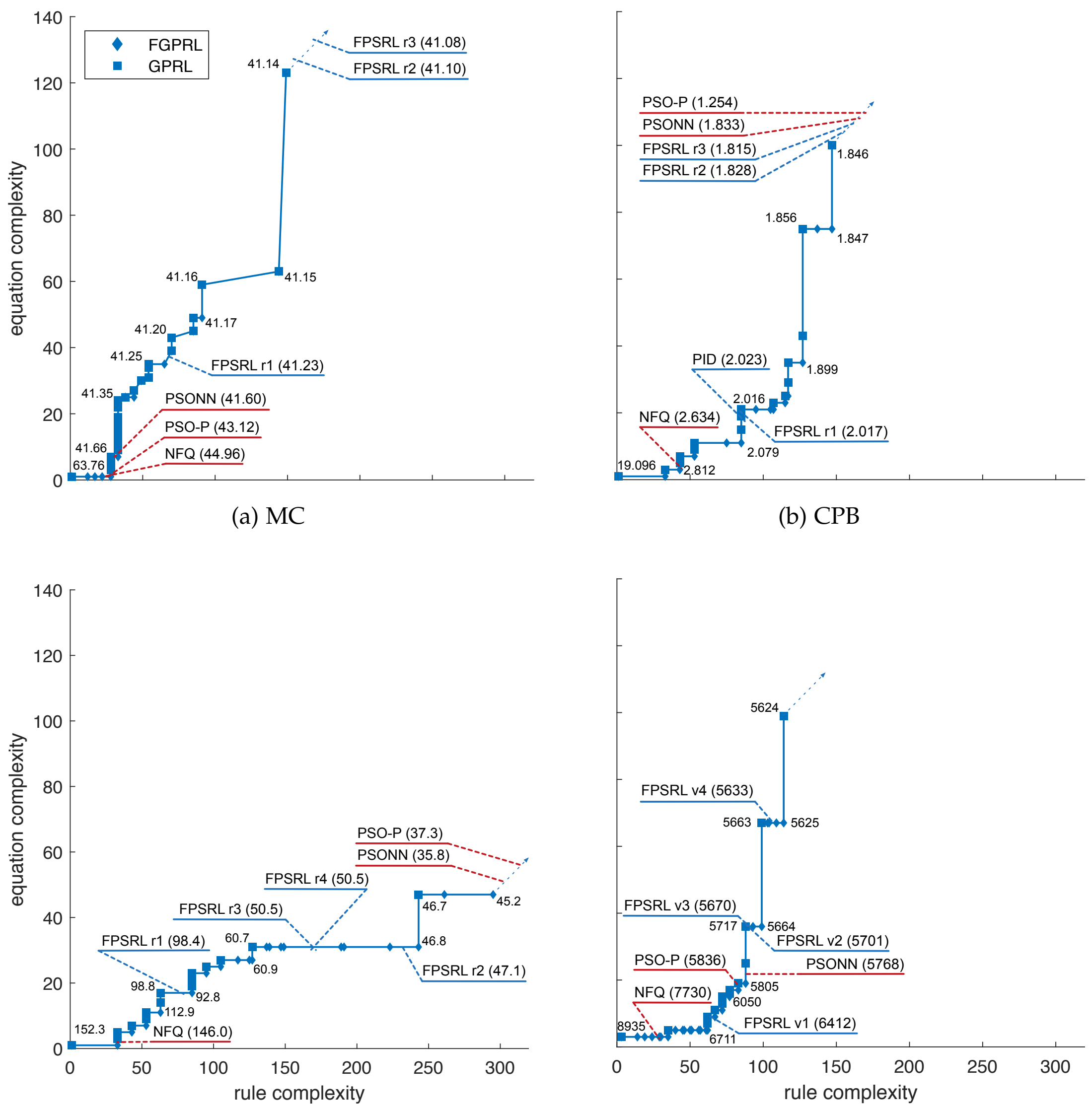
CHAPTER 9

Comparison & Discussion

In this section, the results of all the applied reinforcement learning (RL) methods are compared with respect to their control performance as well as the interpretability of the policies produced. While comparing the methods performance-wise is relatively straightforward, deciding whether rule-based or equation-based solutions are more interpretable remains highly subjective. Note that in this thesis such interpretable RL approaches were investigated by functionally-grounded methods (Section 1.1). This means that for solution classes which have already been evaluated as interpretable, the performance of the solutions is improved, or the sparsity is increased.

In Table 9.1 and Fig. 9.2-9.5 the methods are compared by their performance in terms of increasing the return which is equivalent to reducing the penalty. To yield a fair comparison, all methods started with the same set of recorded state transitions from each of the benchmarks. Fig. 9.1 gives a visualization on how the different methods can be functionally-grounded evaluated based on the achieved benchmark penalties as well as their complexity measures. Since every RL approach, even the model-free neural fitted Q iteration (NFQ) method, employed surrogate models, these models have been trained once before the experiments and re-used during all experiments. Consequently, every RL approach had to deal with the same inaccuracies and miss-predictions. Moreover, training and evaluation always took place on the same sets of randomly drawn training and evaluation states, to eliminate state-dependent effects which cause different penalty values.

Proportional–integral–derivative (**PID**) controllers are only applicable to one of the four benchmarks, namely cart-pole balancing (CPB) (Chapter 4). Using a manual tuning heuristic like Ziegler-Nichols method and integrating prior expert knowledge about the optimal setpoints for angel and position, resulted in well-performing interpretable control strategies. However, replacing the manual tuning process by particle swarm optimization (PSO) significantly reduces the construction time effort and ensures finding the



(a) MC (b) CPB

(c) CPSU (d) IB

Figure 9.1: Functionally-grounded interpretability evaluation based on complexity measures and achieved penalties. Individuals of the median Pareto fronts from FGPRL and GPRL are compared by their performance (blue line) from high (lower left) to low penalty (upper right). The median results of all other RL methods (penalty values in brackets) are placed with respect to their performance on the penalty graph. The FPSRL notations _v and _r represent the utilized numbers of state features and rules, respectively.

		PID	NFQ (last)	NFQ (selected)	PSO-P	PSONN	FPSRL	FGPRL	GPRL
Model	MC	-	61.4 ± 0.6	45.4 ± 0.8	43.04 ± 0.07	41.594 ± 0.007	41.078 ± 0.013	41.15 ± 0.03	41.06 ± 0.03
	CPB	2.0285 ± 0.0003	15.0 ± 1.0	2.71 ± 0.04	1.2535 ± 0.0014	1.833 ± 0.002	1.819 ± 0.005	1.814 ± 0.003	1.88 ± 0.02
	CPSU	-	156.5 ± 0.2	146.1 ± 1.3	37.1 ± 0.2	36.2 ± 1.1	40 ± 2	45 ± 2	39 ± 4
	IB	-	8190 ± 80	7700 ± 20	5835 ± 6	5768 ± 7	5633 ± 3	5580 ± 10	5660 ± 50
System	MC	-	61.0 ± 0.7	44.7 ± 0.9	43.1 ± 0.1	43.31 ± 0.03	42.87 ± 0.13	42.97 ± 0.04	42.97 ± 0.04
	CPB	3.0376 ± 0.0008	15.5 ± 0.9	5.4 ± 0.2	4.20 ± 0.04	2.831 ± 0.003	2.69 ± 0.03	2.798 ± 0.006	2.71 ± 0.03
	CPSU	-	156.1 ± 0.3	150.8 ± 1.3	37.81 ± 0.13	40.1 ± 1.3	44 ± 3	52 ± 4	49 ± 7
	IB	-	7610 ± 110	7050 ± 40	6029 ± 4	6280 ± 20	6134 ± 11	6030 ± 10	6150 ± 50

Table 9.1: Penalty comparison of all evaluated RL methods. Depicted are the mean penalties computed from ten independent experiments for each benchmark setting. The number after $\pm$ represents the standard error.

true global optimum. The best PID controller using Ziegler-Nichols method computes its action for time step t as follows:

$$\begin{aligned} a(t) =& 0.95 \left(-8.6 \cdot e_\theta(t) - 0.15 \cdot \int_{t-40}^{t} e_\theta(t) dt - 120.9 \cdot \frac{de_\theta(t)}{dt} \right) \\ &+ 0.05 \left(-25.6 \cdot e_\rho(t) - 0.15 \cdot \int_{t-28}^{t} e_\rho(t) dt - 1061.4 \cdot \frac{de_\rho(t)}{dt} \right), \end{aligned} \tag{9.1}$$

where $e_\theta(t)$ and $e_\rho(t)$ represent the error with respect to the setpoints $\theta = 0$ and $\rho = 0$, respectively. Note that mountain car (MC), cart-pole swing-up (CPSU), and industrial benchmark (IB) problems could not be solved by applying PID control.

NFQ is the only model-free RL method evaluated in this work (Chapter 5.1). NFQ was chosen because it is a well-established, widely applied, and well-documented RL methodology. It was selected to show both the degree of difficulty of our benchmarks and the challenges arising from batch-based RL. Penalty-wise NFQ produced the worst policies for all of the benchmarks (Table 9.1 and Fig. 9.1). Even for the two-dimensional MC benchmark (Fig. 9.2) stable policy training could not be observed. The method lacks a meaningful metric which would suggest when to stop the iterative process. Using surrogate models first to predict the performance of the policy of each iteration and subsequently select the policy with the highest model-predicted performance, improved the quality significantly for every single benchmark (compare NFQ (last) with NFQ (selected) in Table 9.1). The results from MC (Fig. 9.2) and cart-pole balancing (CPB) (Fig. 9.3) prove that the method is occasionally able to learn adequate policies for these benchmarks. However, it is not able to converge stably to these solutions. On the contrary, the best policies very often were results from early NFQ iterations and the algorithm never returned to these superior solutions. For CPSU (Fig. 9.4) and IB (Fig. 9.5) NFQ failed completely in finding good policies. Although selecting the best results using surrogate models still improved the performance on average, even though the selected solutions were still of relatively poor control quality. In conclusion, it can be said, in our batch-based RL setting NFQ produced implicit and non-interpretable policies of poor control performance, even if world models were used for policy selection.

The second approach evaluated here is the particle swarm optimization policy (**PSO-P**) method (Chapter 5.2). Given an appropriate world model, this model predictive control (MPC) approach does not require any policy training, since it exploits the model by testing action trajectories during policy runtime. Note that no policy representation has to be predefined, which eliminates the difficulties arising from selecting inadequate policy representations or insufficient degrees of freedom with respect to the problem complexity. The experiments on all four benchmarks yielded excellent penalty results. PSO-P's computational effort during runtime generally makes it less feasible for application on real-time tasks. Nevertheless, employing PSO-P-generated actions on the world models, instead of the real system dynamics, can be of great use in verifying the respective models. Exploiting a model for optimal trajectories reveals critical knowledge about how the model works and how optimal trajectories look like. For industrial applications, experts can check the predicted trajectories and evaluate whether the models simulate realistic behavior. Moreover, the achieved penalty values of PSO-P-generated action trajectories are useful as baselines for methods which are trained on the same world models subsequently. For the considered benchmarks, it has been shown that, for the surrogate models, control strategies exist that produce the desired trajectories and are rewarded with low penalty values. However, action trajectories produced by PSO-P are still implicit and non-interpretable.

Particle swarm optimization neural network (**PSONN**) is the first approach considered in this work that creates explicit RL policies (Chapter 5.3). In contrast to implicit policies, explicit policies do not require searching for an action which maximizes a value function (as with NFQ) or to perform trajectory optimizations on a system model during runtime (as with PSO-P). Once an explicit policy has been learned, it is sufficient to deploy only this result on the system. The PSONN approach, as applied here, is a combination of both ideas; training neural network (NN) weights by particle swarm optimization (PSO) and using model-based RL with population-based policy search in order to yield policies of high performance. The experiments on all four benchmarks yielded excellent performance results. Even for the CPSU benchmark (Fig. 9.4), every resulting policy represents a successful control strategy. Due to PSONN's high number of degrees of freedom, which arise from the fully connected NN topology and the vast amount of free parameters, the policies sometimes tend to overfit to the world model during training. This effect can cause a decline in performance regarding generalization, which results in a decreased policy performance during evaluation on the real system dynamics. We can observe this happening with the IB. Fig. 9.5 shows for PSONN that all of the resulting ten policies perform very good on the world model but, evaluated on the real system dynamics, their performance varies substantially. However, even the PSONN policy which performs worst on the real dynamics, still yields an acceptable system controller compared to the best NFQ results, which also represents its policies by NNs. Achieving good policies by PSONN validates the assumption that population-based policy search using rollouts on a system model is a promising approach for generating policies for batch-based RL tasks. In the following approaches, the non-interpretable NN is replaced by interpretable policy representations, and their results can be compared to the PSONN performance to quantify the impact of these representations on the penalty values achieved.

Fuzzy particle swarm reinforcement learning (**FPSRL**) is the first interpretable policy approach in this work. Conceptually, the non-interpretable NNs from the PSONN approach are replaced by interpretable rule-based controllers. Therefore, PSO now optimizes the parameters of these sets of rules instead of some network weights. For FPSRL, decisions concerning the number of rules in the policy and the incorporated state features have to made before the training. For toy benchmarks like MC, CPB, and CPSU it is already known that all of the state features are required to yield optimal policies, which only leaves the question of how many rules are required. A simple brute-force-like approach would be to start with a minimum number of rules and repeat the training with an increased number of rules until a sufficient performance is obtained. Good performance values can be found by applying PSO-P or PSONN on the same surrogate models before FPSRL training. The plots in Fig. 9.2 and 9.3 show that for MC and CPB problems the minimum number of rules (two rules yielding complexity values 43 and 63, respectively) already produces very low penalty values. Increasing the number of rules to four and six did not substantially increase the policy performance. The FPSRL results of the CPSU experiments depicted in Fig. 9.4 show that learning policies with only two rules is not sufficient for yielding an acceptable CPSU control performance. However, evaluating the required number of rules is not the only open question before running FPSRL. For high-dimensional problems like IB, informative state features have to be revealed first. Otherwise, a single rule for the IB would have to be defined over 180 state features, which not only would make the training intractable but, in addition to that, such huge rules would result in highly non-interpretable policies. In Chapter 6, the application of a combination of PSO-P and a mutual-information-based feature selection heuristic which can help to identify important state features is investigated. The FPSRL results depicted in Fig. 9.5 show the performance of policies which incorporated one, two, three, or four of the most informative features in two rules per action dimension. The plots show that using only the most important state feature does not yield good performance values, whereas using two or more features substantially decreases the penalty. However, here we tested only a small subset of the existing combinations of state features and number of rules per action dimension. The fact that an exhaustive search through all of these combinations is often infeasible due to limited computational resources is a significant disadvantage for applying FPSRL in real industry scenarios. Nevertheless, FPSRL can learn interpretable rule-based RL policies that can successfully control all of the tested benchmarks.

The second interpretable policy approach presented in this work is fuzzy genetic programming reinforcement learning (**FGPRL**). Similarly to FPSRL, this approach automatically learns compact rule-based policies. However, to overcome the problem of defining an adequate rule number and critical state features prior to the training manually, FGPRL uses genetic programming (GP) to search the complete space of available rule-feature combinations automatically. The FGPRL penalty results reported in Table 9.1 and depicted in Fig. 9.2-9.5 show that this method is able to find interpretable policies of high performance for all four benchmark problems. While the performances for different complexities are on the same level for FPSRL and FGPRL for MC and CPB benchmarks, FGPRL yields higher penalties for CPSU and lower penalties for IB. Given a similar number of fitness evaluations, FGPRL seems to have problems searching the huge state space of different feature-rule combinations, which gets the method stuck in local optima on the CPSU benchmark. The fact that the approach still found good solutions for rule complexity 125 occasionally, suggests that changes like increasing the number of individuals or iterations, evolving sub-populations to avoid getting stuck in local optima, etc. would most likely result in a more stable learning process. On the other hand, the results achieved by FGPRL on the IB problem (Fig. 9.5) show the huge advantage of this approach. The method automatically identified the most important features for each action dimension and learned sparse rules which are highly interpretable. Furthermore, FGPRL yields a whole Pareto front which contains the best policies found for each complexity level. In industry, this feature is expected to be of high value, since domain experts can, based on their knowledge and experience, select the best compromise between policy complexity and predicted performance.

The third interpretable policy approach presented in this work is genetic programming reinforcement learning (**GPRL**). Since for some problems the policy representation as a set of fuzzy rules might be unfavorable by some domain experts, GPRL applies GP to learn basic algebraic equations of low complexity as its representational form. Fig. 9.2 shows that GPRL finds solutions for MC which are of very low complexity and, at the same time, produce similar performance values as the non-interpretable PSONN approach, or fuzzy policies with multiple rules by FPSRL or FGPRL. The policy $\pi(\boldsymbol{s}) = \dot{\rho}$ literally represents the simplest MC control strategy, which is to drive in the direction of the current velocity. Of course, this strategy is not optimal, but it still yields an adequate controller. In real industry settings, domain experts are often willing to select solutions with suboptimal predicted performance, as long as the control strategy can be easily understood and verified. Good policies for the CPB benchmark are more complex in order to yield low penalty values (Fig. 9.3). However, a policy as simple as $\pi(\boldsymbol{s}) = \rho + 0.94\dot{\rho} + 6.98\theta + \dot{\theta}$ is sufficient for yielding a comparable control performance on the real system dynamics as the best PSONN solution. Even for the complicated CPSU task, one of the GPRL runs produced the following high-performing policy:

$$-0.76\rho - 0.2\theta + \frac{1.34}{\rho + \dot{\rho} + 5.51\theta + \dot{\theta}}. \tag{9.2}$$

Due to the huge search space, GPRL found such an excellent solution with a penalty value of 36.5 for the complexity of 20 only once out of ten experiments. Fig. 9.4 shows how the performance of the resulting Pareto fronts varies in comparison with that of FGPRL. The upside is that the model prediction of the penalty yields an excellent measure for selecting only solutions which are very likely to perform well on the real system dynamics. GPRL also produced very good policies for IB (Fig. 9.5). The best policy found for complexity 29 is

$$\begin{aligned}
\Delta v_0 &= -v_{-4} - 0.96, \\
\Delta g_0 &= 0.41 f_{-1} + f_{-4} - 0.59p + 0.77, \\
\Delta h_0 &= -4.05 h_{-3} - h_{-4} + 2.26p + 0.9.
\end{aligned} \tag{9.3}$$

This solution shows very clearly which state features are important for each action dimension and in which way they contribute to the produced action. Comparing this solution with the FGPRL policy depicted in Fig. 7.7 shows that both approaches selected similar state features to yield the actions. The functionally-grounded evaluation plots in Fig. 9.1 show that for MC, CPB, and IB the GPRL complexity has to be increased more strongly compared to FGPRL to further reduce the model penalties. Compared to that, for the CPSU a relatively small increase in equation complexity reduced the penalty whereas only FGPRL policies of higher rule complexity achieved a similar performance. This evaluation suggests that, the higher the targeted CPSU performance is, the more advantages do equation-based policies provide in comparison to rule-based solutions. The advantage for GPRL also holds for lower targeted performances for all other benchmarks, which is represented by flat rising curves for high penalty values. To summarize the GPRL results, it can be said that the approach produced policies with low penalty values for all of the tested benchmarks. Moreover, the solutions are highly interpretable since they are compact and formulated in the language of basic algebraic equations.

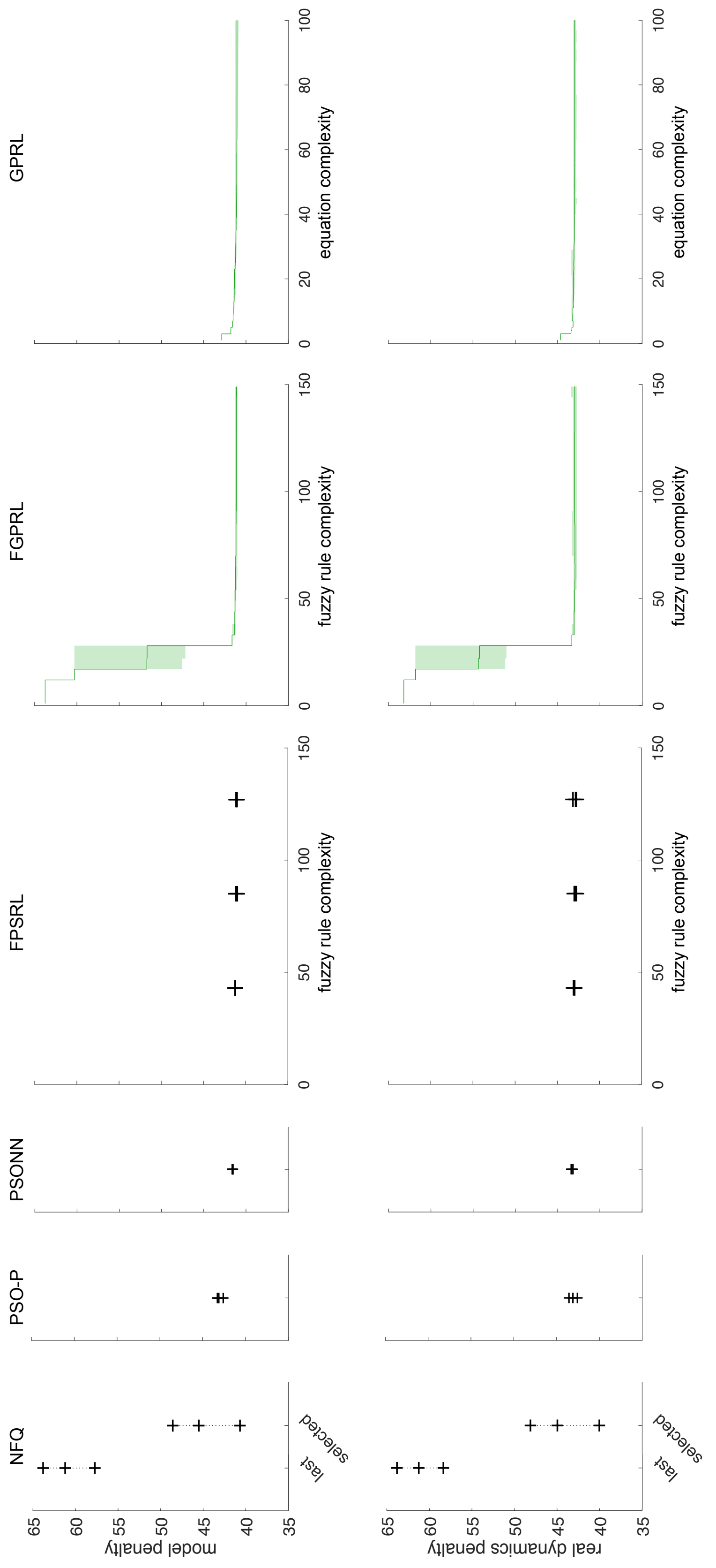


Figure 9.2: Comparing the RL methods of this thesis by their results on the MC benchmark

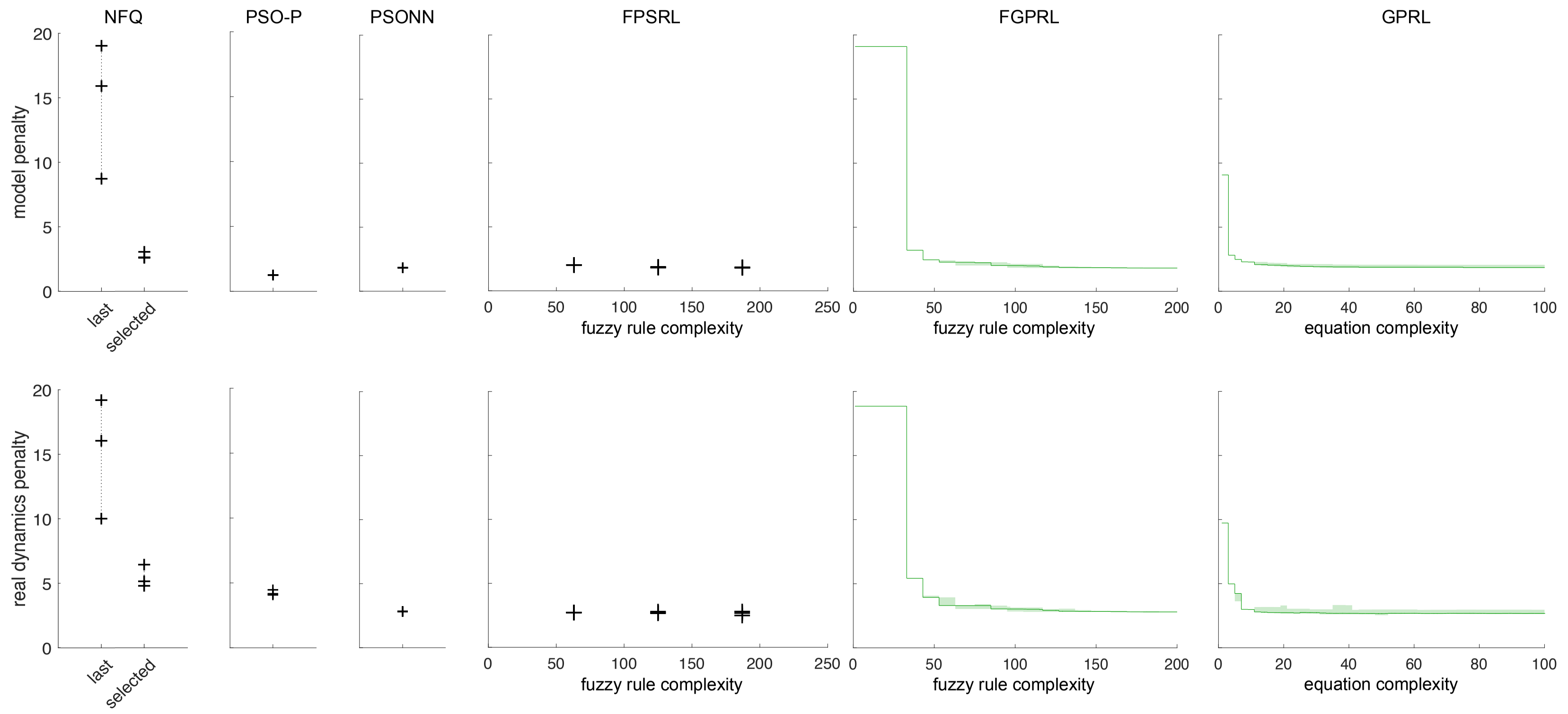


Figure 9.3: Comparing the RL methods of this thesis by their results on the CPB benchmark

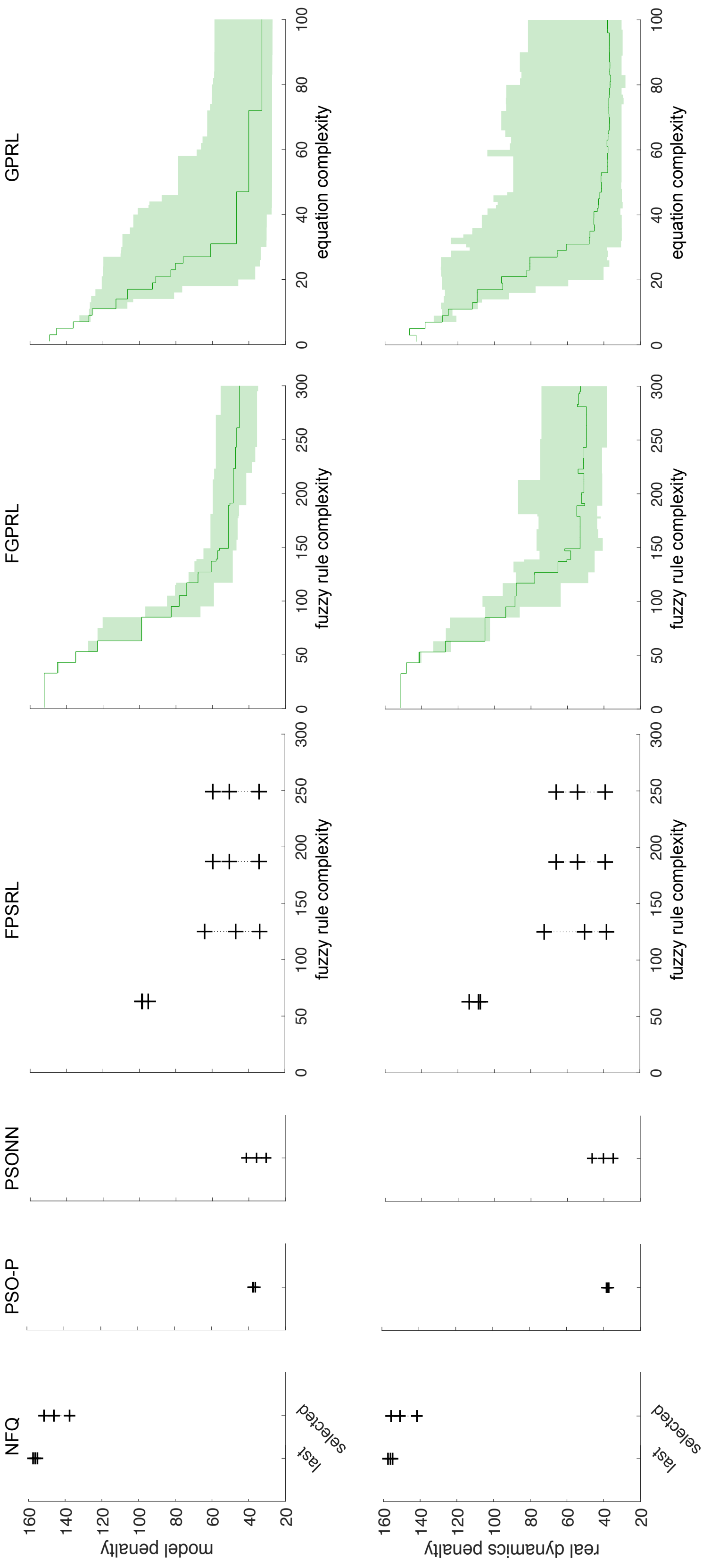


Figure 9.4: Comparing the RL methods of this thesis by their results on the CPSU benchmark

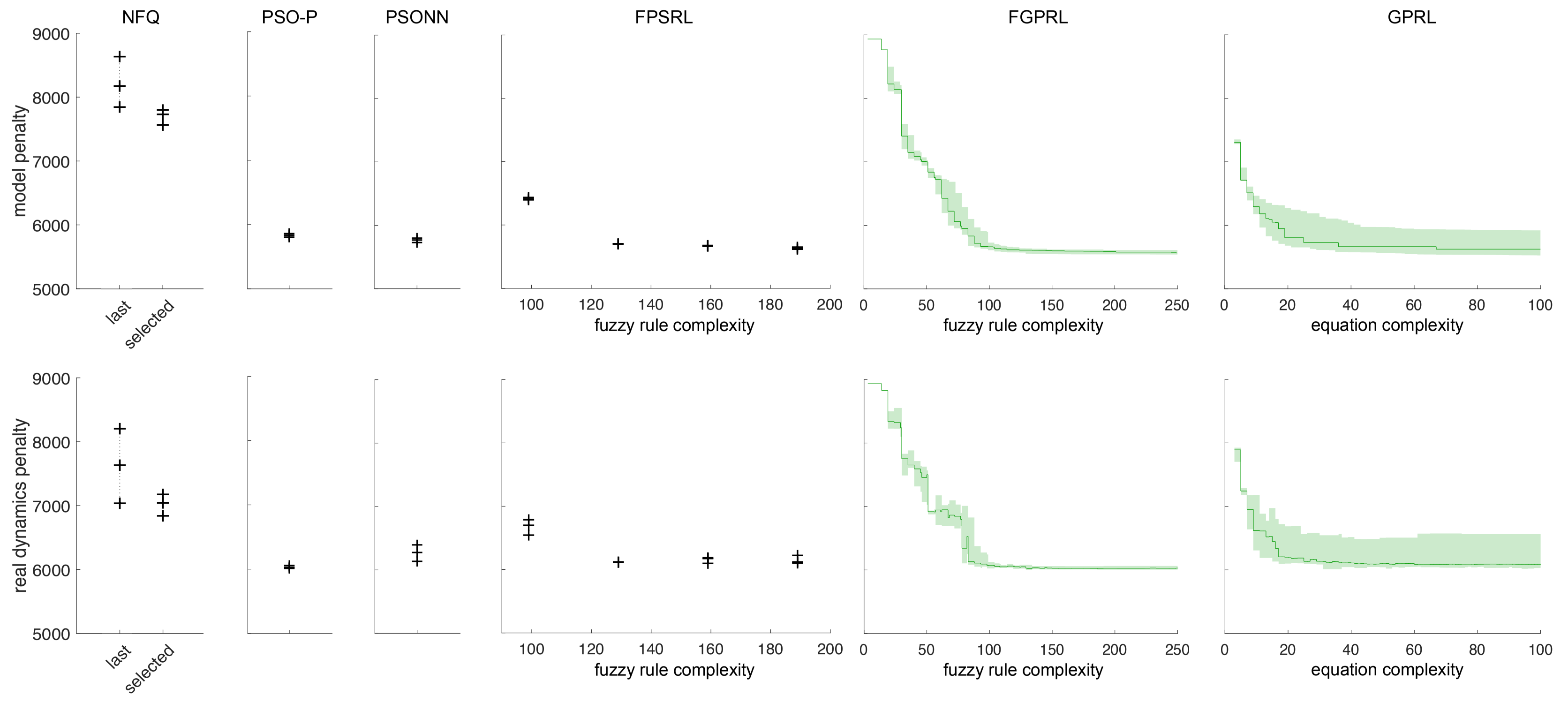


Figure 9.5: Comparing the RL methods of this thesis by their results on the IB

CHAPTER 10

Conclusion, Industrial Evaluation & Future Work

In this thesis, the problem of automatically generating human-interpretable system controllers from already available state transition samples was discussed. Three novel approaches which generate rule-based or equation-based policies to address this problem have been developed and evaluated.

In real-world industry use cases, data batches of transitions generated by default controllers are often available, whereas online reinforcement learning (RL) is prohibited for safety reasons. Therefore, the following model-based batch RL setup is investigated and applied for yielding policies from such data batches: (i) Train surrogate models from the available transition samples; (ii) Use RL policy search methods to find parameters for explicit policy representations by population-based algorithms; (iii) Rate the performance of the population's individuals using value estimations by rollouts on the surrogate models. To verify this training setup, it has been tested and evaluated on four RL benchmarks by using particle swarm optimization (PSO) for finding optimal weights of a non-interpretable, but versatile neural network (NN) policy called particle swarm optimization neural network (PSONN).

The performance has been tested on three well-known RL benchmarks mountain car (MC), cart-pole balancing (CPB), and cart-pole swing-up (CPSU). Furthermore, to evaluate the setup's performance concerning real industrial applications, a novel RL benchmark, called industrial benchmark (IB), is introduced. The IB is designed to emulate several challenging aspects existing in many industrial applications like wind or gas turbines: high-dimensional continuous state space, multi-dimensional actions, stochastic non-linear dynamics, heteroscedastic noise, partial observability, delayed reward, and multiple contradictory objectives.

The first novel interpretable policy learning approach is called fuzzy particle swarm reinforcement learning (FPSRL). Here, optimal parameters of fuzzy rule systems are learned by PSO using the proposed model-based batch RL setup. For the low-dimensional toy benchmarks MC, CPB, and CPSU, well-performing and interpretable policies could be generated straightforwardly. However, for the IB, with its high-dimensional state space, a method capable of identifying relevant state features is required to yield interpretable results with FPSRL. This problem could be solved by a new feature selection approach which first uses the model predictive control (MPC) method particle swarm optimization policy (PSO-P) to exploit the system model for optimal actions and subsequently applies a mutual-information-based heuristic to find important state features for these actions.

The second interpretable RL approach, called fuzzy genetic programming reinforcement learning (FGPRL), uses genetic programming (GP) to learn rule-based policies. The application of GP, with its integrated ability to automatically identify relevant state features as well as the minimum number of rules required, yields a whole Pareto front of solutions from which the domain expert can choose based on subjective preferences. However, the advantage of automatically searching the vast space of all possible feature-to-rule-number constellations comes at a significantly higher computational cost. The experiments show that for easily understandable tasks like MC, CPB, or CPSU, it is more efficient to make use of available prior knowledge, manually define the structure of the policy, and apply FPSRL. For high-dimensional state spaces, as present in the IB, FGPRL takes advantage of its built-in state feature selection capability.

The third interpretable policy learning approach also uses GP. However, since for some problems the policy representation as fuzzy rules might be inexpedient, this method, called genetic programming reinforcement learning (GPRL), searches the space of basic algebraic equations. The results of experiments with the four benchmarks show that it is possible to find very simple solutions, which are human-interpretable but still yield a competitive control performance. By investigating the individuals on the resulting Pareto fronts of several independent GPRL runs on the same benchmark, common policy concepts, crucial state features, as well as important sub-terms of the equations could be identified. However, similarly to FGPRL, the search in a vast space of solutions and maintaining a whole population of potentially essential individuals comes at high computational effort. Nevertheless, by making use of the inherent parallelism of population-based evolutionary computation methods, GP-based approaches become more feasible for a broader range of real-world problems.

All three interpretable approaches FPSRL, FGPRL, and GPRL, were able to produce well-performing human-interpretable policies for all of the benchmark problems, including the industry-inspired IB. While for most of the experiments, the interpretable policies were on a par with the non-interpretable PSONN results regarding the training error using the surrogate models, sometimes the interpretable results surpassed the PSONN policies during evaluation on the real system dynamics. One reason for that could be the fact that PSONN used its significantly higher number of degrees of freedom in terms of policy parameters to overfit its control strategy to inaccuracies of the system models. Since the interpretable solutions are forced to yield compact representations which are mainly designed to cover the most important aspects of controlling the system, these solutions sometimes tend to show better generalization properties on novel situations.

In summary, it can be stated that the novel approaches of this thesis can successfully utilize available transition batch data from challenging system dynamics to automatically learn human-interpretable RL policies that yield excellent control performance and, at the same time, provide a huge variety of potential solutions of different complexities.

Industrial Evaluation

The interpretable methods of this thesis have been evaluated on several real industrial applications, like traffic light and wind turbine optimization. These feasibility and pre-studies were supported by funds of the German Federal Ministry of Education and Research within the scope of the *autonomous learning in complex environments* (ALICE) II project (project number 01IB15001). In one of the industrial evaluations, GPRL has been utilized to generate interpretable policies for several wind turbines in a wind farm based on previously generated exploration data. Subsequently, domain experts interpreted and discussed the learned policies, and selected promising candidates for deployment on the wind farm. Empirical evaluations showed that, despite the compact and easily interpretable form of the policies, their performance in terms of optimizing the wind turbine's power output was on a par with other non-interpretable policy solutions. Both the wind farm operators as well as the funding agency recognized the opportunities in adding additional qualities to RL, like safety guaranties and traceability from interpretable solutions. Consequently, the German Federal Ministry of Education and Research decided in 2018 to fund the followup project ALICE III which, among other topics, contains the working package *domain specific search spaces* which aims at evaluating the capabilities of evolutionary computation for finding solutions in an even broader range of industrial applications.

Future Work

To further continue the application of the approach of learning interpretable policies as introduced in this work, testing and evaluation of the methods on other tasks and benchmarks is essential. Since the world model is a critical component in the setup of the model-based RL methods, the following questions, among others, have to be answered for novel tasks:

- Does the batch data contain sufficient explorational information?
- Is the amount of data sufficient for the model building process?
- Which representational forms are good candidates for certain system dynamics?
- How can model inaccuracies be detected and fixed prior to the RL training process?

To answer these and other questions, we can make use of a huge amount of available literature, knowledge, and experiences from the modeling and simulation as well as the supervised ML communities.

Despite one of the benchmarks considered in this work having highly stochastic system dynamics, the surrogate models which have been used for the rollout-based fitness calculations were just deterministic recurrent neural networks (RNNs). The good results show that using mean estimators can be sufficient to calculate adequate performance predictions for policies. However, modeling techniques that can provide a measure of uncertainty in their predictions, such as Gaussian processes or Bayesian NNs, are expected to yield more stable results with higher performance for a wide range of stochastic problems. Recent developments in modeling stochastic dynamic systems not only provide an approximation of the mean of the next system state but also compute uncertainty for transitions in the state-action space [27].

Furthermore, using model-based rollouts is not the only approach to calculate the performance during policy search. Applying model-free methods which use the Bellman equation to learn the state-action value function for the policy candidates might enable to use the proposed interpretable RL methods of this thesis for problems for which building system models is infeasible.

We have seen in the experiments using the GP-based approaches FGPRL and GPRL, that searching huge solution spaces, like the spaces of fuzzy rule policies or the space of algebraic equations, occasionally converges to local optima. In the studies conducted in this work, repeating the experiments several times has always resulted in at least one Pareto front which contained well-performing policies of low complexity. However, integrating established heuristics like niching, speciation, sub-populations, adaptive parameters, etc. are promising concepts for automatically stabilizing GP to converge to global optima with a higher probability. Achieving more robust results would allow us to reduce the number of required GP runs which eventually would maximize the effectiveness of the invested computational resources.

Since this work considers functionally-grounded evaluation of the introduced approaches, GP has been applied to search for policy representations which are already regarded as interpretable. Nevertheless, conducting application- or human-grounded evaluation with domain experts or other persons is expected to reveal alternative policy representations.

Anyhow, a variety of interpretable policy representations already exist in industry. These representations stem from the requirement of yielding control strategies which need to be understandable due to customer demands or authority specifications. Examples are precisely defined signal plans for traffic control solutions or optimal scheduling policies for manufacturing industries. Since GP is able to generate computer programs, it should be a relatively straightforward process to apply the approaches of this thesis to a variety of other real-world problems.

Bibliography

[1] E. Alba, C. Cotta, and J. M. Troya. "Evolutionary design of fuzzy logic controllers using strongly-typed GP." In: *Mathware and Soft Computing* 6.1 (1999), pp. 109–124.

[2] E. Alba, C. Cotta, and J. M. Troya. "Type-constrained genetic programming for rule-base definition in fuzzy logic controllers." In: *Proceedings of the 1st annual conference on genetic programming*. MIT Press. 1996, pp. 255–260.

[3] K. J. Åström and T. Hägglund. "Revisiting the Ziegler–Nichols step response method for PID control." In: *Journal of process control* 14.6 (2004), pp. 635–650.

[4] K. J. Åström and T. Hägglund. "The future of PID control." In: *Control engineering practice* 9.11 (2001), pp. 1163–1175.

[5] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation." In: *PloS one* 10.7 (2015), e0130140.

[6] B. Bakker. "The state of mind: Reinforcement learning with recurrent neural networks." PhD thesis. Netherlands: Leiden University, 2004.

[7] D. Barash. "A genetic search in policy space for solving Markov decision processes." In: *AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*. 1999.

[8] R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1962.

[9] F. J. Berlanga, A. J. Rivera, M. J. del Jesús, and F. Herrera. "GP-COACH: Genetic Programming-based learning of COmpact and ACcurate fuzzy rule-based classification systems for High-dimensional problems." In: *Information Sciences* 180.8 (2010), pp. 1183–1200.

[10] B. Bischoff, D. Nguyen-Tuong, T. Koller, H. Markert, and A. Knoll. "Learning throttle valve control using policy search." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2013, pp. 49–64.

[11] T. Blickle and L. Thiele. "A Mathematical Analysis of Tournament Selection." In: *ICGA*. 1995, pp. 9–16.

[12] J. A. Boyan and A. W. Moore. "Generalization in reinforcement learning: Safely approximating the value function." In: *Advances in neural information processing systems*. 1995, pp. 369–376.

[13] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Boca Raton, FL: CRC Press, 1984.

[14] L. Busoniu, R. Babuska, B. De Shutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximation*. CRC Press, 2010.

[15] F. Camacho and C. Alba. *Model predictive control*. Springer, 2007.

[16] J. Casillas, O. Cordon, F. Herrera, and L. Magdalena. "Interpretability improvements to find the balance interpretability-accuracy in fuzzy modeling: An overview." In: *Interpretability issues in fuzzy modeling*. Springer, 2003, pp. 3–22.

[17] H. S. Chang, J. Hu, M. C. Fu, and S. I. Marcus. "Population-Based Evolutionary Approaches." In: *Simulation-Based Algorithms for Markov Decision Processes*. Springer, 2007. Chap. 3, pp. 61–87.

[18] X. Chen and Y. Li. "Neural network training using stochastic PSO." In: *International Conference on Neural Information Processing*. Springer. 2006, pp. 1051–1060.

[19] B.-C. Chien, J. Y. Lin, and T.-P. Hong. "Learning discriminant functions with fuzzy attributes for classification using genetic programming." In: *Expert Systems with Applications* 23.1 (2002), pp. 31–37.

[20] H. H. Chin and A. A. Jafari. "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. 1998, pp. 538–543.

[21] M. Conforth and Y. Meng. "Reinforcement learning using swarm intelligence-trained neural networks." In: *Journal of Experimental & Theoretical Artificial Intelligence* 22.3 (2010), pp. 197–218.

[22] O. Cordón, F. Gomide, F. Herrera, F. Hoffmann, and L. Magdalena. "Ten years of genetic fuzzy systems: Current framework and new trends." In: *Fuzzy sets and systems* 141.1 (2004), pp. 5–31.

[23] A. B. Corripio. *Tuning of industrial control systems*. Isa, 2000.

[24] G. Cybenko. "Approximation by superpositions of a sigmoidal function." In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[25] S. B. C. Debnath, P. C. Shill, and K. Murase. "Particle swarm optimization based adaptive strategy for tuning of fuzzy logic controller." In: *International Journal of Artificial Intelligence & Applications* 4.1 (2013), pp. 37–50.

[26] M. P. Deisenroth. *Efficient reinforcement learning using Gaussian processes*. Vol. 9. KIT Scientific Publishing, 2010.

[27] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft. “Learning and policy search in stochastic dynamical systems with Bayesian neural networks.” In: *arXiv preprint arXiv:1605.07127* (2016).

[28] L. Desborough and R. Miller. “Increasing customer value of industrial control performance monitoring - Honeywell’s experience.” In: *AIChE symposium series*. 326. New York; American Institute of Chemical Engineers; 1998. 2002, pp. 169–189.

[29] A. Dix. “Human-computer interaction.” In: *Encyclopedia of database systems*. Springer, 2009, pp. 1327–1331.

[30] F. Doshi-Velez and B. Kim. “Towards a rigorous science of interpretable machine learning.” In: *arXiv preprint arXiv:1702.08608* (2017).

[31] K. L. Downing. “Adaptive Genetic Programs via Reinforcement Learning.” In: *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. GECCO’01. San Francisco, California: Morgan Kaufmann Publishers Inc., 2001, pp. 19–26.

[32] R. Dubčáková. “Eureqa: Software review.” In: *Genetic programming and evolvable machines* 12.2 (2011), pp. 173–178.

[33] S. Duell, S. Udluft, and V. Sterzing. “Solving Partially Observable Reinforcement Learning Problems with Recurrent Neural Networks.” In: *Neural Networks: Tricks of the Trade*. Ed. by G. Montavon, G. Orr, and K.-R. Müller. Vol. 7700. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 709–733.

[34] R. C. Eberhart, P. Simpson, and R. Dobbins. *Computational Intelligence PC Tools*. San Diego, CA, USA: Academic Press Professional, Inc., 1996.

[35] R. Eberhart and Y. Shi. “Comparing inertia weigths and constriction factors in particle swarm optimization.” In: *Proceedings of the IEEE Congress on Evolutionary Computation*. 2000, pp. 84–88.

[36] J. L. Elman. “Finding structure in time.” In: *Cognitive science* 14.2 (1990), pp. 179–211.

[37] A. P. Engelbrecht. *Fundamentals of computational swarm intelligence*. Wiley, 2005.

[38] A. P. Engelbrecht. “Particle swarm optimization: Velocity initialization.” In: *2012 IEEE Congress on Evolutionary Computation*. June 2012, pp. 1–8.

[39] D. Ernst, P. Geurts, and L. Wehenkel. “Tree-based Batch Mode Reinforcement Learning.” In: *Journal of Machine Learning Research* 6 (2005), pp. 503–556.

[40] W. Ertel, M. Schneider, R. Cubek, and M. Tokic. “The teaching-box: A universal robot learning framework.” In: *Proceedings of the 14th International Conference on Advanced Robotics (ICAR 2009), Munich*. 2009.

[41] I. Fantoni and R. Lozano. *Non-linear control for underactuated mechanical systems*. Springer, 2002.

[42] S. Faußer and F. Schwenker. “Neural network ensembles in reinforcement learning.” In: *Neural Processing Letters* 41.1 (2015), pp. 55–69.

[43] H.-M. Feng. “Particle swarm optimization learning fuzzy systems design.” In: *Third International Conference on Information Technology and Applications*. 2005, pp. 363–366.

[44] H.-M. Feng. “Particle swarm optimization learning fuzzy systems design.” In: *Third International Conference on Information Technology and Applications, 2005. ICITA 2005*. Vol. 1. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 2005, pp. 363–366.

[45] H.-M. Feng. “Self-generation fuzzy modeling systems through hierarchical recursive-based particle swarm optimization.” In: *Cybernetics and Systems* 36.6 (2005), pp. 623–639.

[46] R. Findeisen and F. Allgoewer. “An introduction to nonlinear model predictive control.” In: *21st Benelux Meeting on Systems and Control*. 2002, pp. 1–23.

[47] R. Findeisen, F. Allgoewer, and L. Biegler. *Assessment and future directions of nonlinear model predictive control*. Springer-Verlag, 2007.

[48] T. Gabel and M. Riedmiller. “Reducing policy degradation in neuro-dynamic programming.” In: *ESANN*. 2006, pp. 653–658.

[49] C. Gearhart. “Genetic Programming as Policy Search in Markov Decision Processes.” In: *Genetic Algorithms and Genetic Programming at Stanford 2003*. Ed. by J. R. Koza. Stanford, California, 94305-3079 USA: Stanford Bookstore, 2003, pp. 61–67.

[50] A. Geyer-Schulz. “Fuzzy Rule-based Expert Systems and Genetic Machine Learning.” In: *Physica-Verlag, Heidelberg* (1995).

[51] K. Glover and D. C. McFarlane. *Robust Controller Design Using Normalized Coprime Factor Plant Descriptions*. Berlin, Heidelberg: Springer-Verlag, 1989.

[52] F. Gomez, J. Schmidhuber, and R. Miikkulainen. “Efficient non-linear control through neuroevolution.” In: *European Conference on Machine Learning*. Springer. 2006, pp. 654–662.

[53] G. J. Gordon. “Approximate Solutions to Markov Decision Processes.” PhD thesis. Pittsburgh PA: Carnegie Mellon University, 1999.

[54] G. J. Gordon. “Reinforcement learning with function approximation converges to a region.” In: *Advances in neural information processing systems*. 2001, pp. 1040–1046.

[55] G. J. Gordon. “Stable function approximation in dynamic programming.” In: *Proceedings of the Twelfth International Conference on Machine Learning*. Morgan Kaufmann, 1995, pp. 261–268.

[56] A. Graves, A.-r. Mohamed, and G. Hinton. "Speech recognition with deep recurrent neural networks." In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 6645–6649.

[57] L. Gruene and J. Pannek. *Nonlinear model predictive control*. Springer, 2011.

[58] T. O. S. Hanafy. "Design and validation of real time neuro fuzzy controller for stabilization of pendulum-cart system." In: *Life Science Journal* 8.1 (2011), pp. 52–60.

[59] A. Hans and S. Udluft. "Ensembles of neural networks for robust reinforcement learning." In: *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*. IEEE. 2010, pp. 401–406.

[60] S. Haykin. *Neural networks: A comprehensive foundation*. Prentice Hall PTR, 1994.

[61] D. Hein, S. Depeweg, M. Tokic, S. Udluft, A. Hentschel, T. A. Runkler, and V. Sterzing. "A benchmark environment motivated by industrial control problems." In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2017, pp. 1–8.

[62] D. Hein, A. Hentschel, T. A. Runkler, and S. Udluft. "Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies." In: *Engineering Applications of Artificial Intelligence* 65 (2017), pp. 87–98.

[63] D. Hein, A. Hentschel, T. A. Runkler, and S. Udluft. "Particle swarm optimization for model predictive control in reinforcement learning environments." In: *Critical Developments and Applications of Swarm Intelligence*. Ed. by Y. Shi. Hershey, PA, USA: IGI Global, 2018. Chap. 16, pp. 401–427.

[64] D. Hein, A. Hentschel, T. A. Runkler, and S. Udluft. "Reinforcement Learning with Particle Swarm Optimization Policy (PSO-P) in Continuous State and Action Spaces." In: *International Journal of Swarm Intelligence Research (IJSIR)* 7.3 (2016), pp. 23–42.

[65] D. Hein, S. Udluft, M. Tokic, A. Hentschel, T. A. Runkler, and V. Sterzing. "Batch reinforcement learning on the industrial benchmark: First experiences." In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 4214–4221.

[66] D. Hein, S. Udluft, and T. A. Runkler. "Generating Interpretable Fuzzy Controllers Using Particle Swarm Optimization and Genetic Programming." In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '18. Kyoto, Japan: ACM, 2018, pp. 1268–1275.

[67] D. Hein, S. Udluft, and T. A. Runkler. "Interpretable policies for reinforcement learning by genetic programming." In: *Engineering Applications of Artificial Intelligence* 76 (2018), pp. 158–169.

[68] F. Hoffmann and O. Nelles. "Genetic programming for model selection of TSK-fuzzy systems." In: *Information Sciences* 136.1-4 (2001), pp. 7–28.

[69] K. Hornik, M. Stinchcombe, and H. White. "Multilayer Feedforward Networks are Universal Approximators." In: *Neural Networks* 2 (1989), pp. 359–366.

[70] K. Hornik. “Approximation capabilities of multilayer feedforward networks.” In: *Neural networks* 4.2 (1991), pp. 251–257.

[71] J. S. Jang. “Adaptive-network-based fuzzy inference system.” In: *IEEE Transactions on Systems, Man & Cybernetics* 23.3 (1993), pp. 665–685.

[72] T. Johansen. “Introduction to nonlinear model predictive control and moving horizon estimation.” In: *Selected Topics on Constrained and Nonlinear Control*. Ed. by M. Huba, S. Skogestad, M. Fikar, M. Hovd, T. Johansen, and B. Rohal-Ilkiv. STU Bratislava/NTNU Trondheim, 2011.

[73] C.-F. Juang. “A hybrid of genetic algorithm and particle swarm optimization for recurrent network design.” In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34.2 (2004), pp. 997–1006.

[74] S. Kamio and H. Iba. “Adaptation Technique for Integrating Genetic Programming and Reinforcement Learning for Real Robots.” In: *Trans. Evol. Comp* 9.3 (June 2005), pp. 318–333.

[75] H. Katagiri, K. Hirasawa, J. Hu, J. Murata, and M. Kosaka. “Network Structure Oriented Evolutionary Model: Genetic Network Programming.” In: *Transactions of the Society of Instrument and Control Engineers* 38.5 (2002), pp. 485–494.

[76] M. A. Keane, J. R. Koza, and M. J. Streeter. “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. Washington, DC, USA: IEEE Computer Society, 2002, pp. 113–123.

[77] J. Kennedy and R. C. Eberhart. “Particle swarm optimization.” In: *Proceedings of the IEEE International Joint Conference on Neural Networks* (1995), pp. 1942–1948.

[78] A. Kharola and P. Gupta. “Stabilization of inverted pendulum using hybrid adaptive neuro fuzzy (ANFIS) controller.” In: *Engineering Science Letters* 4 (2014), pp. 1–20.

[79] A. S. Koshiyama, T. Escovedo, M. M. B. R. Vellasco, and R. Tanscheit. “GPFIS-Control: A fuzzy Genetic model for Control tasks.” In: *Fuzzy Systems (FUZZ-IEEE), 2014 IEEE International Conference on*. IEEE. 2014, pp. 1953–1959.

[80] R. Kothandaraman and L. Ponnusamy. “PSO tuned adaptive neuro-fuzzy controller for vehicle suspension systems.” In: *Journal of Advances in Information Technology* 3.1 (2012).

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

[82] J. R. Koza. “Human-competitive results produced by genetic programming.” In: *Genetic Programming and Evolvable Machines* 11.3 (2010), pp. 251–284.

[83] W. H. Kwon, A. M. Bruckstein, and T. Kailath. "Stabilizing state-feedback design via the moving horizon method." In: *International Journal of Control* (1983), pp. 631–643.

[84] M. G. Lagoudakis and R. Parr. "Least-squares policy iteration." In: *Journal of Machine Learning Research* (2003), pp. 1107–1149.

[85] N. Le, H. N. Xuan, A. Brabazon, and T. P. Thi. "Complexity measures in Genetic Programming learning: A brief review." In: *Evolutionary Computation (CEC), 2016 IEEE Congress on*. IEEE. 2016, pp. 2409–2416.

[86] S.-M. Lee and H. Myung. "Receding horizon particle swarm optimisation-based formation control with collision avoidance for non-holonomic mobile robots." In: *IET Control Theory & Applications* (2015), pp. 2075–2083.

[87] D. Liberzon. *Switching in systems and control*. Springer Science & Business Media, 2003.

[88] T. Lombrozo. "The structure and function of explanations." In: *Trends in cognitive sciences* 10.10 (2006), pp. 464–470.

[89] S. Mabu, K. Hirasawa, and J. Hu. "Genetic Network Programming with Reinforcement Learning and Its Performance Evaluation." In: *Genetic and Evolutionary Computation - GECCO 2004: Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004. Proceedings, Part II*. Ed. by K. Deb. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 710–711.

[90] S. Mabu, K. Hirasawa, J. Hu, and J. Murata. "Online learning of Genetic Network Programming(GNP)." In: *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*. Ed. by D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton. IEEE Press, 2002, pp. 321–326.

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

[92] L. Magni and R. Scattolini. "Stabilizing model predictive control of nonlinear continuous time systems." In: *Annual Reviews in Control* (2004), pp. 1–11.

[93] E. H. Mamdani and S. Assilian. "An experiment in linguistic synthesis with a fuzzy logic controller." In: *International Journal of Man-Machine Studies* 7.1 (1975), pp. 1–13.

[94] D. P. Mandic and J. Chambers. *Recurrent neural networks for prediction: Learning algorithms, architectures and stability*. John Wiley & Sons, Inc., 2001.

[95] T. E. Marlin. *Process control: Designing processes and control systems for dynamic performance*. Vol. 2. McGraw-Hill New York, 1995.

[96] D. McFarlane and K. Glover. "A loop-shaping design procedure using H infinity synthesis." In: *IEEE transactions on automatic control* 37.6 (1992), pp. 759–769.

[97] G. K. McMillan. *Tuning and control loop performance*. Momentum Press, 2014.

[98] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur. "Recurrent neural network based language model." In: *Eleventh Annual Conference of the International Speech Communication Association*. 2010.

[99] N. Minorsky. "Directional stability of automatically steered bodies." In: *Journal of the American Society for Naval Engineers* 34.2 (1922), pp. 280–309.

[100] M. Montazeri-Gh, S. Jafari, and M. R. Ilkhani. "Application of particle swarm optimization in gas turbine engine fuel controller gain tuning." In: *Engineering Optimization* (2012), pp. 225–240.

[101] A. W. Moore. *Efficient memory-based learning for robot control*. Tech. rep. University of Cambridge, Computer Laboratory, 1990.

[102] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette. "Evolutionary algorithms for reinforcement learning." In: *Journal of Artifcial Intelligence Research* (1999), pp. 241–276.

[103] P. Moscato. "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms." In: *Caltech concurrent computation program, C3P Report* 826 (1989), p. 1989.

[104] D. Ormoneit and S. Sen. "Kernel-based reinforcement learning." In: *Machine learning* 49.2 (2002), pp. 161–178.

[105] Y. Ou, P. Kang, K. M. Jun, and A. A. Julius. "Algorithms for simultaneous motion control of multiple T. pyriformis cells: Model predictive control and particle swarm optimization." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 3507–3512.

[106] A. Perzylo, N. Somani, S. Profanter, I. Kessler, M. Rickert, and A. Knoll. "Intuitive instruction of industrial robots: Semantic process descriptions for small lot production." In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE. 2016, pp. 2293–2300.

[107] S. Piche, J. Keeler, G. Martin, G. Boe, D. Johnson, and M. Gerules. "Neural network based model predictive control." In: *Advances in Neural Information Processing Systems* (2000), pp. 1029–1035.

[108] T. J. Procyk and E. H. Mamdani. "A linguistic self-organizing process controller." In: *Automatica* 15 (1979), pp. 15–30.

[109] T. Raiko and M. Tornio. "Variational Bayesian learning of nonlinear hidden state-space models for model predictive control." In: *Neurocomputing* 72.16-18 (2009), pp. 3704–3712.

[110] L. S. Ramos and J. A. C. González. "A niching scheme for steady state GA-P and its application to fuzzy rule based classifiers induction." In: *Mathware and Soft Computing* 7.2-3 (2000), pp. 337–350.

[111] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptative computation and machine learning series. University Press Group Limited, 2006.

[112] J. Rawlings. “Tutorial overview of model predictive control.” In: *IEEE Control Systems Magazine* (2000), pp. 38–52.

[113] J. Rawlings and D. Mayne. *Model predictive control theory and design*. Nob Hill Publishing, 2009.

[114] “Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation).” In: *OJ* L 119 (May 2008), pp. 1–88.

[115] M. T. Ribeiro, S. Singh, and C. Guestrin. “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. 2016, pp. 1135–1144.

[116] M. Riedmiller. “Neural fitted Q iteration - First experiences with a data efficient neural reinforcement learning method.” In: *Machine Learning: ECML 2005*. Vol. 3720. Springer, 2005, pp. 317–328.

[117] M. Riedmiller. “Neural reinforcement learning to swing-up and balance a real pole.” In: *Systems, Man and Cybernetics, 2005 IEEE International Conference on*. Vol. 4. 2005, pp. 3191–3196.

[118] M. Riedmiller and H. Braun. “A direct adaptive method for faster backpropagation learning: The RPROP algorithm.” In: *Neural Networks, 1993., IEEE International Conference on*. IEEE. 1993, pp. 586–591.

[119] M. Riedmiller and H. Braun. “RPROP - A fast adaptive learning algorithm.” In: *Proceedings of International Symposium on Computer and Information Science VII*. 1992, pp. 279–286.

[120] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange. “Reinforcement learning for robot soccer.” In: *Autonomous Robots* 27.1 (2009), pp. 55–73.

[121] R. Rojas. *Neural Networks: A Systematic Introduction*. Springer, 1996.

[122] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.” In: *Psychological Reviews* 65 (1958), pp. 386–408.

[123] A. A. Saifizul, C. A. Azlan, and N. F. Mohd Nasir. “Takagi-Sugeno fuzzy controller design via ANFIS architecture for inverted pendulum system.” In: *Proceedings of International Conference on Man-Machine Systems*. 2006.

[124] L. Sánchez, I. Couso, and J. A. Corrales. “Combining GP operators with SA search to evolve fuzzy rule based classifiers.” In: *Information Sciences* 136.1-4 (2001), pp. 175–191.

[125] A. M. Schäfer. "Reinforcement Learning with Recurrent Neural Networks." PhD thesis. Germany: University of Osnabrück, 2008.

[126] A. M. Schäfer, D. Schneegass, V. Sterzing, and S. Udluft. "A neural reinforcement learning approach to gas turbine control." In: *IEEE International Conference on Neural Networks - Conference Proceedings*. 2007, pp. 1691–1696.

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

[128] A. M. Schäfer, S. Udluft, and H.-G. Zimmermann. "A Recurrent Control Neural Network for Data Efficient Reinforcement Learning." In: *Proceedings of IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*. 2007, pp. 151–157.

[129] E. M. Scharf and N. J. Mandve. "The application of a fuzzy controller to the control of a multi-degree-freedom robot arm." In: *Industrial Application of Fuzzy Control*. Ed. by M. Sugeno. North-Holland, 1985, pp. 41–62.

[130] D. Schneegass, S. Udluft, and T. Martinetz. "Improving optimality of neural rewards regression for data-efficient batch near-optimal policy identification." In: *Proceedings the International Conference on Artificial Neural Networks*. 2007, pp. 109–118.

[131] D. Schneegass, S. Udluft, and T. Martinetz. "Neural rewards regression for near-optimal policy identification in Markovian and partial observable environments." In: *Proceedings the European Symposium on Artificial Neural Networks*. 2007, pp. 301–306.

[132] H.-P. Schwefel. *Evolution and optimum seeking. Sixth-generation computer technology series*. Wiley, New York, 1995.

[133] H.-P. Schwefel. *Numerical optimization of computer models*. John Wiley & Sons, Inc., 1981.

[134] D. E. Seborg, D. A. Mellichamp, T. F. Edgar, and F. J. Doyle III. *Process dynamics and control*. John Wiley & Sons, 2010.

[135] A. D. Selbst and J. Powles. "Meaningful information and the right to explanation." In: *International Data Privacy Law* 7.4 (2017), pp. 233–242.

[136] S. Shao. "Fuzzy self-organizing controller and its application for dynamic processes." In: *Fuzzy Sets and Systems* 26 (1988), pp. 151–164.

[137] Y. Shi and R. C. Eberhart. "A Modified Particle Swarm Optimizer." In: *Proceedings of the IEEE Congress on Evolutionary Computation* (May 1998), pp. 69–73.

[138] H. Shimooka and Y. Fujimoto. "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. London, UK, UK: Springer-Verlag, 1999, pp. 179–186.

[139] F. G. Shinskey. *Process control systems*. McGraw-Hill, Inc., 1979.

[140] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. "Deterministic policy gradient algorithms." In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, pp. I-387–I-395.

[141] S. Skogestad and I. Postlethwaite. *Multivariable feedback control: Analysis and design*. Vol. 2. Wiley New York, 2007.

[142] C. A. Smith and A. B. Corripio. *Principles and practice of automatic process control*. Vol. 2. Wiley New York, 1985.

[143] C. L. Smith. *Digital computer process control*. Intext Educational Publishers, 1972.

[144] M. Solihin and R. Akmeliawati. "Particle swam optimization for stabilizing controller of a self-erecting linear inverted pendulum." In: *International Journal of Electrical and Electronic Systems Research* (2010), pp. 13–23.

[145] R. S. Sutton. "Generalization in reinforcement learning: Successful examples using sparse coarse coding." In: *Advances in Neural Information Processing Systems* 8 (1996), pp. 1038–1044.

[146] R. S. Sutton. "Learning to predict by the methods of temporal differences." In: *Machine learning* 3.1 (1988), pp. 9–44.

[147] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. 2nd ed. A Bradford Book, 2018.

[148] M. Tesmer and P. A. Estévez. "AMIFS: Adaptive feature selection by using mutual information." In: *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*. Vol. 1. IEEE. 2004, pp. 303–308.

[149] S. Thrun and A. Schwartz. "Issues in using function approximation for reinforcement learning." In: *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*. 1993.

[150] A. Tsakonas. "Local and global optimization for Takagi-Sugeno fuzzy system by memetic genetic programming." In: *Expert Systems with Applications* 40.8 (2013), pp. 3282–3298.

[151] E. Tunstel and M. Jamshidi. "On genetic programming of fuzzy rule-based systems for intelligent control." In: *Intelligent Automation & Soft Computing* 2.3 (1996), pp. 271–284.

[152] H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double Q-learning." In: *30th AAAI Conference on Artificial Intelligence, AAAI 2016*. 2016, pp. 2094–2100.

[153] K. Van Heerden, Y. Fujimoto, and A. Kawamura. "A combination of particle swarm optimization and model predictive control on graphics hardware for real-time trajectory planning of the under-actuated nonlinear acrobot." In: *2014 IEEE 13th International Workshop on Advanced Motion Control (AMC)*. 2014, pp. 464–469.

[154] J.-J. Wang. “Simulation studies of inverted pendulum based on PID controllers.” In: *Simulation Modelling Practice and Theory* 19.1 (2011), pp. 440–449.

[155] L.-X. Wang and J. M. Mendel. “Fuzzy basis functions, universal approximation, and orthogonal least-squares learning.” In: *IEEE Transactions on Neural Networks* 3.5 (1992), pp. 807–814.

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

[157] D. G. Wilson, S. Cussat-Blanc, H. Luga, and J. F. Miller. “Evolving simple programs for playing Atari games.” In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’18. Kyoto, Japan: ACM, 2018, pp. 229–236.

[158] D. Wolpert and W. Macready. “No free lunch theorems for optimization.” In: *IEEE Transactions on Evolutionary Computation* (1997), pp. 67–82.

[159] F. Xu, H. Chen, X. Gong, and Q. Mei. “Fast nonlinear model predictive control on FPGA using particle swarm optimization.” In: *IEEE Transactions on Industrial Electronics* (2016), pp. 310–321.

[160] J. Yao, G. Jiang, S. Gao, H. Yan, and D. Di. “Particle swarm optimization-based neural network control for an electro-hydraulic servo system.” In: *Journal of Vibration and Control* 20.9 (2014), pp. 1369–1377.

[161] L. A. Zadeh. “Fuzzy sets.” In: *Information and Control* 8 (1965), pp. 338–353.

[162] C. Zhang, H. Shao, and Y. Li. “Particle swarm optimisation for evolving artificial neural network.” In: *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*. Vol. 4. IEEE. 2000, pp. 2487–2490.

[163] J. G. Ziegler and N. B. Nichols. “Optimum settings for automatic controllers.” In: *Transactions of the A.S.M.E.* 64.11 (1942).