Doctoral Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

# Symbolic Regression for Reinforcement Learning in Continuous Spaces

Eduard Alibekov

Supervisor: prof. RNDr. Olga Štěpánková, CSc.
Supervisor–specialist: prof. Dr. Ing. Robert Babuška
Study program: Electrical Engineering and Information Technology (P2612)
Branch of study: Artificial Intelligence and Biocybernetics (3902V035)
August 2021

## Acknowledgements

I would like to thank:

- prof. RNDr. Olga Štepánková, CSc. for her gentle guidance through my Ph.D. journey.
- prof. Dr. Ing. Robert Babuška for his invaluable efforts to make a good scientist out of me (despite my best efforts to resist it).
- Ing. Jiří Kubalík, Ph.D. for being a great colleague, supporter, and person.
- prof. Ing. Vladimír Mařík, DrSc., dr. h. c. for showing me what "effectiveness" truly means.
- prof. Ing. Václav Hlaváč, CSc. for his support when it was needed the most.
- My family, for being with me no matter what.
- Ing. Artem Kandaurov for countless inspiring discussions about numerous scientific topics.
- Bc. Victoria Marinyuk for her comprehensive care during the most painful stages of writing.

This research was supported by the Grant Agency of the Czech Republic (GAČR) with the grant no. 15-22731S titled "Symbolic Regression for Reinforcement Learning in Continuous Spaces" and by the Grant Agency of the Czech Technical University in Prague, grant no. SGS16/228/OHK3/3T/13 titled "Knowledge extraction from reinforcement learners in continuous spaces".

## Declaration

I hereby declare I have written this doctoral thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, 31.08.2021

# Abstract

Reinforcement Learning (RL) algorithms can optimally solve dynamic decision and control problems in engineering, economics, medicine, artificial intelligence, and other disciplines. However, state-of-the-art RL methods still have not solved the transition from a small set of discrete states to fully continuous spaces. They have to rely on numerical function approximators, such as radial basis functions or neural networks, to represent the value function or policy mappings. While these numerical approximators are well-developed, the choice of a suitable architecture is a difficult step that requires significant trial-and-error tuning. Moreover, numerical approximators frequently exhibit uncontrollable surface artifacts that damage the overall performance of the controlled system.

Symbolic Regression (SR) is an evolutionary optimization technique that automatically, without human intervention, generates analytical expressions to fit numerical data. The method has gained attention in the scientific community not only for its ability to recover known physical laws, but also for suggesting yet unknown but physically plausible and interpretable relationships. Additionally, the analytical nature of the result approximators allows to unleash the full power of mathematical apparatus.

This thesis aims to develop methods to integrate SR into RL in a fully continuous case. To accomplish this goal, the following original contributions to the field have been developed.

(i) Introduction of policy derivation methods. Their main goal is to exploit the full potential of using continuous action spaces, contrary to the state-of-the-art discretized set of actions.

(ii) *Quasi-symbolic policy derivation (QSPD)* algorithm, specifically designed to be used with a symbolic approximation of the value function. The goal of the proposed algorithm is to efficiently derive continuous policy out of symbolic approximator. The experimental evaluation indicated the superiority of *QSPD* over state-of-the-art methods.

(iii) Design of a symbolic proxy-function concept. Such a function is successfully used to alleviate the negative impacts of approximation artifacts on policy derivation.

(iv) Study on fitness criterion in the context of SR for RL. The analysis indicated a fundamental flaw with any other symmetric error functions, including commonly used mean squared error. Instead, a new error function procedure has been proposed alongside with a novel fitting procedure. The experimental evaluation indicated dramatic improvement of the approximation quality for both numerical and symbolic approximators.

(v) *Robust symbolic policy derivation (RSPD)* algorithm, which adds an extra level of robustness against imperfections in symbolic approximators. The experimental evaluation demonstrated significant improvements in the reachability of the goal state.

All these contributions are then combined into a single, *efficient SR for RL (ESRL)* framework. Such a framework is able to tackle high-dimensional, fully-continuous RL problems out-of-the-box. The proposed framework has been tested on three benchmarks: pendulum swing-up, magnetic manipulation, and high-dimensional drone strike benchmark.

**Keywords:** reinforcement learning, optimal control, function approximation, evolutionary optimization, symbolic regression, robotics, autonomous systems

# Abstrakt

Algoritmy posilovaného učení (RL) umí optimálně řešit problémy dynamického rozhodování a řízení např. v technických disciplínách, ekonomice, medicíně a umělé inteligenci. Ani nejnovější metody RL ale dosud nepřekročily hranici mezi malými prostory diskrétních stavů a spojitými prostory. K reprezentaci užitkové funkce a řídící strategie využívají tyto algoritmy numerické funkční aproximátory, např. ve formě RBF funkcí nebo neuronových sítí. I když numerické aproximátory jsou dobře prostudovanou oblastí, výběr vhodného aproximátoru a jeho architektury je velmi obtížným krokem, který vyžaduje ladění metodou pokus-omyl. Navíc, numerické aproximátory díky své structuře skoro vždy obsahují tzv. artefakty, které mohou uškodit kvalitě řízení kontrolovaného systému.

Symbolická regrese (SR) je evoluční optimalizační metoda, jejímž cílem je nalézt symbolický popis funkce, která co nejpřesněji modeluje trénovací data, a to automaticky, bez zásahu člověka. Tato metoda získala pozornost vědecké komunity nejenom pro její schopnost nalézt přesný zápis známých fyzikálních zákonů, ale i pro schopnost popsat jednoduchou a interpretovatelnou formou jiné složité závislosti v datech. Navíc, analytická podstata řešení umožňuje použití matematického aparátu.

Cílem dizertace je vývoj metod pro integraci symbolické regrese do posilovaného učení v případě kompletně spojitých úloh. Tohoto cíle se podařilo dosáhnout prostřednictvím následujících původních příspěvků do oboru.

(i) Zavedení metod pro odvozování řídicích strategií. Tento přístup se snaží plně využívat potenciálu spojitých akčních prostorů, což jej odlišuje od současných metod, které pracují s diskretizovanými akcemi.

(ii) Návrh *QSPD* algoritmu (Quasi-symbolic policy derivation algorithm) určeného speciálně pro použití v kombinaci se symbolickým aproximátorem užitkové funkce. Cílem navrženého algorithmu je efektivní odvozování spojité řídicí strategie z symbolického aproximátoru. Experimentálně bylo ověřeno, že *QSPD* překoná nejlepší současné metody.

(iii) Návrh konceptu symbolické proxy-funkce. Tato funkce byla s úspěchem využita k odstranění negativního vlivu aproximáčních artefaktů na odvozenou řídící strategii.

(iv) Studie fitness kritéria v kontextu symbolické regrese pro posilované učení. Tato analýza ukázala zásadní nedostatky běžně používané střední kvadratické chyby a jiných symetrických funkcí. Byla navržena nová chybová funkce spolu s novou metodou pro trénování/učení modelu. Experimentální ověření potvrdilo výrazné zlepšení kvality řešení pro numerické a symbolické aproximátory.

(v) Návrh *RSPD* algoritmu (Robust symbolic policy derivation algorithm), který nabízí vyšší robustnost vůči nedokonalostem symbolického aproximátoru. Experimentální ověření ukázalo zásadní zvýšení úspěšnosti dosažení cílového stavu.

Společně byly všechny tyto příspěvky zkombinovány do jednoho výsledného obecného frameworku, který provádí symbolickou regresi pro posilované učení (*ESRL* - efficient SR for RL). Bez nutnosti dalších úprav tento framework zvládá řešení vysoce dimenzionálních, kompletně spojitých problémů. Řešení bylo testováno na třech úlohách: vyšvihnutí inverzního kyvadla, magnetická manipulace a vysoce dimenzionální útok dronem.

**Klíčová slova:** posilované učení, optimální řízení, approximace funkcí, evoluční optimalizace, symbolická regrese, autonomní systémy

# Contents

# Figures

# Tables

# Chapter 1

# Introduction and Specification of the Research Questions

## 1.1 Symbolic regression for reinforcement learning in continuous spaces

We live in a very interesting world: a world of information, constant optimization, goal-driven solutions, and industrialized Artificial Intelligence (AI). There are quite a few fields that are not touched by the AI (yet). We, humanity, are slowly but steadily learning to rely on AI decisions and, simultaneously, improving our ability to describe our needs to machines. Cooperation of that kind is phenomenal and is gradually making its way into the history books as one of the key components of modern civilization.

One of the hot AI topics nowadays is Reinforcement Learning (RL). Over the last decade, there were many sci-fi looking titles involving RL ([1], [2], [3], [4], [5] and many more). Most of these achievements sound more like magic rather than science (which, according to Arthur C. Clarke [6], is quite normal).

However, our new magic is built on top of the well-established mathematical foundation - approximation and optimization. "How to approximate?" and "optimize what?" are two key questions when it comes to the implementation of merely any AI task. RL is no exception. Fortunately, the answer to "optimize what?" is clear and based on the biological foundation, developed into a formalized, powerful and proven framework. Such a framework fully covers the discrete case, where a limited number of system states exists. The solution for that case can be represented as a complete table of optimal actions for any given circumstances (frequently, as a table of epic sizes). Unfortunately, the computation of such a table is subject to the curse of dimensionality. One of the obvious ways to alleviate this drawback is by using approximations. Right after that decision, a new question arises: "How to approximate?"

The current prevalent vector of approximate RL research lies in using neural networks and parallel computing. Additionally, many papers discover the landscape of improvement techniques, e.g., efficient hyper-parameter tuning [7], sampling strategies [8], and incorporating a priori knowledge into RL [9], etc. Despite many successes mentioned above, the vast majority of ongoing research topics accept huge computational requirements as an inevitable evil. In addition, another big drawback of the current research vector is the close-to-zero interpretability of the resulting policy. However, despite these challenges, current methods perform well on large but still discrete spaces.

When moving into the realm of continuous tasks, an approximation is no longer just a desirable option: it is a necessity. Even for low-dimensional tasks, it is no longer possible to map all the states into the decision table. A good answer to the question "how to approximate" then becomes critical. One of the possible candidates is Symbolic Regression (SR) due to the following benefits:

- Approximator is a mathematical formula, which allows analysis and interpretation.
- Due to the stochastic and generative nature of symbolic regression, the computational complexity grows slowly with additional dimensions.
- Symbolic nature allows to employ the whole mathematical apparatus at any stage.
- Known physics can be organically embedded into the learning process.

Despite these outstanding benefits, SR is not inherently suitable for RL as a plug-and-play technique, as will be demonstrated later. There are many challenges to be solved before it can become a viable and competitive technique for the task. Altogether, it results in the following goal of the thesis, split into four research questions:

Develop methods to integrate symbolic regression into reinforcement learning in continuous spaces.

**RQ 1.** How to use a symbolic approximator efficiently? What benefits could be achieved?

**RQ 2.** Is it possible to fit a value function? How to build a minimum valuable product?

**RQ 3.** What criterion should be optimized?

**RQ 4.** How can symbolic regression be integrated into reinforcement learning efficiently?

## 1.2 Thesis structure

The thesis is composed of the papers presented in Appendix C-Publications, as well as from the novel never published material based on the authors' original ideas and their implementations. Presented papers form the second and the third chapters of the thesis and a bigger part of Appendices A-Benchmarks and B-SNGP.

The high-level pattern of the thesis is following. Each chapter assumes the existence and availability of some components, e.g., ideal numerical approximation, dynamics function, etc. For each consecutive chapter, these assumptions are relaxed. Assumptions are presented below and duplicated in corresponding epigraphs for better readability.

The rest of the current chapter is dedicated to necessary preliminaries, used notation, related work, and definition of the scope. Chapters 2-5 have the following structure:

| Chapter | Assumptions | Content | Research question |
|---|---|---|---|
| 2 | Given: symbolic optimal V-function, reward function, dynamics function | Policy derivation methods in general, efficient policy derivation from symbolic approximation. | **RQ 1.** |
| 3 | Given: numerically-approximated optimal V-function, reward function, dynamics function | First attempts to compute simplified symbolic regression of a V-function. | **RQ 2.** |
| 4 | Given: few samples from optimal V-function, reward function, dynamics function | Analysis of a fitness criterion and its influence on a result policy. | **RQ 3.** |
| 5 | Given: dynamics and reward functions | Combination of the achieved results into efficient SR for RL framework. | **RQ 4.** |

Chapter 6 then summarizes the thesis. Additionally, there are four appendices. Appendix A-Benchmarks contains a description and necessary mathematics for the used benchmarks. Appendix B-SNGP describes in details Single Node Genetic Programming (SNGP) - symbolic regression technique used throughout the thesis. Appendix C-Publications provides a list of authors' papers. Finally, used acronyms are collected in Appendix D-Acronyms.

## 1.3 Reinforcement learning preliminaries

Reinforcement Learning algorithms provide a way to solve dynamic decision-making and control problems [10], [11], [12]. An RL agent interacts with the system to be controlled by measuring its state and applying actions according to a certain policy. After applying an action, the agent receives a scalar reward related to the immediate performance. The goal is to find an optimal policy, which maximizes the cumulative reward.

The available RL algorithms can be broadly classified into critic-only, actor-only, and actor-critic methods [13]. Critic-only methods first find the value function (V-function) and then derive an optimal policy from this value function. In contrast, actor-only methods search directly in the policy space. The two approaches can be combined into actor-critic architecture, where the actor and critic are both represented explicitly and trained simultaneously. Each class can be further divided into model-based and model-free algorithms. In the model-based scenario, a system model is used during learning or policy derivation. The system model may be stochastic or deterministic. The simplest possible form of RL is then deterministic, model-based, and either critic-only or actor-only. Actor-only design requires a computationally expensive policy evaluation step at each iteration of the learning process. In a continuous task, each of these steps consists of fitting an approximation of the V-function for a given policy. In other words, actor-only design in continuous space requires building two kinds of approximators simultaneously, for policy function and V-function, respectively. Thus, the deterministic, model-based, and critic-only version can be considered the simplest form of RL. For that reason, it is frequently used throughout the paper since it allows a more precise root-cause analysis of emergent challenges.

The typical learning process, depicted in Figure 1.1, consists of three steps:

1. Data collection – using a model of the system or the system itself, samples in the form $(x_k, u, x_{k+1}, r_{k+1})$ are collected. Here, $x_k$ is the system state, $u$ is the control input (action), $x_{k+1}$ is the state that the system reaches from state $x_k$ after applying action $u$, $r_{k+1}$ is the immediate reward for that transition.
2. Computation of the optimal V-function – based on the samples, an approximation of the V-function is learned, which for each system state predicts the cumulative long-term reward obtained under the optimal policy.
3. Policy derivation – based on the computed V-function, the policy is derived at each sampling time (or simulation step) so that the system can be controlled.

### 1.3.1 Notation

Define an $n$-dimensional state space $\mathcal{X} \subset \mathbb{R}^n$, and $m$-dimensional action space $\mathcal{U} \subset \mathbb{R}^m$. The model is described by the state transition function $x_{k+1} = f(x_k, u)$, with $x_k, x_{k+1} \in \mathcal{X}$ and $u \in \mathcal{U}$. The reward function $\rho(\cdot)$ assigns a scalar reward $r_{k+1} \in \mathbb{R}$ to the state transition from $x_k$ to $x_{k+1}$:

$$\begin{aligned} x_{k+1} &= f(x_k, u) \\ r_{k+1} &= \rho(x_k, u, x_{k+1}) \end{aligned} \tag{1.1}$$

The reward function $\rho(\cdot)$ may depend on less variables, as discussed further in Section 1.3.3.

The value function (V-function), which describes the best possible value of the objective is as a function of the state $x$. It can be computed by solving the Bellman equation:

$$V(x) = \max_u [\rho(x, u, f(x, u)) + \gamma V(f(x, u))] \tag{1.2}$$

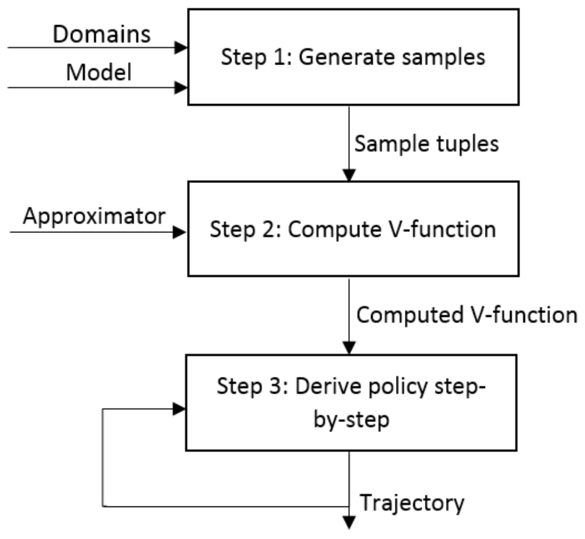

**Figure 1.1:** Typical workflow of model-based critic-only reinforcement learning.

where $\gamma$ is the discount factor (a user-defined parameter), which lies within $[0, 1)$ interval.

Define $\hat{V}$ as optimal V-function. The policy $\pi$ is the mapping:

$$\pi : \mathcal{X} \to \mathcal{U} \tag{1.3}$$

and the optimal policy corresponding to $\hat{V}(x)$ is given by:

$$\pi(x) \in \underset{u}{\operatorname{argmax}} \left[\rho(x, u, f(x, u)) + \gamma\hat{V}(f(x, u))\right] \tag{1.4}$$

## 1.3.2 Fuzzy V-iteration algorithm

There are several methods to approximate the V-function for continuous state spaces. Throughout the thesis, the fuzzy V-iteration algorithm is used [14] due to the multiple benefits. It is guaranteed to converge (under certain conditions) and the fuzzy approximator allows to interpret the values of each fuzzy core directly as the V-function value. The learning process can be briefly described as follows. Define a set of samples $S = \{s_1, s_2, \ldots, s_N\}$ on an equidistant grid in $\mathcal{X}$. The number of samples per dimension is described by vector $B = [b_1, b_2, \ldots, b_n]^T$ with the total number of samples $N = \prod_{i=1}^{n} b_i$. Further define a vector of fixed triangular basis functions $\phi = [\phi_1(x), \phi_2(x), \ldots, \phi_N(x)]^T$ where each $\phi_i(x)$ is centered in $s_i$, i.e., $\phi_i(s_i) = 1$ and $\phi_j(s_i) = 0, \forall j \neq i$. The basis functions are normalized so that $\sum_{j=1}^{N} \phi_j(x) = 1, \forall x \in \mathcal{X}$. Finally, define a parameter vector $\theta = [\theta_1, \theta_2, \ldots, \theta_N]^T, \theta \in \mathbb{R}^N$. The V-function approximation is defined as:

$$\hat{V}(x) = \theta^T\phi(x) \tag{1.5}$$

The fuzzy V-iteration is:

$$\theta_i^{j+1} \leftarrow \max_{u \in U} \left[r(s_i, u) + \gamma \left(\theta^j\right)^T \phi\left(f(s_i, u)\right)\right], i = 1, 2, ..., N \tag{1.6}$$

where $U$ is a discrete control input values $U = \{u_1, u_2, \ldots, u_N\}$ drawn from $\mathcal{U}$. The iteration terminates when the following convergence criterion is satisfied:

$$\epsilon \geq ||\theta^j - \theta^{j-1}||_\infty \tag{1.7}$$

where $\epsilon$ is a convergence threshold.

### 1.3.3 Reward function design

The reward function $\rho(\cdot)$ is usually designed by the experimenter. It provides a scalar reward signal to evaluate immediate performance. Additionally, the reward function is utilized during policy derivation. Currently, there are no best practices of the reward function design in the continuous-valued RL domain. Reward function, typically, may depend on the following parameters - $x$, $u$, and $f(x, u)$. Omitting some of these terms may lead to the consequences listed below:

- Omitted $x$, the reward function is defined as $\rho(f(x, u), u))$: during learning, the immediate reward propagates from the goal state to other states iteratively. Typically, an extremum of the reward function lies in a goal state $x_{des}$. If there exists such $u$ that $f(x_{des}, u) = x_{des}$, the reward at the goal state would be constant, making $V(x_{des}) = max(V(\cdot)) = \rho(f(x, u), u)$. However, if such action does not exist or cannot be found with perfect precision, the reward would not be constant, which causes V-function to drift along the value axis.

- Omitted $u$, reward function is defined as $\rho(x, f(x, u))$: such definition results in the impossibility to select a less costly action in a situation where several inputs lead to the same state. Such situations frequently occur during policy derivation.

- Omitted $f(x, u)$ and $u$, reward function is defined as $\rho(x)$: during policy derivation renders the reward function redundant. At every time step of the policy derivation

$$\begin{aligned} \underset{u \in U}{\operatorname{argmax}} \left[\rho(x) + \gamma \hat{V}(f(x, u))\right] \equiv \\ \underset{u \in U}{\operatorname{argmax}} \left[\textbf{const} + \gamma \hat{V}(f(x, u))\right] \end{aligned} \tag{1.8}$$

For this thesis, the full form of the reward function is used, namely $\rho(x, u, f(x, u)$, unless stated otherwise. To simplify the notation, omitted terms are not included in function definition and term of state space is placed first, e.g., $\rho(f(x, u), u) \equiv \rho(0 \cdot x, u, f(x, u))$.

## 1.4 Symbolic regression preliminaries

Symbolic regression (SR) is an evolutionary optimization technique that automatically, without human intervention, generates analytical expressions to fit numerical data [15]. The method has gained attention in the scientific community for its ability to recover known physical laws, but also to suggest yet unknown but physically plausible and interpretable relationships [16, 17].

More specifically, given a set of data $(\varphi, y)$, where $\varphi$ is a vector of input variables (regressors) and $y$ is an output (regressand), SR searches for a function $\hat{f}(\varphi, \theta)$ that fits the data as well as possible. Both the structure of $\hat{f}$ and its parameters are subject to search. This is achieved by evolving a population of functions until an individual with sufficient representational capabilities is found. The individuals in the population are represented as trees, consisting of *nodes* connected by *branches*. Every node in a tree can represent either a function from some predefined set of functions $\mathcal{F}$ or a terminal from the terminal set $\mathcal{T}$. The set $\mathcal{F}$ also contains operators such as addition, multiplication, time-shift, differentiation, etc. The available functions and the operators are collectively referred to as *function patterns*. The terminal can be either a numerical constant or a variable. When a node represents a function, the number of branches created at this node equals the number of arguments the function requires. The representational capability of an individual in the population is measured by its *fitness*, a numerical measure of the error between $\hat{f}(\varphi)$ and $y$ in some metric, such as the sum of squared errors or a correlation measure between the model output and the data.

The search process starts by creating an initial population of random individuals. Some of the individuals (parents) are selected and modified by evolutionary operators, typically *crossover* and *mutation*. The crossover operation interchanges part of a parent individual with another parent. A random node is selected in each parent, and the *sub-tree* connected to this node is swapped with that of the other parent (the sub-tree contains all the nodes and branches connected to the selected node down the tree). The mutation operation is similar to crossover but involves only one parent. A random node in the parent is selected, and the sub-tree connected to this node is replaced by a randomly created sub-tree. When the new individual (child or offspring) has higher fitness than its parent, it replaces the parent in the new generation. This process keeps repeating over many generations until sufficiently fit individuals are found or until a predefined number of generations is reached. Given the final population evolved by the SR procedure, one usually has to make a trade-off between the complexity of the models and their performance. This involves Pareto optimization and may also include a test for the physical interpretability of the model.

### 1.4.1 Single node genetic programming overview

Genetic programming (GP) belongs to the methods frequently used to solve the symbolic regression problem. Besides the standard Koza's tree-based GP [15], many other variants have been proposed, such as Grammatical Evolution [18] which evolves programs whose syntax is defined by a user-specified grammar, Gene Expression Programming [19] that evolves linear chromosomes that are expressed as tree structures through a genotype-phenotype mapping or graph-based Cartesian GP (CGP) that uses a linear integer representation for expressing programs in the form of a directed graph [20].

For the thesis, a Single Node Genetic Programming (SNGP) [21], [22] is used. Below, a brief introduction of the method is provided. A full description of SNGP can be found in Appendix B-SNGP.

SNGP is a graph-based GP method that evolves a population of individuals, each consisting of a single program node. The node can be either a terminal, i.e., a constant or a variable in case of the symbolic regression problem, or a function chosen from a set of functions defined for the problem at hand. Importantly, the individuals are not entirely distinct, they are interlinked in a graph structure similar to that of CGP, so some individuals act as input operands of other individuals.

Formally, a SNGP population is a set of $L$ individuals $M = \{m_0, \ldots, m_{L-1}\}$, with each individual $m_i$ being a single node represented by the tuple $m_i = \langle u_i, f_i, Succ_i, Pred_i, O_i \rangle$, where

- $u_i \in T \cup F$ is either an element chosen from a function set $F$ or a terminal set $T$ defined for the problem;
- $f_i$ is the fitness of the individual;
- $Succ_i$ is a set of successors of this node, i.e. the nodes whose output serves as the input to the node;
- $Pred_i$ is a set of predecessors of this node, i.e. the nodes that use this node as an operand;
- $O_i$ is a vector of outputs produced by this node.

Typically, the population is partitioned so that the first $L_{term}$ nodes, at positions 0 to $L_{term}-1$, are terminals, followed by function nodes. Importantly, a function node at position $i$ can use as its successor (i.e., the operand) any node that is positioned lower down in the population relative to the node $i$. This means that for each $s \in Succ_i$ there is $0 \leq s < i$. Similarly, predecessors of individual $i$ must occupy higher positions in the population, i.e. for each $p \in Pred_i$ there is $i < p < L$. Note that each function node is, in fact, a root of a program tree that can be constructed by recursively traversing its successors towards the leaf terminal nodes.

An operator called *successor mutation* (*smut*), proposed in [21], is used to modify the population. It takes an individual and replaces one of its successors by a reference to another chosen individual of the population, making sure that the constraint imposed on successors is satisfied. Nodes to be mutated are chosen using a *depthwise* selection proposed in [23], which takes into account both the quality and depth of nodes. Output values of the mutated node and all nodes higher up in the population affected by the mutation operation are recalculated. In addition, the predecessor lists of all affected nodes are updated accordingly.

Finally, the population is evolved using a local search-like procedure. In each iteration, a new population is produced by the *smut* operator, which is then accepted for the next iteration if it is no worse than the original population.

The evolution is carried out via a hill-climbing mechanism using a *smut* operator and an acceptance rule, which can have various forms. Here, the new population is accepted if and only if the best fitness in the population has not been worsened by the mutation operation. Otherwise, the modifications made by the mutation are reversed.

## 1.5 Related work

One of the first contributions to the problem belongs to [24]. In this work, genetic programming was used to generate a structured search space for the Q-learning algorithm [10]. This early contribution showed the potential of the combination, resulting in significant learning speed improvement. However, no following work attempted to extend the achieved results to continuous spaces. Consequent work [25] encoded state-action pairs of the Q-table as leaves of a tree. In combination with genetic programming, it successfully fitted discrete Q-function for the well-known maze problem. This approach used a temporal difference algorithm as a driver for learning. Despite great results, this approach works only for discrete spaces. Consequently, it is not scalable due to the curse of dimensionality.

In [26], the optimal policy was searched directly. Evaluation of the policy was done by means of Monte-Carlo simulations. As a result, this work produced interpretable discrete policies encoded as *if-then* rules. However, the overall performance of such policies was reported to be highly suboptimal. A year later, these poor results were confirmed in [27]. This work combined genetic programming with reinforcement learning, similarly to [26]. The main focus of the research was genetic programming. However, in this work, continuous action spaces were used. Unlike [26], the results were not reportedly poor but questionable in terms of optimality.

Another interesting approach was proposed in [28]. In this work, genetic programming was used to construct meta-actions - sequences of actions, which then were used in the standard Q-learning. While this approach does not produce interpretable symbolic policy, it may be considered as a supplementary method to speed up the learning phase. [29] followed the same philosophy and studied genetically produced synthesis of features. Through this process, significant dimensionality reduction was achieved, which speeded up the learning process. Another supplementary usage of genetic programming was introduced in [30]. This work studied the possibility of producing genetically optimized reward functions. Significant changes in convergence speed were reported. The further paper from the same authors [31] provided a formal analysis of the convergence speed. The work is closely related to so-called *reward shaping* [32].

One of the first attempts to couple evolution with a V-function was introduced in [33]. In this work, genetic programming served as an optimization technique. V-function of the task was approximated by parametric tiles, which served as basis functions. Genetic programming produced a population of tiles of different sizes. Approximate V-iteration was then employed on these tiles. While the overall approach was novel, the optimality of the produced solutions is questionable.

[34] introduced one of the first attempts to represent a policy function as a closed-form algebraic formula. It was based on the explicit generation of algebraic formulas with predefined length, using predefined functions and operators. Additionally, it was the first attempt to work with continuous states. This work was intended to serve as a proof of concept, and the concept was successfully proved. Despite suboptimal performance and scaling issues, the resulting policies were working for continuous states and were interpretable.

In [35], continuous control tasks were solved by means of genetic programming. The fitness function for the optimization was operated through explicit simulations. Unlike [26], this work demonstrated near-optimal performance of the result continuous policy. While it is indeed a feasible approach, explicit simulations are computationally expensive and may have dimensionality-related issues. [36] relaxed interpretability in favor of optimality. This work was based on the *fuzzification* of policies produced by means of multi-gene genetic programming. Again, the fitness function was operated through explicit simulations. However, since this work was intended to be utilized by the industry, the computational budget was not a big concern. Like [35], this work was never tested on high-dimensional benchmarks. One of the possible reasons for it is that explicit simulations may become prohibitive as the dimensionality of the task grows.

One of the first experiments to compute symbolic V-function out of known samples of the given V-function was conducted in [37]. Found algebraic approximator of the V-function was then used to compute a discrete Q-table. That approach was reportedly suboptimal w.r.t. the original V-function. Additionally, it was never tested on continuous tasks.

In [38], approximation of the policy function was constructed on the basis of samples from the known policy. This work restricted itself to a specific type of policy, known as *bang–bang controller*. The result continuous policy function performed equally well in comparison to the provided optimal policy. Despite the fact that this work was tested exclusively on a fully actuated system and still required known policy, the produced trajectories were smooth. It revealed the potential of using the proposed method in industrial applications.

Algebraic formulas are not the only option to encode interpretable policies. In [39], the policy function was fitted in the form of a program based on sequences of actions provided by the expert. Additionally, this work attempted to build a symbolic regressor of the policy function. While policies encoded as programs performed relatively well, symbolic regression of the policy function experienced catastrophic failure in terms of the derived policy. This research was focused on the interpretability aspects of the result approximator, and its main drawback is questionable scalability as the dimensionality of tasks grows.

In [40], *Cartesian genetic programming* [41] was employed to evolve graph-based programs for Atari games. Policies were encoded as sequences of discrete actions, while feedback consisted of the result score. This work revealed the tendency to produce oversimplified policies, mainly due to the unsuccessful escaping from local optima. It resulted in suboptimal performance, which failed to match human players in most of the games.

[42] employed symbolic regression to approximate system dynamics in the context of reinforcement learning. In addition to the good prediction quality, it discovered one of the key benefits of symbolic regression - the ability to generalize complex functions from quite a few samples. [43] provided further evidence for that claim. A year later, [44] applied the same principles in order to find a symbolic approximation of the dynamics of a quadrotor. [45] extended this idea by using symbolic regression inside the V-iteration algorithm, in addition to the symbolic approximation of the dynamics. It was done by using symbolic regression at each iteration of the V-function. Each symbolic candidate at each iteration was tested via explicit simulations; the best-performing candidate was selected to be a result approximator. While this work showed the potential of finding well-performing symbolic V-functions, its main drawback is computational expensiveness - there were reported thousands of symbolic V-function candidates. Additionally, only the discrete action space was considered.

Another attempt to search for a symbolic policy function was presented in [46]. The proposed method consists of solving optimal control tasks by generating symbolic policies, where the fitness function is computed through simulation of trajectories. Found symbolic policies were tested on simple benchmarks, resulting in near-optimal performance. While this work demonstrated the existence of near-optimal interpretable policies, evaluation of such policies through simulations may quickly become prohibitive as the dimensionality of the task grows.

[47] used deep reinforcement learning [48] to learn a policy. Then, it attempted to re-approximate this policy by means of genetic programming. In this work, genetic programming produces rather an algorithm, not an algebraic equation. The overall performance of such a solution was reported to be near-optimal. The main drawback, however, comes from the encoding method used to produce a policy. Such a method restricts itself exclusively to tasks with discrete actions. This work [49] followed the similar logic. The proposed method consists of re-approximating the policy built by means of a neural network with a decision tree. Symbolic regression was responsible for synthesizing features for such a tree. While the results were reported to be near-optimal, the method itself is restricted to discrete actions.

In [50], authors extended their previous idea [39] of fitting a policy function in the form of programs. Instead, the population of such programs was fitted. Then, the whole population was bonded together using fuzzy logic. This step decreased the interpretability of the resulting policy but raised the performance to the near-optimal level. This work was tested exclusively on benchmarks, for which *bang–bang controller* is an optimal choice. Both scalability and suitability for other types of tasks remain questionable.

[51] and [52] studied the combination of symbolic regression and optimal control. These papers attempted to find a continuous-time policy but for only one trajectory. No attempts to find generalized policy functions were reported. Here, symbolic regression was supplementary to the novel optimal control method. These results were extended in [53]. The later work studied the possibility to encode a given optimal policy as a Cartesian genetic programming task. The resulting policy then was represented in the form of an interpretable program. The proposed method was focused on the interpretability aspect, not on the optimality.

[54] attempted to build a symbolic policy for a general Markov decision process. This work distinguishes itself from others by focusing on threshold-based policies. The symbolic policy was fitted to the samples, computed numerically beforehand. The results were reported to be suboptimal even on a relatively simple benchmark.

In [55], the efficient methods of building symbolic regression for dynamic targets were studied. Values from the changing in time V-function were used as the dynamic targets. This work provided several techniques to do it efficiently and demonstrated that symbolic regression is capable of fitting V-functions in terms of the mean squared error. However, the fitted V-functions were never tested from the reinforcement learning perspective. In [56], the effect of adding geometrical constraints to symbolic regression evolution was studied. It was done by modifying mean square error fitness with a novel, two-phase procedure. The proposed method was applied to fitting a symbolic model of system dynamics with reportedly good results.

In a very recent study [57], symbolic policies were constructed by means of a novel method named Deep Symbolic Policy (DSP), based on the sequential use of deep neural networks. The numerical policy was constructed beforehand using another neural network and then used as a reference. Moreover, this work attempted to fit a policy function by means of symbolic regression. While DSP solution performed near-optimally on all benchmarks, the symbolic regression approach suffered a catastrophic failure. The paper claims: "there is an objective function mismatch between the regression objective (minimizing prediction error) and the control objective (obtaining high reward in the environment). In other words, small errors in regression do not necessarily correspond to small decreases in performance when evaluated in the environment."

## 1.6 Definition of the scope

From the RL point of view, this thesis is focused on deterministic, continuous model-based, fully-observable tasks, with continuous state space, continuous action space, and continuous reward function. However, the majority of the proposed methods are designed in a non-restrictive way. In other words, the proposed methods are not restricted to that setting, and many of them could be trivially extended to other cases. From the SR point of view, the whole thesis is based on one of the variants of SNGP, described in Appendix B-SNGP. Again, the proposed methods are not restricted to that particular technique; it is rather used as a customizable platform. Moreover, most of the results are directly applicable to other approximation techniques, e.g., to neural networks, unless explicitly stated otherwise.

# Chapter 2

# Policy Derivation Methods

Assume we have optimal symbolic V-function, reward function, and dynamics function. How to use it efficiently?

*Author at early stages of research*

In this chapter, the critic-only, model-based, and deterministic variant of RL in continuous state and action spaces is considered. For the methods developed, it is irrelevant whether the system model is available a priori or learned online. The policy derivation step only is addressed, assuming that an approximation of the true unknown V-function has already been computed.

Policy derivation can be understood as a hill-climbing process: at each time step, the agent searches for the control input that leads to a state with the highest value given by the right-hand side (RHS) of the Bellman equation. An advantage of this control law is its inherent stability – the value function is analogous to the control Lyapunov function [58, 59]. However, direct policy derivation from the V-function suffers from several problems:

- **Computational inefficiency**. The most common approach to dealing with a continuous action space is to discretize it into a small number of actions, compute the value of the Bellman equation RHS for all of them, and select the one that corresponds to the largest value [10] [60]. The number of possible discrete actions grows exponentially with the input space dimension, and so does the computational complexity of this method.

- **Insufficient smoothness of the V-function**. The above hill-climbing process is adversely affected by the approximate nature of the V-function, which has been observed, e.g., in [61]. A typical approximation by means of basis functions exhibits artifacts that can lead to oscillations, as illustrated in the left column of Figure 2.1. The term "insufficient smoothness" is used for the reference to that effect, without relying on the exact mathematical definition of smoothness.

- **Discrete-valued control input**. The use of discrete actions in combination with insufficient smoothness leads to steady-state errors, as shown in the right column of Figure 2.1. In the long run, the steady-state error can induce big losses in terms of the overall performance.

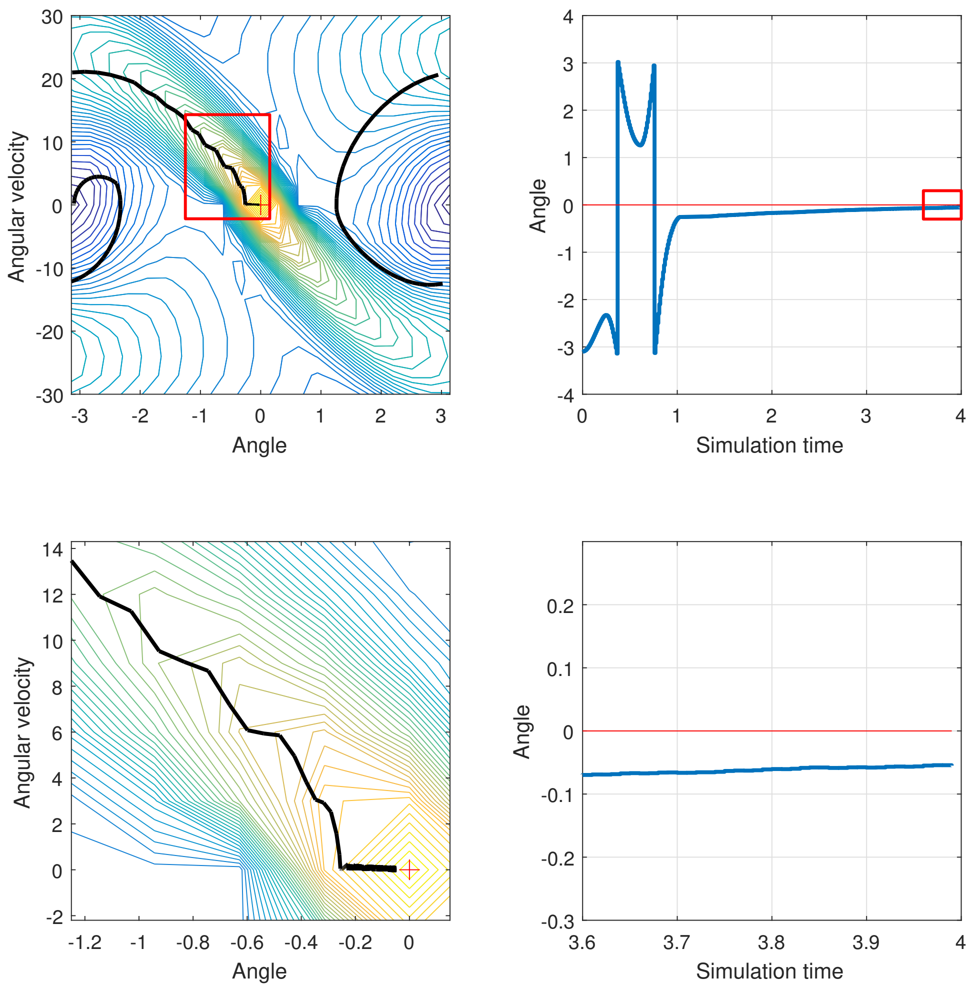

**Figure 2.1:** A sample state trajectory was obtained by simulating the pendulum swing-up task. The bottom plots show an enlarged view of the areas indicated in the upper plots. The wiggly state trajectory (superimposed on the contours of the Bellman equation RHS) and the extremely slow convergence to the desired position result from the insufficient smoothness of the V-function approximation in combination with the use of discrete actions.

The aim of this chapter is to alleviate the above problems. In addition to the policy derivation methods themselves, symbolic regression smoothing is introduced to mitigate the V-function smoothness issue. Additionally, to enable the computationally efficient use of continuous actions, a method based on non-linear optimization is proposed.

## 2.1 Related work

The problem of deriving policies for continuous state and action spaces in critic-only methods has not been sufficiently addressed in the literature. The most common approach is to discretize the action space, compute the RHS of the Bellman equation for all the discrete actions, and select the action that corresponds to the largest value. One of the earliest references to this approach is [62]. The drawbacks of this method were discussed in the previous section.

Another similar approach is based on sampling [63, 64]. Using Monte-Carlo estimation, this approach can find a near-optimal action without resorting to exhaustive search over the discretized action space. However, for a good performance, this method requires a large number of samples and, therefore, is computationally inefficient.

An alternative method would be *policy interpolation* [14], which is based on computing the control actions off-line for a pre-selected set of states and then interpolating these actions online. While computationally less involved, this method does not give any closed-loop stability guarantees and can suffer from severe interpolation errors, especially in constrained problems. Therefore, policy interpolation is not considered in this work.

A different approach relies on translating the continuous action selection step into a sequence of binary decisions [65, 66]. Each decision whether to decrease or increase the control action eliminates a part of the action space. This process stops once a predefined precision is reached. There are two main drawbacks of this approach: it requires a binary code representation of each action, which is difficult to design properly, and it does not resolve the insufficient smoothness problem of the V-function.

There are several approaches to smoothing data in general, e.g., elastic mapping [67], smoothing splines [68], or approximation by neural networks [69]. However, none of them is directly applicable to RL. To the best of authors' knowledge, the only approach that has been developed to address the insufficient smoothness problem in the RL context is [61]. It is based on the concept of a smooth *proxy-function* which encodes the preference for the optimal action, while it does not have to satisfy the Bellman equation. The proxy function is derived through symbolic regression and can be used for policy derivation. This method has two limitations: it does not allow for penalizing the control action in the reward function, and it is restricted to discrete-valued actions.

## 2.2 Policy derivation methods

### 2.2.1 Baseline policy derivation

The common solution from the literature is to evaluate

$$\hat{\pi}(x) \in \underset{u \in U}{\operatorname{argmax}} \left[\rho(x,u,f(x,u)) + \gamma \hat{V}(f(x,u))\right] \tag{2.1}$$

where $U = \left\{u^1, u^2, \dots, u^M\right\}$ is a finite set of actions drawn from $\mathcal{U}$. at every sampling instant, where the maximization is computed over the same discrete action set $U$ on which $\hat{V}$ has been learned by means of fuzzy V-iteration (see Section 1.3.2). This method is used as the baseline approach.

## 2.2.2 Evaluation over a fine grid of actions

It can be beneficial to refine the action space with respect to the space used for computing the V-function. Define $A \subset \mathcal{U}$ as

$$A = A_1 \times A_2 \times \cdots A_m, \tag{2.2}$$

where each set $A_i$ contains points distributed in a suitable way (e.g., equidistantly) along the $i$th dimension of the action space. Set $A$ therefore contains all combinations of the control inputs for which the right-hand side of the Bellman equation (3.3), evaluated for the current state $x$:

$$G_x(u) = \rho(x, u, f(x, u)) + \gamma \hat{V}(f(x, u)) \tag{2.3}$$

Label $G_x$ as the *control surface.* An example of such as a surface for a two-dimensional action space is shown in Figure 2.2.

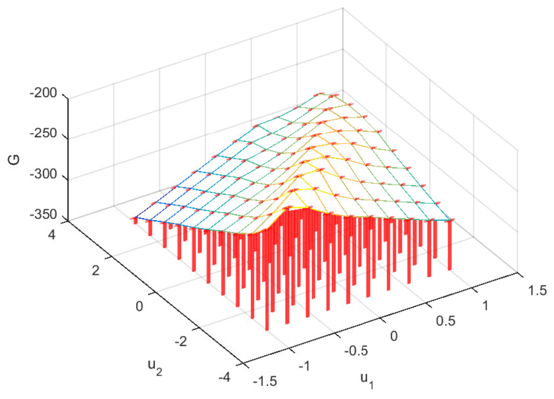

**Figure 2.2:** An illustrative example of grid evaluation over the action space for a given $x$. Each dimension of the action space is discretized in 10 points, i.e., at each time step, a total of 100 points are evaluated according to (2.3).

This fine discretization allows to control the system by applying actions more precisely, while it does not increase the computational complexity during learning.

This policy derivation step is formalized in Algorithm 1.

## 2.2.3 Chebyshev polynomial approximation

The main idea of this method is to find a smooth approximation of the control surface, such as the one shown in Figure 2.2. This smooth approximation facilitates more accurate control and helps to avoid chattering in the vicinity

**Algorithm 1:** Maximization over a fine grid of actions (in the sequel denoted as *Grid*)

```
**Input:** $f, \rho, \gamma, A, \hat{V}, x_0$
$k \leftarrow 0$
**while** *control experiment not terminated* **do**
    $u_k \leftarrow \underset{u' \in A}{\text{argmax}} \; \rho(x_k, u', f(x_k, u')) + \gamma \hat{V}(f(x_k, u'))$;
    $x_{k+1} \leftarrow f(x_k, u_k)$;
    $k \leftarrow k + 1$
**end**
**Output:** trajectory $[x_0, x_1, ...]$, $[u_0, u_1, ...]$
```

of unstable system equilibria. Chebyshev polynomials of the first kind are used for that task. Chebyshev polynomials are defined by the following recurrent relation:

$$\begin{aligned} T_0(\bar{u}) &= 0 \\ T_1(\bar{u}) &= \bar{u} \\ T_{n+1}(\bar{u}) &= 2\bar{u}T_n(\bar{u}) - T_{n-1}(\bar{u}) \end{aligned}$$

They are orthogonal to each other on the interval $[-1, 1]$ with respect to the weighting function $1/\sqrt{1-\bar{u}^2}$. To take advantage of this property, the domain of each control variable must be mapped onto the interval $[-1, 1]$. This is accomplished using the affine transformation:

$$\bar{u} = -1 + 2\frac{u - u_L}{u_H - u_L}$$

where $u_L$ and $u_H$ are the lower and upper bound of $u$ in each input dimension, respectively. The approximation by means of Chebychev polynomials can be extended to higher dimensions by using the Cartesian product of univariate polynomials. The details of this procedure can be found in [70], [71], [72]. *Chebfun* open-source package [73] is used to construct a polynomial function $P(u)$, which receives an action $u \in \mathcal{U}$ and returns the value of the approximated right-hand side of the Bellman equation. Note that this function includes the above affine transformation.

The polynomial structure of the policy approximator allows to efficiently find the maxima in (3.3) by numerically solving the set of algebraic equations obtained by equating the first partial derivatives to zero, see [74]. In some cases, the polynomial attains its maximum inside the domain $\mathcal{U}$, but in other cases, the maximum lies outside of this domain. Therefore, the boundaries must also be tested. In experimental evaluation, the *Chebfun* open-source package [73] is used, since it effectively searches for the potential extrema inside and outside the domain. The argument of the global maximum of $P(\cdot)$ on the $\mathcal{U}$ domain is then the control action sought. This policy derivation step is formalized in Algorithm 2.

**Algorithm 2:** Maximization using Chebyshev polynomials (in the sequel denoted as *Cheby*)

```
Input: $f, \rho, \gamma, A, \hat{V}, x_0$
$k \leftarrow 0$
while control experiment not terminated do
    foreach $u' \in A$ do
        $G[u'] = \rho(x_k, u', f(x_k, u')) + \gamma\hat{V}(f(x_k, u'))$;
    end
    build Chebyshev polynomial approximation $P(\cdot)$ using data $(A, G)$
    $u_k \leftarrow \underset{u' \in \mathcal{U}}{\text{argmax}}\; P(u')$
    $x_{k+1} \leftarrow f(x_k, u_k)$
    $k \leftarrow k + 1$
end
Output: trajectory $[x_0, x_1, ...]$, $[u_0, u_1, ...]$
```

## 2.2.4 Symbolic regression smoothing

To mitigate the insufficient smoothness problem, the numerical approximation of the V-function is smoothed by applying symbolic regression. This results in an analytical expression which accurately describes the V-function, while eliminating artifacts caused by the numerical approximator. In this work, the following basic operators and functions to build the analytical expressions were used: $F = \{*, +, -, square, cube, tanh\}$. Symbolic regression procedure is briefly described in Section 1.4.1, full description can be found in B.2.

The *Baseline*, *Grid* and *Cheby* algorithms can all be enhanced by using a symbolic approximation $\hat{V}^s$ of the V-function instead of the numerical approximation $\hat{V}$. In the sequel the superscript $s$ denotes symbolic functions or operations. The modified algorithms are denoted as *SR-Baseline*, *SR-Grid* and *SR-Cheby*, respectively.

## 2.2.5 Quasi-symbolic policy derivation

To reduce the computational complexity of policy derivation, the Quasi-Symbolic Policy Derivation algorithm (in the sequel denoted as *QSPD*) exploits the symbolic nature of $\hat{V}^s$. Rather than exhaustively enumerating all possible actions in a discrete action set, it applies a standard numerical optimization algorithm to the following problem:

$$u_k = \underset{u' \in \mathcal{U}}{\text{argmax}}\; R^s(x_k, u') \tag{2.4}$$

where $R^s$ stands for the right-hand side of the Bellman equation:

$$R^s(x, u) = \Big[\rho^s(x, u, f^s(x, u)) + \gamma\hat{V}^s(f^s(x, u))\Big] \tag{2.5}$$

with $\rho^s$ and $f^s$ the symbolic representations of the reward and state-transition function, respectively. In model-based RL, the reward function is always designed by the experimenter and therefore can be defined as an analytic function. However, this is often not the case with the state transition function $f$. The system dynamics is typically described in continuous time and the state transitions are generated by means of numerical integration, using e.g. Runge-Kutta methods. In such as case, the symbolic approximation $f^s$ can be obtained by means of the forward Euler method.

The trust region reflective (TRR) algorithm [75] is used for the optimization. To enhance its convergence speed, the symbolic partial derivatives $\nabla R^s(x,u)$ are provided.

$$\nabla R^s(x,u) = \left[\frac{\partial R^s(x,u)}{\partial u_1}, \frac{\partial R^s(x,u)}{\partial u_2}, \ldots \frac{\partial R^s(x,u)}{\partial u_m}\right] \tag{2.6}$$

The *QSPD* algorithm is presented in Algorithm 3.

**Algorithm 3:** Quasi-symbolic policy derivation (*QSPD*)

```
Input: $f, f^s, \rho^s, \gamma, \hat{V}^s, \nabla R^s(x,u), x_0$
$k \leftarrow 0$
while control experiment not terminated do
    $R^s(x,u') \leftarrow \left[\rho^s(x,u',f^s(x,u')) + \gamma\hat{V}^s(f^s(x,u'))\right]$
    $u_k = \underset{u' \in \mathcal{U}}{\text{argmax}}\ R^s(x_k,u')$; TRR optimization
    $x_{k+1} \leftarrow f(x_k,u_k)$;
    $k \leftarrow k+1$
end
Output: trajectory $[x_0, x_1, \ldots]$, $[u_0, u_1, \ldots]$
```

## 2.3 Experimental evaluation

The proposed methods are evaluated on four non-linear control problems: 1-DOF A.1 and 2-DOF A.2 pendulum swing-ups, and magnetic manipulation A.3 in two variants, with two and five coils, respectively. To compute $\hat{V}(x)$, the fuzzy V-iteration algorithm has been used (see Section 1.3.2 for further details). The characteristics and parameter values for each problem are listed in Table 3.1.

The performance of the algorithms is measured by the following criteria:

- Performance percentage ratio

$$P_{alg} = \frac{1}{N}\sum_{j=1}^{N}\left[p^j_{baseline}/p^j_{alg}\right] \cdot 100\%$$

with $p_{alg} = \sum_{k=1}^{T_{sim}/T_s} \rho(x_k, u_k, f(x_k,u_k))$, where the subscript $alg$ refers to the algorithm tested, $N$ is the number of runs, $T_{sim}$ and $T_s$ are the

| | 1-DOF | 2-DOF | Magman2 | Magman5 |
|---|---|---|---|---|
| Number of state dimensions | 2 | 4 | 2 | 2 |
| Number of action dimensions | 1 | 2 | 2 | 5 |
| State space, $\mathcal{X}$ | $[-\pi,\pi]\times$ $[-30,30]$ | $[-\pi,\pi]\times[-2\pi,2\pi]\times$ $[-\pi,\pi]\times[-2\pi,2\pi]$ | $[0,0.05]\times$ $[-0.39,0.39]$ | $[0,0.1]\times$ $[-0.39,0.39]$ |
| Input space, $\mathcal{U}$ | $[-2,2]$ | $[-3,3]\times[-1,1]$ | $[0,0.6]\times$ $[0,0.6]$ | $[0,0.6]\times$ $\ldots\times[0,0.6]$ |
| Samples per input dimension, $B$ | $[21,21]^T$ | $[11,11,11,11]^T$ | $[27,27]^T$ | $[27,27]^T$ |
| Discount factor, $\gamma$ | 0.9999 | 0.99 | 0.999 | 0.999 |
| Convergence threshold, $\epsilon$ | $10^{-5}$ | $10^{-5}$ | $10^{-8}$ | $10^{-8}$ |
| Refined action space size | 27 | $[11,11]^T$ | $[11,11]^T$ | $[11,\cdots,11]^T$ |
| Simulation time, $T_{sim}$ [s] | 10 | 20 | 10 | 10 |
| Sampling period, $T_s$ [s] | 0.01 | 0.01 | 0.005 | 0.005 |

**Table 2.1:** Experiment parameters

simulation time and the sampling period, respectively. The reward functions are defined such that they are negative except in the goal state, where they equal to zero. The performance measure $P_{alg}$ is therefore 100% for the baseline and larger than 100% for the algorithms that outperform the baseline.

- Runtime percentage ratio

$$T_{alg} = \frac{1}{N}\sum_{j=1}^{N}\left[t^j_{alg}/t^j_{baseline}\right]\cdot 100\%$$

where $t_{alg}$ stands for the runtime of the algorithm. Analogously to the performance, $T_{alg}$ is 100% for the baseline and smaller than 100% for the algorithms that run faster than the baseline.

- Mean nomalized distance of the final state to the goal state

$$D_{alg} = \frac{1}{N}\sum_{j=1}^{N}\|x_{des} - x^j_{end}\|$$

with $x_{des}$ the desired (goal) state, $x^j_{end}$ the final state of the $j$-th simulation run, and $\|\cdot\|$ the Euclidean norm. Every dimension of the state space is mapped onto the interval $[0,1]$ to equalize different scales.

- Mean return

$$R_{alg} = \frac{1}{N}\sum_{j=1}^{N} p_{alg}$$

Each algorithm was tested $N = 50$ times on each benchmark with randomly chosen initial states.

To construct Chebyshev polynomials, the *Chebfun* open-source package [73] were used. The Matlab 2015a `fmincon` implementation of the trust region reflective algorithm is used in *QSPD*.

The genetic programming method evolving the symbolic approximations of the V-function was run with the following control parameters setting:

- The total population size was 300. The ratio of the head partition size to the size of the tail partition was 1:2.
- The maximum number of features the evolved LASSO model can be composed of was set to 16. The maximum depth of individual features was 7.
- The number of parallel threads run in a single epoch was 3. The number of epochs was set to 50.

Multiple independent runs were carried out with symbolic regression to find the symbolic value functions $V^s$. The symbolic regression parameters did not change between the runs (except for the random seed). The value function which performed best with respect to $P_{alg}$ is presented in the results.

### 2.3.1 1-DOF swing-up

The inverted pendulum consists of a mass attached to an actuated link that rotates in the vertical plane. The available torque is insufficient to push the pendulum up in a single rotation from the majority of initial states. Instead, from a certain state (e.g., pointing down), the pendulum needs to be swung back and forth to gather energy, before it can be pushed up and stabilized. Full description alongside with technical details about the benchmark can be found in A.1. The set of discrete control inputs is $U = \{-2, -1, 0, 1, 2\}$. The control goal is to stabilize the pendulum in the unstable equilibrium $\alpha = \dot{\alpha} = 0 = x_{des}$ using desired control input $u_{des} = 0$. The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.01\,\text{s}$. This goal is expressed by the following reward function:

$$\rho(x, u, f(x, u)) = -\text{abs}(x_{des} - x)^T Q - \text{abs}(u_{des} - u) \tag{2.7}$$

with $Q = [5, 1]^T$ a weighting vector to adjust the relative penalty between the angle, angular velocity and control input and $abs(\cdot)$ function works element-wise.

The simulation results are presented in Table 3.5, with the best performance printed in bold.

The most important conclusion is that the use of symbolic regression improves the overall performance, with *SR-Cheby* yielding the best results. All the proposed approaches result in longer runtime, except for *SR-Baseline*. *Grid* and *Cheby* algorithms perform worse than the baseline solution with respect to $P_{alg}$. This is caused by the combination of insufficient smoothness of the value function and discrete actions, as illustrated in Figure 2.3. The approximation by means of basis functions produces a “ridge” in the vicinity of the goal state. The state trajectory then moves back and forth from one side of the ridge to the other, slowly converging toward the goal state. The *Baseline* algorithm employs fewer actions and is therefore forced to use actions with a larger magnitude. The *Grid* algorithm, on the contrary, employs more actions with a smaller magnitude. Therefore, it can follow the ridge more precisely, which results in a larger value of the Bellman equation RHS, but a slower overall convergence to the goal. The *Cheby* algorithm uses data provided by the *Grid* algorithm and therefore suffers from the same problem.

The *QSPD* algorithm alleviates this problem thanks to the use of a smooth symbolic approximation of the V-function in combination with continuous actions. The state trajectory is virtually ideal, as depicted in the right-hand side of Figure 2.3.

### 2.3.2 2-DOF swing-up

Double pendulum is a pendulum attached to the end of another pendulum, with both links rotating in the vertical plane, similar to 1-DOF swing-up benchmark. Full description alongside with technical details about the benchmark can be found in A.2. The dynamics of the double pendulum is chaotic and complex, thus making it a good benchmark. The state $x$ contains the angles and angular velocities and is defined as $x = [\alpha_1, \dot{\alpha_1}, \alpha_2, \dot{\alpha_2}]^T$. The angles $[\alpha_1, \alpha_2]$ vary in the interval $[-\pi, \pi)$ rad and wrap around. $u = [u_1, u_2]^T$ is the control input which contains the torques of the two motors. The discrete set of control inputs $U$ is defined as the Cartesian product of the sets $\{-3, 0, 3\}$ and $\{-1, 0, 1\}$ for each link. The transition function $f(x, u)$ is obtained by numerically integrating system dynamics between discrete time samples using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.01$.

The control goal is expressed by the following quadratic reward function:

$$\rho(x, u, f(x, u)) = -\mathrm{abs}(x_{des} - x)^T Q, \text{ where } Q = [1, \ \ 0.05, \ \ 1.2, \ \ 0.05]^T \tag{2.8}$$

where $x_{des}$ represents the goal state and equals $[0, 0, 0, 0]^T$.

The simulation results presented in Table 3.5 exhibit the same pattern as with the 1-DOF pendulum swing-up. Additionally, the symbolically approximated value function leads to a significant improvement in performance.

### 2.3.3 Magnetic manipulation

Magnetic manipulation (abbreviated as Magman) is an challenging nonlinear control problem. The current through the electromagnets is controlled to dynamically shape the magnetic field above the magnets and so to accurately and quickly position a steel ball to the desired set point. Two variants are considered, one with two electromagnets and one with five. Full description with technical details for both variants can be found in A.3.

State $x$ is given by the position and velocity of the ball. The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.005\,\mathrm{s}$. The set of control inputs $U$ is defined as the Cartesian product of the vectors $[0, 0.3, 0.6]$, containing the discrete values of the control

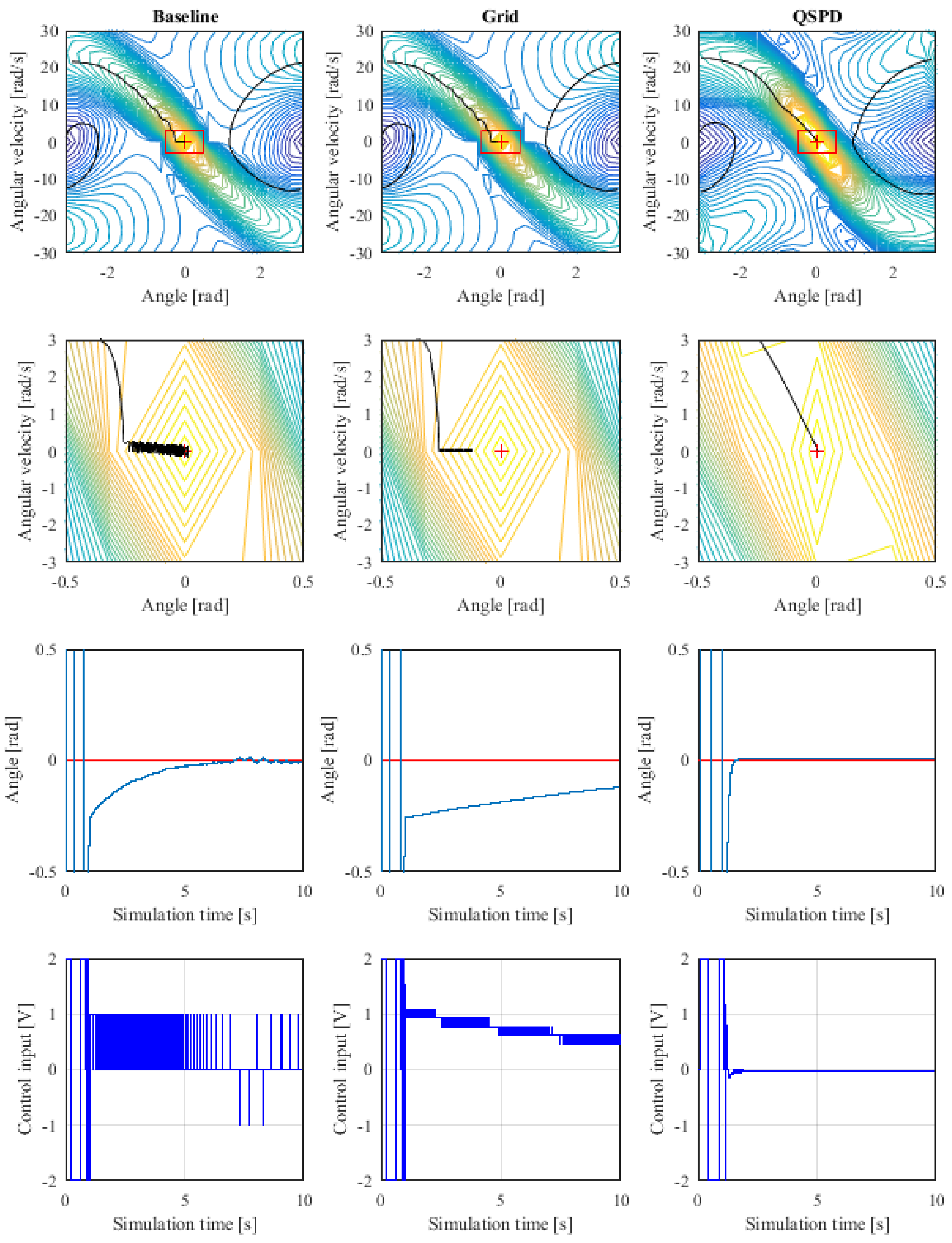

**Figure 2.3:** Trajectories from the initial point $[-\pi, 0]$ to the goal state, obtained using Baseline (left column), Grid (middle column) and *QSPD* (right column) on the 1-DOF pendulum swing-up task. The first row shows the whole state trajectory superimposed on the contours of the Bellman equation RHS, the second row the zoomed area around the goal state, the third row the angle, and the forth row the applied control inputs during simulation.

input to the $i$th coil. The reward function is defined as:

$$\rho(x, u, f(x, u)) = -\text{abs}(x_{des} - x)^T Q, \text{ where } Q = [5, \;\; 1]^T \tag{2.9}$$

where the desired position $x_{des}$ is set to 0.01 (m).

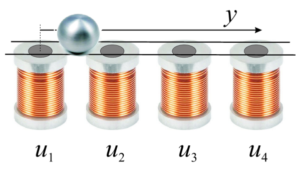

**Figure 2.4:** Magnetic manipulation setup.

The simulation results for both variants of the system are presented in Table 3.5.

The *QSPD* algorithm shows the best result in terms of the overall return. Additionally, *SR-Baseline* tends to be the fastest algorithm. However, when the dimensionality of the action space grows, *QSPD* starts dominating in both measures. The runtime of Baseline, Grid and Cheby algorithms (and their symbolic regression variants) grows exponentially with the input dimension. This prevents applying *Grid* and *Cheby* algorithms (and their symbolic variants) to the Magman5 benchmark, so their results are not reported in the table. The runtime of the *QSPD* algorithm, according to our experiments, grows approximately linearly. This results in the outstanding runtime performance $T_{alg} = 2.85\%$ for the Magman 5 benchmark.

## 2.4 Results and discussion

Table 3.5 summarizes the simulation results. As can be seen, *Grid* and *Cheby* algorithms perform worse than the *Baseline* solution with respect to $P_{alg}$. This is caused by the insufficient smoothness problem, illustrated in Figure 2.3. However, when combined with the smooth approximation of the V-function surface by means of symbolic regression, these algorithms lead to a significant performance improvement. Moreover, the use of the symbolically approximated surface alone is beneficial as well, as illustrated by the *SR-Baseline* results. It can be seen, that the difference between *SR-Grid* and *SR-Cheby* is quite unpredictable and depends on the quality of approximation of the local features of the V-function surface.

The *QSPD* algorithm performs better when the dimensionality of the action space grows. The reason why *QSPD* performs worse compared to symbolic algorithms is that it uses a rather poor Euler approximation $f^s(\cdot)$ of the system dynamics. Instead of the Euler method, another sophisticated approximation could be employed, e.g., the Adams-Bashforth method.

As can be seen in Table 3.5, the exponential growth of computational complexity prevents the use of *Grid* and *Cheby* algorithms (and their symbolic variants) on the Magman5 benchmark. Interestingly, *SR-Baseline* runs faster than the *Baseline* solution. This is thanks to the simpler form of the V-function approximation, which leads to faster computations.

| | | 1DOF swing-up | 2DOF swing-up | Magman2 | Magman5 |
|---|---|---|---|---|---|
| Baseline | $P_{alg}$ | 100% | 100% | 100% | 100% |
| | $T_{alg}$ | 100% | 100% | 100% | 100% |
| | $D_{alg}$ | 0.0493 | 0.0825 | 0.0007 | 0.0019 |
| | $R_{alg}$ | -1288.28 | -869.81 | -14.42 | -35.85 |
| Grid | $P_{alg}$ | 47.45% | 91.25% | 75.09% | - |
| | $T_{alg}$ | 634.19% | 1159.30% | 965.26% | - |
| | $D_{alg}$ | 0.1175 | 0.1132 | 0.0010 | - |
| | $R_{alg}$ | -2582.65 | -969.374 | -19.34 | - |
| Cheby | $P_{alg}$ | 48.03% | 95.69% | 130.77% | - |
| | $T_{alg}$ | 2281.80% | 1937.80% | 3641.10% | - |
| | $D_{alg}$ | 0.1147 | 0.0946 | 0.0003 | - |
| | $R_{alg}$ | -2551.43 | -925.85 | -11.76 | - |
| SR-Baseline | $P_{alg}$ | 129.33% | 247.80% | 163.09% | 242.31% |
| | $T_{alg}$ | **86.01%** | **77.59%** | **84.81%** | 87.26% |
| | $D_{alg}$ | 0.0901 | 0.1050 | 0.0002 | 0.0004 |
| | $R_{alg}$ | -1040.77 | -573.49 | -9.46 | -16.02 |
| SR-Grid | $P_{alg}$ | 166.54% | 287.11% | 178.80% | - |
| | $T_{alg}$ | 589.74% | 909.47% | 885.85% | - |
| | $D_{alg}$ | 0.0065 | 0.0408 | **0.0001** | - |
| | $R_{alg}$ | -887.48 | -547.08 | -8.89 | - |
| SR-Cheby | $P_{alg}$ | **188.89%** | **293.25%** | 176.96% | - |
| | $T_{alg}$ | 2200.80% | 1626.90% | 4092.20% | - |
| | $D_{alg}$ | **0.0000** | **0.0233** | 0.0003 | - |
| | $R_{alg}$ | **-837.32** | **-545.38** | -9.26 | - |
| *QSPD* | $P_{alg}$ | 149.67% | 193.61% | **254.08%** | **332.98%** |
| | $T_{alg}$ | 586.94% | 611.23% | 208.24% | **2.85%** |
| | $D_{alg}$ | 0.0076 | 0.1401 | **0.0001** | **0.0001** |
| | $R_{alg}$ | -955.11 | -667.50 | **-7.39** | **-12.55** |

**Table 2.2:** Summary of the simulation results

## 2.5 Conclusion

Several alternative policy derivation methods for continuous action spaces were studied. The simulation results showed that the proposed symbolic approximation significantly outperforms the standard policy derivation approach. In terms of control performance, *SR-Cheby* outperformed the other algorithms in most of the problems tested (with the exception of the Magman 2 example). However, when the dimensionality of action space grows, the computational complexity of this algorithm becomes prohibitive.

In applications where computational resources are limited (e.g., in robotics), it is recommended to use the *SR-Baseline* algorithm. The *Grid* and *Cheby* algorithms have not performed equally well across all problems tested; they perform worse when the V-function approximation is not smooth. Providing the right criteria for measuring this aspect is difficult, and it may be a part of future research. In conclusion, it is not recommended to use these algorithms out of the box.

The *QSPD* algorithm displays great scalability properties, which allow curbing the exponentially growing computational complexity. At the same time, it is superior to the baseline solution. This effect is achieved through both the symbolic nature of the underlying approximator and the exploitation of the symbolic derivatives. It is reasonable to assume that *QSPD* algorithm can handle tasks with bigger dimensionality. On top of that, it organically works with continuous action space; there is no discrete version. Altogether, it answers **RQ 1**.

The relative success of the symbolically enhanced algorithms is based on the selected best-performing symbolic approximator. The selection process, however, hides a very important topic: how to define the term "best-performing" in such a case. The current approach requires extensive testing of all the candidates, which is time-consuming. Given the fact that reinforcement learning is usually performed iteratively and requires an approximation of V-function at each step, it may result in SR being impractical for the task. Due to that fact, the proposed use of SR should be viewed as a proof of concept rather than an out-of-the-box approximation technique. These challenges are addressed in the consequent chapters.

# Chapter 3

# Symbolic Regression as Outer Approximation

Assume we have a numerical approximator of the V-function, reward function, and dynamics function. Can we re-approximate V-function symbolically, and will that approximation work?

*Author after many unsuccessful attempts to derive reasonable policy from symbolic approximators.*

## 3.1 Symbolic proxy-function

The policy derivation step can be understood as a hill-climbing process. The agent applies the control input at each timestep that leads toward a higher point on the V-function surface. An advantage of this control law is its inherent stability – the value function is analogous to the control Lyapunov function [58], [59]. However, the hill-climbing process is affected by the approximate nature of the V-function. A typical approximation by means of basis functions exhibits artifacts, which lead to the chattering of the control input and even to limit cycles. This problem is illustrated in Figure 3.1.

This undesired behavior typically occurs in the vicinity of the goal state and leads not only to suboptimal performance; but it can also render the goal state unreachable. An obvious approach to alleviating these problems would be to use a smooth approximation of the V-function. A good candidate for such a purpose is a symbolic approximator, which can be constructed, e.g., by means of genetic programming. However, minimizing an error measure between the given V-function data and the resulting symbolic function is misleading. Genetic programming has no information about the purpose of the function and can, therefore, ignore small but important parts of the V-function while still achieving the least possible error. Consequently, the

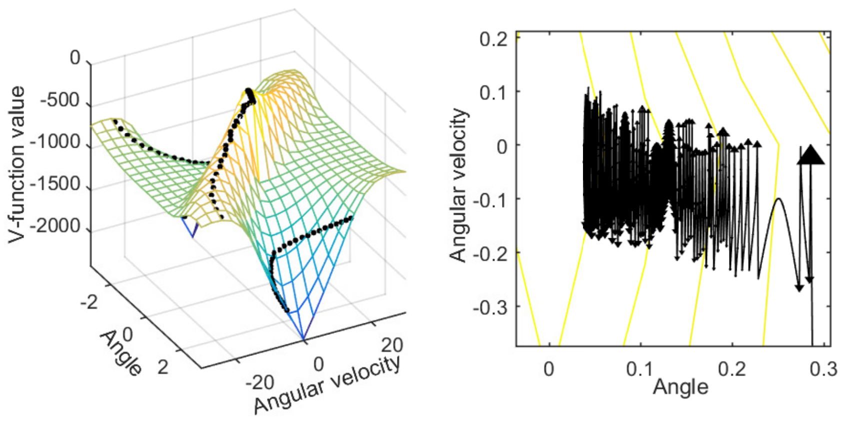

**Figure 3.1:** A sample state trajectory was obtained via hill-climbing on the approximate V-function surface for the pendulum swing-up benchmark (see Appendix A.1). Left: the state trajectory on the V-function surface. Right: the state trajectory in the vicinity of the goal state in $[0, 0]$.

resulting smooth approximation can have virtually the same shape as the V-function, but it can yield a completely different, suboptimal policy.

A novel method that uses genetic programming, in particular, a variant of Single Node Genetic Programming (see Appendix B-SNGP), is presented. Its purpose is to evolve a smooth proxy to the V-function, which is then used for a continuous policy derivation. A concept similar to *advantage updating* [76] is used. The genetic programming algorithm evolves the symbolic proxy-function that maximizes the number of correct choices of the control action for a set of training states. In this way, the search is biased toward a symbolic proxy-function that would be suited for the policy derivation, contrary to a symbolic V-function evolved by minimizing an error measure between the V-function data and the symbolic approximator.

This distinguishes the proposed approach from several works in the literature dealing with the use of genetic programming for V-function fitting. For instance, in [37] a method called Value Function Discovery is proposed that uses GP to evolve an algebraic description of the V-function. In [33] an evolutionary algorithm is used to accelerate the convergence of Q-tables. However, both approaches use an error measure between the given V-function data and the resulting symbolic function as an optimization criterion.

### 3.1.1 Preliminaries

#### Reinforcement learning

Define a finite set of discrete control input values $U = \left\{u^1, u^2, \ldots, u^M\right\}$ drawn from $\mathcal{U}$. An approximate V-function denoted by $\hat{V}(x)$ can be computed by solving the Bellman equation:

$$\hat{V}(x) = \max_{u \in U}[\rho(x,u) + \gamma \hat{V}(f(x,u))] \tag{3.1}$$

where $\gamma$ is the discount factor, a user-defined parameter. The fuzzy V-iteration algorithm (see Section 1.3.2) is employed to find $\hat{V}(x)$. The policy is the mapping:

$$h : \mathcal{X} \rightarrow \mathcal{U} \tag{3.2}$$

and the optimal discrete-valued policy corresponding to $\hat{V}(x)$ is:

$$\hat{h}(x) \in \underset{u \in U}{\operatorname{argmax}} \left[\rho(x,u) + \gamma \hat{V}(f(x,u))\right], \forall x \tag{3.3}$$

In the sequel $RHS(x,u)$ is used to refer to the $\left[\rho(x,u) + \gamma \hat{V}(f(x,u))\right]$ part of equation (3.3).

### 3.1.2 V-function proxy

Define a set of samples $X = \{x^1, x^2, \dots, x^N\} \in \mathcal{X}$. The genetic programming algorithm searches for a symbolic function $P(\cdot)$ that satisfies the following condition:

$$\underset{u \in U}{\operatorname{argmax}} \ [P(f(x,u))] = \underset{u \in U}{\operatorname{argmax}} \ [RHS(x,u)], \forall x \in X \tag{3.4}$$

Observe by comparing (3.3) and (3.14) that for the set $U$ of discrete inputs, $P(\cdot)$ yields the same optimal policy as the V-function. Note that $P(\cdot)$ does not have to satisfy the Bellman equation. Its purpose is to provide at each state the same preferred control action. It works, however, under the assumption that $P(f(x,u))$ has the same distinguishing properties as $RHS(x,u)$. It is only possible when $\rho(\cdot)$ part of $RHS(x,u)$ does not explicitly depend on $u$ (rather than on $f(x,u)$). Otherwise, $P(\cdot)$ will not be capable to robustly work in situations where several different control inputs lead to the same state.

To improve the robustness of the genetic search, the condition (3.14) is strengthened by replacing the *argmax* operator with the *order* operator. The *order* operator produces a partially ordered set of control input indices so that the corresponding control actions are ordered with respect to their evaluation by the given function. The purpose of this modification is to make sure that if the genetic search finds a suboptimal solution, a high-ranked sub-optimal action is chosen instead of the optimal one.

When $P(\cdot)$ is found, the policy can be derived by using (3.3), where $RHS(x,u)$ is replaced with $P(f(x,u))$. The overall setting is schematically depicted in Figure 3.2.

The *order* operator can be formalized as follows:

$$\underset{u \in U}{\operatorname{order}} \ (RHS(x,u)) = \{i, j, ..., \ell\} \tag{3.5}$$

with

$$RHS(x,u^i) \geq RHS(x,u^j) \geq \cdots \geq RHS(x,u^\ell) \tag{3.6}$$

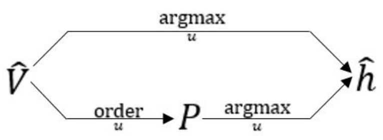

**Figure 3.2:** Proxy-function: schematic principle of work.

### 3.1.3 SNGP for evolving the proxy-function

For fitting proxy-function, a special variant of SNGP (see Appendix B.1 for further details) is proposed. It uses a specific population model and fitness function as described in the following paragraphs.

#### Population model

Typically, the function set contains functions that might produce invalid output values, such as a division by zero. In order to avoid such cases, protected versions of these functions are used instead. These functions are forced to produce a valid output for any input. For example, the protected division outputs a predefined value whenever the denominator is zero. Thus, the output of any candidate expression is ensured to be a valid number. However, due to such hard-coded irregular behavior of the protected functions, expressions using the protected functions can still exhibit undesired behavior, e.g., the expressions might become non-differentiable at some data points or contain local approximation artifacts. Even if it has a minimal error on the training data, such a symbolic function can be effectively useless when applied to new, previously unseen data.

Here, a *partitioned population* is proposed. It is divided into two parts – *head* and *tail*. The head part contains nodes that are roots of constant-valued expressions only. It uses extended function set $F_e$ including the protected functions. Each head node can use other head nodes and constant terminal nodes as its input. The tail part contains nodes that can only be chosen from a set $F_s$ of simple non-conflicting functions (i.e., no protected functions). Tail nodes can use all preceding head and tail nodes and both constants and variables as their input. In this way, a "reasonable" behavior of expressions rooted in tail nodes is ensured since the protected functions are used exclusively to produce constants.

#### Fitness function

When fitting the symbolic proxy-function $P(\cdot)$, a set of $M$ distinct actions $U = \{u^1, \dots, u^M\}$ sampled from the original continuous action space $\mathcal{U}$ is used to generate a set of $N$ training samples. A training sample is generated for each state $x^i \in X$, has the following structure:

$$t^i = [x^i, f(x^i, u^1), \dots, f(x^i, u^M), o^1, \dots, o^M] \tag{3.7}$$

where $o^j$ denotes the order class to which the next state obtained by applying the action $u^j$ to the state $x^i$ is assigned according to $RHS(x^i, u^j)$. The best next state is assigned to order class 0; the second-best next state is assigned to order class 1, and so on. Note that multiple states can be assigned to the same order class.

A candidate function $P(\cdot)$ produces values $P(f(x^i, u^1))$, ..., $P(f(x^i, u^M))$ for each $x^i$ in the training set. Denote by $[o'^1, \dots, o'^M]$ the order classes derived from these values. The SNGP searches for the proxy-function $P(\cdot)$ that is optimal according to the following fitness function:

$$fitness(P(f(x,u))) = \sum_{i=0}^{N} w(x^i) \sum_{j=0}^{M} (fn(j) + fp(j)) \tag{3.8}$$

where

$$fn(j) = \begin{cases} 1 + 0.1\ dist(j), & \text{if } o^j = 0 \text{ and } o'^j \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

and

$$fp(j) = \begin{cases} 1, & \text{if } o^j \neq 0 \text{ and } o'^j = 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.10}$$

The function *false positive*, $fp(j)$, penalizes with penalty 1 the cases where the state $f(x^i, u^j)$ should not belong to the best order class according to $o^j \neq 0$, but it does belong to it (i.e., $o'^j = 0$).

Similarly, the function *false negative*, $fn(j)$, penalizes with penalty 1 the cases where the state $f(x^i, u^j)$ should be the best one according to $o^j = 0$, but it is not (i.e., $o'^j \neq 0$). To refine the fitness function, each false negative case is further penalized with the term $dist(j)$ that is calculated as the absolute difference between the $P(f(x^i, u^k))$ of actually the best next state achieved with action $u^k$ applied to the state $x^i$, and the $P(f(x^i, u^j))$. This way, the fitness can distinguish between candidate proxy-function producing the same number of false negative and false positive cases. The value of $dist(j)$ is bounded from above by 1.0. By weighting $dist(j)$ with the factor 0.1, it is ensured that the suboptimal action plays a secondary role in the fitness.

As mentioned in Section 3.1, not all states are equally important. The effect of chattering is much stronger in the vicinity of the goal state. Thus, it can be beneficial to weigh the overall penalty calculated for a given state with respect to its distance to the goal state. To avoid negative effects caused by the magnitudes of different variables in a state space, each variable of the state space must be mapped into the same range $[0, 1]$. The weight function $w(x^i)$ returns a square of the reciprocal of the Euclidean distance between the state $x^i$ and the goal state. Thus, the errors made in states far from the goal state are penalized less than the errors made in states close to the goal state. When $x^i$ is the goal state, the $w(x^i)$ returns the weight of the state nearest to the goal state.

Since the fitness function expresses how far the candidate proxy-function is from the ideal, the resulting fitness function is to be minimized.

### 3.1.4 Policy derivation methods used

#### Policy derivation using raw function

The first algorithm considered is a direct usage of the computed proxy-function $P(f(x,u))$, as stated in (3.3). Algorithm 4 formalizes this procedure.

**Algorithm 4:** Policy derivation from the raw symbolic function (in the sequel denoted as Raw)

```
Input: $f(x,u)$, $P(f(x,u))$, $U$, $x_0$
$k \leftarrow 0$
while control experiment not terminated do
    $u_k \leftarrow \underset{u' \in U}{\operatorname{argmax}} \; P(f(x_k, u'))$
    $x_{k+1} \leftarrow f(x_k, u_k)$;
    $k \leftarrow k+1$
end
Output: trajectory $[x_0, x_1, ...]$, $[u_0, u_1, ...]$
```

#### Policy derivation using hybrid symbolic function

The main purpose of this method is to provide a robust policy. Due to the stochastic nature of genetic programming, a successful finding of the optimal proxy-function is not guaranteed. Therefore, all areas for which $P(f(x,u))$ provides non-optimal policy should be covered by another approximation. The computed $\hat{V}$ is used for this purpose. Define a vector $C = [c^1, c^2, ..., c^N]^T$ of boolean flags, where $N$ is a number of training samples. Each flag $c^i$ is true if the corresponding training sample $t^i$ is successfully fitted by the proxy-function $P(f(x,u))$, otherwise it is false. At the policy derivation step $k$, a training sample $t^j$ with its state $x^j$ nearest to the current state $x_k$ is found among all $N$ training samples. This can be easily done by using *k-d tree* for the nearest neighbor search. Then, the optimal input to be applied in state $x_k$ is derived using the $P(f(x,u))$ if the corresponding $c^j$ is true. Otherwise, the policy in state $x_k$ is derived using the $RHS(x,u)$. A pseudo-code for this policy derivation method is shown in Algorithm 5 where the function $NN(x)$ provides the nearest neighbor state to the state $x$.

#### Policy derivation using hybrid symbolic function and a fine grid of actions

The key idea behind this method is based on the combination of the Grid policy derivation method (introduced in Section 2.2.2) and the Algorithm 5. The main point can be briefly described as follows.

Define a set of actions $A$ as

$$A = \bar{U}_1 \times \bar{U}_2 \times \cdots \times \bar{U}_m, \quad A \subseteq \mathcal{U} \tag{3.11}$$

**Algorithm 5:** Policy derivation from the hybrid approximation (in the sequel denoted as Hybrid)

```
Input: $f(x,u)$, $\rho(x,u)$, $C$, $P(f(x,u))$, $\gamma$, $\hat{V}$, $U$, $x_0$
$k \leftarrow 0$
while control experiment not terminated do
    if $C[NN(x_k)]$ then
        $u_k \leftarrow \underset{u' \in U}{\text{argmax}}\ P(f(x_k, u'))$
    else
        $u_k \leftarrow \underset{u' \in U}{\text{argmax}}\ [\rho(x_k, u') + \gamma \hat{V}(f(x_k, u'))]$
    end
    $x_{k+1} \leftarrow f(x_k, u_k)$;
    $k \leftarrow k + 1$
end
Output: trajectory $[x_0, x_1, ...]$, $[u_0, u_1, ...]$
```

where each set $\bar{U}_i$ contains points equidistantly distributed along the $i$th dimension of the action space. The set $A$ contains the control inputs that will be considered in the policy derivation using the $P(f(x,u))$ and $RHS(x,u)$, respectively. The policy derivation algorithm is essentially equal to the Algorithm 5 with the action set $U$ being replaced with the set $A$. In the sequel, this algorithm is called HybridGrid.

The size of the set $A$ is given by a vector $A_s = [a_1, a_2, \ldots, a_m]^T$ where each $a_i$ corresponds to the number of points along the $i$th dimension of the action space. By default, $a_1 = a_2 = \cdots = a_m = 11$ are selected.

### Baseline policy derivation

The baseline policy derivation uses equation (3.3) at every policy derivation step $k$, where the maximization is computed over the same discrete action set $U$ on which $\hat{V}(x)$ has been learned. Formally, this can be described by Algorithm 4 with $RHS(x,u)$ substituted for $P(f(x,u))$. In the sequel, this algorithm is abbreviated Baseline.

## 3.1.5 Experimental evaluation

### Pendulum swing-up

The inverted pendulum consists of a mass attached to an actuated link that rotates in the vertical plane. The available torque is insufficient to push the pendulum up in a single rotation from the majority of initial states. Instead, from a certain state (e.g., pointing down), the pendulum needs to be swung back and forth to gather energy before it can be pushed up and stabilized. Full description alongside technical details about the benchmark can be found in Appendix A.1.

The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.01\,\text{s}$. The set of discrete control inputs is $U = \{-2, -1, -0.05, 0, 0.05, 1, 2\}$.

The control goal is to stabilize the pendulum in the unstable equilibrium $\alpha = \dot{\alpha} = 0$, which is expressed by the following quadratic reward function:

$$\rho(x, u) = -f^T(x, u)Qf(x, u), \text{ where } Q = \text{diag}[5, 0] \tag{3.12}$$

## Magnetic manipulation

Magnetic manipulation (abbreviated as Magman) has several advantages compared to traditional robotic manipulation approaches. First of all, it is contactless, which opens new possibilities for actuation on a micro-scale and in environments where it is not possible to use traditional actuators. In addition, magnetic manipulation is not constrained by the robot arm morphology, and it is less constrained by obstacles. Magman is a challenging nonlinear control problem. The current through the electromagnets is controlled to dynamically shape the magnetic field above the magnets and so to accurately and quickly position a steel ball to the desired set point. For the experiments presented in this work, the first two coils, at the positions 0 (m) and 0.025 (m), respectively, have been used. Full description alongside technical details about the benchmark can be found in Appendix A.3. State $x$ is given by the position and velocity of the ball. The control input $u$ is defined as the vector of currents $[u_1 u_2] \in [0, 0.6]$ on the coils. The reward function is defined as:

$$\rho(x, u) = |(x_d - f(x, u))|\ Q, \text{ where } Q = \text{diag}[100, 5] \tag{3.13}$$

The desired position $x^d$ is set to $x^d = 0.01$ (m). The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.005\,\text{s}$. The set of control inputs $U$ is defined as the Cartesian product of the vectors $[0, 0.6]$, containing the discrete values of the control input to the $i$th coil.

## V-function learning algorithm

To compute $\hat{V}(x)$, the fuzzy V-iteration algorithm is used (see Section 1.3.2 for full description). The learning parameters used for both benchmarks are listed in Table 3.1.

| Parameter | Pendulum swing-up | Magman |
|---|---|---|
| Discount factor, $\gamma$ | 0.99999 | 0.999999 |
| Samples per dimension, $B$ | $[21, 21]^T$ | $[21, 27]^T$ |
| Convergence threshold, $\epsilon$ | $10^{-5}$ | $10^{-8}$ |

**Table 3.1:** Fuzzy V-iteration parameters

### Complexity of the proxy-function computation

The computational complexity of proxy-function can be estimated only approximately due to the stochastic nature of the genetic programming. The whole evolution process is running for a limited number $I$ of iterations. In each iteration $L$, candidate solutions are evaluated on every given sample point. The total number of samples is denoted as $N$. Therefore, the complexity linearly depends on the parameters $I$, $L$, and $N$.

## 3.1.6 Results

SNGP with a population of size 200 equally divided into the head and tail parts and function sets $F_e = \{+, -, *, /, square, cube, sqrt, exp, sin, ln\}$ and $F_s = \{+, -, *, square, cube, sqrt, sin\}$ were used in this experiment. The evolution of the proxy-function was run for 5000 iterations, with a maximum time limit of 300 (s).

For measuring the performance of the proxy-function, the coverage ratio between correctly fitted samples and the total number of training samples $N$ was used. The coverage map of the pendulum swing-up benchmark is depicted in Figure 3.3.

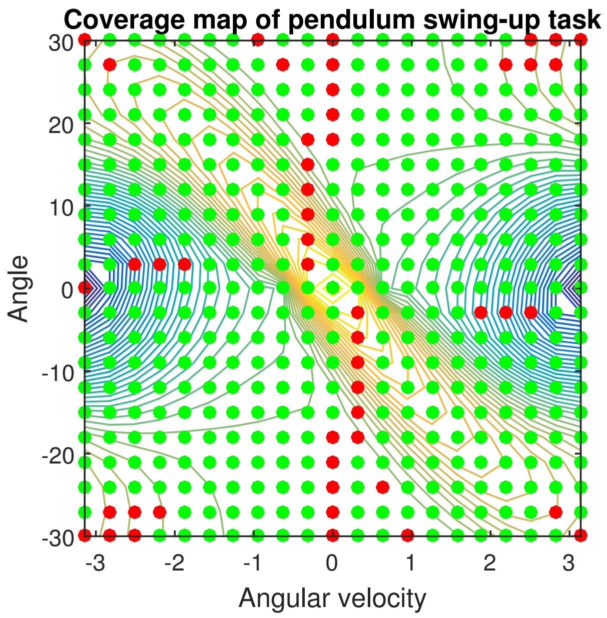

**Figure 3.3:** Coverage map for the pendulum swing-up benchmark. The green dots represent the samples in which the proxy-function is fitted correctly and the red dots otherwise.

Several proxy-functions were fitted for each benchmark. The policy derivation methods were tested 50 times on each proxy-function using the same set of randomly chosen initial states. Using these proxy-functions, the policy derivation methods have been tested 50 times with randomly chosen initial conditions. Simulation time was set to 3 (s) for the pendulum swing-up and to 1 (s) for the magnetic manipulation, respectively. To measure the performance of the algorithms, the following criteria are defined:

- Average return $R_a = \frac{\sum_{j=1}^{50} \sum_{i=0}^{K} r(x_i,u)}{50}$, where $K$ denotes the number of time steps in a control experiment.
- Performance ratio $P_r$ – the ratio between returns obtained by classical policy derivation and the tested algorithm.
- Average performance ratio in percents (denoted as average $P_r$) – a mean of $P_r$ values over 50 simulations.

The results obtained with the best-performing proxy-function (w.r.t. the average $P_r$) for each benchmark are presented in Tables 3.2 and 4.4, respectively. The performance ratios for both benchmarks are depicted in Figure 3.4. Found symbolic proxy-function for the pendulum swing-up and the magnetic manipulation benchmarks are given by the following equations after algebraic simplification.

- Pendulum swing-up proxy-function:
$\sqrt{\left|5^6\, x_1\, |x_0|^{18} - \sin\left(\sqrt{|x_0|} + |x_0|^{\frac{3}{2}} - 0.70711\right) + |x_0|^{\frac{3}{2}}\right|} - x_0$
- Magman proxy-function: $\sin\left(\sqrt{\left|\sin\left(|x_1|^{\frac{3}{2}}\right)^2 + \sqrt{|x_0|}\,(x_1 - 8.0)\right|}\right)$

| Criterion | Baseline | Raw | Hybrid | HybridGrid |
|---|---|---|---|---|
| Average return | -642.09 | -2452.10 | -601.83 | -588.45 |
| Average $P_r$ | 100% | 31.52% | 114.75% | 117.64% |
| Coverage | — | 88.8% | 88.8% | 88.8% |

**Table 3.2:** Pendulum swing-up simulation results

| Criterion | Baseline | Raw | Hybrid | HybridGrid |
|---|---|---|---|---|
| Average return | -106.61 | -79.86 | -77.98 | -71.02 |
| Average $P_r$ | 100% | 162.47% | 163.91% | 193.81% |
| Coverage | — | 89.7% | 89.7% | 89.7% |

**Table 3.3:** Magnetic manipulation simulation results

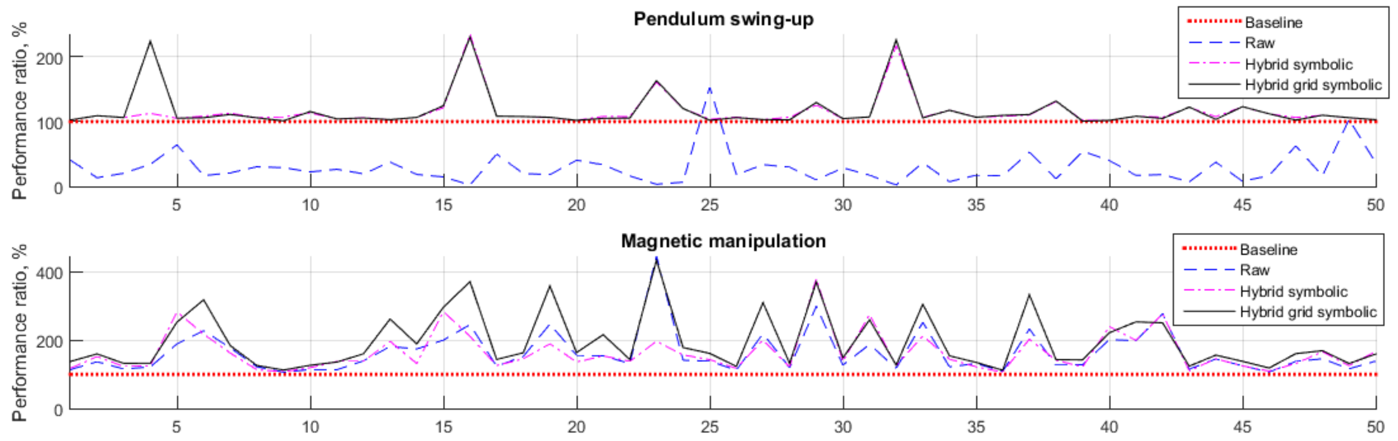

**Figure 3.4:** The performance ratio in percents for 50 simulations by means of different policy derivation methods. Simulations are performed with randomly chosen initial conditions. The upper and lower plots show the pendulum swing-up benchmark and the magnetic manipulation benchmark, respectively. For both figures, the horizontal axis displays separate simulations, which are connected by lines for better visualization.

## Discussion

The Hybrid and HybridGrid algorithms significantly outperform the Baseline algorithm as shown in Figure 3.4 and Tables 3.2 and 4.4. Figure 3.4-top shows poor results for the Raw algorithm on the Pendulum swing-up problem. The reason is quite straightforward. As stated in Section 3.1, states are not equally important and genetic programming cannot guarantee finding the optimal proxy-function, as illustrated in Figure 3.3. It shows the coverage map in which green and red dots represent correctly and incorrectly fitted samples, respectively. An incorrectly fitted sample is a sample in which proxy-function chooses a different control input than the V-function. It is empirically found that the regions in the vicinity of the two red dots, nearest to the state space center, must be fitted correctly in order to reach the goal state. Hybrid and HybridGrid algorithms successfully use $RHS(x, u)$ in those regions, while the Raw algorithm suffered catastrophic failure since it had no ability to do so.

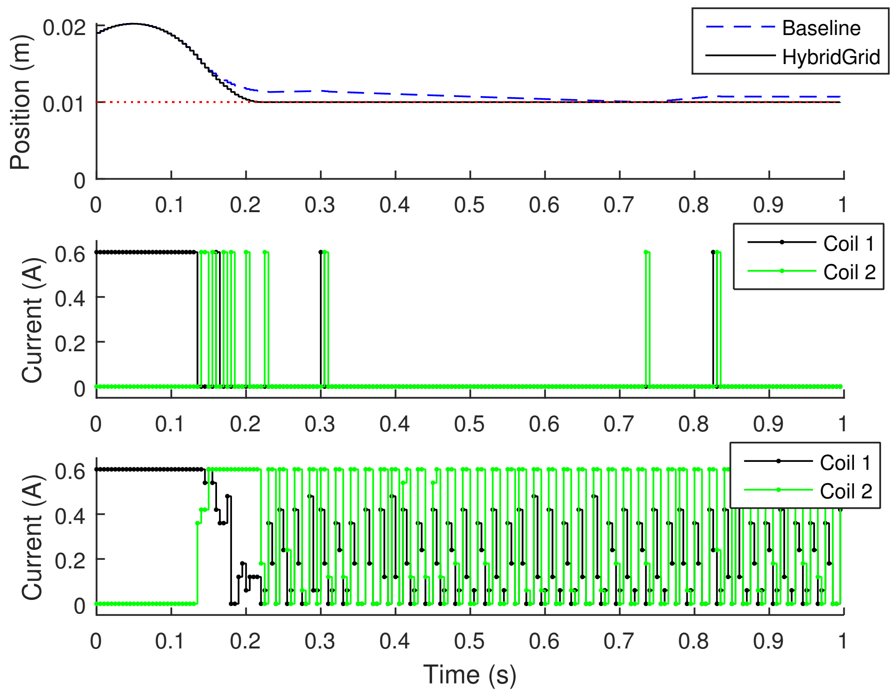

**Figure 3.5:** Simulation of the magnetic manipulation. The top panel represents position of the ball, where 0.01 m is the desired position. The middle and bottom panels show the control inputs, obtained by applying Baseline and HybridGrid algorithms, respectively.

Figure 3.5 shows simulations of Baseline and HybridGrid algorithms on the magnetic manipulation benchmark. It can be seen that the chattering in the state space is significantly suppressed, but chattering in the control inputs is still present. Input chattering can usually be reduced by penalizing the control input in the reward function, which is, however, not possible here due to the choice of the proxy-function structure as $P(f(x, u))$. One way to overcome this limitation is to reformulate the proxy-function as $P(x, u)$ and then use it in policy derivation in the same way as a Q-function. This may be a part of future work.

## 3.2 Generalized proxy-function

This work extends the idea of the originally proposed proxy-function. Originally proposed proxy-functions rely on a binary fitness function, which leads to a non-convex optimization problem and reducing the chances of finding an accurate proxy-function. Consequently, the originally proposed symbolic approximator had to be combined with a numeric approximator. The current work builds upon this result by resolving these outstanding issues of the originally proposed proxy-function. In this section, an enhanced symbolic regression approach which uses linear programming is proposed.

The main idea of the proxy-function method (see Section 3.1 for the details) is to find through symbolic regression a smooth, analytically defined function $P(\cdot)$, which $\forall x \in \mathcal{X}$ satisfies the following equation:

$$\underset{u \in U}{\operatorname{argmax}} \; P(f(x,u)) = \underset{u \in U}{\operatorname{argmax}} \left[\rho(f(x,u)) + \gamma \hat{V}(f(x,u))\right] \tag{3.14}$$

In order to generalize this method, assume that the proxy-function has the form:

$$P(x) = \beta_1 p_1(x) + \beta_2 p_2(x) + \ldots + \beta_q p_q(x) \tag{3.15}$$

where $p_1, \ldots, p_q$ are continuous analytic functions generated by means of evolutionary programming, each of them being defined over the whole state space and $\beta_1, \ldots, \beta_q$ are real-valued coefficients.

Assume that the numerical approximator $\hat{V}(\cdot)$ is given. As described earlier, the policy derivation process can be regarded as hill-climbing. At each time step, the agent selects the control input which leads to the highest value of the right-hand side of the Bellman equation:

$$u^* = \underset{u \in U}{\operatorname{argmax}} \left[\rho(f(x,u)) + \gamma \hat{V}(f(x,u))\right] \tag{3.16}$$

The selected action $u^*$ is then applied to the system, which leads to the new state:

$$x^* = f(x, u^*) \tag{3.17}$$

To find a $P(\cdot)$, or its close approximation, a set of $N$ state samples $X = \{x_1, x_2, \ldots, x_N\} \in \mathcal{X}$ is generated. By using the already defined set of discrete control inputs $U = \{u_1, u_2, \ldots u_M\}$, for each state $x_i \in X$, it is possible to construct the following set of next states:

$$X_i^n = \{x_{ij} \mid x_{ij} = f(x_i, u_j), j = 1, 2, \ldots M\} \tag{3.18}$$

and partition it into optimal and suboptimal next states:

$$\tilde{X}_i = x_i^o \cup X_i^s \tag{3.19}$$

The optimal next state maximizes the right-hand side of the Bellman equation:

$$x_i^o = \underset{x_{ij} \in X_i^n}{\operatorname{argmax}} \left[\rho(x_{ij}) + \gamma \hat{V}(x_{ij})\right] \tag{3.20}$$

and the suboptimal next states are all the remaining ones:

$$X_i^s = \tilde{X}_i \setminus x_i^o \tag{3.21}$$

For simplicity, assume that the optimal state for each sample in X is unique. However, the proposed method can be trivially extended to handle multiple optimal next states.

To define the fitness function for symbolic regression, (3.14) is reformulated as follows:

$$P(x_{ik}^s) - P(x_i^o) < 0, \ \forall i, k \tag{3.22}$$

This means that for each state $x_i$, the proxy-function value for the optimal next state must be larger than the value for suboptimal next states. Index $k$ runs over all $L$ elements in $X_i^s$. Substituting from (3.15), the above inequality becomes:

$$\sum_{j=1}^{q} \beta_j \left(p_j(x_{ik}^s) - p_j(x_i^o)\right) < 0, \quad \forall i, k \tag{3.23}$$

To simplify the notation, define an auxiliary variable $d_j^{ik}$ as:

$$d_j^{ik} = p_j(x_{ik}^s) - p_j(x_i^o) \tag{3.24}$$

To represent (3.23) for all the data in matrix form, define

$$O = \begin{pmatrix} d_1^{11} & d_2^{11} & \cdots & d_q^{11} \\ d_1^{12} & d_2^{12} & \cdots & d_q^{12} \\ \vdots & \vdots & \ddots & \vdots \\ d_1^{1L} & d_2^{1L} & \cdots & d_q^{1L} \\ d_1^{21} & d_2^{21} & \cdots & d_q^{21} \\ d_1^{22} & d_2^{22} & \cdots & d_q^{22} \\ \vdots & \vdots & \ddots & \vdots \\ d_1^{NL} & d_2^{NL} & \cdots & d_q^{NL} \end{pmatrix} \tag{3.25}$$

and solve the problem by linear programming:

$$\begin{aligned} &\min_{\beta} \emptyset \quad \text{such that} \\ &\qquad\qquad O\beta \leq \epsilon \end{aligned} \tag{3.26}$$

where $\epsilon$ represents a small negative constant.
[1]

[1]The purpose of $\epsilon$ is to make $O\beta$ strictly smaller than zero. From the practical point of view, it is recommended to choose the value with respect to the constraint tolerance the particular solver supports. In this work, $\epsilon = -0.001$ has been chosen.

Note that (3.26) defines a feasibility problem rather than a minimization problem. An infeasibility measure of the candidate solution is introduced to guide the evolutionary process toward a feasible solution.

Define a vector of non-positive variables $s = [s_1, \dots, s_{[N \times L]}]^T$. The fitness function for symbolic regression can now be defined as:

$$\begin{aligned} \min_{\beta, s} \sum_{i=1}^{N \times L} -s_i \quad & \text{such that} \\ O\beta + s &\leq \epsilon \\ -\infty \leq s &\leq 0 \end{aligned} \tag{3.27}$$

This formulation adds an extra variable to every inequality, which represents the measure of the infeasibility of the resulting model, and which linear programming then minimizes. The $\beta$ weights of the analytic expressions are defined as free variables with no restrictions.

In this work, a variant of the Single Node Genetic Programming (SNGP) is used (see Appendix B.2) to generate the non-linear analytic expressions $p_1(\cdot), \dots, p_q(\cdot)$, which are then evaluated using (4.17). The whole process repeats until a stopping criterion is satisfied, such as a prescribed number of iterations or an improvement threshold.

### 3.2.1 Experimental evaluation

The proposed method has again been tested on two different benchmarks: pendulum swing-up and magnetic manipulation. Both of them were explicitly discussed previously, including all necessary mathematical details.

For each benchmark, 30 different proxy-functions have been constructed in 30 independent SNGP runs. Each function has been tested on each benchmark with $N = 100$ randomly chosen initial states via simulations. It should be noted that all the functions are presented *"as is"*, which means that there is no selection procedure w.r.t. to some criterion.

Each result is compared with the baseline solution (3.3) (using the same initial states), which is computed beforehand by means of the fuzzy V-iteration algorithm. To evaluate each of proxy-functions, the following criteria are defined:

- Improvement percentage

$$I = \frac{1}{N} \sum_{j=1}^{N} \left[ p^j_{baseline} / p^j_{method} \right] \cdot 100\%$$

  where $p_{method} = \sum_{k=1}^{T_{sim}/T_s} \rho(f(x_k, u_k))$, with $T_{sim}$ stands for the total simulation time, $T_s$ is the sampling period, and *method* represents either *baseline* or *proxy* solution. The reward functions are defined to have maximum value zero in the goal state and to be negative otherwise. Therefore I equals 100% for the baseline and it is bigger than 100% if the proxy-function outperforms the baseline approach.

- Mean distance between the last state (at the end of simulation) $x_{end}$ and the desired goal state $x_{des}$

$$D = \frac{1}{N} \|x_{des} - x_{end}\|$$

where $\| \cdot \|$ is the Mahalanobis norm.

The sampling parameters for both benchmarks, as well as SNGP parameters, are listed in Table 3.4.

| **Fuzzy V-iteration parameters: pendulum swing-up** | |
|---|---|
| State space, $\mathcal{X}$ | $[-\pi, \pi] \times [-30, 30]$ |
| Input space, $\mathcal{U}$ | $[-2, 2]$ |
| State samples per dimension, $B_X$ | $[21, 21]$ |
| Action samples per dimension, $B_U$ | 11 |
| Discount factor, $\gamma$ | 0.95 |
| Convergence threshold, $\epsilon$ | $10^{-4}$ |
| Desired state, $x_{des}$ | $[0, 0]^T$ |
| Sampling period, $T_s$ [s] | 0.02 |
| Simulation time, $T_{sim}$ [s] | 3 |
| **Fuzzy V-iteration parameters: magman** | |
| State space, $\mathcal{X}$ | $[0, 0.05] \times [-0.4, 0.4]$ |
| Input space, $\mathcal{U}$ | $[0, 0.6]$ |
| State samples per dimension, $B_X$ | $[21, 21]$ |
| Action samples per dimension, $B_U$ | $[3, 3]$ |
| Discount factor, $\gamma$ | 0.99 |
| Convergence threshold, $\epsilon$ | $10^{-8}$ |
| Desired state, $x_{des}$ | $[0.01, 0]^T$ |
| Sampling period, $T_s$ [s] | 0.01 |
| Simulation time, $T_{sim}$ [s] | 3 |
| **SNGP parameters** | |
| Population size | 1000 |
| Elementary functions | +, -, $\times$, $x^2$, $x^3$, BentGeneral, Logistic3 |
| Maximal depth of features | 5 |
| Maximal number of features | 30 |
| Epoch length | 500 |
| Local search iterations | 500 |
| Number of epochs | 1 |
| Number of threads | 1 |

**Table 3.4:** Experiment parameters

### 3.2.2 Results

The simulation results for both benchmarks are listed in Table 3.5. The proposed method shows the potential to outperform the baseline solution significantly. According to the experiments, proxy-functions demonstrate significant improvement in a range of 100%-182% w.r.t. the baseline. In the vast majority of cases, it is caused by alleviating the negative impact of numerical artifacts. An example of it is depicted in Figure 3.6. This example demonstrates the comparison of derived policies for the magnetic manipulation benchmark. The left column corresponds to the baseline V-function, computed by the fuzzy V-iteration algorithm, while the right column stands for one of the proxy-functions. It can be seen that the proxy-function significantly alleviates the steady-state error caused by artifacts. Another interesting note is that the proxy-function remarkably violates Lyapunov stability condition, as shown by the bottom row of Figure 3.6. The reason for that behavior is that the proxy-function considers only local properties of its surface, neglecting global geometry.

| | | Median $D$ | Median $I$ |
|---|---|---|---|
| Pendulum swing-up | Baseline | 0.003 | 100% |
| | Generalized proxy | 0.002 | 105% |
| Magnetic manipulation | Baseline | 0.038 | 100% |
| | Generalized proxy | 0.007 | 182% |

**Table 3.5:** Experimental study statistics

The proposed method has several limitations. First of all, with the proposed design, it is not possible to penalize input chattering. Input chattering can usually be reduced by penalizing the control input in the reward function, which is, however, not possible here due to the choice of the proxy-function structure as $P(f(x, u))$. One way to overcome this limitation is to reformulate the proxy-function as $P(x, u)$ and then use it in policy derivation in the same way as a Q-function. Another possible way is to combine MSE-like fitness and proxy-function fitness to represent a trade-off between global V-function geometry and local properties of proxy-function. This may be a part of future work.

## 3.3 Conclusion

The proposed method offers an alternative way to derive policy for RL. Instead of using V-function directly, the proposed approach builds a smooth proxy-function on top of it, from which the better policy can be derived. The proposed method may be combined with any V-function approximation. Moreover, due to the analytic nature of the proxy-function, it can be combined with policy derivation methods (see Chapter 2.2 for details) for further policy improvement. Research on proxy-functions leads to several conclusions listed below:

- Single symbolic expression lacks the flexibility to represent even the simplified function. The original proxy-function was not capable of working without being supported by another approximation technique. Contrary to that, the generalized proxy-function is more flexible due

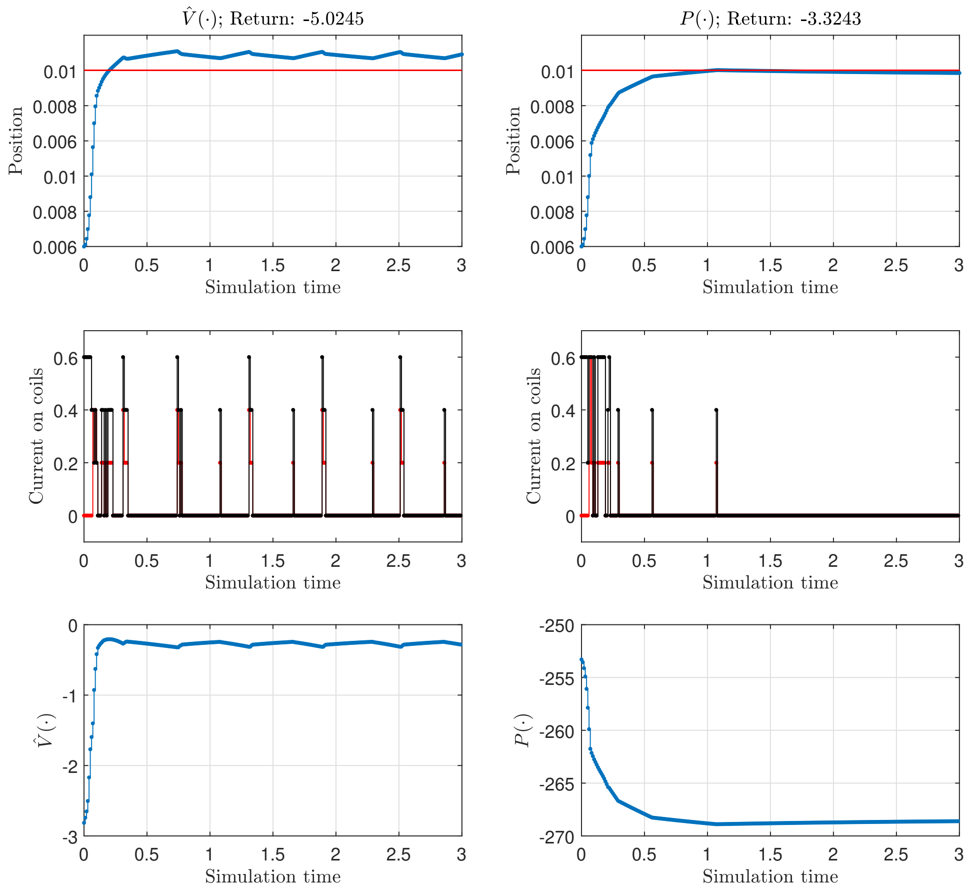

**Figure 3.6:** Magnetic manipulation transient response using the original V-function $\hat{V}(x)$ and one of the proxy-functions. The initial state is $[0.005, 0]^T$ and the goal state $x_{des} = [0.01, 0]^T$. The first row represents the position of the ball; the second stands for the control inputs, where different coils are depicted with different colors; the last row shows changing a value of either value function or proxy-function.

to the combination of several symbolic expressions together. As the complexity of tasks grows, it is beneficial to use the proposed extended version of SR formulation.

- Proxy-functions tend to form surfaces that violate Lyapunov stability. Despite that fact, the derived policy is reasonable and near-optimal. This observation requires additional attention and will be addressed in the next chapter.

- Even simplified proxy-functions are able to outperform standard approximation methods due to their inherent smoothness in the sense of the absence of approximation artifacts. Their main drawback is the need for another ground-truth approximation. The generalized version of proxy-function, however, does not suffer from that drawback. It demonstrated superiority to both original proxy-function and standard methods.

This chapter provides a partial answer to the **RQ 2**. Generalized proxy-function can be considered as a minimum valuable product. Fitting of the V-function, however, requires more attention due to the imperfect definition of the fitness criterion. This challenge is addressed in a consecutive chapter.

# Chapter 4

# Fitness Criterion Study

Assume we have a limited number of samples from some optimal V-function and the reward function and dynamics function. According to what fitness criterion shall we build a symbolic approximation of the V-function?

*Author, after getting annoyed by time-costly attempts to approximate V-function with an unpredictable result.*

## 4.1 Introduction

The main topic of this chapter is based on the question: "what criterion should be used to evaluate the quality of an approximation of the V-function?". The most common approach, which can be found in nearly all approximate RL publications, is to minimize the mean squared error of the estimator w.r.t. the target V-function. However, the utility of using such a criterion is still an open question [77], [57]. In a case where near-zero error cannot be achieved, the actual performance of such a solution is not determined. This situation is demonstrated in Figure 5.1. The top row depicts a trajectory obtained using an approximation technique with proven convergence and near-zero approximation error. The second and the third rows show trajectories obtained using symbolic regression - an approximation technique without proven convergence. The third row depicts a model with an approximation error bigger by an order of magnitude than the second row. Nevertheless, the third row demonstrates a much *better* performance. In the depicted case, there exist critical areas through which the goal state is reachable. If the approximation error affects these areas, that is a worse scenario than where these important areas are not affected. The known solution for that problem relies on using relevancy weights to emphasize a priori known important states[78], [79].

However, it relies either on the a priori knowledge of good trajectories or on a large number of samples. Both requirements seem to be unrealistic for a high-dimensional task, mainly due to the curse of dimensionality.

From the RL point of view, this work is dedicated to the simplest possible setting: the critic-only, model-based, and deterministic variant of RL in continuous state space is considered. Moreover, to make performance evaluation crystally clear, the learning procedure itself is placed outside the scope of analysis. Instead, this work relies on approximately optimal learned samples computed beforehand. The discount factor $\gamma \approx 1$ is used to keep learning aspects beyond the scope. Additionally, for this analysis, it is irrelevant if the system model was given beforehand or approximated. It will be shown that for the vast majority of iterative RL techniques, the considered case corresponds to one iteration of the learning algorithm. In other words, this work concentrates only on the interconnection between the approximation error and the actual performance obtained by the approximated V-function model during policy derivation.

The policy derivation can be understood as a hill-climbing process. At each step, the agent applies the control input that leads to the highest value of the right-hand side of the Bellman equation. An advantage of this control law is its inherent stability – the value function is analogous to the control Lyapunov function [58, 59]. Various types of numerical approximators have been used: expansions with fixed or adaptive basis functions [80, 81], regression trees [82], local linear regression [83, 84], and the increasingly popular deep neural networks [85, 86, 87, 88, 89]. However, the hill-climbing process is virtually always affected by the approximation error and the lack of smoothness of the V-function.

One of the methods to construct a compact smooth approximation is symbolic regression. Based on genetic programming, SR searches for an analytic expression that best fits the given data. There are several works in the literature dealing with the use of genetic programming (GP) for V-function fitting. For instance, in [37], a method called Value Function Discovery uses GP to evolve an analytic description of the V-function. In [33] an evolutionary algorithm is used to accelerate the convergence of Q-tables. However, as demonstrated previously, it is impractical to apply SR to approximate an existing V-function directly. While the symbolic approximator can achieve a low MSE, the resulting policy can be largely sub-optimal. Altogether, it brings us to the motivation for this work.

1. Analyse the impact of approximation errors on policy derivation.
2. Propose a fitting technique that will increase the chances of obtaining a meaningful approximator.
3. Make sophisticated approximators work more or less out of the box without the exhaustive procedure of tuning parameters.

By a sophisticated approximator, this work understands any technique in which the last computation is a weighted sum of globally defined and

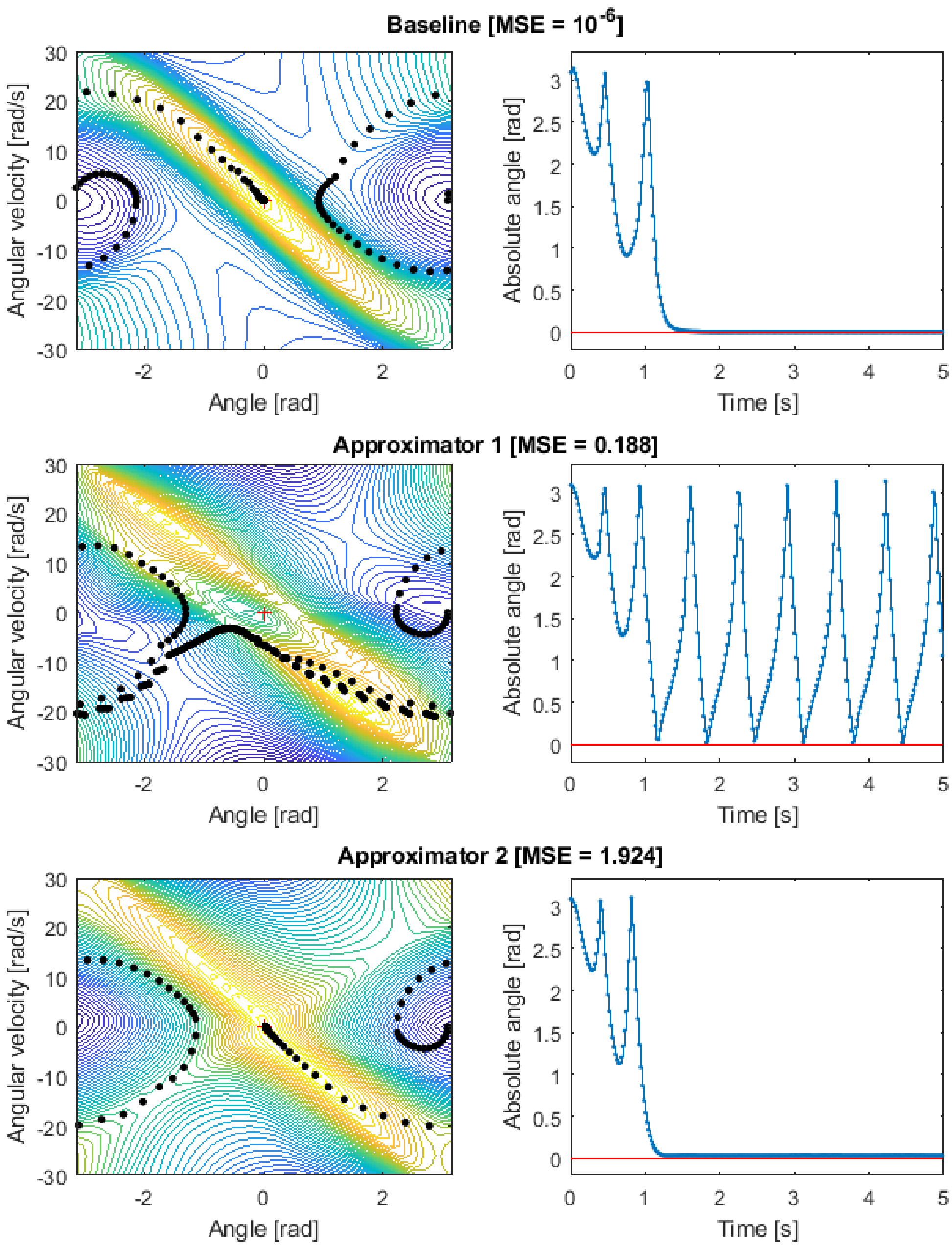

**Figure 4.1:** A sample state trajectory for the pendulum swing-up task (see Appendix A.1). Left column: the state trajectory superimposed on the right-hand side of the approximate Bellman equation. Right column: the position of the pendulum during simulation. The red line shows the goal position 0 [rad]. The top row corresponds to the simulation using a BF-based approximator (see Section 4.2.1) of the V-function with near-zero MSE. The middle row corresponds to the state trajectory derived from the approximator with a bigger approximation error. Approximator is computed by using symbolic regression technique (see Appendix B.2 for further details). The bottom row corresponds with the state trajectory obtained by the same technique with a bigger in magnitude approximation error.

possibly non-linear features. With a bit of imagination, it can be seen that this definition is wide enough to include neural networks, polynomials, basis functions, etc.

## 4.2 Preliminaries

Many different algorithms for finding an approximately optimal V-function exist. However, the vast majority of them rely on two well-known RL algorithms - value-iteration and policy-iteration. Algorithm 6 shows a generalized scheme for both algorithms. The discrete values (marked by the superscript $D$) are separated from the continuous ones (marked by the superscript $C$) to make visible what data are available in every step.

The special case is considered: only the last iteration before convergence is reached and the approximation part only. In other words, it is assumed that optimal-in-samples $V^D(\cdot)$ and $\pi^D(\cdot)$ are given. The reason for that decision is threefold:

1. The main motivation for this work is to study the impact of approximation error on the actual performance. Analysis of the interaction between approximation errors during an iterative learning process is, in general, an unanswered question yet (for the best of authors' knowledge). The formal analysis is provided only under the assumption of guaranteed contraction. For big approximation errors, however, the intuitive answer is that approximation errors between iterations may leverage each other, resulting in the negative synergy, but strong mathematical guarantees are not provided.

2. The ultimate goal of RL is to make the system behave according to the user-defined goal, and there is the only direct way to justify this - to perform simulations. Every other criterion provides indirect information about actual performance, which adds noise to the final evaluation. For simulations, the learned version of the V-function approximator is required.

3. Available variables at the moment of approximation are the same for each iteration, including the last one, as can be observed in the algorithm.

Consequently, the considered case is applicable for every iteration of an RL algorithm, but the clear evaluation is possible only for the last iteration. As a provider of approximately optimal policy, the fuzzy V-iteration algorithm Section 1.3.2 is used since it is guaranteed to converge. This algorithm produces a V-function approximator with near-zero error.

To approximate the V-function, two intrinsically different techniques are used: symbolic regression and fuzzy basis functions. Symbolic regression is based on genetic programming, and its goal is to find an equation describing given data. Symbolic regression is a suitable technique for this task, as it does not assume any kind of a priori knowledge on the shape of the V-function

**Algorithm 6:** Generalized skeleton of popular approximate RL algorithms: approximate V-iteration and approximate policy-iteration.

```
Input: $f(\cdot), \rho(\cdot), \gamma, X^D, U^D, E(\cdot), x_{des}, A$ - transition function,
       reward function, discount factor, finite set of samples drawn
       from the state space, finite number of samples drawn from
       the action space, error function, desired state, set of
       regressors, respectively.
$k \leftarrow 0$
$V_k^C \leftarrow \left[V^C : \mathcal{X} \rightarrow 0\right]$
$\pi_k^D(x) \leftarrow \underset{u \in U^D}{\operatorname{argmax}} \; [\rho(x, u, f(x, u))] \quad \forall x \in X^D$
repeat
    $V_{k+1}^D(x) \leftarrow \left[\rho(x, \pi_k^D(x), f(x, \pi_k^D(x)) + \gamma V_k^C(f(x, \pi_k^D(x)))\right] \; \forall x \in X$
    if *approximate V-iteration* then
        $\forall x \in X :$
        Compose design matrix $M^D$ and targets vector $T^D$ as:
        $M^D(x) \leftarrow A(x)$
        $T^D(x) \leftarrow V_{k+1}^D(x)$
    end
    if *approximate policy-iteration* then
        $\forall x \in X :$
        Compose design matrix $M^D$ and targets vector $T^D$ as:
        $M^D(x) \leftarrow \left[A(x) - \gamma A(f(x, \pi_k^D(x))\right]$
        $T^D(x) \leftarrow \rho(x, u, f(x, \pi_k^D(x)))$
    end
    $V_{k+1}^C \leftarrow$ approximate using $M^D, T^D, E(\cdot)$
    begin
        Data available:
        $X, U, V_{k+1}^D, V_k^C, \pi_k^D, \gamma, \rho(\cdot), f(\cdot), x_{des}, M^D, T^D$
        vary $V_{k+1}^C$ e.g. by changing parameters or/and approximation
          structure
        evaluate fitness of approximation using $E(\cdot)$
        perform until approximation stopping criterion is not satisfied
    end
    $\pi_{k+1}^D(x) \leftarrow \underset{u \in U}{\operatorname{argmax}} \left[\rho(x, u, f(x, u)) + \gamma V_{k+1}^C(f(x, u))\right] \; \forall x \in X^D$
    $k \leftarrow k + 1$
until *convergence criterion is not satisfied*;
Output: $V_k^C$
```

sought. A variant of Single Node Genetic Programming is used, which is described in details in Appendix B.2. Fuzzy basis functions are described in Section 4.2.1.

### 4.2.1 Fuzzy basis function approximation

Define a set of samples $S = \{s_1, s_2, \ldots, s_N\}$ placed on an equidistant rectangular grid in $\mathcal{X}$. The number of grid points per dimension is described by the vector $B = [b_1, b_2, \ldots, b_n]^T$ with the total number of samples $N = \prod_{i=1}^{n} b_i$. Further define a vector of fixed triangular membership functions $\phi = [\phi_1(x), \phi_2(x), \ldots, \phi_N(x)]^T$ where each $\phi_i(x)$ is centered in $s_i$, i.e., $\phi_i(s_i) = 1$ and $\phi_j(s_i) = 0, \forall j \neq i$. The basis functions are normalized so that $\sum_{j=1}^{N} \phi_j(x) = 1, \forall x \in \mathcal{X}$. For a single state variable $x_j$ these functions are defined as follows:

$$\begin{aligned}
\phi_1(x_j) &= \max\left(0, \min\left(1, \frac{s'_2 - x_j}{s'_2 - s'_1}\right)\right), \\
\phi_i(x_j) &= \max\left(0, \min\left(\frac{x_j - s_{i-1}}{s_i - s_{i-1}}, \frac{s_{i+1} - x_j}{s_{i+1} - s_i}\right)\right), \\
& \quad\; i = 2, \ldots, N_j - 1, \\
\phi_{b_j}(x_j) &= \max\left(0, \min\left(\frac{x_j - s_{b_{j-1}}}{s_{b_j} - s_{b_{j-1}}}, 1\right)\right).
\end{aligned}$$

An extension to more dimensions is implemented by using the Cartesian product of the membership functions in the individual dimensions.

## 4.3 Analysis

### 4.3.1 Notation

Assume that the perfect approximator $V^C(\cdot)$ is given. The policy derivation process can be understood as a hill-climbing process. At each step the agent starts with the state $x$ which has the value $V^C(x)$. Then the agent selects the control input which leads to the highest value of the right-hand side of the Bellman equation:

$$u^* = \operatorname*{argmax}_{u \in U^D} \left[\rho(x, u, f(x, \pi^D(x))) + \gamma V^C(f(x, \pi^D(x)))\right] \tag{4.1}$$

The selected action $u^*$ is then applied to the system, which naturally leads to the new state:

$$x^* = f(x, u^*) \tag{4.2}$$

with the new value $V^C(x^*)$. Then algorithm repeats these steps until policy derivation is over. An advantage of this control law is its inherent stability – the value function is analogous to the Control Lyapunov Function (abbreviated as CLF from now on) [58, 59].

To make the analysis easier, assume negative-definite reward function $\rho(\cdot)$ with the maximum at zero. Consequently, the optimal V-function is negative-definite with the maximum value equals zero at the goal state $x_{des}$. For simplicity, assume that the goal state $x_{des}$ is unique. However, the analysis can be intuitively modified for use with multiple goal states. The following issues may appear at the approximation step w.r.t. the policy derivation.

### 4.3.2 Ignored magnitude of errors

From Algorithm 6 it can be seen that the approximation step can be understood as an iterative process. After the approximator of the V-function is built, the quality of the resulting model is evaluated using error function $E(\cdot)$. The purpose of this function is twofold. Firstly, it computes residuals between a discrete set of samples and corresponding targets. Secondly, it provides the mapping from these residuals into a scalar value. In the vast majority of cases, this mapping is either sum or mean value. After the scalar error is produced, the approximation process can proceed further to improve this error or halt if the error is satisfactory.

Consider a set of residuals, where most residuals have low magnitude, while very few ones have a much bigger magnitude. Consequently, from the approximation process point of view, it is beneficial to ignore any improvement on most residuals and concentrate solely on the residuals with a bigger magnitude. In an extreme case, it leads to the situation where all the computational effort is spent on fitting the single sample while the vast majority of the state space is ignored.

One of the ways to deal with this drawback lies in using a mapping that takes into account different magnitudes of the residuals.

### 4.3.3 Ignored structure of the approximator

Assume that only the region near the goal state is affected by the approximation error, and the rest of the function is approximated perfectly. Then, during policy derivation, the goal state may become intractable while achieving lower approximation error in comparison with other approximators (precisely this situation is depicted in Figure 5.1). However, before the approximation process starts, it is already known that every state $x$ except $x_{des}$ should have a smaller value of $V^C(x)$ than the goal state. This knowledge can be incorporated into the approximation process without a priori information about a particular RL task.

Additionally, assume that the result approximation is fitted in a policy-iteration manner (V-iteration manner is briefly considered in subsection 4.3.5). From the Algorithm 6 it can be seen that the error function considers just the difference between $V^C(x)$ and $\gamma V^C(x^*)$, which is targeted to be equal to the reward of the corresponding state transition. Notice that the absolute values of the result approximator are not relevant here; only the difference between two different states matters. It means that the result approximator with the fixed error $\epsilon$ can be shifted along the value-axis without affecting the policy derivation. While it may be considered a good property, it has several drawbacks. First of all, it may violate the negative definiteness of the result approximation, which in many cases may lead to the divergence of the learning process. Additionally, it dramatically increases the number of local optima, making it much harder for an approximation technique to find a global one.

### 4.3.4 Ignored different error regions

As was mentioned in subsection 4.3.1, from the policy derivation point of view, both V-function and CLF work on the same principles. Intuitively, it may be described as follows. For each state $x$ except the goal state, it is possible to find an action $u^*$ that will improve the overall evaluation of the next state $x^*$. Consequently, if it is possible to find a way to improve the overall evaluation in each state, it should be possible to reach the goal state. Notice that the convention for RL is to maximize the achieved value on V-function, while the convention for CLF is to reduce the energy on CLF-function. In this work, the maximization convention is used for both approaches, leaving mathematical details about changing the sign behind the scene. More formally, the key condition for a stable policy derivation process can be described as follows:

$$\begin{aligned} V^C(x) < \gamma V^C(x^*) \quad \forall x \neq x_{des} \\ V^C(x_{des}) = 0; \end{aligned} \tag{4.3}$$

Since this paper works with approximations, satisfying this condition with near-zero error cannot be guaranteed for every $x \in \mathcal{X}$. However, strengthening this conditions at least for the given samples $x \in X^D$ should increase the chances of obtaining the reasonable approximation.

Consider the approximate policy-iteration approach in Algorithm 6 using this framework. It can be seen that the ideal approximation should satisfy the Bellman condition of optimality:

$$V^C(x) - \gamma V^C(x^*) = \rho(x, u^*, x^*); \quad \forall x \in X^D \tag{4.4}$$

Consider one sample state with added approximation error term $\epsilon$. Define $\Delta(x)$ as:

$$\Delta(x) = V^C(x) - \gamma V^C(x^*) \tag{4.5}$$

Then the equation 4.4 become:

$$\underbrace{\Delta(x)}_{\text{regressed part}} + \underbrace{\epsilon}_{\text{error}} = \underbrace{\rho(x, u^*, x^*)}_{\text{targets}} \tag{4.6}$$

Since the reward function $\rho(\cdot)$ is negative-definite, it can be seen that in the case of ideal approximation ($\epsilon$ is zero) the condition 4.3 is satisfied-in-samples: $\Delta(x) \leq 0$ and is equals exactly to the $\rho(\cdot)$. It ensures both *stability-in-samples* (by satisfying condition 4.3) and *optimality-in-samples* (by satisfying condition 4.4). Let us analyse the impacts of possible violation of these conditions for one sample pair $x$ and $x^*$. The following cases are possible:

1. $\epsilon > 0$: In this case, the regressed part becomes smaller which means that the difference between $V^C(x)$ and $\gamma V^C(x^*)$ is becoming bigger. Condition 4.3 is not violated since $\Delta(x)$ is still less than zero, only condition 4.4 is not held. However, since the reward function is negative-definite and samples of the V-function are basically computed as a sum

of discounted rewards as $\gamma^0\rho(\cdot)_1 + \gamma^1\rho(\cdot)_2 + ... + \gamma^n\rho(\cdot)_n$, it can be seen that increasing the difference between $V^C(x)$ and $\gamma V^C(x^*)$ is equivalent of just using the bigger discount factor $\gamma$.

2. $\epsilon = 0$: In this case, both conditions 4.3 and 4.4 are satisfied - it is an ideal scenario.

3. $\epsilon < 0$: This scenario decreases the quality of the policy derivation. Firstly, analogously to the first case, negative error reduces discount factor $\gamma$, making horizon smaller. Secondly, recalling that the reward function is negative-definite and the ideal distance between $V^C(x)$ and $\gamma V^C(x^*)$ equals $\rho(x, u^*, x^*)$, it is possible to split this case into two branches:

   - $\epsilon \leq \rho(x, u^*, x^*)$: The worst scenario - both conditions 4.4 and 4.3 are violated. For policy derivation, it means that the system will have both suboptimal *and* possibly unstable behavior. Altogether it may drastically decrease the performance of the policy derivation by making the goal state intractable. Naturally, this scenario should be avoided in the first place.
   - $\epsilon > \rho(x, u^*, x^*)$: In that case only condition 4.4 is violated, condition 4.3 is still held. In practice, it means that policy derivation may lead to suboptimal policy, but the system itself is stable-in-samples and has better chances to reach the goal state. While this case should be avoided as well, it has a smaller negative impact on the overall performance.

However, typical error function in approximate RL is symmetric, like L2 or L1 norm. These popular criteria naturally ignore different effects of the approximation error. Mentioned regions of errors together with popular error functions $E(\cdot)$ are depicted in Figure 4.2.

### 4.3.5 Ignored policy data

From the Algorithm 6, it can be seen that in the case of V-iteration, the standard approach is to approximate $V^C(\cdot)$ using discrete target values of the best known V-function values. Naturally, this case completely ignores the current optimal policy and does not check if this policy can be derived from the result approximation. Assuming an existence of a significant approximation error, from the policy derivation point of view, the usability of the resulting approximation is a question of luck. Therefore, V-iteration-like approximators are left for future work.

## 4.4 Proposed solutions

This section formalizes the results of the analysis in a way that is applicable to merely any approximation technique. The rest of the section is then dedicated to effectively integrating the derived conditions for linear-in-parameters

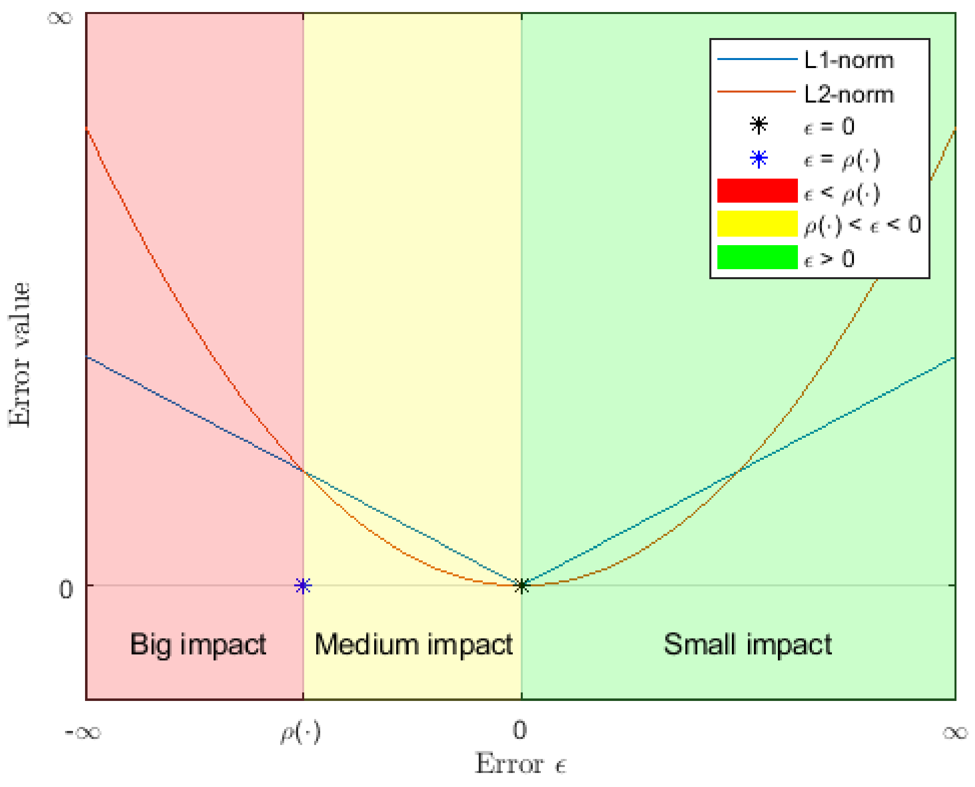

**Figure 4.2:** Popular error functions, superimposed on different error regions. The red area indicates the worst case: the approximation error is negative and smaller than the appropriate reward. The yellow area represents the case with medium negative impact: the approximation error is still negative but is bigger than the reward. The green area demonstrates the case with the smallest impact: the approximation error is positive.

approximation techniques. To evaluate the results, the SNGP technique was used (see Appendix B.2 for further details) as a generator of features. However, the proposed method applies to many types of approximation techniques like, e.g., neural networks.

More specifically, by linear-in-parameters approximation is understood an approximation in the following form:

$$V^C(x) = \beta_0 + \beta_1 p_1(x) + \beta_2 p_2(x) + \ldots + \beta_q p_q(x) \tag{4.7}$$

where $p_1, \ldots, p_q$ are continuous functions with any structure, each of them is defined over the whole state space and $\beta_0, \ldots, \beta_q$ are real-valued coefficients.

The issue with different magnitudes of samples (see subsection 4.3.2) can be solved by applying different weights to each observation. The key idea is to penalize the residual of the approximated value w.r.t. the target value using the percentage-like measure. In other words, a 10% decrease of residual in one sample is always evaluated better in comparison with a 5% decrease of another sample, independently of their magnitudes. In general, computing a residual between approximated value $V^C(x)$ and target value $T^D(x)$ may be described as:

$$\epsilon = \left[T^D(x) - V^C(x)\right] Q(x)$$

where $Q(\cdot)$ returns weight for a sample, and normally it is the identity function. The following redefinition of the function $Q(\cdot)$ equalizes the difference in magnitudes:

$$Q(x) = 1/abs(T^D(x)) \tag{\textbf{A}}$$

A careful reader may notice that this way of equalizing samples cannot be held in the case of $T^D(x) = 0$. In this work, the only sample that suffers from the mentioned drawback is the goal state $x_{des}$. In this particular case, an artificial value can be used instead. Additionally, using absolute values instead of the original $T^D(x)$ does not change signs of samples, which will become important later on. The proposed equalization of magnitudes is named Condition **A**.

The issue with the ignored structure of the approximation (see subsection 4.3.3) is easy to solve using constraints. The following set of inequalities guarantees that the resulting approximation will have the maximum at zero, and all other samples will be lower than the maximum value.

$$\begin{gathered} V^C(x_{des}) = 0 \\ V^C(x) - V^C(x_{des}) < \omega;\ \forall x \neq x_{des} \end{gathered} \tag{\textbf{B}}$$

where $\omega$ is a small negative constant. The proposed set of structural constraints is named Condition **B**.

The issue with ignored different regions can be solved by penalizing approximation error using different weights w.r.t the corresponding region. Instead of one error variable in (4.6) the approximation error term can be split into three errors as:

$$\begin{gathered} \epsilon = \begin{cases} w_1\epsilon_1 & \text{if } \epsilon_1 \geq 0 \\ w_2\epsilon_2 & \text{if } \rho(x, u^*, x^*) < \epsilon_2 < 0 \\ w_3\epsilon_3 & \text{if } \epsilon_3 \leq \rho(x, u^*, x^*) \end{cases} \\ w_1 \ll w_2 \ll w_3 \end{gathered} \tag{\textbf{C}}$$

where each case corresponds to the appropriate error region depicted in Figure 4.2. The proposed split of the approximation error is named Condition **C**.

The proposed method consists of applying these conditions for the linear-in-parameters fitting procedure. To find a $V^C$, or its close approximation, a set of $N$ current state samples $X^D$ is generated as:

$$X^D = \{x_i^c \mid x_i^c \in \mathcal{X}, i = 1, 2, \ldots N; x_i^c \neq x_{des}\} \tag{4.8}$$

and the set of next states is constructed by applying approximately optimal action as:

$$X^N = \{x_i^* \mid x_i^* = f(x_i^c, u_i^*), i = 1, 2, \ldots N\} \tag{4.9}$$

Additionally, the set of appropriate rewards $\hat{R}$ is constructed as:

$$\hat{R} = \{r_i \mid r_i = \rho(x_i^c, u^*, x_i^*), i = 1, 2, \ldots N\} \tag{4.10}$$

Define matrices $M^c$ and $M^n$ for current and next states and vector $R$ for the corresponding rewards, respectively, as follows:

$$\begin{aligned}
M^c &= \begin{pmatrix} 1 & p_1(x_1^c) & p_2(x_1^c) & \cdots & p_q(x_1^c) \\ 1 & p_1(x_2^c) & p_2(x_2^c) & \cdots & p_q(x_2^c) \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & p_1(x_N^c) & p_2(x_N^c) & \cdots & p_q(x_N^c) \end{pmatrix} \\
M^n &= \begin{pmatrix} 1 & p_1(x_1^*) & p_2(x_1^*) & \cdots & p_q(x_1^*) \\ 1 & p_1(x_2^*) & p_2(x_2^*) & \cdots & p_q(x_2^*) \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & p_1(x_N^*) & p_2(x_N^*) & \cdots & p_q(x_N^*) \end{pmatrix} \\
R &= [r_1, r_2, \dots, r_N]^T
\end{aligned} \tag{4.11}$$

where $N$ is the total number of states in $X^D$ and $X^N$ and the first column in matrices stands for bias.

To calculate the difference $\Delta(x)$ (see (4.5)) for all the data in a matrix form, define

$$O = M^c - \gamma M^n \tag{4.12}$$

In order to describe Condition **B**, define vector $T^c_{des}$ and matrix $M^c_{des}$ as:

$$\begin{aligned} T^c_{des} &= [1,\ p_1(x_{des}), \dots p_q(x_{des})] \\ M^c_{des} &= J_{N,q} \cdot diag(T^c_{des}) \end{aligned} \tag{4.13}$$

where $J_{N,q}$ is all-ones matrix of sizes $N \times q$ and function $diag(\cdot)$ creates a diagonal matrix out of the argument vector.

Condition **B** is then can be described as:

$$\begin{aligned} T^c_{des}\beta &= 0 \\ [M^c - M^c_{des}]\beta &< \omega \end{aligned} \tag{4.14}$$

where $\omega$ is a small negative constant.

In order to describe Condition **C** define three vectors of free variables as:

$$\begin{aligned} s^{--} &= [s_1^{--}, \dots, s_N^{--}]^T \\ s^{-} &= [s_1^{-}, \dots, s_N^{-}]^T \\ s^{+} &= [s_1^{+}, \dots, s_N^{+}]^T \end{aligned} \tag{4.15}$$

In order to describe Condition **A** effectively, define the diagonal matrix $L$:

$$L = diag(Q(r_1), Q(r_2), \dots, Q(r_N)) \tag{4.16}$$

The fitting problem, which includes all three conditions, is now can be formulated as:

$$\begin{aligned}
&\min_{\beta, s^{--}, s^{-}, s^{+}} -w_3 \sum_{i=1}^{N} s_i^{--} - w_2 \sum_{i=1}^{N} s_i^{-} + w_1 \sum_{i=1}^{N} s_i^{+} \quad \text{s. t.} \\
&\textit{Condition } \mathbf{A} \begin{cases} \hat{T} = LR \\ \hat{O} = LO \end{cases} \\
&\textit{Condition } \mathbf{B} \begin{cases} T^c_{des}\beta = 0 \\ [M^c - M^c_{des}]\beta < \omega \end{cases} \\
&\textit{Condition } \mathbf{C} \begin{cases} \hat{O}\beta + s^{--} + s^{-} + s^{+} = \hat{T} \\ -\infty \leq s^{--} \leq 0 \\ \hat{T} \leq s^{-} \leq 0 \\ 0 \leq s^{+} \leq \infty \\ w_1 \ll w_2 \ll w_3 \end{cases}
\end{aligned} \tag{4.17}$$

This formulation adds free variables to every equality, namely $s^{--}$, $s^{-}$, and $s^{+}$. These variables correspond to the relevant regions of the error space. Variables are then optimized using linear programming. This formulation with all defined conditions is named Mode**ABC** from now on.

Additionally, let us formulate several reduced versions of the above minimization. For these formulations, it is possible to use standard MSE fitness criterion, since there is no dependence on the sign of the residuals. Define the skeleton as:

$$\begin{aligned}
&\min_{\beta} \frac{1}{2}\|O\beta - R\|_2^2 \quad \text{s. t.} \\
&\textit{Condition } \mathbf{A} = \begin{cases} O \leftarrow LO \\ R \leftarrow LR \end{cases} \\
&\textit{Condition } \mathbf{B} = \begin{cases} T^c_{des}\beta = 0 \\ [M^c - M^c_{des}]\beta < \omega \end{cases}
\end{aligned} \tag{4.18}$$

From this skeleton, Mode**AB**, Mode**A**, and the Baseline can be derived by applying a different number of conditions.

Now, when all ingredients are ready, the whole algorithm can be constructed. The block scheme is given in Figure 4.3.

Two different approximation techniques are employed to construct the continuous functions $A$: either fuzzy basis functions (see Section 4.2.1) or SNGP (see Appendix B.2-SNGP). However, any other suitable technique can be used instead, e.g., neural network. The whole algorithm repeats until a stopping criterion is satisfied. It can be either a predefined number of iterations or an improvement threshold.

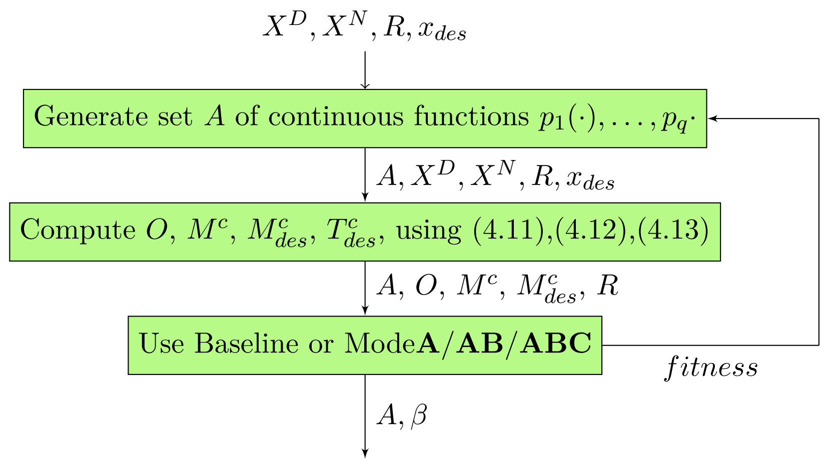
$X^D, X^N, R, x_{des}$
Generate set $A$ of continuous functions $p_1(\cdot), \dots, p_q\cdot$
$A, X^D, X^N, R, x_{des}$
Compute $O$, $M^c$, $M^c_{des}$, $T^c_{des}$, using (4.11),(4.12),(4.13)
$A$, $O$, $M^c$, $M^c_{des}$, $R$
Use Baseline or Mode**A**/**AB**/**ABC**
$fitness$
$A, \beta$

**Figure 4.3:** Block-scheme of the $V^C(\cdot)$ approximation algorithm.

## 4.5 Experimental evaluation

The proposed methods have been tested on two different benchmarks: the well-known pendulum swing-up and a non-linear control problem called magnetic manipulation (abbreviated as magman). Two different approximation architectures have been used: symbolic regression by means of SNGP and numerical approximation by means of fuzzy basis functions (abbreviated as BFs). Each combination is then described by the abbreviation in the form <approximation>_<method>, where <approximation> can take values *SNGP* or *BFs*, and <method> stands for Baseline, Mode**A**, Mode**AB**, Mode**ABC**.

For each combination with BFs approximation architecture, 26 different $V^C(\cdot)$ models have been constructed and tested. The appearance of different approximation errors was ensured by using a gradually decreased number of BFs while keeping the number of samples the same for all experiments. For each combination of SNGP approximators, 30 different $V^C(\cdot)$ models have been constructed and tested.

All tested combinations within a benchmark use identical parameters and identical datasets to exclude the influence of these factors in comparison. The only difference is the fitting method used.

The approximately optimal policy has been constructed beforehand by means of the fuzzy V-iteration algorithm (see Section 1.3.2 for further details). The discount factor $\gamma$ is chosen close to 1, to avoid the influence of not properly learned V-functions. Additionally, all initial states were tested on a fuzzy approximator to ensure the reachability of the goal state.

Every combination has been tested through simulations with $K = 100$ randomly chosen initial states, which are the same for each combination.

It should be noted that all the models $V^C(\cdot)$ are presented "as is", which means that there is no selection procedure w.r.t. some criterion.

To measure the performance of one approximator, the following criteria are defined:

- MSE, denoted as $MSE_a$.
- *Original* fitness, denoted as $F_a$. By original is understood the fitness which was obtained directly from the method used. It equals to $MSE_a$ in the case of Baseline method.
- Median of returns, abbreviated as $MRet_a$:

  $$MRet_a = median(\Sigma^1, \Sigma^2, \dots, \Sigma^i)$$

  where

  $$\Sigma^i = \underbrace{\sum_{q=1}^{T_{sim}/T_s} \rho(x_q, u_q, f(x_q, u_q))}_{\text{one simulation}}$$

  where $T_{sim}$ and $T_s$ are the total simulation time and the sampling period, respectively, and index $i$ stands for simulation number from 1 to $K = 100$.
- Number of successful simulations, denoted as $S_a$. Simulation is considered successful if the last 25 steps of the trajectory lie within 5% interval around the goal state $x_{des}$. An example of the successful trajectory and appropriate interval is given on the Figure 4.4.

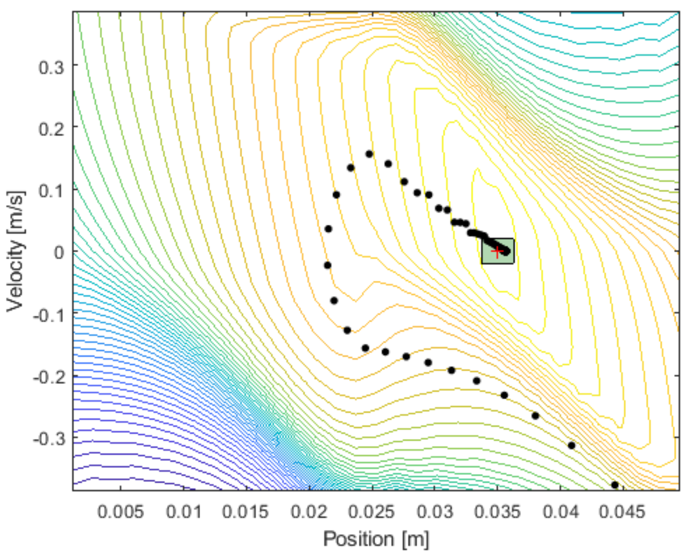

**Figure 4.4:** The successful state trajectory superimposed on the right-hand side of the Bellman equation level curves for magman benchmark (see Section 4.5.2 for further details). The green rectangle represents 5% interval around the goal state. If trajectory ends up within this interval, it is considered to be successful.

To measure the performance of the methods themselves, the following criteria are defined:

- Mean MSE within all fitted approximators, abbreviated as $MSE_m$.

- Median of returns from all approximators, abbreviated as $MRet_m$.
- Percentage ratio of successful simulations within all fitted approximators, abbreviated as $S_m$.
- Median of successful simulations within all fitted approximators, abbreviated as $\hat{S}_m$.

### 4.5.1 Pendulum swing-up

The inverted pendulum consists of a mass attached to an actuated link that rotates in the vertical plane. The available torque is insufficient to push the pendulum up in a single rotation from the majority of initial states. Instead, from a certain state (e.g., pointing down), the pendulum needs to be swung back and forth to gather energy before it can be pushed up and stabilized. Full description alongside technical details about the benchmark can be found in Appendix A.1. The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.02\,\text{s}$.

The control goal is to stabilize the pendulum in the unstable equilibrium $x_{des} = [0, 0]$ (rad, rad/s), which is expressed by the following reward function:

$$\rho(x, u, f(x, u)) = -abs(x_{des}^T - f(x, u))Q \tag{4.19}$$

with $Q = [1, 0.01]^T$ a weighting vector to adjust the relative importance of the angle and angular velocity and $abs(\cdot)$ function working element-wise. The complete list of experiment parameters is listed in Table 4.1.

The results, shown in the Table 4.2, support the hypothesis that pure MSE is not the best criterion to predict the quality of models. The vast majority of models obtained by the Baseline method are effectively useless for both approximation techniques, despite their low MSE. Additionally, for both approximation techniques, satisfying Condition **A** solely leads to the increased by order of magnitude $MSE_m$, but drastically increases the quality of the result models.

For BFS-based approximations, there is no significant difference between Mode**A** and Mode**AB**. The reason for it is that the BFs-based approximator has a basis function exactly at the goal state. Consequently, it helps to form a global optimum at the right place. Since enforcing the global optimum at the goal state is the only difference between the two methods, the benefits of Mode**AB** are deactivated for this type of approximation architecture. However, a significant difference between Mode**A** and Mode**AB** can be seen in the case of SNGP approximations. Satisfying both Conditions **A** and **B** leads to further improvement while MSE becomes larger again. Mode**ABC** indicates to have the best result in both approximation techniques. Figure 4.5 depicts the distribution of the SNGP-based results w.r.t. their median of returns $MRet_a$ and original fitness $F_a$. In the figure, it can be seen

| **Fuzzy V-iteration parameters** | |
|---|---|
| State space, $\mathcal{X}$ | $[-\pi, \pi] \times [-30, 30]$ |
| Input space, $\mathcal{U}$ | $[-2, 2]$ |
| Action samples per dimension, $B_U$ | 11 |
| Discount factor, $\gamma$ | 0.95 |
| Convergence threshold, $\epsilon$ | $10^{-4}$ |
| Desired state, $x_{des}$ | $[0, 0]^T$ |
| Sampling period, $T_s$ [s] | 0.02 |
| Simulation time, $T_{sim}$ [s] | 3 |
| **SNGP-related parameters** | |
| Population size | 1000 |
| Elementary functions | +, -, $\times$, $x^2$, $x^3$, BentGeneral, Logistic3 |
| Maximal depth of features | 5 |
| Maximal number of features | 30 |
| Epoch length | 500 |
| Local search iterations | 500 |
| Number of epochs | 2 |
| Number of threads | 1 |
| State samples per dimension, $B_X$ | $[31, 31]$ |
| **BFs-related parameters** | |
| State samples per dimension, $B_X$ | $[61, 61]$ |
| Number of BFs on grid | $61 \times 61$, $59 \times 59$, $\ldots$, $11 \times 11$ |
| **ModeA/AB/ABC parameters** | |
| Small negative constant $\omega$ | $-0.001$ |
| Big impact error weight $w_3$ | 100 |
| Medium impact error weight $w_2$ | 10 |
| Small impact error weight $w_1$ | 1 |

**Table 4.1:** Pendulum swing-up experiment parameters

that the standard baseline technique has a chaotic distribution of the well-performed approximators. However, according to the distribution, both Mode**A** and Mode**AB** improve the probability of *all* approximators to be relatively successful, not the approximators with good fitnesses. Meanwhile, Mode**ABC** demonstrates the ability to greatly reduce the number of bad approximators with good fitness. Altogether it results in a remarkably better median number of successful simulations in comparison with the Baseline method.

### 4.5.2 Magnetic manipulation

Magnetic manipulation (abbreviated as Magman) is a challenging non-linear control problem. The current through the electromagnets is controlled to dynamically shape the magnetic field above magnets, which allows to accurately and quickly control the position of a steel ball. For the experiments presented in this work, first three coils at 0, 0.025, and 0.05 (m), respectively, have been

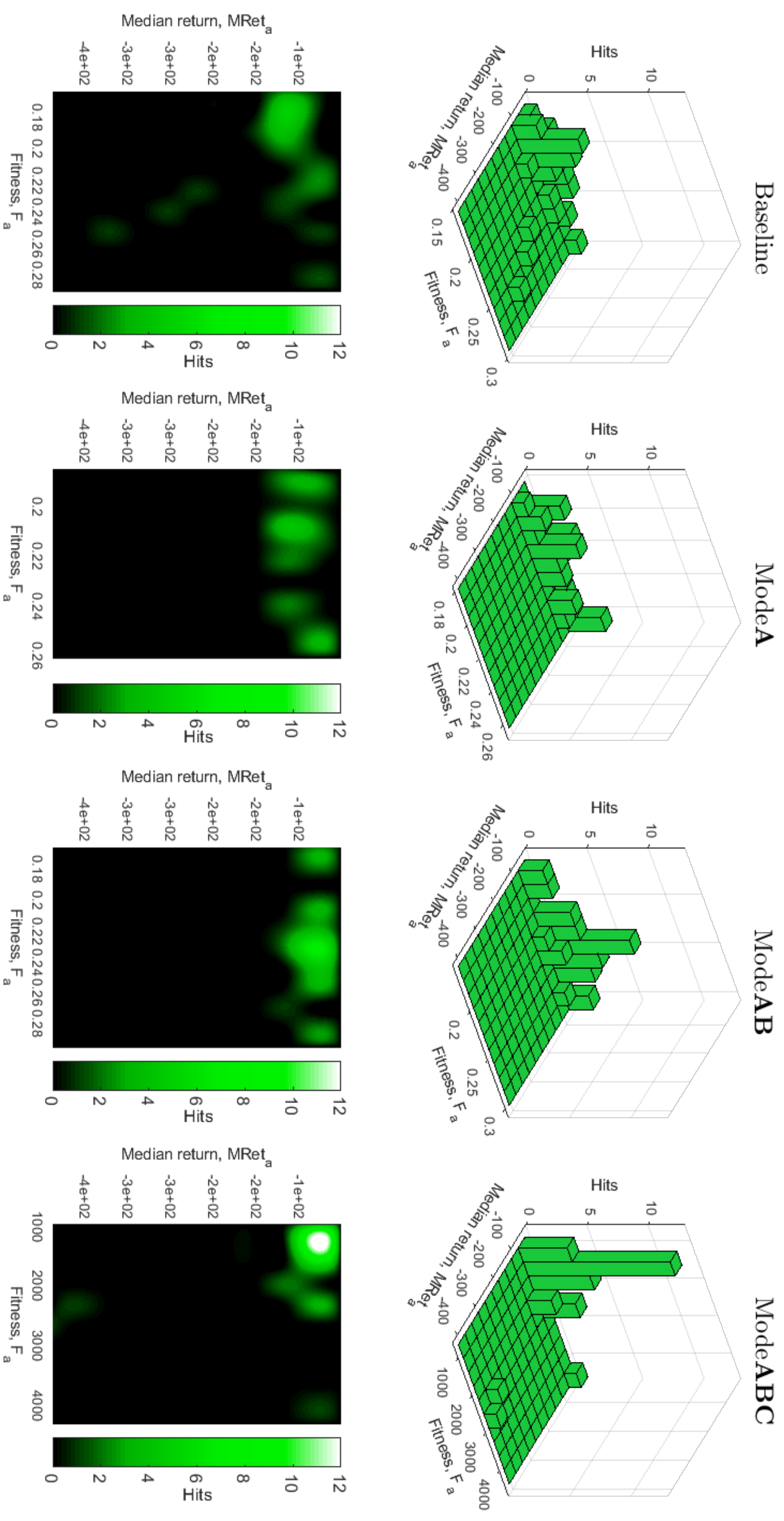

**Figure 4.5:** Distribution of the result models for pendulum swing-up benchmark using SNGP approximation technique. The top row represents the histogram of the obtained models for each method. The bottom row shows a density projection of the models.

| | $MSE_m$ | $MRet_m$ | $S_m$ | $\hat{S}_m$ |
|---|---|---|---|---|
| BFs_Baseline | 0.0299 | -91.948 | 3.84 | 0 |
| BFs_Mode**A** | 0.2532 | -53.692 | 90.6 | 100 |
| BFs_Mode**AB** | 0.2539 | -53.934 | 90.7 | 100 |
| BFs_Mode**ABC** | 0.2476 | -52.276 | 92.0 | 100 |
| SNGP_Baseline | 0.1937 | -76.697 | 41.7 | 0 |
| SNGP_Mode**A** | 1.5772 | -71.742 | 52.0 | 75 |
| SNGP_Mode**AB** | 1.8198 | -62.516 | 76.8 | 99 |
| SNGP_Mode**ABC** | 6.1889 | -59.619 | 93.7 | 99 |

**Table 4.2:** Pendulum swing-up experiment results

used. Full description alongside technical details about the benchmark can be found in Appendix A.3. State $x$ is given by the position and velocity of the ball. The control input $u$ is defined as the vector of currents $[u_1 u_2 u_3] \in [0, 0.6]$ on the coils. The reward function is defined as:

$$\rho(x, u, f(x, u)) = -abs(x_{des}^T - f(x, u))Q, \text{ with } Q = [10, 1]^T \tag{4.20}$$

where the desired position $x_{des}$ is set to $[0.035, 0]$ (m, m/s), $Q$ is a weighting vector and function $abs(\cdot)$ works element-wise. The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.005\,\text{s}$. The set of control inputs $U$ is defined as the Cartesian product of the vectors $[0, 0.3, 0.6]^T$, containing the discrete values of the control input to the $i$th coil.

Parameters for fuzzy V-iteration are listed in Table 4.3, while all other parameters are the same as for the pendulum swing-up setting.

| **Fuzzy V-iteration parameters** | |
|---|---|
| State space, $\mathcal{X}$ | $[0, 0.05] \times [-0.4, 0.4]$ |
| Input space, $\mathcal{U}$ | $[0, 0.6]$ |
| State samples per dimension, $B_X$ | $[31, 31]$ |
| Action samples per dimension, $B_U$ | $[3, 3, 3]$ |
| Discount factor, $\gamma$ | 0.95 |
| Convergence threshold, $\epsilon$ | $10^{-4}$ |
| Desired state, $x_{des}$ | $[0.035, 0]$ |
| Sampling period, $T_s$ [s] | 0.01 |
| Simulation time, $T_{sim}$ [s] | 3 |

**Table 4.3:** Magnetic manipulation experiment parameters

The results, shown in Table 4.4, demonstrate the same pattern as for the pendulum swing-up benchmark, except Mode**AB**. The Baseline method did not succeed in finding a single well-performed model while achieving an impressively low $MSE_m$ in the case of the SNGP approximation technique. After applying Condition **A** it became significantly more successful. Figure 4.6 shows the comparison of the found BFs approximators w.r.t. to their median returns and MSEs. For both approximation techniques, there is no

| | $MSE_m$ | $MRet_m$ | $S_m$ | $\hat{S}_m$ |
|---|---|---|---|---|
| BFs_Baseline | 0.0002 | -32.678 | 4.61 | 0 |
| BFs_Mode**A** | 0.0004 | -5.743 | 74.07 | 96.5 |
| BFs_Mode**AB** | 0.0004 | -5.752 | 74.11 | 96.5 |
| BFs_Mode**ABC** | 0.0019 | -4.800 | 89.03 | 100 |
| SNGP_Baseline | 1.4886 | -640.90 | 0 | 0 |
| SNGP_Mode**A** | 2.2351 | -12.445 | 17.36 | 7 |
| SNGP_Mode**AB** | 2.3509 | -11.876 | 15.9 | 7 |
| SNGP_Mode**ABC** | 14.467 | -7.749 | 53.66 | 56.5 |

**Table 4.4:** Magman experiment results

significant difference between Mode**A** and Mode**AB**. For the BFs technique, all the benefits of using Mode**AB** are again deactivated due to the specific architecture of the approximator. Meanwhile, it can be seen that the majority of Mode**ABC** approximators consistently find better trajectories w.r.t. other methods.

## 4.6 Discussion

The proposed methods show the potential to significantly outperform the baseline approach as shown in Tables 4.2 and 4.4. To the best of the authors' knowledge, it is a first attempt to analyze the direct influence of the approximation error on the policy derivation. The concrete algorithms are given for the linear-in-parameters approximations. However, the analysis itself holds for other architectures as well.

The proposed methods have several limitations.

- It requires the knowledge and the special design of the reward function: with the goal state equals exactly zero. Despite that the reward function is, in practice, given by the experimenter, it is a part of the environment in traditional RL. However, if the bounds of the reward function are known, it is still possible to apply the proposed method by simply translating the rewards to have goal or goal states in zero.

- This work assumes that all goal states are equally important, and the V-function in these states will be equal to zero once learning is ready. Despite that this case is relatively frequent and the majority of tasks can be transformed into this case, exceptions are still possible.

- The third limitation is rather practical than theoretical. It is still unclear how to use the proposed methods for the learning procedure itself. Although the output models obtained by the proposed methods work significantly better than the baseline models, the approximation error is still much more prominent than zero. Intuitively, approximation errors between iterations will leverage each other in iterative approximate RL algorithms, resulting in negative synergy. One of the possible ways

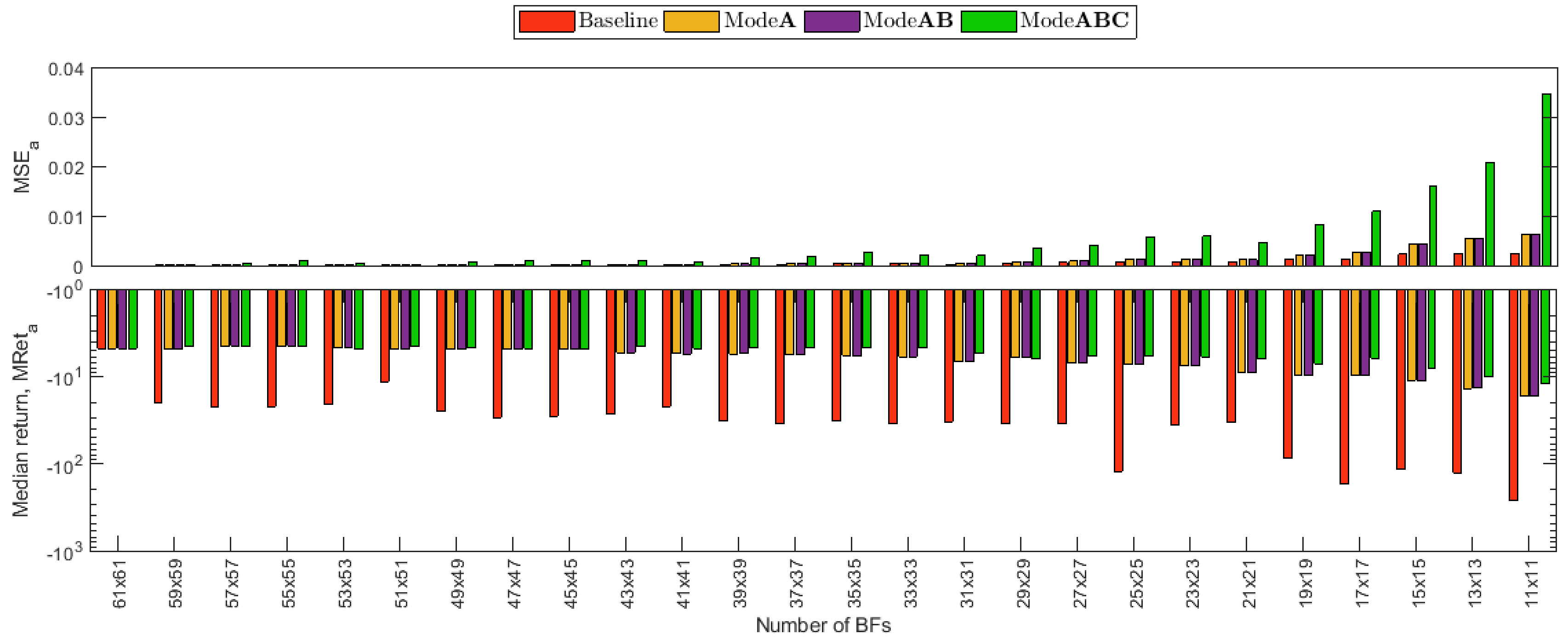

**Figure 4.6:** Comparison of the result BFs approximators for magnetic manipulation benchmark. The bottom plot compares median returns, while the top plot shows the corresponding MSEs.

to overcome that limitation lies in computing V-function at once, using approximate linear programming in opposite to iterative methods (see [78] for further details).

- The last limitation is that it is unclear how to apply the proposed analysis to V-iteration-like algorithms. Only Conditions **A** and **B** are directly applicable to this approximation style. However, it is still not clear how to apply Condition **C** (which exploits, in fact, desired policy) to the method which does not use policy in the approximation process. Overcoming this limitation may be a part of future work.

## 4.7 Conclusion

This work attempts to formalize the influence of approximation error on policy derivation in the RL domain. The results show the potential to significantly outperform the standard MSE-based technique as shown in Tables 4.2 and 4.4. The reason is that the proposed methods redistribute the approximation error w.r.t. the policy derivation process. Altogether, it resulted in the successful fitting of symbolic V-function approximators. This achievement answers the **RQ 3** and the unanswered part of the **RQ 2**.

The proposed analysis applies to any kind of value function approximation technique. Incorporating the proposed methods directly into reinforcement learning may be a part of future work. The proposed analysis may help to leverage the performance of a wide variety of methods.

# Chapter 5

# Efficient Symbolic Regression for Reinforcement Learning

Assume we have just the reward function and dynamics function. How can we learn V-function symbolically?

*Author, after countless nights of processing the thought: "It should be possible!"*

## 5.1 Analysis of the current state

Before building an efficient SR for RL framework, it is beneficial to identify the desired properties of such a method. Key desired properties are listed below:

- Efficient sample collection – as a generative technique, SR relies on an evaluation of all given samples for each generated candidate solution. As the dimensionality of the task grows, exhaustive evaluation may quickly become prohibitive. However, many papers (e.g., [90, 91, 92]) claim that SR is efficient at generalization from few number of samples.
- Incorporate region importance – as discussed in Section 3.1, not all regions of the state space are equally important for the resulting policy. The ideal SR for RL method should identify these key areas.
- Continuous action space should be used – all previous chapters relied on the discretization of the action space at the learning time. Despite that it is a working strategy to learn a V-function, it may lead to undesired behavior. Additionally, using discrete actions in the learning process goes in contradiction to the goals of the thesis.
- Computational feasibility – the most trustworthy criterion of the performance could be achieved virtually exclusively through simulations.

For that purpose, symbolic approximators should be produced within a reasonable time frame, given the available computational budget.

Previous research is systematized and mapped onto the taxonomy depicted in Figure 5.1.

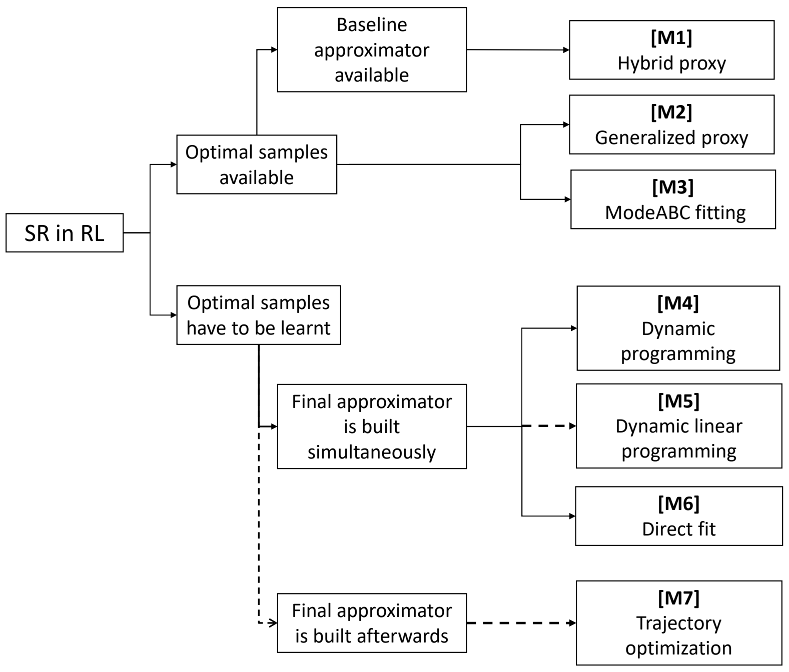
SR in RL
Optimal samples available
Baseline approximator available
[M1] Hybrid proxy
[M2] Generalized proxy
[M3] ModeABC fitting
Optimal samples have to be learnt
Final approximator is built simultaneously
[M4] Dynamic programming
[M5] Dynamic linear programming
[M6] Direct fit
Final approximator is built afterwards
[M7] Trajectory optimization

**Figure 5.1:** Taxonomy of the branches of research: previous and potentially useful combined.

All invented or researched methods are analyzed through the desired properties derived above. In addition, potentially useful branches of research are added for the sake of the same analysis. Analyzed methods are listed below:

**M1.** Hybrid proxy (introduced in Section 3.1.4) justifies the necessity of having region importance but not solving the issue. Moreover, it explicitly relies on discrete actions and corresponding transitions, collected uniformly from the state space. While the exhaustive sampling strategy potentially could be redesigned, other desired properties are violated beyond repair.

**M2.** Generalized proxy (introduced in Section 3.2) could potentially be used to work with continuous action space if the knowledge about continuous optimal actions is extracted from the underlying approximator. Multiple policy derivation methods may ensure it. However, such an underlying approximator should be learned beforehand. Moreover, it relies on uniform sampling in the same manner as a Hybrid proxy. These two drawbacks are critical and cannot be easily fixed.

**M3.** Mode**ABC** fitting procedure(introduced in Section 4.4) can be adapted for continuous action space, depending on the knowledge of optimal actions. Additionally, it partially exploits the usage of the importance of different regions. Additionally, it is independent of the sampling strategy, but the efficiency of different sampling was never tested. Similarly to **M2**, the main drawback is the lack of known optimal actions beforehand.

**M4.** Dynamic programming approach for RL is the most convenient one among others. It does not necessarily depend on the exhaustive samples collection, it can potentially gain information about region importance using prioritized sweeping [93], and may work with continuous action spaces directly. However, in combination with SR, it has severe drawbacks. Dynamic programming requires a fitted approximator for each iteration. Since fitting a symbolic approximator requires significant time, it alone may render using SR ineffective. Additionally, as was analyzed previously (see Section 4.5 for further details), even with symbolic approximation at the last stage, it is prone to suboptimal policies. Given the fact that the dynamic programming approach consists of many iterations, intermediate suboptimal policies may render the final solution unsatisfactory both from the quality and computational time perspectives.

**M5.** Dynamic linear programming casts RL problem to a single optimization task, which is then solved using linear programming [94, 95, 96]. It is not limited to the exhaustive sample collection but may work neither with prioritized sweeping nor continuous actions. Moreover, the dynamic linear programming approach in combination with SR requires solving optimization tasks for every candidate solution. This property makes using SR impractical from the computational point of view.

**M6.** Direct fit of the Bellman equation requires a calculation of the next optimal action for each sample and for each candidate approximator. The most suitable method, *QSPD* (introduced in Section 2.2.5), requires partial symbolic derivatives to work efficiently. Therefore, these derivatives are required for each candidate solution. Computing symbolic derivatives is a time-consuming process, making the whole approach computationally expensive. Moreover, the direct fit of the Bellman equation provides no information about the importance of different regions, which may render the final approximator suboptimal.

**M7.** Trajectory optimization is a method of finding feasible and optimal trajectories from the selected starting point. Unlike RL, trajectory optimization does not require samples of the whole state space since it does not produce V-function. Additionally, trajectory optimization works inherently with continuous action space. There are known examples of finding trajectories in high-dimensional tasks [97, 98, 99]. These properties provide a reason to analyze the potential of using trajectory optimization techniques combined with SR in RL.

Based on this quick analysis, it is possible to exclude methods **M1**, **M2**, **M4**, **M5** and **M6** from the following work, and concentrate on the rest. Additionally, *QSPD* algorithm has the potential to be incorporated into the final design, assuming successful fitting of a symbolic V-function approximator.

## 5.2 Trajectory optimization sampling

### 5.2.1 Proposed method

Trajectory optimization is a wide and well-studied branch of methods of finding feasible and optimal trajectories. Unlike RL, which looks for a closed-loop solution, trajectory optimization seeks an open-loop solution to an optimal control problem. While it does not produce V-function *per se*, trajectory optimization can be utilized as a sampling technique. In Chapter 4, it was discovered that Mode**ABC** fitting procedure is able to fit a symbolic V-function out of the limited number of samples from the optimal V-function. These samples should be in the form of tuples $(x_k, f(x_k, u_k^*), \rho(x_k, u_k^*, f(x_k, u_k^*)))$, where $x_k$ represents some state and $u_k^*$ stands for the optimal action for this state. Luckily, the output of trajectory optimization can be transformed to the form of such tuples.

In general, a trajectory optimization problem works in continuous-time domain and can be defined as:

$$\min_{t_0, t_F, x(t), u(t)} \underbrace{\int_{t_0}^{t_F} \rho\left(x(t), u(t), f(t, x(t), u(t))\right) dt}_{Cost-to-go\ or\ Lagrange\ Term} + \underbrace{\mathcal{M}(t_0, x(t_0), t_F, x(t_F))}_{Terminal\,cost,\,Mayer\,Term} \quad \text{s. t.}$$

$$\begin{aligned}
&\dot{x}(t) = f(t, x(t), u(t)) \quad \text{dynamics enforcing constraint}\\
&x_{low} \le x(t) \le x_{upp} \quad \text{state boundaries constraint}\\
&u_{low} \le u(t) \le u_{upp} \quad \text{action boundaries constraint}\\
&t_{low} \le t_0 \le t_F \le t_{upp} \quad \text{bounds on initial and final time}\\
&x(t_0) = \text{initial state}\\
&x(t_F) = \text{desired state}\\
&+ \text{additional constraints if necessary, e.g. collision avoidance}
\end{aligned} \tag{5.1}$$

where transition dynamics is taking time $t$ as an extra parameter, state $x$ and action $u$ are transformed into time-dependent functions, and Mayer term represents the terminal cost in addition to the standard to RL cost-to-go. Due to the continuous-time formulation, it is possible to solve this task with a variable horizon. There exist many methods to solve the above task, and their description lies outside the scope of the thesis. However, great overviews can be found in, e.g., [100, 101, 102].

For the relevant RL setting, which uses discrete time steps, the above formulation requires several adaptations. First of all, the variable horizon is to be placed outside the optimization problem due to the lack of continuous-time dynamics. Second of all, Mayer Term is not used in the RL formulation from the thesis. And the last modification consists of fixing $t_0$ to zero. Overall, relevant to RL formulation is given below:

$$
\begin{aligned}
\min_{N\in\mathbb{Z}^+} J \quad & \text{s. t.} \\
& \min_{[x_0,\dots,x_N],[u_0,\dots,u_N]} J = \sum_{k=0}^{N} \rho(x_k, u_k, f(x_k, u_k)) \quad \text{s. t.} \\
& x_{k+1} = f(x_k, u_k) \\
& x_{low} \leq x_k \leq x_{upp} \\
& u_{low} \leq u_k \leq u_{upp} \\
& x_0 = \text{initial state} \\
& x_N = \text{desired state} \\
& + \text{ additional constraints if necessary}
\end{aligned}
\tag{5.2}
$$

Most of the methods for solving continuous-time versions of trajectory optimization are able to handle inner optimization as well. However, outer optimization is responsible for the optimal number of discrete time steps and is tricky to optimize by conventional methods. One of the ways to perform such optimization is using binary search for $N$, solving the corresponding inner optimization several times.

### 5.2.2 Proof of concept

The bi-level optimization approach was tested on the pendulum swing-up benchmark (see Appendix A.1 for mathematical details) and compared with a closed-loop solution by means of fuzzy approximation. List of parameters for building fuzzy approximation is presented in Table 5.1.

| **Fuzzy V-iteration parameters** | |
|---|---|
| State space, $\mathcal{X}$ | $[-\pi, \pi] \times [-30, 30]$ |
| Input space, $\mathcal{U}$ | $[-2, 2]$ |
| Action samples per dimension, $B_U$ | 11 |
| Discount factor, $\gamma$ | 0.95 |
| Convergence threshold, $\epsilon$ | $10^{-4}$ |
| Desired state, $x_{des}$ | $[0, 0]^T$ |
| Sampling period, $T_s$ [s] | 0.02 |
| Simulation time, $T_{sim}$ [s] | 3 |

**Table 5.1:** Pendulum swing-up experiment parameters

Trajectories are then obtained from a learned fuzzy approximator by the baseline policy derivation algorithm, using 30 different initial points. The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.02$ s. The control goal is to stabilize the pendulum in the unstable equilibrium $x_{des} = [x_{1_{des}}, x_{2_{des}}] = [0, 0]$ (rad, rad/s), which is expressed by the following reward function:

$$\begin{aligned} P(a) &= \sqrt{1+a^2} - 1 \quad \text{Pseudo-Huber loss} \\ \rho(x, u, f(x, u)) &= P(\cos x_1 - 1)q_1 + P(x_2)q_2 + P(u)q_3 \end{aligned} \tag{5.3}$$

where $[q_1, q_2, q_3] = [100, 1, 0.1]$ are weights to adjust the relative importance of the angle, angular velocity and control current, respectively.

For inner trajectory optimization, the package iLQG [103, 103] with default parameters was employed. The testing procedure consists of deriving optimal trajectories from 30 randomly selected initial points using two presented methods. An example of trajectories as well as the results of testing are presented in Figure 5.2.

It can be seen that trajectory optimization achieves competitive results w.r.t. the fuzzy approximation. The results are slightly in favor of trajectory optimization. The main reason for that is that trajectory optimization works directly with continuous action spaces. On the other hand, both the fuzzy approximator and baseline policy derivation use a discrete set of actions to operate. However, there are outliers demonstrating results in favor of the fuzzy approximator. Since trajectory optimization is based on non-linear optimization, it is prone to stacking in local minima. This problem is solved in practice by running multiple optimizations using different initial vectors, which is not implemented in vanilla iLQG.

## 5.3 Robust symbolic policy derivation

### 5.3.1 Proposed method

Policy derivation can be understood as a hill-climbing process: at each time step, the agent searches for the control input that leads to a state with the highest value given by the right-hand side (RHS) of the Bellman equation. An advantage of this control law is its inherent stability – the value function is analogous to the control Lyapunov function [58, 59]. As analyzed in Section 4.3, for each step of policy derivation on the optimal V-function, there should always exist a reachable state with a higher value (except for the goal state). However, symbolically approximated V-functions introduce merely inevitable approximation error, which may damage this property. It forces policy derivation to choose a state with a lower value than the current one. Additionally, symbolically approximated V-function often contains local maxima, which may lead to catastrophic failures, even if the general shape of the V-function is satisfactory. As demonstrated in Figure 5.3, it is possible to find a better trajectory by extending the policy derivation horizon from one-step-ahead to N-step-ahead.

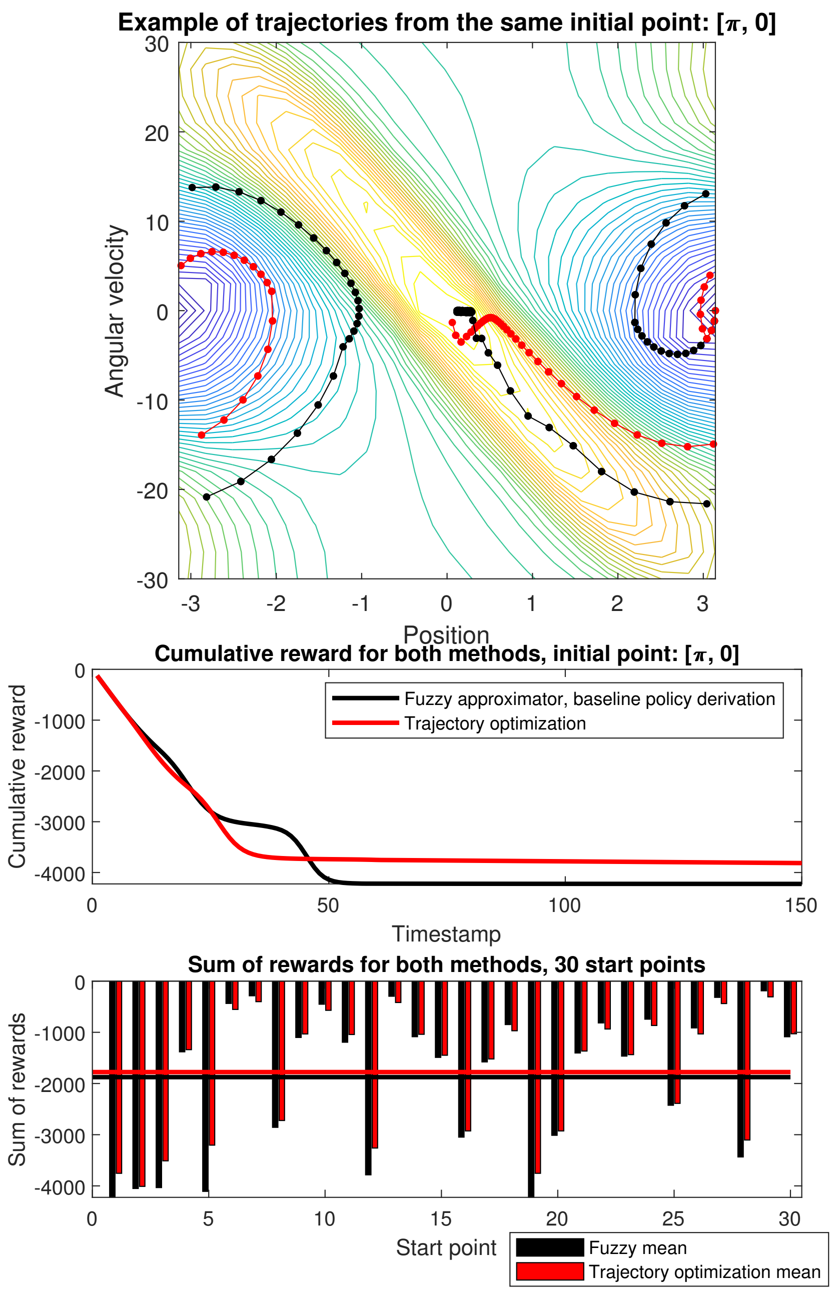

**Figure 5.2:** Comparison of trajectories, obtained by different methods: fuzzy approximator vs. trajectory optimization on pendulum swing-up benchmark (see Appendix A.1 for mathematical details). Top plot depicts trajectories starting from $[\pi, 0]$ point, superimposed on a learned fuzzy approximation of the V-function. Middle plot illustrates the evolution of cumulative rewards for both methods. Bottom plot depicts the sum of rewards obtained by both methods for 30 randomly chosen initial points, as well as their mean sum of rewards.

The key idea of the proposed method consists of modifying the policy derivation algorithm with an extended variable horizon. This work blends together ideas of model predictive control (MPC) [104, 105, 106] and *QSPD* algorithm (see Section 2.2.5). MPC can be briefly described as follows: it seeks for the trajectory of a given finite horizon, optimizing the sum of rewards, then executes the first found action and repeats the whole process. *QSPD* provides the basement for efficient and scalable policy derivation in continuous spaces.

The optimization task involves finding a sequence of $N$ actions, which results in the highest possible value at the last step. Unlike MPC, values along the path are ignored for the sake of avoiding local extrema.

The task is formalized as follows:

$$\max_{\mathbf{u}=[u_1,\ldots,u_H]} R^s(x,\mathbf{u}) = \left[\rho\left(\hat{f}(x,u_{H-1}),u_H,\hat{f}(x,u_H)\right) + \gamma V(\hat{f}(x,u_H))\right] \tag{5.4}$$

where $\hat{f}(\cdot)$ is defined as:

$$\hat{f}(x,u_H) = \begin{cases} \hat{f}(x,u_0) = x \\ \hat{f}(x,u_1) = f(x,u_1) \\ \hat{f}(x,u_H) = f(\hat{f}(x,u_{H-1}),u_H) \end{cases} \tag{5.5}$$

The aforementioned optimization occurs at each step of policy derivation, but only action $u_1$ is applied. For $H = 1$ this optimization is reduced to *QSPD*, which solves it using the trust region reflective (TRR) algorithm. This method requires symbolic partial derivatives for each dimension of the action space to work efficiently. Therefore, $\rho(\cdot)$ and $f(\cdot)$ should be symbolic representations of the reward and state transition function, respectively, while the V-function is given a priori in a symbolic form. In model-based RL, the experimenter virtually always designs the reward function and can easily define it as an analytic one. However, this is often not the case with the state transition function $f(\cdot)$. The system dynamics is typically described in continuous time, while the state transitions are generated through numerical integration, using, e.g., Runge-Kutta methods. The superscript $s$ is used in the sequel to distinguish symbolic functions and operators from their numeric counterparts.

For extension of the *QSPD* algorithm to a longer horizon, partial derivatives are required for all actions in the sequence and all dimensions of the action space. In this work, Matlab Symbolic Math Toolbox [107] was successfully employed for computing them analytically. Actual values of the partial derivatives are then computed by the chain rule when needed for the optimization algorithm.

This optimization technique implicitly assumes that each found value would be higher than the previous one. However, it is not always the case. Consider two consecutive policy derivation steps, at points $x_k$ and $x_{k+1}$, respectively. Solving (5.4) for the first point would result in a sequence of actions $[u_1^1, u_2^1, \ldots, u_H^1]$ with final value $V^1$, from which action $u_1^1$ is applied to

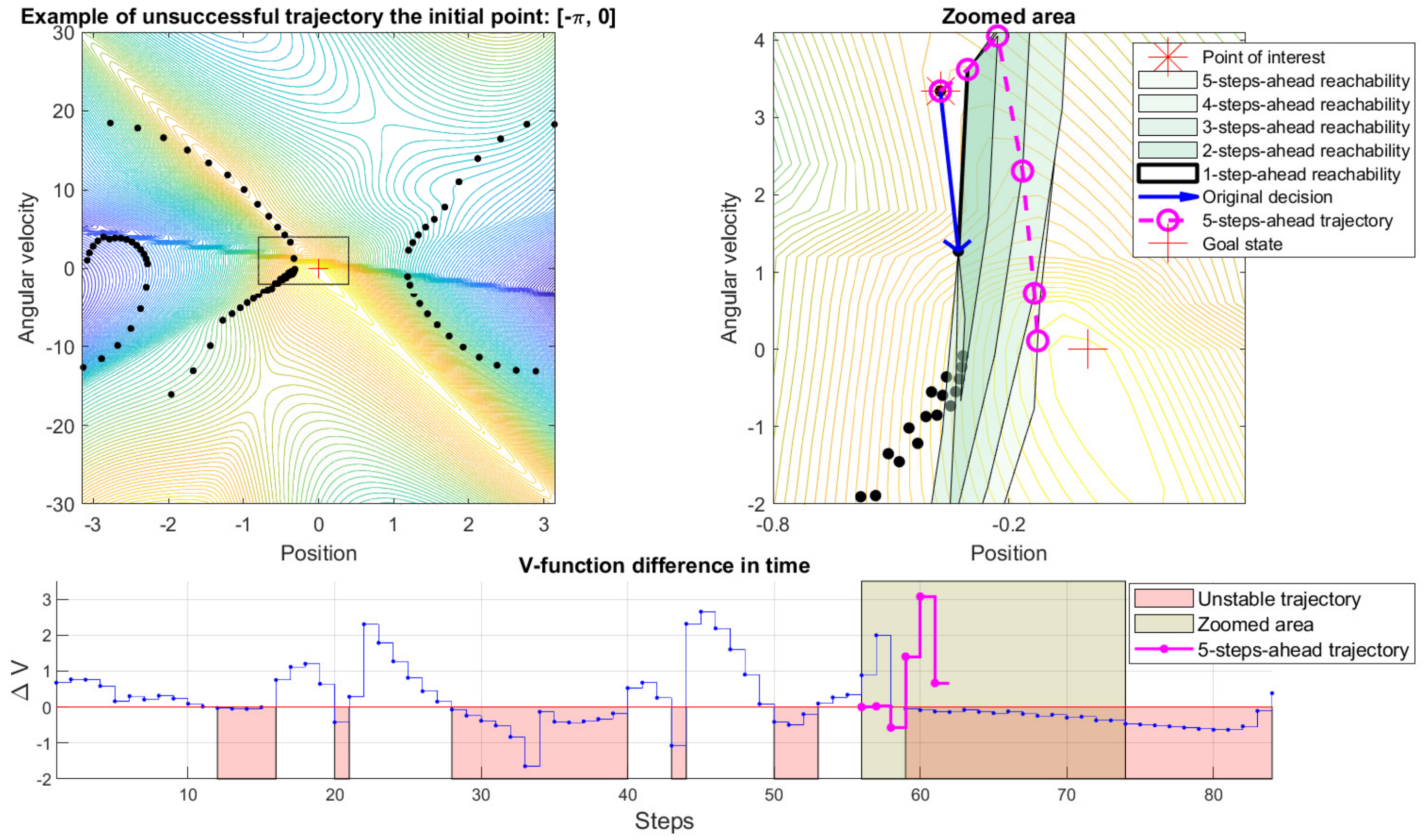

**Figure 5.3:** Analysis of single trajectory on pendulum swing-up benchmark (see Appendix A.1 for mathematical details) using one of the unsuccessful symbolic V-functions. The top-left plot demonstrates a single trajectory starting from $[-\pi, 0]$ superimposed on one of the symbolic V-functions. The top-right plot depicts a zoomed region of interest, where a single decision from an asterisk point is considered. The blue line shows the original decision obtained by the baseline policy derivation method. Purple line demonstrates better (in terms of the final result) trajectory, obtained by gradually increasing horizon and performing Monte Carlo simulation. The bottom plot shows value difference in time with highlighted streaks of forced decision to choose a state with a lower value than currently achieved.

get to the point $x_{k+1}$. Solving the same optimization problem for $x_{k+1}$ would result in another sequence $[u_1^2, u_2^2, \ldots, u_H^2]$ with value $V^2$. What action should be preferred in case $V^2 < V^1$: $u_2^1$ or $u_1^2$? Following the hill-climbing principle, the best-found sequence of actions should be cached until the sequence with a better value is found. Only after that, policy derivation should use a new sequence.

When all ingredients are prepared, the Robust Symbolic Policy Derivation (*RSPD*) can be summarized in Algorithm 7.

**Algorithm 7:** Robust Symbolic Policy Derivation (*RSPD*)

```
Input: $f, \hat{f}^s, \rho^s, \gamma, R^s, H, x_0$
$k \leftarrow 0$
$\mathbf{u_{cache}} \leftarrow [u_1, \ldots, u_H]$
$V_{cache} \leftarrow -\infty$
$z \leftarrow 0$
while control experiment not terminated do
    $\mathbf{u} \leftarrow$ initialize with random actions within $\mathcal{U}^H$ domain
    $\mathbf{u_{cand}} \leftarrow \underset{\mathbf{u} \in \mathcal{U}^H}{\text{argmax}}\ R^s(x_k, \mathbf{u})$; TRR optimization
    $V_{cand} \leftarrow R^s(x_k, \mathbf{u_{cand}})$
    if $(V_{cand} \geq V_{cache})$ or $(z = H)$ then
        $V_{cache} \leftarrow V_{cand}$
        $\mathbf{u_{cache}} \leftarrow \mathbf{u_{cand}}$
        $z \leftarrow 0$
    $z \leftarrow z + 1$
    $k \leftarrow k + 1$
    $\dot{u}_k \leftarrow u_z$
    $x_{k+1} \leftarrow f(x_k, \dot{u}_k)$
end
Output: trajectory $[x_1, x_2, \ldots]$, $[\dot{u}_1, \dot{u}_2, \ldots]$
```

## 5.3.2 Proof of concept

The proposed concept is tested on two benchmarks: pendulum swing-up and magnetic manipulation with three coils.

Pendulum swing-up benchmark used parameters from Section 4.5.1, except for the reward function, which is expressed as follows: The control goal is to stabilize the pendulum in the unstable equilibrium $x_{des} = [x1_{des}, x2_{des}] = [0, 0]$ (rad, rad/s), which is expressed by the following reward function:

$$\begin{aligned} P(a) &= \sqrt{1 + a^2} - 1 \quad \text{Pseudo-Huber loss} \\ \rho(x, u, f(x, u)) &= P(\cos x_1 - 1) q_1 + P(x_2) q_2 + P(u) q_3 \end{aligned} \tag{5.6}$$

where $[q_1, q_2, q_3] = [100, 1, 0.1]$ are weights to adjust the relative importance of the angle, angular velocity, and control current, respectively. Symbolic model of the V-function is drawn manually from the pool of suboptimal models, produced by Mode**ABC** fitting procedure (refer to Section 4.4 for further details). Trajectories are obtained by setting horizon $H$ to $5, 10, 20, 40$, respectively. These trajectories are then compared with the baseline policy derivation introduced earlier. The results are depicted in Figure 5.4.

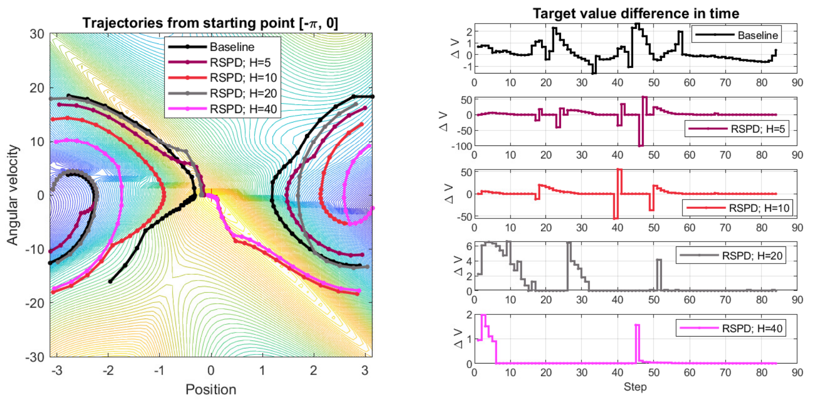

**Figure 5.4:** Comparison of trajectories, obtained by using *RSPD* algorithm with different horizons $H$. Left plot demonstrates a set of trajectories starting from $[-\pi, 0]$ point superimposed on one of the suboptimal symbolic V-functions. Plots on the right depict the expected value difference in time.

It can be seen in Figure 5.4-left, that *RSPD* is able to improve policy derivation dramatically - all trajectories (except the one derived by baseline method) reached the vicinity of the goal state despite suboptimal approximation of the V-function. Furthermore, with sufficiently long horizon $H$ (namely, for $H = 20$ and $H = 40$), *RSPD* successfully produced never decreasing sequences of expected values. However, insufficiently long horizons may still violate the hill-climbing principle, but the length of decreasing streaks is decreased to single steps.

Magnetic manipulation benchmark used the same parameters as were in Section 4.5.2, except for the following:

- Goal state is set to $x_{des} = [x_{des_1}, x_{des_2}] = [0.0150]$
- Simulation time is reduced to $1.5(s)$
- The reward function is defined as:

$$\begin{aligned} P(a) &= \sqrt{1 + a^2} - 1 \quad \text{Pseudo-Huber loss} \\ \rho(x, u, f(x, u)) &= P(x_1 - x_{des_1})q_1 + P(x_2 - x_{des_2})q_2 \end{aligned} \tag{5.7}$$

  where $[q_1, q_2] = [1350, 150]$ are weights to adjust the relative importance of the position and velocity, respectively.

Symbolic model of the V-function is produced by using Mode**ABC** fitting procedure (see Section 4.4 for further details). Trajectories are obtained by setting horizon $H$ to $2, 5, 10, 20$, respectively. These trajectories are again compared with the one derived by the baseline algorithm. The results are depicted in Figure 5.5.

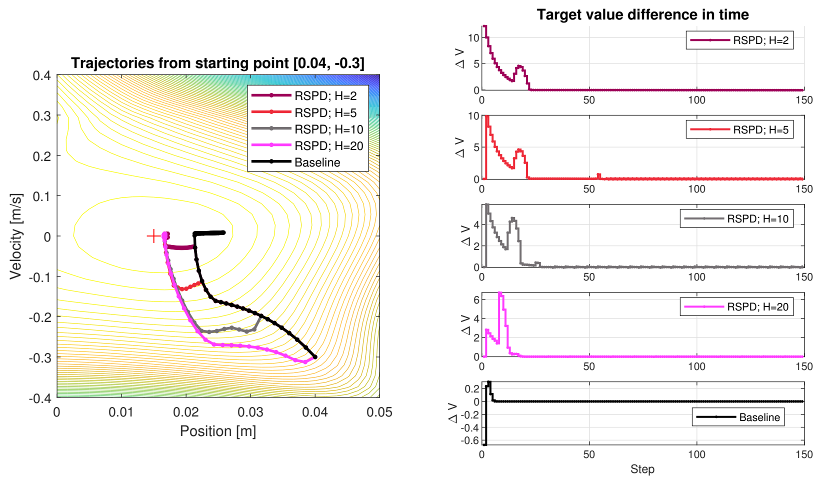

**Figure 5.5:** Comparison of trajectories, obtained by using *RSPD* algorithm with different horizons $H$. Left plot demonstrates a set of trajectories starting from $[0.04, -0.3]$ point superimposed on one of the suboptimal symbolic V-functions. Plots on the right depict expected value difference in time.

The results again support the hypothesis that *RSPD* is able to improve policy derivation dramatically, all trajectories except for the baseline ended up in close vicinity of the goal state. Figure 5.5-left demonstrates how different horizons affect the optimality of the trajectory.

## 5.3.3 Discussion

*RSPD* algorithm demonstrates the ability to recover policy derivation from suboptimal symbolic approximations of the V-function. It relies on a specific feature of the SR - the good grasp of the general shape. However, several drawbacks need to be addressed:

- Computational time - due to its nature, *RSPD* is substantially slower than other policy derivation methods. Since it is based on the *QSPD* algorithm, it scales well as the dimensionality of the action space grows. However, in the case of *RSPD*, action space dimensionality is multiplied by the horizon length. It is beneficial to select the horizon as low as possible considering the quality of the V-function approximation.

- Non-linear optimization - the core of *RSPD* is based on the non-linear optimization, which is prone to fall in local minima. The caching mechanism helps to mitigate this effect. Different strategies for avoiding local minima may be a part of future work.

- Suboptimal policy - *RSPD* algorithm considers only the last state in the sequence of actions, effectively ignoring values along the found trajectory. While it is highly beneficial for the stability of the solution, it damages the optimality at the same time. For that reason, it is desirable to optimize the horizon length. In other words, in the case of near-ideal approximation of the V-function, *RSPD* is not needed and should be degraded to *QSPD*.

*RSPD* algorithm ensures that policy derivation always goes to a higher state, assuming a long enough horizon to mitigate imperfections of a V-function. However, it may be counter-productive in the case when some portion of the V-function ended up being higher than the goal state. It may happen either due to the approximation error or by means of underfitting. Thus, the goal state (or states) requires additional attention. For this work, the Mode**ABC** fitting procedure explicitly anchors the goal state above the whole state space. Additionally, the selection of the optimal horizon is still an open question, which will be addressed in the consequent section.

## 5.4 Efficient symbolic RL framework

Now, all ingredients for building an efficient SR for RL method are prepared to be wired together into a framework. Conceptually, it consists of the following components, briefly recapitulated here for better readability.

- Trajectory optimization as a sampling technique (introduced in Section 5.2) - solves the task of generating optimal trajectories without knowing the V-function beforehand. Running it multiple times generates a required amount of samples for fitting symbolic approximation of the V-function.
- Mode**ABC** fitting method (introduced in Section 4.4)- using non-conventional error function and set of specific constraints, guides an evolution process to the direction, desirable for RL. As a distinctive feature, Mode**ABC** explicitly bounds the goal state or states to be the highest points on the surface of the V-function. This fitting method relies on the data in the form of tuples $(x_k, x^*_{k+1}, \rho(x_k, u^*, x^*_{k+1})$, where $x^*_{k+1}$ and $u^*$ are the next optimal state and optimal action for the point $x_k$, respectively. Data in that specific format are collected from trajectory optimization sampling.
- *RSPD* policy derivation method (introduced in Section 5.3) - ensures that policy derivation is not affected by local extrema. It is done by extending the horizon for which policy derivation predicts future values in combination with controlled geometry of the V-function (ensured by Mode**ABC**). *RSPD* algorithm guarantees the reachability of the goal state, assuming a sufficiently long horizon. However, it does not guarantee the optimality of the trajectory.

This chapter is devoted to the identification and filling of missing gaps between these components. The resulting algorithm is then composed in a way that satisfies all the desired properties, defined in Section 5.1. Consequently, the algorithm is tested on pendulum swing-up and magnetic manipulation benchmarks. Additionally, it is tested on a new benchmark with higher dimensionality - drone strike.

### 5.4.1 Proposed method

The first gap to be solved comes directly from the desired properties defined in Section 5.1 - incorporating region importance into the fitting process. The importance of particular areas of the state space is illustrated in Section 4.1 and can be briefly described as follows: there may exist critical areas only through which the goal state is reachable. If the approximation error affects these areas, the whole approximator may be rendered useless since policy derivation may never choose a path through these areas. For that reason, fitting such areas should be prioritized, even at the cost of worsening approximation errors in other regions. Luckily, using trajectory optimization as a sampling technique naturally fills this gap. If such critical areas exist for the task, all trajectories that reach the goal state will pass through them. Consequently, the number of samples in critical areas is increased compared to the rest of the state space. Increased density of samples in critical areas leads to the increased importance of these regions from the fitness function point of view. This effect is depicted in Figure 5.6.

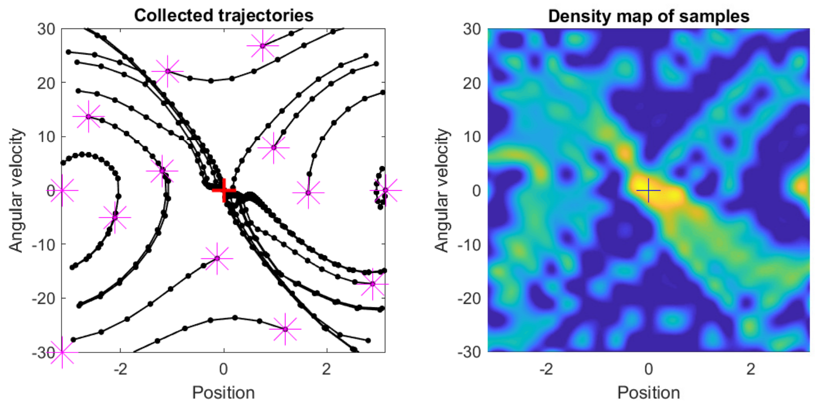

**Figure 5.6:** Illustration of the dataset, collected by trajectory optimization. The left plot demonstrates a set of trajectories starting from quasi-randomly selected initial points, marked by asterisks. The right plot depicts the density map of collected samples.

The second open gap is the selection of the optimal horizon $H$ for the *RSPD* algorithm. Ideally, the horizon should possess the following qualities:

1. It is sufficiently long to ensure the reachability of the goal state.
2. It is as short as possible to minimize computational expenses.
3. The length is not constant and varies for different approximators.
4. Horizon length should be incorporated into fitness function.

One option to find a suitable horizon relies on exhaustive simulations for each candidate model. While this method should work, it is computationally expensive and, thus, impractical. Alternatively, the desired horizon may be found by the following procedure:

1. Collect additional trajectories by trajectory optimization and form test dataset.
2. Start evolution of the symbolic approximation.
3. For each candidate model of the V-function initialize the length of horizon $H$ with the length of the longest trajectory:
   a. For each trajectory from test dataset
      (i) Match trajectory against candidate model.
      (ii) Compute difference of the value, analogously to Figure 5.3-bottom.
      (iii) Identify the length of the longest streak of decreasing values (marked red on the plot) and store it in $H_{cand}$.
      (iv) Set $H = H_{cand}$ if $H_{cand} < H$.
   b. Use $H$ in computation of fitness.
4. Return $H$ as the desired horizon for *RPDS* for this particular model.

There are multiple options how to use horizon length within the fitness function. In this work, the multiplicative variant is selected, namely:

$$fitness = H * F_{\mathbf{ABC}} \tag{5.8}$$

where $F_{\mathbf{ABC}}$ stands for the fitness, obtained by using Mode**ABC**. By using multiplicative variant of the fitness function in combination with Mode**ABC**, candidate models form the following hierarchy of preferences, from highest to lowest:

- Reduce required horizon
- Reduce error on samples that violate stability
- Reduce error on samples that violate optimality

The third gap is partially created by switching to trajectory optimization sampling from equidistant sampling, used in original Mode**ABC** fitting procedure. The purpose of Condition **B** is to guarantee that the resulting approximation will have the maximum at zero and all other samples will be lower than the maximum value. The equidistant sampling is required to cover the whole state space to work reliably. However, since trajectory optimization does not guarantee a homogeneous distribution of the samples, condition **B** should use another dataset. Additionally, equidistant sampling may become prohibitive as the dimensionality of a task grows. One possible way to solve this issue is to sample $K$ support points quasi-randomly specifically for Condition **B**.

Now, Efficient SR for RL (*ESRL*) framework can be composed into the block-scheme, presented in Figure 5.7.

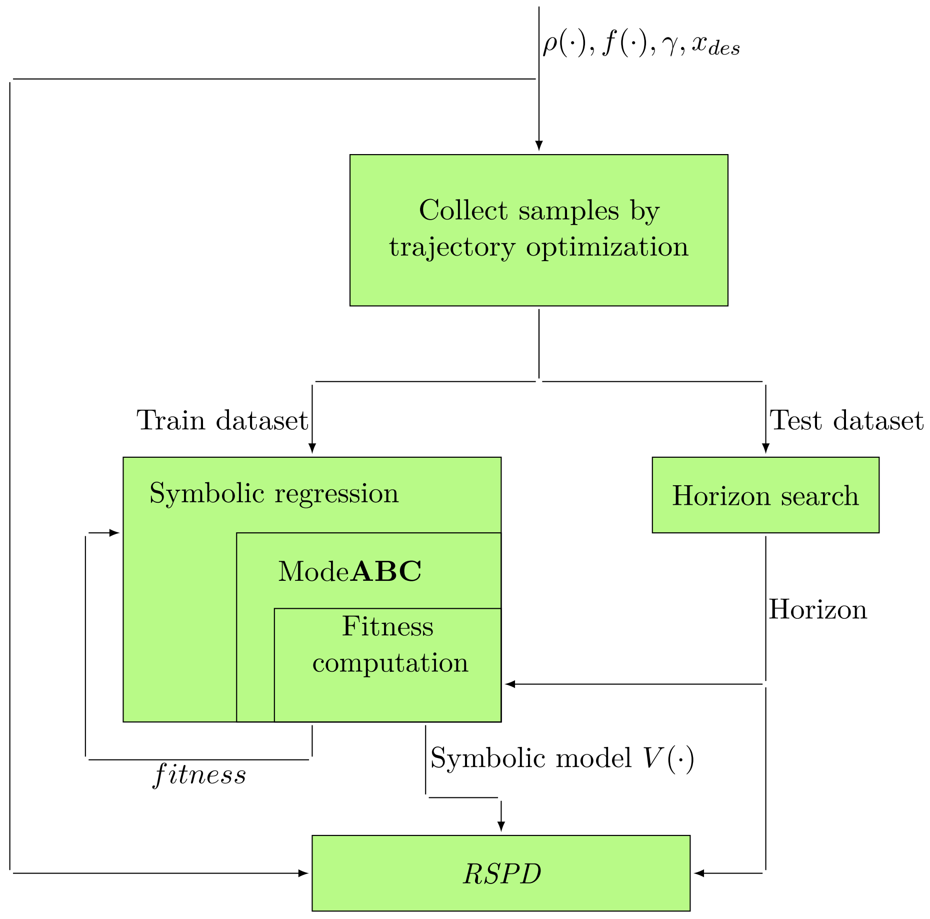

**Figure 5.7:** Block-scheme of the Efficient SR for RL (*ESRL*) framework.

### 5.4.2 Experimental evaluation

The *ESRL* framework has been tested on three benchmarks. The first two are well-known pendulum swing-up and magnetic manipulation (abbreviated as magman). The third one is a drone strike, which is explicitly designed to test the scalability aspects of the proposed solution. The framework is run 15 times for each benchmark, resulting in 15 different symbolic models. Symbolic models are built by the modified version of SNGP (see Appendix B.3 for further details). All models within a benchmark are constructed using identical parameters (except the random seed) to exclude any influence of different parameters from the comparison. Each model is then tested through simulations with $K = 50$ randomly chosen initial states, which are the same for each benchmark.

It should be explicitly noted that all the models $V^C(\cdot)$ are presented "as is", which means that there is no selection procedure w.r.t. some criterion.

To measure the performance of a single trajectory, the following criteria are defined:

- Euclidean distance to the goal state denoted as $D_t$. For pendulum swing-up, the angle is used for computation. For both magman and drone strike, the position is used.

- Return, collected via obtained trajectory, defined as:

$$R_t = \sum_{q=1}^{T_{sim}/T_s} \rho(x_q, u_q, f(x_q, u_q))$$

where $T_{sim}$ and $T_s$ are the total simulation time and the sampling period, respectively.

- Optimality level, computed by applying trajectory optimization on the same initial state. Return of the optimized trajectory is defined as $R_t^o$ and computed in the same manner as $R_t$. Optimality level is then defined as:

$$O_t = \frac{R_t^o}{R_t} \cdot 100\%$$

- Binary flag of the success of the simulation, denoted as $S_t$. Simulation is considered successful if the last 25 steps of the trajectory lie within 5% interval around the goal state $x_{des}$. An example of the successful trajectory and appropriate interval is given in Figure 4.4.

Subscript $t$ stands for "trajectory".

The performance of a single symbolic approximator is measured by the following criteria:

- Mean euclidean distance to the goal state, denoted as $D_m$.
- Percent of successful simulations, denoted as $S_m$.
- Mean trajectory return, denoted as $\dot{R}_m$.
- Median trajectory return, denoted as $\ddot{R}_m$.
- Mean optimality level, denoted as $\dot{O}_m$
- Median optimality level, denoted as $\ddot{O}_m$
- Used horizon $H_m$.

Subscript $m$ stands for "model".

Finally, the benchmark-wise performance is measured using the following criteria:

- Mean euclidean distance to the goal state, denoted as $D$.
- Percent of successful simulations, denoted as $S$.
- Mean return, denoted as $\dot{R}$.
- Mean return, denoted as $\ddot{R}$.
- Mean optimality level, denoted as $\dot{O}$
- Median optimality level, denoted as $\ddot{O}$
- Mean horizon rounded to ceiling value, denoted as $H$.

## Pendulum swing-up

| | |
|---|---|
| *System parameters* | |
| State space, $\mathcal{X}$ | $[-\pi, \pi] \times [-30, 30]$ |
| Input space, $\mathcal{U}$ | $[-2, 2]$ |
| Discount factor, $\gamma$ | 0.95 |
| Desired state, $x_{des}$ | $[0, 0]^T$ |
| Sampling period, $T_s$ [s] | 0.02 |
| Simulation time, $T_{sim}$ [s] | 3 |
| *Trajectory optimization parameters* | |
| Minimal number of train samples, $N^s_{train}$ | 1000 |
| Minimal number of train trajectories, $N^t_{train}$ | 20 |
| Minimal number of test samples, $N^s_{test}$ | 1000 |
| Minimal number of test trajectories, $N^t_{test}$ | 20 |
| *Mode***ABC** *parameters* | |
| Small impact error weight | 100 |
| Medium impact error weight | 10 |
| Big impact error weight | 1 |
| Number of support points, $K$ | 500 |
| Minimal difference with the goal state, $\omega$ | -0.1 |
| *SNGP parameters* | |
| Population size | 1000 |
| Elementary functions | *, +, -, $x^2$, $x^3$, $x^4$, Logistic3 BentGeneral |
| Maximal depth of features | 5 |
| Maximal number of features | 30 |
| Max number of generations | 1500 |
| Local search iterations | 1500 |
| Max number of threads | 10 |
| Number of runs | 15 |

**Table 5.2:** Pendulum swing-up experiment parameters

The inverted pendulum consists of a mass attached to an actuated link that rotates in the vertical plane. The available torque is insufficient to push the pendulum up in a single rotation from the majority of initial states. Instead, from a certain state (e.g., pointing down), the pendulum needs to be swung back and forth to gather energy, before it can be pushed up and stabilized. Full description alongside technical details about the benchmark can be found in Appendix A.1. The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.02\,\text{s}$.

The control goal is to stabilize the pendulum in the unstable equilibrium $x_{des} = [x_{1_{des}}, x_{2_{des}}] = [0, 0]$ (rad, rad/s), which is expressed by the following reward function:

$$\begin{aligned} P(a) &= \sqrt{1 + a^2} - 1 \quad \text{Pseudo-Huber loss} \\ \rho(x, u, f(x, u)) &= P(\cos x_1 - 1)q_1 + P(x_2)q_2 + P(u)q_3 \end{aligned} \tag{5.9}$$

where $[q_1, q_2, q_3] = [100, 1, 0.1]$ are weights to adjust the relative importance of the angle, angular velocity and control current, respectively.

The comprehensive set of experiment parameters is presented in Table 5.2 and the results of evaluation are presented in Table 5.3.

| | $D_m$, [rad] | $S_m$, [%] | $\dot{R}_m$ | $\ddot{R}_m$ | $\dot{O}_m$, [%] | $\ddot{O}_m$, [%] | $H_m$ |
|---|---|---|---|---|---|---|---|
| Model 1 | 0.002 | 100 | -1061.728 | -979.206 | 99.95 | 97.63 | 17 |
| Model 2 | 0.001 | 100 | -1067.065 | -991.437 | 96.57 | 96.56 | 17 |
| Model 3 | 0.037 | 100 | -1141.668 | -1002.267 | 92.99 | 95.39 | 18 |
| Model 4 | 0.003 | 100 | -1048.297 | -1000.167 | 97.26 | 97.35 | 18 |
| Model 5 | 0.002 | 100 | -1062.029 | -988.43 | 98.34 | 98.60 | 17 |
| Model 6 | 0.001 | 100 | -1120.773 | -1011.652 | 89.85 | 96.04 | 18 |
| Model 7 | 0.002 | 100 | -1237.914 | -1032.698 | 86.91 | 93.31 | 18 |
| Model 8 | 0.002 | 100 | -1066.323 | -992.224 | 96.93 | 96.99 | 18 |
| Model 9 | 0.002 | 100 | -1055.86 | -988.126 | 98.84 | 97.04 | 17 |
| Model 10 | 0.002 | 100 | -1049.406 | -991.552 | 99.25 | 98.47 | 17 |
| Model 11 | 0.002 | 100 | -940.314 | -922.988 | 109.14 | 102.19 | 17 |
| Model 12 | 0.001 | 100 | -1203.769 | -1014.179 | 91.11 | 93.10 | 17 |
| Model 13 | 0.037 | 100 | -1052.471 | -991.686 | 99.46 | 97.82 | 18 |
| Model 14 | 0.003 | 100 | -1220.83 | -1050.374 | 84.70 | 92.97 | 18 |
| Model 15 | 0.002 | 100 | -1101.34 | -1012.204 | 94.70 | 96.17 | 17 |
| Benchmark | $D$, [rad] | $S$, [%] | $\dot{R}$ | $\ddot{R}$ | $\dot{O}$, [%] | $\ddot{O}$, [%] | H |
| | 0.006 | 100 | -1095.319 | -992.224 | 95.73 | 96.99 | 18 |

**Table 5.3:** Pendulum swing-up experiment results

At first sight, it can be seen that computed V-functions are underdeveloped, which is indicated by horizon lengths $H_m$. Despite that fact, all simulations achieved the goal state. Moreover, recalling the definition of success $S_t$, all models successfully stabilized themselves at the goal state. Optimality level is indicated to be high, despite the underdevelopment of V-functions. These results support the hypothesis that *ESRL* framework is able to perform even on poorly fitted V-functions. Additionally, it correctly guesses the required horizon beforehand. The underdevelopment of symbolic models can be potentially overcome by increasing the maximal depth of features or and by increasing the number of epochs. However, it is possible that with the current set of settings, the target shape of the V-function cannot be approximated accurately. Research on the theoretical flexibility of a given set of features may be a part of future work.

### Magnetic manipulation

Magnetic manipulation (abbreviated as magman) is a challenging non-linear control problem. The current through the electromagnets is controlled to dynamically shape the magnetic field above the magnets and so to accurately and quickly position a steel ball to the desired set point. The first three coils at 0, 0.025, and 0.05 (m), respectively, have been used for the experiments presented in this work. Full description alongside technical details about the benchmark can be found in Appendix A.3.

The goal of the benchmark is to move the steel ball to the position $x_{des} = [0.035, 0]$ (m, m/s) and stabilize it. Coordinated work of two coils is required

| *System parameters* | |
|---|---|
| State space, $\mathcal{X}$ | $[0, 0.05] \times [-0.4, 0.4]$ |
| Input space, $\mathcal{U}$ | $[0, 0.6] \times [0, 0.6] \times [0, 0.6]$ |
| Discount factor, $\gamma$ | 0.995 |
| Desired state, $x_{des}$ | $[0.035, 0]$ |
| Sampling period, $T_s$ [s] | 0.005 |
| Simulation time, $T_{sim}$ [s] | 1 |
| *Trajectory optimization parameters* | |
| Minimal number of train samples, $N^s_{train}$ | 500 |
| Minimal number of train trajectories, $N^t_{train}$ | 20 |
| Minimal number of test samples, $N^s_{test}$ | 500 |
| Minimal number of test trajectories, $N^t_{test}$ | 20 |
| *Mode***ABC** *parameters* | |
| Small impact error weight | 100 |
| Medium impact error weight | 10 |
| Big impact error weight | 1 |
| Number of support points, $K$ | 300 |
| Minimal difference with the goal state, $\omega$ | -0.1 |
| *SNGP parameters* | |
| Population size | 1000 |
| Elementary functions | *, +, -, $x^2$, $x^3$, $x^4$, Logistic3, BentIdentityGeneral |
| Maximal depth of features | 5 |
| Maximal number of features | 30 |
| Max number of generations | 1500 |
| Local search iterations | 1500 |
| Max number of threads | 10 |
| Number of runs | 15 |

**Table 5.4:** Magnetic manipulation experiment parameters

to achieve that task since the desired position is placed midway between the second and the third coil. The reward function is defined as:

$$\begin{aligned} P(a) &= \sqrt{1+a^2} - 1 \quad \text{Pseudo-Huber loss} \\ \rho(x, u, f(x, u)) &= 10^4 \cdot [P(x_1 - x_{des_1})q_1 + P(x_2 - x_{des_2})q_2] \end{aligned} \tag{5.10}$$

where $[q_1, q_2] = [130, 1]$ are weights to adjust the relative importance of the position and velocity, respectively. The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.005\,\text{s}$.

The comprehensive set of experiment parameters is presented in Table 5.4 and the results of evaluation are presented in Table 5.5.

Results of the magnetic manipulation benchmark evaluation indicate the same pattern as results on the pendulum swing-up benchmark. All simulations for all models successfully reached the goal state with minimal steady-state error. The optimality level is oscillating around 100%, indicating a near-optimal solution. Additionally, results demonstrate a significant difference between mean and median returns. A few simulations with extremely high costs are responsible for it. Since reward function penalizes being distant

| | $D_m$, [m] | $S_m$, [%] | $\dot{R}_m$ | $\ddot{R}_m$ | $\dot{O}_m$, [%] | $\ddot{O}_m$, [%] | $H_m$ |
|---|---|---|---|---|---|---|---|
| Model 1 | 0.00025 | 100 | -10491.648 | -6115.424 | 101.75 | 100.14 | 27 |
| Model 2 | 0.00029 | 100 | -10436.98 | -6080.711 | 102.38 | 100.49 | 26 |
| Model 3 | 0.00041 | 100 | -10396.521 | -5971.884 | 102.85 | 100.76 | 25 |
| Model 4 | 0.00049 | 100 | -10346.883 | -5874.124 | 103.28 | 101.49 | 24 |
| Model 5 | 0.00051 | 100 | -10262.946 | -5769.442 | 104.24 | 102.33 | 22 |
| Model 6 | 0.00025 | 100 | -10391.828 | -5960.179 | 102.97 | 100.95 | 25 |
| Model 7 | 0.00029 | 100 | -10514.302 | -6206.78 | 101.65 | 100.16 | 27 |
| Model 8 | 0.00041 | 100 | -10385.331 | -6019.408 | 102.97 | 101.13 | 25 |
| Model 9 | 0.00049 | 100 | -10579.787 | -6468.588 | 100.99 | 100.00 | 28 |
| Model 10 | 0.00051 | 100 | -10514.693 | -6137.47 | 101.72 | 100.09 | 27 |
| Model 11 | 0.00031 | 100 | -10540.729 | -6242.944 | 101.30 | 100.00 | 27 |
| Model 12 | 0.00034 | 100 | -10428.79 | -6052.878 | 102.48 | 100.53 | 26 |
| Model 13 | 0.00033 | 100 | -10487.768 | -6104.459 | 101.92 | 100.27 | 27 |
| Model 14 | 0.00024 | 100 | -10439.034 | -6036.734 | 102.30 | 100.49 | 26 |
| Model 15 | 0.00025 | 100 | -10402.308 | -6022.305 | 102.72 | 100.84 | 25 |
| Benchmark | $D$, [m] | $S$, [%] | $\dot{R}$ | $\ddot{R}$ | $\dot{O}$, [%] | $\ddot{O}$, [%] | $H$ |
| | 0.00035 | 100 | -10441.303 | -6052.878 | 102.37 | 100.64 | 26 |

**Table 5.5:** Magnetic manipulation experiment results

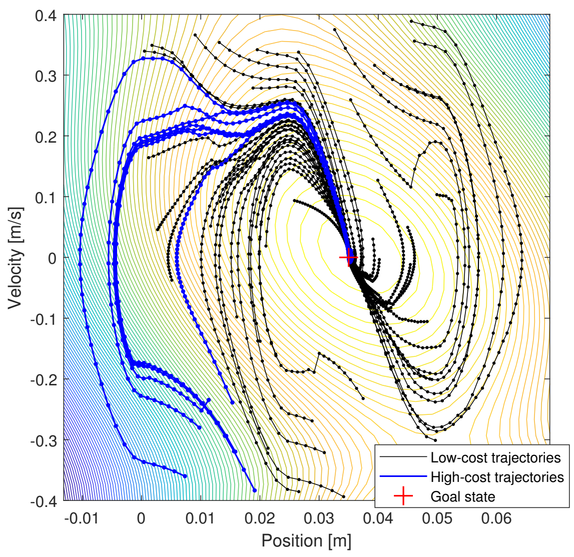

**Figure 5.8:** Set of magnetic manipulation trajectories superimposed on found symbolic V-function, obtained by *ESRL* framework. Trajectories are split into two subsets (marked black and blue) with approximately equal sums of their returns $R_t$.

from the goal state, all simulations with distant starting points inevitably produce costly trajectories. This effect is depicted in Figure 5.8. Here, the set of all trajectories is split into two subsets with approximately equal sums of their returns $R_t$. It can be seen that these subsets are heavily imbalanced, which resulted in the difference between $\dot{R}_m$ and $\ddot{R}_m$.

## Drone strike

Drone strike is an extremely challenging problem to tackle on a standard PC. Due to the highly non-linear 12-D dynamics described in different frames, this benchmark has been designed specifically to test the scalability aspects of the proposed solution. In addition, the action space consists of four dimensions which are required to operate simultaneously to control the drone. An initial state represents a violently thrown drone. The main goal of the benchmark is to stabilize it and then hit the desired point $x_{des}^{pos} = [1, 1, 0]$ (m) in a kamikaze way. Additionally, it is desirable to align Euler angles (roll, pitch, yaw) to $x_{des}^{angles} = [0, 0, 0]$ (rad), drop velocity of the drone to zero $x_{des}^{v} = [0, 0, 0]$ (m/s), as well as angular velocity $x_{des}^{angular_v} = [0, 0, 0]$ (rad/s). The drone is controlled through its overall thrust and moments for each Euler axis. Desired controls are limited to $u_{des} = [0, 0, 0, 0]$ (N, Nm, Nm, Nm), which forces the drone to minimize energy consumption. Only one hit to the ground as allowed, which leaves no room for errors. Full description alongside technical details about the benchmark can be found in Appendix A.4.

The reward function is then defined as follows:

$$\begin{aligned}
P(a) &= \sqrt{1+a^2} - 1 \quad \text{Pseudo-Huber loss} \\
r^{pos} &= 10^3 \cdot P(x_1 - x_{des_1}^{pos}) + 10^3 \cdot P(x_2 - x_{des_2}^{pos}) + 10^3 \cdot P(x_1 - x_{des_3}^{pos}) \\
r^{angles} &= P(x_4 - x_{des_1}^{angles}) + P(x_5 - x_{des_2}^{angles}) + P(x_6 - x_{des_3}^{angles}) \\
r^{v} &= P(x_7 - x_{des_1}^{v}) + P(x_8 - x_{des_2}^{v}) + P(x_9 - x_{des_3}^{v}) \\
r^{angular_v} &= P(x_{10} - x_{des_1}^{angular_v}) + P(x_{11} - x_{des_2}^{angular_v}) + P(x_{12} - x_{des_3}^{angular_v}) \\
r^{u} &= P(u_1 - u_{des_1}) + P(u_2 - u_{des_2}) + P(u_3 - u_{des_3}) + P(u_4 - u_{des_4}) \\
\rho(x, u, f(x,u)) &= r^{pos} + r^{angles} + r^{v} + r^{angular_v} + r^{u}
\end{aligned} \tag{5.11}$$

The discrete-time transitions for the trajectory optimization part are obtained numerically, integrating the continuous-time dynamics using the fourth-order Runge-Kutta. For the policy derivation part, the Euler method was used analogously to *QSPD* algorithm. It is done due to the computational expensiveness of computing partial derivatives symbolically, which are required for *RSPD* algorithm. In both cases, the sampling period is set to $T_s = 0.01\,\text{s}$.

The comprehensive set of experiment parameters is presented in Table 5.6 and the results of evaluation are presented in Table 5.7.

Results of the *ESRL* evaluation on the drone strike benchmark demonstrate comparable performance w.r.t. previous tasks. On average, 98% of simulations were able to hit the goal state successfully. An example of single trajectory is

| System parameters | |
|---|---|
| State space, $\mathcal{X}$ | $[-2,2]\times[-2,2]\times[0,2]\times$ $[-1.5,1.5]\times[-1.5,1.5]\times[-1.5,1.5]\times$ $[-2,2]\times[-2,2]\times[-2,2]\times$ $[-1.3,1.3]\times[-1.3,1.3]\times[-1.3,1.3]$ |
| Input space, $\mathcal{U}$ | $[-3.3354,5.0031]\times[-0.408,0.408]\times$ $[-0.408,0.408]\times[-0.408,0.408]$ |
| Discount factor, $\gamma$ | 0.95 |
| Desired state, $x_{des}$ | $[1,1,0,0,0,0,0,0,0,0,0,0]^T$ |
| Desired control, $u_{des}$ | $[0,0,0,0]^T$ |
| Sampling period, $T_s$ [s] | 0.01 |
| Simulation time, $T_{sim}$ [s] | 3 |
| **Trajectory optimization parameters** | |
| Minimal number of train samples, $N^s_{train}$ | 1500 |
| Minimal number of train trajectories, $N^t_{train}$ | 15 |
| Minimal number of test samples, $N^s_{test}$ | 5000 |
| Minimal number of test trajectories, $N^t_{test}$ | 30 |
| **ModeABC parameters** | |
| Small impact error weight | 100 |
| Medium impact error weight | 10 |
| Big impact error weight | 1 |
| Number of support points, $K$ | 1000 |
| Minimal difference with the goal state, $\omega$ | -0.1 |
| **SNGP parameters** | |
| Population size | 1000 |
| Elementary functions | *, +, -, $x^2$, $x^3$, $x^4$, Logistic3, BentIdentityGeneral |
| Maximal depth of features | 5 |
| Maximal number of features | 30 |
| Max number of generations | 2000 |
| Local search iterations | 2000 |
| Max number of threads | 10 |
| Number of runs | 15 |

**Table 5.6:** Drone strike experiment parameters

depicted in Figure 5.9. Steady-state error, however, tends to be significantly bigger in comparison with previous benchmarks. The main reason for it is the difference in state transition computation: fourth-order Runge-Kutta has been used for computing "true" transition, while *RSPD* algorithm used a less accurate forward Euler method for the prediction. Analogously to *QSPD* algorithm, it led to slightly worse results. This issue, however, can be solved by using an input-output system model instead of numeric integration of the dynamics, as described in [45]. Incorporating input-output models into *ESRL* framework may be a part of future research. Found symbolic models of V-function are severely underdeveloped, as indicated by their mean required horizon $H$. However, it was expected since the drone strike benchmark requires a much more complex approximation than previous benchmarks due to the larger dimensionality. Additionally, the set of training samples is relatively sparse w.r.t. the dimensionality. While the sparsity dramatically

| | $D_m$, [m] | $S_m$, [%] | $\dot{R}_m$ | $\ddot{R}_m$ | $\dot{O}_m$, [%] | $\ddot{O}_m$, [%] | $H_m$ |
|---|---|---|---|---|---|---|---|
| Model 1 | 0.037 | 98 | -143629.326 | -128541.661 | 91.90 | 95.79 | 32 |
| Model 2 | 0.199 | 96 | -162594.685 | -132081.578 | 93.16 | 96.75 | 32 |
| Model 3 | 0.166 | 98 | -165728.213 | -134850.924 | 86.36 | 92.43 | 32 |
| Model 4 | 0.029 | 98 | -145980.945 | -119953.478 | 90.03 | 92.67 | 32 |
| Model 5 | 0.174 | 98 | -163982.415 | -125444.518 | 90.75 | 96.60 | 32 |
| Model 6 | 0.037 | 100 | -154423.695 | -127660.585 | 89.64 | 96.00 | 32 |
| Model 7 | 0.199 | 100 | -140998.949 | -120375.886 | 91.88 | 96.19 | 32 |
| Model 8 | 0.166 | 100 | -144918.915 | -123635.479 | 91.37 | 96.45 | 32 |
| Model 9 | 0.029 | 98 | -140950.084 | -127051.71 | 90.71 | 95.48 | 31 |
| Model 10 | 0.174 | 96 | -153894.384 | -132719.763 | 89.97 | 95.25 | 32 |
| Model 11 | 0.037 | 98 | -152588.482 | -128071.35 | 92.66 | 96.45 | 32 |
| Model 12 | 0.199 | 100 | -139246.01 | -118179.925 | 92.33 | 95.12 | 32 |
| Model 13 | 0.166 | 98 | -158296.068 | -122275.878 | 90.55 | 96.06 | 32 |
| Model 14 | 0.029 | 96 | -154320.318 | -122752.728 | 91.97 | 97.42 | 32 |
| Model 15 | 0.174 | 96 | -151477.022 | -133821.119 | 91.13 | 95.89 | 32 |
| Benchmark | $D$, [m] | $S$, [%] | $\dot{R}$ | $\ddot{R}$ | $\dot{O}$, [%] | $\ddot{O}$, [%] | $H$ |
| | 0.121 | 98 | -151535.301 | -124328.031 | 90.96 | 95.64 | 32 |

**Table 5.7:** Drone strike experiment results

benefits the computational time, it damages the overall quality of the result approximation. However, despite underdeveloped symbolic models, *RSPD* algorithm was able to recover from that drawback.

## 5.5 Conclusion

This chapter consists of three main parts. Firstly, it analyses the researched landscape of SR in RL, mapping potentially useful strategies onto taxonomy. It helps to define the most promising research areas and key requirements for the *ESRL* framework.

Secondly, it identifies missing components on the landscape of symbolic reinforcement learning: trajectory optimization sampling and extension of the *QSPD* algorithm. Both techniques were evaluated in a proof-of-concept manner with satisfactory results. While each of them is not novel nor directly applicable to the thesis, their synergy allowed building the framework.

Thirdly, the Efficient Symbolic RL framework is composed of components or research conclusions from the whole thesis. The results demonstrate both robustness and scalability, as indicated in Tables 5.3, 5.5 and 5.7. Additionally, the resulting framework satisfies all the desired requirements defined earlier in Section 5.1. Successful creation of *ESRL* answers the **RQ 4** and fulfils the main goal of this thesis.

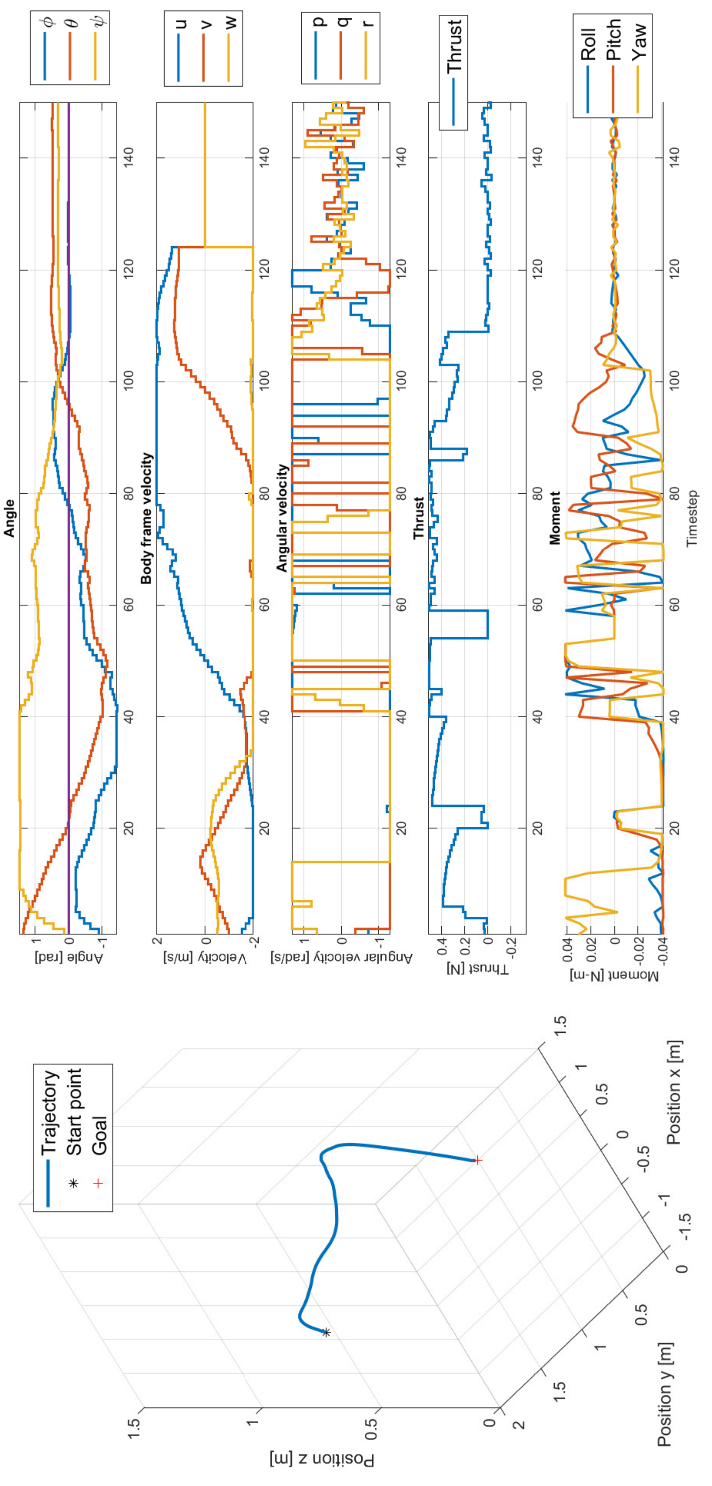

**Figure 5.9:** Example of a single trajectory from the drone strike benchmark, obtained via efficient symbolic RL algorithm

# Chapter 6

# Conclusion

The dissertation is divided into five parts. The first chapter consists of introductory material that provides an overview of the current state of the research in the field and the definition of the key research questions. Following the short introduction, short preliminaries on reinforcement learning and symbolic regression are given. Then, it is followed by a comprehensive overview of the related work. The introduction ends with a strict definition of the scope for both RL and SR. From RL perspective, the scope is defined to be: discrete-time, deterministic, model-based, fully-observable reinforcement learning, with continuous state spaces, action spaces, and reward function. From the SR point of view, the scope is not limited to specific techniques, but several variants of SNGP were used throughout the dissertation.

The subsequent four parts are designed using the following high-level pattern: something is assumed to be known or given; what issues will arise in attempts to solve correspondig research question? Each part is dedicated to particular research question, duplicated below for better readability:

**RQ 1.** How to use a symbolic approximator efficiently? What benefits could be achieved?

**RQ 2.** Is it possible to fit a value function? How to build a minimum valuable product?

**RQ 3.** What criterion should be optimized?

**RQ 4.** How can symbolic regression be integrated into reinforcement learning efficiently?

In Chapter 2, symbolic optimal V-function, as well as reward and dynamic functions are assumed to be given. A variety of policy derivation methods have been introduced using these components. Their key purpose is to study and mitigate the negative influence of discrete actions on overall performance. During this study, the positive effect of applying SR onto known V-function approximation has been discovered. In addition to that, a Quasi-Symbolic Policy Derivation (*QSPD*) algorithm has been invented. While solving the issue of using the discrete actions mentioned earlier, it proved to be exceptionally well suitable to use with a symbolic approximation of the V-function. All the proposed methods have been evaluated on three benchmarks. The results indicated the superiority of invented methods w.r.t. the standard policy derivation approach. This chapter fully answers the **RQ 1**.

Chapter 3 relaxed assumption on having symbolic optimal V-function. Instead, any optimal V-function is assumed to be given, regardless of its structure. A concept of *proxy-function* has been introduced using these assumptions. A proxy-function is defined in a way that translates to the most simplified SR task. It led to the following conclusions:

- Single symbolic expression lacks the flexibility to represent even this simplified function.
- Even simplified proxy-function can outperform standard approximation methods due to its inherent smoothness in the sense of the absence of approximation artifacts.
- Proxy-functions tend to form surfaces that violates Lyapunov stability. Despite that fact, the derived policy is reasonable and optimal.

In addition to the initially proposed proxy-function, the concept of generalized proxy-function has been proposed. It was done by redefining the fitness function for SR using multiple symbolic expressions combined utilizing linear programming. The generalized version of proxy-function demonstrated superiority to both original proxy-function and standard methods. This chapter provides a minimum viable product required by the **RQ 2**. However, the first part of the research question is to be addressed in the following chapter.

In Chapter 4, assumptions have been relaxed again, providing only a few samples from the optimal V-function. The main research question was "why fitness value does not correlate with the quality of the derived policy?". During this study, the influence of approximation error on each sample has been studied. In addition to that, several conditions have been established to force SR into generating "reasonable" symbolic approximations. The result of analysis alongside defined conditions has been encoded into linear programs (Mode**A/AB/ABC**), analogously to generalized proxy-function. All these linear programs have been evaluated using both fuzzy approximation and SR. The results indicated that Mode**ABC** consistently produced better V-function approximators w.r.t. the derived policy. Moreover, the fitness value, produced by the Mode**ABC**, started to correlate with the expected quality of the policy derivation. Altogether, it allowed using symbolic V-function with the best-achieved fitness value instead of an exhaustive evaluation of all the candidates. This chapter fully answers to the **RQ 3** and provides the missing part of the answer for **RQ 2**.

Chapter 5 relaxed the assumption on having any knowledge of the optimal V-function. Instead, the optimal V-function was meant to be learned from scratch. At the beginning of the section, the whole landscape of previous research has been analyzed. It helped to identify the desired properties of the hypothetical efficient SR for RL algorithm. At the same time, it prioritized a few areas of research with the potential: Mode**ABC** fitting procedure, *QSPD* algorithm, and trajectory optimization as the sampling technique. The latter one has been defined and tested in a proof-of-concept manner. Results indicated that it could be a viable method of collecting samples of optimal V-function while satisfying all the desired properties defined earlier. Then, the *QSPD* algorithm has been extended into Robust Symbolic Policy Derivation (*RSPD*) algorithm, which added a level of resistance against suboptimal symbolic approximators. In the subsequent section, all these components were wired into a solid system, forming the Efficient SR for RL *ESRL* framework. It has been extensively tested on three benchmarks: pendulum swing-up, magnetic manipulation, and a drone strike. The last one has higher dimensionality than any benchmark used throughout this thesis and most of the benchmarks found in related works. The results indicated both the robustness and scalability of the proposed solution. Additionally, the resulting framework satisfies all the desired properties defined earlier. This chapter fully answers **RQ 4**.

The overall goal of the thesis has been defined as:

Develop methods to integrate symbolic regression into reinforcement learning in continuous spaces.

Invented methods combined with experimental results allow considering the goal of this dissertation to be fulfilled. This thesis can facilitate the next step towards making the large-scale industrial implementation of symbolic regression for reinforcement learning in continuous spaces.

## 6.1 Suggestions for further work

The number of arising challenges throughout the research for this dissertation appeared to be overwhelmingly high. It resulted in many open questions and challenges left for future research. In some cases, parts of this dissertation may form a solid baseline for research. In other cases, the surface is barely scratched. In a list of suggestions presented below, the subjective measure of effort is indicated by "+" (easy) and "*" (hard).

+ Is that possible to use an adaptive grid instead of a fixed one in *Grid* algorithm (see Section 2.2.2 for further details)?

+ What approximation techniques may achieve the same effect as SR on V-function surface smoothing? (see Section 2.2.4).

* How to measure approximation artifacts' influence on policy derivation without explicit simulations?

* Is that possible to combine Sontag's formula [108] with *QSPD*?

+ Generalized proxy-function (see Section 3.2) can be incorporated into neural-network-based approximators as a last layer. How will it simplify the architecture for such a network?

* * Ideal V-function should be analogous to Control-Lyapunov function. However, in the approximate case, a formal analysis is missing.
* + How Mode**ABC** (see Section 4.4) fitting procedure will influence learning part of approximate RL using conventional algorithms?
* * Mode**ABC** uses quite a rough shape of the error function, which is responsible for error redistribution. Is that possible to use more sophisticated error functions? Can they be optimized for the task?
* + *RSPD* algorithm (see Section 5.3) uses fixed look-ahead horizon for the whole policy derivation. Can it be adaptive?
* + *RSPD* requires symbolic partial derivatives of system dynamics to be functional. If an accurately approximated version of such dynamics will be used instead, how may it speed up the computation time?
* * Is SR with a limited number of features, but the unlimited depth of such features can be considered as a universal approximator analogously to deep neural networks? Which features should be used?

# Appendix A

# Benchmarks

## A.1 1-DOF pendulum swing-up

The inverted pendulum consists of a mass $m$ attached to an actuated link that rotates in the vertical plane (see Figure A.1). The available torque is insufficient to push the pendulum up in a single rotation from the majority of initial states. Instead, from a certain state (e.g., pointing down), the pendulum needs to be swung back and forth to gather energy, before it can be pushed up and stabilized.

The continuous-time model of the pendulum dynamics is:

$$\ddot{\alpha} = \frac{1}{J} \cdot \left[ mgl \sin(\alpha) - b\dot{\alpha} - \frac{K^2}{R}\dot{\alpha} + \frac{K}{R}u \right] \tag{A.1}$$

where $J = 1.91 \cdot 10^{-4}\,\mathrm{kgm}^2$, $m = 0.055\,\mathrm{kg}$, $g = 9.81\,\mathrm{ms}^{-2}$, $l = 0.042\,\mathrm{m}$, $b = 3 \cdot 10^{-6}\,\mathrm{Nms/rad}$, $K = 0.0536\,\mathrm{Nm/A}$, $R = 9.5\,\Omega$. The angle $\alpha$ varies in the interval $[-\pi, \pi]$, with $\alpha = 0$ pointing up, and 'wraps around' so that e.g. $\alpha = 3\pi/2$ corresponds to $\alpha = -\pi/2$. The state vector is $x = [\alpha, \dot{\alpha}]^T$. The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics. The control action $u$ is limited to $[-2, 2]\,\mathrm{V}$, which is insufficient to push up the pendulum in one go.

The control goal is to stabilize the pendulum in the unstable equilibrium $\alpha = \dot{\alpha} = 0 = x_{des}$ using desired control input $u_{des} = 0$.

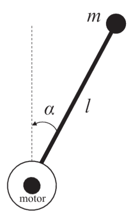

**Figure A.1:** Inverted pendulum schematic.

## A.2 2-DOF pendulum swing-up

The double pendulum is described by the continuous-time fourth-order non-linear model:

$$
\begin{aligned}
u &= M(\alpha)\ddot{\alpha} + C(\alpha,\dot{\alpha})\alpha + G(\alpha) \\
M(\alpha) &= \begin{bmatrix} P_1 + P_2 + 2P_3\ \cos(\alpha_2) & P_2 + P_3\ \cos(\alpha_2) \\ P_2 + P_3\ \cos(\alpha_2) & P_2 \end{bmatrix} \\
C(\alpha,\dot{\alpha}) &= \begin{bmatrix} b_1 - P_3\dot{\alpha}_2 \sin(\alpha_2) & -P_3(\dot{\alpha}_1 + \dot{\alpha}_2)\sin(\alpha_2) \\ P_3\dot{\alpha}_1 \sin(\alpha 2) & b_2 \end{bmatrix} \\
G(\alpha) &= \begin{bmatrix} -F_1 \sin(\alpha_1) - F_2 \sin(\alpha_1 + \alpha_2) \\ -F_2 \sin(\alpha_1 + \alpha_2) \end{bmatrix}
\end{aligned}
\tag{A.2}
$$

where $\alpha = [\alpha_1, \alpha_2]^T$ contains the angular positions of the two links, $u = [u_1, u_2]^T$ is the control input which contains the torques of the two motors, $M(\alpha)$ is the mass matrix, $C(\alpha, \dot{\alpha})$ is the Coriolis and centrifugal forces matrix and $G(\alpha)$ is the gravitational forces vector. The state $x$ contains the angles and angular velocities and is defined as $x = [\alpha_1, \dot{\alpha}_1, \alpha_2, \dot{\alpha}_2]^T$. The angles $[\alpha_1, \alpha_2]$ vary in the interval $[-\pi, \pi)$ rad and wrap around. The auxiliary variables are defined as $P_1 = m_1c_1^2 + m_2l_1^2 + I_1$, $P_2 = m_2c_2^2 + I_2$, $P_3 = m_2l_1c_2$, $F_1 = (m_1c_1 + m_2l_2)g$ and $F_2 = m_2c_2g$. The control action $u$ is limited to $[-3, 3]$ for the first link and $[-1, 1]$ for the second link. The control goal is to stabilize the two links at the unstable equilibrium $\alpha = \dot{\alpha} = 0$. The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics. The meaning and values of the system parameters are given in Table A.1.

| Model parameter | Symbol | Value | |
|---|---|---|---|
| | Unit | | |
| Link lengths | $l_1, l_2$ | 0.4, 0.4 | m |
| Link masses | $m_1, m_2$ | 1.25, 0.8 | kg |
| Link inertias | $I_1, I_2$ | 0.0667, 0.0427 | $\mathrm{kg\,m^2}$ |
| Center of mass coordinates | $c_1, c_2$ | 0.2, 0.2 | m |
| Damping in the joints | $b_1, b_2$ | 0.08, 0.02 | kg/s |
| Gravitational acceleration | $g$ | 9.81 | $\mathrm{m/s^2}$ |

**Table A.1:** Double pendulum parameters

## A.3 Magnetic manipulation

Magnetic manipulation (abbreviated as Magman) is an challenging nonlinear control problem. In the setup used for this thesis, the current through the electromagnets is controlled to dynamically shape the magnetic field above the magnets and so to accurately and quickly position a steel ball to the desired set point. The setup is depicted schematically in Figure A.2.

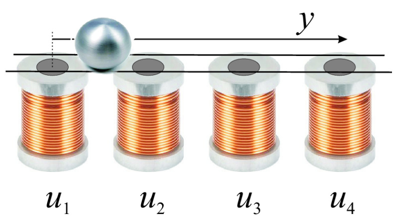

**Figure A.2:** Magman schematic.

The horizontal acceleration of the ball is given by:

$$\ddot{y} = -\frac{b}{m}\dot{y} + \frac{1}{m}\sum_{i=0}^{1} g(y,i)\, u_i \tag{A.3}$$

with

$$g(y,i) = \frac{-c_1\,(y - 0.025i)}{\left((y - 0.025i)^2 + c_2\right)^3}. \tag{A.4}$$

Here, $y$ denotes the position of the ball, $\dot{y}$ its velocity and $\ddot{y}$ the acceleration. With $u_i$ the current through coil $i$, $g(y,i)$ is the nonlinear magnetic force equation, $m$ (kg) the ball mass, and $b$ $(\frac{Ns}{m})$ the viscous friction of the ball on the rail. The number of coils in the benchmark is adjustable. E.g. the five-coil variant uses $i = 0, 1, 2, 3, 4$. The model parameters are listed in Table A.2. The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics. State $x$ is given by the position and velocity of

| Model parameter | Symbol | Value | Unit |
|---|---|---|---|
| Ball mass | $m$ | $3.200 \cdot 10^{-2}$ | kg |
| Viscous damping | $b$ | $1.613 \cdot 10^{-2}$ | Nms |
| Empirical parameter | $c_1$ | $5.520 \cdot 10^{-10}$ | $\mathrm{Nm^5A^{-1}}$ |
| Empirical parameter | $c_2$ | $1.750 \cdot 10^{-4}$ | $\mathrm{m^2}$ |

**Table A.2:** Magnetic manipulation system parameters

the ball.

## A.4 Drone strike

Drone strike is a challenging nonlinear control problem designed to test the scalability aspects of the proposed methods. The harsh initial state represents thrown drone. The goal of the benchmark is to stabilize it and then hit the desired point in a kamikaze way. The state-space of the system is represented by a 12-D vector, while a 4-D vector represents the action space. For this work, the dynamics model was adapted from [109], including a comprehensive mathematical model. Drone parameters are given for commercially available HobbyKing FPV250 V2 Quadcopter. The parameters are obtained by modeling the plant in Solidworks, as described in [110, 111].

The dynamics of the drone is described in two coordinate systems: the inertial and body-fixed coordinates. The transformation between these two systems is done using Euler angles, where $\phi$, $\theta$, and $\psi$ stand for roll, pitch, and yaw, respectively. The rotation matrix from the body frame to the inertial frame is described by:

$$R_b^i = \begin{bmatrix} \cos(\theta)\cos(\psi) & \sin(\phi)\sin(\theta)\cos(\psi) - \cos(\phi)\sin(\psi) & \cos(\psi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi) \\ \cos(\theta)\sin(\psi) & \sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \cos(\psi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) \\ -\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix} \tag{A.5}$$

Inverse transformation is defined as:

$$R_i^b = \begin{bmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ \sin(\phi)\sin(\theta)\cos(\psi) - \cos(\phi)\sin(\psi) & \sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \sin(\phi)\cos(\theta) \\ \cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi) & \cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) & \cos(\phi)\cos(\theta) \end{bmatrix} \tag{A.6}$$

The setup is depicted schematically in Figure A.3 alongside with both coordinate systems.

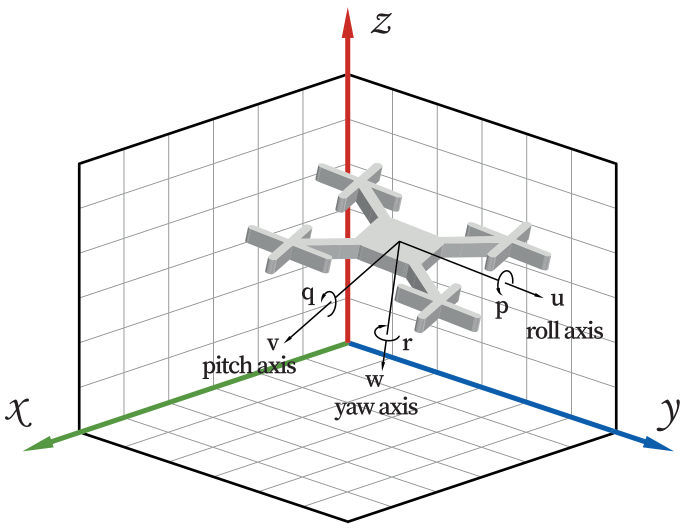

**Figure A.3:** Drone schematic.

The following set of equations represents the complete 6-DOF drone model:

$$
\begin{aligned}
\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} &= R_b^i \begin{bmatrix} u \\ v \\ w \end{bmatrix} \\
\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} &= \begin{bmatrix} 1 & \frac{\sin(\phi)\sin(\theta)}{\cos(\theta)} & \frac{\cos(\phi)\sin(\theta)}{\cos(\theta)} \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \frac{\sin(\phi)}{\cos(\theta)} & \frac{cos(\phi)}{\cos(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \\
\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \\ -T/m \end{bmatrix} + R_i^b \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \begin{bmatrix} rv - qw \\ pw - ur \\ qu - pv \end{bmatrix} \\
\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} &= \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}} qr \\ \frac{I_{zz}-I_{xx}}{I_{yy}} pr \\ \frac{I_{xx}-I_{yy}}{I_{zz}} pq \end{bmatrix} + \begin{bmatrix} \frac{1}{I_{xx}} l \\ \frac{1}{I_{yy}} m \\ \frac{1}{I_{zz}} n \end{bmatrix}
\end{aligned}
\tag{A.7}
$$

All the variables are described in Table A.3 alongside plant parameters.

| | Variable | Meaning | Unit | Lower limit | Upper limit |
|---|---|---|---|---|---|
| State | $x$ | Position along x axis | $m$ | -2 | 2 |
| | $y$ | Position along y axis | $m$ | -2 | 2 |
| | $z$ | Position along z axis | $m$ | 0 | 2 |
| | $\phi$ | Roll angle | $rad$ | -1.5 | 1.5 |
| | $\theta$ | Pitch angle | $rad$ | -1.5 | 1.5 |
| | $\psi$ | Yaw angle | $rad$ | -1.5 | 1.5 |
| | $u$ | Velocity along roll axis | $m/s$ | -2 | 2 |
| | $v$ | Velocity along pitch axis | $m/s$ | -2 | 2 |
| | $w$ | Velocity along yaw axis | $m/s$ | -2 | 2 |
| | $p$ | Roll rate | $rad/s$ | -1.3 | 1.3 |
| | $q$ | Pitch rate | $rad/s$ | -1.3 | 1.3 |
| | $r$ | Yaw rate | $rad/s$ | -1.3 | 1.3 |
| Action | $T$ | Thrust | $N$ | -3.3354 | 5.0031 |
| | $l$ | Rolling moment | $Nm$ | -0.408 | 0.408 |
| | $m$ | Pitching moment | $Nm$ | -0.408 | 0.408 |
| | $n$ | Yawing moment | $Nm$ | -0.408 | 0.408 |
| Parameter | $I_{xx}$ | Moment of inertia along roll axis | $kgm^2$ | $321862.60 \cdot 10^{-9}$ | |
| | $I_{yy}$ | Moment of inertia along pitch axis | $kgm^2$ | $305825.42 \cdot 10^{-9}$ | |
| | $I_{zz}$ | Moment of inertia along yaw axis | $kgm^2$ | $576255.87 \cdot 10^{-9}$ | |
| | $g$ | Gravitational acceleration | $m/s^2$ | 9.81 | |

**Table A.3:** Drone strike variables and system parameters

The benchmark is designed w.r.t. the following assumptions:

- Aerodynamic forces are insignificant and can be neglected. Thus, only the propulsion and gravitational forces are considered.

- Drone is symmetric along both roll and pitch axes. In other words, it is assumed that the drone is a standard quadcopter.
- No noise is presented for the sake of simplicity.
- Any crash to land is considered catastrophic; thus, the drone's position is not changing after the strike.

# Appendix B

# Single Node Genetic Programming

Single Node Genetic Programming (SNGP) is a graph-based GP method that evolves a population of individuals, each consisting of a single program node. The node can be either a terminal, i.e., a constant or a variable in case of the symbolic regression problem, or a function chosen from a set of functions defined for the problem at hand. The individuals are not entirely distinct; instead, they are interlinked in a graph structure, so some individuals act as input operands of other individuals.

Formally, a SNGP population is a set of $L$ individuals $M = \{m_0, m_1, \ldots, m_{L-1}\}$, with each individual $m_i$ being a single node represented by the tuple $m_i = \langle e_i, f_i, Succ_i, Pred_i, O_i \rangle$, where

- $e_i \in T \cup F$ is either an element chosen from a function set $F$ or a terminal set $T$ defined for the problem;
- $f_i$ is the fitness of the individual;
- $Succ_i$ is a set of successors of this node, i.e. the nodes whose output serves as the input to the node;
- $Pred_i$ is a set of predecessors of this node, i.e. the nodes that use this node as an operand;
- $O_i$ is a vector of outputs produced by this node.

The following basic operators and functions to build the symbolic expressions were used throughout the thesis, namely, $F = \{*, +, -, square, cube, sin, tanh, BentGeneral, Logistic3\}$ where

$$BentGeneral(x_1, \ldots, x_N) = \prod_{i=1}^{N} [x_i + (\sqrt{x_i^2 + 1.0} - 1)/2] \tag{B.1}$$

and

$$\begin{aligned} Logistic3(x_1, x_2, x_3) = x_1(1 - (1/(1 + e^{-x_3})))) + \\ + x_2(1/(1 + e^{-x_3})) \end{aligned} \tag{B.2}$$

and N is the arity of input. The terminal set $T$ consisted of the state variables $x_i$ and basic constants 0, 1, 2, and 3.

Typically, the population is partitioned so that the first $L_{term}$ nodes are terminals, basic constants, and variables, followed by function nodes. Links between nodes in the population must satisfy a condition that any function node can use as its successor (i.e., the operand) only nodes positioned lower down in the population. This means that for each $m_j \in Succ_i$ it holds $0 \leq j < i$. Similarly, predecessors of individual $i$ must occupy higher positions in the population, i.e. $\forall m_j \in Pred_i,\ i < j < L$. Note that each function node is, in fact, a root of a program tree that can be constructed by recursively traversing its successors towards the leaf terminal nodes.

An operator called *successor mutation* (*smut*), proposed in [21], is used to modify the population. It takes an individual and replaces one of its successors by a reference to another randomly chosen individual of the population, making sure that the constraint imposed on the successors is satisfied. Output values of the mutated node and all nodes higher up in the population affected by the mutation operation are recalculated. Moreover, the predecessor lists of all affected nodes are updated accordingly.

Finally, the population is evolved using a local search-like procedure. In each iteration, a new population is produced by the *smut* operator, which is then accepted for the next iteration if it is no worse than the original population.

The SNGP implementation used in this thesis evolved over time, resulting in three different versions. They are all based on the implementation described in [22, 21] but differ in several aspects listed below. The presented version is considered to be a default one unless some aspects are stated to be different.

## B.1 SNGP v1

- Organization of nodes in the population. Typically, the function set contains functions that might produce invalid output values, such as a division by zero. In order to avoid such cases, protected versions of these functions are used instead. These functions are forced to produce a valid output for any input. For example, the protected division outputs a predefined value whenever the denominator is zero. Thus, the output of any candidate expression is ensured to be a valid number. However, due to such hard-coded irregular behavior of the protected functions, expressions using the protected functions can still exhibit undesired behavior, e.g., the expressions might become non-differentiable at some data points or contain local approximation artifacts. Even though it has a minimal error on the training data, such a symbolic function can be effectively useless when applied to previously unseen data. Here, a *partitioned population* is proposed. It is divided into two parts – *head* and *tail*. The head part contains nodes that are roots of constant-valued expressions only. It uses extended function set $F_e$ including the protected functions. Each head node can use other head nodes and constant terminal nodes as its input. The tail part contains nodes that can only be chosen from a set $F_s$ of simple non-conflicting functions (i.e., no protected functions). Tail nodes can use all preceding head and tail nodes and both constants and variables as their input. In this way, a "reasonable" behavior of expressions rooted in tail nodes is ensured since the protected functions are used only to produce constants.

- Selection strategy used to choose the nodes to be mutated. The original SNGP selects the nodes to be mutated purely at random. Here the so-called depth-wise selection introduced in [112] is used. This selection method is biased toward deeper nodes of well-performing expressions. The idea behind such a strategy is that changes made to the nodes at deeper levels are more likely to bring an improvement than changes applied to the nodes closer to the root of the expression. The quality of the individual nodes is assessed as the mean squared error produced by the expression rooted in the node.

## B.2 SNGP v2

This version extends SNGP v1 by further modifications of the following aspects:

- Form of the final model. A hybrid SNGP proposed in [112] and denoted as the Single-Run SNGP with LASSO (s-SNGPL) is used. It produces a generalized linear regression model composed of possibly nonlinear features represented by expressions rooted in the tail partition nodes. The generalized linear regression models are built using the Least Absolute Shrinkage and Selection (LASSO) regression technique [113]. In this way, precise, linear-in-parameters nonlinear regression models can be produced. The complexity of the LASSO model is controlled by:
  - the maximal depth of features evolved in the population
  - the maximum number of features the LASSO model can be composed of
- Fitness function. The generalized regression models are optimized with respect to the mean squared error calculated over the set of training samples unless stated otherwise.
- Processing mode to evolve the population. The process of evolving the population is carried out in epochs. In each epoch, multiple independent parallel threads are run for a predefined number of generations, all of them starting from the same population – the best final population out of the previous epoch threads. This reduces the chance of getting stuck in a local optimum.

The result symbolic model $A(x)$ is then composed from the linear combination of possibly non-linear analytic expressions $p_1, \dots, p_q$, as:

$$A(x) = \beta_0 + \beta_1 p_1(x) + \beta_2 p_2(x) + \dots + \beta_q p_q(x)$$

the following restrictions are applied in order to control overfitting:

- For every point in the dataset, the result of an analytic expression lies within $[-10^8, 10^8]$ interval.
- Difference between the maximum and minimum values of an analytic expression computed on the given dataset lies within $[10^{-5}, 10^3]$ interval.
- Weights $\beta_1, \ldots, \beta_q$ lie within $[-1, 1]$ interval.

## B.3 SNGP v3

SNGP v3 modifies the processing mode to evolve the population from SNGP v2. The maximal number of epochs $N$ is shaped dynamically, as well as the number of parallel threads in each epoch $T_k$ and length of that epoch $L_k$. These parameter depend on the maximal number of generations $G_{max}$ and maximal number of threads $T_{max}$. Relations are defined through the following algorithm:

**Algorithm 8:** Computation of number of epochs, length of each epoch and number of threads in each epoch.

```
Input: $N_{max,T_{max}}$
$k \leftarrow 1$
$T_k \leftarrow T_{max}$
$t = T_{max}$
while $t > 0.5$ do
    $t \leftarrow t \cdot 0.4$
    $k \leftarrow k + 1$
    $T_k \leftarrow \lceil t \rceil$
end
$N \leftarrow k$
for $k \leftarrow 1 to N$ do
    $L_k \leftarrow \lceil (G_{max}/N)/T_k \rceil$
end
Output: $N, [T_1 \ldots T_N], [L_1 \ldots L_N]$
```

At the end of each $k$ epoch, best $T_{k+1}$ solutions are selected according to the defined fitness function. These solutions continue in the evolution process, while others are neglected. This computation scheme runs many short threads at the beginning, allowing SNGP to test more initial vectors. This reduces the chance of getting stuck in a local optimum right at the start of the evolution process.

# Appendix C
# Publications

**Journal Publications (indexed in Web of Science) related to the dissertation topic**

1. [114] Alibekov, Eduard and Kubalík, Jiří and Babuška, Robert. "Policy derivation methods for critic-only reinforcement learning in continuous spaces". 2018 Pergamon, Engineering Applications of Artificial Intelligence. doi:10.1016/J.ENGAPPAI.2017.12.00

   **Impact factor: 6.212**
   **Quartile: Q1**
   **Citations: 7**

**Conference Publications (indexed in Web of Science) related to the dissertation topic**

1. [61] Alibekov, Eduard and Kubalík, Jiří and Babuška, Robert. "Symbolic method for deriving policy in reinforcement learning". 2016 IEEE, 55th Conference on Decision and Control (CDC). doi:10.1109/CDC.2016.7798684

   **Core: A**
   **Citations: 10**

2. [115] Alibekov, Eduard and Kubalík, Jiří and Babuška, Robert, "Policy derivation methods for critic-only reinforcement learning in continuous action spaces", 2016 Elsevier, IFAC-PapersOnLine. doi:10.1016/J.IFACOL.2016.07.127

   **Core: C**
   **CiteScore: 2.1**
   **Citations: 5**

3. [38] Kubalík, Jiří and Alibekov, Eduard and Babuška, Robert. "Optimal control via reinforcement learning with symbolic policy approximation". 2017 Elsevier, IFAC-PapersOnLine. doi:10.1016/J.IFACOL.2017.08.805

   **Core: C**
   **CiteScore: 2.1**
   **Citations: 5**

4. [116] Alibekov, Eduard and Kubalík, Jiří and Babuška, Robert, “Proxy Functions for Approximate Reinforcement Learning”, 2019 Elsevier, IFAC-PapersOnLine. doi:10.1016/J.IFACOL.2019.09.145

   **Core: C**
   **CiteScore: 2.1**

**Other Publications (indexed in Web of Science) related to the dissertation topic**

1. [112] Kubalík, Jiří and Alibekov, Eduard and Žegklitz, Jan and Babuška, Robert. “Hybrid Single Node Genetic Programming for Symbolic Regression”. 2016 Springer, Transactions on Computational Collective Intelligence XXIV. doi:10.1007/978-3-662-53525-7_4 **Citations: 6**

# Appendix D

# Acronyms

- AI - Artificial Intelligence
- RL - Reinforcement Learning
- SR - Symbolic Regression
- SNGP - Single Node Genetic Programming
- RQ - Research Question
- GP - Genetic Programming
- CGP - Cartesian Genetic Programming
- DSP - Deep Symbolic Policy
- BF - Basis Function
- RBF - Radial Basis Function
- MSE - Mean Squared Error
- RHS - Right-Hand-Side
- LHS - Left-Hand-Side
- *QSPD* - Quasi-Symbolic Policy Derivation
- TRR - Trust Region Reflective
- DOF - Degree(s) of Freedom
- CLF - Control Lyapunov Function
- MPC - Model Predictive Control
- *RSPD* - Robust Symbolic Policy Derivation
- *ESRL* - Efficient SR for RL

# Bibliography

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[3] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," *arXiv preprint arXiv:1909.07528*, 2019.

[4] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[6] A. C. Clarke, *Profiles of the Future.* Hachette UK, 2013.

[7] S. Paul, V. Kurin, and S. Whiteson, "Fast efficient hyperparameter tuning for policy gradients," *arXiv preprint arXiv:1902.06583*, 2019.

[8] J. Tarbouriech, M. Pirotta, M. Valko, and A. Lazaric, "A provably efficient sample collection strategy for reinforcement learning," *arXiv preprint arXiv:2007.06437*, 2020.

[9] T. Tompa and S. Kovács, "Applying expert heuristic as an a priori knowledge for friq-learning," *Acta Polytechnica Hungarica*, vol. 17, no. 4, pp. 27–45, 2020.

[10] R. Sutton and A. Barto, *Reinforcement learning: An introduction*, vol. 116. Cambridge Univ Press, 1998.

[11] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics.," *Journal of Intelligent and Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[12] L. Kuvayev and R. S. Sutton, "Model-based reinforcement learning with an approximate, learned model," in *Proc. Yale Workshop Adapt. Learn. Syst*, pp. 101–105, 1996.

[13] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *SIAM Journal on Control and Optimization*, pp. 1008–1014, MIT Press, 2000.

[14] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*, vol. 39. CRC press, 2010.

[15] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[16] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 104, no. 24, pp. 9943–9948, 2007.

[17] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *science*, vol. 324, no. 5923, pp. 81–85, 2009.

[18] C. Ryan, J. Collins, and M. O. Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *European Conference on Genetic Programming*, pp. 83–96, Springer, 1998.

[19] C. Ferreira, *Gene Expression Programming in Problem Solving*, pp. 635–653. London: Springer London, 2002.

[20] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *European Conference on Genetic Programming*, pp. 121–132, Springer, 2000.

[21] D. Jackson, "A new, node-focused model for genetic programming," in *European Conference on Genetic Programming*, pp. 49–60, Springer, 2012.

[22] D. Jackson, "Single node genetic programming on problems with side effects," in *International Conference on Parallel Problem Solving from Nature*, pp. 327–336, Springer, 2012.

[23] J. Kubalík and R. Babuška, "An improved single node genetic programming for symbolic regression," in *2015 7th International Joint Conference on Computational Intelligence (IJCCI)*, vol. 1, pp. 244–251, IEEE, 2015.

[24] H. Iba, "Multi-agent reinforcement learning with genetic programming," *Genetic Programming 1998*, 1998.

[25] K. L. Downing, "Adaptive genetic programs via reinforcement learning," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pp. 19–26, 2001.

[26] C. Gearhart, "Genetic programming as policy search in markov decision processes," *Genetic Algorithms and Genetic Programming at Stanford*, pp. 61–67, 2003.

[27] S. Mabu, K. Hirasawa, and J. Hu, "Genetic network programming with reinforcement learning and its performance evaluation," in *Genetic and Evolutionary Computation Conference*, pp. 710–711, Springer, 2004.

[28] S. Kamio and H. Iba, "Adaptation technique for integrating genetic programming and reinforcement learning for real robots," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 318–333, 2005.

[29] S. Girgin and P. Preux, "Feature discovery in reinforcement learning using genetic programming," in *European Conference on Genetic Programming*, pp. 218–229, Springer, 2008.

[30] S. Niekum, A. G. Barto, and L. Spector, "Genetic programming for reward function search," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 83–90, 2010.

[31] S. Niekum, L. Spector, and A. Barto, "Evolution of reward functions for reinforcement learning," in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pp. 177–178, 2011.

[32] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Icml*, vol. 99, pp. 278–287, 1999.

[33] M. Davarynejad, J. van Ast, J. Vrancken, and J. van den Berg, "Evolutionary value function approximation," in *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 151–155, IEEE, 2011.

[34] 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 *International Conference on Discovery Science*, pp. 37–51, Springer, 2012.

[35] D. C. Dracopoulos, D. Effraimidis, and B. D. Nichols, "Genetic programming as a solver to challenging reinforcement learning problems," *International Journal of Computer Research*, vol. 20, no. 3, p. 351, 2013.

[36] A. S. Koshiyama, T. Escovedo, M. M. Vellasco, and R. Tanscheit, "Gpfis-control: A fuzzy genetic model for control tasks," in *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1953–1959, IEEE, 2014.

[37] M. Onderwater, S. Bhulai, and R. van der Mei, "Value function discovery in markov decision processes with evolutionary algorithms," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 9, pp. 1190–1201, 2016.

[38] J. Kubalík, E. Alibekov, and R. Babuška, "Optimal control via reinforcement learning with symbolic policy approximation," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4162–4167, 2017.

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

[40] 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*, pp. 229–236, 2018.

[41] J. F. Miller and S. L. Harding, "Cartesian genetic programming," in *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*, pp. 2701–2726, 2008.

[42] E. Derner, J. Kubalík, and R. Babuška, "Data-driven construction of symbolic process models for reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5105–5112, IEEE, 2018.

[43] E. Derner, J. Kubalík, N. Ancona, and R. Babuška, "Constructing parsimonious analytic models for dynamic systems via symbolic regression," *Applied Soft Computing*, vol. 94, p. 106432, 2020.

[44] W. Fang and Z. Chen, "A symbolic regression method for dynamic modeling and control of quadrotor uavs," *arXiv preprint arXiv:2105.03032*, 2021.

[45] E. Derner, J. Kubalík, and R. Babuška, "Reinforcement learning with symbolic input-output models," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3004–3009, IEEE, 2018.

[46] J. Gout, M. Quade, K. Shafi, R. K. Niven, and M. Abel, "Synchronization control of oscillator networks using symbolic regression," *Nonlinear Dynamics*, vol. 91, no. 2, pp. 1001–1021, 2018.

[47] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," in *International Conference on Machine Learning*, pp. 5045–5054, PMLR, 2018.

[48] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[49] H. Zhang, A. Zhou, and X. Lin, "Interpretable policy derivation for reinforcement learning based on evolutionary feature synthesis," *Complex & Intelligent Systems*, vol. 6, no. 3, pp. 741–753, 2020.

[50] D. Hein, S. Limmer, and T. A. Runkler, "Interpretable control by reinforcement learning," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 8082–8089, 2020.

[51] A. I. Diveev, O. Hussein, and E. Y. Shmalko, "Symbolic regression based solution for the optimal control problem with constraints," *International Journal of Open Information Technologies*, vol. 8, no. 9, 2020.

[52] S. Konstantinov and A. I. Diveev, "Control system synthesis based on optimal trajectories approximation by symbolic regression for group of robots," in *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*, vol. 1, pp. 19–24, IEEE, 2020.

[53] A. Diveev, "Cartesian genetic programming for synthesis of optimal control system," in *Proceedings of the Future Technologies Conference*, pp. 205–222, Springer, 2020.

[54] A. Hristov, J. W. Bosman, S. Bhulai, and R. D. van der Mei, "Deriving explicit control policies for markov decision processes using symbolic regression," in *Proceedings of the 13th EAI International Conference on Performance Evaluation Methodologies and Tools*, pp. 41–47, 2020.

[55] J. Žegklitz and P. Pošík, "Symbolic regression in dynamic scenarios with gradually changing targets," *Applied Soft Computing*, vol. 83, p. 105621, 2019.

[56] J. Kubalík, E. Derner, and R. Babuška, "Symbolic regression driven by training data and prior knowledge," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 958–966, 2020.

[57] M. Landajuela, B. K. Petersen, S. Kim, C. P. Santiago, R. Glatt, N. Mundhenk, J. F. Pettit, and D. Faissol, "Discovering symbolic policies with deep reinforcement learning," in *International Conference on Machine Learning*, pp. 5979–5989, PMLR, 2021.

[58] F. L. Lewis, D. L. Vrabie, and V. L. Syrmos, "Optimal control of discrete-time systems," *Optimal Control, Third Edition*, pp. 19–109.

[59] J. A. Primbs, V. Nevistić, and J. C. Doyle, "Nonlinear optimal control: A control lyapunov function and receding horizon perspective," *Asian Journal of Control*, vol. 1, no. 1, pp. 14–24, 1999.

[60] D. P. Bertsekas, “Dynamic programming and optimal control 3rd edition, volume ii,” *Belmont, MA: Athena Scientific*, 2011.

[61] E. Alibekov, J. Kubalík, and R. Babuška, “Symbolic method for deriving policy in reinforcement learning,” in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 2789–2795, IEEE, 2016.

[62] J. C. Santamaria, R. S. Sutton, and A. Ram, “Experiments with reinforcement learning in problems with continuous state and action spaces,” 1996.

[63] B. Sallans and G. E. Hinton, “Reinforcement learning with factored states and actions,” *J. Mach. Learn. Res.*, vol. 5, pp. 1063–1088, Dec. 2004.

[64] H. Kimura, “Reinforcement learning in multi-dimensional state-action space using random rectangular coarse coding and gibbs sampling,” in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 88–95, 2007.

[65] J. Pazis and M. G. Lagoudakis, “Binary action search for learning continuous-action control policies,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML ’09, (New York, NY, USA), pp. 793–800, ACM, 2009.

[66] J. Pazis and M. Lagoudakis, “Reinforcement learning in multidimensional continuous action spaces,” in *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*, pp. 97–104, 2011.

[67] J. Yang, S. Huang, K. Lin, J. Czernin, P. Wolfenden, M. Dahlbom, C. Hoh, and M. Phelps, “A new axial smoothing method based on elastic mapping,” *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 3355–3360, 1996.

[68] N. Graham, “Smoothing with periodic cubic splines,” *Bell System Technical Journal*, vol. 62, no. 1, pp. 101–110, 1983.

[69] S. Ferrari and R. F. Stengel, “Smooth function approximation using neural networks,” *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 24–38, 2005.

[70] R. O. Hays, “Multi-dimensional extension of the Chebyshev polynomials,” *Mathematics of computation*, vol. 27, no. 123, pp. 621–624, 1973.

[71] L. N. Townsend, Alex; Trefethen, “An extension of Chebfun to two dimensions,” *SIAM Journal on Scientific Computing*, vol. 35, 01 2013.

[72] B. Hashemi and L. N. Trefethen, “Chebfun in three dimensions,” *Under review by the SIAM Journal on Scientific Computing*, 2016.

[73] T. A. Driscoll, N. Hale, and L. N. Trefethen, *Chebfun Guide*. Pafnuty Publications, 2014.

[74] J. P. Boyd, “Computing zeros on a real interval through Chebyshev expansion and polynomial rootfinding,” *SIAM Journal on Numerical Analysis*, vol. 40, no. 5, pp. 1666–1682, 2002.

[75] M. A. Branch, T. F. Coleman, and Y. Li, “A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems,” *SIAM Journal on Scientific Computing*, vol. 21, no. 1, pp. 1–23, 1999.

[76] L. C. Baird, “Reinforcement learning in continuous time: Advantage updating,” in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, vol. 4, pp. 2448–2453, IEEE, 1994.

[77] C. Szepesvari, *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[78] D. P. De Farias and B. Van Roy, *The linear programming approach to approximate dynamic programming: Theory and application*. PhD thesis, Ph. D. Thesis, Stanford University, 2002.

[79] G. Taylor, C. Geer, and D. Piekut, “An analysis of state-relevance weights and sampling distributions on l1-regularized approximate linear programming approximation accuracy,” in *International Conference on Machine Learning*, pp. 451–459, 2014.

[80] R. Munos and A. Moore, “Variable resolution discretization in optimal control,” *Machine learning*, vol. 49, no. 2, pp. 291–323, 2002.

[81] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, “Cross-entropy optimization of control policies with adaptive basis functions,” *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 41, no. 1, pp. 196–209, 2011.

[82] D. Ernst, P. Geurts, and L. Wehenkel, “Tree-based batch mode reinforcement learning,” *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005.

[83] C. G. Atkeson, A. W. Moore, and S. Schaal, “Locally weighted learning,” *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 11–73, 1997.

[84] I. Grondman, M. Vaandrager, L. Buşoniu, R. Babuška, and E. Schuitema, “Efficient model learning methods for actor–critic control,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 3, pp. 591–602, 2012.

[85] S. Lange, M. Riedmiller, and A. Voigtlander, “Autonomous reinforcement learning on raw visual input data in a real world application,” in *Proceedings 2012 International Joint Conference on Neural Networks (IJCNN)*, (Brisbane, Australia), pp. 1–8, June 2012.

[86] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” vol. arxiv.org/abs/1312.5602, 2013.

[87] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[88] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning.” arXiv:1509.02971 [cs.LG], 2015.

[89] T. de Bruin, J. Kober, K. Tuyls, and R. Babuška, “Off-policy experience retention for deep actor-critic learning,” in *Deep Reinforcement Learning Workshop, Advances in Neural Information Processing Systems (NIPS)*, 2016.

[90] M. Kommenda, B. Burlacu, G. Kronberger, and M. Affenzeller, “Parameter identification for symbolic regression using nonlinear least squares,” *Genetic Programming and Evolvable Machines*, pp. 1–31, 2019.

[91] C. Wilstrup and J. Kasak, “Symbolic regression outperforms other models for small data sets,” *arXiv preprint arXiv:2103.15147*, 2021.

[92] E. Derner, J. Kubalík, and R. Babuška, “Selecting informative data samples for model learning through symbolic regression,” *IEEE Access*, vol. 9, pp. 14148–14158, 2021.

[93] A. W. Moore and C. G. Atkeson, “Prioritized sweeping: Reinforcement learning with less data and less time,” *Machine learning*, vol. 13, no. 1, pp. 103–130, 1993.

[94] D. P. De Farias and B. Van Roy, “The linear programming approach to approximate dynamic programming,” *Operations research*, vol. 51, no. 6, pp. 850–865, 2003.

[95] D. P. De Farias and B. Van Roy, “On constraint sampling in the linear programming approach to approximate dynamic programming,” *Mathematics of operations research*, vol. 29, no. 3, pp. 462–478, 2004.

[96] T. Wang, M. Bowling, and D. Schuurmans, “Dual representations for dynamic programming and reinforcement learning,” in *2007 IEEE*

*International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 44–51, IEEE, 2007.

[97] M. Posa, S. Kuindersma, and R. Tedrake, “Optimization and stabilization of trajectories for constrained dynamical systems,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1366–1373, IEEE, 2016.

[98] A. Malik, T. Henderson, and R. J. Prazenica, “Trajectory generation for a multibody robotic system using the product of exponentials formulation,” in *AIAA Scitech 2021 Forum*, p. 2016, 2021.

[99] F. Morbidi, R. Cano, and D. Lara, “Minimum-energy path generation for a quadrotor uav,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1492–1498, IEEE, 2016.

[100] D. R. Herber, “Basic implementation of multiple-interval pseudospectral methods to solve optimal control problems,” tech. rep., 2015.

[101] R. Tedrake, “Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation,” *Course Notes for MIT*, vol. 6, 2016.

[102] M. Kelly, “An introduction to trajectory optimization: How to do your own direct collocation,” *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017.

[103] Y. Tassa, “iLQG/DDP trajectory optimization.” https://www.mathworks.com/matlabcentral/fileexchange/52069-ilqg-ddp-trajectory-optimization, 2021.

[104] C. E. Garcia, D. M. Prett, and M. Morari, “Model predictive control: Theory and practice—a survey,” *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[105] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[106] F. Allgöwer and A. Zheng, *Nonlinear model predictive control*, vol. 26. Birkhäuser, 2012.

[107] I. The MathWorks, *Symbolic Math Toolbox*. Natick, Massachusetts, United State, 2019.

[108] Y. Lin and E. D. Sontag, “A universal formula for stabilization with bounded controls,” *Systems & Control Letters*, vol. 16, no. 6, pp. 393–397, 1991.

[109] R. W. Beard and T. W. McLain, *Small unmanned aircraft*. Princeton university press, 2012.

[110] B. Prabhakaran, M. Kothari, *et al.*, “Nonlinear control design for quadrotors,” in *2015 IEEE Workshop on Computational Intelligence: Theories, Applications and Future Directions (WCI)*, pp. 1–6, IEEE, 2015.

[111] B. Prabhakaran, “Quadrotor modeling.” `https://github.com/Prabhu-369/Quadcopter_6DOF_Matlab_files/blob/master/Chapter2_Quad_6DOF.pdf`, 2015. Accessed: 2020-07-20.

[112] J. Kubalík, E. Alibekov, J. Žegklitz, and R. Babuška, “Hybrid single node genetic programming for symbolic regression,” in *Transactions on Computational Collective Intelligence XXIV*, pp. 61–82, Springer, 2016.

[113] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society, Series B*, vol. 58, pp. 267–288, 1994.

[114] E. Alibekov, J. Kubalík, and R. Babuška, “Policy derivation methods for critic-only reinforcement learning in continuous spaces,” *Engineering Applications of Artificial Intelligence*, vol. 69, pp. 178–187, 2018.

[115] E. Alibekov, J. Kubalik, and R. Babuska, “Policy derivation methods for critic-only reinforcement learning in continuous action spaces,” *IFAC-PapersOnLine*, vol. 49, no. 5, pp. 285–290, 2016.

[116] E. Alibekov, J. Kubalík, and R. Babuška, “Proxy functions for approximate reinforcement learning,” *IFAC-PapersOnLine*, vol. 52, no. 11, pp. 224–229, 2019.