IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2021.DOI

Symbolic Regression Methods for Reinforcement Learning

JIŘÍ KUBALÍK[1], ERIK DERNER[1,2], JAN ŽEGKLITZ[2], and ROBERT BABUŠKA[1,3]
[1]Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, Prague, 16000, Czech Republic
[2]Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, 16627, Czech Republic
[3]Cognitive Robotics, Delft University of Technology, Delft, 2628 CD, The Netherlands

Corresponding author: Jiří Kubalík (e-mail: jiri.kubalik@cvut.cz).

This work was supported by the European Regional Development Fund under the project Robotics for Industry 4.0 (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000470), 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. SGS19/174/OHK3/3T/13.

ABSTRACT Reinforcement learning algorithms can be used to optimally solve dynamic decision-making and control problems. With continuous-valued state and input variables, reinforcement learning algorithms must rely on function approximators to represent the value function and policy mappings. Commonly used numerical approximators, such as neural networks or basis function expansions, have two main drawbacks: they are black-box models offering no insight in the mappings learned, and they require significant trial and error tuning of their meta-parameters. In this paper, we propose a new approach to constructing smooth value functions in the form of analytic expressions by means of symbolic regression. We introduce three off-line methods for finding value functions based on a state transition model: symbolic value iteration, symbolic policy iteration, and a direct solution of the Bellman equation. The methods are illustrated on four nonlinear control problems: velocity control under friction, one-link and two-link pendulum swing-up, and magnetic manipulation. The results show that the value functions not only yield well-performing policies, but also are compact, mathematically tractable and easy to plug into other algorithms. This makes them potentially suitable for further analysis of the closed-loop system. A comparison with an alternative approach using neural networks shows that our method outperforms the neural network-based one.

INDEX TERMS Reinforcement learning, value iteration, policy iteration, symbolic regression, genetic programming, nonlinear optimal control.

I. INTRODUCTION

REINFORCEMENT learning (RL) in continuous-valued state and input spaces relies on function approximators. Various types of numerical approximators have been used to represent the value function and policy mappings: expansions with fixed or adaptive basis functions [1], [2], regression trees [3], local linear regression [4], [5], and deep neural networks [6]–[10].

The choice of a suitable approximator, in terms of its structure (number, type and distribution of the basis functions, number and size of layers in a neural network, etc.), is an ad hoc step which requires significant trial and error tuning. There are no guidelines on how to design good value function approximator and, as a consequence, a large amount of expert knowledge and haphazard tuning is required when applying RL techniques to continuous-valued problems. In addition, these approximators are black box, yielding no insight and little possibility for analysis. Moreover, approaches based on deep neural networks often suffer from the lack of reproducibility, caused in large part by nondeterminism during the training process [11]. Finally, the interpolation properties of numerical function approximators may adversely affect the control performance and result in chattering control signals and steady-state errors [12]. In practice, this makes RL inferior to alternative control design methods, despite the theoretic potential of RL to produce optimal control policies [13].

To overcome these limitations, we propose a novel approach which uses symbolic regression (SR) to automatically construct an analytic representation of the value function. Symbolic regression has been used in nonlinear data-driven modeling with quite impressive results [14]–[17]. To our best knowledge, there have been no reports in the literature on the use of symbolic regression for constructing value functions. The closest related research is the use of genetic programming for fitting already available value functions (V-functions) [18], [19], which, however, is completely different from our approach. In [18], authors use GP to find an algebraic expression that fits the sample points of the optimal value function, obtained via value iteration. Contrary to [18], in [19] they use the fact that the so called threshold policy for the solved MDP is known a priori and they use GP to find a description of this threshold policy in terms of the MDP parameters. In both cases, the task solved is to fit an algebraic expression to a set of data sampled from some known value or policy function. In our case, the task is to construct the V-function in the form of analytic expressions based on raw data sampled using the state transition model.

The paper is organized as follows. Section II describes the reinforcement learning framework considered in this work. Section III presents the proposed symbolic methods: symbolic value iteration, symbolic policy iteration, and a direct solution of the Bellman equation. In Section IV, we present the experimental results obtained with the proposed methods on four nonlinear control problems: velocity control under nonlinear friction, one-link and two-link pendulum swing-up and magnetic manipulation. Performance of the methods and other related aspects are discussed in Section V. Section VI concludes the paper.

II. RL FRAMEWORK

The dynamic system of interest is described by the state transition function

$$x_{k+1} = f(x_k, u_k) \tag{1}$$

with $x_k, x_{k+1} \in \mathcal{X} \subset \mathbb{R}^n$ and $u_k \in \mathcal{U} \subset \mathbb{R}^m$, where subscript k denotes discrete time instants. Function f is assumed to be given, but it does not have to be stated by explicit equations; it can be, for instance, a generative model given by a numerical simulation of complex differential equations. The control goal is specified through a reward function which assigns a scalar reward $r_{k+1} \in \mathbb{R}$ to each state transition from x_k to x_{k+1}:

$$r_{k+1} = \rho(x_k, u_k, x_{k+1})\,. \tag{2}$$

This function is defined by the user and typically calculates the reward based on the distance of the current state from a given reference (goal) state x_r to be attained. The state transition model and the associated reward function form the Markov decision process (MDP).

The goal of RL is to find an optimal control policy $\pi : \mathcal{X} \to \mathcal{U}$ such that in each state it selects a control action so that the cumulative discounted reward over time, called the return, is maximized:

$$R^\pi = E\Big\{\sum_{k=0}^{\infty} \gamma^k \rho\big(x_k, \pi(x_k), x_{k+1}\big)\Big\}\,. \tag{3}$$

Here $\gamma \in (0, 1)$ is a discount factor and the initial state x_0 is drawn uniformly from the state space domain $\mathcal{X}$ or its subset. The return is approximated by the value function $V^\pi : X \to \mathbb{R}$ defined as:

$$V^\pi(x) = E\Big\{\sum_{k=0}^{\infty} \gamma^k \rho\big(x_k, \pi(x_k), x_{k+1}\big)\Big|\, x_0 = x\Big\}\,. \tag{4}$$

An approximation of the optimal V-function, denoted by $\hat{V}^*(x)$, can be computed by solving the Bellman optimality equation

$$\hat{V}^*(x) = \max_{u\in\mathcal{U}} \Big[\rho\big(x, u, f(x, u)\big) + \gamma \hat{V}^*\big(f(x, u)\big)\Big]\,. \tag{5}$$

To simplify the notation, in the sequel, we drop the hat and the star superscript: $V(x)$ will therefore denote the approximately optimal V-function. Based on $V(x)$, the optimal control action in any given state x is found as the one that maximizes the right-hand side of (5):

$$\pi(x) = \operatorname*{argmax}_{u\in\mathcal{U}}\ \big[\rho\big(x, u, f(x, u)\big) + \gamma V\big(f(x, u)\big)\big] \tag{6}$$

for all $x \in \mathcal{X}$.

In this paper, we use a RL framework based on V-functions. However, the proposed methods can be applied to Q-functions as well, where the $Q^\pi(x, u)$ is the return obtained when first taking action u in state x and then following policy π until the end of the episode.

III. SOLVING BELLMAN EQUATION BY SYMBOLIC REGRESSION

We employ symbolic regression to construct an analytic approximation of the value function. Symbolic regression is a technique based on genetic programming and its purpose is to find an analytic equation describing given data. Our specific objective is to find an analytic equation for the value function that satisfies the Bellman optimality equation (5). Symbolic regression is a suitable technique for this task, as it does not rely on any prior knowledge on the form of the value function, which is generally unknown, and it has the potential to provide more compact representations than, for instance, deep neural networks or basis function expansion models. In this work, we employ two different symbolic regression methods: a variant of Single Node Genetic Programming [20]–[23] and a variant of Multi-Gene Genetic Programming [24]–[26].

A. SYMBOLIC REGRESSION

Symbolic regression methods were reported to perform better when using a linear combination of nonlinear functions found by means of genetic algorithms [27], [28]. Following this approach, we define the class of symbolic models as:

$$V(x) = \beta_0 + \sum_{i=1}^{n_f} \beta_i \varphi_i(x)\,. \tag{7}$$

The nonlinear functions $\varphi_i(x)$, called features, are constructed by means of genetic programming using a predefined set of elementary functions $\mathcal{F}$ provided by the user. These functions can be nested and the SR algorithm evolves their combinations by using standard evolutionary operations such as mutation. The complexity of the symbolic models is constrained by two user-defined parameters: the maximum number of features in the symbolic model and the maximum depth of expression trees of the features. The coefficients β are estimated by least squares, with or without regularization.

B. DATA SET

To apply symbolic regression, we first generate a set of n_x states sampled from $\mathcal{X}$:

$$X = \{x_1, \ldots, x_{n_x}\} \subset \mathcal{X},$$

and a set of n_u control inputs sampled from $\mathcal{U}$:

$$U = \{u_1, \ldots, u_{n_u}\} \subset \mathcal{U}\,.$$

The generic training data set for symbolic regression is then given by:

$$D = \{d_1, \ldots, d_{n_x}\} \tag{8}$$

with each training sample d_i being the tuple:

$$d_i = \langle x_i, x_{i,1}, r_{i,1}, \ldots, x_{i,n_u}, r_{i,n_u}\rangle$$

consisting of the state $x_i \in X$, all the next states $x_{i,j}$ obtained by applying in x_i all the control inputs $u_j \in U$ to the system model (1), and the corresponding rewards $r_{i,j} = \rho\big(x_i, u_j, f(x_i, u_j)\big)$.

In the sequel, V denotes the symbolic representation of the value function, generated by symbolic regression applied to data set D. We present three possible approaches to solving the Bellman equation by using symbolic regression.

C. DIRECT SYMBOLIC SOLUTION OF BELLMAN EQUATION

This approach (`direct`) directly evolves the symbolic value function so that it satisfies (5). The optimization criterion (fitness function) is the mean-squared error between the left-hand side and right-hand side of the Bellman equation, i.e., the Bellman error over all the training samples in D:

$$J^{\text{direct}} = \frac{1}{n_x}\sum_{i=1}^{n_x}\Big[\max_j\big(r_{i,j} + \gamma \underbrace{V(x_{i,j})}_{\text{evolved}}\big) - \underbrace{V(x_i)}_{\text{evolved}}\Big]^2 . \tag{9}$$

Unfortunately, the problem formulated in this way proved too hard to be solved by symbolic regression, as illustrated later in Sections IV-A and IV. We hypothesize that this difficulty stems from the fact that the fitness of the value function to be evolved is evaluated through the complex implicit relation in (9), which is not a standard regression problem. This means, there is no strict target defined to which the value function should be fitted. By modifying symbolic regression, the problem might be rendered feasible, but in this paper we successfully adopt an iterative approach, leading to the symbolic value iteration and symbolic policy iteration, as described below. In the above equation and equations to follow, evolved means that the approximator of the given function is constructed by means of symbolic regression (i.e. evolved using genetic programming).

D. SYMBOLIC VALUE ITERATION

In symbolic value iteration (`SVI`), the optimal value function is found iteratively, just like in standard value iteration [29]. In each iteration ℓ, the value function $V_{\ell-1}$ from the previous iteration is used to compute the target for improving the value function V_ℓ in the current iteration. For each state $x_i \in X$, the target $t_{i,\ell} \in \mathbb{R}$ is calculated by evaluating the right-hand-side of (5):

$$t_{i,\ell} = \max_{u\in U}\Big(\rho(x_i, u, f(x_i, u)) + \gamma V_{\ell-1}\big(f(x_i, u)\big)\Big)\,. \tag{10}$$

Here, the maximization is carried out over the predefined discrete control action set U. Note that virtually all control systems use discrete control actions – either as the result of digital-to-analog conversion or due to the nature of the actuator itself, e.g., a stepping motor. As the sensitivity of the control loop to discrete control action is low (approximately the reciprocal of the loop gain), most control loops tolerate even a small number of control actions, as long as the action corresponding to the desired goal state is included in the control action set. In principle, it would also be possible to use numerical or even symbolic optimization over the original continuous set $\mathcal{U}$. However, this is computationally more expensive, as the optimization problem would have to be solved n_x times at the beginning of each iteration. For this reason, we prefer the maximization over U, as stated in (10). In addition, as the next states and rewards are pre-computed and provided to the `SVI` algorithm in the data set D (8), we can replace (10) by its computationally more efficient equivalent:

$$t_{i,\ell} = \max_j\big(r_{i,j} + \gamma V_{\ell-1}(x_{i,j})\big). \tag{11}$$

Given the target $t_{i,\ell}$, an improved value function V_ℓ is constructed by applying symbolic regression with the following fitness function:

$$J_\ell^{\text{SVI}} = \frac{1}{n_x}\sum_{i=1}^{n_x}\Big[\underbrace{t_{i,\ell}}_{\text{target}} - \underbrace{V_\ell(x_i)}_{\text{evolved}}\Big]^2 . \tag{12}$$

This fitness function is again the mean-squared Bellman error. However, as opposed to (9), the above criterion (12) defines a true regression problem: the target to be fitted is fixed as it is based on $V_{\ell-1}$ from the previous iteration. In the first iteration, V_0 can be initialized either by some suitable function, or as $V_0(x) = 0$ for all $x \in \mathcal{X}$, in the absence of better initial value. In the latter case, the first target becomes the largest reward over all the next states.

In each iteration, the training data set for symbolic regression is composed as follows:

$$D_\ell^{\text{SVI}} = \{d_1, \ldots, d_{n_x}\} \text{ with } d_i = \langle x_i, t_{i,\ell}\rangle$$

i.e., each sample contains the state x_i, and the corresponding target $t_{i,\ell}$ computed by (11).

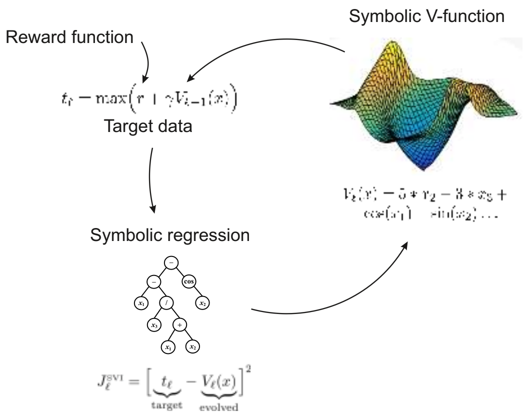
Reward function
Symbolic V-function
$t_\ell = \max\left(r + \gamma V_{\ell-1}(x)\right)$
Target data
$V_\ell(x) = 5 * x_2 - 3 * x_3 +$
$\cos(x_1) - \sin(x_2) \dots$
Symbolic regression
$J_\ell^{\text{SVI}} = \Big[\underbrace{t_\ell}_{\text{target}} - \underbrace{V_\ell(x)}_{\text{evolved}}\Big]^2$

FIGURE 1. Symbolic value iteration loop. In each iteration, the target data for symbolic regression are computed using the Bellman equation right-hand side. Symbolic regression then improves the value function and the process repeats.

The SVI procedure terminates once a predefined maximum number of iterations n_i has been reached. Other stopping criteria can be employed, such as terminating the iteration when the following condition is satisfied:

$$\max_i |V_\ell(x_i) - V_{\ell-1}(x_i)| \le \epsilon \tag{13}$$

with ϵ a user-defined convergence threshold. The resulting symbolic value iteration algorithm is given in Algorithm 1 and depicted in Figure 1. In each iteration, the symbolic regression algorithm is run for n_g generations.

Algorithm 1: Symbolic value iteration (SVI)

```
Input: training data set $D$, $n_i$
$\ell \leftarrow 0,\ V_0(x) = 0,\ \ \forall x \in \mathcal{X}$
while $\ell < n_i$ do
    $\ell \leftarrow \ell + 1$
    $\forall x_i \in X$ compute $t_{i,\ell}$ by using (11)
    $D_\ell^{\text{SVI}} \leftarrow \{d_1, \dots, d_{n_x}\}$ with $d_i = \langle x_i, t_{i,\ell}\rangle$
    $V_\ell \leftarrow \text{SymbolicRegression}(D_\ell^{\text{SVI}}, J_\ell^{\text{SVI}})$
end
$V \leftarrow V_\ell$
Output: Symbolic value function $V$
```

E. SYMBOLIC POLICY ITERATION

Also the symbolic policy iteration (SPI) algorithm iteratively improves the V-function estimate. However, rather than using $V_{\ell-1}$ to compute the target in each iteration, we derive from $V_{\ell-1}$ the currently optimal policy and plug it into the Bellman equation, so eliminating the maximum operator.

Given the value function $V_{\ell-1}$ from the previous iteration, for each state $x_i \in X$, the corresponding currently optimal control action u_i^* is computed by:

$$u_i^* = \underset{u \in U}{\text{argmax}} \Big(\rho(x_i, u, f(x_i, u)) + \gamma V_{\ell-1}\big(f(x_i, u)\big)\Big), \tag{14}$$

$\forall x_i \in X$. Again, the maximization can be carried out over the original continuous set $\mathcal{U}$, rather than the discrete set U, which would incur higher computational costs.

Now, for each state x_i and the corresponding optimal control action u_i^*, the optimal next state x_i^* and the respective reward r_i^* can be computed:

$$x_i^* = f(x_i, u_i^*), \quad r_i^* = \rho(x_i, u_i^*, x_i^*)\,. \tag{15}$$

As the next states and rewards are provided to the SPI algorithm in the data set D (8), we can replace (14) by its computationally more efficient equivalent. The index j^* of the optimal control action selected from U is found by

$$j^* = \underset{j}{\text{argmax}}\, \big(r_{i,j} + \gamma V_{\ell-1}(x_{i,j})\big)\,, \tag{16}$$

$$x_i^* = x_{i,j^*}\,, \quad r_i^* = r_{i,j^*} \tag{17}$$

with x_{i,j^*} and r_{i,j^*} selected from D. Given these samples, we can now construct the training data set for SR as follows:

$$D_\ell^{\text{SPI}} = \{d_1, \dots, d_{n_x}\} \text{ with } d_i = \langle x_i, x_i^*, r_i^*\rangle\,.$$

This means that each sample d_i contains the state x_i, the currently optimal next state x_i^* and the respective reward r_i^*. In each iteration ℓ of SPI, an improved approximation V_ℓ is sought by means of symbolic regression with the following fitness function:

$$J_\ell^{\text{SPI}} = \frac{1}{n_x} \sum_{i=1}^{n_x} \Big(\underbrace{r_i^*}_{\text{target}} - [\underbrace{V_\ell(x_i)}_{\text{evolved}} - \gamma \underbrace{V_\ell(x_i^*)}_{\text{evolved}}]\Big)^2\,. \tag{18}$$

The fitness is again the mean-squared Bellman error, where only the currently optimal reward serves as the target for the difference $V_\ell(x_i) - \gamma V_\ell(x_i^*)$, with V_ℓ evolved by SR. The resulting symbolic policy iteration algorithm is given in Algorithm 2.

Algorithm 2: Symbolic policy iteration (SPI)

```
Input: training data set $D$, $n_i$
$\ell \leftarrow 0,\ V_0(x) = 0,\ \ \forall x \in \mathcal{X}$
while $\ell < n_i$ do
    $\ell \leftarrow \ell + 1$
    $\forall x_i \in X$ select $x_i^*$ and $r_i^*$ from $D$ by (16) and (17)
    $D_\ell^{\text{SPI}} \leftarrow \{d_1, \dots, d_{n_x}\}$ with $d_i = \langle x_i, x_i^*, r_i^*\rangle$
    $V_\ell \leftarrow \text{SymbolicRegression}(D_\ell^{\text{SPI}}, J_\ell^{\text{SPI}})$
end
$V \leftarrow V_\ell$
Output: Symbolic value function $V$
```

F. PERFORMANCE MEASURES FOR EVALUATING VALUE FUNCTIONS

Note that the convergence of the iterative algorithms is not necessarily monotonic, similarly to other approximate solutions, like the fitted Q-iteration algorithm [3]. Therefore, it is not meaningful to retain only the last solution. Instead, we store the intermediate solutions from all iterations and use a posteriori analysis to select the best value function according to the performance measures described below.

Root mean squared Bellman error (BE) is calculated over all n_x state samples in the training data set D according to

$$\mathrm{BE} = \sqrt{\frac{1}{n_x}\sum_{i=1}^{n_x}\Big[\max_j\big(r_{i,j} + \gamma V(x_{i,j})\big) - V(x_i)\Big]^2}.$$

In the optimal case, the Bellman error is equal to zero.

The following two measures are calculated based on closed-loop control simulations with the state transition model (1). The simulations start from n_s different initial states in the set X_{init} ($n_s = |X_{\text{init}}|$) and run for a fixed amount of time T_{sim}. In each simulation time step, the optimal control action is computed according to the argmax policy (6).

Mean discounted return (R_γ) is calculated over the simulations from all the initial states in X_{init}:

$$R_\gamma = \frac{1}{n_s}\sum_{s=1}^{n_s}\sum_{k=0}^{T_{\text{sim}}/T_s} \gamma^k \rho\big(x_k^{(s)}, \pi(x_k^{(s)}), x_{k+1}^{(s)}\big)$$

where (s) denotes the index of the simulation, $x_0^{(s)} \in X_{\text{init}}$ and T_s is the sampling period. Larger values of R_γ indicate a better performance.

Percentage of successful simulations (S) within all n_s simulations defined as

$$S = 100\frac{n_{succ}}{n_s}\,\%\,,$$

where n_{succ} is the number of successful simulations. A simulation is considered successful if the state x stays within a predefined neighborhood of the goal state for the last T_{end} seconds of the simulation. Generally, the neighborhood $N(x_r)$ of the goal state in n-dimensional state space is defined using a neighborhood size parameter $\varepsilon \in \mathbb{R}^n$ as follows:

$$N(x_r) = \{x : |x_{r,i} - x_i| \le \varepsilon_i, \text{ for } i = 1 \dots n\}.$$

Larger values of S correspond to a better performance.

G. EXPERIMENTAL EVALUATION SCHEME

Each of the three proposed approaches (`direct`, `SVI`, and `SPI`) was implemented in two variants, one using the Single Node Genetic Programming (SNGP) algorithm and the other one using the Multi-Gene Genetic Programming (MGGP) algorithm. A detailed explanation of the SR algorithms and their parameters is beyond the scope of this paper and we refer the interested reader for more details on the implementation of SNGP to [23] and for MGGP to [26]. Both SR methods use linear combinations of original input variables. In SNGP, the weights assigned to the variables are tuned purely by means of mutation operators, while in MGGP the weights are tuned using a gradient method based on back-propagation algorithm. A specific feature of the SNGP implementation is that it adds to the raw fitness function (i.e., (9), (12), and (18)) a penalty for a wrong position of the maximum of the V-function model. In particular, the raw fitness value is multiplied by a penalty factor $(1+d)$, where d grows linearly with the distance of the actual position of the V-model's maximum and the desired position of the sought V-function, which is at the goal state of the given problem.

There are six algorithms in total to be tested: `direct-SNGP`, `direct-MGGP`, `SPI-SNGP`, `SPI-MGGP`, `SVI-SNGP` and `SVI-MGGP`. Note, however, that our goal is not to compare the two symbolic regression algorithms. Instead, we want to demonstrate that the proposed symbolic RL methods are general and can be implemented by using more than one specific symbolic regression algorithm.

Each of the algorithms was run $n_r = 30$ times with the same parameters, but with a different randomization seed. Each run delivers three winning V-functions, which are the best ones with respect to R_γ, BE and S, respectively. Statistics such as the median, min, and max calculated over the set of n_r respective winner V-functions are used as performance measures of the particular method (`SVI`, `SPI` and `direct`) and the SR algorithm (SNGP, MGGP). For instance, the median of S is calculated as

$$\underset{r=1..n_r}{\mathrm{med}}\Big(\max_{i=1..n_i}(\mathrm{S}_{r,i})\Big) \tag{19}$$

where $\mathrm{S}_{r,i}$ denotes the percentage of successful simulations in iteration i of run r. For the `direct` method, the above maximum is calculated over all generations of the SR run.

For comparison purposes, we have calculated a baseline solution, which is a numerical V-function approximation calculated by the fuzzy V-iteration algorithm [13] with triangular basis functions.

IV. EXPERIMENTS

In this section, experiments are reported for four non-linear control problems: friction compensation, 1-DOF and 2-DOF pendulum swing-up, and magnetic manipulation. In the end of this section, experiments with neural network-based approximators (NN approximators) used in the place of the symbolic models in the `SVI` method are described.

While the chosen test problems have low-dimensional input and state spaces, they are representative of challenging control problems as none of them can be solved by linear control methods. Moreover, they are challenging even for neural network-based reinforcement learning methods as shown in our experiments as well as in the literature, see for example [30].

The parameters of the experiments are listed in Table 8. The symbolic regression methods worked with the following setting. The number of iterations, n_i, was 50 and 30 for `SVI` and `SPI`, respectively. It was smaller for `SPI` as this method converges faster and needs fewer iterations. The `direct` method ran for 50 000 generations (the method does not iterate in the sense the SPI and SVI methods do). We used the same set of elementary functions, $\mathcal{F} = \{*, +, -,$ square, cube, bent identity[1]$\}$, for all methods. The maximum number of features in the model was set to 10 and the maximum depth of features was set to 7. The remaining parameters of the experiment are listed in Table 8.

A. FRICTION COMPENSATION

We start by illustrating the working of the proposed methods on a practically relevant first-order, nonlinear motion-control problem. Many applications require high-precision position and velocity control, which is often hampered by the presence of friction. Without proper nonlinear compensation, friction causes significant tracking errors, stick-slip motion and limit cycles. To address these problems, we design a nonlinear velocity controller for a DC motor with friction by using the proposed symbolic methods.

The continuous-time system dynamics are given by:

$$I\dot{v}(t) + (b + \frac{K^2}{R})v(t) + F_c\big(v(t), u(t), c\big) = \frac{K}{R}u(t) \quad (20)$$

with $v(t)$ and $\dot{v}(t)$ the angular velocity and acceleration, respectively. The angular velocity varies in the interval $[-10, 10]$ rad·s^{-1}. The control input $u \in [-4, 4]$ V is the voltage applied to the DC motor and the parameters of the system are: moment of inertia $I = 1.8 \times 10^{-4}$ kg·m^2, viscous friction coefficient $b = 1.9 \times 10^{-5}$ N·m·s·rad^{-1}, motor constant $K = 0.0536$ N·m·A^{-1}, armature resistance $R = 9.5\,\Omega$, and Coulomb friction coefficient $c = 8.5 \times 10^{-3}$ N·m.

The Coulomb friction force F_c is modeled as [31]:

$$F_c\big(v(t), u(t), c\big) = \begin{cases} c & \text{if } v(t) > 0 \text{ or } v(t) = 0 \text{ and } u(t) > c\frac{R}{K} \\ -c & \text{if } v(t) < 0 \text{ or } v(t) = 0 \text{ and } u(t) < -c\frac{R}{K} \\ \frac{K}{R}u(t) & \text{if } v(t) = 0 \text{ and } \left|\frac{K}{R}u(t)\right| \leq c \end{cases}$$

The discrete-time transitions are obtained by numerically integrating the continuous-time dynamics using the fourth-order Runge-Kutta method and a sampling period $T_s = 0.001$ s. The state is the velocity, $x = v$, and the reward function is defined as:

$$r_{k+1} = \rho(x_k, u_k, x_{k+1}) = -\sqrt{|x_r - x_k|} \quad (21)$$

with $x_r = 7$ rad·s^{-1} the desired velocity (goal state).

In each of the 30 runs, we selected the best V-function with respect to S. Figure 2 shows the median values of S calculated for the V-functions over all 30 runs according to (19). The SVI method is consistently the best one, followed by SPI and direct.

The performance measures R$_\gamma$, BE and S are listed in Table 1. For the S measure, the first two numbers in the square brackets are the minimum and maximum value and the number in parentheses is the frequency of the maximum value. Interestingly, we found that low BE does not necessarily correlate with high performance of the V-function in the control task.

[1]https://en.wikipedia.org/wiki/Bent_function

TABLE 1. Performance of the symbolic methods on the friction compensation problem. The performance of the baseline V-function is R$_\gamma$ = -42.158, BE $= 1.7 \times 10^{-5}$, S = 100 %.

SNGP	direct	SPI	SVI
R$_\gamma$ [–]	−48.184	−42.563	−42.339
BE [–]	0.301	0.0212	2.571
S [%]	0	100	100
	[0, 100 (11)]	[0, 100 (22)]	[100, 100 (30)]
MGGP	**direct**	**SPI**	**SVI**
R$_\gamma$ [–]	−48.184	−42.565	−42.274
BE [–]	0.719	1.552	2.619
S [%]	0	100	100
	[0, 0 (30)]	[0, 100 (28)]	[100, 100 (30)]

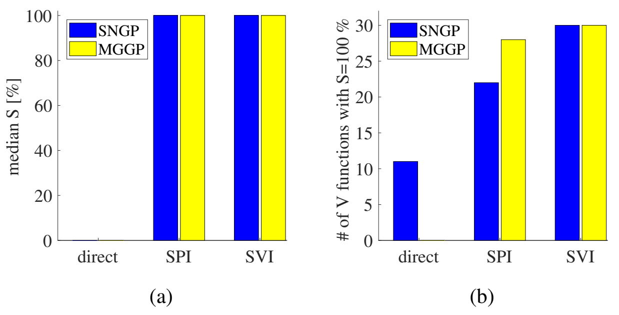

FIGURE 2. Performance on the friction compensation problem: (a) median percentage of successful simulations S, (b) the number of runs in which a V-function with S=100 % was found.

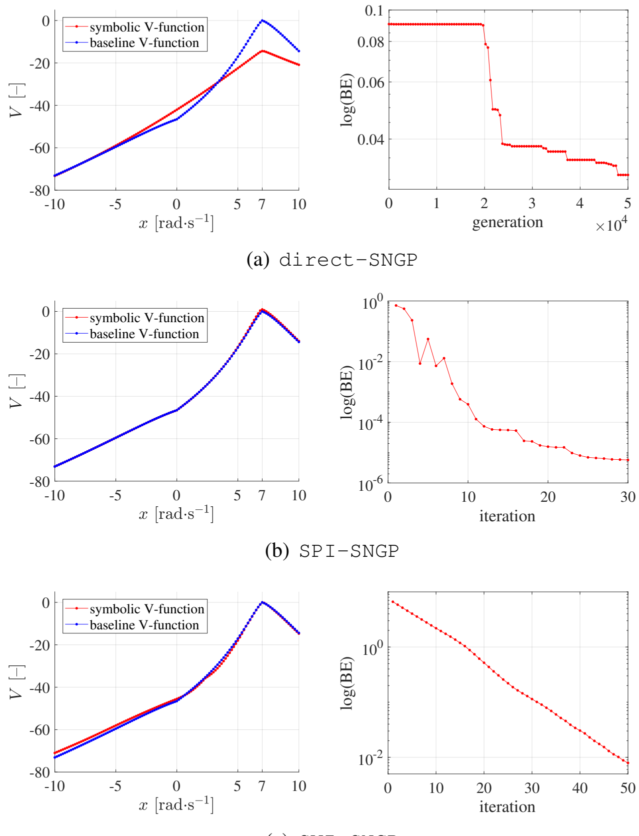

FIGURE 3. Examples of typical well-performing V-functions found for the friction compensation problem. Left: the symbolic V-function compared to the baseline. Right: the Bellman error.

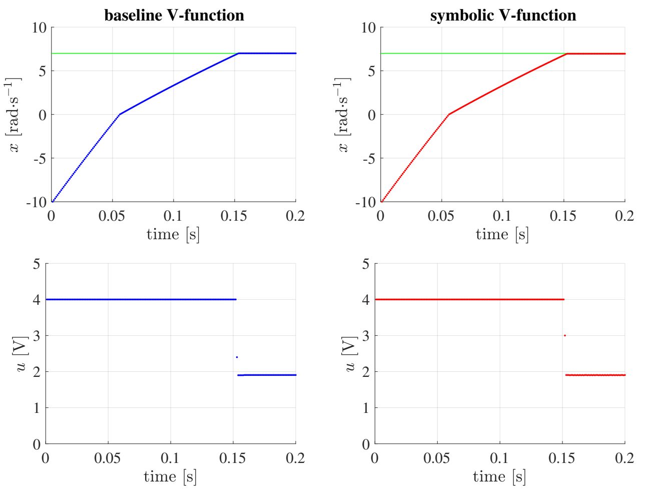

FIGURE 4. Simulations of the friction compensation problem with the baseline V-function (left) and the symbolic V-function (right) presented in Figure 3b). The upper plots show the state trajectory from $x_0 = -10$ rad·s^{-1}. The lower plots show the corresponding control inputs. Only the first 0.2 s of the simulation are shown as the variables remain constant afterwards.

Figure 3 shows examples of well-performing symbolic V-functions found through symbolic regression, compared to a baseline V-function calculated using the numerical approximator [13]. A closed-loop simulation is presented in Figure 4. Both the symbolic and baseline V-function yield optimal performance. The proposed symbolic methods reliably find well-performing V-functions for the friction compensation problem. Interestingly, even the `direct` approach can solve this problem when using the SNGP algorithm. However, it finds a well-performing V-function with respect to S only in approximately one third of the runs.

B. 1-DOF PENDULUM SWING-UP

The inverted pendulum (denoted as 1DOF) consists of a weight of mass m attached to an actuated link that rotates in a vertical plane. The available torque is insufficient to push the pendulum up in a single rotation from many initial states. Instead, from certain states (e.g., when the pendulum is pointing down), it needs to be swung back and forth to gather energy, prior to being pushed up and stabilized. The continuous-time model of the pendulum dynamics is:

$$\ddot{\alpha} = \frac{1}{I}\cdot\left[mgl\sin(\alpha) - b\dot{\alpha} - \frac{K^2}{R}\dot{\alpha} + \frac{K}{R}u\right] \qquad (22)$$

where $I = 1.91 \times 10^{-4}$ kg·m^2, $m = 0.055$ kg, $g = 9.81$ m·s^{-2}, $l = 0.042$ m, $b = 3 \times 10^{-6}$ N·m·s·rad^{-1}, $K = 0.0536$ N·m·A^{-1}, $R = 9.5\,\Omega$. The angle α varies in the interval $[0, 2\pi]$ rad, with $\alpha = \pi$ rad pointing up, and 'wraps around' so that e.g., a rotation of $5\pi/2$ rad corresponds to $\alpha = \pi/2$ rad. The state vector is $x = [\alpha, \dot{\alpha}]^\top$. The sampling period is $T_s = 0.05$ s, and the discrete-time transitions are obtained by numerically integrating the continuous-time dynamics (22) by using the fourth-order Runge-Kutta method. The control input u is limited to $[-2, 2]$ V, which is insufficient to push the pendulum up in one go.

TABLE 2. Performance of the symbolic methods on the 1DOF problem. The performance of the baseline V-function is $\mathrm{R}_\gamma = -9.346$, $\mathrm{BE} = 0.0174$, S = 100 %.

SNGP	`direct`	`SPI`	`SVI`
R$_\gamma$ [–]	−26.083	−10.187	−10.013
BE [–]	0.478	0.242	0.615
S [%]	0	100	100
	[0, 0 (30)]	[81.3, 100 (27)]	[93.8, 100 (28)]
MGGP	`direct`	`SPI`	`SVI`
R$_\gamma$ [–]	−26.083	−10.487	−9.917
BE [–]	0.776	0.797	0.623
S [%]	0	100	100
	[0, 6.25 (2)]	[0, 100 (17)]	[0, 100 (29)]

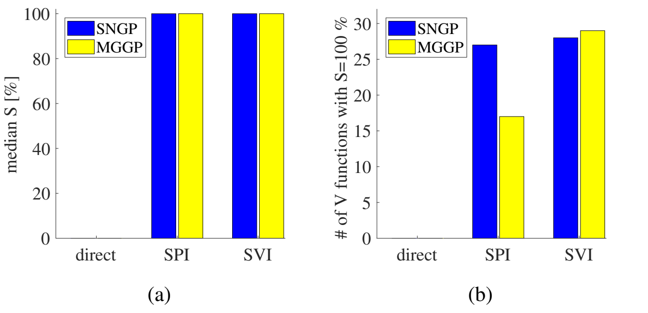

FIGURE 5. Performance on the 1DOF problem: (a) median S, (b) the number of runs in which a V-function with S=100 % was found.

The control goal is to stabilize the pendulum in an unstable equilibrium defined by the goal state $x_r = [\pi, 0]^\top$, which is expressed by the following reward function:

$$\rho(x, u, f(x, u)) = -|x_r - x|^\top Q \qquad (23)$$

with $Q = [0.5, 0]^\top$ a weighting vector to adjust the relative importance of the angle and angular velocity.

The statistical results obtained from 30 independent runs are presented in Figure 5 and Table 2.

Figure 5 shows that the `SVI` and `SPI` methods achieve comparable performance, while the `direct` method fails.

An example of a well-performing symbolic V-function found through symbolic regression, compared to a baseline V-function calculated using the numerical approximator [13], is shown in Figure 6.

The symbolic V-function is smoother than the numerical baseline, which can be seen on the level curves and on the state trajectory. The difference is particularly notable in the vicinity of the goal state, which is a significant advantage of the proposed method.

A simulation with the symbolic V-function, as well as an experiment with the real system [5], is presented in Figure 7. The trajectory of the control signal u on the real system shows the typical bang-bang nature of optimal control, which illustrates that symbolic regression found a near optimal value function.

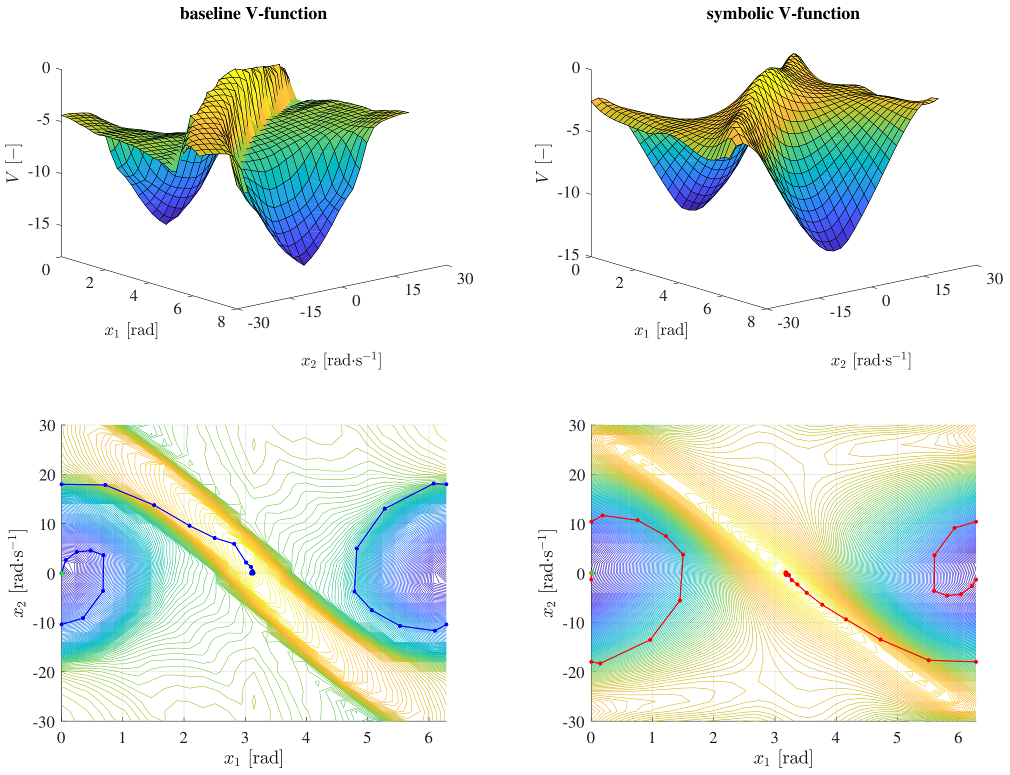

FIGURE 6. Baseline and symbolic V-function produced by the `SVI-SNGP` method on the 1DOF problem. The symbolic V-function is smoother than the numerical baseline V-function, which can be seen on the level curves and on the state trajectory, in particular near the goal state.

The symbolic V-function depicted in Figure 6, constructed by the `SVI-SNGP` method, has the following form:

$$\begin{aligned}
&V(x) = 1.7 \times 10^{-5}(10x_2 - 12x_1 + 47)(4.3 \times 10^{-2}x_2 - 3.5x_1 + 11)^3 \\
&- 7.1 \times 10^{-4}x_2 - 4.6x_1 - 8.2 \times 10^{-6}(4.3 \times 10^{-2}x_2 - 3.5x_1 \\
&+ 11)^3(0.2x_1 + 0.3x_2 - 0.5)^3 - 9.8 \times 10^{-3}(0.4x_1 + 0.1x_2 - 1.1)^6 \\
&+ 11(0.1x_1 - 1.5)^3 + 11((0.6x_1 + 6.3 \times 10^{-2}x_2 - 1.7)^2 + 1)^{0.5} \\
&+ 8.7 \times 10^{-6}((10x_2 - 12x_1 + 47)^2(4.3 \times 10^{-2}x_2 - 3.5x_1 + 11)^6 \\
&+ 1)^{0.5} + 0.3((1.1x_1 + 0.4x_2 - 3.3)^2 + 1)^{0.5} + (3.9 \times 10^{-3}(4.3\times \\
&\times 10^{-2}x_2 - 3.5x_1 + 11)^2(0.2x_1 + 0.3x_2 - 0.5)^2 + 1)^{0.5} + 6.5\times \\
&\times 10^{-5}((1.2x_1 + 14x_2 - 10)^2(9.1 \times 10^{-2}x_2 - 2.9x_1 + 0.5((9.1\times \\
&\times 10^{-2}x_2 - 2.9x_1 + 8.3)^2 + 1)^{0.5} + 7.8)^2 + 1)^{0.5} - 5.5 \times 10^{-2}(4.3\times \\
&\times 10^{-2}x_2 - 3.5x_1 + 11)(0.2x_1 + 0.3x_2 - 0.5) - 1.7((3.6x_1 + 0.4x_2- \\
&- 11)^2 + 1)^{0.5} - 2((x_1 - 3.1)^2 + 1)^{0.5} - 1.3 \times 10^{-4}(1.2x_1 + 14x_2- \\
&- 10)(9.1 \times 10^{-2}x_2 - 2.9x_1 + 0.5((9.1 \times 10^{-2}x_2 - 2.9x_1 + 8.3)^2+ \\
&+ 1)^{0.5} + 7.8) + 23 \;.
\end{aligned} \tag{24}$$

The example shows that symbolic V-functions are compact, analytically tractable and easy to plug into other algorithms. The number of parameters in the example is 100.

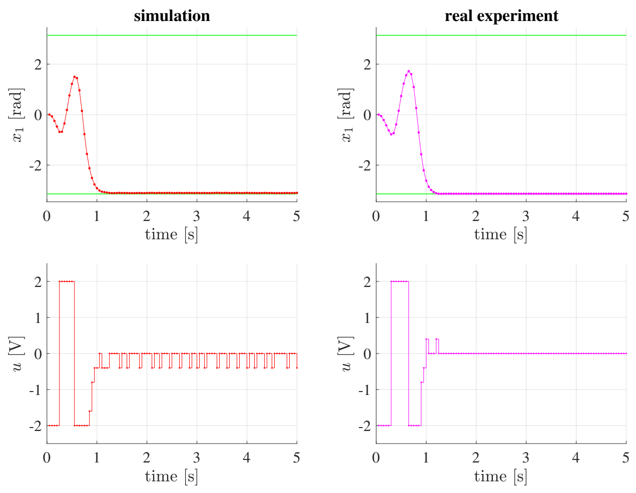

FIGURE 7. An example of a well-performing symbolic V-function found with the `SVI-SNGP` method on the 1DOF problem, used in a simulation (left) and on the real system (right). The performance of the SVI method is near-optimal even in the real experiment.

C. 2-DOF SWING-UP

The double pendulum (denoted as 2DOF) is described by the following continuous-time fourth-order nonlinear model:

$$M(\alpha)\ddot{\alpha} + C(\alpha, \dot{\alpha})\alpha + G(\alpha) = u \tag{25}$$

with $\alpha = [\alpha_1, \alpha_2]^\top$ the angular positions of the two links, $u = [u_1, u_2]^\top$ the control input, which are the torques of the two motors, $M(\alpha)$ the mass matrix, $C(\alpha, \dot{\alpha})$ the Coriolis and centrifugal forces matrix and $G(\alpha)$ the gravitational forces vector. The state vector x contains the angles and angular velocities and is defined by $x = [\alpha_1, \dot{\alpha_1}, \alpha_2, \dot{\alpha_2}]^\top$. The angles α_1, α_2 vary in the interval $[-\pi, \pi)$ rad and wrap around. The angular velocities $\dot{\alpha_1}$, $\dot{\alpha_2}$ are restricted to the interval $[-2\pi, 2\pi)$ rad·s^{-1} using saturation. Matrices $M(\alpha)$,

TABLE 3. Double pendulum parameters.

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	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^{-1}
Gravitational acceleration	g	9.8	m·s^{-2}

TABLE 4. Results obtained on the 2DOF problem. The performance of the baseline V-function is $\mathrm{R}_\gamma = -80.884$, $\mathrm{BE} = 8 \times 10^{-6}$, S = 23 %.

SNGP	`direct`	`SPI`	`SVI`
R_γ [–]	−89.243	−85.607	−81.817
BE [–]	4.23	2.00	5.79
S [%]	15.4	38.5	53.8
	[7.7, 23.1 (14)]	[0, 100 (4)]	[7.7, 100 (4)]
MGGP	`direct`	`SPI`	`SVI`
R_γ [–]	−84.739	−84.116	−82.662
BE [–]	5.19	1.98	3.29
S [%]	23.1	26.9	69.2
	[0, 30.8 (1)]	[0, 100 (5)]	[7.7, 100 (2)]

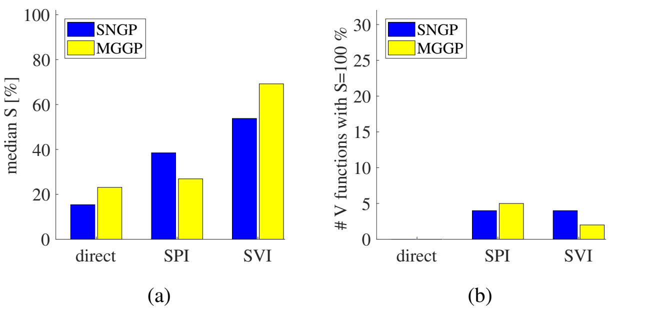

FIGURE 8. Performance on the 2DOF problem: a) median S, b) the number of runs, out of 30, in which a V-function achieving S=100 % was found.

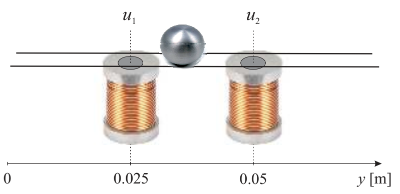

FIGURE 9. Magman schematic.

$C(\alpha, \dot\alpha)$ and $G(\alpha)$ are defined by:

$$M(\alpha) = \begin{bmatrix} P_1 + P_2 + 2P_3\ \cos(\alpha_2) & P_2 + P_3\ \cos(\alpha_2) \\ P_2 + P_3\ \cos(\alpha_2) & P_2 \end{bmatrix},$$

$$C(\alpha, \dot\alpha) = \begin{bmatrix} b_1 - P_3\dot{\alpha_2}\sin(\alpha_2) & -P_3(\dot{\alpha_1} + \dot{\alpha_2})\sin(\alpha_2) \\ P_3\dot{\alpha_1}\sin(\alpha 2) & b_2 \end{bmatrix},$$

$$G(\alpha) = \begin{bmatrix} -F_1\sin(\alpha_1) - F_2\sin(\alpha_1 + \alpha_2) \\ -F_2\sin(\alpha_1 + \alpha_2) \end{bmatrix}$$

with $P_1 = m_1c_1^2 + m_2l_1^2 + I_1$, $P_2 = m_2c_2^2 + I_2$, $P_3 = m_2l_1c_2$, $F_1 = (m_1c_1 + m_2l_2)g$ and $F_2 = m_2c_2g$. The meaning and values of the system parameters are given in Table 3. The transition function $f(x, u)$ is obtained by numerically integrating (25) between discrete time samples using the fourth-order Runge-Kutta method with the sampling period $T_s = 0.01$ s.

The control goal is to stabilize the two links in the upper equilibrium, which is expressed by the following quadratic reward function:

$$\rho(x, u, f(x, u)) = -(x_r - x) \odot (x_r - x)Q \qquad (26)$$

with the desired goal state $x_r = [0, 0, 0, 0]$ and $Q = [1, 0, 1.2, 0]^\top$ a weighting vector to specify the relative importance of the angles and angular velocities.

The statistical results obtained from 30 independent runs are presented in Figure 8 and Table 4.

D. MAGNETIC MANIPULATION

The magnetic manipulation (denoted as Magman) has several advantages compared to traditional robotic manipulation approaches. 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.

A schematic of a magnetic manipulation setup [32] with two coils is shown in Figure 9.

The two electromagnets are positioned at 0.025 m and 0.05 m. The current through the electromagnet coils is controlled to dynamically shape the magnetic field above the magnets and so to position a steel ball, which freely rolls on a rail, accurately and quickly to the desired set point.

The horizontal acceleration of the ball is given by:

$$\ddot y = -\frac{b}{m}\dot y + \frac{1}{m}\sum_{i=1}^{2} g(y, i)\, u_i \qquad (27)$$

with

$$g(y, i) = \frac{-c_1\ (y - 0.025i)}{\left((y - 0.025i)^2 + c_2\right)^3}. \qquad (28)$$

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 the ball mass, and b the viscous friction of the ball on the rail. The model parameters are listed in Table 5.

State x is given by the position and velocity of the ball. The control goal is to stabilize the ball at the goal state $x_r = [0.01, 0]$. The reward function is defined by:

$$\rho(x, u, f(x, u)) = -(x_r - x) \odot (x_r - x)Q \qquad (29)$$

TABLE 5. Magnetic manipulation system parameters.

Model parameter	Symbol	Value	Unit
Ball mass	m	3.200×10^{-2}	kg
Viscous damping	b	1.613×10^{-2}	N·s·m^{-1}
Empirical parameter	c_1	5.520×10^{-10}	N·m^5·A^{-1}
Empirical parameter	c_2	1.750×10^{-4}	m^2

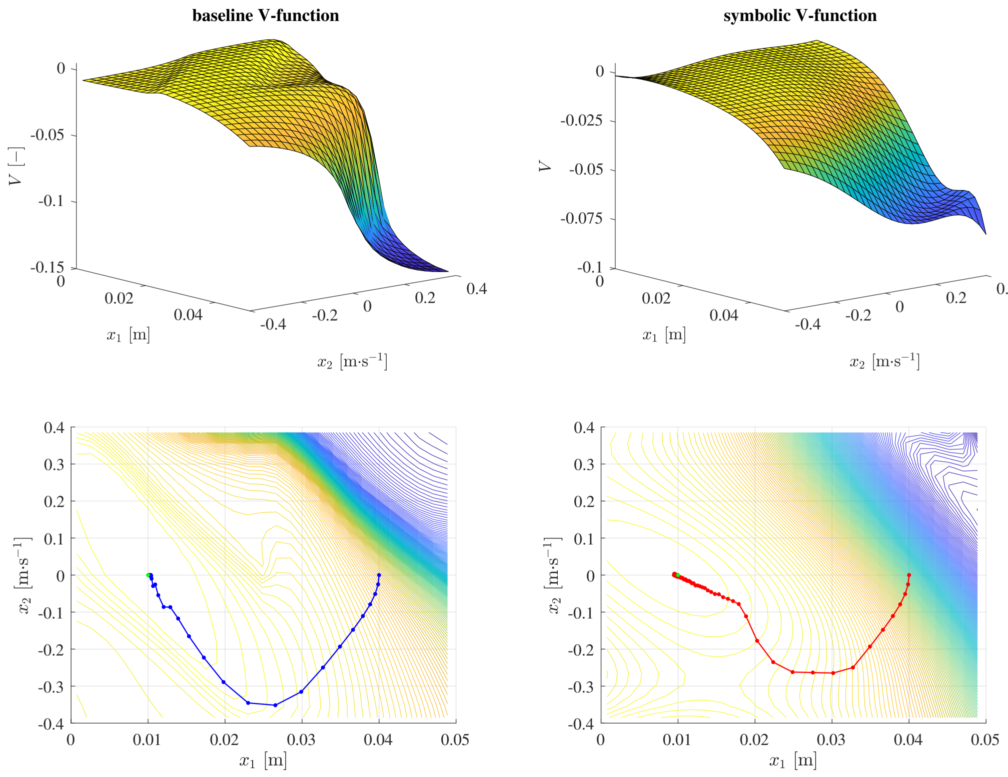

FIGURE 10. Baseline and symbolic V-function produced by the SVI-SNGP method on the Magman problem. The symbolic V-function is smoother than the numerical baseline V-function and it performs the control task well. However, the way of approaching the goal state by using the symbolic V-function is inferior to the trajectory generated with the baseline V-function. This example illustrates the trade-off between the complexity of the V-function and the ability of the algorithm to find those intricate details on the V-function surface that matter for the performance.

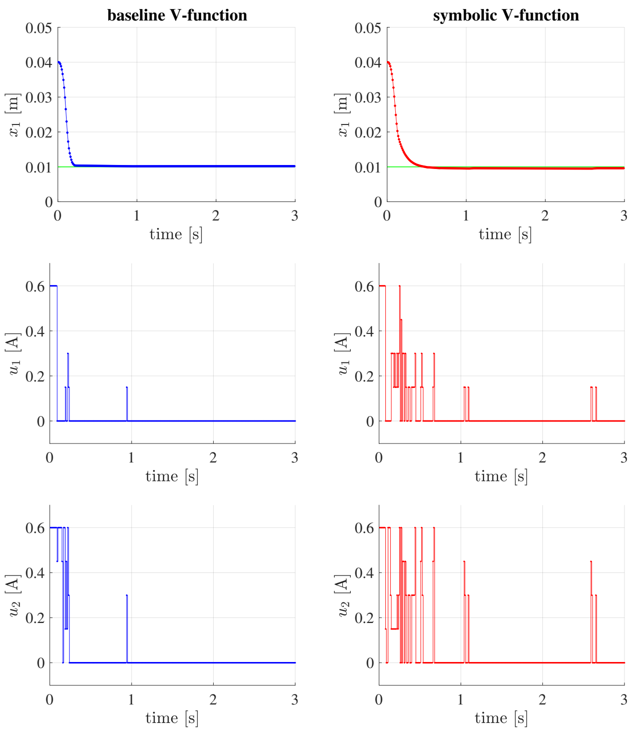

FIGURE 11. Simulations with the baseline V-function (left) and the symbolic V-function (right) found with the SVI-SNGP method on the Magman problem.

TABLE 6. Results obtained on the Magman problem. The performance of the baseline V-function is $\mathrm{R}_\gamma = -0.0097$, $\mathrm{BE} = 1.87 \times 10^{-4}$, S = 100 %.

SNGP	direct	SPI	SVI
R_γ	-9.917	-0.010	-0.011
BE	0.623	0.084	0.00298
S	100	100	100
	[7.14, 100 (27)]	[100, 100 (30)]	[100, 100 (30)]
MGGP	**direct**	**SPI**	**SVI**
R_γ	-0.164	-0.010	-0.169
BE	0.004	15.74	0.061
S	14.3	100	0
	[0, 100 (5)]	[0, 100 (16)]	[0, 100 (4)]

with the vector $Q = [5, 0]^\top$ specifying the relative importance of the ball's position and velocity.

The statistical results obtained from 30 independent runs are presented in Figure 12 and Table 6. An example of a well-performing symbolic V-function found through symbolic regression, compared to the baseline V-function calculated using the numerical approximator [13], is shown in Figure 10. The symbolic V-function is smoother than the one of the numerical approximator. It has only 77 parameters compared to 729 parameters of the numerical approximator.

The course of the state and the control actions within a simulation with the symbolic and baseline V-functions from Figure 10 is presented in Figure 11. The symbolic one performs well, however, the way it approaches the goal state

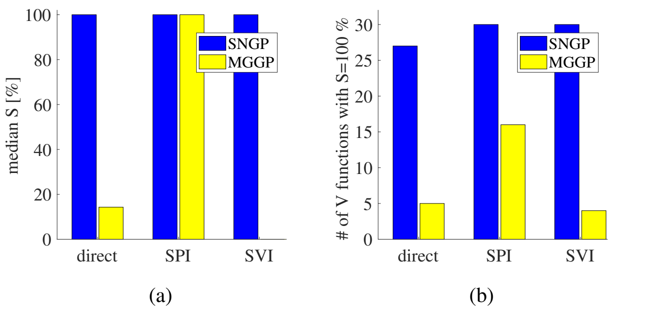

FIGURE 12. Performance on the Magman problem: a) median S, b) the number of runs, out of 30, in which a V-function achieving S=100 % was found.

is suboptimal. This result demonstrates the trade-off between the complexity and the smoothness of the V-function.

E. VALUE ITERATION WITH NEURAL NETWORK APPROXIMATOR

Other types of V-function representation than the symbolic models and the numerical approximators can be used. Here, we show an analysis of the neural network-based approximators when plugged into the V-iteration method. We used the Matlab implementation of the feed-forward neural network, the `fitnet` function.

Neural networks have many (hyper)parameters that must be tuned for a particular problem to be solved. To provide a reasonably fair analysis, we tested the neural networks with various topologies and settings of two learning control parameters. Four topologies with two hidden layers having 8, 12, 20, and 42 neurons in each of them were tested. These topologies represent models of different complexity with roughly 100, 200, 500, and 2000 parameters to be tuned (we use W to denote the model's complexity). The other two control parameters are the maximum number of training epochs before the training is stopped, i.e., $E \in \{1000, 2000, 5000, 10000\}$ and the maximum number of validation checks before the training is stopped, i.e., $F \in \{20, 50\}$. In each `SVI` iteration, the training data set is split into training and validation sets and the neural network training stops when there was no improvement in the validation performance for the last F training epochs. The hidden layers' neurons used the hyperbolic tangent sigmoid activation function, which is a sigmoid function returning output values in the interval $(-1, 1)$. The output neuron calculated the pure linear activation function. The gradient descent with momentum and adaptive learning rate backpropagation, the `traingdx` method, was used to train the network's weights.

The NN approximators were tested with all $|W|\times|E|\times|F|$ configurations on all test problems and the results are summarized in Table 7. Only the S performance metric, as the most important one, is used to assess the NN approximators' performance. Results for the best performing configuration $[E,\ F]$ in terms of the number of runs that produced a 100 % correctly working model are presented for each NN approximator's complexity W.

V. DISCUSSION

A. PERFORMANCE OF METHODS

The `SPI` and `SVI` methods are able to produce V-functions allowing to successfully solve the underlying control task (indicated by the maximum value of S equal to 100 %) for all the problems tested. They also clearly outperform the `direct` method. The best performance was observed on the 1DOF problem (`SVI-SNGP` and `SVI-MGGP` generate 28 and 29 models with S=100 %, respectively) and the Magman (both `SPI-SNGP` and `SVI-SNGP` generate 30 models with S=100 %). However, we observe the MGGP method is worse than the SNGP one, particularly when used in `SPI` on both 1DOF and Magman problems and in `SVI` on Magman. This can be attributed to the fact that MGGP does not penalize for a misplacement of the maximum of the V-function model. Note that a wrong position of the V-function's maximum might prevent reaching the goal state in the simulations.

The performance of all methods was significantly worse on the 2DOF problem where the SR methods found a model that works perfectly in simulations (i.e., S=100%) in 2 to 5 runs out of 30. The baseline numerical V-function exhibited even worse performance as only 3 out of all simulations started from 13 initial states successfully ended up in the neighborhood of the goal state (i.e., S=23%). That is a much smaller success rate than the median success rate obtained with the SR methods. This can be attributed to the rather sparse coverage of the state space since the approximator was constructed using a regular grid of $11 \times 11 \times 11 \times 11$ triangular basis functions. Note, however, that sampling the state space by using a coarse grid is often necessary for higher-dimensional problems. The number of samples grows exponentially with the state space dimension, leading to prohibitive memory and computational demands for fine grids. The results show that the SR methods are able to generate reliable models even if only sparsely sampled training data are available.

Interestingly, the `direct` method implemented with SNGP was able to find several perfect V-functions with respect to S on the Magman. On the contrary, it completely failed to find such a V-function on the 2DOF and even on the 1DOF problem. We observed that although the 1DOF and Magman systems both had 2D state-space, the 1DOF problem is harder for the symbolic methods in the sense that the V-function has to be very precise at certain regions of the state space in order to allow for successful closed-loop control. This is not the case in the Magman problem, where V-functions that only roughly approximate the optimal V-function can perform well.

Overall, the two symbolic regression methods, SNGP and MGGP, performed well, although SNGP was better on the 1DOF and Magman problem. Note, however, that a thorough comparison of symbolic regression methods was not a primary goal of the experiments. We have also not tuned the control parameters of the algorithms at all and it is quite likely that if the parameters of the algorithms were optimized their performance would improve.

TABLE 7. Performance of neural network-based V-function approximators. The S performance metric is presented for the best neural network configuration $[E, F]$ per approximator's complexity W. For easier orientation, the results obtained with SNGP and MGGP are copied from Table 2, Table 4, and Table 6.

			S: 1DOF				S: 2DOF				S: Magman
SNGP			100, [94, 100 (28)]	SNGP			54, [8, 100 (4)]	SNGP			100, [100, 100 (30)]
MGGP			100, [0, 100 (29)]	MGGP			69, [8, 100 (2)]	MGGP			0, [0, 100 (4)]
Neural network				Neural network				Neural network			
W	E	F		W	E	F		W	E	F	
100	1000	50	100, [0, 100 (25)]	100	2000	50	0, [0, 100 (2)]	100	2000	50	79, [0, 100 (11)]
200	1000	50	100, [63, 100 (27)]	200	10000	50	0, [0, 100 (3)]	200	5000	50	79, [0, 100 (8)]
500	1000	50	100, [13, 100 (26)]	500	10000	50	0, [0, 100 (3)]	500	5000	20	86, [0, 100 (5)]
2000	2000	50	100, [0, 100 (28)]	2000	10000	50	0, [0, 100 (3)]	2000	5000	50	46, [0, 100 (2)]

TABLE 8. Experiment parameters.

Problem	Friction compensation	1-DOF	2-DOF	Magman
n	1	2	4	2
$\mathcal{X}$	$[-10, 10]$	$[0, 2\pi] \times [-30, 30]$	$[-\pi, \pi] \times [-2\pi, 2\pi] \times [-\pi, \pi] \times [-2\pi, 2\pi]$	$[0, 0.05] \times [-0.4, 0.4]$
x_r	7	$[\pi\ 0]$	[0 0 0 0]	[0.01 0]
m	1	1	2	2
$\mathcal{U}$	$[-4, 4]$	$[-2, 2]$	$[-3, 3] \times [-1, 1]$	$[0, 0.6] \times [0, 0.6]$
n_u	41	11	9	25
U	$\{-4, -3.8, \dots, 4\}$	$\{-2, -1.6, \dots, 2\}$	$\{-3, 0, 3\} \times \{-1, 0, 1\}$	$\{0, 0.15, 0.3, 0.45, 0.6\} \times \{0, 0.15, 0.3, 0.45, 0.6\}$
n_x	121	961	14641	729
γ	0.95	0.95	0.95	0.95
n_s	7	16	13	14
T_s [s]	0.001	0.05	0.01	0.01
T_{sim} [s]	1	5	10	3
ε	0.05	$[0.1,\ 1]^\top$	$[0.1,\ 1,\ 0.1,\ 1]^\top$	$[0.001,\ 1]^\top$
T_{end} [s]	0.01	2	2	1

B. COMPARISON WITH NEURAL NETWORK-BASED APPROXIMATORS

NN approximators worked best on the 1DOF problem, where they achieved a performance comparable to the SR methods. Here, larger networks worked slightly better than the smaller ones as they exhibited more stable performance across all possible configurations tested.

On the 2DOF problem, the most difficult one, the best NN approximator was able to produce a model with S=100 % in 3 runs out of 30. Again, larger networks learning for a higher number of epochs performed slightly better than the smaller ones. However, the SR methods are significantly better than the neural networks in the median S value, which is 54 % for SNGP and 69 % for MGGP, compared to 0 % for the neural networks. This means that the SR methods were able to deliver models working correctly for more than 50 % of the n_s initial states in at least 15 runs. NN approximators are much worse in this respect as only up to 10 runs ended up with non-zero (mostly much smaller than 50 %) S metric models. Interestingly, not even significantly larger network topology with $W = 2000$ led to an improved performance.

On the Magman problem, the overall best neural network performance was observed for $W = 100$. Here, the neural networks worked comparably to the MGGP SR method as both methods were much worse than the SNGP in the median S value as well as in the number of runs producing 100 % correct model. Interestingly, the larger NN topologies only worsened the performance. Likewise the MGGP, the NN approximator does not penalize models for an incorrectly positioned maximum. In fact, it is even impossible to incorporate this kind of desired model's property into the gradient-based learning algorithm.

To conclude, this analysis shows that the SR methods, especially SNGP, outperform the NN approximators. All the more so, because the parameters of the NN approximators were tuned for each individual problem, while the SR methods were run with the same setting on all test problems.

C. NUMBER OF PARAMETERS

One of the advantages of the proposed symbolic methods is the compactness of the value functions, which can be demonstrated, for instance, on the 1DOF problem. The symbolic value function found by using the `SVI-SNGP` method (Figure 6, right) has 100 free parameters, while the baseline numerically approximated value function has 961 free parameters.

D. EASE OF USE

The proposed methods do not require a large amount of expert knowledge in order to be applied to the particular problem at hand. The main parameters of the GP method are the set of elementary functions that are sufficient for creating diverse non-linear features and the maximum complexity of the models. It is not difficult to choose these parameters. In this work, we used a rather small function set consisting of three simple arithmetical operators $\{*, +, -\}$ and three univariate functions {square, cube, bent identity}. Depending on the problem, the function set can be arbitrarily enriched, for example by adding trigonometric functions. Furthermore, it is easy to use additional prior knowledge and constraints in SR methods in order to generate models with desired properties, see [33], [34].

The complexity of the models evolved is defined in terms of the maximum number of features and their maximum depth. We used the maximum feature's depth of 7 and the maximum number of features of 10. However, a heuristic guideline for setting up these parameters is that if more complex models are needed then this could be effectively achieved by increasing the number of features rather than by increasing the maximum feature's depth. There are two reasons for that, (1) the maximum depth of 7 is sufficient to represent complex non-linear features and (2) keeping rather small feature's depth prevents from uncontrolled bloat of evolved expressions that is a severe issue in genetic programming [35].

Although the aforementioned heuristics can guide the setting of the parameters to achieve better results, the methods' performance is not particularly sensitive to the precise choice of these parameters. Neither are the proposed methods dependent on any particular symbolic regression algorithm. This was demonstrated while testing the methods with two different GP algorithms that are conceptually very different and also have a different set of control parameters. Both GP algorithms were run with rather standard parameter settings inspired by works using these algorithms for other problems. No parameter tuning was performed in this work. In principle, hyperparameter tuning algorithms can be applied to SR. However, these approaches are extremely computationally expensive.

E. POST-PROCESSING AND ANALYSIS

Symbolic models are black-box, however, they are better suited for post-processing and analysis than other types of numerical approximators. Models in the form of analytic expressions are more expressive than, for example, the weights of neural networks and are amenable to further analyses. One can verify properties of the model, such as a monotonicity on a certain interval, positions of extremes, symmetry w.r.t. input variables or other domain-specific properties. Approaches based on satisfiability modulo theories solvers are used for this kind of formal constraint verifications in the literature [34].

Further, a contribution of the individual features to the overall performance of the analytic approximation of the V-function can be assessed and used to select the most significant features by using for example backward elimination. Over-simplified models can be further refined by adding new feature(s) in order to improve the current model's accuracy. This is hard to do with numerical approximators such as neural networks or basis function expansions.

Moreover, given a moderate number of internal parameters, it is possible to efficiently fine-tune them using global optimization techniques such as the pattern search [36] and evolution strategies [37], [38].

F. COMPUTATIONAL COMPLEXITY

The time needed for a single run of the `SVI`, `SPI` or `direct` method ranges from several minutes for the illustrative example to around 24 hours for the 2DOF problem on a standard desktop PC. The running time of the algorithm increases linearly with the size of the training data. However, the size of the training data set may grow exponentially with the state space dimension. In this article, we have generated the data on a regular grid. The efficiency gain depends on the way the data set is constructed. Other data generation methods are part of our future research. For high-dimensional problems, symbolic regression has the potential to be computationally more efficient than numerical approximation methods such as deep neural networks.

VI. CONCLUSIONS

We have proposed three methods based on symbolic regression to construct an analytic approximation of the V-function in a Markov decision process. The methods were experimentally evaluated on four nonlinear control problems: one first-order system, two second-order systems and one fourth-order system.

The main advantage of the approach proposed is that it produces smooth, compact V-functions, even if only sparsely sampled training data are available. The models produced are mathematically tractable and easy to analyse.

The number of their parameters is an order of magnitude smaller than in the case of a basis function approximator. The control performance in simulations and in experiments on a real setup is excellent. Moreover, the approximator in the form of a set of analytic expressions allows for easy post-processing and fine-tuning. It can also be easily reused within and plugged into other algorithms. For example, smooth policy derivation methods exploit the analytic nature of the symbolic V-function model.

No parameter tuning was used for the SR methods. We consider it as an important advantage of SR methods that they work well without any particular tuning. This is in contrast to e.g. deep neural network methods that often require substantial tuning before one gets them even to converge in a given RL problem.

The most significant current limitation of the approach is its high computational complexity. However, as the dimensionality of the problem increases, numerical approximators start to be limited by the computational power and memory capacity of standard computers. Symbolic regression does not suffer from such a limitation.

In our future work, we will evaluate the method on higher-dimensional problems, where we expect a large benefit over numerical approximators in terms of computational complexity. In relation to that, we will investigate smart methods for generating the training data. We will also investigate the use of input–output models instead of state-space models and closed-loop stability analysis methods for symbolic value functions. We will also develop techniques to incrementally control the complexity of the symbolic value function depending on its performance.

REFERENCES

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

[2] 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.

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

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

[5] 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.

[6] 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, Jun. 2012, pp. 1–8.

[7] 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.

[8] 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.

[9] 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," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[10] 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.

[11] P. Nagarajan and G. Warnell, "The impact of nondeterminism on reproducibility in deep reinforcement learning," 2018.

[12] E. Alibekov, J. Kubalík, and R. Babuška, "Policy derivation methods for critic-only reinforcement learning in continuous action spaces," *Engineering Applications of Artificial Intelligence*, 2018.

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

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

[15] E. Vladislavleva, T. Friedrich, F. Neumann, and M. Wagner, "Predicting the energy output of wind farms based on weather data: Important variables and their correlation," *Renewable Energy*, vol. 50, pp. 236–243, 2013.

[16] N. Staelens, D. Deschrijver, E. Vladislavleva, B. Vermeulen, T. Dhaene, and P. Demeester, "Constructing a No-Reference H. 264/AVC Bitstream-based Video Quality Metric using Genetic Programming-based Symbolic Regression," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 99, pp. 1–12, 2012.

[17] C. Brauer, "Using Eureqa in a Stock Day-Trading Application," 2012, cypress Point Technologies, LLC.

[18] 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, Sept 2016.

[19] ——, "Learning optimal policies in markov decision processes with value function discovery?" *SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 2, pp. 7–9, Sep. 2015.

[20] D. Jackson, *A New, Node-Focused Model for Genetic Programming*. Berlin, Heidelberg: Springer, 2012, pp. 49–60.

[21] ——, *Single Node Genetic Programming on Problems with Side Effects*. Berlin, Heidelberg: Springer, 2012, pp. 327–336.

[22] E. Alibekov, J. Kubalík, and R. Babuška, "Symbolic method for deriving policy in reinforcement learning," in *Proceedings 55th IEEE Conference on Decision and Control (CDC)*, Las Vegas, USA, Dec. 2016, pp. 2789–2795.

[23] J. Kubalík, E. Derner, and R. Babuška, "Enhanced symbolic regression through local variable transformations," in *Proceedings 9th International Joint Conference on Computational Intelligence (IJCCI 2017)*, Madeira, Portugal, Nov. 2017, pp. 91–100.

[24] M. Hinchliffe, H. Hiden, B. McKay, M. Willis, M. Tham, and G. Barton, "Modelling chemical process systems using a multi-gene genetic programming algorithm," in *Late Breaking Paper, GP'96*, Stanford, USA, 1996, pp. 56–65.

[25] D. P. Searson, "GPTIPS 2: An open-source software platform for symbolic data mining," in *Handbook of Genetic Programming Applications*. Springer International Publishing, 2015, pp. 551–573. [Online]. Available: https://doi.org/10.1007/978-3-319-20883-1_22

[26] J. Žegklitz and P. Pošík, "Linear combinations of features as leaf nodes in symbolic regression," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '17. New York, NY, USA: ACM, 2017, pp. 145–146. [Online]. Available: http://doi.acm.org/10.1145/3067695.3076009

[27] I. Arnaldo, K. Krawiec, and U.-M. O'Reilly, "Multiple regression genetic programming," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '14. New York, NY, USA: ACM, 2014, pp. 879–886. [Online]. Available: http://doi.acm.org/10.1145/2576768.2598291

[28] I. Arnaldo, U.-M. O'Reilly, and K. Veeramachaneni, "Building predictive models via feature synthesis," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '15. New York, NY, USA: ACM, 2015, pp. 983–990. [Online]. Available: http://doi.acm.org/10.1145/2739480.2754693

[29] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998, vol. 1, no. 1.

[30] T. de Bruin, J. Kober, K. Tuyls, and R. Babuška, "Experience selection in deep reinforcement learning for control," *Journal of Machine Learning Research*, vol. 19, no. 9, pp. 1–56, 2018. [Online]. Available: http://jmlr.org/papers/v19/17-131.html

[31] K. Verbert, R. Tóth, and R. Babuška, "Adaptive friction compensation: A globally stable approach," *IEEE/ASME Transactions on Mechatronics*, vol. 21, no. 1, pp. 351–363, 2016.

[32] J. Damsteeg, S. Nageshrao, and R. Babuška, "Model-based real-time control of a magnetic manipulator system," in *Proceedings 56th IEEE Conference on Decision and Control (CDC)*, Melbourne, Australia, Dec. 2017, pp. 3277–3282.

[33] 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*, ser. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 958–966.

[34] I. Błądek and K. Krawiec, "Solving symbolic regression problems with formal constraints," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '19. New York, NY, USA: ACM, 2019, pp. 977–984.

[35] B. Doerr, T. Kötzing, J. A. G. Lagodzinski, and J. Lengler, "Bounding bloat in genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '17. New York, NY, USA: ACM, 2017, pp. 921–928. [Online]. Available: http://doi.acm.org/10.1145/3071178.3071271

[36] C. Audet and J. Dennis, "Analysis of generalized pattern searches," *SIAM Journal on Optimization*, vol. 13, no. 3, pp. 889–903, 2002.

[37] N. Hansen, *The CMA Evolution Strategy: A Comparing Review*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 75–102. [Online]. Available: https://doi.org/10.1007/3-540-32494-1_4

[38] T. Glasmachers, T. Schaul, S. Yi, D. Wierstra, and J. Schmidhuber, "Exponential natural evolution strategies," *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 393–400, 2010. [Online]. Available: http://infoscience.epfl.ch/record/163869

DR. JIŘÍ KUBALÍK received the M.Sc. degree in computer science and the Ph.D. degree in artificial intelligence and biocybernetics from the Czech Technical University in Prague, in 1994 and 2001, respectively. He is a senior researcher at CTU in Prague, the Czech Institute of Informatics, Robotics and Cybernetics. His research has mainly been focused on various types of evolutionary computation techniques and their applications to hard optimization problems. He is a (co-)author of more than 30 papers in this area.

ERIK DERNER received the M.Sc. degree (Hons.) in artificial intelligence and computer vision from Czech Technical University (CTU) in Prague, where he is currently pursuing the Ph.D. degree. His research interests include sample-efficient methods for mobile robotics, genetic programming, reinforcement learning, and computer vision. He currently focuses on long-term autonomy in robot control and navigation. The central topic in his research is the use of symbolic regression to automatically construct nonlinear models of dynamic systems.

JAN ŽEGKLITZ is a Ph.D. candidate at the Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University (CTU) in Prague, Czech Republic. He received his M.Sc. degree from CTU in Artificial Intelligence. His research interests are in the field of evolutionary algorithms and genetic programming. The main research focus is on Multi-Gene Genetic Programming.

PROF. DR. ROBERT BABUŠKA received the M.Sc. (Hons.) degree in control engineering from the Czech Technical University in Prague, in 1990, and the Ph.D. (cum laude) degree from Delft University of Technology, the Netherlands, in 1997. He has had faculty appointments with the Czech Technical University in Prague and with the Electrical Engineering Faculty, TU Delft. Currently, he is a full professor of Intelligent Control and Robotics at TU Delft, Faculty 3mE, Department of Cognitive Robotics. In the past, he made seminal contributions to the field of nonlinear control and identification with the use of fuzzy modeling techniques. His current research interests include reinforcement learning, adaptive and learning robot control, nonlinear system identification and state-estimation. He has been involved in the applications of these techniques in various fields, ranging from process control to robotics and aerospace.