Constructing parsimonious analytic models for dynamic systems via symbolic regression

Derner, Erik; Kubalík, Jiří; Ancona, Nicola; Babuška, Robert

DOI
10.1016/j.asoc.2020.106432

Publication date
2020

Document Version
Accepted author manuscript

Published in
Applied Soft Computing Journal

Constructing Parsimonious Analytic Models for Dynamic Systems via Symbolic Regression

Erik Derner[a,b], Jiří Kubalík[a], Nicola Ancona[c], Robert Babuška[a,c]

Corresponding author: Erik Derner, erik.derner@cvut.cz

[a] *Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, Prague, 16000, Czech Republic*
[b] *Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, 12135, Czech Republic*
[c] *Cognitive Robotics, Delft University of Technology, Delft, 2628 CD, The Netherlands*

Abstract

Developing mathematical models of dynamic systems is central to many disciplines of engineering and science. Models facilitate simulations, analysis of the system's behavior, decision making and design of automatic control algorithms. Even inherently model-free control techniques such as reinforcement learning (RL) have been shown to benefit from the use of models, typically learned online. Any model construction method must address the tradeoff between the accuracy of the model and its complexity, which is difficult to strike. In this paper, we propose to employ symbolic regression (SR) to construct parsimonious process models described by analytic equations. We have equipped our method with two different state-of-the-art SR algorithms which automatically search for equations that fit the measured data: Single Node Genetic Programming (SNGP) and Multi-Gene Genetic Programming (MGGP). In addition to the standard problem formulation in the state-space domain, we show how the method can also be applied to input–output models of the NARX (nonlinear autoregressive with exogenous input) type. We present the approach on three simulated examples with up to 14-dimensional state space: an inverted pendulum, a mobile robot, and a bipedal walking robot. A comparison with deep neural networks and local linear regression shows that SR in most cases outperforms these commonly used alternative methods. We demonstrate on a real pendulum system that the analytic model found enables a RL controller to successfully perform the swing-up task, based on a model constructed from only 100 data samples.

Keywords: symbolic regression, genetic programming, model learning, reinforcement learning

1. Introduction

Numerous methods rely on an accurate model of the system. Model-based techniques comprise a wide variety of methods such as model predictive control [1, 2], time series prediction [3], fault detection and diagnosis [4, 5], or reinforcement learning (RL) [6, 7].

Even though model-free algorithms are available, the absence of a model slows down convergence and leads to extensive learning times [8, 9, 10]. Various model-based methods have been proposed to speed up learning [11, 12, 13, 14, 15]. To that end, many model-learning approaches are available: time-varying linear models [16, 17], Gaussian processes [18, 19] and other probabilistic models [20], basis function expansions [21, 22], regression trees [23], deep neural networks [24, 25, 7, 26, 27, 28, 29] or local linear regression [30, 31, 32].

All the above approaches suffer from drawbacks induced by the use of the specific approximation technique, such as a large number of parameters (deep neural networks), local nature of the approximator (local linear regression), computational complexity (Gaussian processes), etc. In this article, we propose another way to capture the system dynamics: using analytic models constructed by means of the symbolic regression method (SR). Symbolic regression is based on genetic programming and it has been used in nonlinear data-driven modeling, often with quite impressive results [33, 34, 35, 36, 37].

Symbolic regression appears to be quite unknown to the machine learning community as only a few works have been reported on the use of SR for control of dynamic systems. For instance, modeling of the value function by means of genetic programming is presented in [38], where analytic descriptions of the value function are obtained based

Article accepted to the Applied Soft Computing Journal. https://doi.org/10.1016/j.asoc.2020.106432

on data sampled from the optimal value function. Another example is the work [39], where SR is used to construct an analytic function, which serves as a proxy to the value function and a continuous policy can be derived from it. A multi-objective evolutionary algorithm was proposed in [40], which is based on interactive learning of the value function through inputs from the user. SR is employed to construct a smooth analytic approximation of the policy in [41], using the data sampled from the interpolated policy.

To our best knowledge, there have been no reports in the literature on the use of symbolic regression for constructing a process model in model-based control methods. We argue that the use of SR for model learning is a valuable element missing from the current nonlinear control schemes and we demonstrate its usefulness.

In this paper, we extend our previous work [42, 43], which indicated that SR is a suitable tool for this task. It does not require any basis functions defined a priori and contrary to (deep) neural networks it learns accurate, parsimonious models even from very small data sets. Symbolic regression can handle also high-dimensional problems and it does not suffer from the exponential growth of the computational complexity with the dimensionality of the problem, which we demonstrate on an enriched set of experiments including a complex bipedal walking robot system. In this work, we extend the use of the method to the class of input–output models, which are suitable in cases when the full state vector cannot be measured. By testing our method with two different state-of-the-art genetic programming algorithms, we demonstrate that the method is not dependent on the particular choice of the SR algorithm.

The paper is organized as follows. Sections 2 and 3 present the relevant context for model learning and the proposed method. The experimental evaluation of the method is reported in Section 4 and the conclusions are drawn in Section 5. Appendix A describes the RL method used in this paper and Appendix B lists detailed results of the experiments.

2. Theoretical Background

The discrete-time nonlinear state-space process model is described as

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

with the state $x_k, x_{k+1} \in \mathcal{X} \subset \mathbb{R}^n$ and the input $u_k \in \mathcal{U} \subset \mathbb{R}^m$. Note that the actual process can be stochastic (typically when the sensor readings are corrupted by noise), but in this paper we aim at constructing a deterministic process model (1).

The full state vector cannot be directly measured for a vast majority of processes and a state estimator would have to be used. In the absence of an accurate process model, such a reconstruction is inaccurate and has a negative effect on the overall performance of the control algorithm on the real system. Note that this problem has not been explicitly addressed in the literature, as most results are demonstrated on simulation examples in which the state information is available.

Therefore, next to state-space models, we also investigate the use of dynamic input–output models of the NARX (nonlinear autoregressive with exogenous input) type. The NARX model establishes a relation between the past input–output data and the predicted output:

$$y_{k+1} = g\left(y_k, y_{k-1}, \ldots, y_{k-n_y+1}, u_k, u_{k-1}, \ldots, u_{k-n_u+1}\right), \tag{2}$$

where n_y and n_u are user-defined integer parameters based on the expected system's order, and g is a static function, different from the function f used in the state-space model (1).

For the ease of notation, we group the lagged outputs and inputs into one vector:

$$\varphi_k = [y_k, y_{k-1} \ldots, y_{k-n_y+1}, u_{k-1}, \ldots, u_{k-n_u+1}]$$

and write model (2) as:

$$y_{k+1} = g\left(\varphi_k, u_k\right). \tag{3}$$

Note that in this setting, the model function and also the control policy are found from data samples which live in a space that is very different from the state space. The lagged outputs $y_k, y_{k-1}, \ldots, y_{k-n_y+1}$ are highly correlated and therefore span a deformed space. This presents a problem for many types of approximators. For instance, basis

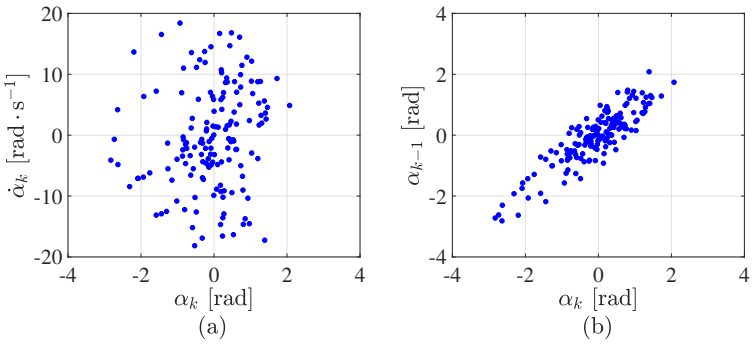


Figure 1: An example of trajectory samples obtained from the real inverted pendulum (see Section 4.3) in the original state space (a), and in the space formed by the current and previous output (b).

functions defined by the Cartesian product of the individual lagged variables will cover the whole product space $y_k \times y_{k-1} \times \ldots \times y_{k-n_y+1}$, while data samples only span a small, diagonally oriented part of the space, as illustrated in Figure 1. The SR approach described in this paper does not suffer from such drawbacks.

In this paper, we use reinforcement learning as the control method of choice. Please refer to Appendix A for details on the RL method used.

3. Method

In this section, we explain the principle of our method, briefly describe two variants of genetic programming algorithms used in this work, and discuss the computational complexity of our approach.

3.1. Symbolic Regression

Symbolic regression is employed to approximate the unknown state transition function f in the state-space model (1) or g in the input–output model (2). The analytic expressions describing the process to be controlled are constructed through genetic programming. SR methods were reported in the literature to work faster when using a linear combination of evolved nonlinear functions instead of evolving the whole analytic expression at once [44, 45]. Therefore, we define the class of analytic state-space models as:

$$f(x,u) = \beta_0 + \sum_{i=1}^{n_f} \beta_i f_i(x,u) \tag{4}$$

and the class of analytic input–output (NARX) models as:

$$g(\varphi,u) = \beta_0 + \sum_{i=1}^{n_f} \beta_i g_i(\varphi,u)\,. \tag{5}$$

The nonlinear functions $f_i(x,u)$ or $g_i(\varphi,u)$, called features, are constructed from a set of user-defined elementary functions. These functions can be nested and are evolved by means of standard evolutionary algorithm operations, such as mutation, so that the mean-square error calculated over the training data set is minimized. No a priori knowledge on the structure of the nonlinear model is needed. The set of elementary functions may be broad to let the SR algorithm select functions that are most suitable for fitting the given data. However, it is also possible to provide the algorithm with a partial knowledge about the problem. A narrower selection of elementary functions restricts the search space and speeds up the evolution process.

To avoid over-fitting, we control the complexity of the regression model by imposing a limit on the number of features n_f and the maximum depth d of the tree representation of the features. The coefficients β_i are estimated by least squares.

3.2. Genetic Programming Methods Used

In order to demonstrate that our method is not dependent on the particular choice of the SR algorithm, we test our approach with two different genetic programming methods: a modified version of Single Node Genetic Programming (SNGP) [46, 47, 48] and a modified version of Multi-Gene Genetic Programming (MGGP) [37]. Both methods have been successfully used for symbolic regression, with several applications in the RL and robotics domains [49, 41, 43, 42].

SNGP is a graph-based genetic programming technique that evolves a population of nodes organized in the form of an ordered linear array. The nodes can be of various types depending on the particular problem. In the context of SR, the node can either be a terminal, i.e., a constant or a variable, or some operator or function chosen from a set of functions defined by the user for the problem at hand. The individuals are interconnected in the left-to-right manner, meaning that an individual can act as an input operand only of those individuals which are positioned to its right in the population. Thus, the whole population represents a graph structure with multiple expressions rooted in the individual nodes. Expressions rooted in the function nodes can represent non-linear symbolic functions of various complexity. The population is evolved through a local search procedure using a single reversible mutation operator.

MGGP is a tree-based genetic programming algorithm utilizing multiple linear regression. The main idea behind MGGP is that each individual is composed of multiple independent expression trees, called genes, which are put together by a linear combination to form a single final expression. The parameters of this top-level linear combination are computed using multiple linear regression where each gene acts as an independent feature. In this article, we build upon a particular implementation of MGGP – GPTIPS2 [50]. This particular instance of MGGP uses two crossover operators: (i) high-level crossover that combines gene sets of two parents; (ii) low-level crossover which is a classical Koza-style [51] subtree crossover operating on corresponding pairs of parental genes. Also, there are two mutation operators: (i) subtree mutation, which is a classical Koza-style subtree mutation; (ii) constant mutation, which alters the numerical values of leaves representing constants. Both the crossover and mutation operators are chosen stochastically.

Detailed explanation of these algorithms and their parameters is beyond the scope of this paper and we refer the interested reader to [48] and [37].

3.3. Computational Complexity

The computational complexity of the symbolic regression algorithms used in this work increases linearly with the size of the training data set as well as with the dimensionality of the problem. For example, considering a problem with one-dimensional state, one-dimensional input, one-dimensional output, and a data set of 1000 samples, a single run of the SNGP or MGGP algorithm with the default configuration takes about 3 minutes on a single core of a standard desktop PC. For a system with a 14-dimensional regressor and a 6-dimensional input, a single run takes up to 20 minutes.

4. Experimental Results

We have carried out experiments with three nonlinear systems: a mobile robot, a 1-DOF inverted pendulum and a bipedal walking robot. The data, the codes and the detailed configuration of the experiments is available in our repository[1].

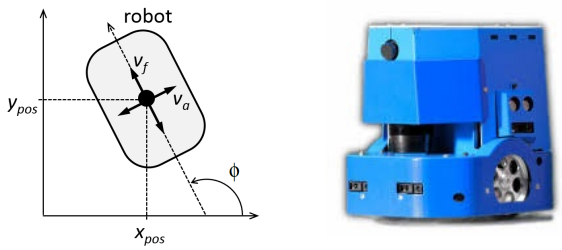
The simulation experiment with the mobile robot illustrates the use of the presented method, showing the precision and compactness of the models found in the case where the ground truth is known (Section 4.1). We show that the method is not dependent on the particular choice of the SR algorithm by comparing the performance of two SR methods, SNGP and MGGP. The subsequent experiment with the walking robot presents a more complex example and shows the performance of the method in a high-dimensional space (Section 4.2). With this example, we demonstrate the ability of the method to construct standard state-space models as well as input–output (NARX) models and we show how the method performs compared to two deep neural networks with different architectures. We conclude our set of experiments with the inverted pendulum system (Section 4.3). Similarly as in the experiment with the mobile

[1] https://github.com/erik-derner/symbolic-regression

robot, we evaluate the method with SNGP and MGGP, and we compare the results to two alternative approaches: neural networks and local linear regression. In addition to measuring the model prediction error, we perform real-time closed-loop control experiments with a lab setup to evaluate the performance of the algorithm in real-world conditions.

4.1. Mobile Robot

The state of a two-wheel mobile robot, see Figure 2 and [52], is described by $x = [x_{pos}, y_{pos}, \phi]^\top$, with x_{pos} and y_{pos} the position coordinates and ϕ the heading. The control input is $u = [v_f, v_a]^\top$, where v_f represents the forward velocity and v_a the angular velocity of the robot.



(a) (b)

Figure 2: Mobile robot schematic (a) and photograph (b).

The continuous-time dynamic model of the robot is:

$$\begin{aligned} \dot{x}_{pos} &= v_f \cos(\phi), \\ \dot{y}_{pos} &= v_f \sin(\phi), \\ \dot{\phi} &= v_a \,. \end{aligned} \tag{6}$$

4.1.1. Data Sets

We generated a noise-free data set by using the Euler method to simulate the differential equations (6). With a sampling period $T_s = 0.05$ s, the discrete-time approximation of (6) becomes:

$$\begin{aligned} x_{pos,k+1} &= x_{pos,k} + 0.05\, v_{f,k} \cos(\phi), \\ y_{pos,k+1} &= y_{pos,k} + 0.05\, v_{f,k} \sin(\phi), \\ \phi_{k+1} &= \phi_k + 0.05\, v_{a,k} \,. \end{aligned} \tag{7}$$

We generated training data sets of different sizes n_s. The initial state x_0 and the control input u_k for the whole simulation were randomly chosen from the ranges:

$$\begin{aligned} x_{pos} &\in [-1, 1]\,\mathrm{m}, \\ y_{pos} &\in [-1, 1]\,\mathrm{m}, \\ \phi &\in [-\pi, \pi]\,\mathrm{rad}, \\ v_f &\in [-1, 1]\,\mathrm{m \cdot s^{-1}}, \\ v_a &\in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]\,\mathrm{rad \cdot s^{-1}}. \end{aligned} \tag{8}$$

A test data set was generated in order to assess the quality of the analytic models on data different from the training set. The test data set entries were sampled on a regular grid with 11 points spanning evenly each state and action component domain, as defined by (8). These samples were stored together with the next states calculated by using the Euler approximation.

4.1.2. Experiment Setup

The purpose of this experiment was to test the ability of the SNGP and MGGP algorithms to recover from the data the analytic process model described by a known state-transition function. In order to assess the performance depending on the size of the data set and the complexity of the model, different combinations of the number of features n_f and the size of the training set n_s were tested. As the used algorithms only allow modeling one output at a time, they were run independently for each of the state components $x_{pos,k+1}$, $y_{pos,k+1}$ and ϕ_{k+1}.

The size of the SNGP population was set to 500 individuals and the evolution duration to 30000 generations. The set of elementary functions was defined as $\{*, +, -, \sin, \cos\}$. The maximum depth d of the evolved nonlinear functions was set to 7 and the number of features was $n_f \in \{1, 2, 10\}$. To ensure a fair evaluation, the parameters of the MGGP algorithm were set similarly to provide both methods with a comparable amount of computational resoruces, taking into account the conceptual differences between the two algorithms.

4.1.3. Results

The models found by symbolic regression were evaluated by calculating the RMSE median on the test data set over 30 independent runs of the SR algorithm. Note that each run yields a different model because the evolution process is guided by a unique sequence of random numbers. The results are listed in Table B.1 in Appendix B.

An example of a process model found by running SNGP with the parameters $n_f = 2$ and $n_s = 100$ is:

$$\begin{aligned}
\hat{x}_{pos,k+1} &= 1.0\, x_{pos,k} + 0.0499998879\, v_{f,k}\, \cos(\phi_k)\,, \\
\hat{y}_{pos,k+1} &= 1.000000023\, y_{pos,k} + 0.0500000056\, v_{f,k} \sin(\phi_k) + 0.0000000191\,, \\
\hat{\phi}_{k+1} &= 0.9999982931\, \phi_k + 0.0500000536\, v_{a,k} - 0.0000059844\,.
\end{aligned} \tag{9}$$

The coefficients are rounded to 10 decimal digits in order to demonstrate the magnitude of the error compared to the original Euler approximation (7). The results show that even with a small training data set, a precise, parsimonious analytic process model can be found based on noise-free data.

The results also demonstrate how the number of features n_f plays an important role in the setting of the experiment parameters. In general, the RMSE decreases with an increasing number of features, whereas the complexity naturally grows by adding more features to the final model (4). The higher RMSE error when using only one feature is caused mainly by the fact that all parameters have to be evolved by the genetic algorithm, which is hard. On the other hand, when using more features, the least squares method can quickly and accurately find the coefficients of the features. These results support our choice to define the class of analytic models as a linear combination of features, as explained in Section 3.1. As a corollary, if the outline of the model structure is known in advance, it is recommended to set the number of features at least equal to the number of terms expected in the underlying function. Otherwise, it is advisable to set the number of features large enough, e.g. $n_f = 10$.

4.2. Walking Robot

The robot LEO is a 2D bipedal walking robot [53], see Figure 3. It has 7 actuators: two in the ankles, knees and hips and one in its shoulder that allows the robot to stand up after a fall. LEO is connected to a boom with a parallelogram construction. This keeps the hip axis always horizontal, which makes it effectively a 2D robot and thus eliminates the sideways stability problem.

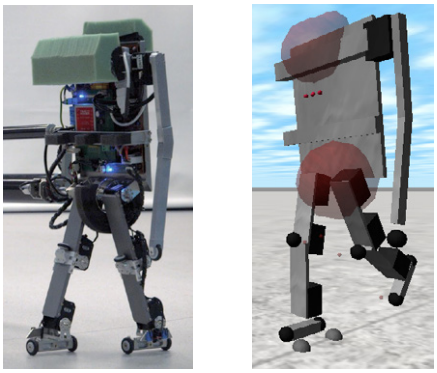
The state vector of LEO $x = [\psi, \dot{\psi}]^\top$ consists of 14 components, where

$$\psi = [\psi_{TRS}, \psi_{LH}, \psi_{RH}, \psi_{LK}, \psi_{RK}, \psi_{LA}, \psi_{RA}]^\top \tag{10}$$

represents the angles of the torso, left and right hip, the knee and the ankle. Likewise,

$$\dot{\psi} = [\dot{\psi}_{TRS}, \dot{\psi}_{LH}, \dot{\psi}_{RH}, \dot{\psi}_{LK}, \dot{\psi}_{RK}, \dot{\psi}_{LA}, \dot{\psi}_{RA}]^\top \tag{11}$$

are the angular velocities of the torso, hips, knees and ankles. The action space of LEO comprises the voltage inputs to the seven joint actuators.



(a) (b)

Figure 3: The walking robot LEO: photograph (a) and simulation model rendering (b). [54].

4.2.1. Data Sets

In order to apply symbolic regression, the walking robot LEO was modeled using the Rigid Body Dynamics Library (RBDL) [55] and the data sets were generated using the Generic Reinforcement Learning Library (GRL) [56], which allowed us to record trajectories while the robot was learning to walk.

We split the data set into two disjoint subsets: a training set and a test set. Both subsets are composed of consecutive samples from the simulation, which was run with a sampling period $T_s = 0.03\,\mathrm{s}$.

4.2.2. Experiment Setup

The experiment was designed to evaluate the performance of our method on a more complex, high-dimensional example and to construct input–output (NARX) models in addition to the standard state-space models. We chose to use only the SNGP algorithm for this experiment and the main parameters were configured as follows. The population size was set to 500 and the number of generations to 30000. The depth limit d was fixed to 5 and the number of features was $n_f \in \{1, 5, 10\}$. The elementary function set was defined as $\{*, +, -, \sin, \cos, \mathrm{square}, \mathrm{cube}\}$.

During the simulation used to obtain the data sets, the shoulder was not actuated. Therefore, the input vector had only 6 components, one for each actuator. As in the case of the mobile robot, SNGP was run separately for each of the 14 state components.

In *Experiment B1*, we used SR to generate standard state-space models. In *Experiment B2*, we generated input–output models with the regression vector defined as $\varphi = [\psi_k, \psi_{k-1}, u_{k-1}]$.

4.2.3. Results

In order to evaluate the ability of SNGP to approximate the state-transition function, we calculated the RMSE medians over 30 runs of the algorithm on the test data set. The results for the state-space models are reported in Table B.2 and for the input–output models in Table B.3 in Appendix B.

The results show the expected trend, which can be seen in all experiments: the quality of the models improves with the size of the training data set. However, it is noteworthy that the difference between the RMSE for models trained on 100 samples and for those trained on 5000 samples are in most cases negligible. This confirms our earlier observation that SR can be used to find accurate analytic process models on batches of data as small as 100 samples [42] even for high-dimensional systems.

The results for the input–output models are generally just slightly worse than those for the state-space models, with the benefit of speeding up the algorithm by reducing the number of modeled variables to a half.

4.2.4. Comparison with Alternative Methods

Deep neural networks are widely used to model an unknown system. In order to compare our method to alternative state-of-the-art methods, we have constructed two different neural networks:

- Deep neural network DNN-A was implemented in PyTorch. It consists of an input linear layer of size 20×200, followed by three linear layers with the size of 200×200, with a ReLU activation function used after each linear layer. The output layer has 200×14 units. The batch size was set to 32. The SGD algorithm [57] was used with the learning rate of 8.5×10^{-4}.
- Deep neural network DNN-B was implemented in TensorFlow. It is a fully connected network with 1 hidden layer, consisting of 512 units with ELU nonlinearity and 50% dropout. The batch size was set to 8. The Adam optimizer [58] was used with a learning rate of 10^{-3} and early stopping.

We chose the RMSE medians of SNGP with $n_s = 1000$ and $n_f = 10$ as the benchmark configuration. State-space models were used in this scenario. The training and test sets were the same for all compared methods. Figure 4 shows an overview of the performance of the two variants of DNN compared to the SNGP algorithm and detailed results are presented in Table B.4 in Appendix B. The results show that the SNGP algorithm is able to find substantially better models than the neural networks for the angles, while the performance on the angular velocities is comparable among all the tested methods.

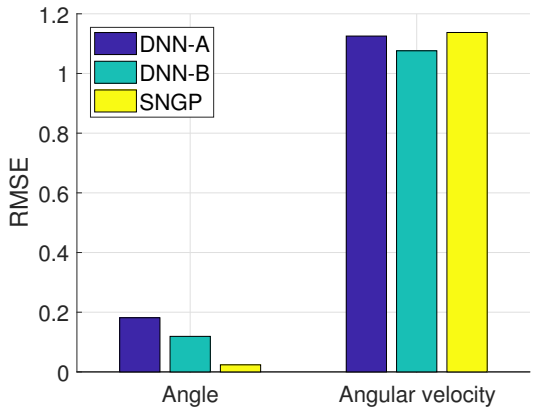


Figure 4: Comparison of two DNN variants with the SNGP algorithm on the walking robot example. The bars show the mean RMSE over the 7 angles and over the 7 angular velocities on the test data set.

4.3. Inverted Pendulum

The inverted pendulum system consists of a weight of mass m attached to an actuated link which rotates in the vertical plane, see Figure 5a. The state vector is $x = [\alpha, \dot{\alpha}]^\top$, where α is the angle and $\dot{\alpha}$ is the angular velocity of the link. The control input is the voltage u. The continuous-time model of the pendulum dynamics is:

$$\ddot{\alpha} = \frac{1}{J} \cdot \left(\frac{K}{R} u - m\, g\, l\, \sin(\alpha) - b\, \dot{\alpha} - \frac{K^2}{R}\, \dot{\alpha} - c\, \mathrm{sign}(\dot{\alpha}) \right) \tag{12}$$

with $J = 1.7937 \times 10^{-4}\,\mathrm{kg \cdot m^2}$, $m = 0.055\,\mathrm{kg}$, $g = 9.81\,\mathrm{m \cdot s^{-2}}$, $l = 0.042\,\mathrm{m}$, $b = 1.94 \times 10^{-5}\,\mathrm{N \cdot m \cdot s \cdot rad^{-1}}$, $K = 0.0536\,\mathrm{N \cdot m \cdot A^{-1}}$, $R = 9.5\,\Omega$ and $c = 8.5 \times 10^{-4}\,\mathrm{kg \cdot m^2 \cdot s^{-2}}$. The angle is $\alpha = 0$ or $\alpha = 2\pi$ for the pendulum pointing down and $\alpha = \pi$ for the pendulum pointing up.

The reward function used in the RL experiments was defined as follows:

$$\rho(x_k, u_k, x_{k+1}) = -0.5|\alpha_r - \alpha_k| - 0.01|\dot{\alpha}_r - \dot{\alpha}_k| - 0.05|u_k|, \tag{13}$$

where $[\alpha_r, \dot{\alpha}_r]^\top$ is a constant reference (goal) state.

4.3.1. Data Collection

As we will present an experiment with the inverted pendulum performing a control task, we start this section by a short overview of the data collection methods used. Two different situations can be distinguished: initial model learning and model learning under a given policy.

(a) (b)

Figure 5: Inverted pendulum schematic (a) and the real inverted pendulum system (b).

Initial Model Learning. At the beginning, when the control policy is not yet available, the system can be excited by a test signal in order to obtain a sufficiently rich data set. Various methods for designing suitable test signals are described in the literature, such as the generalized binary noise (GBN) sequence [59]. The important parameters to be selected are the input signal amplitude, the way the random signal is generated (e.g., the 'switching' probability) and the experiment duration.

Model Learning Under a Given Policy. Once an acceptable control policy has been learned, the system can be controlled to execute the required task. Data can be collected while performing the control task and used to further improve the model. As the information captured in the data under steady operating conditions might not be sufficient in certain situations, the control input can be adjusted by adding a test signal in this case as well. The characteristics of this test signal are usually different from the one used for initial model learning; for instance, it typically has a lower amplitude.

4.3.2. Data Sets

We used both simulated and real measured data in the experiments with the inverted pendulum. In all experiments, the discrete-time sampling period used was $T_s = 0.05\,\mathrm{s}$.

At first, we generated a noise-free data set for *Experiment C1* by using the Euler method to simulate the differential equation (12):

$$\begin{aligned} \alpha_{k+1} &= \alpha_k + 0.05\,\dot{\alpha}_k, \\ \dot{\alpha}_{k+1} &= 0.9102924564\,\dot{\alpha}_k - 0.2369404025\,\mathrm{sign}(\dot{\alpha}_k) \\ &\quad + 1.5727561084\,u_k - 6.3168590065\,\sin(\alpha_k). \end{aligned} \tag{14}$$

The data set for *Experiment C2* was created by integrating (12) by using the fourth-order Runge-Kutta method and adding Gaussian noise. The transformation from the original states $x = [\alpha, \dot{\alpha}]^\top$ to the states with Gaussian noise $x_n = [\alpha_n, \dot{\alpha}_n]^\top$ is defined as

$$\begin{aligned} \alpha_n &= \alpha + \pi\lambda r_{n,1}\,, \\ \dot{\alpha}_n &= \dot{\alpha} + 40\lambda r_{n,2}\,, \end{aligned} \tag{15}$$

where $r_{n,1}$, $r_{n,2}$ are random numbers drawn from a normal distribution with zero mean and a standard deviation of 1. The constant $\lambda \in \{0, 0.01, 0.05, 0.1\}$ controls the amount of noise and the constants π and 40 make sure that the added noise is approximately proportional to the range of each variable.

In both *Experiments C1* and *C2*, the initial state was $\alpha = 0$, $\dot{\alpha} = 0$ and the control input was chosen randomly at each time step k from the range $u_k \in [-5, 5]\,\mathrm{V}$.

The test data sets were created similarly as in Section 4.1.1. The samples were generated on a regular grid of $31 \times 31 \times 31$ points, spanning the state and action domain: $\alpha \in [-\pi, \pi]\,\mathrm{rad}$, $\dot{\alpha} \in [-40, 40]\,\mathrm{rad \cdot s^{-1}}$ and $u \in [-5, 5]\,\mathrm{V}$. For all samples, the next states in the test set for *Experiment C1* were calculated using the Euler approximation. In *Experiment C2*, we generated a noise-free test set by applying the fourth-order Runge-Kutta method to all samples on the grid.

The real data for *Experiment C3* were measured on the real inverted pendulum system shown in Figure 5b. At first, the system was excited by applying a uniformly distributed random control input u_k within the range $[-5, 5]$ V at each time step k. The random interaction with the system lasted for 5 seconds and the recorded data set comprised 100 samples. The data are shown in Figure 6. The data set was later enriched by samples recorded while applying the control policy (A.5) to perform the swing-up task on the real system, which will be described in the following section.

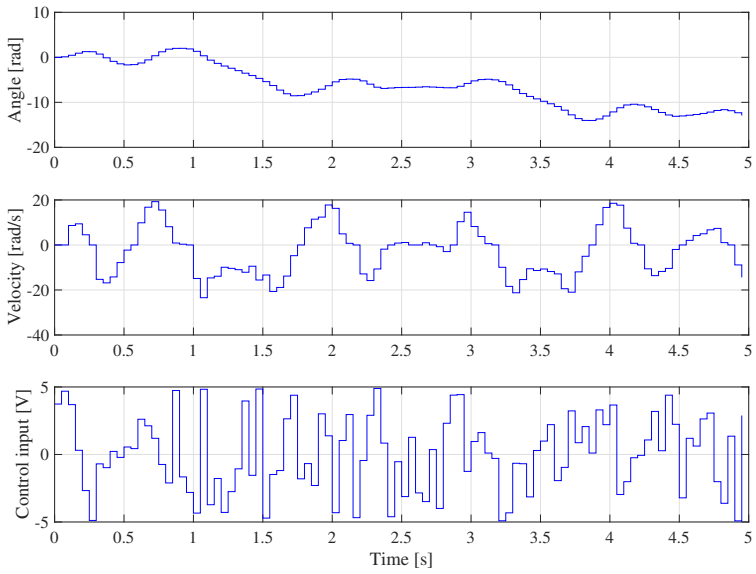


Figure 6: Initial data set obtained on the real inverted pendulum system as a response to the random input shown in the bottom panel.

The sequences recorded for *Experiment C3* were split into training and test subsets. Every third sample was used for the test set, while the remaining samples formed the training set. In all experiments, the reported RMSE values were calculated on the respective test data set.

4.3.3. Experiment Setup

Similarly as in the experiment with the mobile robot, the SNGP and MGGP algorithms were first employed in *Experiment C1* to test the ability of SR to generate precise models for the inverted pendulum system using a data set generated by the Euler method. The experiment serves to evaluate how the training data set size n_s and the number of features n_f influence the quality of the model.

Experiment C2 demonstrates how the analytic process models are evolved using the Runge-Kutta simulation data set with noise. The maximum number of features n_f in the symbolic regression algorithms was set to 10 in order to facilitate the evolution of models capturing the more complex underlying function. This experiment tests the behavior of the method in environments with noisy measurements.

We conclude the experiments with *Experiment C3*, which shows the intended use of the method within RL on the example of the underactuated swing-up task, performed on a real inverted pendulum system. The control goal is to stabilize the pendulum in the unstable equilibrium $x_r = [\alpha_r, \dot{\alpha}_r]^\top = [\pi, 0]^\top$. As the input is limited to the range $u \in [-2, 2]$ V, the available torque is insufficient to push the pendulum directly up from the majority of initial states, and therefore it has to be first swung back and forth to gather energy. At first, we constructed 30 analytic models using the data set recorded under random input and then selected the model with the lowest RMSE on the test set. This *initial model* was employed to calculate the policy for the swing-up task (see Appendix A). To find an approximation of the optimal value function, we used the fuzzy V-iteration algorithm [60]. We applied the policy to the real system in four independent runs, starting at the initial state $x_0 = [0, 0]^\top$. In addition, we performed other four swing-ups with exploration noise added to the control input. The exploration noise was normally distributed with the standard deviation ranging from 0.2 to 0.5 V. All eight sequences, each consisting of approximately 50 measurements, were

added to the initial data set recorded under random input. Using this extended data set, 30 refined analytic process models were learned and the model with the lowest error on the test set was chosen as the final *refined model*. Like in *Experiment C2*, the number of features was set to $n_f = 10$ to facilitate modeling the more complex state-transition function.

In all experiments, the size of the SNGP population was set to 500 and the evolution was limited to 30000 generations. The elementary function set was $\{*, +, -, \sin, \cos, \text{sign}\}$. The maximum depth d was set to 7. In *Experiment C1*, various numbers of features were tested: $n_f \in \{1, 2, 10\}$ for α and $n_f \in \{1, 4, 10\}$ for $\dot{\alpha}$. The parameters of the MGGP algorithm in *Experiment C1* and *C2* were set similarly, taking into account the conceptual differences between the two algorithms to allow for a fair comparison.

4.3.4. Results

The results of *Experiment C1* are summarized in Table B.5 in Appendix B for the SNGP and MGGP algorithm. Similarly as in the previous examples, the results indicate that the precision of the models increases with increasing number of features. The overall performance of both SR algorithms is comparable.

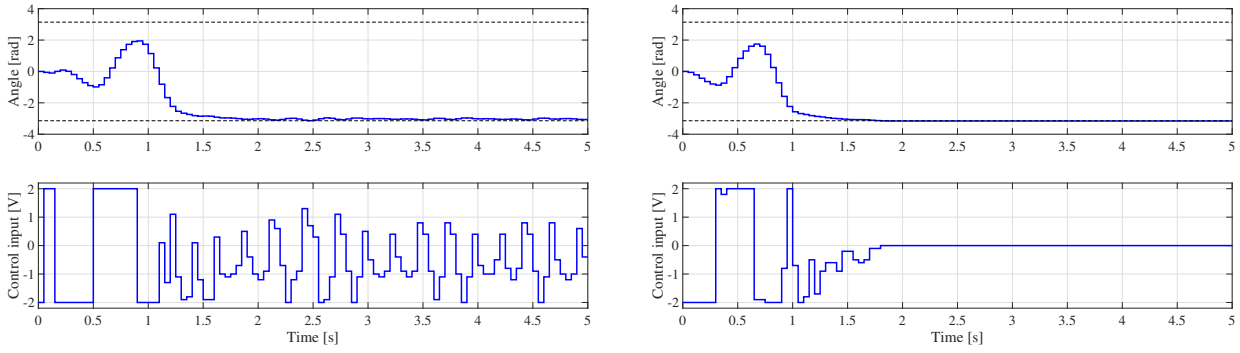
An example of an analytic process model found with the parameters $n_f = 2$ for α, $n_f = 4$ for $\dot{\alpha}$ and $n_s = 20$ is:

$$\begin{aligned} \hat{\alpha}_{k+1} &= \alpha_k + 0.05\,\dot{\alpha}_k - 0.0000000001\,, \\ \hat{\dot{\alpha}}_{k+1} &= 0.9102924745\,\dot{\alpha}_k - 0.2369403835\,\text{sign}(\dot{\alpha}_k) + 1.5727561072\,u_k \\ &\quad - 6.3168589936\,\sin(\alpha_k) + 0.0000000013\,. \end{aligned} \tag{16}$$

The error of the analytic model w.r.t. the Euler approximation (14) is very small. These results confirm that the proposed method can find precise models even on small data sets.

The results of *Experiment C2* presented in Table B.6 in Appendix B show that the analytic models are able to approximate the state-transition function well even on data with a reasonable amount of noise. The use of the Runge-Kutta method to generate data sets leads to substantially more complicated models than when using the data generated by using the Euler method. Again, the performance of the SNGP and MGGP algorithm is comparable.

In *Experiment C3*, we have shown that SR is able to find analytic process models using data collected on the real system. Already after a short (5 s) interaction under the random input, an analytic process model is found which enables RL to perform the swing-up, see Figure 7a. Performing the swing-up task allows to collect more data in important parts of the state space around the trajectory to the goal state. Figure 7b shows that the performance of the model further improves after adding data collected while performing the swing-up task with the initial model. Figure 8 compares the swing-up response with the initial and the refined model. The histogram in Figure 9 and a two-sample t-test with unpooled variance applied to the discounted return show that the performance improvement between the policy based on the initial and the refined analytic process model is statistically significant ($p = 2\times10^{-22}$). The RMSE medians over 30 runs of the SNGP algorithm were 1.70×10^{-2} for α and 6.03×10^{-1} for $\dot{\alpha}$ in case of the initial model and 1.16×10^{-2} for α and 3.35×10^{-1} for $\dot{\alpha}$ in case of the refined model.



(a) (b)

Figure 7: A typical real swing-up experiment with the initial model (a) and the refined model (b).

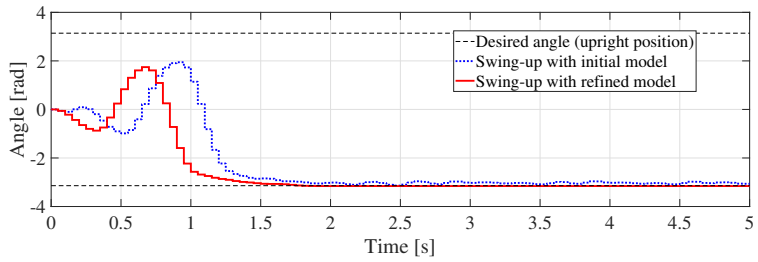


Figure 8: Comparison of the real swing-up response with the initial model, learnt from the random data, and the refined model, learnt from the random data merged with additional data from eight real swing-up experiments.

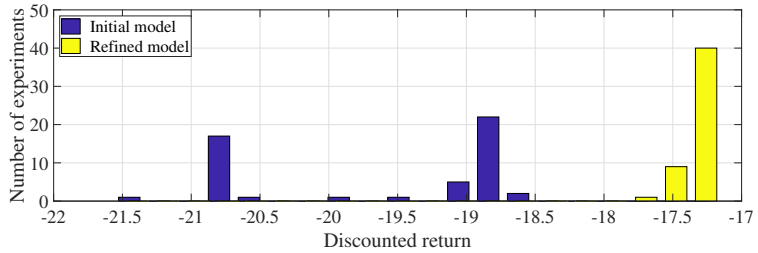


Figure 9: Histograms of 50 real experiments with the initial model and the refined model measured by the discounted return. The performance improvement is statistically significant ($p = 2 \times 10^{-22}$).

4.3.5. Comparison with Alternative Methods

We compared our modeling results with local linear regression (LLR) [30]. We selected the Runge-Kutta data set with 1000 samples and zero noise as a reference training set and the regular grid as a test set (see Section 4.3.2 for details). The LLR memory contained 1000 samples and the number of nearest neighbors was set to 10. The RMSE achieved by LLR was 1.73×10^{-1} for α and 6.93×10^{0} for $\dot{\alpha}$. In both cases, the SNGP algorithm achieved a better RMSE by at least one order of magnitude (6.11×10^{-3} for α and 5.04×10^{-1} for $\dot{\alpha}$).

We also compared the results of our method to a neural network. Given the relative simplicity of the problem, the network had one hidden layer, consisting of 40 neurons, and it was trained using the Levenberg-Marquardt algorithm. The number of neurons in the hidden layer was tuned by testing networks with 5 to 100 neurons and choosing the one that performed best on the test data. The RMSE achieved on the aforementioned reference data set was 6.82×10^{-2} for α and 2.59×10^{0} for $\dot{\alpha}$. Again, compared to the RMSE values achieved by our method (stated at the end of the previous paragraph), symbolic regression finds substantially better models in terms of RMSE compared to those found by the neural network.

5. Conclusions

We showed that symbolic regression is a very effective method for constructing dynamic process models from data. It generates parsimonious models in the form of analytic expressions, which makes it a good alternative to black-box models, especially in problems with limited amounts of data. Prior knowledge on the type of nonlinearities and model complexity can easily be included in the symbolic regression procedure. Despite the technique is not yet broadly used in the field of robotics and dynamic systems, we believe that it will become a standard tool for system identification.

The experiments with the walking robot demonstrate that symbolic regression can be used to construct precise process models even for high-dimensional systems. We have confirmed empirically that the computational complexity of the algorithm grows linearly with the dimensionality of the system. It is also worth mentioning that the complexity of the analytic models does not grow significantly with the complexity of the system.

The real-world experiment with the inverted pendulum shows that already after 5 seconds of interaction with the system, an initial analytic process model is found, which not only accurately predicts the process behavior, but also

serves as a reliable model for the design of an RL controller. By collecting the data during several executions of the swing-up task using the initial analytic model and adding them to the data set used by SR to learn the model, the performance on the swing-up task further improves.

Our evaluation shows that two distinct symbolic regression algorithms, SNGP and MGGP, perform comparably well on the evaluated systems. This indicates that the proposed method is not dependent on the particular choice of the symbolic regression method. We compared the performance of symbolic regression with alternative state-of-the-art methods, in particular with neural networks and with local linear regression. The results show that the proposed method performs in most cases significantly better than the alternatives.

Another important outcome is that SR can be used to find both state-space and input–output models. The use of input–output models is beneficial because it does not require the observations of the full state vector and it also makes the algorithm faster because of modeling a reduced number of variables.

We have identified several possibilities for future extensions of this work. The main objective is to apply SR methods within the entire RL scheme, i.e., also for approximating the V-function, and also to use analytic models in combination with actor-critic online RL. In some cases, especially when using many features, analytic models tend to be unnecessarily complex. In our future work, we will investigate systematic reduction of analytic models.

Acknowledgments

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) and by the Grant Agency of the Czech Technical University in Prague, grant no. SGS19/174/OHK3/3T/13.

The authors thank Tim de Bruin and Jonáš Kulhánek for their help with the DNN experiments for the walking robot and Jan Žegklitz for his help with experiments using the MGGP algorithm.

Appendix A. Reinforcement Learning

The system for which an optimal control strategy is to be learnt can be described by the nonlinear state-space model (1) or the input–output (NARX) model (2). The following text describes the case using the state-space models. For the input–output models, the reward function, the return, the value function and the optimal control action (A.1)–(A.5) can be defined analogously using the regressor φ instead of the state variable x.

The *reward function* assigns a scalar reward $r_{k+1} \in \mathbb{R}$ to the state transition from x_k to x_{k+1}, under action u_k:

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

The reward function ρ specifies the control goal, typically as the distance of the current state to a given goal state.

Based on model (1), we compute the optimal control policy $\pi : \mathcal{X} \to \mathcal{U}$ such that in each state it selects a control action so that the expected cumulative discounted reward over time, called the return, is maximized:

$$R^{\pi} = E\Big\{\sum_{k=0}^{\infty} \gamma^k \rho(x_k, \pi(x_k), x_{k+1})\Big\}\,. \tag{A.2}$$

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

$$V^{\pi}(x) = E\Big\{\sum_{k=0}^{\infty} \gamma^k \rho(x_k, \pi(x_k), x_{k+1})\Big|\, x_0 = x\Big\}\,. \tag{A.3}$$

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

$$\hat{V}^*(x) = \max_{u \in \mathcal{U}} \Big[\rho(x, u, f(x, u)) + \gamma\, \hat{V}^*\big(f(x, u)\big)\Big]\,. \tag{A.4}$$

To simplify the notation, we drop the superscripts; $V(x)$ therefore denotes an approximation of the optimal V-function. Based on $V(x)$, the corresponding approximately optimal control action is found as the one that maximizes the right-hand side of (A.4):

$$u = \operatorname*{argmax}_{u' \in U} \left[\rho(x, u', f(x, u')) + \gamma\, V(f(x, u'))\right]\,. \tag{A.5}$$

In this work, the above equation is used online as the control policy π with a set of discretized inputs U, so that the near-optimal control action can be found by enumeration.

Appendix B. Detailed Experiment Results

In this appendix, we present detailed results of the experiments described in Section 4.

Table B.1: Comparison of analytic process models for the experiment with the mobile robot. The table shows the RMSE medians over 30 runs of the SNGP (grey) and MGGP (white) algorithm for different numbers of features n_f and different numbers of training samples n_s.

Variable	n_f	Number of training samples n_s					
		20	50	100	200	500	1000
x_{pos}	1	1.77×10^{-1}	1.37×10^{-1}	9.98×10^{-2}	7.88×10^{-1}	3.25×10^{-2}	2.87×10^{-2}
		9.03×10^{-1}	7.66×10^{-1}	6.97×10^{-1}	9.16×10^{-1}	3.88×10^{-2}	3.29×10^{-2}
	2	6.44×10^{-8}	3.19×10^{-9}	2.05×10^{-9}	3.55×10^{-9}	5.93×10^{-9}	1.53×10^{-9}
		2.21×10^{-9}	5.64×10^{-10}	5.15×10^{-6}	1.11×10^{-2}	1.30×10^{-10}	1.37×10^{-10}
	10	3.78×10^{-5}	2.29×10^{-7}	1.09×10^{-7}	1.45×10^{-7}	5.21×10^{-9}	2.68×10^{-9}
		2.37×10^{-5}	1.39×10^{-7}	8.50×10^{-9}	5.54×10^{-9}	2.70×10^{-9}	5.26×10^{-10}
y_{pos}	1	9.06×10^{-1}	4.34×10^{-1}	1.39×10^{-1}	1.74×10^{-1}	3.21×10^{-2}	3.16×10^{-2}
		8.74×10^{-1}	9.47×10^{-1}	7.75×10^{-1}	7.33×10^{-1}	3.31×10^{-2}	3.16×10^{-2}
	2	4.87×10^{-1}	1.81×10^{-8}	1.18×10^{-8}	4.09×10^{-9}	1.93×10^{-8}	2.39×10^{-8}
		3.39×10^{-1}	3.38×10^{-2}	2.89×10^{-10}	2.76×10^{-10}	2.68×10^{-10}	2.14×10^{-2}
	10	4.48×10^{-4}	2.04×10^{-7}	4.11×10^{-7}	1.91×10^{-7}	1.60×10^{-8}	1.25×10^{-8}
		9.32×10^{-5}	1.16×10^{-7}	7.54×10^{-9}	6.45×10^{-9}	2.34×10^{-9}	8.33×10^{-10}
ϕ	1	9.81×10^{-2}	2.60×10^{-2}	6.44×10^{-4}	6.57×10^{-5}	6.79×10^{-4}	5.55×10^{-3}
		3.38×10^{0}	3.19×10^{0}	2.49×10^{-2}	1.34×10^{-3}	5.51×10^{-5}	5.48×10^{-5}
	2	7.05×10^{-8}	1.36×10^{-8}	6.16×10^{-9}	3.78×10^{-8}	7.78×10^{-9}	5.16×10^{-8}
		1.47×10^{-9}	5.08×10^{-10}	4.16×10^{-10}	4.02×10^{-10}	4.00×10^{-10}	4.01×10^{-10}
	10	5.35×10^{-6}	1.85×10^{-6}	2.09×10^{-6}	4.01×10^{-8}	6.00×10^{-9}	3.69×10^{-8}
		6.45×10^{-8}	9.34×10^{-9}	2.87×10^{-9}	1.35×10^{-9}	4.07×10^{-10}	4.00×10^{-10}

Table B.2: Comparison of the state-space analytic process models for the walking robot LEO in *Experiment B1*. The table shows the RMSE medians over 30 runs of the SNGP algorithm for varying number of features n_f and number of training samples n_s.

Variable	n_f	Number of training samples n_s					
		100	200	500	1000	2000	5000
ψ_{TRS}	1	4.09×10^{-2}	1.72×10^{-2}	4.15×10^{-2}	5.38×10^{-2}	5.46×10^{-2}	5.68×10^{-2}
	5	1.90×10^{-2}	1.55×10^{-2}	1.46×10^{-2}	1.40×10^{-2}	1.39×10^{-2}	1.38×10^{-2}
	10	2.01×10^{-2}	1.62×10^{-2}	1.44×10^{-2}	1.32×10^{-2}	1.29×10^{-2}	1.25×10^{-2}
ψ_{LH}	1	2.99×10^{-2}	2.99×10^{-2}	3.60×10^{-2}	2.24×10^{-2}	2.34×10^{-2}	8.81×10^{-2}
	5	2.78×10^{-2}	2.38×10^{-2}	2.15×10^{-2}	2.07×10^{-2}	2.02×10^{-2}	2.01×10^{-2}
	10	2.99×10^{-2}	2.53×10^{-2}	2.20×10^{-2}	2.06×10^{-2}	1.95×10^{-2}	1.92×10^{-2}
ψ_{RH}	1	1.05×10^{-1}	9.16×10^{-2}	4.01×10^{-2}	3.71×10^{-2}	3.36×10^{-2}	2.84×10^{-2}
	5	3.81×10^{-2}	3.19×10^{-2}	2.69×10^{-2}	2.71×10^{-2}	2.61×10^{-2}	2.56×10^{-2}
	10	4.15×10^{-2}	3.54×10^{-2}	2.65×10^{-2}	2.65×10^{-2}	2.46×10^{-2}	2.46×10^{-2}
ψ_{LK}	1	5.52×10^{-2}	8.01×10^{-2}	2.43×10^{-2}	2.35×10^{-2}	2.40×10^{-2}	2.28×10^{-2}
	5	3.10×10^{-2}	2.68×10^{-2}	2.29×10^{-2}	2.15×10^{-2}	2.06×10^{-2}	2.07×10^{-2}
	10	3.43×10^{-2}	2.87×10^{-2}	2.22×10^{-2}	2.10×10^{-2}	1.97×10^{-2}	1.89×10^{-2}
ψ_{RK}	1	2.82×10^{-2}	2.36×10^{-2}	2.31×10^{-2}	2.31×10^{-2}	2.15×10^{-2}	2.13×10^{-2}
	5	2.77×10^{-2}	2.28×10^{-2}	2.01×10^{-2}	2.01×10^{-2}	2.01×10^{-2}	1.90×10^{-2}
	10	3.08×10^{-2}	2.45×10^{-2}	1.96×10^{-2}	1.87×10^{-2}	1.86×10^{-2}	1.77×10^{-2}
ψ_{LA}	1	9.31×10^{-2}	1.11×10^{-1}	1.10×10^{-1}	1.10×10^{-1}	7.07×10^{-2}	5.74×10^{-2}
	5	5.18×10^{-2}	4.00×10^{-2}	3.11×10^{-2}	2.95×10^{-2}	2.80×10^{-2}	2.81×10^{-2}
	10	5.66×10^{-2}	4.31×10^{-2}	3.16×10^{-2}	2.92×10^{-2}	2.73×10^{-2}	2.62×10^{-2}
ψ_{RA}	1	4.65×10^{-2}	4.51×10^{-2}	4.24×10^{-2}	4.54×10^{-2}	4.33×10^{-2}	7.37×10^{-2}
	5	4.98×10^{-2}	4.49×10^{-2}	3.77×10^{-2}	3.66×10^{-2}	3.65×10^{-2}	3.52×10^{-2}
	10	5.35×10^{-2}	4.70×10^{-2}	3.84×10^{-2}	3.65×10^{-2}	3.50×10^{-2}	3.39×10^{-2}
$\dot{\psi}_{TRS}$	1	8.91×10^{-1}	8.51×10^{-1}	8.19×10^{-1}	7.99×10^{-1}	7.94×10^{-1}	7.84×10^{-1}
	5	1.07×10^{0}	8.72×10^{-1}	7.78×10^{-1}	7.19×10^{-1}	6.92×10^{-1}	6.86×10^{-1}
	10	1.20×10^{0}	9.27×10^{-1}	7.93×10^{-1}	7.00×10^{-1}	6.67×10^{-1}	6.41×10^{-1}
$\dot{\psi}_{LH}$	1	1.44×10^{0}	1.23×10^{0}	1.16×10^{0}	1.15×10^{0}	1.14×10^{0}	1.14×10^{0}
	5	2.22×10^{0}	1.41×10^{0}	1.17×10^{0}	1.15×10^{0}	1.11×10^{0}	1.08×10^{0}
	10	2.07×10^{0}	1.48×10^{0}	1.20×10^{0}	1.16×10^{0}	1.10×10^{0}	1.06×10^{0}
$\dot{\psi}_{RH}$	1	1.49×10^{0}	1.32×10^{0}	1.31×10^{0}	1.28×10^{0}	1.25×10^{0}	1.24×10^{0}
	5	1.92×10^{0}	1.47×10^{0}	1.38×10^{0}	1.25×10^{0}	1.17×10^{0}	1.14×10^{0}
	10	1.97×10^{0}	1.57×10^{0}	1.52×10^{0}	1.27×10^{0}	1.17×10^{0}	1.12×10^{0}
$\dot{\psi}_{LK}$	1	1.59×10^{0}	1.25×10^{0}	1.14×10^{0}	1.11×10^{0}	1.10×10^{0}	1.09×10^{0}
	5	1.79×10^{0}	1.47×10^{0}	1.15×10^{0}	1.09×10^{0}	1.05×10^{0}	9.94×10^{-1}
	10	1.90×10^{0}	1.57×10^{0}	1.20×10^{0}	1.11×10^{0}	1.08×10^{0}	9.87×10^{-1}
$\dot{\psi}_{RK}$	1	1.02×10^{0}	9.35×10^{-1}	9.18×10^{-1}	9.24×10^{-1}	9.16×10^{-1}	9.05×10^{-1}
	5	1.13×10^{0}	9.98×10^{-1}	9.40×10^{-1}	9.29×10^{-1}	8.64×10^{-1}	8.32×10^{-1}
	10	1.24×10^{0}	1.07×10^{0}	9.83×10^{-1}	9.63×10^{-1}	8.73×10^{-1}	8.20×10^{-1}
$\dot{\psi}_{LA}$	1	1.76×10^{0}	1.52×10^{0}	1.32×10^{0}	1.31×10^{0}	1.28×10^{0}	1.26×10^{0}
	5	2.01×10^{0}	1.63×10^{0}	1.29×10^{0}	1.24×10^{0}	1.14×10^{0}	1.10×10^{0}
	10	2.18×10^{0}	1.67×10^{0}	1.35×10^{0}	1.25×10^{0}	1.15×10^{0}	1.10×10^{0}
$\dot{\psi}_{RA}$	1	1.69×10^{0}	1.64×10^{0}	1.60×10^{0}	1.58×10^{0}	1.58×10^{0}	1.58×10^{0}
	5	1.84×10^{0}	1.75×10^{0}	1.52×10^{0}	1.48×10^{0}	1.43×10^{0}	1.38×10^{0}
	10	1.92×10^{0}	1.86×10^{0}	1.62×10^{0}	1.51×10^{0}	1.43×10^{0}	1.35×10^{0}

Table B.3: Comparison of the input–output analytic process models for the walking robot LEO in *Experiment B2*. The table shows the RMSE medians over 30 runs of the SNGP algorithm for varying number of features n_f and number of training samples n_s.

Variable	n_f	Number of training samples n_s					
		100	200	500	1000	2000	5000
ψ_{TRS}	1	5.78×10^{-2}	5.77×10^{-2}	5.71×10^{-2}	5.58×10^{-2}	5.60×10^{-2}	5.69×10^{-2}
	5	3.84×10^{-2}	3.60×10^{-2}	3.15×10^{-2}	2.65×10^{-2}	2.45×10^{-2}	2.33×10^{-2}
	10	4.07×10^{-2}	3.36×10^{-2}	2.88×10^{-2}	2.48×10^{-2}	2.09×10^{-2}	2.06×10^{-2}
ψ_{LH}	1	6.75×10^{-2}	6.57×10^{-2}	6.57×10^{-2}	5.90×10^{-2}	1.07×10^{-1}	6.67×10^{-2}
	5	3.80×10^{-2}	2.97×10^{-2}	2.62×10^{-2}	2.71×10^{-2}	2.56×10^{-2}	2.53×10^{-2}
	10	3.85×10^{-2}	3.15×10^{-2}	2.65×10^{-2}	2.64×10^{-2}	2.46×10^{-2}	2.40×10^{-2}
ψ_{RH}	1	8.04×10^{-2}	8.57×10^{-2}	6.62×10^{-2}	1.14×10^{-1}	1.06×10^{-1}	7.03×10^{-2}
	5	4.92×10^{-2}	3.81×10^{-2}	3.29×10^{-2}	3.20×10^{-2}	3.11×10^{-2}	3.04×10^{-2}
	10	5.40×10^{-2}	4.01×10^{-2}	3.25×10^{-2}	3.08×10^{-2}	2.88×10^{-2}	2.85×10^{-2}
ψ_{LK}	1	8.06×10^{-2}	5.52×10^{-2}	8.22×10^{-2}	7.52×10^{-2}	7.52×10^{-2}	5.77×10^{-2}
	5	3.66×10^{-2}	2.95×10^{-2}	2.46×10^{-2}	2.31×10^{-2}	2.20×10^{-2}	2.10×10^{-2}
	10	3.96×10^{-2}	3.09×10^{-2}	2.44×10^{-2}	2.21×10^{-2}	2.09×10^{-2}	2.04×10^{-2}
ψ_{RK}	1	8.69×10^{-2}	3.65×10^{-2}	2.99×10^{-2}	2.56×10^{-2}	8.88×10^{-2}	3.83×10^{-2}
	5	3.20×10^{-2}	2.62×10^{-2}	2.36×10^{-2}	2.26×10^{-2}	2.20×10^{-2}	2.18×10^{-2}
	10	3.49×10^{-2}	2.76×10^{-2}	2.24×10^{-2}	2.16×10^{-2}	2.08×10^{-2}	2.01×10^{-2}
ψ_{LA}	1	7.50×10^{-2}	1.07×10^{-1}	4.41×10^{-2}	1.03×10^{-1}	5.85×10^{-2}	1.03×10^{-1}
	5	5.44×10^{-2}	4.22×10^{-2}	3.27×10^{-2}	3.00×10^{-2}	2.89×10^{-2}	2.79×10^{-2}
	10	5.94×10^{-2}	4.62×10^{-2}	3.26×10^{-2}	2.96×10^{-2}	2.75×10^{-2}	2.66×10^{-2}
ψ_{RA}	1	1.19×10^{-1}	1.20×10^{-1}	9.65×10^{-2}	9.63×10^{-2}	1.02×10^{-1}	5.11×10^{-2}
	5	5.10×10^{-2}	4.45×10^{-2}	3.79×10^{-2}	3.69×10^{-2}	3.65×10^{-2}	3.60×10^{-2}
	10	5.51×10^{-2}	4.49×10^{-2}	3.79×10^{-2}	3.59×10^{-2}	3.44×10^{-2}	3.38×10^{-2}

Table B.4: Comparison of the RMSE of the state-space process models calculated on the test data set for the walking robot LEO using two variants of a deep neural network (DNN-A and DNN-B) and SNGP. The reference configuration of SNGP used for this comparison was $n_f = 10$ and $n_s = 1000$.

Variable	Method		
	DNN-A	DNN-B	SNGP
ψ_{TRS}	1.33×10^{-1}	9.27×10^{-2}	1.32×10^{-2}
ψ_{LH}	1.86×10^{-1}	1.54×10^{-1}	2.06×10^{-2}
ψ_{RH}	2.08×10^{-1}	1.23×10^{-1}	2.65×10^{-2}
ψ_{LK}	2.24×10^{-1}	1.37×10^{-1}	2.10×10^{-2}
ψ_{RK}	2.02×10^{-1}	1.10×10^{-1}	1.87×10^{-2}
ψ_{LA}	1.62×10^{-1}	1.24×10^{-1}	2.92×10^{-2}
ψ_{RA}	1.54×10^{-1}	9.36×10^{-2}	3.65×10^{-2}
$\dot{\psi}_{TRS}$	7.39×10^{-1}	6.38×10^{-1}	7.00×10^{-1}
$\dot{\psi}_{LH}$	1.13×10^{0}	1.12×10^{0}	1.16×10^{0}
$\dot{\psi}_{RH}$	1.22×10^{0}	1.20×10^{0}	1.27×10^{0}
$\dot{\psi}_{LK}$	1.08×10^{0}	1.06×10^{0}	1.11×10^{0}
$\dot{\psi}_{RK}$	9.49×10^{-1}	8.68×10^{-1}	9.63×10^{-1}
$\dot{\psi}_{LA}$	1.23×10^{0}	1.23×10^{0}	1.25×10^{0}
$\dot{\psi}_{RA}$	1.54×10^{0}	1.42×10^{0}	1.51×10^{0}

Table B.5: Comparison of analytic process models for the inverted pendulum system in *Experiment C1*. The table shows the RMSE medians over 30 runs of the SNGP (grey) and MGGP (white) algorithm for varying number of features n_f and varying number of training samples n_s.

Variable	n_f	Number of training samples n_s					
		20	50	100	200	500	1000
α	1	9.19×10^{-4}	3.80×10^{-4}	2.45×10^{-2}	2.28×10^{-3}	1.89×10^{-3}	2.38×10^{-3}
		5.51×10^{-1}	3.33×10^{-1}	4.46×10^{-1}	3.15×10^{-1}	2.44×10^{-1}	3.25×10^{-1}
	2	2.09×10^{-7}	2.39×10^{-7}	1.06×10^{-7}	1.82×10^{-9}	1.94×10^{-8}	4.60×10^{-9}
		3.94×10^{-10}	3.78×10^{-10}	3.77×10^{-10}	3.76×10^{-10}	3.76×10^{-10}	3.76×10^{-10}
	10	5.03×10^{-9}	4.44×10^{-7}	4.41×10^{-9}	1.35×10^{-9}	8.45×10^{-10}	4.56×10^{-10}
		4.29×10^{-10}	3.87×10^{-10}	3.87×10^{-10}	3.80×10^{-10}	3.77×10^{-10}	3.76×10^{-10}
$\dot{\alpha}$	1	7.97×10^{-1}	3.17×10^{-1}	2.51×10^{-1}	2.61×10^{-1}	2.34×10^{-1}	5.11×10^{-1}
		9.21×10^{-1}	3.64×10^{-1}	2.42×10^{-1}	1.52×10^{-1}	3.42×10^{-1}	2.14×10^{-1}
	4	1.12×10^{-6}	5.61×10^{-7}	1.19×10^{-6}	1.61×10^{-6}	8.17×10^{-7}	6.75×10^{-7}
		1.73×10^{-9}	1.64×10^{-9}	1.56×10^{-9}	1.55×10^{-9}	1.51×10^{-9}	1.50×10^{-9}
	10	5.16×10^{-7}	1.83×10^{-7}	2.64×10^{-7}	4.40×10^{-7}	5.15×10^{-7}	2.66×10^{-6}
		1.90×10^{-9}	1.66×10^{-9}	1.60×10^{-9}	1.56×10^{-9}	1.54×10^{-9}	1.53×10^{-9}

Table B.6: Comparison of analytic process models for the inverted pendulum system in *Experiment C2*. The table shows the comparison of the RMSE medians over 30 runs of the SNGP (grey) and MGGP (white) algorithm depending on the Gaussian noise standard deviation coefficient λ and the number of training samples n_s.

Variable	λ	Number of training samples n_s		
		20	100	1000
α	0	9.58×10^{-2}	1.79×10^{-2}	6.11×10^{-3}
		8.13×10^{-2}	1.05×10^{-2}	1.36×10^{-2}
	0.01	3.95×10^{-1}	1.45×10^{-1}	2.80×10^{-2}
		3.96×10^{-1}	1.37×10^{-1}	2.85×10^{-2}
	0.05	1.15×10^{0}	4.89×10^{-1}	1.43×10^{-1}
		8.69×10^{-1}	5.01×10^{-1}	1.42×10^{-1}
	0.1	1.90×10^{0}	7.61×10^{-1}	3.54×10^{-1}
		2.26×10^{0}	8.22×10^{-1}	3.59×10^{-1}
$\dot{\alpha}$	0	4.56×10^{0}	7.65×10^{-1}	5.04×10^{-1}
		3.89×10^{0}	7.56×10^{-1}	5.38×10^{-1}
	0.01	4.22×10^{0}	2.28×10^{0}	8.13×10^{-1}
		4.71×10^{0}	2.75×10^{0}	8.18×10^{-1}
	0.05	7.89×10^{0}	6.07×10^{0}	3.14×10^{0}
		7.39×10^{0}	6.75×10^{0}	2.76×10^{0}
	0.1	1.26×10^{1}	9.61×10^{0}	6.65×10^{0}
		1.26×10^{1}	8.99×10^{0}	6.53×10^{0}

References

[1] J. B. Rawlings, D. Q. Mayne, Model predictive control: Theory and design, Nob Hill Pub. Madison, Wisconsin, 2009.

[2] D. Q. Mayne, J. B. Rawlings, C. V. Rao, P. O. Scokaert, Constrained model predictive control: Stability and optimality, Automatica 36 (6) (2000) 789–814.

[3] T. Masters, Neural, novel and hybrid algorithms for time series prediction, John Wiley & Sons, Inc., 1995.

[4] J. Gertler, Fault detection and diagnosis, Springer, 2013.

[5] V. Venkatasubramanian, R. Rengaswamy, K. Yin, S. N. Kavuri, A review of process fault detection and diagnosis: Part i: Quantitative model-based methods, Computers & chemical engineering 27 (3) (2003) 293–311.

[6] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[7] 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, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.

[8] S. Gu, T. P. Lillicrap, I. Sutskever, S. Levine, Continuous deep q-learning with model-based acceleration, CoRR abs/1603.00748 (2016).

[9] J. Peters, S. Schaal, Policy gradient methods for robotics, in: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, pp. 2219–2225. doi:10.1109/IROS.2006.282564.

[10] J. Kober, J. Peters, Reinforcement learning in robotics: A survey, in: M. Wiering, M. van Otterlo (Eds.), Reinforcement Learning: State-of-the-Art, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 579–610. doi:10.1007/978-3-642-27645-3_18.

[11] L. Kuvayev, R. S. Sutton, Model-based reinforcement learning with an approximate, learned model, Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems (1996) 101–105.

[12] J. Forbes, D. Andre, Representations for learning control policies, in: Proc. 19th Int. Conf. Mach. Learn. Workshop Develop. Represent., 2002, pp. 7–14.

[13] T. Hester, P. Stone, Intrinsically motivated model learning for developing curious robots, Artif. Intell. 247 (C) (2017) 170–186. doi:10.1016/j.artint.2015.05.002.

[14] N. K. Jong, P. Stone, Model-based function approximation in reinforcement learning, in: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '07, ACM, New York, NY, USA, 2007, pp. 95:1–95:8. doi:10.1145/1329125.1329242.

[15] R. S. Sutton, Dyna, an integrated architecture for learning, planning, and reacting, SIGART Bull. 2 (4) (1991) 160–163. doi:10.1145/122344.122377.

[16] S. Levine, P. Abbeel, Learning neural network policies with guided policy search under unknown dynamics, in: Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 1071–1079.

[17] R. Lioutikov, A. Paraschos, J. Peters, G. Neumann, Sample-based information-theoretic stochastic optimal control, in: Proceedings of 2014 IEEE International Conference on Robotics and Automation, IEEE, 2014, pp. 3896–3902.

[18] M. Deisenroth, C. E. Rasmussen, PILCO: A model-based and data-efficient approach to policy search, in: International Conference on Machine Learning (ICML), 2011, p. 465472.

[19] J. Boedecker, J. T. Springenberg, J. Wülfing, M. Riedmiller, Approximate real-time optimal control based on sparse gaussian process models, in: 2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2014, pp. 1–8. doi:10.1109/ADPRL.2014.7010608.

[20] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, E. Liang, Autonomous inverted helicopter flight via reinforcement learning, in: M. H. Ang, O. Khatib (Eds.), Experimental Robotics IX: The 9th International Symposium on Experimental Robotics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 363–372. doi:10.1007/11552246_35.

[21] R. Munos, A. Moore, Variable resolution discretization in optimal control, Machine learning 49 (2) (2002) 291–323.

[22] L. Buşoniu, D. Ernst, B. De Schutter, R. Babuška, Cross-entropy optimization of control policies with adaptive basis functions, IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 41 (1) (2011) 196–209.

[23] D. Ernst, P. Geurts, L. Wehenkel, Tree-based batch mode reinforcement learning, Journal of Machine Learning Research 6 (2005) 503–556.

[24] S. Lange, M. Riedmiller, 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, 2012, pp. 1–8.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning arxiv.org/abs/1312.5602 (2013).

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

[27] N. Heess, G. Wayne, D. Silver, T. P. Lillicrap, Y. Tassa, T. Erez, Learning continuous control policies by stochastic value gradients, CoRR abs/1510.09142 (2015).

[28] T. de Bruin, J. Kober, K. Tuyls, R. Babuška, Integrating state representation learning into deep reinforcement learning, IEEE Robotics and Automation Letters 3 (3) (2018) 1394–1401.

[29] T. de Bruin, J. Kober, K. Tuyls, R. Babuška, Experience selection in deep reinforcement learning for control, Journal of Machine Learning Research 19 (9) (2018) 1–56.
URL http://jmlr.org/papers/v19/17-131.html

[30] C. G. Atkeson, A. W. Moore, S. Schaal, Locally weighted learning, Artificial Intelligence Review 11 (1-5) (1997) 11–73.

[31] I. Grondman, M. Vaandrager, L. Buşoniu, R. Babuška, E. Schuitema, Efficient model learning methods for actor–critic control, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 42 (3) (2012) 591–602.

[32] R. Lieck, M. Toussaint, Temporally extended features in model-based reinforcement learning with partial observability, Neurocomputing 192 (2016) 49–60.

[33] M. Schmidt, H. Lipson, Distilling free-form natural laws from experimental data, Science 324 (5923) (2009) 81–85.

[34] N. Staelens, D. Deschrijver, E. Vladislavleva, B. Vermeulen, T. Dhaene, 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 99 (2012) 1–12.
[35] C. Brauer, Using Eureqa in a Stock Day-Trading Application, cypress Point Technologies, LLC (2012).
[36] E. Vladislavleva, T. Friedrich, F. Neumann, M. Wagner, Predicting the energy output of wind farms based on weather data: Important variables and their correlation, Renewable Energy 50 (2013) 236–243.
[37] J. Žegklitz, P. Pošík, Linear combinations of features as leaf nodes in symbolic regression, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '17, ACM, New York, NY, USA, 2017, pp. 145–146. doi:10.1145/3067695.3076009.
URL http://doi.acm.org/10.1145/3067695.3076009
[38] M. Onderwater, S. Bhulai, R. van der Mei, Value function discovery in markov decision processes with evolutionary algorithms, IEEE Transactions on Systems, Man, and Cybernetics: Systems 46 (9) (2016) 1190–1201. doi:10.1109/TSMC.2015.2475716.
[39] E. Alibekov, J. Kubalík, R. Babuška, Symbolic method for deriving policy in reinforcement learning, in: 2016 IEEE 55th Conference on Decision and Control (CDC), 2016, pp. 2789–2795. doi:10.1109/CDC.2016.7798684.
[40] J. Branke, S. Greco, R. SÅowiÅski, P. Zielniewicz, Learning value functions in interactive evolutionary multiobjective optimization, IEEE Transactions on Evolutionary Computation 19 (1) (2015) 88–102. doi:10.1109/TEVC.2014.2303783.
[41] J. Kubalík, E. Alibekov, R. Babuška, Optimal control via reinforcement learning with symbolic policy approximation, IFAC-PapersOnLine 50 (1) (2017) 4162 – 4167, 20th IFAC World Congress. doi:https://doi.org/10.1016/j.ifacol.2017.08.805.
URL http://www.sciencedirect.com/science/article/pii/S2405896317312594
[42] E. Derner, J. Kubalík, R. Babuška, Data-driven construction of symbolic process models for reinforcement learning, in: Proceedings IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 2018.
[43] E. Derner, J. Kubalík, R. Babuška, Reinforcement learning with symbolic input-output models, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 3004–3009. doi:10.1109/IROS.2018.8593881.
[44] I. Arnaldo, K. Krawiec, U.-M. O'Reilly, Multiple regression genetic programming, in: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO '14, ACM, New York, NY, USA, 2014, pp. 879–886. doi:10.1145/2576768.2598291.
URL http://doi.acm.org/10.1145/2576768.2598291
[45] I. Arnaldo, U.-M. O'Reilly, K. Veeramachaneni, Building predictive models via feature synthesis, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO '15, ACM, New York, NY, USA, 2015, pp. 983–990. doi:10.1145/2739480.2754693.
URL http://doi.acm.org/10.1145/2739480.2754693
[46] D. Jackson, A new, node-focused model for genetic programming, in: A. Moraglio, S. Silva, K. Krawiec, P. Machado, C. Cotta (Eds.), Genetic Programming: 15th European Conference, EuroGP 2012, Málaga, Spain, April 11-13, 2012. Proceedings, Springer, Berlin, Heidelberg, 2012, pp. 49–60.
[47] D. Jackson, Single node genetic programming on problems with side effects, in: C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, M. Pavone (Eds.), Parallel Problem Solving from Nature - PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I, Springer, Berlin, Heidelberg, 2012, pp. 327–336.
[48] J. Kubalík, E. Derner, R. Babuška, Enhanced symbolic regression through local variable transformations, in: Proceedings of the 9th International Joint Conference on Computational Intelligence, 2017, pp. 91–100. doi:10.5220/0006505200910100.
[49] J. Kubalík, J. Zegklitz, E. Derner, R. Babuska, Symbolic regression methods for reinforcement learning, CoRR abs/1903.09688 (2019). arXiv:1903.09688.
URL http://arxiv.org/abs/1903.09688
[50] 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. doi:10.1007/978-3-319-20883-1_22.
URL https://doi.org/10.1007/978-3-319-20883-1_22
[51] J. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems), MIT Press Ltd, 1992.
[52] J. Faigl, J. Chudoba, K. Košnar, M. Kulich, M. Saska, L. Přeučil, Syrotek - a robotic system for education, AT&P journal PLUS 2 (2010) 31–36.
[53] E. Schuitema, M. Wisse, T. Ramakers, P. Jonker, The design of LEO: a 2D bipedal walking robot for online autonomous reinforcement learning, in: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, IEEE, 2010, pp. 3238–3243.
[54] I. Koryakovskiy, H. Vallery, R. Babuška, W. Caarls, Evaluation of physical damage associated with action selection strategies in reinforcement learning, IFAC-PapersOnLine 50 (1) (2017) 6928–6933.
[55] M. L. Felis, RBDL: an efficient rigid-body dynamics library using recursive algorithms, Autonomous Robots (2016) 1–17doi:10.1007/s10514-016-9574-0.
URL http://dx.doi.org/10.1007/s10514-016-9574-0
[56] W. Caarls, Generic Reinforcement Learning Library, https://github.com/wcaarls/grl (2018).
[57] J. Kiefer, J. Wolfowitz, Stochastic estimation of the maximum of a regression function, Ann. Math. Statist. 23 (3) (1952) 462–466. doi:10.1214/aoms/1177729392.
URL https://doi.org/10.1214/aoms/1177729392
[58] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
[59] H. J. Tulleken, Generalized binary noise test-signal concept for improved identification-experiment design, Automatica 26 (1) (1990) 37 – 49. doi:https://doi.org/10.1016/0005-1098(90)90156-C.
[60] L. Buşoniu, D. Ernst, R. Babuška, B. De Schutter, Approximate dynamic programming with a fuzzy parameterization, Automatica 46 (5) (2010) 804–814.