Symbolic Regression for Constructing Analytic Models in Reinforcement Learning

Erik Derner ERIK.DERNER@CVUT.CZ
Czech Institute of Informatics, Robotics, and Cybernetics
and
Department of Control Engineering, Faculty of Electrical Engineering
Czech Technical University in Prague
Prague, 16000, Czech Republic

Jiří Kubalík JIRI.KUBALIK@CVUT.CZ
Czech Institute of Informatics, Robotics, and Cybernetics
Czech Technical University in Prague
Prague, 16000, Czech Republic

Nicola Ancona N.ANCONA93@GMAIL.COM
Cognitive Robotics
Delft University of Technology
Delft, 2628 CD, The Netherlands

Robert Babuška R.BABUSKA@TUDELFT.NL
Cognitive Robotics
Delft University of Technology
Delft, 2628 CD, The Netherlands
and
Czech Institute of Informatics, Robotics, and Cybernetics
Czech Technical University in Prague
Prague, 16000, Czech Republic

Abstract

Reinforcement learning (RL) is a widely used approach for controlling systems with unknown or time-varying dynamics. Even though RL does not require a model of the system, it is known to be faster and safer when using models learned online. We propose to employ symbolic regression (SR) to construct parsimonious process models described by analytic equations for real-time RL control. We have tested our method with two different state-of-the-art SR algorithms which automatically search for equations that fit the measured data. 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 biped 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 RL to successfully perform the swing-up task, based on a model constructed from only 100 data samples.

Keywords: Model learning for control, symbolic regression, reinforcement learning, optimal control, genetic programming.

1. Introduction

In reinforcement learning, the task is to find an optimal policy maximizing the long-term cumulative reward. The policy is found through interactions with the environment. Even though model-free RL algorithms are available, the absence of a model slows down convergence and leads to extensive learning times (Gu et al., 2016; Peters and Schaal, 2006; Kober and Peters, 2012). Various model-based methods have been proposed to speed up learning (Kuvayev and Sutton, 1996; Forbes and Andre, 2002; Hester and Stone, 2017; Jong and Stone, 2007; Sutton, 1991). To that end, many model-learning approaches are available: time-varying linear models (Levine and Abbeel, 2014; Lioutikov et al., 2014), Gaussian processes (Deisenroth and Rasmussen, 2011; Boedecker et al., 2014) and other probabilistic models (Ng et al., 2006), basis function expansions (Munos and Moore, 2002; Buşoniu et al., 2011), regression trees (Ernst et al., 2005), deep neural networks (Lange et al., 2012; Mnih et al., 2013, 2015; Lillicrap et al., 2015; Heess et al., 2015; de Bruin et al., 2018b,a) or local linear regression (Atkeson et al., 1997; Grondman et al., 2012; Lieck and Toussaint, 2016).

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 (Schmidt and Lipson, 2009; Staelens et al., 2012; Brauer, 2012; Vladislavleva et al., 2013; Žegklitz and Pošík, 2017).

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 within the RL domain. For instance, modeling of the value function by means of genetic programming is presented in Onderwater et al. (2016), where analytic descriptions of the value function are obtained based on data sampled from the optimal value function. Another example is the work of Alibekov et al. (2016), 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 Branke et al. (2015), 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 Kubalík et al. (2017a), 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 reinforcement learning. We contend that the use of SR for model learning is a valuable element missing from the current RL schemes and we demonstrate its usefulness. We build on our previous research (Derner et al., 2018a), (Derner et al., 2018b), which indicated that SR is a suitable tool for this task, as 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 biped 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. We demonstrate that our method is not dependent on the particular choice of the SR algorithm by testing it with two different state-of-the-art genetic programming methods.

The paper is organized as follows. Sections 2 and 3 present the theoretical background of the reinforcement learning framework and the proposed method. The experimental evaluation of the method is reported in Section 4 and the conclusions are drawn in Section 5.

2. Reinforcement Learning Preliminaries

The system for which an optimal control strategy is to be learnt is commonly described in discrete time by the nonlinear state-space model

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

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

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:

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

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 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 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:

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

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 superscripts; $V(x)$ will therefore denote 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 (5):

$$u = \underset{u' \in U}{\operatorname{argmax}}\ \left[\rho(x, u', f(x, u')) + \gamma\, V(f(x, u'))\right]\,. \tag{6}$$

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

In the standard reinforcement learning (RL) formulation, the system to be controlled is described by the state-transition function (1). However, for a vast majority of processes, the full state vector cannot be directly measured and therefore a state estimator would have to be used. In the absence of an accurate process model (which is usually the main reason for using RL), such a reconstruction is inaccurate and has a negative effect on the overall performance of the RL 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:

$$y_{k+1} = g\left(y_k, y_{k-1}, \ldots, y_{k-n_y+1}, u_k, u_{k-1}, \ldots, u_{k-n_u+1}\right), \tag{7}$$

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 (7) as:

$$y_{k+1} = g\left(\varphi_k, u_k\right). \tag{8}$$

The reward function, the return, the value function and the optimal control action (2)–(6) are defined analogously using the regressor φ instead of the state variable x.

Note that in this setting, the model function and also the policy has to be found from data samples living 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 creates a problem for many types of approximators. For instance, basis 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.

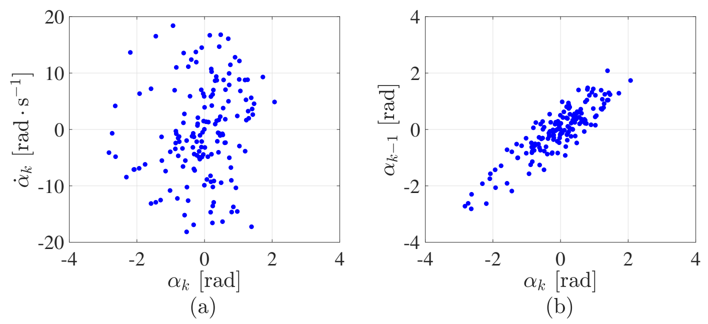


Figure 1: An example of trajectory samples obtained from the real inverted pendulum (see Section 4.3) in the original state space (a), and in the space formed by the current and previous output (b).

3. Method

In this section, we explain the principle of our method within the model-based reinforcement learning scheme, we describe how we collect the data and we discuss the computational complexity of our approach.

3.1 Model-Based Reinforcement Learning Scheme

The RL scheme we use is depicted in Figure 2 and it consists of the following elements:

1. Low-level control loop with the RL controller. This controller can be either the policy (6) based on the V-function learnt offline using value iteration, or it can be any model-based RL algorithm whose parameters are adjusted online. Examples of such algorithms are the standard DYNA (Sutton and Barto, 1998; Caarls and Schuitema, 2016) or more recent ones, such as MLAC (Grondman et al., 2012).

2. Input-state samples of the form $[u_k, x_k]$ are collected in the data buffer. For the input–output model (8), the state is replaced by the vector φ_k of past inputs and outputs.

3. Symbolic regression is periodically applied to the available data set, yielding a model of the process described by analytic equations. This model can be used in several ways: a) offline to (re)compute the value function and policy, b) online by the RL algorithm to assist the parameter update (Grondman et al., 2012), c) online to generate transition samples which can be used in the parallel DYNA setting (Caarls and Schuitema, 2016), among other possibilities. Note that also the data stored in the buffer can be reused by the online RL algorithm, using experience replay (de Bruin et al., 2018a).

In the sequel we describe the application of symbolic regression to the process data collected online and the evaluation of the models obtained.

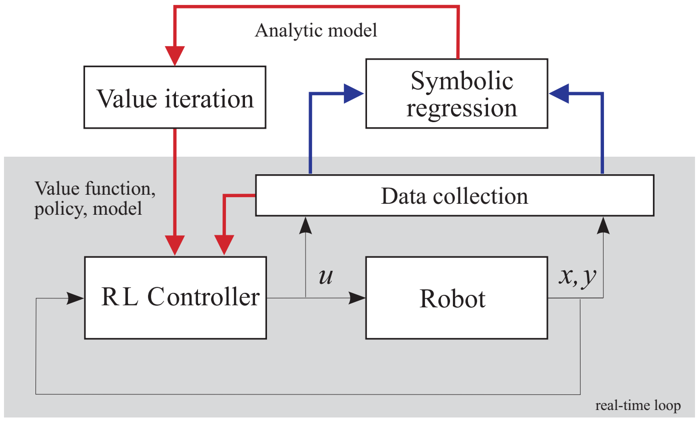


Figure 2: Model-based reinforcement learning scheme: the RL control loop and the data logging in the buffer run in real time, symbolic regression and value iteration are computed offline in a parallel process.

3.2 Data Collection

Two different situations can be distinguished: initial model learning and model learning under a given policy.

3.2.1 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 (Tulleken, 1990). 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. In the experiments section we detail the choice of these parameters for our examples.

3.2.2 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 usually has a lower amplitude.

Each data set is divided into a training set and a test set. The training set is used to construct models by means of symbolic regression, whereas the test set serves to rank the models according to their predictive power based on the root mean square error (RMSE).

3.3 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 (7). 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 (Arnaldo et al., 2014, 2015). Therefore, we define the class of analytic state-space models as:

$$f(x,u) = \sum_{i}^{n_f} \beta_i f_i(x,u) \tag{9}$$

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

$$g(\varphi,u) = \sum_{i}^{n_f} \beta_i g_i(\varphi,u)\,. \tag{10}$$

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 functions. The coefficients β_i are estimated by least squares.

In this work, we use two different genetic algorithms: a modified version of Single Node Genetic Programming (SNGP) (Jackson, 2012a,b; Kubalík et al., 2017b) and a modified version of Multi-Gene Genetic Programming (MGGP) (Žegklitz and Pošík, 2017). Detailed explanation of these algorithms and their parameters is beyond the scope of this paper and we refer the interested reader to Kubalík et al. (2017b) and Žegklitz and Pošík (2017).

3.4 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 a two-dimensional regressor, 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

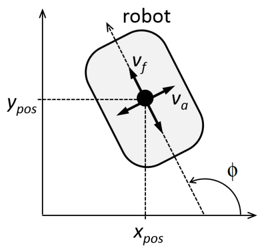
We have carried out experiments with three nonlinear systems: a mobile robot, a 1-DOF inverted pendulum and a bipedal walking robot. The data, the codes and the detailed configuration of the experiments is available in our repository[1].

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). 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). On this example, we demonstrate the ability of the method to construct standard state-space models as well as input–output (NARX) models. We conclude our set of experiments with the inverted pendulum system (Section 4.3). In addition to simulations, we perform experiments with a lab setup to evaluate the performance of the algorithm in real-world conditions.

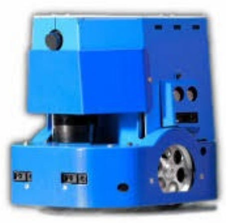
For the walking robot and for the inverted pendulum system, we compare our modeling results with alternative methods: deep neural networks and in the latter case also with local linear regression.

4.1 Mobile Robot

The state of a two-wheel mobile robot, see Figure 3 and Faigl et al. (2010), 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 3: Mobile robot schematic (a) and photograph (b).

The continuous-time dynamic model of the robot is:

$$\begin{aligned} \dot{x}_{pos} &= v_f \cos(\phi), \\ \dot{y}_{pos} &= v_f \sin(\phi), \\ \dot{\phi} &= v_a \,. \end{aligned} \tag{11}$$

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

4.1.1 Data Sets

We generated a noise-free data set by using the Euler method to simulate the differential equations (11). With a sampling period $T_s = 0.05\,\mathrm{s}$, the discrete-time approximation of (11) becomes::

$$\begin{aligned} x_{pos,\,k+1} &= x_{pos,\,k} + 0.05\,v_{f,\,k}\,\cos(\phi), \\ y_{pos,\,k+1} &= y_{pos,\,k} + 0.05\,v_{f,\,k}\,\sin(\phi), \\ \phi_{k+1} &= \phi_k + 0.05\,v_{a,\,k}\,. \end{aligned} \tag{12}$$

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:

$$\begin{aligned} x_{pos} &\in [-1,1]\,\mathrm{m}, \\ y_{pos} &\in [-1,1]\,\mathrm{m}, \\ \phi &\in [-\pi,\pi]\,\mathrm{rad}, \\ v_f &\in [-1,1]\,\mathrm{m}\cdot\mathrm{s}^{-1}, \\ v_a &\in \left[-\frac{\pi}{2},\frac{\pi}{2}\right]\,\mathrm{rad}\cdot\mathrm{s}^{-1}. \end{aligned} \tag{13}$$

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 (13). 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 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 $n_f \in \{1, 2, 10\}$. The parameters of the MGGP algorithm were set to achieve a comparable performance relative to SNGP, taking into account the differences between the 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 1 in Appendix A.

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

$$\begin{aligned}
\hat{x}_{pos,\,k+1} &= 1.0\, x_{pos,\,k} + 0.0499998879\, v_{f,\,k} \cos(\phi_k)\,, \\
\hat{y}_{pos,\,k+1} &= 1.000000023\, y_{pos,\,k} + 0.0500000056\, v_{f,\,k} \sin(\phi_k) + 0.0000000191\,, \\
\hat{\phi}_{k+1} &= 0.9999982931\, \phi_k + 0.0500000536\, v_{a,\,k} - 0.0000059844\,.
\end{aligned} \tag{14}$$

The coefficients are rounded to 10 decimal digits in order to demonstrate the magnitude of the error compared to the original Euler approximation (12). The results show that even with a very 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 (9). 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 3.3. 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 (Schuitema et al., 2010), see Figure 4. 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

$$\psi = [\psi_{TRS}, \psi_{LH}, \psi_{RH}, \psi_{LK}, \psi_{RK}, \psi_{LA}, \psi_{RA}]^\top \tag{15}$$

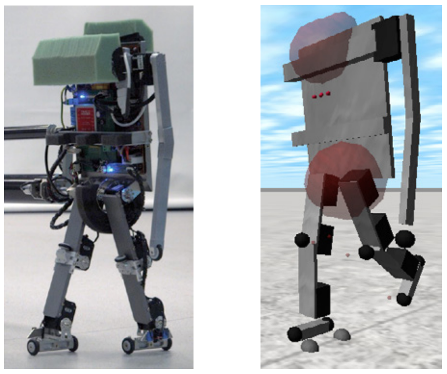
represents the angles of the torso, left and right hip, the knee and the ankle. Likewise,

$$\dot{\psi} = [\dot{\psi}_{TRS}, \dot{\psi}_{LH}, \dot{\psi}_{RH}, \dot{\psi}_{LK}, \dot{\psi}_{RK}, \dot{\psi}_{LA}, \dot{\psi}_{RA}]^\top \tag{16}$$

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.

4.2.1 Data Sets

In order to apply symbolic regression, the walking robot LEO was modeled using the Rigid Body Dynamics Library (RBDL) (Felis, 2016) and the data sets were generated using the Generic Reinforcement Learning Library (GRL) (Caarls, 2018), which allowed us to record trajectories while the robot was learning to walk.



(a) (b)

Figure 4: The walking robot LEO: photograph (a) and simulation model rendering (b). (Koryakovskiy et al., 2017).

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 2 and for the input–output models in Table 3 in Appendix A.

The results show the expected trend, which can be seen in all experiments, that the quality of the models grows 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 even on batches of data as small as 100 samples 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.

An example of a state-space analytic model found in the configuration with 5 features and 1000 samples is:

$$\begin{aligned}
\hat{\psi}_{LH,k+1} = {} & 1.6 \times 10^{-2} u_{LH,k} + 3.8 \times 10^{-3} u_{RH,k} - 4.9 \times 10^{-3} u_{LK,k} + 1.9 \times 10^{-3} u_{RK,k} \\
& - 8.5 \times 10^{-4} u_{LA,k} + 3.5 \times 10^{-4} u_{RA,k} + 1.5 \times 10^{-2} \psi_{TRS,k} - 1.3 \times 10^{-3} \psi_{LH,k} \\
& - 1.3 \times 10^{-3} \psi_{RH,k} - 2.7 \times 10^{-3} \psi_{LK,k} - 1.5 \times 10^{-2} \psi_{RK,k} + 1.9 \times 10^{-2} \psi_{LA,k} \\
& - 1.2 \times 10^{-2} \psi_{RA,k} + 7.9 \times 10^{-3} \dot{\psi}_{TRS,k} + 1.1 \times 10^{-2} \dot{\psi}_{LH,k} - 7 \times 10^{-3} \dot{\psi}_{RH,k} \\
& + 4 \times 10^{-3} \dot{\psi}_{LK,k} - 4.3 \times 10^{-3} \dot{\psi}_{RK,k} + 8.1 \times 10^{-4} \dot{\psi}_{LA,k} - 1.2 \times 10^{-3} \dot{\psi}_{RA,k} \\
& - 3.4 \times 10^{-3} \cos(0.7 u_{LK,k} - 6.8 u_{RH,k} - 8.4 u_{LH,k} - 2.2 u_{RK,k} + 2.8 u_{RA,k} + 0.5 \psi_{TRS,k} \\
& - 1.2 \psi_{LH,k} + 0.5 \psi_{RH,k} + 24 \psi_{LK,k} + 1.5 \psi_{RK,k} - \psi_{LA,k} - 14 \psi_{RA,k} + 0.3 \dot{\psi}_{TRS,k} + 0.5 \dot{\psi}_{LH,k} \\
& + 2.3 \dot{\psi}_{RH,k} - 8 \dot{\psi}_{LK,k} - 1.5 \dot{\psi}_{RK,k} + 0.7 \dot{\psi}_{LA,k} + 2.3 \dot{\psi}_{RA,k} - 10) - 6.6 \times 10^{-4} \sin(0.5 u_{RH,k} \\
& + 8 u_{LK,k} + 0.3 u_{RK,k} + 3.3 \times 10^{-2} u_{LA,k} + 8.6 u_{RA,k} - \psi_{TRS,k} + 4 \psi_{LH,k} + 28 \psi_{RH,k} - 7 \psi_{LK,k} \\
& + 89 \psi_{RK,k} + 5.2 \psi_{LA,k} + 8 \psi_{RA,k} + 0.7 \dot{\psi}_{TRS,k} + 0.3 \dot{\psi}_{LH,k} + 0.3 \dot{\psi}_{RH,k} - 0.4 \dot{\psi}_{LK,k} + 4.6 \dot{\psi}_{RK,k} \\
& + 7.8 \dot{\psi}_{LA,k} + \dot{\psi}_{RA,k} + 4) - 3.6 \times 10^{-3} \psi_{LH,k} (\sin(7.5 \times 10^{-2} u_{LH,k} - 2.3 u_{RH,k} + 0.4 u_{LK,k} \\
& + 9 u_{RK,k} + 24 u_{LA,k} - 1.9 u_{RA,k} + 0.8 \psi_{TRS,k} - 4.7 \psi_{LH,k} + 0.8 \psi_{RH,k} - 0.7 \psi_{LK,k} + 399 \psi_{RK,k} \\
& + 4.5 \psi_{LA,k} + 9.2 \psi_{RA,k} + 3 \dot{\psi}_{TRS,k} + 1.6 \dot{\psi}_{RH,k} - 0.4 \dot{\psi}_{LK,k} - 2.3 \dot{\psi}_{RK,k} - 0.1 \dot{\psi}_{LA,k} \\
& + 300 \dot{\psi}_{RA,k} - 24) - 288) - 1.4 \times 10^{-7} (29 u_{LH,k} + 3.9 u_{RH,k} - 8 u_{LK,k} + 3.3 u_{RK,k} - 1.3 u_{LA,k} \\
& + 0.8 u_{RA,k} + 25 \psi_{TRS,k} - 6 \psi_{LH,k} + 2.8 \psi_{RH,k} - 12 \psi_{LK,k} - 21 \psi_{RK,k} + 22 \psi_{LA,k} - 20 \psi_{RA,k} \\
& + 8.6 \dot{\psi}_{TRS,k} + 17 \dot{\psi}_{LH,k} - 9.8 \dot{\psi}_{RH,k} + 5.5 \dot{\psi}_{LK,k} - 6.7 \dot{\psi}_{RK,k} + 1.3 \dot{\psi}_{LA,k} - 1.5 \dot{\psi}_{RA,k} + 411)^2 \\
& + 1.5 \times 10^{-6} (11 u_{LH,k} - 4 u_{RH,k} - u_{LK,k} + 0.8 u_{RK,k} + 0.6 u_{RA,k} + 4.7 \psi_{TRS,k} - 8.4 \psi_{LH,k} \\
& + 9.9 \psi_{RH,k} - 16 \psi_{LK,k} + 5.5 \psi_{RK,k} - 15 \psi_{LA,k} - 4.9 \psi_{RA,k} - 7.2 \dot{\psi}_{TRS,k} + 0.6 \dot{\psi}_{LH,k} \\
& + 1.6 \dot{\psi}_{RH,k} - 1.4 \dot{\psi}_{LK,k} - 0.2 \dot{\psi}_{RK,k} + 0.2 \dot{\psi}_{LA,k} + 0.5 \dot{\psi}_{RA,k} + 0.4)^2 - 6.5 \times 10^{-3} \psi_{LH,k}^2 \\
& + 4.2 \times 10^{-3} \; .
\end{aligned} \tag{17}$$

This example demonstrates that the models generated by the proposed method are analytically tractable and easy to plug in other algorithms. The number of parameters in the example is 130, which is low compared to deep neural networks, as illustrated in the following comparison.

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 and an output layer with dimensions 200×14. ReLU activation function was used after each linear layer with the exception of the output layer. The batch size was set to 32. The SGD algorithm (Kiefer and Wolfowitz, 1952) 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 (Kingma and Ba, 2014) 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 5 shows an overview of the performance of the two variants of DNN compared to the SNGP algorithm and detailed results are presented in Table 4 in Appendix A. 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.

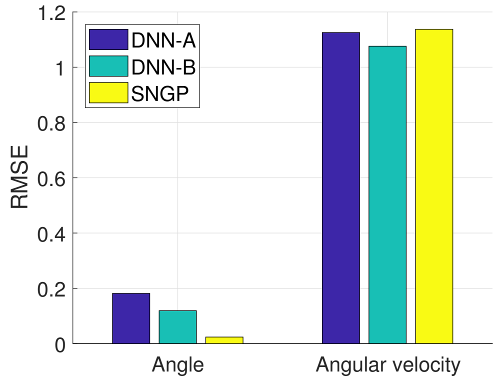


Figure 5: Comparison of two DNN variants with the SNGP algorithm on the walking robot example. The bars show the mean RMSE over the 7 angles and over the 7 angular velocities on the test data set.

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 6(a). 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:

$$\ddot{\alpha} = \frac{1}{J} \cdot \left(\frac{K}{R}\, u - m\, g\, l\, \sin(\alpha) - b\, \dot{\alpha} - \frac{K^2}{R}\, \dot{\alpha} - c\, \text{sign}(\dot{\alpha}) \right) \tag{18}$$

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.

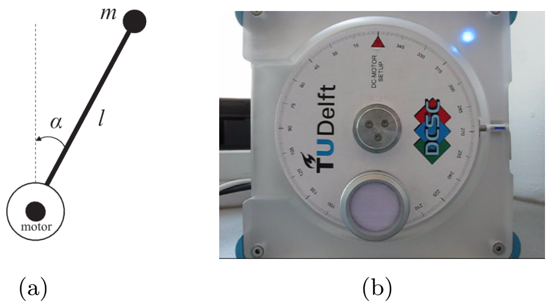


Figure 6: Inverted pendulum schematic (a) and the real inverted pendulum system (b).

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

$$\rho(x_k, u_k, x_{k+1}) = -0.5|\alpha_r - \alpha_k| - 0.01|\dot{\alpha}_r - \dot{\alpha}_k| - 0.05|u_k|, \tag{19}$$

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

4.3.1 Data Sets

We used both simulated and real measured data in the experiments with the inverted pendulum system. 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 (18):

$$\begin{aligned} \alpha_{k+1} &= \alpha_k + 0.05\,\dot{\alpha}_k, \\ \dot{\alpha}_{k+1} &= 0.9102924564\,\dot{\alpha}_k - 0.2369404025\,\mathrm{sign}(\dot{\alpha}_k) \\ &\quad + 1.5727561084\,u_k - 6.3168590065\,\sin(\alpha_k). \end{aligned} \tag{20}$$

The data set for *Experiment C2* was created by integrating (18) 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

$$\begin{aligned} \alpha_n &= \alpha + \pi\lambda r_{n,1}\,, \\ \dot{\alpha}_n &= \dot{\alpha} + 40\lambda r_{n,2}\,, \end{aligned} \tag{21}$$

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 are chosen so that the added noise is approximately proportional to the range of each variable.

Both in *Experiment 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: $\alpha \in [-\pi, \pi]\,\mathrm{rad}$, $\dot{\alpha} \in [-40, 40]\,\mathrm{rad} \cdot \mathrm{s}^{-1}$ 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 6(b). At first, the system was excited by applying a uniformly distributed random control input u_k within the range $[-5, 5]\,\mathrm{V}$ at each time step k. The random interaction with the system lasted 5 seconds and the recorded data set comprised 100 samples. The data are shown in Figure 7. The data set was later enriched by samples recorded while applying policy (6) to perform the swing-up task on the real system, which will be described in detail in the following section.

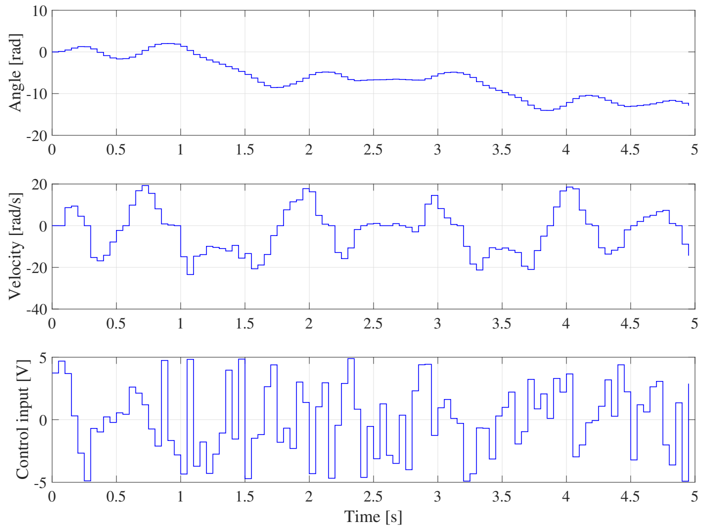


Figure 7: Initial data set obtained on the real inverted pendulum system as a response to the random input shown in the bottom panel.

The sequences recorded for *Experiment C3* were split into training and test subsets. Every third sample was used for the test set, while the rest of the samples formed the training set.

In all experiments, the reported RMSE values were calculated on the respective test data set.

4.3.2 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 $u \in [-2, 2]\,\mathrm{V}$, 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 Section 2). To find an approximation of the optimal value function, we used the fuzzy V-iteration algorithm (Buşoniu et al., 2010). 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 $\{*, +, -, \sin, \cos, \mathrm{sign}\}$. 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 accordingly to achieve a comparable performance as the SNGP algorithm.

4.3.3 Results

The results of *Experiment C1* are summarized in Table 5 in Appendix A 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:

$$\begin{aligned}\hat{\alpha}_{k+1} &= \alpha_k + 0.05\,\dot{\alpha}_k - 0.0000000001\,,\\ \hat{\dot{\alpha}}_{k+1} &= 0.9102924745\,\dot{\alpha}_k - 0.2369403835\,\mathrm{sign}(\dot{\alpha}_k) + 1.5727561072\,u_k\\ &\quad - 6.3168589936\,\sin(\alpha_k) + 0.0000000013\,.\end{aligned} \tag{22}$$

The error of the analytic model w.r.t. the Euler approximation (20) 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 6 in Appendix A 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 10(a). 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 10(b) 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 ($p = 2 \times 10^{-22}$). 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.

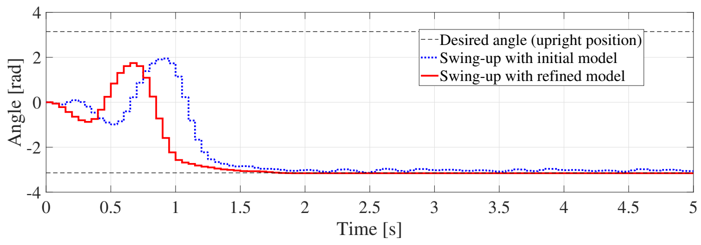


Figure 8: Comparison of the real swing-up response with the initial model, learnt from the random data, and the refined model, learnt from the random data merged with additional data from eight real swing-up experiments.

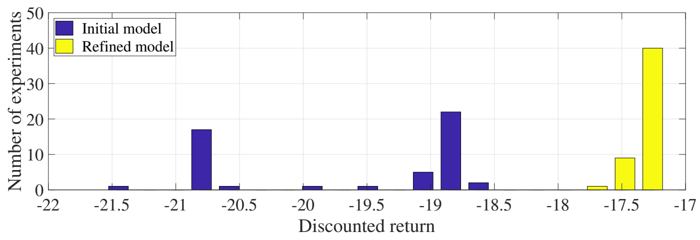
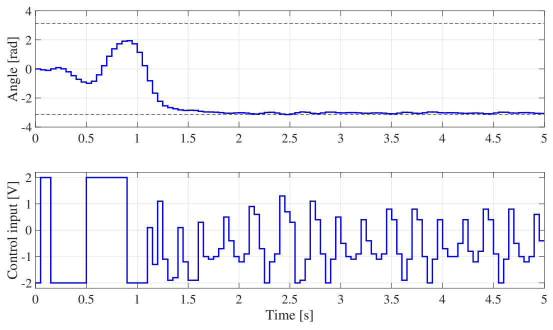
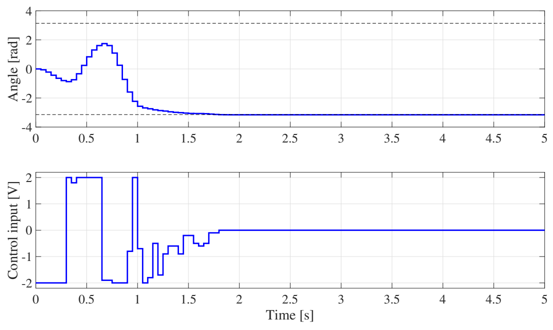


Figure 9: Histograms of 50 real experiments with the initial model and the refined model measured by the discounted return. The performance improvement is statistically significant ($p = 2 \times 10^{-22}$).



(a)



(b)

Figure 10: A typical real swing-up experiment with the initial model (a) and the refined model (b).

4.3.4 Comparison with Alternative Methods

We compared our modeling results with local linear regression (LLR) (Atkeson et al., 1997). 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 4.3.1 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 suitable 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 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), 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. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

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.

The appendix presents detailed results of the experiments described in Section 4.

Table 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 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 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 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 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 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

E. Alibekov, J. Kubalík, and R. Babuška. Symbolic method for deriving policy in reinforcement learning. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 2789–2795, Dec 2016. doi: 10.1109/CDC.2016.7798684.

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*, GECCO '14, pages 879–886, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2662-9. doi: 10.1145/2576768.2598291. URL `http://doi.acm.org/10.1145/2576768.2598291`.

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*, GECCO '15, pages 983–990, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3472-3. doi: 10.1145/2739480.2754693. URL `http://doi.acm.org/10.1145/2739480.2754693`.

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

J. Boedecker, J. T. Springenberg, J. Wülfing, and 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)*, pages 1–8, Dec 2014. doi: 10.1109/ADPRL.2014.7010608.

J. Branke, S. Greco, R. Sowiski, and P. Zielniewicz. Learning value functions in interactive evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 19(1):88–102, Feb 2015. ISSN 1089-778X. doi: 10.1109/TEVC.2014.2303783.

C. Brauer. Using Eureqa in a Stock Day-Trading Application. Cypress Point Technologies, LLC, 2012.

L. Buşoniu, D. Ernst, R. Babuška, and B. De Schutter. Approximate dynamic programming with a fuzzy parameterization. *Automatica*, 46(5):804–814, 2010.

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*, 41(1):196–209, 2011.

W. Caarls. Generic Reinforcement Learning Library, 2018. `https://github.com/wcaarls/grl`.

W. Caarls and E. Schuitema. Parallel online temporal difference learning for motor control. *IEEE Transactions on Neural Networks and Learning Systems*, 27(7):1457–1468, July 2016. ISSN 2162-237X. doi: 10.1109/TNNLS.2015.2442233.

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

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

M. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, page 465472, 2011.

E. Derner, J. Kubalík, and R. Babuška. Reinforcement learning with symbolic input-output models. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3004–3009, Oct 2018a. doi: 10.1109/IROS.2018.8593881.

E. Derner, J. Kubalík, and R. Babuška. Data-driven construction of symbolic process models for reinforcement learning. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 2018b.

D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

J. Faigl, J. Chudoba, K. Košnar, M. Kulich, M. Saska, and L. Přeučil. Syrotek - a robotic system for education. *AT&P journal PLUS*, 2:31–36, 2010.

M. L. Felis. RBDL: an efficient rigid-body dynamics library using recursive algorithms. *Autonomous Robots*, pages 1–17, 2016. ISSN 1573-7527. doi: 10.1007/s10514-016-9574-0. URL `http://dx.doi.org/10.1007/s10514-016-9574-0`.

J. Forbes and D. Andre. Representations for learning control policies. In *Proc. 19th Int. Conf. Mach. Learn. Workshop Develop. Represent.*, pages 7–14, 2002.

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*, 42(3):591–602, 2012.

S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. *CoRR*, abs/1603.00748, 2016.

N. Heess, G. Wayne, D. Silver, T. P. Lillicrap, Y. Tassa, and T. Erez. Learning continuous control policies by stochastic value gradients. *CoRR*, abs/1510.09142, 2015.

T. Hester and P. Stone. Intrinsically motivated model learning for developing curious robots. *Artif. Intell.*, 247(C):170–186, June 2017. ISSN 0004-3702. doi: 10.1016/j.artint.2015.05.002.

D. Jackson. A new, node-focused model for genetic programming. In A. Moraglio, S. Silva, K. Krawiec, P. Machado, and C. Cotta, editors, *Genetic Programming: 15th European Conference, EuroGP 2012, Málaga, Spain, April 11-13, 2012. Proceedings*, pages 49–60. Springer, Berlin, Heidelberg, 2012a. ISBN 978-3-642-29139-5.

D. Jackson. Single node genetic programming on problems with side effects. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I*, pages 327–336. Springer, Berlin, Heidelberg, 2012b. ISBN 978-3-642-32937-1.

N. K. Jong and 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, pages 95:1–95:8, New York, NY, USA, 2007. ACM. ISBN 978-81-904262-7-5. doi: 10.1145/1329125.1329242.

J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, 23(3):462–466, 09 1952. doi: 10.1214/aoms/1177729392. URL `https://doi.org/10.1214/aoms/1177729392`.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

J. Kober and J. Peters. Reinforcement learning in robotics: A survey. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning: State-of-the-Art*, pages 579–610. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-27645-3. doi: 10.1007/978-3-642-27645-3_18.

I. Koryakovskiy, H. Vallery, R. Babuška, and W. Caarls. Evaluation of physical damage associated with action selection strategies in reinforcement learning. *IFAC-PapersOnLine*, 50(1):6928–6933, 2017.

J. Kubalík, E. Alibekov, and R. Babuška. Optimal control via reinforcement learning with symbolic policy approximation. *IFAC-PapersOnLine*, 50(1):4162 – 4167, 2017a. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2017.08.805. URL `http://www.sciencedirect.com/science/article/pii/S2405896317312594`. 20th IFAC World Congress.

J. Kubalík, E. Derner, and R. Babuška. Enhanced symbolic regression through local variable transformations. In *Proceedings of the 9th International Joint Conference on Computational Intelligence*, pages 91–100, 2017b. ISBN 978-989-758-274-5. doi: 10.5220/0006505200910100.

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

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)*, pages 1–8, Brisbane, Australia, June 2012.

S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and

K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1071–1079. Curran Associates, Inc., 2014.

R. Lieck and M. Toussaint. Temporally extended features in model-based reinforcement learning with partial observability. *Neurocomputing*, 192:49–60, 2016.

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

R. Lioutikov, A. Paraschos, J. Peters, and G. Neumann. Sample-based information-theoretic stochastic optimal control. In *Proceedings of 2014 IEEE International Conference on Robotics and Automation*, pages 3896–3902. IEEE, 2014.

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

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*, 518(7540):529–533, 2015.

R. Munos and A. Moore. Variable resolution discretization in optimal control. *Machine learning*, 49(2):291–323, 2002.

A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous inverted helicopter flight via reinforcement learning. In M. H. Ang and O. Khatib, editors, *Experimental Robotics IX: The 9th International Symposium on Experimental Robotics*, pages 363–372. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-33014-1. doi: 10.1007/11552246_35.

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*, 46(9):1190–1201, Sept 2016. ISSN 2168-2216. doi: 10.1109/TSMC.2015.2475716.

J. Peters and S. Schaal. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225, Oct 2006. doi: 10.1109/IROS.2006.282564.

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

E. Schuitema, M. Wisse, T. Ramakers, and 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*, pages 3238–3243. IEEE, 2010.

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*, 99:1–12, 2012.

R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991. ISSN 0163-5719. doi: 10.1145/122344.122377.

H. J. Tulleken. Generalized binary noise test-signal concept for improved identification-experiment design. *Automatica*, 26(1):37 – 49, 1990. ISSN 0005-1098. doi: https://doi.org/10.1016/0005-1098(90)90156-C.

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*, 50:236–243, 2013.

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*, GECCO '17, pages 145–146, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4939-0. doi: 10.1145/3067695.3076009. URL `http://doi.acm.org/10.1145/3067695.3076009`.