Reinforcement Learning with Symbolic Input–Output Models

Erik Derner, Jiří Kubalík, and Robert Babuška

Presented at the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), October 2018, Madrid, Spain.

DOI: 10.1109/IROS.2018.8593881
https://ieeexplore.ieee.org/document/8593881

IEEE Copyright Notice

BibTeX Record

```
@INPROCEEDINGS{Derner2018-IROS,
author="Derner, Erik and Kubal{\'i}k, Ji{\v{r}}{\'i} and Babu{\v{s}}ka, Robert",
booktitle={2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)},
title={Reinforcement Learning with Symbolic Input-Output Models},
year={2018},
volume={},
number={},
pages={3004-3009},
doi={10.1109/IROS.2018.8593881},
issn={2153-0866},
month={Oct},}
```

Reinforcement Learning with Symbolic Input–Output Models

Erik Derner[1], Jiří Kubalík[2], and Robert Babuška[3]

***Abstract*— It is well known that reinforcement learning (RL) can benefit from the use of a dynamic prediction model which is learned on data samples collected online from the process to be controlled. Most RL algorithms are formulated in the state-space domain and use state-space models. However, learning state-space models is difficult, mainly because in the vast majority of problems the full state cannot be measured on the system or reconstructed from the measurements. To circumvent this limitation, we propose to use input–output models of the NARX (nonlinear autoregressive with exogenous input) type. Symbolic regression is employed to construct parsimonious models and the corresponding value functions. Thanks to this approach, we can learn accurate models and compute optimal policies even from small amounts of training data. We demonstrate the approach on two simulated examples, a hopping robot and a 1-DOF robot arm, and on a real inverted pendulum system. Results show that our proposed method can reliably determine a good control policy based on a symbolic input–output process model and value function.**

***Index Terms*— Model learning, symbolic regression, reinforcement learning, optimal control.**

I. Introduction

In the standard reinforcement learning (RL) formulation, the system to be controlled is described by a state transition function $x_{k+1} = g(x_k, u_k)$, with $x_k, x_{k+1} \in \mathcal{X} \subset \mathbb{R}^n$ and $u_k \in \mathcal{U} \subset \mathbb{R}^m$. The goal of RL is to find the optimal control policy $\pi : \mathcal{X} \to \mathcal{U}$, which is nothing else than a nonlinear state-feedback control law. In problems with continuous-valued state and input spaces, the state transition function g and the policy π are represented by using a function approximator, such as a basis function expansion [1], [2], a regression tree [3], local linear regression [4], [5], a deep neural network [6], [7], [8], [9], [10], or Gaussian process model [11], among other possibilities.

This approach has one main inherent drawback: the full state vector cannot be directly measured for the vast majority of processes. In such a case, the state has to be reconstructed from the available measurements, by using a state estimator. In the absence of an accurate process model (which is usually the main reason for using RL), such a reconstruction is inaccurate and jeopardizes 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.

In this paper, we investigate the use of dynamic input–output models of the NARX (nonlinear autoregressive with exogenous input) type, instead of state-space models. The NARX model establishes a relation between the past input–output data and the predicted output: $y_{k+1} = f\left(y_k, \ldots, y_{k-n_y+1}, u_k, \ldots, u_{k-n_u+1}\right)$, where n_y and n_u are integers related to the system's order, and f is a static function, different from the function g used in the state-space model. Instead of the state vector, the NARX model uses a regression vector which is a collection of past inputs and outputs. This means that the model function f and also the policy π has to be found from data samples living in a space that is very different from the state space. The lagged outputs y_k, $y_{k-1}, \ldots$ are highly correlated and therefore span a skewed space, which is a problem for many types of approximators. For instance, basis function defined by the Cartesian product of functions defined for the individual lagged variables will evenly cover the whole product space $y_k \times y_{k-1} \times \cdots$, while data samples only span a small, diagonally oriented part of the space, as illustrated in Fig. 1.

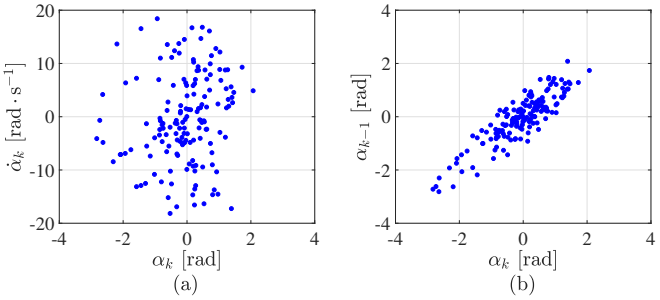

Fig. 1. An example of a trajectory of the real inverted pendulum (see Section III-C) in the original state space (a), and in the space formed by the current and previous output (b).

To deal with this problem, we employ symbolic regression [12], [13] to build an analytic representation of the model and of the value function (also denoted as V-function in the sequel), which serves as a basis for deriving the control policy π. Symbolic regression 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 constructs parsimonious models, even from small data sets [13].

[1]Erik Derner is with the Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Czech Republic and with the Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic erik.derner@cvut.cz

[2]Jiří Kubalík is with the Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Czech Republic jiri.kubalik@cvut.cz

[3]Robert Babuška is with Cognitive Robotics, Faculty of 3mE, Delft University of Technology, The Netherlands and with the Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Czech Republic r.babuska@tudelft.nl

II. Method

The NARX model of the system for which an optimal control strategy is to be learned is described in discrete time as follows:

$$\hat{y}_{k+1} = f\left(y_k, y_{k-1}, \ldots, y_{k-n_y+1}, u_k, u_{k-1}, \ldots, u_{k-n_u+1}\right) \quad (1)$$

where n_y and n_u are user-defined integer parameters based on the expected system's order, and f is a static function sought by a symbolic regression (SR) algorithm. Note that the sensor readings can be corrupted by noise, but here we aim at constructing a deterministic model.

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

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

and write model (1) as:

$$\hat{y}_{k+1} = f\left(\xi_k, u_k\right). \quad (2)$$

The control goal is specified through a reward function which assigns a scalar reward $r_{k+1} \in \mathbb{R}$ to each new sample:

$$r_{k+1} = \rho(\xi_k, u_k, y_{k+1}). \quad (3)$$

Function ρ is defined by the user and typically calculates the reward based on the distance of the current output to a given desired output.

Based on model (2), we compute the optimal control policy π which in each state selects a control action so that the cumulative discounted reward over time, called the return, is maximized:

$$R^{\pi} = E\left\{\sum_{k=0}^{\infty} \gamma^k \rho\left(\xi_k, \pi(\xi_k), y_{k+1}\right)\right\}. \quad (4)$$

Here $\gamma \in (0,1)$ is a discount factor and the initial value ξ_0 is drawn uniformly from the input–output domain. The return is captured by the value function defined as:

$$V^{\pi}(\xi) = E\left\{\sum_{k=0}^{\infty} \gamma^k \rho\left(\xi_k, \pi(\xi_k), y_{k+1}\right) \middle| \xi_0 = \xi\right\}. \quad (5)$$

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

$$\hat{V}^*(\xi) = \max_{u \in \mathcal{U}} \left[\rho\left(\xi, \pi(\xi), f(\xi,u)\right) + \gamma \hat{V}^*\left(f(\xi,u)\right)\right]. \quad (6)$$

In the sequel, we drop the hat and the star superscript: $V(\xi)$ will stand for the approximated optimal V-function. Based on $V(\xi)$, the corresponding optimal control action is found as the argument that maximizes the right-hand side of (6):

$$u = \operatorname*{argmax}_{u' \in U} \left[\rho(\xi, u', f(\xi, u')) + \gamma V(f(\xi, u'))\right] \quad (7)$$

where U is a set of discretized actions, so that the near-optimal action can be found by enumeration.

The process model (2) is sought by means of symbolic regression as follows. The class of symbolic models is defined as:

$$f(\xi, u) = \sum_{i}^{n_f} \beta_i f_i(\xi, u) \quad (8)$$

where $f_i(\xi, u)$ are nonlinear functions (called features), constructed by genetic programming. Coefficients β_i are estimated by least squares and n_f is the user-defined maximum number of features.

The features f_i are constructed from a set of user-defined elementary functions $\mathcal{F}$. These functions can be nested and are evolved by means of standard evolutionary algorithm operations such as mutation. To control the complexity of the regression model, we limit the maximal depth d of the evolved symbolic expressions. The elementary functions can be chosen based on prior knowledge about the system modeled, or, in the absence of prior knowledge, one can use general functions such as tanh.

The algorithm we use in this work is based on Single Node Genetic Programming (SNGP) [14], [15]. It is a graph-based technique that evolves the symbolic model to minimize the mean-squared error calculated over the training data set. In particular, we use a modified version of SNGP with partitioned population and linearly transformed variables [12].

Once the symbolic process model is available, it is used in value iteration. We employ symbolic regression to find an approximate value function $V(\xi)$ iteratively, just like in standard value iteration [16]. In each iteration, the value function from the previous iteration is used to compute the target for improving the value function in the current iteration. The improved symbolic value function is constructed by using symbolic regression with the mean-squared Bellman error as the fitness function.

III. Experimental Evaluation

We have selected the following non-linear control problems as a benchmark for our method: a hopping robot, a 1-DOF robot arm operating under the influence of gravity and a real inverted pendulum system.

A. Hopping Robot

1) System description: The hopping robot (see Fig. 2) consists of two masses, the body m_1 and the foot m_2, connected by a spring.

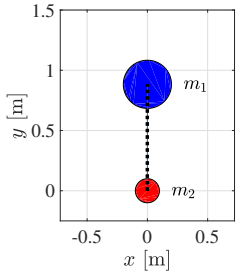

Fig. 2. Hopping robot schematic. The body m_1 and the foot m_2 are connected by a spring, shown as a dotted line.

The body has the following continuous-time dynamics:

$$\begin{aligned} \ddot{x}_1 &= \frac{\kappa \Delta x}{m_1 l}(L_0 - l), \\ \ddot{y}_1 &= -g + \frac{\kappa \Delta y}{m_1 l}(L_0 - l), \end{aligned} \quad (9)$$

where κ is a variable spring constant, $g = 9.81\,\mathrm{m}\cdot\mathrm{s}^{-2}$ is the gravitational acceleration, $m_1 = 10\,\mathrm{kg}$ and $m_2 = 2\,\mathrm{kg}$ are the masses of the body and the foot, respectively, and $L_0 = 1\,\mathrm{m}$ is the equilibrium spring length. The actual spring length l is given by the formula

$$l = \sqrt{\Delta x^2 + \Delta y^2}\,. \tag{10}$$

The foot has the following dynamics:

$$\begin{aligned} \ddot{x}_2 &= \frac{\kappa \Delta x}{m_2 l}(L_0 - l) - \frac{1}{m_2} b \dot{x}_2\,, \\ \ddot{y}_2 &= -g + \frac{\kappa \Delta y}{m_2 l}(L_0 - l) - \frac{1}{m_2} b \dot{y}_2\,, \end{aligned} \tag{11}$$

where $b = 10\,\mathrm{kg}\cdot\mathrm{s}^{-1}$ is the damping coefficient. The ground contact phenomenon is simulated by setting $y_2 = \dot{y}_2 = 0$ whenever the result of numerically integrating (11) would yield $y_2 < 0$.

We simplify the problem by fixing the x-coordinate to 0, which allows the robot to jump only in the vertical direction. The state is described by $x = [y_1, v_1, y_2, v_2]^\top$, where y_1 is the vertical position of the robot body and y_2 of its foot and analogously, v_1 is the velocity of the robot body and v_2 of its foot. The control input u is variable stiffness, added to the spring constant. We therefore define the spring constant as $\kappa = \kappa' + u$, where $\kappa' = 1000\,\mathrm{kg}\cdot\mathrm{s}^{-2}$ is the nominal spring constant.

We set the initial position of the robot's body to $y_{1,0} = 1.2\,\mathrm{m}$ and its foot to $y_{2,0} = 0.2\,\mathrm{m}$. When the robot falls from this position with a constant input $u = 0$, it reaches the ground, jumps once and never lifts the foot above the ground afterwards, see Fig. 3. The control goal is to keep the robot jumping as high as possible by adjusting the spring constant κ through the input u. The reward function is defined as

$$\rho(\xi_k, u_k, y_{k+1}) = -(y_{2,r} - y_{2,k})^2\,, \tag{12}$$

where $y_{2,r} = 2\,\mathrm{m}$ is the reference for the robot's foot position y_2.

2) Process model: At the first stage, the symbolic process model has to be found. We use the fourth-order Runge-Kutta integration method to simulate the system described by the above physical model. The sampling period was $T_s = 0.002\,\mathrm{s}$. The discrete[1] control input $u \in U$ can take one of the following five values: $U = \{-200, -100, 0, 100, 200\}\,\mathrm{kg}\cdot\mathrm{s}^{-2}$.

Based on the prior knowledge of the system dynamics, we selected the regression vector to be $\xi_k = [y_{1,k}, y_{1,k-1}, y_{2,k}, y_{2,k-1}]^\top$. An alternative choice would be to include u_{k-1} as well, but our experience is that for larger regression vectors, symbolic models can be harder to find.

The samples ξ_k were collected by a short (30 s) interaction with the system, which consisted of three parts. In two of them, a random input was applied to the system. In the first case, the random input was changed in every time step, whereas in the second case, it was changed only every 100 time steps. The third part was constructed by applying the heuristics

$$u_k = h(\xi_k) = \begin{cases} -200 & \text{if } y_{1,k} - y_{1,k-1} < 0\,, \\ 200 & \text{if } y_{1,k} - y_{1,k-1} > 0\,, \\ 0 & \text{otherwise}\,, \end{cases} \tag{13}$$

which keeps the robot jumping. Each of the three parts had 5000 samples. A random subset of 3000 samples was selected from the 15 000 samples to speed up the symbolic regression. Finally, the data set was divided into the training and validation subset in the ratio 2:1.

The parameters of the SNGP algorithm are summarized in Table I. A detailed explanation of the parameters can be found in [12].

TABLE I
COMMON PARAMETERS OF THE SNGP ALGORITHM FOR CONSTRUCTING THE PROCESS MODEL.

Parameter	Symbol	Value
Population size	n_p	500
Number of generations	n_g	30 000
Depth limit	d	7
Number of features	n_f	10
Number of runs	n_r	30

We chose the elementary function set to be $\mathcal{F} = \{*, +, -,$ square, cube, bent identity[2]$\}$. Since SNGP only allows modeling one output at a time, we ran the algorithm twice, for $\hat{y}_{1,k+1}$ and $\hat{y}_{2,k+1}$.

All 30 models were evaluated on the validation set and the model with the lowest RMSE was selected as the final symbolic process model. The best model for $\hat{y}_{1,k+1}$ had a RMSE of 2.30×10^{-6} and the RMSE median over all 30 models was 2.40×10^{-6}. In case of $\hat{y}_{2,k+1}$, the best model reached a RMSE of 5.55×10^{-6} and the RMSE median over all 30 models was 1.11×10^{-4}. The final symbolic process model performs well in the simulation, see Fig. 3.

One SNGP run took approximately 10 minutes on a single core of a standard desktop PC.

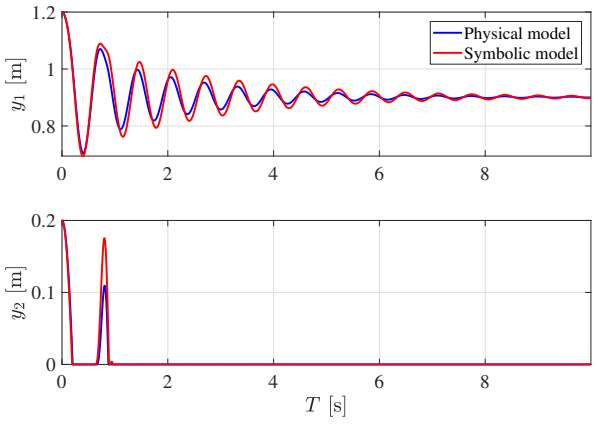

Fig. 3. Trajectory of the body (y_1) and the foot (y_2) of the hopping robot falling from the position $y_1 = 1.2\,\mathrm{m}$, $y_2 = 0.2\,\mathrm{m}$ under control $u = 0$. The plot shows a simulation using the physical (blue) and symbolic (red) model.

[1]Symbolic regression does not require discrete inputs, however, symbolic value iteration currently does. In principle, one could collect a training set with continuous inputs for constructing the process model.

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

3) V-function: Once the final symbolic process model is available, the symbolic V-function can be estimated. The data set that was used to train the symbolic process model served as a basis for the data set for learning the V-function. For each sample, a set of next states for all possible discrete inputs $u \in U$ was generated using the symbolic process model.

The settings of the symbolic value iteration algorithm are summarized in Table II. The number of iterations n_i was 100. The discount factor γ was set to 0.999 because of the very short time step T_s.

TABLE II
COMMON PARAMETER VALUES OF SYMBOLIC VALUE ITERATION.

Parameter	Symbol	Value
Population size	n_p	500
Generations per iteration	n_l	500
Function set	$\mathcal{F}$	$\{*, +, -$, square, cube, bent identity$\}$
Depth limit	d	7
Number of features	n_f	10
Number of runs	n_r	30

A symbolic V-function was generated in each iteration, which yielded a total of 3030 functions (including the initial ones). All symbolic V-functions were simulated on the symbolic process model and the V-function with the highest cumulative reward, calculated using (12) in every time step k, was selected as the final V-function. The best V-function achieved a cumulative reward of -3.79×10^3 and the median cumulative reward of all V-functions was -3.95×10^3.

The policy given by the symbolic V-function keeps the robot hopping in a simulation on the physical model, as shown in Fig. 4. Note that if we change the initial state to one with a ground contact, for instance, $y_{1,0} = 1\,\text{m}$, $y_{2,0} = 0\,\text{m}$, the robot starts and keeps hopping as well.

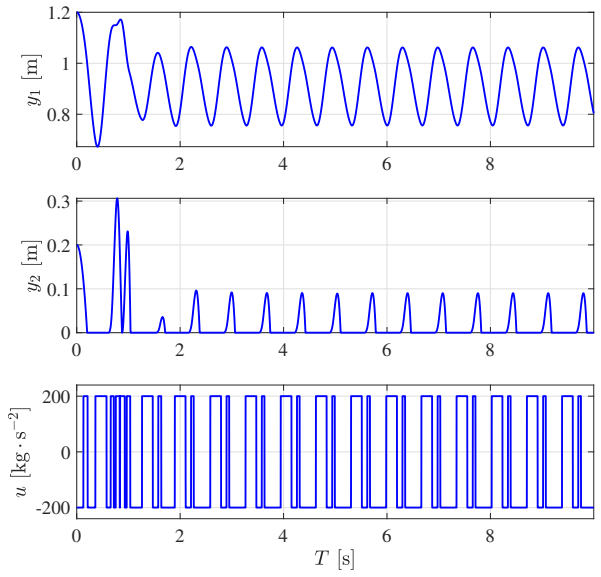

Fig. 4. Trajectory of the body (y_1) and the foot (y_2) of the hopping robot starting at the position $y_{1,0} = 1.2\,\text{m}$, $y_{2,0} = 0.2\,\text{m}$ under the policy given by the best found symbolic V-function. The plot shows a simulation using the physical model. The control input u is shown in the bottom panel.

B. 1-DOF Robot Arm

1) System description: The 1-DOF robot arm system consists of a weight of mass m attached to an actuated link which rotates in the vertical plane, see Fig. 5. The state vector is $x = [\alpha, \dot{\alpha}]^\top$ with α the angle and $\dot{\alpha}$ the angular velocity of the link. The control input is the voltage u. The continuous-time model of the 1-DOF robot arm dynamics is:

$$\ddot{\alpha} = \frac{1}{J} \cdot \left(\frac{K}{R} u - m g l \sin(\alpha) - b \dot{\alpha} - \frac{K^2}{R} \dot{\alpha} - c \operatorname{sign}(\dot{\alpha}) \right) \quad (14)$$

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

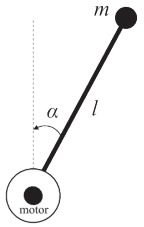

Fig. 5. 1-DOF robot arm schematic.

The control goal is to reach and stay in the reference (goal) state $\alpha_r = \pi/4\,\text{rad}$ from a given initial state. The reward function for the robot arm is defined as follows:

$$\rho(\xi_k, u_k, x_{k+1}) = -(\alpha_r - \alpha_k)^2. \quad (15)$$

2) Process model: First, we need to find the symbolic process model. We chose the regression vector to be $\xi_k = [\alpha_k, \alpha_{k-1}, u_{k-1}]^\top$. Again, the fourth-order Runge-Kutta integration method was applied to simulate the system using the physical model. The sampling time was set to $T_s = 0.05\,\text{s}$. The discrete control input u can take any integer value between -5 and $5\,\text{V}$, including the boundaries.

As in the case of the hopping robot, the data set of samples ξ_k was composed of three parts. The first two data sets were recorded under a random input. The input was changed in every time step in the first case and it remained constant for 20 time steps in the latter case. The third part consisted of 14 runs, each 2.5 s long, under control $u = 0\,\text{V}$ and starting at the following initial angles:

$$\alpha_{\text{init}} \in \left\{ \frac{i\pi}{8}\,,\ i \in [1..15] \setminus \{8\} \right\}. \quad (16)$$

The initial angular velocity $\dot{\alpha}$ was zero in all cases. Note that only the values of α_k and the control inputs u_k are recorded in the data sets in each time step k, i.e., the values of $\dot{\alpha}_k$ are not used. The random parts of the data set consisted of 600 samples each and the last part had 700 samples. The complete data set of 1900 samples was divided into the training and validation subset in the ratio 2:1.

The SNGP algorithm was run on the training set with parameters summarized in Table I. We chose the elementary function set to be $\mathcal{F} = \{*, +, -, \sin, \cos, \text{sign}\}$.

As with the hopping robot, all 30 models were evaluated on the validation set and the model with the lowest RMSE was chosen as the final symbolic process model. The best model achieved a RMSE of 4.43×10^{-3} and the RMSE median over all 30 models was 1.02×10^{-2}. The final symbolic process model performs reasonably well in a simulation, see Fig. 6.

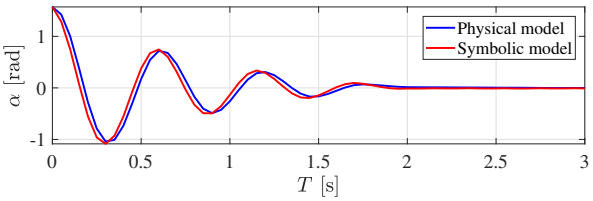

Fig. 6. Trajectory of the 1-DOF robot arm falling from the position $\alpha_0 = \pi/2$ rad under control $u = 0$. The plot shows the first 3 seconds of a simulation using the physical (blue) and symbolic (red) model.

3) V-function: The symbolic V-function was sought on a data set prepared in the same way as for the hopping robot. For each sample in the data set used for learning the process model, a set of next states was generated using the symbolic process model for all possible discrete inputs $u \in U$.

The configuration of the symbolic value iteration algorithm is presented in Table II. The number of iterations n_i was 200 and the discount factor γ was set to 0.99.

Symbolic value iteration yielded 6030 symbolic V-functions in total, including the initial ones. The symbolic process model was used to simulate all the V-functions. The simulations for each V-function started in 16 initial angles, i.e., in 14 states described by (16), in 0 and in π rad. The function with the highest cumulative reward averaged over all initial states was selected as the final V-function. The best V-function reached a cumulative reward of -2.50×10^1 and the median cumulative reward of all V-functions was -7.29×10^2.

The policy given by the best symbolic V-function reaches the reference state α_r within 5 seconds for all 16 initial states in a simulation both with the symbolic process model and with the physical model. An example of a simulation on the physical model starting at $\alpha_0 = \pi$ rad is shown in Fig. 7.

C. Real Inverted Pendulum

1) System description: The real inverted pendulum system used in the experiment is described in [5] and it can be modeled by the differential equation (14). However, note that this model is not used here and all data are collected on the real system.

As in Section III-B, the control goal is to achieve the reference state $\alpha_r = \pi/4$ rad and the reward is defined by (15).

2) Process model: The symbolic process model was trained on a data set $\xi_k = [\alpha_k, \alpha_{k-1}, u_{k-1}]^\top$ measured on the real system. The interaction with the system took 30 seconds, which yielded 600 samples with the sampling time

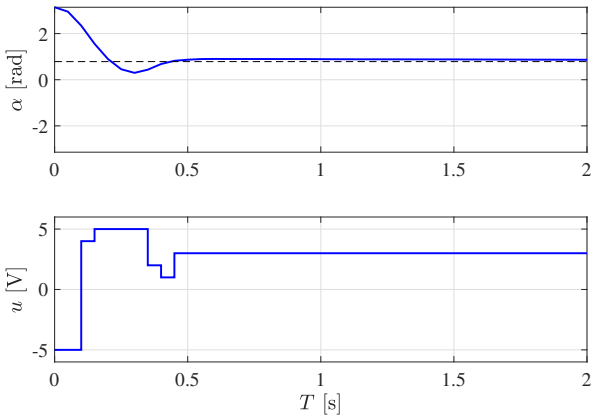

Fig. 7. Trajectory of the 1-DOF robot arm starting at the position $\alpha_0 = \pi$ rad under the policy given by the best found symbolic V-function. The plot shows the first 2 seconds of a simulation using the physical model. The dashed line denotes the goal state $\alpha_r = \pi/4$ rad. The control input u is shown in the bottom panel.

$T_s = 0.05$ s. The control input u was randomly chosen in each time step among 11 discrete inputs evenly spacing the range between -5 and 5 V. The data set was divided into the training set of 400 samples and the validation set, containing 200 samples.

The SNGP algorithm was run on the training set with the same settings as described in Section III-B.2. The resulting 30 models were evaluated on the validation set and the model with the lowest RMSE was chosen as the final symbolic process model. The best model had a RMSE of 3.96×10^{-3} and the RMSE median over all 30 models was 5.14×10^{-3}.

3) V-function: Same as in the previous experiments, the samples in the data set used for learning the process model served as a base to generate next states for all possible discrete inputs $u \in U$ using the symbolic process model. The resulting data set was used in the symbolic value iteration algorithm. The configuration of the algorithm was the same as in Section III-B.3.

Symbolic value iteration yielded 6030 symbolic V-functions. The symbolic process model found on the data from the random interaction with the real pendulum system was used to simulate all the V-functions. Similarly as in the experiment with the 1-DOF robot arm, the simulations for each V-function started in 16 initial angles, spanning evenly around a circle. The function with the highest cumulative reward averaged over all initial states was selected as the final V-function. The best V-function achieved a cumulative reward of -2.83×10^1 and the median cumulative reward of all V-functions was -8.69×10^2. The policy given by the best symbolic V-function reaches within 5 seconds the reference state α_r for all 16 initial states in a simulation with the symbolic process model.

The best symbolic V-function performs well on the real system, see Fig. 8 and the attachment video. The steady state error of 0.1 rad and the chattering of the control input can be explained by the discrete set of actions, which does not contain the optimal control input [17].

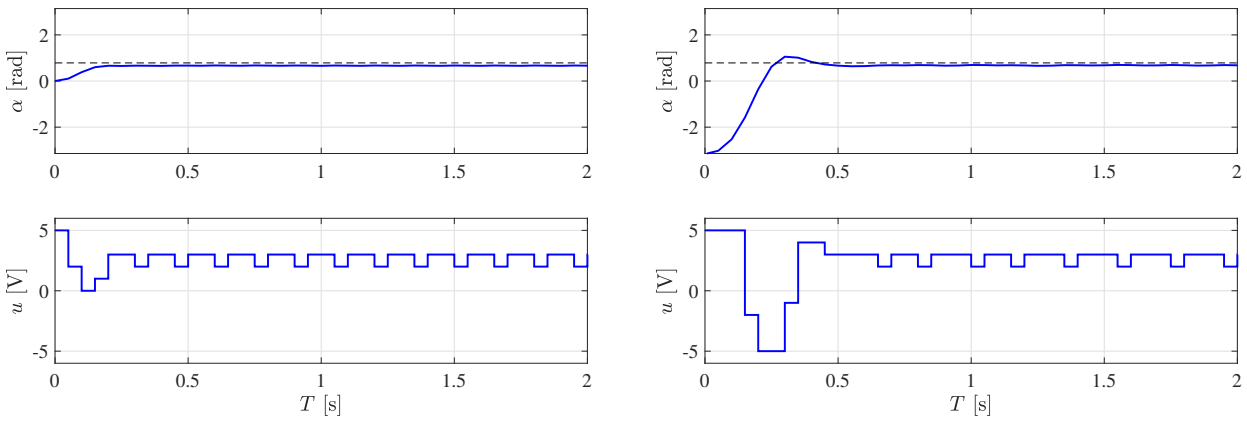

Fig. 8. Performance of the final symbolic V-function on the real pendulum system for two initial states $\alpha_0 = 0\,\text{rad}$ (a) and $\alpha_0 = -\pi\,\text{rad}$ (b). The dashed line denotes the reference (goal) state $\alpha_r = \pi/4\,\text{rad}$.

IV. Conclusions

Symbolic input–output models are suitable for application in the RL scheme as the process models and value functions. The strength of symbolic regression is that only a small amount of data is needed to find precise models. In addition, symbolic models are parsimonious and well interpretable by humans.

Experimental validation shows that our method reliably finds symbolic process models and value functions which are used to form a policy that achieves the desired control goal, both in simulations and in a real experiment. The most important limitation of this approach is its computational complexity.

In our future work, we propose to evaluate the method on higher-dimensional problems, compare symbolic regression with alternative modeling approaches and extend the method to batchwise learning in parallel with real-time control.

V. Acknowledgements

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. SGS16/232/OHK3/3T/13. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the program "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

References

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

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

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

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

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

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

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

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

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

[10] T. de Bruin, J. Kober, K. Tuyls, and R. Babuška, "Integrating state representation learning into deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1394–1401, 2018.

[11] M. P. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *In Proceedings of the International Conference on Machine Learning*, 2011.

[12] 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*, 2017, pp. 91–100.

[13] 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 2018.

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

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

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

[17] J. Kubalík, E. Alibekov, and R. Babuška, "Optimal control via reinforcement learning with symbolic policy approximation," in *Preprints 20th IFAC World Congress (IFAC-17)*, Toulouse, France, July 2017.