Multi-objective symbolic regression for physics-aware dynamic modeling

Kubalík, Jiří; Derner, Erik; Babuška, Robert

DOI
10.1016/j.eswa.2021.115210

Publication date
2021

Document Version
Accepted author manuscript

Published in
Expert Systems with Applications

Multi-objective Symbolic Regression for Physics-aware Dynamic Modeling

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

[a]Corresponding author. Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, 16636 Prague, Czech Republic, Email: jiri.kubalik@cvut.cz, Phone: +420 22435 4161
[b]Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 16627 Prague, Czech Republic, Email: erik.derner@cvut.cz
[c]Cognitive Robotics, Delft University of Technology, 2628 CD Delft, The Netherlands, Email: r.babuska@tudelft.nl

---

Keywords: symbolic regression, genetic programming, model learning for control, motion control

---

Abstract

Virtually all dynamic system control methods benefit from the availability of an accurate mathematical model of the system. This includes also methods like reinforcement learning, which can be vastly sped up and made safer by using a dynamic system model. However, obtaining a sufficient amount of informative data for constructing dynamic models can be difficult. Consequently, standard data-driven model learning techniques yield models that are partly incorrect, for instance, in terms of their steady-state characteristics or local behavior. However, often some knowledge about the desired physical properties of the model is available. Therefore, this knowledge should be incorporated into the model learning process to compensate for data insufficiency.

In this paper, we consider a multi-objective symbolic regression method that optimizes models with respect to their training error and the measure of how well they comply with the desired physical properties. We propose an extension to the existing algorithm that helps generate a diverse set of high-quality models. Further, we propose a method for selecting a single final model out of the pool of candidate output models. We experimentally demonstrate the approach on three real systems: the TurtleBot 2 mobile robot, the Parrot Bebop 2 drone and the magnetic manipulation system. The results show that the proposed model-learning algorithm yields accurate models that are physically justified. The improvement in terms of the model's compliance with prior knowledge over the models obtained when no prior knowledge was involved in the learning process is of several orders of magnitude.

1. Introduction

This paper deals with data-driven approaches for learning models of dynamic systems. Methods proposed in this paper aim at learning models that in addition to a small training error also exhibit high compliance with the physical properties of the given system.

Many data-driven model-learning approaches have been described in the literature: time-varying linear models (Levine & Abbeel, 2014; Lioutikov et al., 2014), Gaussian processes (Deisenroth & Rasmussen, 2011; Boedecker et al., 2014) and other probabilistic models (Ng et al., 2006), deep neural networks (de Bruin et al., 2016; Heess et al., 2015), and symbolic regression (Koza, 1992; Schmidt & Lipson, 2009; Derner et al., 2018a,b).

All of these methods have one common feature: they are trained using a set of data samples by minimizing only a training error measure, such as the traditional mean squared error. These methods sometimes yield models that are partially incorrect, for instance, in terms of their steady-state characteristics, local or even global behavior, see (Kubalík et al., 2020). This is typically due to insufficient information content of the identification data, when the data set does not sufficiently cover the input space or when some parts of the input space are completely omitted in the data set. In the system identification literature, this problem was recognized long time ago. It is typically addressed by designing a data collection experiment that provides sufficiently long and informative data sequence obtained by means of a proper excitation signal applied to the system (Goodwin & Payne, 1977; Tulleken, 1990). However, for physical systems learning on the task, data-collection experiments cannot often be designed in order to maximize the information context of the data, as this would deteriorate its performance. On the other hand, domain-specific knowledge or prior

knowledge about the desired properties of the modelled system is often available. This knowledge can serve as complementary to the training data and help to drive the search process towards more accurate and relevant models.

In this paper, we consider symbolic regression (SR), a method that generates models in the form of analytic equations. Typically, SR is realized using genetic programming (GP). SR has been used in nonlinear data-driven modeling with quite impressive results (Schmidt & Lipson, 2009; Vladislavleva et al., 2013; Staelens et al., 2013; Alibekov et al., 2016; Derner et al., 2018a,b, 2020). Moreover, contrary to data-hungry approaches such as neural networks, SR can construct good models even from very small training data sets. SR is also a suitable candidate method for dealing with prior knowledge. Validity checks can be incorporated into the fitness evaluation or as an additional optimization objective (Kubalík et al., 2020). Other approaches dynamically change the training data set through the course of the modeling process. In particular, counterexamples on which otherwise promising models fail to be consistent with prior knowledge are generated on the fly and used to drive the search towards valid models that comply with prior knowledge (Błądek & Krawiec, 2019; Ashok et al., 2020).

We build on the SR approach introduced by Kubalík et al. (2020). This is a bi-objective SR method that optimizes models to fit well the training data and at the same time to comply with the prior knowledge about the given system. We further improve on this method in two ways and experimentally evaluate it on three real physical systems. The main contributions of this paper are:

- *Feature mixing phase*. The original method evolves a population of models, each of them being a linear combination of possibly nonlinear features. The features, evolved by means of genetic programming, are combined using a multiple regression technique. Importantly, each model in the final population is a result of its own mutation-based search trajectory. The models do not combine with each other through the optimization process. We extend the method in order to generate a more diverse set of well-performing models. This is realized by the feature mixing phase added to the end of the algorithm. It operates on the set of features collected from the models produced in the first mutation-based phase and evolves new models as combinations of these features.

- *Final model selection method.* The method in (Kubalík et al., 2020) can generate realistic models. However, the method produces a whole set of candidate output models, out of which the final one has to be selected. The final model was selected mostly based on expert knowledge about the ground truth reference model. So, the selection strategy that would choose the final model without using any reference model remained an open issue. Here, we propose a method to automatically select a single final model at the end of the algorithm's run.

- *Experimental evaluation on real physical systems*. The proposed method is experimentally evaluated on three physical systems with different dynamics: the TurtleBot 2 mobile robot, the Parrot Bebop 2 drone, and the magnetic manipulation system. The motion models are constructed from collected data sequences, complemented by prior knowledge reflecting the basic physics of the systems.

The paper is organized as follows. Related work is surveyed in Section 2. Preliminaries defining the context for the propose method are introduced in Section 3. The proposed two-phase method and the final model selection method are described in Section 4. Section 5 describes the experiments set up. Results obtained on the three physical systems are presented and discussed in Section 6. Finally, Section 7 concludes the paper.

2. Related work

The combination of a priori knowledge with data-driven modeling is not new. Standard gray-box modeling methods include a priori information typically in the form of constraints on the model parameters and variables (Tulleken, 1993; Karny et al., 1995; Johansen, 1996; Timmons et al., 1997). For instance, it is well known that the poles of a linear discrete-time model that emerge from a properly sampled stable continuous-time system cannot be situated in the left half complex plane. This knowledge can be translated into inequality constraints on the model parameters (Tulleken, 1993; Abonyi et al., 2000). However, these methods can only be applied to linear models.

Combining SR with prior knowledge is rather a new research topic. There are only few such SR approaches described in the literature that take into account information about the model sought other than just the minimum training error. One of the most relevant is the Counterexample-Driven Symbolic Regression (CDSR) (Błądek & Krawiec, 2019), which is based on the Counterexample-Driven Genetic Programming (Krawiec et al., 2017). The domain knowledge is represented as a set of constraints, with each constraint expressed in the form of a logical formula. GP is used to synthesize regression models that not only comply with the training data set but also meet formal constraints imposed on the model. A Satisfiability Modulo Theories (SMT) solver is used to verify whether a given model meets the formal specification. If it does not, then a new counterexample is generated and added to the set of test cases on which candidate models are evaluated. An appropriate search gradient is adjusted automatically as the test set is gradually filled with new relevant tests.

A similar approach called Logic Guided Genetic Algorithms (LGGA) was proposed by Ashok et al. (2020). Here, the domain-specific knowledge is called *auxiliary truths* (AT) that are simple mathematical facts known a priori about the unknown function sought. ATs are represented as mathematical formulas. Similar to the previous approach, new counterexamples are generated and used to augment the training data set during the course of the optimization process. Candidate models are evaluated using a weighted sum of the classical training mean squared error and the truth error, which is a measure of how much the model violates given ATs. Contrary to CDSR, LGGA performs computationally light consistency checks via ATs' formulas evaluations on the data set.

Recently, several neural network-based methods have been proposed to address prior knowledge. The AI Feynman algorithm first proposed by Udrescu & Tegmark (2020) and further improved by Udrescu et al. (2020) builds on the fact that formulas appearing in physics and many other scientific applications have simplifying properties such as symmetry, separability, compositionality, physical units, etc. Neural networks and other techniques (e.g., polynomial fitting, brute force search) are used to discover some of these properties in the training data that is further ex-

ploited to break down the original SR problem into smaller problems, which can then be solved separately. Importantly, models with the best accuracy for a given complexity are generated.

3. Preliminaries

In this section, preliminaries defining the context for the proposed two-phase bi-objective symbolic regression method and the final model selection method are introduced. Firstly, the form of models the method generates is defined. Then, the data used to learn the models, as well as the two optimization objectives that guide the search process are described.

3.1. Model structure

The dynamic system model is described in discrete time by the following nonlinear difference equation:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \tag{1}$$

where $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ is the function of n-dimensional state $\mathbf{x}_k = (x_k^1, x_k^2, \ldots, x_k^n)^\top$ and m-dimensional input $\mathbf{u}_k = (u_k^1, u_k^2, \ldots, u_k^m)^\top$ and k denotes the discrete time step. While the actual process can be stochastic (e.g., when the sensor readings are corrupted by noise), in this paper, we construct a deterministic model.

For model learning, we define the model $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ as a vector of models $f^j(\mathbf{x}_k, \mathbf{u}_k)$, each producing a prediction of a single state variable x_{k+1}^j, with $j = 1, \ldots, n$:

$$\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \left(f^1(\mathbf{x}_k, \mathbf{u}_k), f^2(\mathbf{x}_k, \mathbf{u}_k), \ldots, f^n(\mathbf{x}_k, \mathbf{u}_k)\right)^\top. \tag{2}$$

In the sequel, we drop the superscripts to simplify the notation. The generic term $f(\mathbf{x}_k, \mathbf{u}_k)$ corresponds to a model of a single state variable, while the model of the whole system is denoted by $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$. Similarly, x_k refers to a single generic state variable, whereas $\mathbf{x}_k$ represents the full state vector. We assume that the states are measured, but the method also applies to nonlinear input–output models of the form $\mathbf{y}_{k+1} = \mathbf{g}(\mathbf{y}_k, \mathbf{y}_{k-1}, \ldots, \mathbf{u}_k, \mathbf{u}_{k-1}, \ldots)$, where the state is represented by the vector of past inputs and outputs (Leonaritis & Billings, 1985).

3.2. Prior knowledge

The prior knowledge captures important high-level characteristics of the system's physical laws without requiring in-depth knowledge of the physical model. These characteristics are usually independent of specific parameter values, which are generally unknown. Prior knowledge can be expressed as constraints on the model parameters or function values, data representing steady-state behavior of the system, velocity and acceleration trends under specific input, etc. In the context of symbolic regression, the most straightforward and effective way is to represent prior knowledge in the form of synthetic data samples describing the desired behavior. This allows for direct quantification of the extent to which the candidate models comply with the prior knowledge.

3.3. Data and optimization objectives

The problem of analytic process model construction has two equally important goals. The model should (1) fit the training data as precisely as possible and (2) be consistent with the prior knowledge about the system. The goals are formally defined by the following optimization objectives:

- minimize C_t, where C_t is a root-mean-square error (RMSE) calculated over a training data set. The training data set consists of data samples of the form $\mathbf{d}_k = (\mathbf{x}_k, \mathbf{u}_k, x_{k+1})$.
- minimize C_c, where C_c is a measure of how well the analytic process model complies with the prior knowledge. We assume that any type of prior knowledge can be written as nonlinear inequality and equality constraints that the system must obey. In particular, synthetic data samples (i.e., samples not measured on the system), called constraint samples, are generated specifically for a given constraint and the desired inequality or equality relation is defined on them. For each such constraint sample, we can calculate how much the model violates the desired inequality or equality relation. C_c is then calculated as the RMSE calculated over the whole constraint data set.

For more details on the formal definition of C_t and C_c refer to Kubalík et al. (2020). Besides the training data set, validation and test data sets are generated to assess the quality of evolved analytic process model. Contrary to the training data set, the validation and test data sets have the form of sequences collected during the operation of the system.

In the sequel, we will use the term *prior knowledge* when referring to the characteristics the evolved models should exhibit (e.g., the motion laws behind the studied system). The terms *constraint* and *constraint sample* refer to particular data samples used to evaluate the compliance of the model with prior knowledge.

3.4. Base SNGP

The multi-objective symbolic regression algorithm that we build on here is based on the SNGP variant presented in (Kubalík et al., 2017). It evolves nonlinear functions $f(\mathbf{x}_k, \mathbf{u}_k)$ represented by

$$f(\mathbf{x}_k, \mathbf{u}_k) = \beta_0 + \sum_{i=1}^{n_f} \beta_i \varphi_i(\mathbf{x}_k, \mathbf{u}_k)\,, \tag{3}$$

where the nonlinear functions $\varphi_i(\mathbf{x}_k, \mathbf{u}_k)$ are *features* constructed by means of tree-based genetic programming. The coefficients β_i are not evolved using genetic operators. Instead, they are estimated using a multiple regression technique, e.g., least squares. The complexity of the evolved analytic models is controlled by two user-defined parameters: n_f is the maximum number of features the analytic model can be composed of, and δ is the maximal depth of the feature's tree representation. GP using this kind of compound regression models has recently been shown in (Arnaldo et al., 2015; Searson, 2014) to outperform conventional GP that evolves models represented by a single-tree structure and has been successfully used for several SR tasks from the reinforcement learning and robotics domains (Alibekov et al., 2016; Derner et al., 2018a,b, 2020).

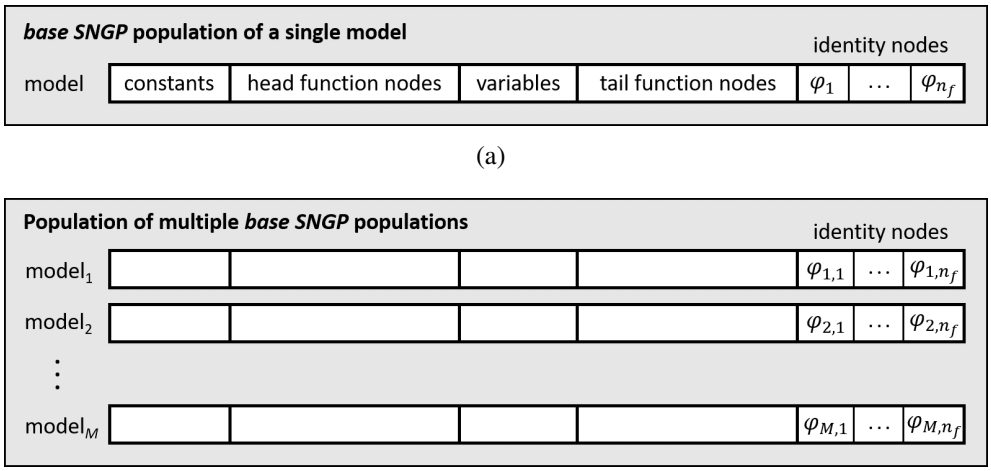


Figure 1: Forms of the evolved population. (a) A population structure of the single-model base SNGP. (b) A multi-model population of multiple base SNGP populations.

SNGP is a graph-based genetic programming technique that evolves a population of individuals, i.e., program nodes, organized in an ordered linear array structure, see Figure 1(a). A program node can be either a terminal, i.e., a constant or a variable, or some function chosen from the set of elementary functions, $\mathcal{F}$, defined by a user for the problem at hand. Nodes are interconnected in a left-to-right manner, meaning that a node can act as an input operand only of the nodes positioned to its right in the population. Thus, the whole population represents a graph structure with multiple tree expressions rooted in the function nodes that represent nonlinear analytic functions. One such population, see Figure 1(a), represents a single analytic model (3) whose features, $\varphi(\mathbf{x})$, are rooted in so-called identity nodes, where each identity node points to some non-constant-output node in the population. The population is evolved through a first-improvement iterative local search procedure using a simple mutation operator, which varies inputs of the function nodes while minimizing the RMSE on the training data set.

3.5. Bi-objective SNGP

The multi-objective SNGP works with a population of models represented by separate base SNGP populations, see Figure 1(b). The population of models is evolved using the NSGA-II algorithm (Deb et al., 2002), which is a representative of domination-based multi-objective evolutionary algorithms. The *domination* principle is defined as follows: A solution $\mathbf{s}^{(1)}$ is said to dominate another solution $\mathbf{s}^{(2)}$, if $\mathbf{s}^{(1)}$ is not worse than $\mathbf{s}^{(2)}$ in any objective and $\mathbf{s}^{(1)}$ is strictly better than $\mathbf{s}^{(2)}$ in at least one objective.

The workflow of the algorithm is sketched in Algorithm 1, on lines 1 through 21. It is a standard evolutionary algorithm with several specifics related to the multi-objective nature of the symbolic regression problem considered:

- Model evaluation – Each time a newly created model is to be evaluated, coefficients β_i are determined first. Here, we use the local search procedure proposed by Kubalík et al. (2020)

which has been shown to outperform the standard least squares method. It uses the concept of *domination* when tuning the coefficients in order for the final model to be optimal w.r.t. both optimization objectives. Then, both performance metrics, C_t and C_c, are calculated for the model.

- Model acceptance – Newly created child model is accepted if and only if it is not dominated by the original parent nor by the current child, lines 17–18.

- Population update – In each generation, an intermediate population of new models, $interPop$, is created and then merged with the current population $population\_I$, lines 7–20, resulting in a new population. While merging the populations, non-dominated models are preferred to the dominated ones and among models of the same non-dominated front the more unique ones are preferred. For more details, please refer to (Deb et al., 2002).

An important aspect of this algorithm relates to the way new models are generated. When a model is selected to produce its child, line 9, its base SNGP population undergoes a standard perturbation through a sequence of mutations, line 15. A newly created model is always 'just' a modified version of its single ancestor. No information from the other models is utilized. There is no way to share or mix their features. In its current form, this algorithm is good at generating fine-tuned high-quality models. On the other hand, it happens very often that at the end of each run, all models in the population are derivatives of the same distant ancestor, a single member of the initial population (line 1). Effectively this means that the algorithm's search scope shrinks to just a single-root tree-based exploration, despite there are multiple seeds in the initial population. This is deemed a limitation, as some portions of the solution space might be omitted.

To remedy this issue, a new parameter A is introduced in the `NSGAII_merge` procedure, which defines the maximum number of models in the population having a common distant ancestor. This way, multiple evolution paths are enforced. In particular, the final population of the mutation-based phase contains descendants of at least M/A different models of the initial population. We will refer to this method in the sequel by the name 'one-phase bi-objective SNGP' and its acronym `BOSNGP-1p`.

4. Proposed method

In this section, the proposed two-phase bi-objective symbolic regression method and the final model selection method are described.

4.1. Two-phase bi-objective SNGP

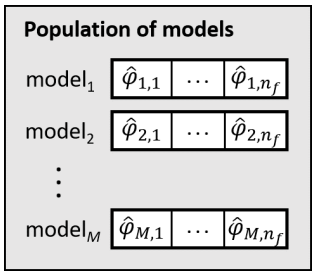
The proposed extension to the one-phase bi-objective SNGP algorithm focuses on a generation of a more diverse set of high-quality models. This is attained by adding a new phase to the algorithm after the first mutation-based one, lines 22–39. It takes as its input all features evolved in the first phase and tries to combine them into new well-performing models. A similar idea is also behind the approach proposed in (Cheng & Zhong, 2020).

Algorithm 1: two-phase bi-objective symbolic regression algorithm

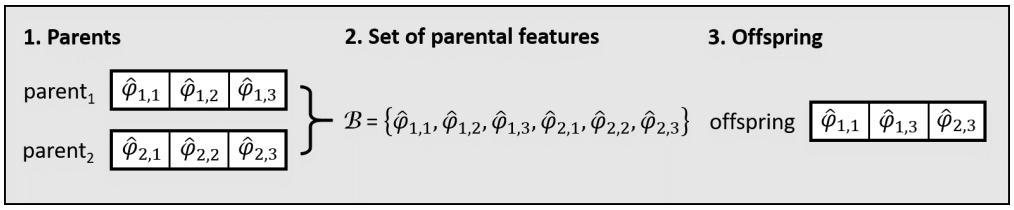
Input: M … size of the population of models
n_f … maximum number of features a model can be composed of
δ … maximum feature depth
A … maximum number of models in the population with a common distant ancestor
D … training data set
C … set of constraint samples
G^I, G^{II} … maximum number of generations in phases I and II
I … maximum number of iterations carried out to produce an offspring model from the parent one

Output: Set of all final models produced in phases I and II

```
/* phase I – mutation-based */
population_I.init(M, n_f, δ)
for ∀model ∈ population_I do
    model.evaluate(D, C)
generation ← 0
while generation < G^I do
    generation ← generation + 1
    interPop ← {}
    while interPop.size() < M do
        parent ← selectTournament(population_I)
        child ← parent.clone()
        i ← 0
        while i < I do
            i ← i + 1
            temp ← child.clone()
            temp.applyMutations()
            temp.evaluate(D, C)
            if !parent.dominates(temp) ∧ !child.dominates(temp) then
                child ← temp
        interPop.add(child)
    population_I ← NSGAII_merge(population_I, interPop, A)
S ← population_I
/* phase II – features mixing */
features ← S.extractFeatures()
population_II.init(population_I)
generation ← 0
while generation < G^II do
    generation ← generation + 1
    interPop ← {}
    while interPop.size() < M do
        parent_1 ← selectTournament(population_II)
        parent_2 ← selectRandom(population_II)
        children ← parent_1.crossWith(parent_2)
        children ← children ∪ parent_1.addFeature(features)
        children ← children ∪ parent_1.replaceFeature(features)
        children ← children ∪ parent_1.removeFeature()
    for ∀model ∈ interPop do
        model.evaluate(D, C)
    population_II ← NSGAII_merge(population_II, interPop)
S ← S ∪ population_II
return S
```



(a)



(b)

Figure 2: (a) A population of models evolved in the feature mixing phase. Models are composed of encapsulated features, $\hat{\varphi}_{i,j}$, that are immutable during this phase. (b) An illustration of the crossover operator used to create a new model as a combination of features of two parental models. The offspring model is composed of two features, $\hat{\varphi}_{1,1}$ and $\hat{\varphi}_{1,3}$, inherited from the first parental model and one feature, $\hat{\varphi}_{2,3}$, inherited from the second parental model.

The whole two-phase bi-objective SNGP algorithm starts with the mutation-based first phase that yields the first set of final models, $\mathcal{S}^I$. At the beginning of the second phase, a pool of features is extracted out of the models in the set $\mathcal{S}^I$, line 22. By the extraction, we mean that the features are encapsulated. Their structure, i.e., their tree form, is frozen; hence the features become immutable for the rest of the run. Then, a starting population of models is initialized. Note, the models now have the form of a set of encapsulated features, $\hat{\varphi}_{i,j}$, see Figure 2(a). Likewise in the first phase, an intermediate population of models is created in each generation, which is afterward merged with the current population of models. The *interPop* is filled in in the loop, lines 28–34, where several models are added to it in each iteration. Two parental models are involved in the process of creating new models. The first parent is chosen by a binary tournament selection method. The other one is chosen randomly. This is to increase the exploration. The first new model is created by mating two parental models. Up to three other models are created by mutation of the first parent. Once the *interPop* has been completed, all of its models are evaluated. In the end, a set of final models produced in both phases is returned.

The crossover operator, see Figure 2(b), used to mix features of two parental models, works in three steps:

1. A collection $\mathcal{B}$ of features present in the two parents is created.
2. A number k is randomly chosen from the interval $\langle 1, n_f \rangle$.
3. A new model is created using k features randomly selected from $\mathcal{B}$.

The following three operators are used to produce a mutated version of the parental model:

- `addFeature(`$features$`)` – If the number of features in the parental model is less than n_f then a new feature randomly chosen from $features$ is added to the model.
- `replaceFeature(`$features$`)` – A randomly chosen feature of the parental model is replaced with another feature randomly chosen from $features$.
- `removeFeature` – If the number of features in the parental model is greater than 1 then a randomly chosen feature of the model is removed from the model.

All of the crossover and mutation operators allow for the evolution of variable-size models in terms of the number of features.

In the sequel, we will refer to this method by the name 'two-phase bi-objective SNGP' and its acronym `BOSNGP-2p`.

4.2. Final model selection

The `BOSNGP-2p` algorithm outputs a large set of models, $\mathcal{S}$, often many of them are non-dominated. This is almost always the case when solving multi-objective optimization using domination-based evolutionary algorithm. The question is how to choose the final model as the ultimate result of the algorithm's run. Without any other knowledge about the desired model's properties, all of the non-dominated models of the set $\mathcal{S}$ are equal.

Here, we propose a method that selects the final model based on two performance metrics – the simulation RMSE calculated on the validation data set (i.e., validation sequence) and the validation C_c calculated on the set of validation constraint samples. Given a sequence of l consecutive data samples of the form $\mathbf{d}_k = (\mathbf{x}_k, \mathbf{u}_k, x_{k+1})$, the simulation RMSE is calculated as

$$RMSE_{sim} = \sqrt{\frac{1}{l}\sum_{k=1}^{l}(f(\mathbf{x}_k, \mathbf{u}_k) - x_{k+1})^2}, \tag{4}$$

where $f(\mathbf{x}_k, \mathbf{u}_k)$ is an analytic model for a particular state variable x and the value of that x in $\mathbf{x}_k$ is set to the model's response from the previous simulation step $f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1})$ for all steps $k = 2 \ldots l$. The set of validation constraint samples is generated in the same way as the constraint samples used in the model learning phase.

There are many possible ways to choose the final model. Some trade-off solution is typically a right choice as the solutions with the best value of one performance criterion are doing very poorly with respect to the other criteria. Still, the question is which of the trade-off solutions to choose. Here, we propose a method that chooses a model with a small simulation RMSE and small constraint violation at the same time.

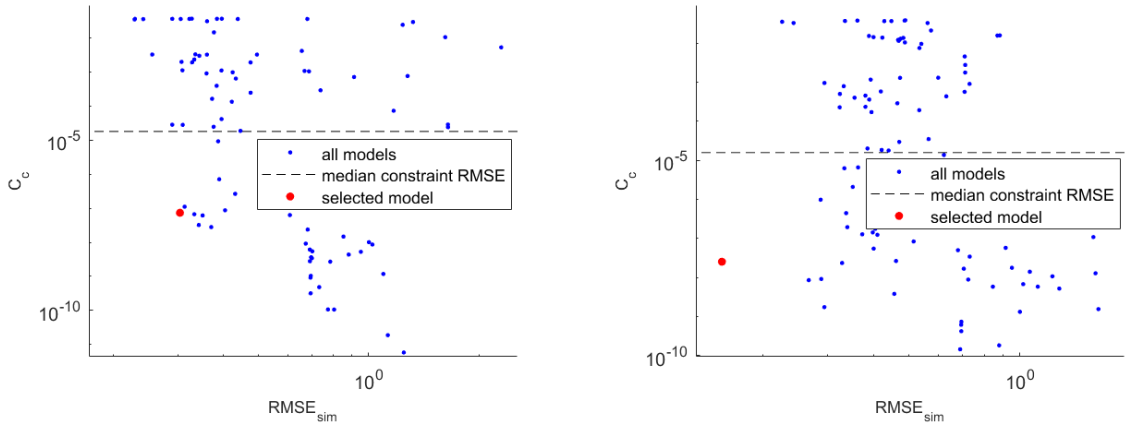
The pseudo-code of the method is listed in Algorithm 2. It takes the whole pool of models $\mathcal{S}$ as its input. First, the set of models is sorted according to the RMSE_{sim} on the validation data set in ascending order. Then, a median constraint violation value of the set $\mathcal{S}$ is found. Finally, the models are searched starting from the one with the best RMSE_{sim} value. The first model met that has its constraint violation value smaller than the population median is chosen as the final model.

There are several nice properties of the method. First, it always selects a model on the non-dominated front. Second, a position of the selected model is not explicitly bound to any specific

Algorithm 2: Final model selection

Input: $\mathcal{S}$ … Set of all final models produced in phases I and II
Output: Selected trade-off model

```
1 S ← S.sortBySimulationError()
2 m ← S.getMedianConstraintViolation()
3 i ← 0
4 while S[i].C_c > m do
5     i ← i + 1
6 return S[i]
```



(a) (b)

Figure 3: Final model selection. (a) shows the case the selected model is a trade-off solution. (b) shows the case the selected model is an extreme solution. Both examples are from real experiments on the drone problem.

region of the non-domination front, like the middle of the front, etc. This is demonstrated in Figure 3. Any of the non-dominated models can be selected. It just has to do well in terms of the RMSE_{sim}, which is the primary criterion and have the constraint violation no worse than the population median.

The method uses the median statistic, so it should be given a sample as large as possible to get a reliable estimate of that statistic. That is why it works with the whole set $\mathcal{S}$ even that only a non-dominated model is to be delivered in the end. Note that the non-dominated front often covers the whole set of models very unevenly. The front itself might fail to capture the real distribution of the constraint violation values in $\mathcal{S}$ reliably. Thus, if only the models on the non-dominated front were used, the selection process might be biased inappropriately.

5. Experiments

In this section, the experimental setup is described. Firstly, three real robotic systems – the TurtleBot 2 mobile robot, the Parrot Bebop 2 drone, and the magnetic manipulation system – chosen to demonstrate the performance of the methods are described, including the training, validation, and test data sets and the types of prior knowledge used in the `BOSNGP-1p` and `BOSNGP-2p`

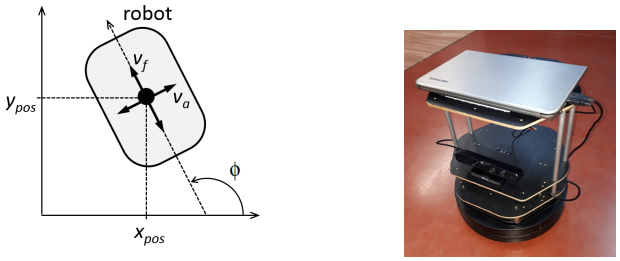
methods. Then, the compared algorithms are listed together with their configurations used in the experiments. Finally, the evaluation scheme used to analyze the algorithms' performance is described.

5.1. Turtlebot

5.1.1. System description

The state of the two-wheel mobile robot (Fig. 4) is described by the state vector $\mathbf{x} = (x_{pos}, y_{pos}, \phi)^\top$, where x_{pos} and y_{pos} are the position coordinates of the robot and ϕ is the robot's heading. The control input is $\mathbf{u} = (v_f, v_a)^\top$, with v_f and v_a the desired forward and angular velocity, respectively. Models for the three state variables have the following form

$$\begin{aligned} x_{pos,k+1} &= f^{x_{pos}}(x_{pos,k}, y_{pos,k}, \phi_k, v_{f,k}, v_{\mathrm{a},k}), \\ y_{pos,k+1} &= f^{y_{pos}}(x_{pos,k}, y_{pos,k}, \phi_k, v_{f,k}, v_{a,k}), \\ \phi_{k+1} &= f^{\phi}(x_{pos,k}, y_{pos,k}, \phi_k, v_{f,k}, v_{a,k}). \end{aligned}$$



(a) (b)

Figure 4: TurtleBot mobile robot. A schematic (a) and a photo of the system (b).

5.1.2. Data sets

We used experimental data collected during an operation of the real robot. Five sequences of samples starting from the initial state $x_0 = (0,0,0)^\top$ were generated with a sampling period $T_s = 0.2\,\mathrm{s}$. In each sequence, we steered the robot by 30 different pairs of random inputs v_f, v_a, where we kept each pair of inputs constant for 5 samples. This yielded 150 samples per sequence. The random inputs were drawn from the domain $v_f \in [0, 0.3]\,\mathrm{m \cdot s^{-1}}$, $v_a \in [-1, 1]\,\mathrm{rad \cdot s^{-1}}$. Of these five sequences, training, validation and test data sets were created for each state variable. A randomly chosen sequence was used for the training data set, another one for the validation data set and the remaining three sequences were used for the test data sets.

5.1.3. Prior knowledge

All the prior knowledge defined for the TurtleBot is of the invariant type. When a model for a particular variable is evaluated on the relevant constraint sample, it should always output the value equal to the original value of the variable. The following four types of prior knowledge about the TurtleBot were used:

a) Steady-state behavior: If the control inputs, v_f and v_a, are set to zero, then the robot may change neither its position nor its heading. This is represented by the following equality constraints

$$x_{pos} = f^{x_{pos}}(x_{pos}, y_{pos}, \phi, 0, 0) \quad \text{and} \quad y_{pos} = f^{y_{pos}}(x_{pos}, y_{pos}, \phi, 0, 0)\,.$$

b) Axis-parallel moves: If the robot makes axis-parallel moves, then its x_{pos} or y_{pos} may not change. This is represented by the following equality constraints

$$x_{pos} = f^{x_{pos}}(x_{pos}, y_{pos}, -\pi/2, v_f, 0)\,, \quad x_{pos} = f^{x_{pos}}(x_{pos}, y_{pos}, \pi/2, v_f, 0)$$

and

$$y_{pos} = f^{y_{pos}}(x_{pos}, y_{pos}, 0, v_f, 0)\,, \quad y_{pos} = f^{y_{pos}}(x_{pos}, y_{pos}, \pi, v_f, 0)\,.$$

The former two cases are for the moves that are parallel to the y-axis and the latter two are for the x-axis parallel moves.

c) Turning on the spot: If the forward velocity is zero, the robot may not change its position. This is represented by the following equality constraints

$$x_{pos} = f^{x_{pos}}(x_{pos}, y_{pos}, \phi, 0, v_a) \quad \text{and} \quad y_{pos} = f^{y_{pos}}(x_{pos}, y_{pos}, \phi, 0, v_a)\,.$$

d) Straight forward moves: This prior knowledge states that the robot may not change its heading if the angular velocity control input is zero. This is represented by the following equality constraint

$$\phi = f^{\phi}(x_{pos}, y_{pos}, \phi, v_f, 0)\,.$$

The values of the state variables x_{pos}, y_{pos}, ϕ, and of the control inputs v_f and v_a were randomly sampled within the same limits as for the training data. We generated 50 constraint samples for each prior knowledge type, so 200 samples in total.

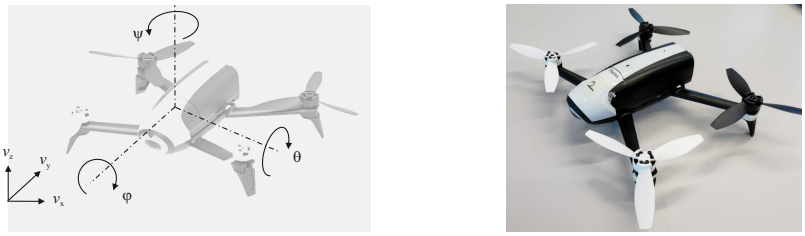
5.2. Drone

5.2.1. System description

The state of the drone is described by the state vector $\mathbf{x} = (v_x, v_y, v_z, \theta, \varphi, \psi)^\top$, where v_x, v_y and v_z are the translational velocities measured by the OptiTrack motion-capture system in the fixed world frame and θ, φ and ψ are the body angles, denoting the pitch, roll and yaw, respectively. The drone is controlled by the input vector $\mathbf{u} = (\theta_c, \varphi_c, \omega_c, v_{z_c})^\top$, which denotes the desired pitch, roll, and yaw rates, respectively, and the vertical velocity. Figure 5 shows a schematic and a photo of the drone.

While we have applied the method to build prediction models for all the measured variables, here we present only the models for the translational velocities that are of the following form

$$v_{x,k+1} = f^{v_x}(v_{x,k}, \theta_k, \varphi_k, \psi_k)\,,$$
$$v_{y,k+1} = f^{v_y}(v_{y,k}, \theta_k, \varphi_k, \psi_k)\,.$$



(a) (b)

Figure 5: Parrot Bebop 2: A schematic (a) and a photo of the system (b).

The main reason is that the vertical velocity and the angles can be accurately predicted by using linear models. Moreover, the experimental data have been recorded in experiments where the vertical velocity and the yaw remained almost constant, which does not allow us to assess their model accuracy. Note that the model for the given translational velocity does not take the other one as its input since the model's output is fully determined by the translational velocity in question plus the drone angles (Zhu & Alonso-Mora, 2019).

5.2.2. Data sets

We have collected various data sets by teleoperating the drone:

- Training data set – A data sequence was collected while controlling the drone to follow an 8-shape flight trajectory by using a model-predictive controller. The training data set is composed of 160 consecutive data samples taken from this sequence that cover one flight through the 8-shape trajectory.

- Validation data set – Similarly, the validation data set was composed of 160 samples collected in another independent flight through the 8-shape trajectory.

- Test data sets – Three test data set were generated. The first one is a sequence of 250 samples measured while controlling the drone to follow an approximately square flight trajectory by using model-predictive control. In the sequel, we will refer to this test data set by the name 'test-square'.

  The other two data sequences were generated while heading the drone parallel to the x-axis of the coordinate system and periodically changing a single control input, θ_c and φ_c, respectively. The values of the control input alter between the maximum and minimum value every 30 time steps and stay constant during the period. This way, a sequence is generated where the drone moves sideways as the φ_c swaps between the maximum and minimum value while θ_c is zero. The drone is supposed to make large moves in the y-axis and minimal moves in the x-axis. Similarly, a sequence is generated for the drone tilting forth and back as the θ_c swaps between the maximum and minimum value while φ_c is zero. This time, the drone makes much larger moves in direction of the x-axis than in the y-axis. The test data set denoted as 'test-pitch' consists of 200 samples spanning over several periods of altering the pitch control input. Similarly, the test data set denoted as 'test-roll' consists of 150 samples covering several periods of altering the roll control input.

When collecting the experimental data, we used the sampling period $T_s = 0.05\,\mathrm{s}$. We have chosen a shorter sampling period than for the mobile robot to capture the faster dynamics of the drone.

5.2.3. Prior knowledge

Similarly to the turtlebot, the invariant type of prior knowledge was defined for the drone. For both translational velocities considered, the prior knowledge describes the situations in which one of the translational velocities stays zero while the other one varies. The following two types of prior knowledge were used:

a) Sideways moves: The drone is oriented parallel either to the x-axis or to the y-axis, it has zero pitch (i.e., it does not move forward or backward), and the roll φ is sampled randomly from the interval $[-\pi/15, \pi/15]$ rad. This is represented by the following equality constraints

$$0 = f^{v_x}(0,0,\varphi,0)\,, \quad 0 = f^{v_x}(0,0,\varphi,\pi)\,,$$
$$0 = f^{v_y}(0,0,\varphi,-\pi/2)\,, \quad 0 = f^{v_y}(0,0,\varphi,\pi/2)\,.$$

b) Forward and backward moves: Again, the drone is oriented parallel either to the x-axis or to the y-axis, it has zero roll, and the pitch θ is sampled randomly from the interval $[-\pi/15, \pi/15]$ rad. So, the drone moves in direction of the respective axis while its velocity in the other axis is zero. This is represented by the following equality constraints

$$0 = f^{v_x}(0,\theta,0,-\pi/2)\,, \quad 0 = f^{v_x}(0,\theta,0,\pi/2)\,,$$
$$0 = f^{v_y}(0,\theta,0,0)\,, \quad 0 = f^{v_y}(0,\theta,0,\pi)\,.$$

For each prior knowledge type, 100 samples were generated, resulting in 200 constraint samples in total.

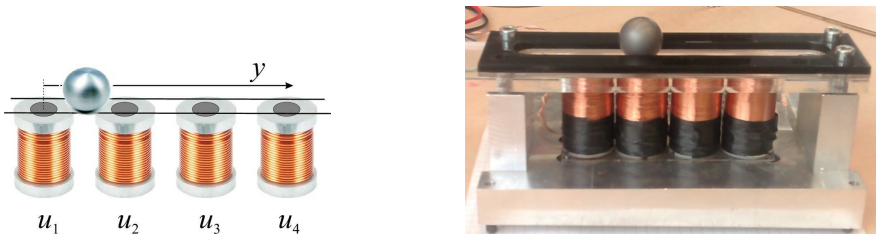
5.3. Magman

5.3.1. System description

The magnetic manipulation system (magman) consists of an iron ball moving along a rail with four electromagnets under the rail. Here, we consider the scenario that only one of the four coils can be activated at a time. The goal is to find a transition model in the form

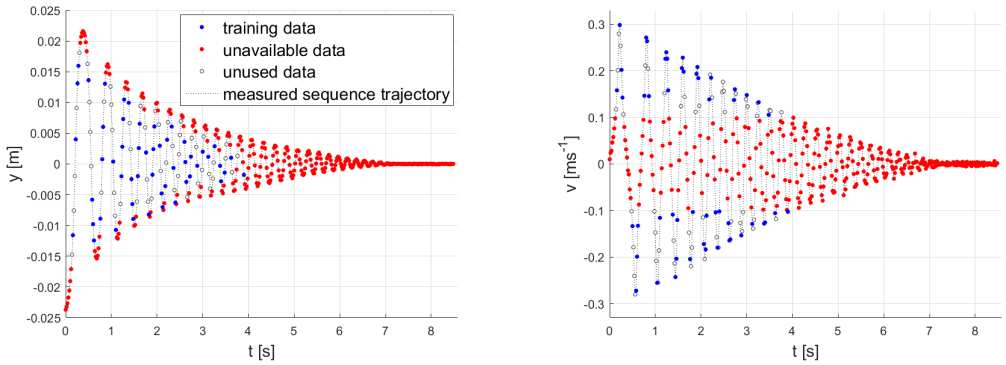
$$y_{k+1} = f^{y}(y_k, v_k)\,,$$

where y is the distance between the iron ball and the activated coil and v is the current velocity of the ball. Since a constant current through the coil is always applied, we do not include this control input into the transition model. Figure 6 shows a schematic and a photo of the system.



(a) (b)

Figure 6: Magman: A schematic (a) and a photo of the system (b).



(a) (b)

Figure 7: Training data for (a) y and (b) v extracted from the data sequence measured for the first coil. Red dots are the samples with $v < 0.1\,\mathrm{m}\cdot\mathrm{s}^{-1}$, which are not considered for training. Out of the remaining samples, half of them are randomly selected for the training data set (the blue dots) leaving the other half unused (the black circles).

5.3.2. Data sets

Experimental data for each coil were measured using the following scenario. Firstly, the ball is positioned within a certain distance from the respective coil. Then, the coil is activated with a constant current of 0.6 A until the ball settles exactly above the activated coil and stays still there. The data were collected using a sampling period of $T_s = 0.02\,\mathrm{s}$.

- Training data set – Here, we emulate a situation when only limited training data are available. This might happen for many reasons. For example, the data insufficiency can be caused by a rather low resolution of the measurement equipment. Or it might be impossible to excite the system in all regions of the state-action space. We prepared the training data from a sequence measured when the third coil was activated. Out of the sequence, the samples with $v < 0.1\,\mathrm{m}\cdot\mathrm{s}^{-1}$, which represent the "unavailable data", were filtered out. From the remaining data, samples chosen randomly with the probability of 0.5 were used in the training data set, see Figure 7. The resulting training data set contains 52 samples.

- Validation data set – As the validation data set, another data sequence measured on the third coil was used. This is a complete sequence of 425 samples.

- Test data sets – Three test data sets were used. These are complete sequences, each containing 421 samples, measured on the first, second, and fourth activated coil.

5.3.3. Prior knowledge

A single prior knowledge type was used in the `magman` experiments. It describes the steady-state behavior that if the ball is located right above the activated coil, i.e., $y = 0\,\mathrm{m}$ and its velocity is $v = 0\,\mathrm{m}\cdot\mathrm{s}^{-1}$, it should not change its position. This is represented by the following equality constraint

$$0 = f^{y}(0,0)\,.$$

5.4. Compared algorithms

Three methods were compared:

- `Baseline` – base SNGP described in Section 3.4 that minimizes only the RMSE on the training data set.
- `BOSNGP-1p` – one-phase bi-objective SNGP described in Section 3.5 that carries out just the mutation-based phase of Algorithm 1.
- `BOSNGP-2p` – the proposed two-phase bi-objective SNGP algorithm described in Section 4.1 that adds the model-mixing phase to the one-phase bi-objective SNGP algorithm.

The algorithms were tested with the following parameter setting:

- base SNGP population size 400, $\delta = 7$, $M = 50$, $A = 5$, $I = 50$, $n_f = 10$ (`turtlebot`), $n_f = 5$ (`drone`, `magman`), $G^I = 100$ (`Baseline`, `BOSNGP-1p`), $G^I = 50$ (`BOSNGP-2p`), $G^{II} = 0$ (`Baseline`, `BOSNGP-1p`), $G^{II} = 50$ (`BOSNGP-2p`)
- Elementary functions:
  - `turtlebot`: $\mathcal{F} = \{+, -, *, \mathit{cube}, \mathit{sin}, \mathit{cos}, \mathit{sign}\}$
  - `drone`: $\mathcal{F} = \{+, -, *, \mathit{square}, \mathit{cube}, \mathit{sin}, \mathit{cos}, \mathit{tan}\}$
  - `magman`: $\mathcal{F} = \{+, -, *, \mathit{cube}, \mathit{sin}, \mathit{cos}\}$

The values of the parameters were either adopted from (Kubalík et al., 2020) or chosen based on our previous experience with the SNGP algorithm. All of the algorithms run for 100 generations in total. That value has been chosen as it proved to be sufficient for all algorithms to converge on all test problems. The number of features n_f was set to 10 for the `turtlebot` problem and to 5 on the other two problems to reflect the problem complexity. The elementary function sets $\mathcal{F}$ have been chosen differently for each problem based on the problem's nature and properties.

We did not use any parameter tuning technique to find optimal values of the parameters. Rather than focusing on achieving the best possible results, we want to analyze the relative performance of the three methods compared.

5.5. Evaluation

Fifty independent runs were carried out with each method on each problem. Firstly, we analyze the distribution of models produced by one-phase bi-objective SNGP and two-phase bi-objective SNGP, respectively. This analysis is to show whether the proposed model mixing phase generates different models than the mutation-based phase alone and leads to a richer pool of models produced in one run.

Then, a set of final models is created for each method and each problem as a collection of models selected using the method described in Section 4.2 from all of the fifty runs. Finally, the median training RMSE, median RMSE_{sim} on the validation data set, median RMSE_{sim} on the test data sets, and median validation C_c of the models in each final set are calculated. These values are used to compare the methods w.r.t. the quality of selected final models.

For analyses of the drone experiments, another method called `Gray-box` was added to the set of compared methods. `Gray-box` uses a discrete-time form of a simplified nonlinear physical model of the drone (Zhu & Alonso-Mora, 2019):

$$v_{x,k+1} = a_x v_{x,k} + b_x(\cos\psi_k \frac{\tan\theta_k}{\cos\phi_k} + \sin\psi_k \tan\phi_k)g \quad (5)$$

$$v_{y,k+1} = a_y v_{x,k} + b_y(\sin\psi_k \frac{\tan\theta_k}{\cos\phi_k} - \cos\psi_k \tan\phi_k)g \quad (6)$$

where g is the acceleration due to gravity and a_x, a_y, b_x, b_y are coefficients estimated by ordinary least squares, using the training data set. We refer to this model as `Gray-box`, as it is a combination of approximate physical relations with parameters estimated from data. Unlike the SNGP-based methods, the `Gray-box` method is deterministic and produces just a single solution.

In order to assess the statistical significance of the differences among the methods, we analyze them in a pair-wise manner using the Wilcoxon rank-sum test, which rejects the null hypothesis that the compared results sets are sampled from continuous distributions with equal medians at the 5 % significance level.

6. Results

Firstly, we analyze the effect of the second phase introduced in Section 4.1. Figure 8 shows a distribution of models generated for the variable x_{pos} of the `turtlebot` problem by `BOSNGP-1p` (the green dots) and by the second phase of `BOSNGP-2p` (the blue dots). The two sets of models have been accumulated over fifty runs of the algorithms. It clearly illustrates the difference between the operation mode when only the mutation-based phase is run and the mode when the second half of generations is replaced by the phase mixing the models found in the first phase. In this case, the models produced in the second phase of `BOSNGP-2p` have better median statistics than the `BOSNGP-1p` models. They are better in validation RMSE_{sim} as well as in validation C_c. Note, this does not hold generally for all problem-variable cases as shown in Table 1. However, an important observation is that mixing the models can help to efficiently generate useful models that may not be discovered just by the first mutation-based phase.

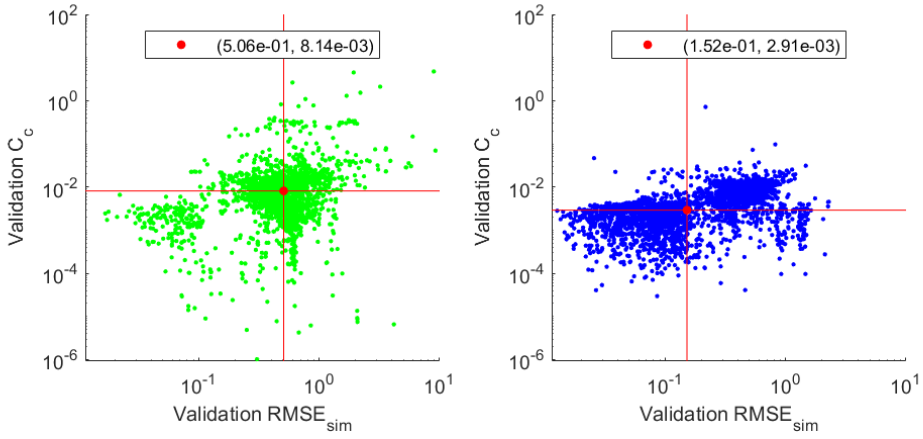


Figure 8: A pool of all final models created for the state variable x_{pos} of the `turtlebot` problem in 50 runs of `BOSNGP-1p` and `BOSNGP-2p`. Green dots are models produced by `BOSNGP-1p`. Blue dots are models produced in the second phase of `BOSNGP-2p`. The median-median model is marked with the red dot.

Table 1: Statistics of the pool of models produced by `BOSNGP-1p` and `BOSNGP-2p`. Median validation C_c and median RMSE_{sim} on validation data set over all final models created in fifty runs are presented. The best values are highlighted in bold. Cells with significantly better values are colored in gray.

		phase I		phase II	
problem	var	validation RMSE_{sim}	validation C_c	validation RMSE_{sim}	validation C_c
`turtlebot`	x_{pos}	$5.06 \cdot 10^{-1}$	$8.14 \cdot 10^{-3}$	$\mathbf{1.52 \cdot 10^{-1}}$	$\mathbf{2.91 \cdot 10^{-3}}$
	y_{pos}	$3.53 \cdot 10^{-1}$	$8.91 \cdot 10^{-3}$	$\mathbf{1.05 \cdot 10^{-1}}$	$\mathbf{1.42 \cdot 10^{-3}}$
	ϕ	$1.48 \cdot 10^{0}$	$2.94 \cdot 10^{-2}$	$\mathbf{1.44 \cdot 10^{0}}$	$\mathbf{2.28 \cdot 10^{-2}}$
`drone`	v_x	$7.74 \cdot 10^{-1}$	$\mathbf{2.71 \cdot 10^{-7}}$	$\mathbf{1.69 \cdot 10^{-1}}$	$1.86 \cdot 10^{-3}$
	v_y	$6.95 \cdot 10^{-1}$	$\mathbf{6.57 \cdot 10^{-5}}$	$\mathbf{2.03 \cdot 10^{-1}}$	$1.22 \cdot 10^{-3}$
`magman`	y	$\mathbf{2.58 \cdot 10^{-3}}$	$\mathbf{9.01 \cdot 10^{-10}}$	$3.57 \cdot 10^{-3}$	$1.1 \cdot 10^{-8}$

6.1. Turtlebot

Results obtained on the `turtlebot` problem are summarized in Table 2. There are several observations. The `Baseline` has the best training RMSE out of all compared methods. This is not surprising as the `Baseline` focuses purely on the minimization of this measure while paying no attention to other properties of the evolved models. Thus, the `Baseline` is by one up to two orders of magnitude worse than `BOSNGP` methods in terms of the validation C_c. `BOSNGP-2p` is the best and significantly better than `BOSNGP-1p` in terms of the validation RMSE_{sim} for all variables. As for the test performance, both `BOSNGP` methods are significantly better than `Baseline` on all test data sets and all variables with the only exception of the second test data set and the variable x_{pos}. This is also illustrated in Figure 9 where the performance of models with median test RMSE_{sim} is shown. Clearly, the `BOSNGP` models are good at capturing the right shape of the measured sequence trajectory throughout the whole 30-second long experiment while the `Baseline` models exhibit large deviations from the measured data characteristics.

Table 2: Comparison on the `turtlebot` problem. The best values are highlighted in bold. If one `BOSNGP` method is significantly better than the other, then the respective cell of the better one is colored in gray.

var	method	training RMSE	validation RMSE_{sim}	validation C_c	test 1 RMSE_{sim}	test 2 RMSE_{sim}	test 3 RMSE_{sim}
x_{pos}	`Baseline`	$\mathbf{1.56 \cdot 10^{-3}}$	$9.38 \cdot 10^{-2}$	$1.86 \cdot 10^{-2}$	$9.43 \cdot 10^{-1}$	$1.44 \cdot 10^{-1}$	$2.74 \cdot 10^{0}$
	`BOSNGP-1p`	$2.62 \cdot 10^{-3}$	$5.06 \cdot 10^{-2}$	$2.39 \cdot 10^{-3}$	$1.04 \cdot 10^{-1}$	$\mathbf{1.21 \cdot 10^{-1}}$	$\mathbf{9.79 \cdot 10^{-2}}$
	`BOSNGP-2p`	$2.87 \cdot 10^{-3}$	$\mathbf{3.79 \cdot 10^{-2}}$	$\mathbf{2.04 \cdot 10^{-3}}$	$\mathbf{9.38 \cdot 10^{-2}}$	$1.44 \cdot 10^{-1}$	$1.27 \cdot 10^{-1}$
y_{pos}	`Baseline`	$\mathbf{1.40 \cdot 10^{-3}}$	$8.2 \cdot 10^{-2}$	$2.61 \cdot 10^{-2}$	$8.86 \cdot 10^{-1}$	$2.83 \cdot 10^{-1}$	$8.53 \cdot 10^{-2}$
	`BOSNGP-1p`	$2.0 \cdot 10^{-3}$	$4.8 \cdot 10^{-2}$	$7.88 \cdot 10^{-4}$	$1.11 \cdot 10^{-1}$	$\mathbf{6.08 \cdot 10^{-2}}$	$5.23 \cdot 10^{-2}$
	`BOSNGP-2p`	$2.64 \cdot 10^{-3}$	$\mathbf{2.5 \cdot 10^{-2}}$	$\mathbf{4.28 \cdot 10^{-4}}$	$\mathbf{6.01 \cdot 10^{-2}}$	$1.05 \cdot 10^{-1}$	$\mathbf{3.85 \cdot 10^{-2}}$
ϕ	`Baseline`	$\mathbf{2.83 \cdot 10^{-2}}$	$2.47 \cdot 10^{0}$	$5.26 \cdot 10^{-1}$	$6.91 \cdot 10^{0}$	$5.32 \cdot 10^{0}$	$1.99 \cdot 10^{0}$
	`BOSNGP-1p`	$3.51 \cdot 10^{-2}$	$3.28 \cdot 10^{-1}$	$\mathbf{7.5 \cdot 10^{-3}}$	$9.29 \cdot 10^{-1}$	$5.8 \cdot 10^{-1}$	$4.72 \cdot 10^{-1}$
	`BOSNGP-2p`	$3.49 \cdot 10^{-2}$	$\mathbf{1.94 \cdot 10^{-1}}$	$8.8 \cdot 10^{-3}$	$\mathbf{7.67 \cdot 10^{-1}}$	$\mathbf{4.59 \cdot 10^{-1}}$	$\mathbf{2.84 \cdot 10^{-1}}$

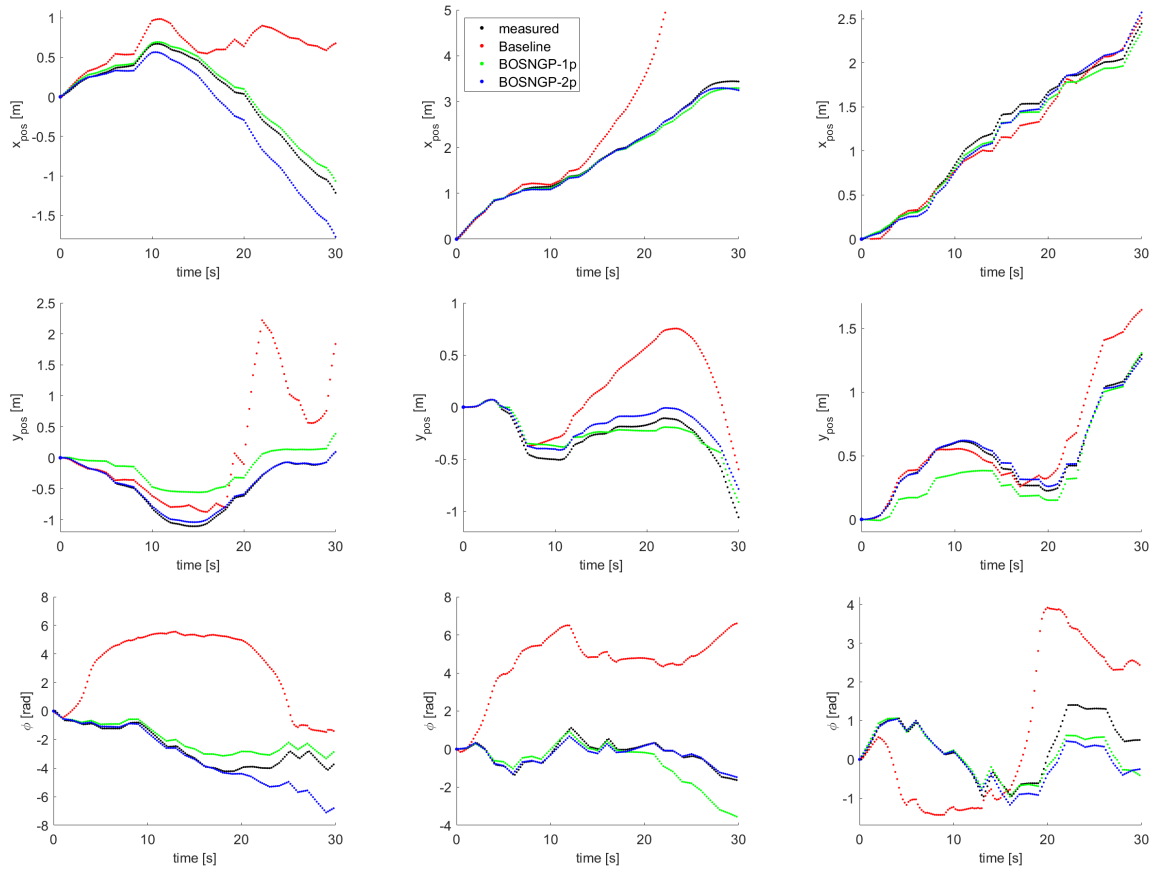


Figure 9: Simulations with selected models of state variables x_{pos}, y_{pos}, and ϕ on three test sequences of the `turtlebot` problem. Models with the test RMSE_{sim} value closest to the median value presented in Table 2 are selected for each state variable. From left to right are simulations on test 1, test 2, and test 3 data set.

Note that the compliance with the measured data in terms of the direction of change in the model's output value is an important indicator of the model's performance.

6.2. Drone

Table 3 shows performance statistics obtained on the `drone` problem. They differ from the `turtlebot` in the trend observed on the validation and test data. Here, the `Baseline` achieves significantly better validation RMSE_{sim} than the `BOSNGP` methods. It has also better test RMSE_{sim} for v_x models on the test-square data and for v_y models on the test-roll data. However, this comes at the cost of large validation C_c values that are for the `Baseline` by several orders of magnitude higher than for `BOSNGP` methods. This fact is also reflected in the results obtained on the test-roll and test-pitch data. In particular, the case of v_x models on test-roll data and the case of v_y models on test-pitch data, respectively. Note, in both cases the drone is oriented parallel to the x-axis. In the former case, the control input is such that the roll is set to a large positive or negative value while the pitch is kept close to zero making the drone move substantially in direction of y-axis while moving significantly less in direction of the x-axis. The opposite holds for the latter case. We can see, the `BOSNGP` models have significantly smaller error under these test scenarios. This can be attributed to the type of prior knowledge used in the `BOSNGP` methods, see Section 5.2.3. This prior knowledge reflects the fact that a drone oriented parallel to x-axis should move neither forward/backward if the pitch is zero nor sideways if the roll is zero.

The `Gray-box` models are by far the best in terms of the validation C_c, which is in agreement with our expectation. As for the training RMSE and validation RMSE_{sim}, the `Gray-box` performs comparably to `BOSNGP` methods. On test-roll and test-pitch data it is competitive with other methods and is doing very well under the two scenarios described above, the v_x on test-roll and v_y on test-pitch. Again, this can be attributed to its very good compliance with prior knowledge. However, on the test-square data, the `Gray-box` performs the worst and the second to worst, respectively, of all the methods.

Table 3: Comparison on the `drone` problem. The best values are highlighted in bold. If one `BOSNGP` method is significantly better than the other, then the respective cell of the better one is colored in gray.

var	method	training RMSE	validation RMSE_{sim}	validation C_c	test-square RMSE_{sim}	test-roll RMSE_{sim}	test-pitch RMSE_{sim}
vx	`Baseline`	$\mathbf{1.77 \cdot 10^{-3}}$	$\mathbf{1.27 \cdot 10^{-2}}$	$7.3 \cdot 10^{-1}$	$\mathbf{1.54 \cdot 10^{-1}}$	$6.51 \cdot 10^{-1}$	$3.75 \cdot 10^{-1}$
	`Gray-box`	$5.95 \cdot 10^{-3}$	$2.56 \cdot 10^{-1}$	$\mathbf{3.72 \cdot 10^{-18}}$	$3.72 \cdot 10^{-1}$	$\mathbf{1.2 \cdot 10^{-1}}$	$\mathbf{2.05 \cdot 10^{-1}}$
	`BOSNGP-1p`	$7.89 \cdot 10^{-3}$	$2.63 \cdot 10^{-1}$	$2.91 \cdot 10^{-8}$	$4.0 \cdot 10^{-1}$	$1.45 \cdot 10^{-1}$	$3.05 \cdot 10^{-1}$
	`BOSNGP-2p`	$6.18 \cdot 10^{-3}$	$1.29 \cdot 10^{-1}$	$3.46 \cdot 10^{-6}$	$2.74 \cdot 10^{-1}$	$1.57 \cdot 10^{-1}$	$3.2 \cdot 10^{-1}$
vy	`Baseline`	$\mathbf{1.88 \cdot 10^{-3}}$	$\mathbf{1.91 \cdot 10^{-2}}$	$1.69 \cdot 10^{-1}$	$2.87 \cdot 10^{-1}$	$\mathbf{1.55 \cdot 10^{-1}}$	$8.13 \cdot 10^{-1}$
	`Gray-box`	$7.24 \cdot 10^{-3}$	$3.73 \cdot 10^{-1}$	$\mathbf{3.88 \cdot 10^{-18}}$	$4.87 \cdot 10^{-1}$	$2.41 \cdot 10^{-1}$	$3.95 \cdot 10^{-1}$
	`BOSNGP-1p`	$8.12 \cdot 10^{-3}$	$3.2 \cdot 10^{-1}$	$9.49 \cdot 10^{-8}$	$4.12 \cdot 10^{-1}$	$2.61 \cdot 10^{-1}$	$\mathbf{3.13 \cdot 10^{-1}}$
	`BOSNGP-2p`	$5.72 \cdot 10^{-3}$	$1.19 \cdot 10^{-1}$	$6.69 \cdot 10^{-6}$	$\mathbf{2.60 \cdot 10^{-1}}$	$4.23 \cdot 10^{-1}$	$3.39 \cdot 10^{-1}$

Table 4: Comparison on the magman problem. The best values are highlighted in bold. If one BOSNGP method is significantly better than the other, then the respective cell of the better one is colored in gray.

var	method	training RMSE	validation RMSE_{sim}	validation C_c	test 1 RMSE_{sim}	test 2 RMSE_{sim}	test 3 RMSE_{sim}
y	Baseline	$\mathbf{1.17 \cdot 10^{-4}}$	$2.08 \cdot 10^{-3}$	$2.75 \cdot 10^{-5}$	$2.02 \cdot 10^{-3}$	$1.95 \cdot 10^{-3}$	$2.0 \cdot 10^{-3}$
	BOSNGP-1p	$3.91 \cdot 10^{-4}$	$1.8 \cdot 10^{-3}$	$\mathbf{7.64 \cdot 10^{-11}}$	$1.69 \cdot 10^{-3}$	$\mathbf{1.61 \cdot 10^{-3}}$	$1.57 \cdot 10^{-3}$
	BOSNGP-2p	$3.81 \cdot 10^{-4}$	$\mathbf{1.55 \cdot 10^{-3}}$	$1.17 \cdot 10^{-9}$	$\mathbf{1.51 \cdot 10^{-3}}$	$1.64 \cdot 10^{-3}$	$\mathbf{1.48 \cdot 10^{-3}}$

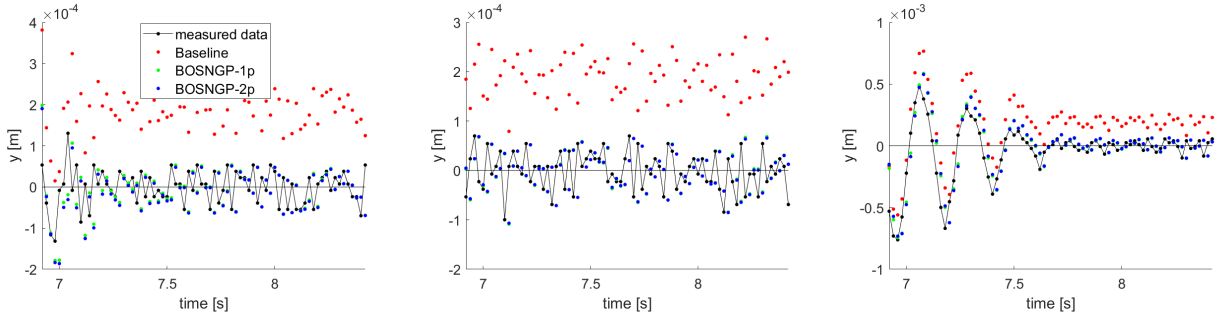


Figure 10: The end phase of simulations carried out on the three test sequences of the magman problem. The last 1.5 s of the simulations are shown.

6.3. Magman

Results obtained on the magman problem are in Table 4. The observations are the same as for the turtlebot problem. Baseline has the best training RMSE at the cost of inferior validation C_c. BOSNGP methods outperform Baseline in terms of both the validation as well as the test RMSE_{sim}.

The positive effect of using the prior knowledge in the BOSNGP methods on the steady-state behavior of the evolved models is clearly demonstrated in Figure 10. It shows the end phase, i.e., the last 1.5 s, of simulations carried out on the three test sequences. Models with the validation RMSE_{sim} value closest to the median value presented in Table 4 are used for each method. One can see that models produced by the Baseline method are inaccurate when the ball gets stabilized above the activated coil. The ball settles with a noticeable offset from the desired position $y = 0$. On the contrary, models generated by the BOSNGP methods bring the ball precisely above the activated coil. For all the three test sequences the median RMSE_{sim} calculated over the last 1.5 s for the Baseline is significantly worse than the error calculated for both BOSNGP methods.

7. Conclusions

We proposed an extension to the multi-objective symbolic regression algorithm that optimizes models with respect to the training accuracy and the compliance with prior knowledge about the properties of the sought model simultaneously. It adds to the original algorithm a phase that generates models via mixing features of models produced in the first phase.

Out of the whole set of candidate output models, the final one is then automatically chosen using the proposed non-parametric selection method. This is an important feature as in multi-objective optimization the final solution is typically selected with the use of some expert knowledge. That is the case of the original algorithm as well.

The method has been experimentally evaluated on three real physical systems – the mobile robot TurtleBot 2, the Parrot Bebop 2 drone, and the magnetic manipulation system, and compared to the baseline method that is a conventional symbolic regression not using the prior knowledge. The results clearly demonstrate its capability to produce a diverse set of high-quality models. On the turtlebot and magman problems, the models produced by the multi-objective method achieve significantly better simulation RMSE than the baseline models. On the drone problem, the baseline models are better in terms of the simulation RMSE. However, the proposed method always produces models that are much more physically justified than the baseline models. The improvement factor is one to seven orders of magnitude.

In our future research, we will investigate ways to automatically distill the prior knowledge from a simple (approximate) physical model of the system. If the differential equations are known, some solutions will be independent of the parameter values and can be used as prior knowledge. This seems quite straightforward for mobile robots, both on the ground and in the air, and a little more involved for manipulators or walking robots (the state variables are much more coupled in that case). We will also investigate different fitness measures used in genetic programming. We have observed that the RMSE of the one-step-ahead prediction on the training set does not lead to robust process models. We will focus on simulation-based fitness functions.

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).

Authors also thank Hai Zhu for carrying out the experiments with the drone.

References

Abonyi, J., Babuška, R., Verbruggen, H., & Szeifert, F. (2000). Incorporating prior knowledge in fuzzy model identification. *International Journal of Systems Science*, *31*, 657–667.

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

Arnaldo, I., O'Reilly, U.-M., & Veeramachaneni, K. (2015). Building predictive models via feature synthesis. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* GECCO '15 (p. 983–990). New York, NY, USA: Association for Computing Machinery. doi:10.1145/2739480.2754693.

Ashok, D., Scott, J., Wetzel, S., Panju, M., & Ganesh, V. (2020). Logic guided genetic algorithms. arXiv:2010.11328.

Błądek, I., & Krawiec, K. (2019). Solving symbolic regression problems with formal constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference* GECCO '19 (pp. 977–984). New York, NY, USA: ACM. doi:10.1145/3321707.3321743.

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

de Bruin, T., Kober, J., Tuyls, K., & Babuška, R. (2016). Improved deep reinforcement learning for robotics through distribution-based experience retention. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Cheng, T., & Zhong, J. (2020). An efficient memetic genetic programming framework for symbolic regression. *Memetic Computing*, *12*, 1–17. doi:10.1007/s12293-020-00311-8.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*, 182–197. doi:10.1109/4235.996017.

Deisenroth, M., & Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)* (p. 465–472).

Derner, E., Kubalík, J., Ancona, N., & Babuska, R. (2020). Constructing parsimonious analytic models for dynamic systems via symbolic regression. *Appl. Soft Comput.*, *94*, 106432. doi:10.1016/j.asoc.2020.106432.

Derner, E., Kubalík, J., & Babuska, R. (2018a). Data-driven construction of symbolic process models for reinforcement learning. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)* (pp. 5105–5112). Brisbane, Australia.

Derner, E., Kubalík, J., & Babuska, R. (2018b). Reinforcement learning with symbolic input-output models. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 3004–3009).

Goodwin, G., & Payne, R. (1977). *Dynamic System Identification: Experiment Design and Data Analysis*. New York, USA: Academic Press.

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

Johansen, T. (1996). Identification of non-linear systems using empirical data and a priori knowledge – an optimisation approach. *Automatica*, *32*, 337–356.

Karny, M., Haluskova, A., & Nedoma, P. (1995). Recursive approximation by ARX model: A tool for grey-box modelling. *Int. J. Adaptive Control and Signal Processing*, *9*, 525–546.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press.

Krawiec, K., Błądek, I., & Swan, J. (2017). Counterexample-driven genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference* GECCO '17 (pp. 953–960). New York, NY, USA: ACM. doi:10.1145/3071178.3071224.

Kubalík, J., Derner, E., & Babuška, R. (2020). Symbolic regression driven by training data and prior knowledge. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* GECCO '20 (p. 958–966). New York, NY, USA: Association for Computing Machinery. doi:10.1145/3377930.3390152.

Kubalík, J., Derner, E., & Babuška, R. (2017). Enhanced symbolic regression through local variable transformations. In *9th International Joint Conference on Computational Intelligence (IJCCI 2017)* (pp. 91–100).

Leonaritis, I., & Billings, S. (1985). Input-output parametric models for non-linear systems. *International Journal of Control*, *41*, 303–344.

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

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

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., & Liang, E. (2006). Autonomous

inverted helicopter flight via reinforcement learning. In M. H. Ang, & O. Khatib (Eds.), *Experimental Robotics IX: The 9th International Symposium on Experimental Robotics* (pp. 363–372). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/11552246_35.

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

Searson, D. P. (2014). Gptips 2: An open-source software platform for symbolic data mining. In *Handbook of Genetic Programming Applications*.

Staelens, N., Deschrijver, D., Vladislavleva, E., Vermeulen, B., Dhaene, T., & Demeester, P. (2013). Constructing a no-reference h.264/avc bitstream-based video quality metric using genetic programming-based symbolic regression. *IEEE Trans. Cir. and Sys. for Video Technol.*, *23*, 1322–1333. doi:10.1109/TCSVT.2013.2243052.

Timmons, W., Chizeck, H., & Katona, P. (1997). Parameter-constrained adaptive control. *Ind. Eng. Chem. Res.*, *36*, 4894–4905.

Tulleken, H. (1990). Generalized binary noise test-signal concept for improved identification experiment design. *Automatica*, *26*, 37–49.

Tulleken, H. (1993). Gray-box modelling and identification using physical knowledge and Bayesian techniques. *Automatica*, *29*, 285–308.

Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., & Tegmark, M. (2020). Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. arXiv:2006.10782.

Udrescu, S.-M., & Tegmark, M. (2020). Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, *6*, eaay2631. doi:10.1126/sciadv.aay2631.

Vladislavleva, E., Friedrich, T., Neumann, F., & Wagner, M. (2013). Predicting the energy output of wind farms based on weather data: Important variables and their correlation. *Renewable Energy*, *50*, 236–243.

Zhu, H., & Alonso-Mora, J. (2019). Chance-Constrained Collision Avoidance for MAVs in Dynamic Environments. *IEEE Robotics and Automation Letters*, *4*, 776–783.