Open Source Robotics-Research Article

INTERNATIONAL JOURNAL OF
ADVANCED ROBOTIC SYSTEMS

International Journal of Advanced Robotic Systems
May-June 2020: 1–9
© The Author(s) 2020
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/1729881420913741
journals.sagepub.com/home/arx
SAGE

BOLeRo: Behavior optimization and learning for robots

Alexander Fabisch[1,2], Malte Langosz[1] and Frank Kirchner[1,2]

Abstract

Reinforcement learning and behavior optimization are becoming more and more popular in the field of robotics because algorithms are mature enough to tackle real problems in this domain. Robust implementations of state-of-the-art algorithms are often not publicly available though, and experiments are hardly reproducible because open-source implementations are often not available or are still in a stage of research code. Consequently, often it is infeasible to deploy these algorithms on robotic systems. BOLeRo closes this gap for policy search and evolutionary algorithms by delivering open-source implementations of behavior learning algorithms for robots. It is easy to integrate in robotic middlewares and it can be used to compare methods and develop prototypes in simulation.

Keywords

Reinforcement learning, evolutionary algorithms, robotics, black-box optimization

Date received: 23 February 2019; accepted: 1 February 2020

Topic: Robotics Software Design and Engineering
Associate Editor: Francisco Rico
Topic Editor: David Portugal

Background and related work

Designing behaviors for robots is tedious work. Even though planning is possible, it usually requires a detailed description of the solution and an accurate model of the world. Machine learning and optimization can mitigate this problem by learning in reality or exploiting simulations. Behavior optimization and learning for robots (BOLeRo) is designed to support the process of research and development of behavior learning and optimization for robots. Its main goals are (1) to be a benchmark framework for behavior search algorithms (learning and optimization), (2) to provide a set of reference implementations for benchmark problems, behavior representations, and well-established and reliable behavior search (learning or optimization) algorithms, (3) while it should be easy to integrate behaviors or behavior search algorithms in real robotic systems.

Some of BOLeRo's goals overlap with the goals of other open-source software packages: some focus on behavior learning algorithms and others on benchmark problems. A major problem is that many state-of-the-art experiments are hardly reproducible, particularly if robotic simulations are involved. Small changes in the simulation setup can significantly influence the problem complexity and quite often not all simulation details are published. For example, changing contact parameters for a walking system might influence the preferred walking pattern. In BOLeRo, it is easy to create environments with the MARS simulation software (https://github.com/rock-simulation/mars).

Hence, it simplifies the publication of reference implementations of robotic learning environments. A more recent development that covers the same aspects as BOLeRo is OpenAI Gym.[1] It provides very simple environments and complex robotic environments for

[1] Robotics Innovation Center, DFKI GmbH, Bremen, Germany
[2] Robotics Research Group, University of Bremen, Bremen, Germany

Corresponding author:
Alexander Fabisch, DFKI GmbH Robotics Innovation Center, Robert-Hooke-Straße 1, D-28359 Bremen, Germany.
Email: alexander.fabisch@dfki.de

reinforcement learning (RL) to improve reproducibility of experiments and enable rigorous comparison of methods. There are other works that build upon OpenAI Gym, for example,[2] extend it with various simulated robotic scenarios.

There are several behavior learning libraries that focus on classical RL,[3,4] which often does not work out of the box for robotic problems. Although there are certainly works that apply standard RL[5] on real robotic systems, it has been struggling with challenging problems posed by the real world for a long time: continuous state and action spaces, complex dynamics, noisy sensors, sparse rewards, and above all the demand for sample efficiency. A more recent branch of RL is deep RL.[6] Deep RL is very interesting for robotics because it handles complex sensors like cameras easily. There are some open-source implementations available, for example, OpenAI Baselines,[7] Dopamine,[8] TF-Agents,[9] or Garage.[10] It is, however, not easily applicable to learning on real robotic systems yet. Currently, there is no established evaluation procedure in the community[11] and it is hard to tell whether a method is robust enough to be worth the effort of implementing it for roboticists. However, its main problem is sample efficiency. For example, hindsight experience replay (HER), a recently published method,[12] needs about 40,000 episodes to learn how to push a puck on a table to multiple goals with approximately 95% success rate. A breakthrough for deep RL in robotics has been achieved by Levine et al.,[13] but this achievement relied on additional pretraining procedures, a fully observable state space during the training phase and additional methods from optimal control, which makes the whole process of generating new behaviors very complicated and difficult to engineer. Publicly available implementations are often either research code or rather designed to learn behaviors of agents in simple virtual worlds, for example, video games. We see a niche here for a library that provides implementations of established behavior search algorithms that work well for robots and that is designed to support the behavior development workflow for a robot and we propose BOLeRo as a solution. Reliable and successful solutions come from the fields of RL,[14–23] optimal control,[24] and black-box optimization.[25–30] These approaches often rely on special types of policies for robotic problems, for example, dynamical movement primitives.[31–35] For details on RL in robotics please refer to Kober et al.[36] BOLeRo provides implementations for several of these algorithms.

Design and features

One of the main design decisions of BOLeRo is to keep interfaces rather simple, which allows to quickly integrate external methods into it. Decoupling through interfaces also provides flexibility in combining methods and applications. It also enables us to easily combine Python and C++. There are Python bindings for C++ components and, vice versa, it is possible to run Python components from C++ via the BOLeRo interfaces.

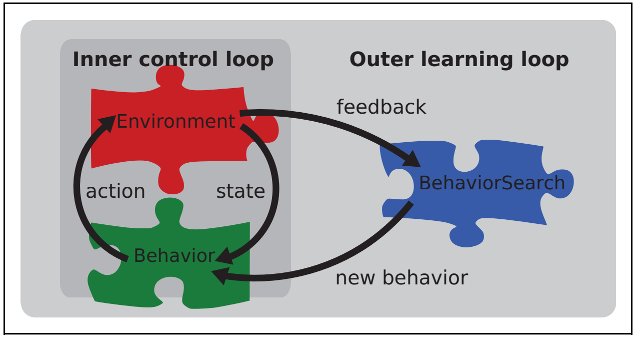

Figure 1. Main cycles during episodic learning process.

BOLeRo can be used as a library that provides behaviors, behavior search algorithms (learning and optimization), or simulation environments that represent behavior learning problems. It can also be used as a framework for comparison of behavior learning algorithms.

Figure 1 illustrates how BOLeRo can be used as a benchmark framework. An environment defines the learning problem. A behavior is evaluated in the environment in a control loop: in each step, the behavior observes the current state of the environment, computes an action, and the action is executed in the environment. After the control loop completes, which is indicated by the environment, feedback (e.g. reward or fitness) is given from the environment to the behavior search algorithm. The behavior search uses feedback to generate new behaviors in a learning loop. This episodic learning process is implemented by a *controller* in BOLeRo. This can be done with a C++ controller that loads benchmark configuration files or we can use Python scripts to define the benchmark (see Algorithm 1, for an example). In this case, the control flow is defined by BOLeRo and it is used as a framework, but it is not the only way to make use of the individual components.

It is not required to change the design of the existing software to use BOLeRo. It can be used as a library that provides behaviors, behavior search algorithms, or environments. For integration in robotic middlewares, we suggest using BOLeRo as a library and let the developer define the control flow based on the robotic framework. The transition from learning in simulation to a robotic system is a major challenge. The choice of languages, methods, dependencies, and interfaces often complicates this matter. Most robotic frameworks use C++ as their core language (e.g. ROS[37]). Python on the other hand is one of the most important languages for machine learning with a great ecosystem for scientific programming. We want to make BOLeRo compatible with both worlds. It is possible to use it in both C++ and Python.

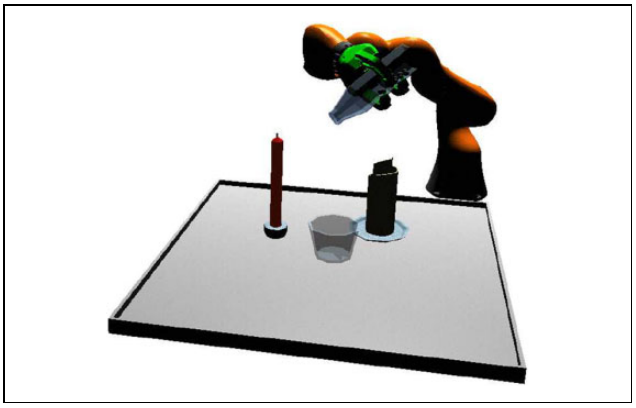

Figure 2. Example of a MARS environment in BOLeRo. The Kuka arm has to pour a couple of marbles.

We describe the BOLeRo interfaces and available implementations in the next paragraphs.

Environments

Environments in BOLeRo define a behavior learning problem. Having a robotic simulation to test the feasibility of approaches, new methods, or to be able to train complete behaviors that can be used on real robots is necessary in a behavior learning library for robots. There are various simulations that can be used to define robotic environments, for example, MARS, Bullet (http://bulletphysics.org), Gazebo,[38] v-rep (http://www.coppeliarobotics.com), or MuJoCo.[39] Communication with simulations is often not easy and not all simulations are suitable for behavior learning. We prepared a base class for environments that use the MARS simulation software in BOLeRo. With graphical tools like Phobos (https://github.com/rock-simulation/phobos), new robotic environments can be defined. An example is shown in Figure 2. Implementing new environments is also possible with the physics engine Bullet, for example, via the convenient pybullet API.

Since the key idea of OpenAI Gym[1] to provide reproducible learning environments correlates to the concept of BOLeRo, a coupling of OpenAI Gym is provided by BOLeRo. There is a wrapper available in BOLeRo for OpenAI Gym environments.

Behaviors

BOLeRo provides implementations for recent movement primitives like Cartesian space dynamical movement primitives[35] and probabilistic movement primitives.[40]

Behavior search algorithms

Behavior search algorithms often combine behavior representations (e.g. neural networks in deep RL and neuroevolution or movement primitives in policy search) with behavior optimization (e.g. policy gradient algorithms in deep RL or black-box optimization in episodic policy search).

Implementations of policy search algorithms are available in BOLeRo, for example, episodic[16] and contextual relative entropy policy search (REPS[20]), covariance matrix adaptation evolution strategies (CMA-ES),[26] ACM-ES (CMA-ES with a ranking support vector machine as surrogate model)[41] and contextual CMA-ES,[30] and natural evolution strategies (NES).[42] A wrapper around scikit-optimize,[43] a library for model-based optimization, is integrated. An additional package (https://github.com/rock-learning/bayesian_optimization) for Bayesian optimization[27] and Bayesian optimization for contextual policy search (BO-CPS)[29] is available. Step-based RL and deep RL algorithms are planned as next features.

2.1 Organization and source code

BOLeRo currently has two maintainers. It is an open-source software and we would like to encourage everyone to contribute to the project by reporting issues or submitting pull requests to the public git repository. For example, new environments, behavior search algorithms, behavior representations, improved documentation, and tests would be very helpful contributions. BOLeRo is a modular system, and it is easily possible to write extensions without having write access to its core repository. It is only required to implement BOLeRo's interfaces to use it.

Source code of BOLeRo is released at https://github.com/rock-learning/bolero under 3-clause BSD license. At https://rock-learning.github.io/bolero, a detailed documentation is available. We support Ubuntu 18.04 LTS (although it also works with other versions), Mac, and to some extent Windows.

Examples and applications

In this section, we will show how BOLeRo can be used in practice. For example, we present an experiment in a simple Python script and a more complex study with a simulated walking robot.

Simple example

An episodic learning process (see Figure 1) can be organized by a controller in a simulation. A simple source code example is shown in Listing 1. The behavior in this case is a dynamical movement primitive (`DMPBehavior`), the behavior search is a black-box search (`BlackBoxSearch`), which uses the black-box optimization algorithm CMA-ES (`CMAESOptimizer`) to modify the weights of the `DMPBehavior`. The environment is a simple trajectory planning problem in 2D (`OptimumTrajectory`): Three

Listing 1. Short version of an example in BOLeRo's documentation: https://rock-learning.github.io/bolero/auto_examples/behavior_search/plot_obstacle_avoidance.html

```
import numpy as np
from bolero.environment import OptimumTrajectory
from bolero.behavior_search import BlackBoxSearch
from bolero.optimizer import CMAESOptimizer
from bolero.representation import DMPBehavior
from bolero.controller import Controller

x0 = np.zeros(2)  # initial state
g = np.ones(2)    # goal state
execution_time = 1.0
dt = 0.01

beh = DMPBehavior(
    execution_time, dt, n_features=10)
env = OptimumTrajectory(
    x0, g, execution_time, dt,
    obstacles=[np.array([0.5, 0.5]),
               np.array([0.6, 0.8]),
               np.array([0.8, 0.6])],
    penalty_goal_dist=1.0,
    penalty_obstacle=1000.0,
    penalty_acc=1.0)
opt = CMAESOptimizer(
    variance=100.0 ** 2, random_state=0)
bs = BlackBoxSearch(beh, opt)
controller = Controller(
    environment=env, behavior_search=bs,
    n_episodes=500, record_inputs=True)

rewards = controller.learn(
    ["x0", "g"], [x0, g])

controller.episode_with(
    bs.get_best_behavior(),
    ["x0", "g"], [x0, g])
# get last trajectory
X = np.asarray(controller.inputs[-1])
```

circular obstacles have to be avoided on the path from start to goal while minimizing acceleration. The corresponding learning curve is displayed in Figure 3 on the left side. The environment, the final trajectory, and several intermediate solutions are shown on the right side.

Learning to walk

Another example application is the development of locomotion patterns for a legged robot. The purpose of this section is to describe a benchmark for legged locomotion. The benchmark can be used to compare different approaches concerning the controller of a legged system, the learning algorithm, and the evaluation function used. This is of interest since legged locomotion is a complex problem and many publications in that field are hard to compare due to different simulation setups used. For this

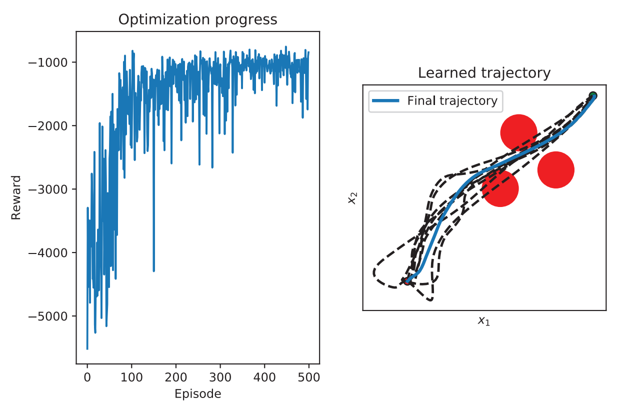

Figure 3. Left: Learning curve for simple 2D planning problem. Right: 2D environment. Large red circles represent obstacles, a blue line indicates the final path, and dashed lines indicate several intermediate solutions.

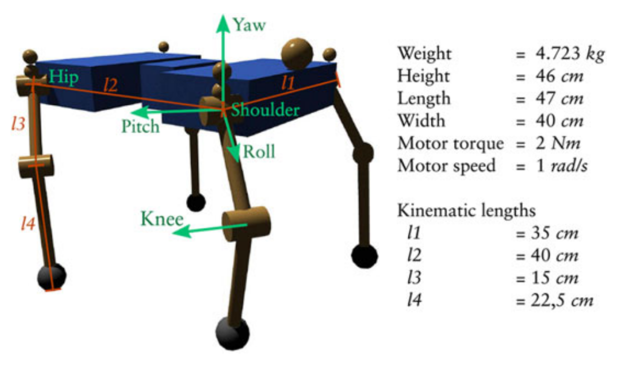

Figure 4. Simulation robot EASY4 used to evaluate different joint pattern representations and a reactive control concept.

benchmark, a four-legged walking robot is designed which is shown in Figure 4. The physical properties of the system are chosen such that it could represent a real robotic system. The motors are controlled by a position controller defining the speed of the motors, while the required motor torques to generate the speed are produced by the simulation MARS. For the benchmark, a general locomotion environment is implemented (https://github.com/rock-learning/locomotion_environment). The configuration of the environment allows to specify evaluation criteria such as motor torques, structure load, velocity, or measured feet slippage. Additionally, a model for the robot to load and a scene defining the simulated environment can be specified. For a first experiment on this benchmark, the genetic algorithms SABRE[44] is used to optimize a controller defined in the graph-based programming language BAGEL.[45] In BAGEL, a control algorithm is implemented through hierarchical dataflow graphs where the nodes represent "atomic" computations or a lower level BAGEL graph. Due to this representation of algorithms, a learning method can adapt the algorithm structure together with its parameters. A central

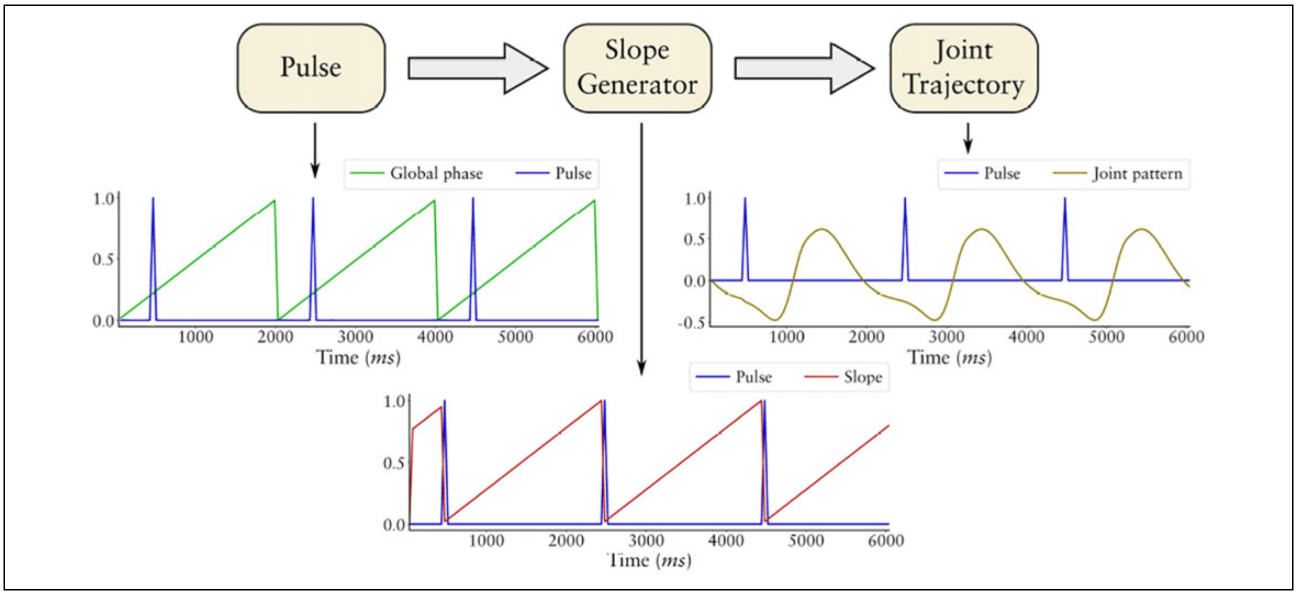

Figure 5. Basic concept for the CPG-based joint trajectory generation. The global `Phase Generator` module is not depicted though its output (Global phase) is plotted in the first graph. Based on the global phase the `Pulse` module generates a pulse pattern that is transferred to a "local" phase in the `Slope Generator` module. The local phase is aligned by the pulse pattern and it is transferred to a joint pattern in the `Joint Trajectory` module.

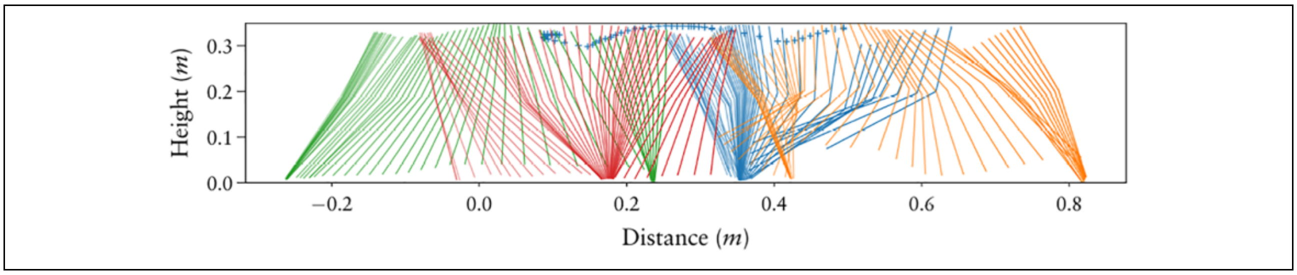

Figure 6. Example behavior generated with the Gaussian pattern approach. The graph shows the corpus center position (blue marker) and the legs from the side view performing one control cycle of 2 s. The legs are separated by the colors (front left = blue, front right = orange, rear left = green, rear right = red).

pattern generator (CPG)[46] is designed with BAGEL. The general concept of the controller is shown in Figure 5. The CPG produces a global phase ϕ with $0 \leq \phi \leq 1$, multiple phase shifts ϕ_i and local phases ϕ_i. It is used as a base pattern generator, whereby the projection $j(\phi)$ from the local phases to the angular joint patterns is generated by the genetic algorithm SABRE. 6. As representation for joint trajectories a Gaussian pattern approach is used. Hence, the genetic algorithm can generate multiple parameterized Gaussian core patterns that are overlaid by a weighted sum to produce a joint trajectory. The Gaussian pattern approach is similar to a radial basis function network, but optimized to generate a periodic function and to support asymmetric shapes.

Each behavior is evaluated in the simulation for a test period of 30 s. The first second is ignored for calculating the fitness value, removing the first acceleration phase of the robot from the stability evaluation. The evaluation is aborted if other parts of the robot than the feet have collisions ($ngc > 0$). In that case, the fitness is defined by a constant value (100,000,000) subtracted by the simulation time in milliseconds until the collision is detected. Otherwise, a feet slippage percentage f_s is combined with two fitness terms f_{t_1} and f_{t_2} as defined by f_{ind}

$$f_{\text{ind}} = \begin{cases} 100,000,000 - t, & \text{if } \; ngc > 0 \\ f_s + 0.5(f_{t_1} + f_{t_2}), & \text{else} \end{cases} \tag{1}$$

The first term f_{t_1} includes the average motor torques ϑ and joint loads ρ (torques applied to the joints not on the motor axis) divided by the final forward position of the robot p_x. The robot position is limited to a value of 3 m. Thus, after the optimization creates behaviors that reach the 3 m mark, it continues to optimize stability instead of producing faster individuals. Furthermore, to prevent to large fitness values or a division by zero, the robot position is cut to positive values starting from 0.01 m. To include the starting position at 0 m an offset of 10 cm is added previously to the robot position.

The second term f_{t_2} represents the *stability fitness* and includes the standard deviation of the motor torques σ_ϑ, the joint loads σ_ρ, the roll σ_α and pitch σ_β angle of the robot, the robot's velocity σ_{v_x}, and the height σ_{p_z} of the torso. The first term represents the general goal to move the robot forward with minimal effort and minimal stress on the mechanics. The second term increases the focus on steady behaviors to reduce the probability for the optimization to converge in local optima with very dynamic, fast, and unstable behaviors. The two fitness terms are defined by

$$f_{t_1} = \begin{cases} \dfrac{\vartheta+\rho}{3.1} & \text{if } p_x \geq 3.0 \\ \dfrac{\vartheta+\rho}{p_x+0.1} & \text{else if } p_x \geq -0.09 \\ \dfrac{\vartheta+\rho}{0.01} & \text{else} \end{cases} \tag{2}$$

and

$$f_{t_2} = \sigma_\vartheta + \sigma_\rho + \sigma_\alpha + \sigma_\beta + \sigma_{v_x} + \sigma_{p_z} \tag{3}$$

A result generated with the proposed setup is depicted in Figure 6. The evolved controller produces a stable walking behavior and manages 4.5 m in 20 s including a short acceleration phase in the first 2.5 s resulting in an average walking speed of 0.25m/s after the acceleration.

The robot model used for this experiment can be found at https://github.com/rock-simulation/easy4_robot. The source of the behavior representation Bagel, the genetic algorithm Sabre, and the generic locomotion environment is in preparation to being published.

Listing 2. A minimal configuration file for the locomotion learning example.

```
BehaviorSearch:
  type: sabre_behaivor_search
  ConfigFile: GaussConfig/Sabre.yml
Environment:
  type: locomotion_environment
  EvaluationCountdown: 1000
  MaxTime: 40000
  FeetNodeNames:
    - name: Sphere.012
    - name: Sphere.013
    - name: Sphere.014
    - name: Sphere.015
  ContactSensorID: 1
  SupportNodeName: root
  RobotScene: easy4_model/smurf/easy4.smurf
  GroundScene: plane.yml
  FootPosZ: 0.07
  EvaluationNodeName: root
  BagelGraphFitness: eval/fitness.yml
Controller:
  GenerateFitnessLog: true
  LogResults: true
  MaxEvaluations: 1000000
```

An example configuration file is shown in Listing 2 whereby the configuration for the genetic algorithm is given in a separate configuration file. Langosz provided a more detailed information of Sabre, Bagel, the locomotion environment, and how they can be used to generate locomotion solutions.[47]

Reproducible research

Reproducing results from publications is a notoriously hard problem in the domain of robotics and machine learning. One of the goals of BOLeRo is to provide well-tested

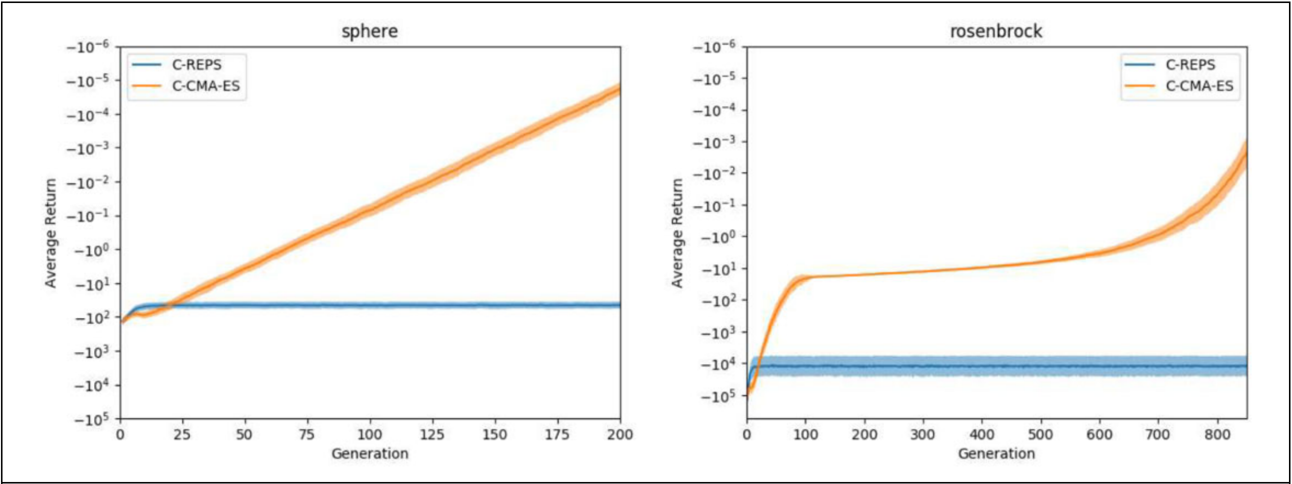

Figure 7. Reproduction of experiments from Figure 1 of.[30] Code and documentation is available at https://github.com/rock-learning/bolero/tree/master/benchmarks/ccmaes.

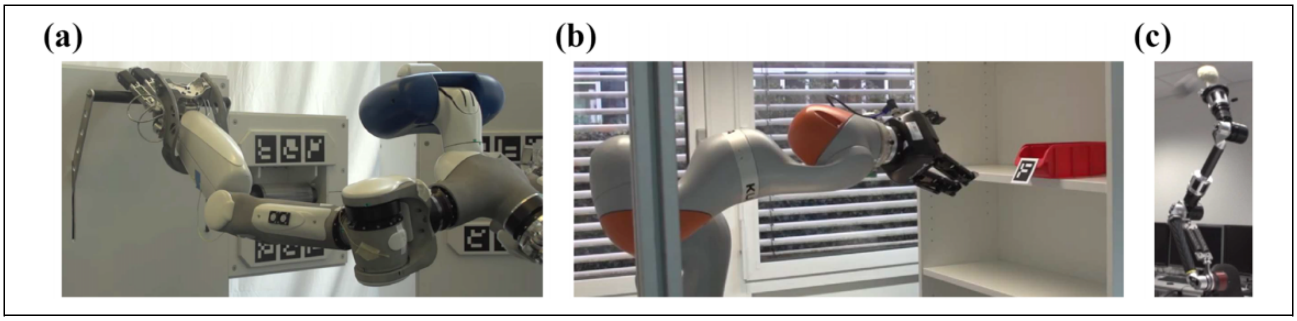

Figure 8. Robotic problems: (a) pulling a lever, (b) grasping a box, and (c) throwing a ball. More information are provided by Gutzeit et al.[48]

implementations of algorithms and test applications. If a simulation tool and model is used to produce scientific results, the experiments can often not be reproduced if the tools and models are not published in the exact configuration they were used. To provide the exact learning configuration, used in publications based on BOLeRo, is one of the main goals of the framework.

Another idea that we pursue with BOLeRo is to produce reference implementations of existing published algorithms. An example of how we imagine this to be done is the reimplementation of contextual CMA-ES.[30] The learning curves for C-REPS and C-CMA-ES in Figure 1(a) and (b) of the original publication could be reproduced with BOLeRo's implementation of the algorithm (see Figure 7).

Other applications

BOLeRo has been used in various research projects (e.g. BesMan (https://robotik.dfki-bremen.de/de/forschung/projekte/besman-1.html), LIMES (https://robotik.dfki-bremen.de/de/forschung/projekte/limes.html)) to learn skills for different robots, and it has been used in various publications.[21,22,29,47–53] In addition to the skills displayed in Figure 8, BOLeRo has also been used to learn walking behaviors for a mantis-like robotic system.

Conclusion and outlook

BOLeRo provides a development and test environment for new learning approaches and scenarios. We split learning problems, learning algorithms, and behavior representation via defined interfaces which makes BOLeRo open for extensions and its parts reusable in other settings. We provide easy-to-use interfaces that support many of the common learning setups used in RL and evolutionary computation. Additionally, using the learning controller, BOLeRo is best suited to perform benchmarks of learning methods.

A benchmark framework that allows to configure a set of learning problems and algorithms is currently in development. The framework will automatically summarize results and evaluations by creating some default statistics and plots. Furthermore, it will provide a cloud computing interface to distribute a set of defined benchmarks or single experiments. The latter allows the distributed evaluation of a whole population of an evolutionary application.

Acknowledgments

The authors would like to thank Jan Hendrik Metzen in particular for initiating the conception of BOLeRo as a collaboration between researchers in RL and evolutionary algorithms and significant contributions to the software. The authors would like to thank Patrick Draheim for helpful feedback on an earlier version of this manuscript and our anonymous reviewers who significantly improved the quality of this article.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported through four grants of the German Federal Ministry of Economics and Technology [BMWi, FKZ 50 RA 1216, FKZ 50 RA 1217, FKZ 50 RA 1218, and FKZ 50 RA 1219], two grants of the German Federal Ministry of Education and Research [FKZ 01IW15001, FKZ 01IW18003], and one grant from the European Union's Horizon 2020 research and innovation program [no. H2020-FOF 2016 723853].

ORCID iD

Alexander Fabisch https://orcid.org/0000-0003-2824-7956

References

1. Brockman G, Cheung V, Pettersson L, et al. Openai gym. 2016; arXiv:1606.01540.
2. Lopez NG, Nuin YLE, Moral EB, et al. gym-gazebo2, a toolkit for reinforcement learning using ros 2 and gazebo. 2019; arXiv:1903.06278.
3. Frezza-Buet H and Geist M. A C++ template-based reinforcement learning library: fitting the code to the mathematics. *J Mach Learn Res* 2013; 14(1): 625–628.
4. Geramifard A, Dann C, Klein RH, et al. RLPy: a value-function-based reinforcement learning framework for education and research. *J Mach Learn Res* 2015; 16: 1573–1578.

5. Sutton RS and Barto AG. *Reinforcement learning: an introduction*. Cambridge: MIT Press, 1998.
6. Arulkumaran K, Deisenroth MP, Brundage M, et al. Deep reinforcement learning: a brief survey. *IEEE Signal Process Mag* 2017; 34(6): 26–38.
7. Dhariwal P, Hesse C, Klimov O, et al. Openai baselines. https://github.com/openai/baselines (accessed 28 May 2020).
8. Castro PS, Moitra S, Gelada C, et al. Dopamine: a research framework for deep reinforcement learning. 2018. URL http://arxiv.org/abs/1812.06110 (accessed 28 May 2020).
9. Sergio G, Anoop K, Oscar R, et al. TF-Agents: A library for reinforcement learning in tensorflow. URL https://github.com/tensorflow/agents. [Online; 2018, accessed 30 November 2018].
10. Duan Y, Chen X, Houthooft R, et al. Benchmarking deep reinforcement learning for continuous control. In: *Proceedings of the 33rd international conference on machine learning*, New York, NY, USA, June 2016, pp. 1329–1338.
11. Colas C, Sigaud O, and Oudeyer P. How many random seeds? Statistical power analysis in deep reinforcement learning experiments. 2018. CoRR abs/1806.08295. URL http://arxiv.org/abs/1806.08295 (accessed 28 May 2020).
12. Andrychowicz M, Wolski F, Ray A, et al. Hindsight experience replay. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, and Garnett R (eds) *Advances in neural information processing systems 30*. New York, USA: Curran Associates, Inc., 2017. pp. 5048–5058.
13. Levine S, Finn C, Darrell T, et al. End-to-end training of deep visuomotor policies. *J Mach Learn Res* 2016; 17(39): 1–40.
14. Peters J and Schaal S.Policy gradient methods for robotics. In: *Proceedings of the IEEE international conference on intelligent robotics systems (IROS 2006)*, Beijing, China, 9–15 October 2006.
15. Peters J and Schaal S. Reinforcement learning by reward-weighted regression for operational space control. In: *Proceedings of ICML*, Corvallis, Oregon, USA, 20–24 June 2007.
16. Peters J, Mülling K, and Altun Y. Relative entropy policy search. In: *AAAI*, Atlanta, Georgia, USA, 11–15 July 2010.
17. Neumann G. Variational inference for policy search in changing situations. In: *Proceedings of the 28th international conference on machine learning*. Bellevue, Washington, USA, 28 June–2 July 2011. ISBN 978-1-4503-0619-5, pp. 817–824.
18. Kober J, Wilhelm A, Oztop E, et al. Reinforcement learning to adjust parametrized motor primitives to new situations. *Auton Robots* 2012; 33(4): 361–379.
19. Daniel C, Neumann G, and Peters J. Hierarchical relative entropy policy search. In: *Proceedings of the international conference on artificial intelligence and statistics*. La Palma, Canary Islands, Spain, 21–23 April 2012.
20. Kupcsik AG, Deisenroth MP, Peters J, et al. Data-efficient generalization of robot skills with contextual policy search. In: *AAAI*, Bellevue, Washington, USA, 14–18 July 2013.
21. Fabisch A and Metzen JH. Active contextual policy search. *Mach Learn Res* 2014; 15: 3371–3399.
22. Fabisch A, Metzen JH, Krell MM, et al. Accounting for task-difficulty in active multi-task robot control learning. *KI—Künstliche Intelligenz* 2015; 29(4): 369–377.
23. Tangkaratt V, van Hoof H, Parisi S, et al. Policy search with high-dimensional context variables. In: Singh SP and Markovitch S (eds) *Proceedings of the 31st AAAI conference on artificial intelligence*. Palo Alto, California, USA: AAAI Press, 2017, pp. 2632–2638.
24. Theodorou E, Buchli J, and Schaal S. A generalized path integral control approach to reinforcement learning. *J Mach Learn Res* 2010; 11: 3137–3181.
25. Heidrich-Meisner V and Igel C.Evolution strategies for direct policy search. In: *Parallel problem solving from nature—PPSN X. Lecture notes in computer science*, Vol. 5199, pp. 428–437, Berlin, Heidelberg: Springer, 2008.
26. Hansen N and Ostermeier A. Completely derandomized self-adaptation in evolution strategies. *Evolut Comput* 2001; 9(2): 159–195.
27. Brochu E, Cora VM, and de Freitas N. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. 2010. eprint arXiv:1012.2599, arXiv.org.
28. Fabisch A, Kassahun Y, Wöhrle H, et al. Learning in compressed space. *Neural Netw* 2013; 42: 83–93.
29. Metzen JH, Fabisch A, and Hansen J. Bayesian optimization for contextual policy search. In: *Proceedings of the second machine learning in planning and control of robot motion workshop (MLPC-2015)*, 2015, IROS.
30. Abdolmaleki A, Price B, Lau N, et al. Contextual covariance matrix adaptation evolutionary strategies. In: *Proceedings of the 26th international joint conference on artificial intelligence, IJCAI-17*. Melbourne, Australia, 19–25 August 2017. pp. 1378–1385. DOI: 10.24963/ijcai.2017/191.
31. Pastor P, Hoffmann H, Asfour T, et al. Learning and generalization of motor skills by learning from demonstration. In: *Robotics and automation (ICRA)*, Kobe, Japan, 12–17 May 2009, pp. 763–768.
32. Mülling K, Kober J, and Peters J. A biomimetic approach to robot table tennis. *Adapt Behav* 2011; 19(5): 359–376.
33. Mülling K, Kober J, Kroemer O, et al. Learning to select and generalize striking movements in robot table tennis. *Int J Robot Res* 2013; 32(3): 263–279.
34. Ijspeert AJ, Nakanishi J, Hoffmann H, et al. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural Comput* 2013; 25(2): 328–373.
35. Ude A, Nemec B, Petrić T, et al. Orientation in Cartesian space dynamic movement primitives. In: *Robotics and automation (ICRA)*, Hong Kong, China, 31 May–7 June 2014, pp. 2997–3004.
36. Kober J, Bagnell JA, and Peters J.Reinforcement learning in robotics: A survey. *Int J Robot Res* 2013; 32: 1238–1274.
37. Quigley M, Conley K, Gerkey B, et al. ROS: an open-source robot operating system. In: *ICRA workshop on open source software, Vol*. 3, 2009.
38. Koenig N and Howard A.Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *IEEE/RSJ*

international conference on intelligent robots and systems. Sendai, Japan, 28 September–2 October, 2004. pp. 2149–2154.

39. Todorov E, Erez T, and Tassa Y.Mujoco: a physics engine for model-based control. In: *2012 IEEE/RSJ international conference on intelligent robots and systems IROS,* Vilamoura, Portugal, 7–12 October 2012, ISBN 978-1-4673-1737-5, 2012, pp. 5026–5033. New York, USA: IEEE.
40. Paraschos A, Daniel C, Peters J, et al. Using probabilistic movement primitives in robotics. *Auton Robots* 2018; 42(3): 529–551.
41. Loshchilov I, Schoenauer M, and Sebag M. Comparison-based optimizers need comparison-based surrogates. In: *PPSN*. 2010, pp. 364–373. Berlin: Springer.
42. Wierstra D, Schaul T, Glasmachers T, et al. Natural evolution strategies. *J Mach Learn Res* 2014; 15: 949–980.
43. Head T, Coder M, Louppe G, et al. scikit-optimize/scikit-optimize: v0.5.2. 2018. DOI: 10.5281/zenodo.1207017. https://zenodo.org/record/1207017 (accessed 28 May 2020).
44. Langosz M, von Szadkowski KA, and Kirchner F. Introducing particle swarm optimization into a genetic algorithm to evolve robot controllers. In: *Proceedings of the companion publication of the 2014 annual conference on genetic and evolutionary computation, GECCO Comp'14*. New York, NY, USA, 14 July 2014. ISBN 978-1-4503-2881-4, pp. 9–10. Vancouver, Canada: ACM. DOI: 10.1145/2598394.2598474. URL http://doi.acm.org/10.1145/2598394.2598474
45. Langosz M, Quack L, Dettmann A, et al. A behavior-based library for locomotion control of kinematically complex robots. In: *Nature-inspired mobile robotics*, 2013. pp. 495–502. DOI: 10.1142/9789814525534_0063. URL https://www.worldscientific.com/doi/abs/10.1142/9789814525534_0063 (accessed 28 May 2020).
46. Ijspeert AJ. Central pattern generators for locomotion control in animals and robots: a review. *Neural Netw* 2008; 21(4): 642–653.
47. Langosz M. *Evolutionary legged robotics*. Doctoral dissertation, University of Bremen, Germany. 2019.
48. Gutzeit L, Fabisch A, Otto M, et al. The BesMan learning platform for automated robot skill learning. *Front Robot AI* 2018; 5: 43.
49. Metzen JH, Fabisch A, Senger L, et al. Towards learning of generic skills for robotic manipulation. *Künstliche Intelligenz* 2013; 28(1): 15–20.
50. Metzen JH. Minimum regret search for single- and multi-task optimization. In: *International conference on international conference on machine learning*. New York, NY, USA, 2016. JMLR.org, pp. 192–200.
51. Dettmann A, Langosz M, von Szadkowski KA, et al. Towards lifelong learning of optimal control for kinematically complex robots. In: *Workshop on modelling, estimation, perception and control of all terrain mobile robots. IEEE international conference on robotics and automation (ICRA-2014)*. Hong Kong, China, 31 May–7 June 2014. IEEE. URL http://wmepc14.irccyn.ec-nantes.fr/material/paper/paper-Dettmann.pdf (accessed 28 May 2020).
52. Fabisch A. A comparison of policy search in joint space and Cartesian space for refinement of skills. In: *Advances in intelligent systems and computing, Vol. 980*. 2019. Berlin: Springer.
53. Fabisch A.Empirical evaluation of contextual policy search with a comparison-based surrogate model and active covariance matrix adaptation. In: *GECCO: Proceedings of the genetic and evolutionary computation conference companion*. Prague, Czech Republic, July 2019. pp. 251–255.