The Journal of Open Source Software

movement_primitives: Imitation Learning of Cartesian Motion with Movement Primitives

Alexander Fabisch [1]

1 Robotics Innovation Center, German Research Center for Artificial Intelligence (DFKI GmbH), Bremen, Germany

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Yasmin Mzayek

Reviewers:

- @ishaanamahajan
- @gautam-sharma1

Submitted: 22 March 2024
Published: unpublished

Summary

Movement primitives are a common representation of movements in robotics (Maeda et al., 2017) for imitation learning, reinforcement learning, and black-box optimization of behaviors. There are many types and variations. The Python library *movement_primitives* focuses on imitation learning, generalization, and adaptation of movement primitives in Cartesian space. It implements dynamical movement primitives, probabilistic movement primitives, as well as Cartesian and dual Cartesian movement primitives with coupling terms to constrain relative movements in bimanual manipulation. They are implemented in Cython to speed up online execution and batch processing in an offline setting. In addition, the library provides tools for data analysis and movement evaluation.

Statement of Need

Movement primitives are a common group of policy representations in robotics. Although movement primitives are limited in their capacity to represent behavior that takes into account complex sensor data during execution in comparison to general function approximators such as neural networks, several instances (e.g., dynamical movement primitives) have proven to be a reliable and effective tool in robot learning. A reliable tool deserves a similarly reliable implementation. However, there are only a few actively maintained, documented, and easy to use implementations. One of these is *movement_primitives*, which we present in this article.

Movement Primitives

Dynamical Movement Primitives (DMPs) are the most prominent example of movement primitives (Ijspeert et al., 2002, 2013). From a high-level perspective (Fabisch & Metzen, 2014), a DMP is a policy

$$x_{t+1} = \pi_{w,v}(x_t, t),$$

where x_t is the state of an agent (position, velocity, and acceleration) at time t, w are the weights (parameters) that define the shape of the movement, and v are meta-parameters. The exact definition of the meta-parameters v depends on the DMP type, but most types allow to set the initial state x_0, the final state g, and the duration of the movement τ. A DMP generates a trajectory in state space so that a controller that translates states x_t, x_{t+1} to control commands is required.

DMPs have been used for imitation learning, in which one demonstration is enough to learn a DMP. DMPs can also be used in a reinforcement learning setting, in which the weights of the DMP or the meta-parameters can be learned. Saveriano et al. (2023) provide a survey of DMPs and how they can be used.

In the *movement_primitives* library, we implement several types that are important for Cartesian movement generation: an extension that includes the final velocity as a meta-parameter (Mülling et al., 2013), DMPs for Cartesian poses in three dimensions with unit quaternions (Ude et al., 2014), and DMPs that define bimanual movements by introducing a coupling term that controls the relative motion of two arms (Gams et al., 2013).

Another type of movement primitives implemented in this library are Probabilistic Movement Primitives (ProMPs) (Paraschos et al., 2013) that capture the distribution of multiple demonstrations. Their probabilistic formulation allows to modify movements by conditioning, for instance, on viapoints.

Implementations of Movement Primitives

The *movement_primitives* library is a reimplementation and extension of the movement primitive features of BOLeRo (Fabisch et al., 2020). BOLeRo is a C++/Python framework for behavior learning and optimization. However, the focus is very broad and more on reinforcement learning and behavior parameter optimization than on imitation learning.

Another similar library is dmpbbo (Stulp & Raiola, 2019), which has a general DMP implementation and additional components to optimize the parameters of DMPs in reinforcement learning settings. The library is designed to train DMPs in Python and execute them in C++. Both implementations are not well-suited for imitation learning because additional tooling for data analysis and deployment is required. Switching between C++ and Python is also not convenient for various reasons: building and installing these packages is complicated, continuous integration is hard to set up, code maintenance is complicated, and it does not integrate easily with the Python scientific ecosystem.

There are more implementations listed by Saveriano et al. (2023) (available at https://gitlab.com/dmp-codes-collection/third-party-dmp). A lot of these are examplary Matlab scripts and not maintained anymore, or only implementations of specific papers. Other libraries do not support Cartesian movement primitives, which are only available in BOLeRo and *movement_primitives*. The latter also supports bimanual movements through dual Cartesian DMPs.

Design and Features

The main contributions of *movement_primitives* are (1) a fast Python-only library for movement primitives, and (2) robust implementations of several types of movement primitives (see Table 1). Our focus is on Cartesian movement primitives that are used to control one or two robotic arms and offer exemplary implementations of coupling terms for Cartesian (bimanual) DMPs. These can be used for obstacle avoidance and to constrain dual arm motions to relative positions and/or orientations.

Table 1: Overview of implemented movement primitives.

Class	Description	Publication
DMP	Standard DMP	Ijspeert et al. (2013)
DMP	Smooth spatial scaling	Pastor et al. (2009)
DMPWithFinalVelocity	Allows final velocity	Mülling et al. (2013)
CartesianDMP	DMP of Cartesian poses	Ude et al. (2014)
DualCartesianDMP	DMP of two Cartesian poses	Gams et al. (2013)
ProMP	Standard ProMP	Paraschos et al. (2013)

Furthermore, *movement_primitives* supports the whole imitation learning pipeline, including data analysis through plotting and visualization (based on pytransform3d (Fabisch, 2019) and Open3D (Zhou et al., 2018)), data preprocessing for imitation learning, good integration with the scientific ecosystem in Python, simulation of learned movement primitives (in PyBullet (Coumans & Bai, 2016--2021)), export to permanent data formats (pickle, JSON, YAML), and analysis of kinematic feasibility. Although it has several dependencies and requires compilation because of its Cython (Dalcin et al., 2011) components, it is possible to simply install it with pip from PyPI.

Example: Rotating a Compact Solar Panel with a Humanoid

Figure 1 and Figure 2 show a humanoid robot rotating an object with two hands. The movement is generated by a dual Cartesian DMP trained on a demonstrated rotation movement. The width of the object is known. Hence, it can easily be adapted for similar objects with a different size through a coupling term defined by (Gams et al., 2013).

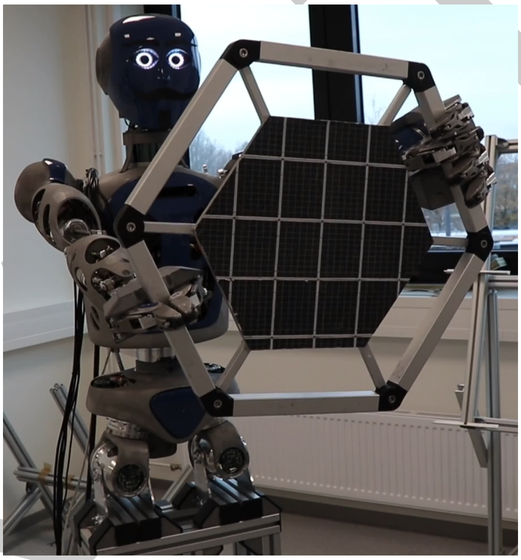

Figure 1: RH5 Manus (Boukheddimi et al., 2022) rotating a compact solar panel.

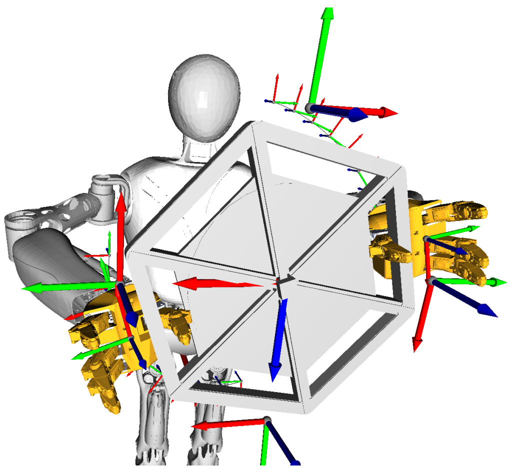

Figure 2: Visualization of similar rotation trajectory with another humanoid robot.

A similar task has been solved by Mronga & Kirchner (2021) with two Kuka iiwa arms. They record a dataset for different panel sizes via kinesthetic teaching and use Gaussian mixture regression to represent the distribution of solutions and condition it on the object width to generalize. This is easier with ProMPs: for each demonstration, we compute ProMP weights, concatenate them with the task parameters over which we want to generalize, and learn a Gaussian mixture model, which we can condition on task parameters to generate ProMPs that define trajectory distributions to solve these tasks (Figure 3 and Figure 4).

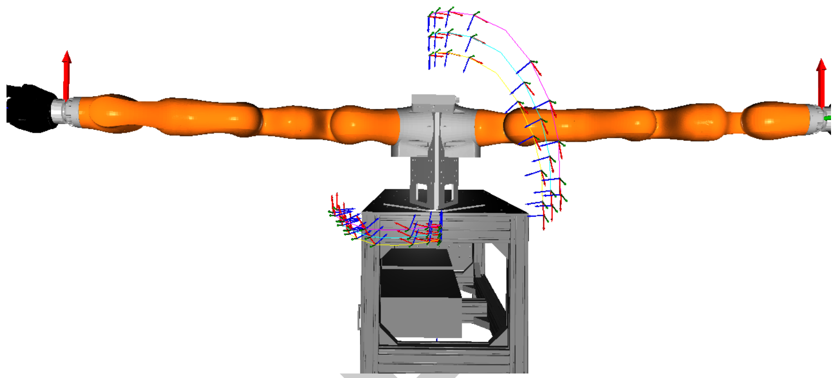

Figure 3: Mean trajectories for conditional ProMPs and panel widths 30/40/50 cm.

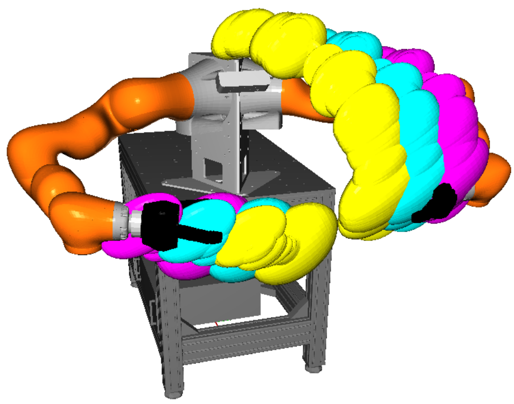

Figure 4: At each step, the position distribution defined by the conditioned ProMP is indicated by an equiprobable ellipsoid. The arms are at the mean start position for width 50 cm.

Benchmark of DMP Implementations

Since execution speed of DMPs is relevant in robotics, we compare several DMP implementations from dmpbbo and *movement_primitives*. For this purpose, we create a minimum jerk trajectory of N dimensions that moves from $0 \in \mathbb{R}^N$ to $1 \in \mathbb{R}^N$ in one second, train a DMP on it, and execute the DMP step by step. We use M weights per dimension, and step through the DMP with $\Delta t = 0.001s$. The concept of dmpbbo is to train in Python and run DMPs in C++. We still analyze the Python version and the C++ version of dmpbbo as well as *movement_primitives* with various implementations of the integration (Euler integration with $h = 0.1 \cdot \Delta t$ and RK4 integration, both in Python and Cython). The default integration method of dmpbbo is RK4. Results for varying configurations of N and M are summarized in Figure 5, Figure 6 and Table 2. While the number of weights per dimension and the number of dimensions have a considerable influence on the runtime of dmpbbo, the influence on the runtime of *movement_primitives* is negligible because NumPy (Harris et al., 2020) vectorization is used. More specifically, computing all steps of a DMP with 1 s duration at 1 kHz ($\Delta t = 0.001s$) with $N = 50$ dimensions and $M = 60$ weights per dimension takes $0.0822 \pm 0.0015s$ with the *movement_primitives* library and RK4 integration in Cython, which means 8.51% of the DMP's runtime is spent on computing steps. This allows online adaptation of the trajectory. dmpbbo's C++ implementation is the best candidate for a low number of dimensions and weights per dimension. In this domain it outperforms all other implementations by a considerable margin. However, it scales linearly with these numbers. Hence, it is considerably slower for $N = 50$ and $M = 60$ than any RK4 implementation of *movement_primitives*. The Python version of dmpbbo is not able to run some configurations in real time. For example, $N = 6, M = 30$ needs $5.9292 \pm 0.0955s$ to compute.

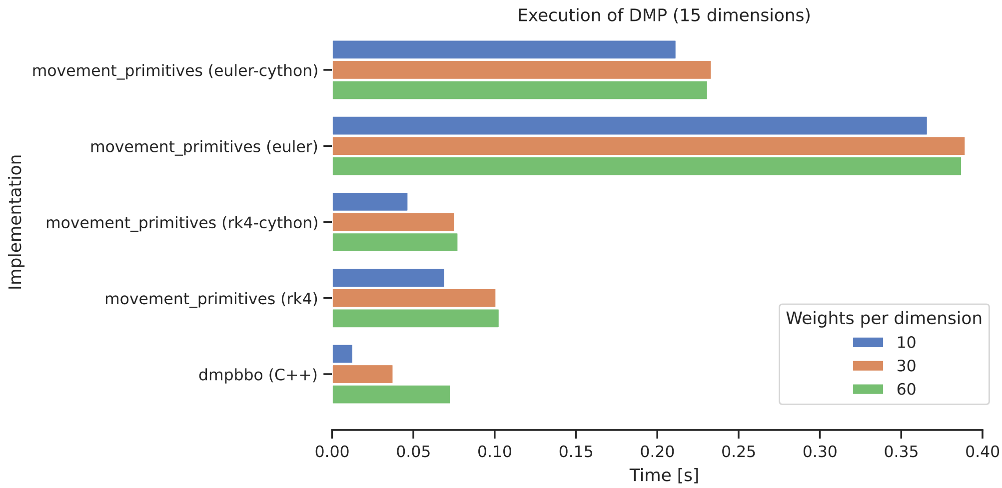

Figure 5: Benchmark of execution speed for various DMP implementations and configurations. Each bar shows an average over 100 stepwise executions of a DMP. Varying number of weights per dimension M, number of dimensions $N = 6$.

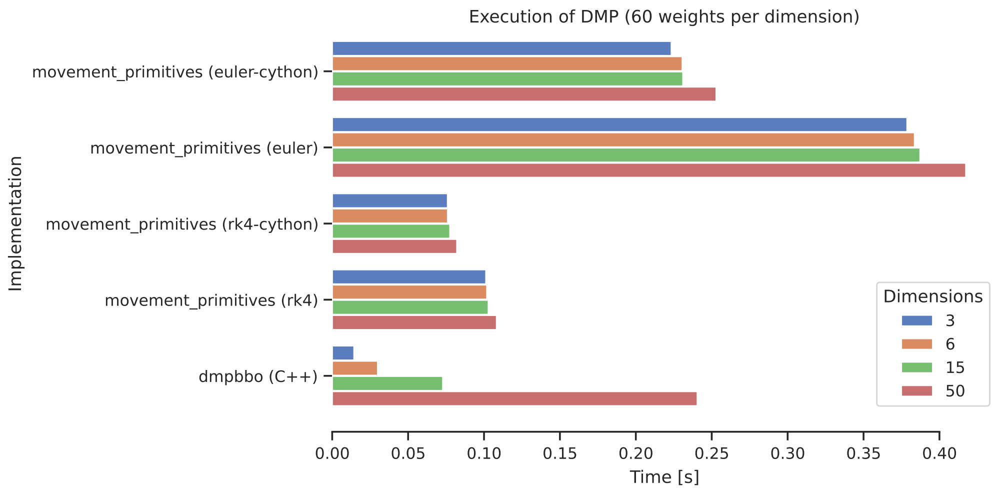

Figure 6: Benchmark of execution speed for various DMP implementations and configurations. Each bar shows an average over 100 stepwise executions of a DMP. Varying number of dimensions N, number of weights per dimension $M = 30$.

Table 2: Benchmark results for DMP execution. Best performance per setup in **bold**.

Library	Implementation	N	M	Time $\mu \pm \sigma$ [s]
dmpbbo	C++	3	10	**0.0027** $\pm$ 0.0001
			30	**0.0077** $\pm$ 0.0001
			60	**0.0144** $\pm$ 0.0004
		6	10	**0.0049** $\pm$ 0.0001
			30	**0.0146** $\pm$ 0.0002
			60	**0.0300** $\pm$ 0.0052
		15	10	**0.0129** $\pm$ 0.0028
			30	**0.0376** $\pm$ 0.0059
			60	**0.0729** $\pm$ 0.0103
		50	10	**0.0401** $\pm$ 0.0068
			30	0.1236 $\pm$ 0.0174
			60	0.2405 $\pm$ 0.0308
dmpbbo	Python	3	10	0.8137 $\pm$ 0.0164
			30	1.6986 $\pm$ 0.0319
			60	3.0244 $\pm$ 0.0454
		6	10	1.3946 $\pm$ 0.0228
			30	3.1676 $\pm$ 0.0746
			60	5.9292 $\pm$ 0.0955
		15	10	3.2079 $\pm$ 0.0593
			30	7.4972 $\pm$ 0.1366
			60	14.2590 $\pm$ 0.2811
		50	10	9.7134 $\pm$ 0.0448
			30	24.6018 $\pm$ 2.0579
			60	47.4420 $\pm$ 2.0075
movement_primitives	euler-cython	3	10	0.1946 $\pm$ 0.0019
			30	0.2223 $\pm$ 0.0070
			60	0.2234 $\pm$ 0.0031
		6	10	0.1912 $\pm$ 0.0033
			30	0.2301 $\pm$ 0.0043
			60	0.2306 $\pm$ 0.0060
		15	10	0.2117 $\pm$ 0.0067
			30	0.2334 $\pm$ 0.0041
			60	0.2310 $\pm$ 0.0013
		50	10	0.2260 $\pm$ 0.0009
			30	0.2547 $\pm$ 0.0273
			60	0.2529 $\pm$ 0.0044
movement_primitives	rk4-cython	3	10	0.0447 $\pm$ 0.0006
			30	0.0737 $\pm$ 0.0018
			60	0.0760 $\pm$ 0.0003
		6	10	0.0471 $\pm$ 0.0036
			30	0.0733 $\pm$ 0.0003
			60	0.0761 $\pm$ 0.0003
		15	10	0.0468 $\pm$ 0.0022
			30	0.0754 $\pm$ 0.0005
			60	0.0776 $\pm$ 0.0002
		50	10	0.0752 $\pm$ 0.0002
			30	**0.0794** $\pm$ 0.0063
			60	**0.0822** $\pm$ 0.0015

Conclusion

Although movement primitives are a popular tool in robot learning, there is a lack of well maintained implementations in particular for bimanual and Cartesian movements. *movement_primitives* provides a well-tested, robust implementation of various movement primitives with the goal of generating Cartesian robot movements. It integrates well with the existing Python scientific ecosystem.

Acknowledgements

This work was supported by a grant of the German Federal Ministry of Economic Affairs and Energy (BMWi, FKZ 50 RA 1701) and by the European Commission under the Horizon 2020 framework program for Research and Innovation (project acronym: APRIL, project number: 870142).

References

Boukheddimi, M., Kumar, S., Peters, H., Mronga, D., Budhiraja, R., & Kirchner, F. (2022). Introducing RH5 manus: A powerful humanoid upper body design for dynamic movements. *2022 International Conference on Robotics and Automation (ICRA)*, 01–07. https://doi.org/10.1109/ICRA46639.2022.9811843

Coumans, E., & Bai, Y. (2016--2021). *PyBullet, a python module for physics simulation for games, robotics and machine learning*. http://pybullet.org.

Dalcin, L., Bradshaw, R., Smith, K., Citro, C., Behnel, S., & Seljebotn, D. (2011). Cython: The best of both worlds. *Computing in Science & Engineering*, *13*(02), 31–39. https://doi.org/10.1109/MCSE.2010.118

Fabisch, A. (2019). pytransform3d: 3D transformations for python. *Journal of Open Source Software*, *4*(33), 1159. https://doi.org/10.21105/joss.01159

Fabisch, A., Langosz, M., & Kirchner, F. (2020). BOLeRo: Behavior optimization and learning for robots. *International Journal of Advanced Robotic Systems*, *17*(3). https://doi.org/10.1177/1729881420913741

Fabisch, A., & Metzen, J. H. (2014). Active contextual policy search. *Journal of Machine Learning Research*, *15*(97), 3371–3399. http://jmlr.org/papers/v15/fabisch14a.html

Gams, A., Nemec, B., Zlajpah, L., Wächter, M., Ijspeert, A., Asfour, T., & Ude, A. (2013). Modulation of motor primitives using force feedback: Interaction with the environment and bimanual tasks. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5629–5635. https://doi.org/10.1109/IROS.2013.6697172

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., & Schaal, S. (2013). Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, *25*(2), 328–373. https://doi.org/10.1162/NECO_a_00393

Ijspeert, A. J., Nakanishi, J., & Schaal, S. (2002). Movement imitation with nonlinear dynamical systems in humanoid robots. *Proceedings 2002 IEEE International Conference on Robotics and Automation*, *2*, 1398–1403. https://doi.org/10.1109/ROBOT.2002.1014739

Maeda, G. J., Neumann, G., Ewerton, M., Lioutikov, R., Kroemer, O., & Peters, J. (2017). Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks. *Autonomous Robots*, *41*, 593–612. https://doi.org/10.1007/s10514-016-9556-2

Mronga, D., & Kirchner, F. (2021). Learning context-adaptive task constraints for robotic manipulation. *Robotics and Autonomous Systems*, *141*, 103779. https://doi.org/10.1016/j.robot.2021.103779

Mülling, K., Kober, J., Kroemer, O., & Peters, J. (2013). Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, *32*(3), 263–279. https://doi.org/10.1177/0278364912472380

Paraschos, A., Daniel, C., Peters, J. R., & Neumann, G. (2013). Probabilistic movement primitives. In C. J. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 26). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2013/file/e53a0a2978c28872a4505bdb51db06dc-Paper.pdf

Pastor, P., Hoffmann, H., Asfour, T., & Schaal, S. (2009). Learning and generalization of motor skills by learning from demonstration. *2009 IEEE International Conference on Robotics and Automation*, 763–768. https://doi.org/10.1109/ROBOT.2009.5152385

Saveriano, M., Abu-Dakka, F. J., Kramberger, A., & Peternel, L. (2023). Dynamic movement primitives in robotics: A tutorial survey. *The International Journal of Robotics Research*, 02783649231201196. https://doi.org/10.1177/02783649231201196

Stulp, F., & Raiola, G. (2019). DmpBbo: A versatile python/c++ library for function approximation, dynamical movement primitives, and black-box optimization. *Journal of Open Source Software*, *4*(37), 1225. https://doi.org/10.21105/joss.01225

Ude, A., Nemec, B., Petrić, T., & Morimoto, J. (2014). Orientation in cartesian space dynamic movement primitives. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2997–3004. https://doi.org/10.1109/ICRA.2014.6907291

Zhou, Q.-Y., Park, J., & Koltun, V. (2018). Open3D: A modern library for 3D data processing. *arXiv:1801.09847*.