A Reduced-Order Approach to Assist with Reinforcement Learning for Underactuated Robotics

Jérémy Augot[1,2], Aaron J. Snoswell[2] and Surya P. N. Singh[2]

***Abstract*— Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.**

I. INTRODUCTION

As robots become increasingly variable and compliant they become more capable, and complex. Underactuated robots, for example, provide great design freedom, yet their adoption has been limited by the need for manual controller designs.

Deep Reinforcement Learning (Deep Reinforcement Learning) methods offer automated tools for robotic control by identifying a policy that maximizes the expected sum of rewards [1]. This has been extended to continuous control domains via algorithms such as the Deep Deterministic Policy Gradient (DDPG) [2], Proximal Policy Optimization (PPO) [3], Soft Actor-Critic (SAC) [4], and Twin Delayed Deep Deterministic (TD3) [5]. The power of these methods comes, in part, from the high-dimensional, nonlinear function approximation of both the policy and the expected value by neural networks. The dimensionality of these rich representations also presents challenges, particularly for sample collection, stability, and convergence [6]. Reducing the order seems a natural approach to addressing these challenges, but doing so directly requires carefully defining suitable (state-space) features as the aforementioned methods are sensitive to the chosen representation [7].

Interestingly, many robots are underactuated by design, or at times, by circumstance (e.g., due to motor saturation). Such systems achieve their tasks due to the inherent inter-dependence of their actuation states [8]. This implicit structure suggests that an (automatic) reduction of the actuation space may make these systems more compatible with (model-free) Deep Reinforcement Learning methods, which, in turn, would allow for more variable underactuated systems. Towards this, we introduce a highly variable, single actuator robotic locomotion benchmark (the 'Jitterbug' problem) based around a passive compliant toy system with tasks of varying difficulty, particularly around the sparsity of the reward signal (see also Fig. 1).

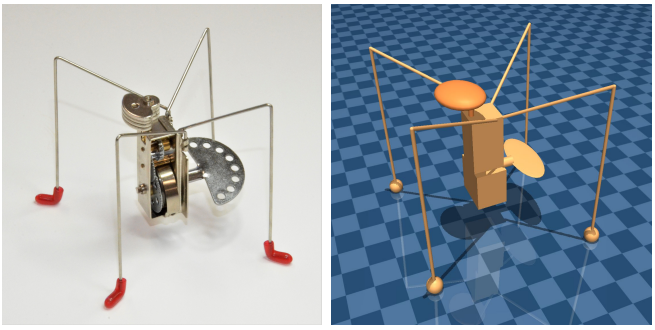

Fig. 1: **The Jitterbug Problem**: The wind-up children's toy 'Katita' (left) was the inspiration for our underactuated 'Jitterbug' continuous control task (right). In the simulated robot the wind-up spring is replaced with a controlled single degree-of-freedom motor. For scale, the blue checks on the simulated floor on the right are 1cm in size.

Reduced order approaches are a common strategy in engineering. An issue herein is that an under-actuated system, such as the Jitterbug problem, is highly variant in state space and thus difficult to explicitly model in an analytic form. Note that 'highly variable' does not imply 'chaotic'; that is, the system is deterministic for over short horizons in the state-space. However, the motion is clearly non-linear due to the contacts and locomotion phases and gaits (indeed, it has been reported that the Katita toy, on which the problem is based, exhibits a 'galloping' locomotion mode [9]). A nonlinear approach for automatically discovering interesting structure in empirical models is unsupervised learning. Autoencoders are a well know neural network approach for this that can automatically discover underlying correlations and their interdependencies [10]. In this way, initial (random) exploration helps characterize the implicit structure of the (underactuated) action space that then focuses subsequent Deep Reinforcement Learning control policy learning.

The two contributions of this paper are: (1) a consideration of autoencoder-based model reduction for Deep Reinforcement Learning in highly variable, underactuated robot domains; and, (2) the introduction and characterization of the Jitterbug Problem as a diverse underactuated robotics task with dynamic, compliant and non-trivial motion in five scenarios of varying gradations of task complexity. We find that reduced order models can have similar (reward) performance,

[1]Ecole CentraleSupélec, Paris, France
[2]The Robotics Design Lab at The University of Queensland, Brisbane, Australia

yet empirically have less training variance and slightly better learning rates. The challenging nature of the Jitterbug task may also help inform future research in automatic controls for flexible robots.

II. RELATED WORK

Para

Model Reduction

Reinforcement Learning is an ....

Reinforcement Learning (RL) methods offer automated tools for controller design of such systems via trail and error [11], but have been been limited by feature representations and forward models [12]. Deep Reinforcement Learning (Deep Reinforcement Learning) algorithms for continuous control address this through the use of deep neural networks for both the policy ('actor') and value function ('critic') [2], [4]. Such systems have been ...

A. Deep Reinforcement Learning

...and have shown promise in a host of challenging applications including visualmotor learning [13] and VR teleoperation [14].

B. Reduced Order Approaches for Underactuated Robotics

Underacted robots are fundamentally correlated in their actuation space [15]. When a model is available control strategies including optimal control [16], direct collocation [17] and trajectory optimization [18] have been applied. Some novel robots, such as compliant legged robots or soft robots, tend to exhibit complex dynamics that limit dynamical system and model reduction approaches as an explicit physical model may not be available [19].

Model reduction is a common strategy in robotics, linear dynamical systems, and machine learning [16], [11]. Variations of this approach include method such as: model linearizion, mixing of open-loop and (typically linear) closed-loop control [20], and tunable models [21]. The issue with these approaches is that explicit system model(s) are typically needed.

Alternatively, statistical methods allow for empirical model. An approach such as Principle Components Analysis (PCA) may help 'reduce' this empirical model by trying to identify linearly correlated values. However, the highly variable of the robot and the piecewise nature of contact (e.g., in legged locomotion or grasping) are better described by non-linear models and non-Gaussian distributions.

In the underactuated domain, these ideas have been used to get an empirical model that then inform subsequent model-based control. For example, Nagabandi, *et al.* [22] use a small legged robot in which a neural network forecasts the robot's state at the next timestep which is then used via Model Predictive Control (MPC). Nishimura, *et al.* use an autoencoder in a similar fashion to inform manipulation with a soft-gripper [23]. In both these cases the control model is analytic and the engineering effort is needed to synchronize the empirical model with the controller, which can limit the types of policies and strategies that the robot can learn and execute.

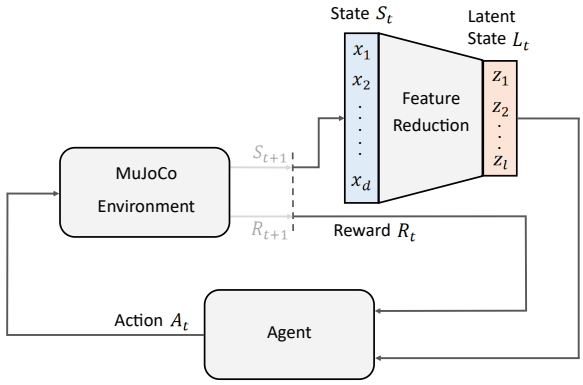

Fig. 2: The architecture of our system.

However, in many robotics tasks the underactuation is dependent on the state

) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components.

An alternative approach might be to approximate probability distribution(s) as compared to arbitrary nonlinear function or transformation

is to model the There are also statistical methods such as

Highly variable motion is not random. Given that actuation space for underactuated robots is structured, an autoencoder would be able to automatically discover some of these correlations [10], [24]. A concern, however, is that the reduced model is not always intuitive, which would complicate the subsequent synchronization with a model based control strategy. Model-free deep reinforcement learning (Deep Reinforcement Learning) continuous control methods, however, are rather compatible with an autoencoder's reduced state description.

The use model reduction complements said Deep Reinforcement Learning methods...

...This has been considered for reinforcement learning ...

This has been particularly effective in cases with very high dimensional inputs, such as from video, for low-dimensional independent constraints, such as obstacle locations [13], [25]....

III. METHOD

We use the intuition that an underacted system has correlations between states and thus there is a lower-dimensional representation that could describe this process.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec

vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

IV. A NOVEL, UNDER-ACTUATED CONTINUOUS CONTROL BENCHMARK

We implemented our Jitterbug benchmark using the DeepMind Control Suite (DMC) framework [26]. DMC is a framework and set of benchmark tasks for continuous control published by Google DeepMind in 2018. DMC benchmarks consist of a *domain* defining a robotic and environment model and *tasks* which are instances of that domain with specific MDP structure

DMC uses the robust Multi-Joint dynamics with Contact (MuJoCo) robotics physics engine for simulation [27]. To aid comparison across tasks, DMC imposes constraints on rewards ($R \in [0, 1]$) and episode length ($H = 1000$). As such, for any DMC task, cumulative episode return ≈ 1000 indicates success.

DMC tasks are compatible subsets of the popular OpenAI Gym framework [28], meaning many popular RL algorithm frameworks can be used with these benchmarks. For our evaluations, we used the `stable_baselines` library of RL algorithms [29], which is derived from the OpenAI `baselines` package [30].

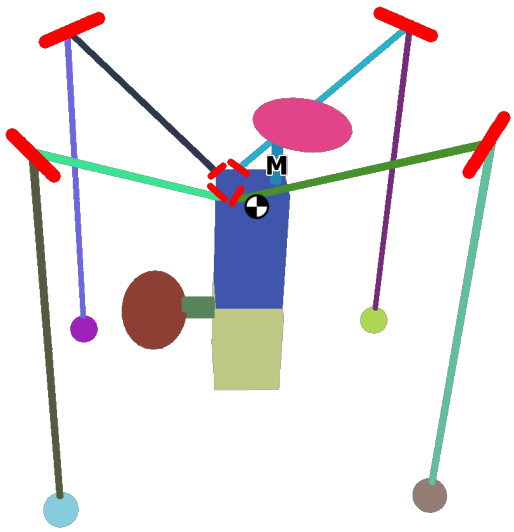

Fig. 3: Schematic representation of the Jitterbug model. Individual rigid bodies are in different colours and we highlight the position of the center of mass ( ), hinge joints ( ) and the single motor ( ).

A. The Jitterbug Domain

Our Jitterbug model was inspired by the children's toy Katita (Figure 1). We aimed to reproduce the physical dynamics of this toy while enabling control by replacing the wind-up spring with a single actuator of equivalent torque. Our Jitterbug model conforms to the dimensions and mass of the Katita, however we replace the wind-up spring with a controlled single degree-of-freedom torque actuator. We retain the (non-functional) wind up crank to more closely model the mass distribution of the physical Katita. Figure 3 shows the schematic composition of our simulated Jitterbug model.

We used high-speed recording and visual tachometry to measure the Katita motor speed and leg vibration modes. By reverse-engineering the Katita gearbox we estimated the torque output of the drive spring and configured the MuJoCo actuator appropriately. We modelled the legs as rigid bodies with shoulder and elbow hinge joints. The hinge stiffness was manually tuned to reproduce the dominant leg vibration mode observed in our high-speed footage. The Jitterbug model density was set using standard values for stainless steel (7700 kg/m^3) for the body and tough plastic (1100 kg/m^3) for the feet.

Due to the importance of contact and stiff dynamics in the Jitterbug's locomotion, we found it necessary to adjust MuJoCo's default settings, selecting an integration timestep of 0.0002s and semi-implicit Euler integration. To simplify the control problem, we use a control timestep of 0.01s. Thus, one step in the MDP results in 50 steps of the physics simulation and corresponds to 10ms of elapsed time.

Accordingly, an episode (1000 MDP steps) corresponds to 10s of elapsed time. As is standard practice, during the time between control steps the most recent commanded action is repeated. With these settings we qualitatively observed a close correspondence between the Katita and the simulated dynamics under constant motor actuation on the Jitterbug.

DMC supports the definition of physically-based camera models to enable learning from raw pixels if desired. We defined several cameras for the Jitterbug domain including an overhead, tracking and ego-centric view.

B. The Jitterbug Task Suite

The Jitterbug dynamics naturally induce very high variance motion under a range of motor velocities. We defined a collection of five tasks of increasing difficulty based on the Jitterbug domain. The tasks were designed with increasingly sparse reward signals to increase the difficulty.

For all tasks we choose $\gamma = 0.99$ and consider a task solved when cumulative episode reward is $\gtrapprox 900$. In all tasks the Jitterbug is reset to a random pose near the origin at the start of an episode. All tasks have a single continuous action controlling the motor $\mathcal{A} = [-1, 1]$ (larger/smaller values are clipped). All tasks also share the same continuous state space $\dim(\mathcal{S}) = 31$ but differ in the size of the observation space.

For each task, we report ($\dim(\mathcal{O})$) and a brief description of the reward structure. Tasks are reported here in approximate order of increasing difficulty.

1) *Move From Origin* (15): The Jitterbug must move away from the origin in any direction. Note that a sufficiently fast constant motor velocity is sufficient to solve this task.
2) *Face In Direction* (16): The Jitterbug must rotate to face in a randomly selected yaw direction.
3) *Move In Direction* (19): The Jitterbug is rewarded for velocity in a randomly selected direction in the X, Y plane.
4) *Move To Position* (18): The Jitterbug must move to a randomly selected position in the X, Y plane.
5) *Move To Pose* (19): The Jitterbug must move to a randomly selected position in the X, Y plane and rotate to face in a randomly selected yaw direction. Note that due to the multiplication of position and yaw reward components, this task has a *very* sparse reward signal!

In addition, for all tasks the Jitterbug must remain upright to achieve reward. Falling does not terminate the episode early as the leg dynamics are sufficiently springy that bouncing into the upright pose again can allow recovery from this condition (albeit at the loss of some reward). Indeed - we observed some learned strategies that appeared to utilize this mode of locomotion!

V. EXPERIMENTS

A. Characterising The Jitterbug Tasks

To verify feasibility, we hand-crafted heuristic policies that can solve each task. To characterise the difficulty of the Jitterbug task suite, we performed preliminary hyper-parameter tuning to select reasonable settings and trained several RL algorithms on the tasks.

TABLE I: Algorithm hyper-parameters for DDPG and PPO. Bold items were changed from the defaults offered by the `stable_baselines` package.

Parameter	Value
Shared	
Optimizer	Adam [31]
Learning Rate (α)	**$1_E{-4}$**
Network Architecture(s)	Fully Connected
Number of Hidden Layers	2
Hidden Layer Sizes	**[350, 250]**
Activation Functions	**ReLU**
DDPG	
Batch Size	**256**
Training Steps	50
Rollout and Evaluation Steps	100
Replay Buffer Size	**1_E6**
Soft Update Coefficient (τ)	$1_E{-3}$
Parameter Noise	None
Action Noise	**Ornstein-Uhlenbeck $\mu = 0.3, \sigma = 0.3, \theta = 0.15$**
PPO	
Steps / Environment / Update	**256**
Entropy Coefficient	**$1_E{-2}$**
Value Function Coefficient	0.5
Max Gradient Norm	0.5
Bias-Variance Coefficient (λ)	0.95
Minibatches	4
Policy Clipping Range	0.2
Value Clipping Range	None
Surrogate Optimization Epochs	4

Figure 7 reports training curves for example on- and off-policy algorithms. We contrast the performance of PPO (an on-policy method) and DDPG (an off-policy method). We also overlay the performance of our heuristic policies for comparison. Each figure shows the median and 10^{th} - 90^{th} percentile episode return across 10 different seeds.

Our selected hyper-parameters are reported in Table I. For all cases, we used fully-connected neural networks with hidden layers of size 350 and 250 with ReLU activation. Where applicable, we use separate networks for the actor and critic (i.e. no shared weights).

We ran additional experiments using TRPO, A2C and SAC and observed similar performance to the reported results. Training curves for these algorithms are not included here for brevity.

B. Characterising Learned Policies

To verify the learned policies were sensible (i.e. to confirm the absence of 'reward hacking') we qualitatively and quantitatively investigated the exhibited behaviours.

We observed that a key difference between successful and unsuccessful trained policies seemed to be the ability

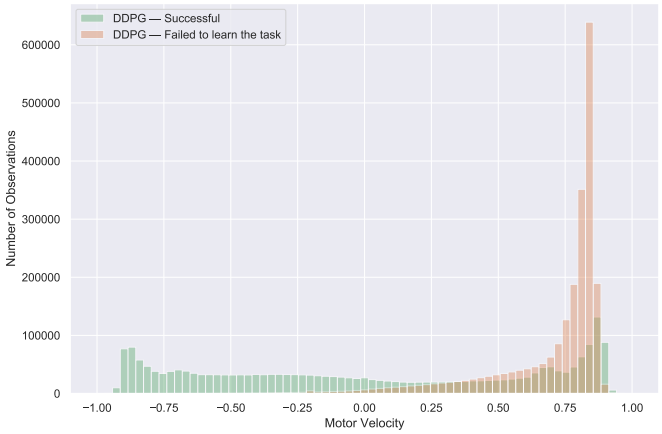

Fig. 4: Characterising policy behaviours for the *Move In Direction* task. We plot the distribution of motor velocities across many episodes for a successful policy (green) and unsuccessful policy (orange). The successful policy learns to pulse the motor in alternating directions as indicated by the spikes near ± 1 in the x-axis. Interestingly, this is the same strategy used by our heuristic policies. In contrast, the unsuccessful policy gets stuck in a local minima where the motor is continuously driven in one direction.

Fig. 5: Heatmap showing Jitterbug position over 100 episodes before (left) and after (right) training DDPG on the task *Move In Direction*. In the second figure, to evaluate the agent, the target direction was fixed at $+45°$.

to learn piecewise control functions. For example, for all tasks but *Move From Origin*, achieving high reward requires careful modulation of the reactive torque applied to the Jitterbug body by the motor counterweight. One way to achieve this is by pulsing the motor in different directions – this is the method we use in our heuristic policies. We observed that successful policies learned to pulse the motor in short bursts in alternating directions (e.g. $\sim 180°$ at a time), whereas unsuccessful policies would often drive the motor continuously (Figure 4). In doing so, the successful policies were able to achieve high cumulative episode return, and accomplish the high-level task encoded by the reward (Figure 5).

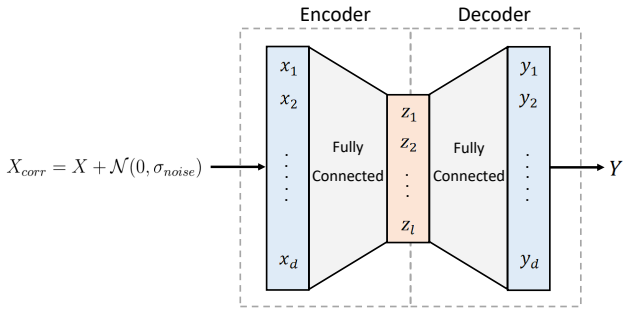

Fig. 6: WE used a De-Noising AutoEncoder as a means to learn a reduced-order state representation.

C. Reduced-Order Training

We use a De-noising AutoEncoder approach to learn a reduced-order observation representation (Figure 6). Input states X are corrupted by adding random unbiased noise to each of the features $X_{\text{corr}} = X + \mathcal{N}(0, 0.1)$. The AutoEncoder is then trained with input corrupted data X_{corr} and uncorrupted target labels X. We implement this using the mid-level API in the `tensorflow` library [32]. To optimize the AutoEncoder network we use the Mean Squared Error between the output Y and the original data X as a loss and use the Adam optimizer with default settings [31].

Training data were gathered by running a random policy on the Jitterbug for 5 million steps, and we used 80% of this data for training and 20% as a held-out test set. Once trained, the decoder side of the AutoEncoder was discarded and the encoder used to provide an augmented observation representation to a Deep Reinforcement Learning algorithm.

VI. DISCUSSION

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

VII. CONCLUSION

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

ACKNOWLEDGMENT

We thank Dr. Hanna Kurniawati at the ANU Robust Decision-making and Learning Laboratory for discussions and simulation assistance. This research was partly supported by an Australian Research Council Discovery Project (DP160100714). A. J. Snoswell is supported in part through and Australian Government Research Training Program Scholarship.

REFERENCES

[1] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[5] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

[6] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," *arXiv preprint arXiv:1708.04133*, 2017.

[7] S. Bhatnagar, D. Precup, D. Silver, R. S. Sutton, H. R. Maei, and C. Szepesvári, "Convergent temporal-difference learning with arbitrary smooth function approximation," in *Advances in Neural Information Processing Systems*, pp. 1204–1212, 2009.

[8] M. W. Spong, "Underactuated mechanical systems," in *Control problems in robotics and automation*, pp. 135–150, Springer, 1998.

[9] J. G. Nichol, *Design for energy loss and energy control in a galloping artificial quadruped*. PhD thesis, Stanford University, 2005.

[10] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. 1998.

[12] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, pp. 1329–1338, 2016.

[13] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 512–519, IEEE, 2016.

[14] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8, IEEE, 2018.

[15] R. Tedrake, "Underactuated robotics: Learning, planning, and control for efficient and agile machines: Course notes for mit 6.832," tech. rep., Massachusetts Institute of Technology, 2009.

[16] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*, vol. 19. SIAM, 2010.

[17] O. Von Stryk, "Numerical solution of optimal control problems by direct collocation," in *Optimal Control*, pp. 129–143, Springer, 1993.

[18] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*, pp. 4569–4574, IEEE, 2011.

[19] K. Nakajima, H. Hauser, T. Li, and R. Pfeifer, "Information processing via physical soft body," *Scientific reports*, vol. 5, p. 10487, 2015.

[20] J. Z. Kolter, C. Plagemann, D. T. Jackson, A. Y. Ng, and S. Thrun, "A probabilistic approach to mixed open-loop and closed-loop control, with application to extreme autonomous driving," in *International Conference on Robotics and Automation*, pp. 839–845, IEEE, 2010.

[21] G. J. Maeda, S. P. Singh, and H. Durrant-Whyte, "A tuned approach to feedback motion planning with rrts under model uncertainty," in *2011 IEEE International Conference on Robotics and Automation*, pp. 2288–2294, IEEE, 2011.

[22] A. Nagabandi, G. Yang, T. Asmar, R. Pandya, G. Kahn, S. Levine, and R. S. Fearing, "Learning image-conditioned dynamics models for control of underactuated legged millirobots," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4606–4613, IEEE, 2018.

[23] T. Nishimura, K. Mizushima, Y. Suzuki, T. Tsuji, and T. Watanabe, "Thin plate manipulation by an under-actuated robotic soft gripper utilizing the environment," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1236–1243, IEEE, 2017.

[24] A. Ng, "Sparse autoencoder," cs294a lecture notes, Stanford University, 2011.

[25] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet, "Learning latent plans from play," *arXiv preprint arXiv:1903.01973*, 2019.

[26] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[27] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai hym," *arXiv preprint arXiv:1606.01540*, 2016.

[29] A. Hill, A. Raffin, M. Ernestus, A. Gleave, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines." `https://github.com/hill-a/stable-baselines`, 2018.

[30] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines." `https://github.com/openai/baselines`, 2017.

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

[32] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

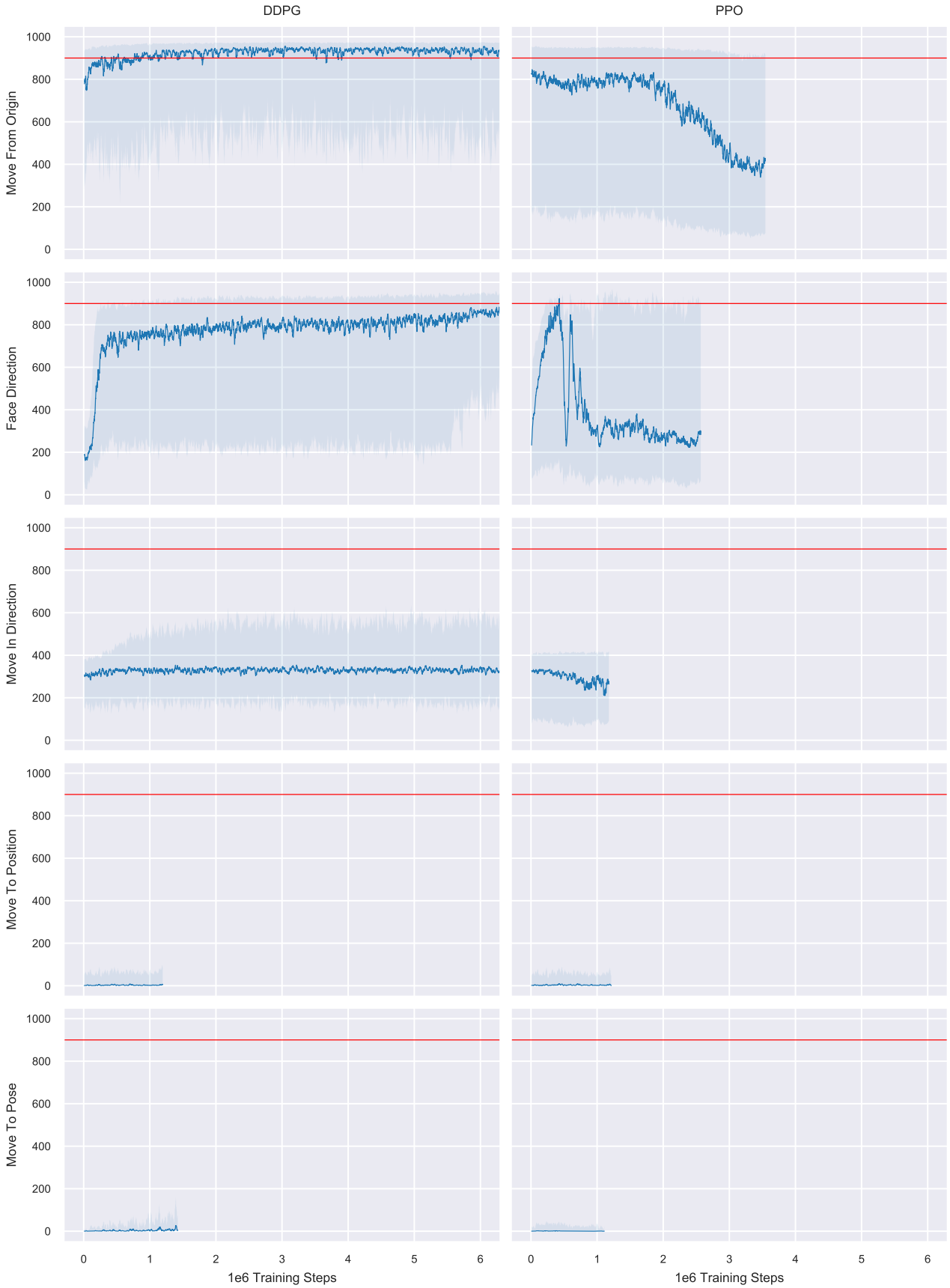

Fig. 7: Characterising the Jitterbug task suite. We compare the training progress of DDPG and PPO up to 6 million training steps (6000 episodes). We show median (solid line) and the 10^{th} and 90^{th} quartiles (shaded area) of per-episode episode return across 10 random seeds in each figure. A task is considered 'solved' if the trained agent consistently scores $\gtrapprox 900$ return per episode (red line). All plots are filtered with a 20_{E}^{3} step moving average filter.