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*— Underactuated robot designs are enticing due to their electromechanical simplicity; but, their operation is complex especially for compliant designs that may not have an explicit model. Deep reinforcement learning methods together with multi-joint dynamic simulation may help overcome this control bottleneck. However, such systems tend to have large continuous action spaces and exhibit sparse rewards, thus making control policy exploration variable and non-trivial.**

Using the observations: (1) that an underactuated system has coupled states, and (2) that a deep function approximator may generalize across compressed (and potentially nonintuitive) states, we consider a reduced order approach based around a generative autoencoder so as to automatically find a better representation to focus a control policy exploration process.

To help evaluate this approach, we also introduce *The Jitterbug Problem*, which is a series of increasingly specific and challenging motion control tasks for a highly compliant legged toy robot with just a single motor locomoting across a field. We benchmarked this problem against off-policy and on-policy deep reinforcement learning algorithms and find that an off-policy approach had better median rewards, but has higher variability. Through this study we find that reducing the model by taking advantage of its latent structure may exhibit similar (reward) performance, yet empirically has less training variance and slightly better learning rates.

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 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]. Simplifying the system 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].

[1]Ecole CentraleSupélec, Paris, France

[2]The Robotics Design Lab at The University of Queensland, Brisbane, Australia. {j.augot, a.snoswell, spns}@uq.edu.au

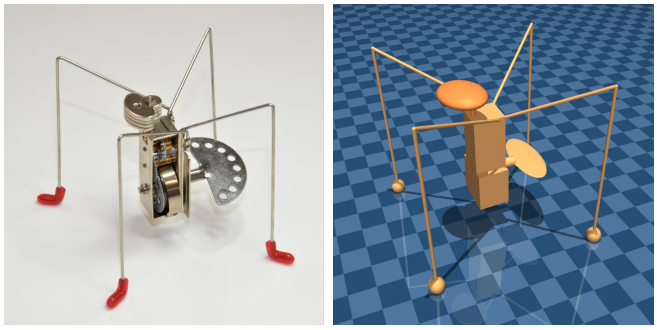

Fig. 1: **The Jitterbug Problem**: The wind-up children's toy 'Katita' (left) was the inspiration for our underactuated 'Jitterbug' continuous control task (right). In the simulated robot the wind-up spring is replaced with a controlled single degree-of-freedom motor. For scale, the blue checks on the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

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 interdependence 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, which we term '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).

Reduced order approaches are a common strategy in engineering. An issue herein is that an underactuated 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 over short horizons in the state-space. However, the motion is clearly non-linear due to the contacts and locomotion phases/gaits. (Indeed, it has been reported that the Katita toy, on which the problem is inspired, exhibits a 'galloping' locomotion mode [9].) An approach for automatically discovering interesting structure in empirical models is unsupervised learning. Autoencoders are a well know neural network approach for this that can discover underlying nonlinear 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.

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

II. Related Work

There is a good deal of work, some rather recent, in both the robotics and machine learning communities that is relevant to this work. The underactuated physical robotics task has the property of continuous states and actions with some correlations between these states. Thus, the control strategy has to operate over continuous actions, and, ideally, could take advantage of the latent structure.

A. Deep Reinforcement Learning for Continuous Control

Reinforcement Learning (RL) methods offer automated tools for controller design of such systems via trial and error [11], but have been been limited by feature representations and forward models [12]. Deep reinforcement learning algorithms typically address this through the use of deep neural networks for both the policy ('actor') and value function ('critic') which has allowed for operation to the continuous control domain [2], [13]. Such systems may be grouped as using 'on-policy' or 'off-policy' learning strategies; where on-policy methods, such as PPO [3] and TRPO [14], are those that use a current policy's action to update the value (Q) and off-policy approaches are those methods, such as DDPG [2], SAC [4], TD3 [5], that update the value via the best action and its total discounted future reward.

In the context of highly variable, underactuated robotics, on-policy methods may exhibit have lower sample efficiency as they would require new samples to be collected for each gradient step [4]. Conversely, off-policy methods may utilize a replay buffer to minimize correlations between samples [2], but may have difficulties with brittleness to state-space parameters, stability, and convergence [7]. In both cases, however, reducing the dimensionality and using a more aligned state-space may reduce these issues.

B. Reduced Order Approaches for Underactuated Robotics

Underactuated robots are fundamentally coupled in their actuation space [15]. When a model is available control strategies including from 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 do stymie dynamical system and model reduction approaches as an explicit physical model may not be available [19].

Given that actuation space for underactuated robots is structured, an autoencoder would be able to automatically discover some of this structure [10], [20]. A concern, however, is that the reduced model is not always intuitive, which would complicate the subsequent synchronization with a model based control strategy. In this way model-free deep reinforcement learning continuous control methods are rather compatible with an autoencoder's reduced state description.

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.* [21] control a small legged robot in which a neural network forecasts the robot's state at the next timestep that 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 [22]. In both these cases the control model is analytic and 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. More generally, model-based methods enjoy sample efficiency, but pose challenges of generalization due the control's model and its assumptions.

In the machine learning domain, the combination of an autoencoder and deep reinforcement network has been considered for various contexts. This has been particularly effective in cases with high dimensional inputs, such as from video, interacting with low-dimensional independent constraints, such as obstacle locations [23]–[25]. It has shown promise in a host of challenging applications including visualmotor learning [23] and VR teleoperation [26].

The system architecture of this work relates to Lange, *et al.* [27] in that we train an autoencoder network, remove the decoder side, and then use the remaining encoder network to compress the input features that are used as part of a deep reinforcement learning control policy search. In difference to this work, we do see a benefit with a random policy, sparing the need for a 'hint-to-goal' heuristic. This idea has also been considered more recently by Kimura [28] for reducing the number of trials during the RL policy training phase. Also, in this work, the task is slightly different in that the action space is correlated not only across states, but also across subsequent states in time.

III. Method

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

We choose to use an autoencoder to reduce the state space dimension since it can generate a latent space by exploiting feature correlations nonlinearly [10]. More specifically, we use a denoising autoencoder to increase robustness of the feature compression [29]. This can be split into two parts:

- Defining and training an autoencoder that enables an efficient dimension reduction without loosing too much information in the data compression.

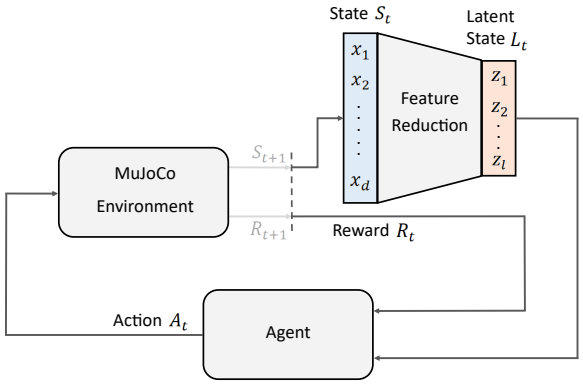

Fig. 2: **The architecture of our system to make an agent learn a task with a Encoder to reduce the state dimension.** The states (simulated via MuJoCo) are directly sent through the Encoder to generate new latent states with lower dimension.

- Integrating the trained encoder to the learning process in order to reduce the dimensionality of the states.

For the first part, we defined a denoising autoencoder composed of two fully-connected layers with $tanh$ activation functions. We train it to compress the states from an input to a latent space (i.e., encode), and then regenerate them as accurately as possible (i.e., decode). The training process is described in section V-B. Although unused in our later process, the decoder plays a critical part in making sure that the latent space is representative enough of the input.

Once trained, only the encoder is kept for the second step of our process. A Jitterbug agent is trained to perform a specific task on reduced dimension state space - see section IV-B for more details on the tasks. To do so, the state S_t and the reward R_t of the Jitterbug are read from the MuJoCo environment [30] at each time step t. The state goes through the encoder which outputs a latent variable L_t with a lower dimension. L_t is then sent to the agent along with the reward R_t in order to generate an action A_t. In this way, the Jitterbug agent never sees the real state S_t, but instead chooses an action by only considering the reduced latent state L_t. The DDPG algorithm is used because it has a reply buffer to minimize correlations between samples and because its target value (Q) network gives consistent targets subsequent iterations, which should help capture some of the sequencing need for underactuated robots. It shows promising results on solving tasks (see also Fig. 9 in Appendix).

IV. A Novel, Underactuated Control Benchmark

We implemented our Jitterbug benchmark using the DeepMind Control Suite (DMC) framework [31]. 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

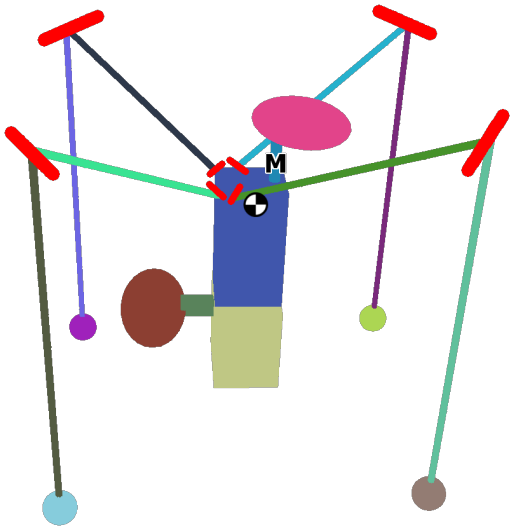

Fig. 3: **Schematic representation of the Jitterbug model.** Individual rigid bodies are in different colors and we highlight the position of the center of mass ( ), hinge joints ( ) and the single motor (M).

DMC uses the robust Multi-Joint dynamics with Contact (MuJoCo) robotics physics engine for simulation [30]. 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 [32], meaning many popular RL algorithm frameworks can be used with these benchmarks. For our evaluations, we used the `stable_baselines` library of RL algorithms [33], which is derived from the OpenAI `baselines` package [34].

A. The Jitterbug Domain

Our Jitterbug model was inspired by the children's toy Katita (Fig. 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 modeled 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.

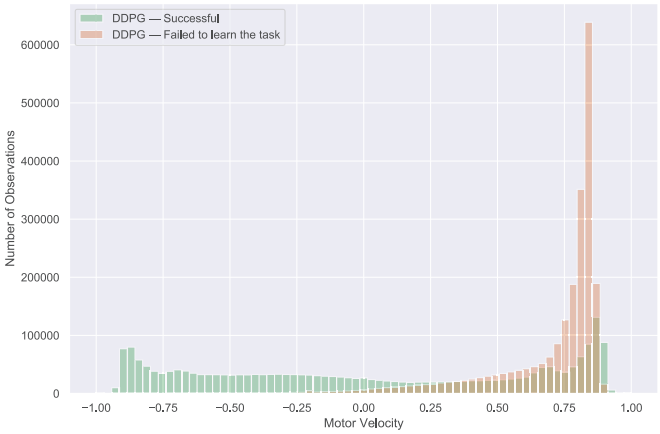

Fig. 4: **Characterizing policy behaviors for the *Move In Direction* task.** We plot the distribution of motor velocities across many episodes for a successful policy (green) and unsuccessful policy (orange). The successful policy learns to pulse the motor in alternating directions as indicated by the 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.

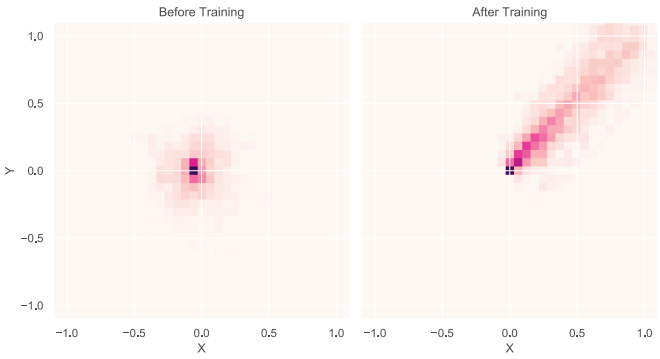

Fig. 5: **Heatmap showing Jitterbug position over 100 episodes.** The data were collected before (left) and after (right) training DDPG on the task *Move In Direction*. In the second figure, to evaluate the agent, the target direction was fixed at $+45°$.

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 observe some learned strategies that appeared to utilize this mode of locomotion!

V. Experiments

A. Characterizing Learned Policies

To verify feasibility, we hand-crafted heuristic policies that can solve each task. To characterize the difficulty of the Jitterbug task suite, we performed preliminary hyperparameter 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 [35]
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 9 (in Appendix) details 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). 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.

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

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

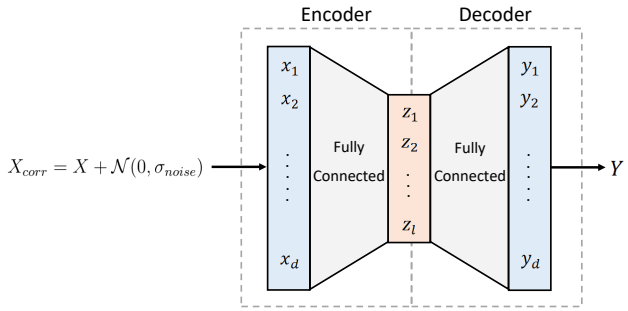

Fig. 6: **Schematic representation of a Denoising Autoencoder.** We use a Denoising Autoencoder as a means to learn a reduced order observation representation in an unsupervised manner. The inputs X have d features and the latent variables Z have $l < d$ dimensions.

– 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 (Fig. 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 (Fig. 5).

B. Learning Reduced-Order Representations

We use a Denoising Autoencoder (DAE) approach to learn a reduced-order observation representation (Fig. 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 [36]. 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 [35]. The MSE for each input X^i is defined in equation 1, with N being the total number of data points and d the number of features.

$$\forall i \in [1..N], MSE = \frac{1}{d}\sum_{k=1}^{d}(Y_k^i - X_k^i)^2 \qquad (1)$$

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.

C. Deep RL with Reduced-Order Representations

The next step is to learn the control policy. This is done by connecting the output of the DAE's encoder to the input to the agent for subsequent deep reinforcement learning. For this study, we used the DDPG algorithm as it is one of many widely accessible off-policy baseline methods. While different solvers may be used, the focus of this part was to investigate the influence of the reduced order approach.

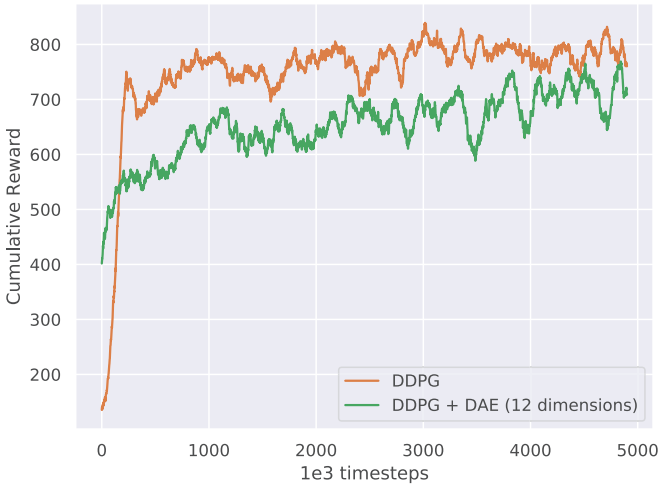

Fig. 7: **Performance of a deep reinforcement learning agent with a reduced-order representation.** We plot the training performance of a DDPG agent with a full and reduced-order model on the task *Face In Direction* up to 5 million steps. Orange: agent with full-order observations (16 dimensions). Green: agent with reduced-order observations from Denoising Autoencoder (12 dimensions). A single random seed is shown and a 100_{E}^{3} moving average filter is applied.

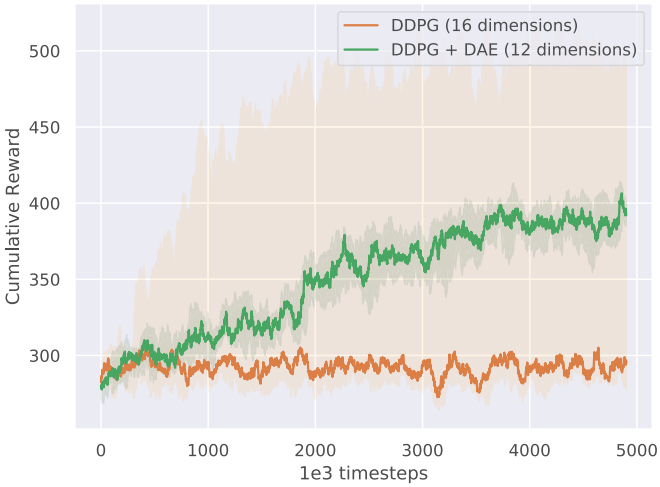

Fig. 8: **Apparent reduction in training process variance.** For a slightly more complex task *Move In Direction*, we compare the training progress of DDPG with a full (orange) and reduced-order representation (12 dimensions, green). This is done over 5 million training steps (5000 episodes). We show median (solid line) and the 25$^{\text{th}}$ and 75$^{\text{th}}$ 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). A a 100_{E}^{3} moving average filter is applied.

We considered this for a variety of tasks in the Jitterbug Problem. For example, in a simpler orienting task, such as *Face In Direction*, we observe that the training performance of a (half) reduced order representation is on par with a policy trained using the full representation (see also Fig. 7) with very slightly improved learning rates. Interestingly, as the ask got more complex, such as in the *Move In Direction* task, we empirically observe a reduction in the variance of the training process with a compressed representation (see also Fig. 8). While the median rewards are higher with the reduced order representation, the maximum reward found is higher with full action space. This is consistent with the observation that model-free control policy methods are more data intensive and can be more difficult to stabilize.

VI. Conclusion

In this paper we consider a reduced order approach based around a generative autoencoder so as to automatically find a better representation to focus a control policy exploration process for a variable and compliant underactuated robot. This is based on the intuition that underactuated systems have dynamic couplings between their state features by their very nature, and therefore can still be efficiently controlled by reducing the input state space dimension. Also, as part of this, we introduce the Jitterbug problem as an example of such a robot with the job of navigating in an environment. It has one actuator (the minimal possible) and has navigation tasks with up to three degrees of specification.

These assumptions and the approach are investigated by showing that a Jitterbug agent can be controlled with fewer features and still solve a task as efficiently as with using the original state space. As part of this, we chose a denoising autoencoder to reduce the state space dimension since it is a common tool widely used for data compression and correlation finding. Empirically we find that for more complex tasks the reduced order representation seems to reduce the training variance and have higher median reward. For lower complexity tasks, we see similar results between the approaches. Perhaps, this also intuits from model order in a planning and control perspective. In general, when planning for a robot one assumes that there is a forward model which may guide the subsequent selection of future actions. The more information one has, the more there is to exploit, but also the more there is to explore. Thus, a reduced order model might help automatically focus the inherent exploration and exploitation trade-off present. In conclusion, we see that a 'reduced order' modeling approach that takes advantage of the latent structure offers benefits and may help inform an automatic control software process for a novel and interesting class of 'reduced order' (underactuated) robot hardware.

Appendix

Benchmark Performance of the Jitterbug Task Suite

We consider (in Fig. 9) the benchmark performance of the various tasks in the Jitterbug problem task suite using both DDPG [2] (an off-policy method) and PPO [3] (an on-policy method). We find that DDPG is able to find rewards more stably, which we assume is because of its replay buffer.

ACKNOWLEDGMENT

We thank Dr. Hanna Kurniawati at the ANU Robust Decision-making and Learning Laboratory for discussions and computing assistance. This research was partly supported by an Australian Research Council Discovery Project (DP160100714). A. J. Snoswell is supported in part through an 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*. A Bradford Book, 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] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[14] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[15] R. Tedrake, "Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for MIT 6.832)," tech. rep., Massachusetts Institute of Technology, 2019.

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

[21] 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.

[22] 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.

[23] 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.

[24] S. Bitzer, M. Howard, and S. Vijayakumar, "Using dimensionality reduction to exploit constraints in reinforcement learning," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3219–3225, IEEE, 2010.

[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] 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.

[27] S. Lange and M. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2010.

[28] D. Kimura, "DAQN: Deep auto-encoder and q-network," *arXiv preprint arXiv:1806.00630*, 2018.

[29] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, ACM, 2008.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

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

[36] 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.

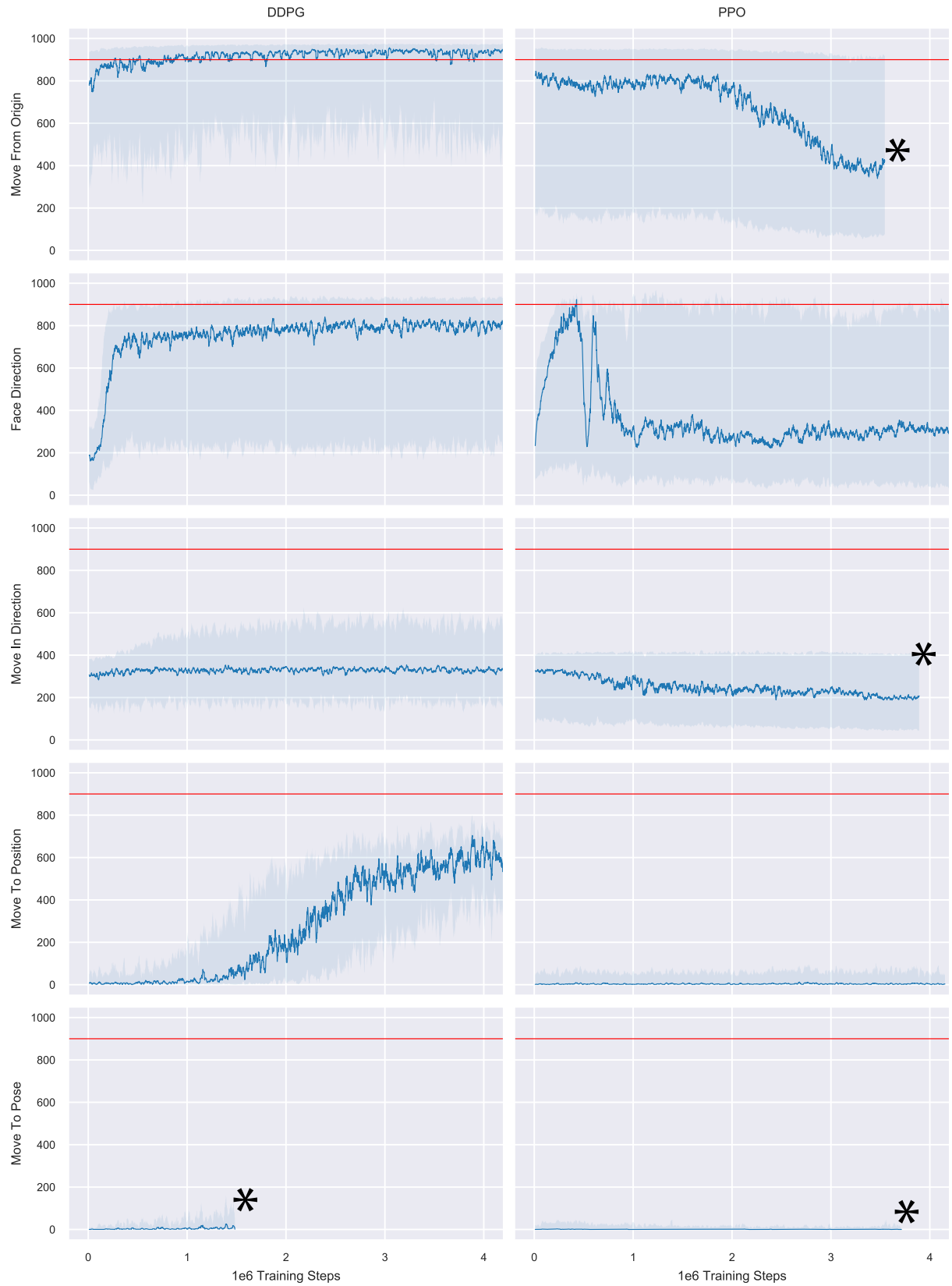

Fig. 9: **Characterizing the Jitterbug task suite.** We compare the training progress of DDPG and PPO up to 4 million training steps (4000 episodes). We show median (solid line) and the 10^{th} and 90^{th} quartiles (shaded area) of per-episode episode return across 10 random seeds in each figure. A task is considered 'solved' if the trained agent consistently scores $\gtrapprox$ 900 return per episode (red line). All plots are filtered with a $20_{\text{E}}{}^3$ step moving average filter. For the cases where the results stop early (noted with a $*$), this is because the simulations were limited to have the same compute time for comparison.