Regularizing Action Policies for Smooth Control with Reinforcement Learning – Supplementary Material –

Siddharth Mysore
Department of Computer Sciences
Boston University United States
sidmys@bu.edu

Bassel Mabsout
Department of Computer Sciences
Boston University United States
bmabsout@bu.edu

Renato Mancuso
Department of Computer Sciences
Boston University United States
rmancuso@bu.edu

Kate Saenko
Department of Computer Sciences
Boston University United States
saenko@bu.edu

Keywords: Reinforcement Learning, Smooth Control, Regularization

A Toy Environment

A.1 Setup

The code for the toy environment and corresponding training is provided in the zip folder included with this submission and, in the case of the paper's acceptance, will be made public through github. The simple toy environment is constructed as follows:

- **State space**: At time, t, the agent has a required goal, $g_t \in [-10, 10]$, and a current state, $o_t \in [-10, 10]$. These are stored internally. The state observation returned to the agent is a 1D error, $s_t = g_t - o_t$.
- **Goal generation**: Goals are procedurally generated where g_t is the result of a function mapping $g(t) \rightarrow [-10, 10]$. Available goal generators include: [simplex[1], constant, step]
- **Action Space**: Actions are limited to the range of $a_t \in [-1, 1]$.
- **System Dynamics**: The new state in response to action, a_t is computed as $o_{t+1} = \text{clip}(o_t + a_t, -10, 10)$.
- **Reward**: Reward is computed as $r_t = -|g_t - o_{t+1}|$.
- *Ideal solution*: This task could be easily solved by acting proportionally to s_t, with the ideal response being $a_t = \text{clip}(s_t, -1, 1)$.

A.2 Filtering

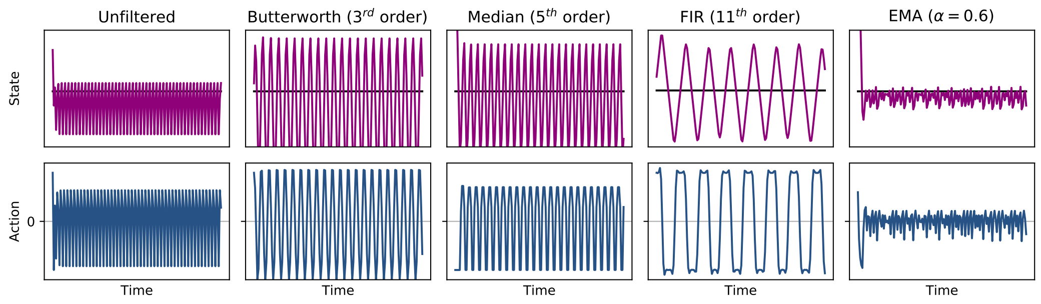


Figure 1: Filtering on the behavior of an agent that is not smooth. In our main paper, we showed that filtering on a well trained agent can result in poor tracking performance. Here we show that filtering a functional, but poorly trained agent — evidenced by a rudimentary ability of the agent to track the signal when unfiltered — can result in a catastrophic loss of control if filters are tuned for a controller that works well. With the lack of reproducibility in vanilla RL, this could result in two similarly trained agents behaving remarkably differently when filtered if filters are not individually tuned, adding to the difficulty of reproduction of functional control.

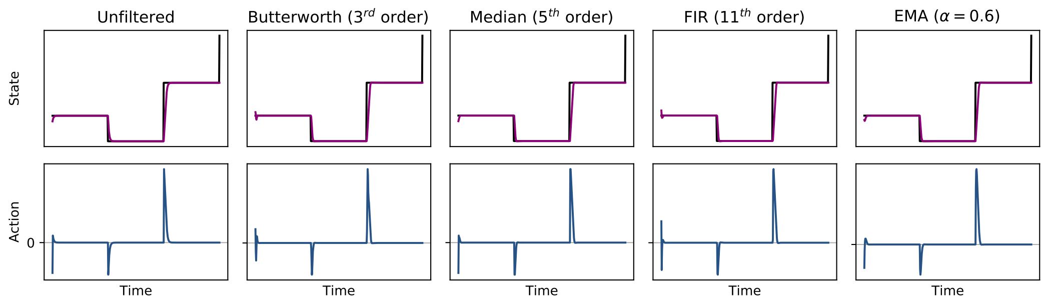


Figure 2: Filtering used on the smooth agents shown in the main paper applied to the PID controller on which they were tuned. Note that we do not observe any similar adverse effects of filtering with any of these filters that we demonstrated in Fig. 3 of the main paper. We tuned the filters for critical damping on the PID controller. The cut-off frequency and filter order were chosen to minimize overshoot and lag.

A.3 Demonstration of Mapping

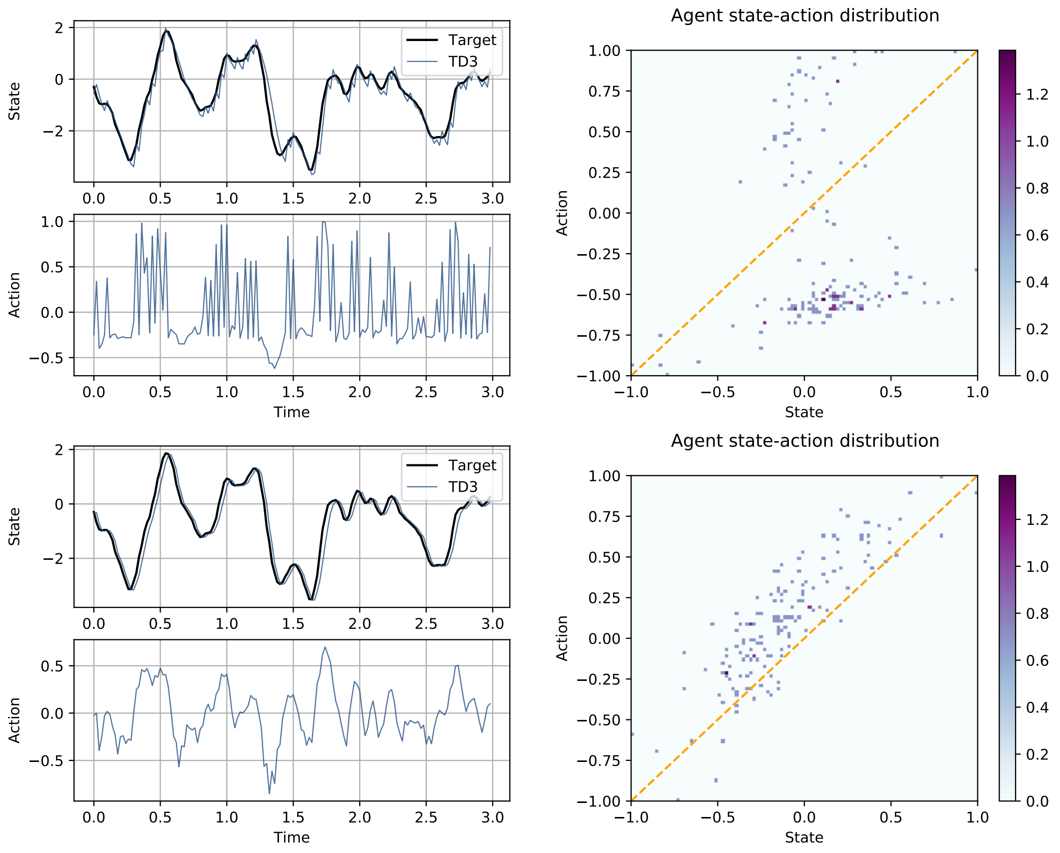


Figure 3: Comparison of state-action mapping of an unregularized TD3 policy (above) vs a CAPS regularized TD3 policy of the same architecture (below), with color-bars representing the log-scaled magnitudes (for clearer visualization) of the 2D action histograms. Note that the the actions of the regularized agent are much closer to the ideal linear mapping (shown by the orange line) which would optimally solve this toy problem.

B OpenAI Gym Benchmarks

B.1 Setup — Algorithms & Basic Hyper-parameters

We used 4 algorithms in our tests: PPO [2], DDPG [3], SAC [4], and TD3 [5]. For all the algorithms except PPO, we used the implementations provided by OpenAI's Spinning Up code-base [6]. For PPO, we used the implementation provided by OpenAI Baselines [7] as the Spinning up implementation omitted the running mean and standard deviation estimation for simplicity, but this does inherently change the behavior of the algorithms. For consistency and reproducibility. the hyper-parameters used for each environment/algorithm were copied from the RL zoo[8], provided by the stable baselines library [9].

We used the following 4 training environments in our experiments:
We used Pendulum-v0 (Pendulum), LunlarLanderContinuous-v2 (Lunar Lander), Ant-v2 (Ant), and Reacher-v2 (Reacher).

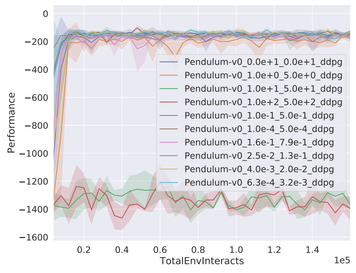
The code training with CAPS is provided in the zip folder included with this submission and, in the case of the paper's acceptance, will be made public through github.

B.2 Choosing CAPS Regularization Weights

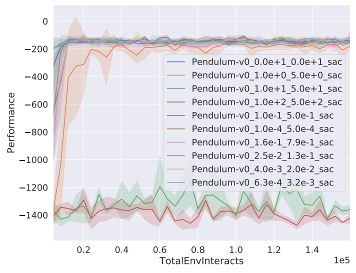
In order to choose the regularization parameters for CAPS, we compared unregularized agents to CAPS-regularized agents each trained with at least 11 random seeds and different regularization values. We then selected regularization that consistently yielded the best tradeoff of smoothness and performance on rewards. Training plots are shown in Fig. 4. These figures do not however show results for PPO as it was not based on the Spinup [6] code-base and was incompatible with the visualization tools, however we followed a similar procedure for determining good agents, only based on console outputs instead of graphs.

We settled on the following regularization weights, λ_a and λ_s for each environment:

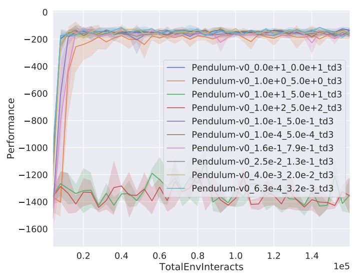
- Pendulum
  - PPO: $\lambda_a = 1.0e - 2$, $\lambda_s = 5.0e - 2$
  - *others*: $\lambda_a = 1$, $\lambda_s = 5$
- Lunar Lander
  - PPO: $\lambda_a = 1.0e - 3$, $\lambda_s = 5.0e - 3$
  - *Others*: $\lambda_a = 1.0e - 1$, $\lambda_s = 5.0e - 1$
- Reacher, Ant: $\lambda_a = 1.0e - 1$, $\lambda_s = 5.0e - 1$



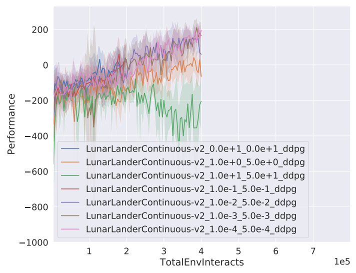
(a) Pendulum DDPG



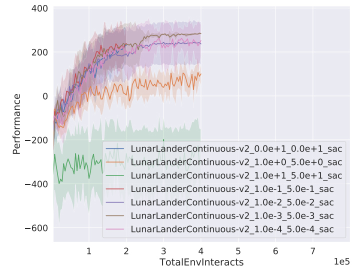
(b) Pendulum SAC



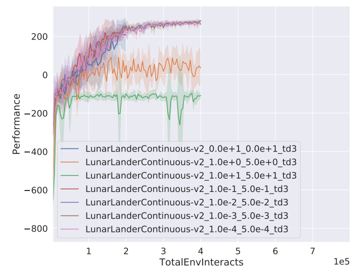
(c) Pendulum TD3



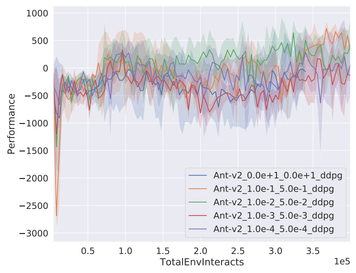
(d) Lunar Lander DDPG



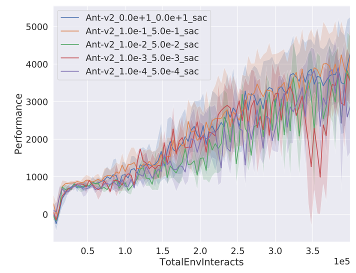
(e) Lunar Lander SAC



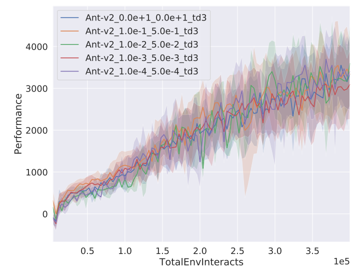
(f) Lunar Lander TD3



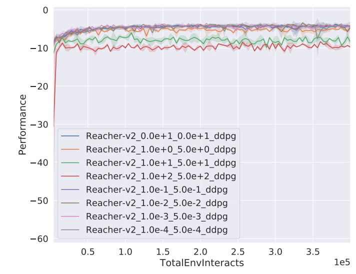
(g) Ant DDPG



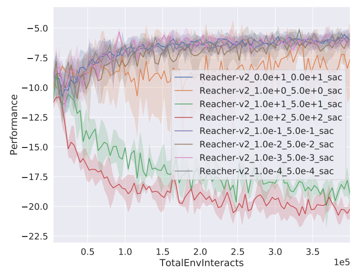
(h) Ant SAC



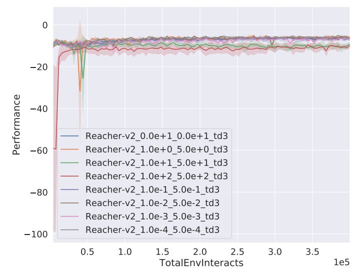
(i) Ant TD3



(j) Reacher DDPG



(k) Reacher DDPG



(l) Reacher DDPG

Figure 4: Comparing regularization with DDPG (left), SAC (middle), TD3 (left)

B.3 Vanilla Training vs. CAPS — Frequency Spectrum of Actions

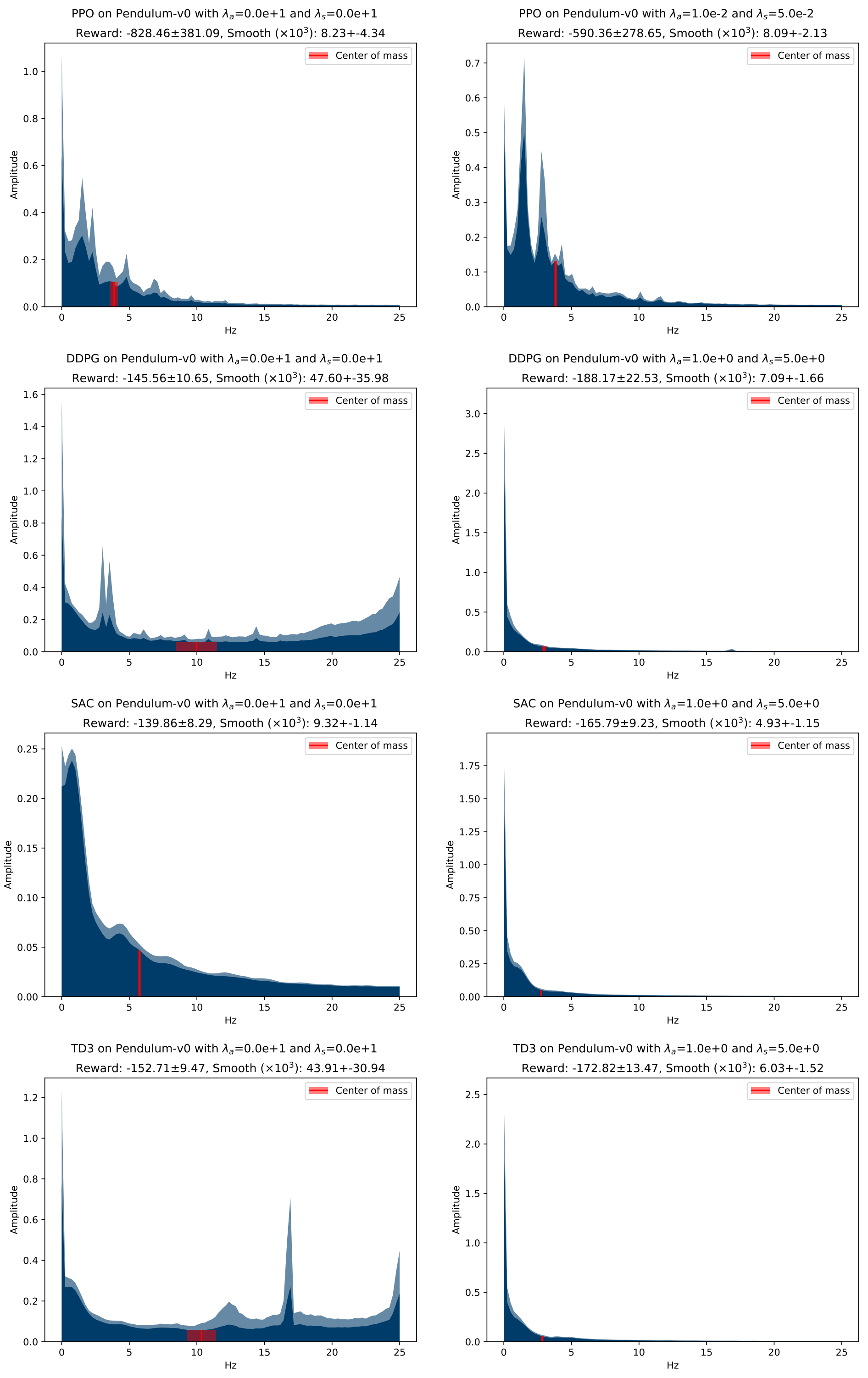


Figure 5: Comparing FFTs for vanilla agents vs CAPS on Pendulum (note the scale of the amplitude when comparing plots). Represented in dark-blue is the mean FFT while the variance is captured in light-blue. We also show the center of mass (CoM) to demonstrate how the CoM moves as a function of CAPS regularization.

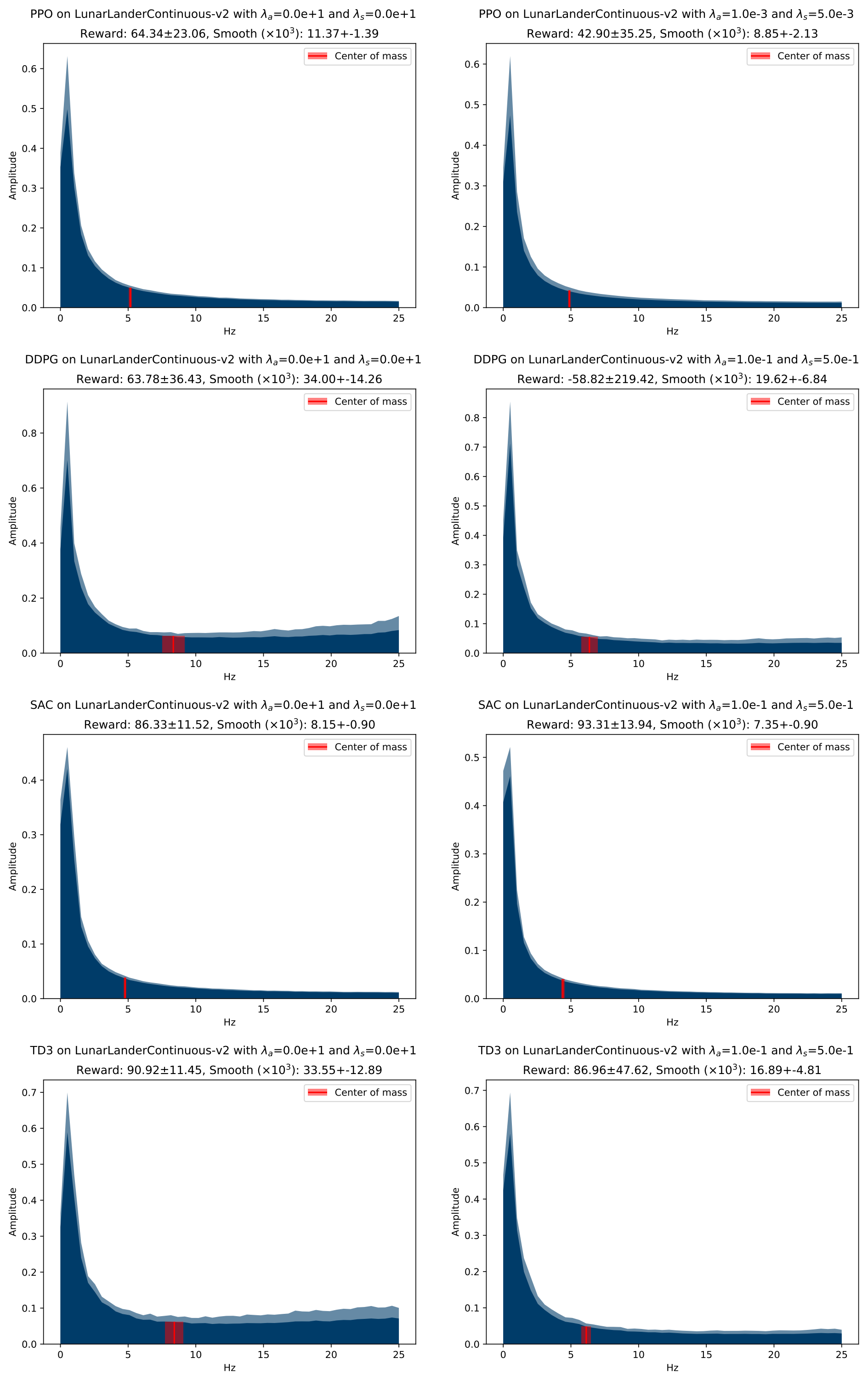


Figure 6: Comparing FFTs for vanilla agents vs CAPS on Lunar Lander (note the scale of the amplitude when comparing plots). Represented in dark-blue is the mean FFT while the variance is captured in light-blue. We also show the center of mass (CoM) to demonstrate how the CoM moves as a function of CAPS regularization.

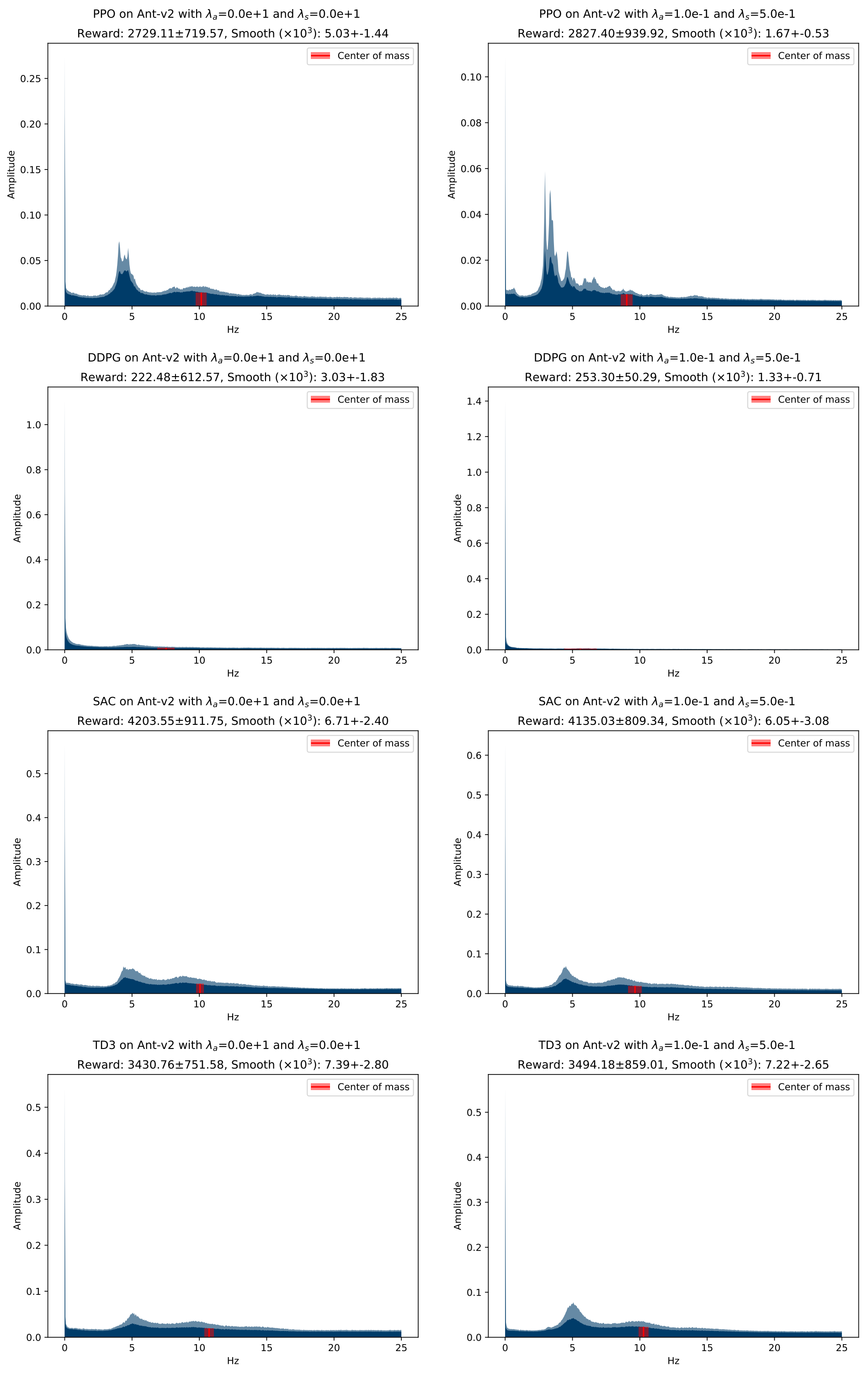


Figure 7: Comparing FFTs for vanilla agents vs CAPS on Ant (note the scale of the amplitude when comparing plots). Represented in dark-blue is the mean FFT while the variance is captured in light-blue. We also show the center of mass (CoM) to demonstrate how the CoM moves as a function of CAPS regularization.

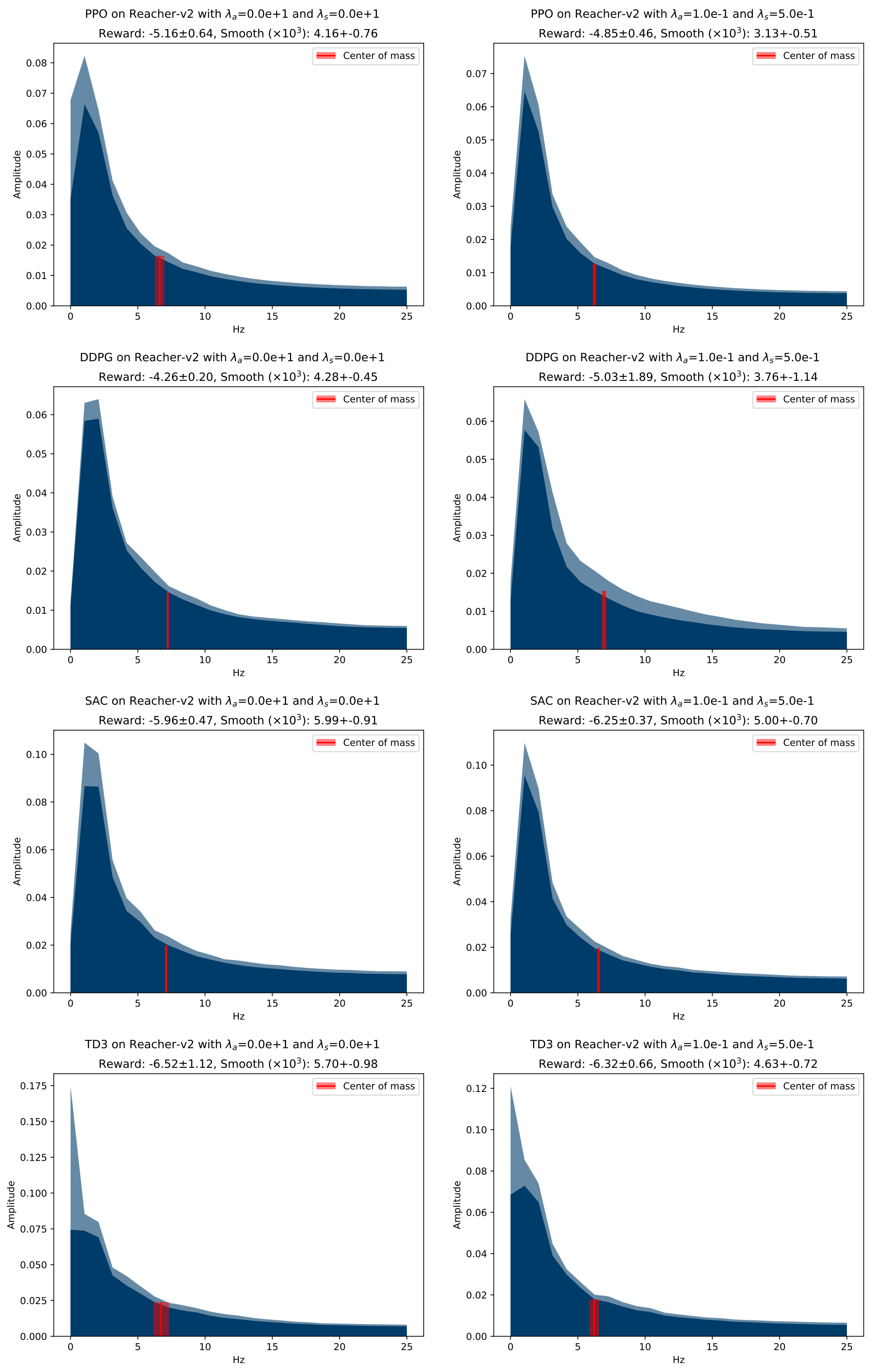


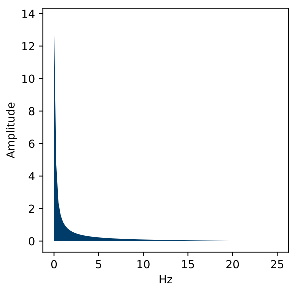
Figure 8: Comparing FFTs for vanilla agents vs CAPS on Reacher (note the scale of the amplitude when comparing plots). Represented in dark-blue is the mean FFT while the variance is captured in light-blue. We also show the center of mass (CoM) to demonstrate how the CoM moves as a function of CAPS regularization.

B.4 Frequency Response

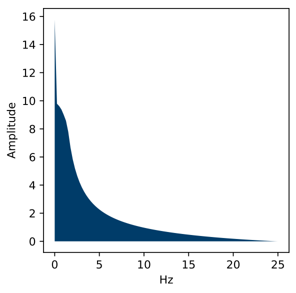
We performed an analysis of each of the simulated environments' frequency responses. To do this, we analyzed the responses of different parts of the simulated control systems (example: joint rotation) to actuation signals of different frequencies. Performing this analysis allowed us to get a better sense for how much head-room was available for improving the smoothness of control in the different environments.

We began by analyzing the simplest environment: Pendulum. By disabling simulated gravity, we were able to analyze the responses of differently weighted pendulum masses. We see that with a default mass of 1 unit, the responsiveness of the system is limited to low frequency control, with higher frequency components have negligible impact on the system. As a sanity-test, we also analyzed the responses of lighter pendulums. As we reduce the mass, we observe that the system's response to higher frequency actuation increases, as expected, as lower mass results in lower inertia. We show this in Fig. 9. It can be concluded based on these results that high frequency actuation has little value to the system and can be significantly reduced.

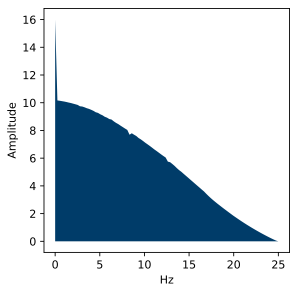
Similar to the pendulum, we observe that there is limited efficacy in high-frequency control for the Lunar Lander as well as the shoulder for the Reacher. With environments like the Ant and Reacher however, at least one of the the joints (or all the joints in the Ant's case) maintain significant responsiveness even at higher frequencies. Furthermore, the peak responsiveness is at a higher frequency. This means that it is very explicitly in the agents' best interests to operate at higher frequencies to gain the best rewards. This is reflected in the regularization results too as agents do not behave significantly more smoothly in terms of CoM frequencies. The CAPS agents in these cases however are able to exploit what additional smoothness they do gain however for consistently improved rewards.



(a) mass=1

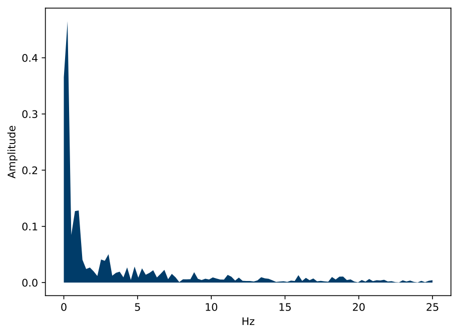


(b) mass=0.1

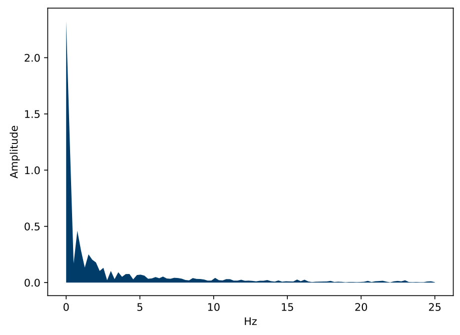


(c) mass=0.01

Figure 9: Figures 9a, 9b, and 9c show the response to torque applied on the single joint attached to the pole with differing masses showcasing the effect it has on the system's dynamics

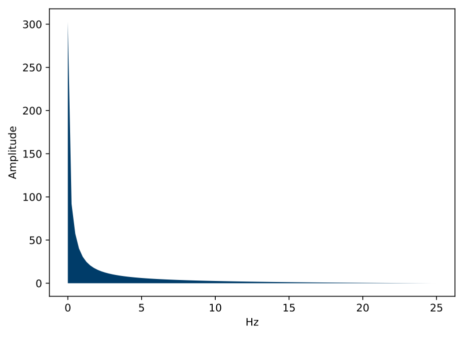


(a) Horizontal thruster

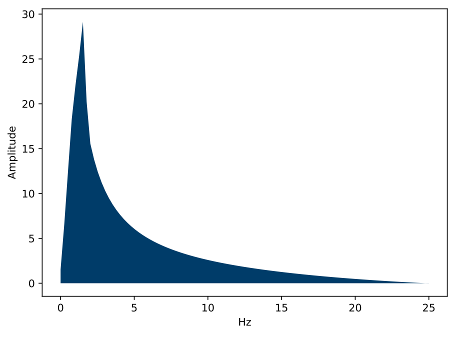


(b) Vertical thruster

Figure 10: Figures 10a, and 10b show the horizontal and vertical thruster responses to the horizontal and vertical velocity respectively for LunarLanderContinuous-v2

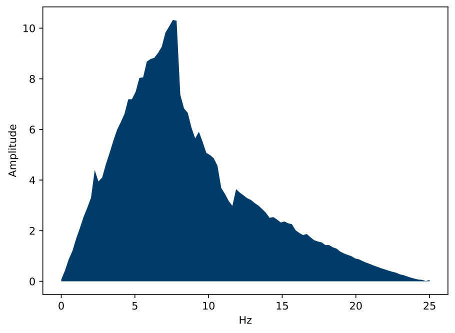


(a) Shoulder rotational rate

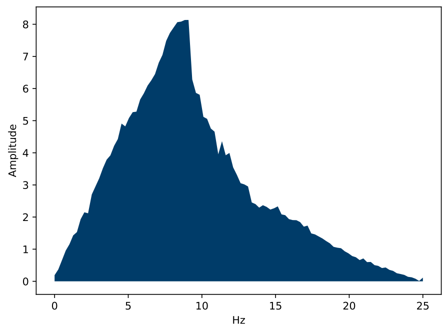


(b) Elbow rotational rate

Figure 11: Figures 11a, and 11b show the response to torque applied on the shoulder and elbow for Reacher-v2



(a) Hip rotational rate



(b) Ankle rotational rate

Figure 12: Figures 12a, and 12b show the response to torque applied on a single limb's hip and ankle for Ant-v2

C Quadrotor Drone

C.1 Training setup

While we utilize the general flight controller training pipeline developed by Koch [10], Koch et al. [11], we made some alterations to facilitate simpler training and to further illustrated the benefits of CAPS. Koch [10] developed a highly tuned reward signal, but they limited their training to just using step inputs. We noted that the trained controller appeared to have been over-fit to this signal structure and was not good at generalizing to different signal inputs, even in the same simulated environments with the same system dynamics, as demonstrated in Fig. 13. We found that training instead with a Perlin-noise-based signal [12] allowed for a better exploration of the state space during training and allowed for faster and more stable convergence. This signal also more closely reflects inputs from a pilot in real flight.

In addition to changing the target goal generation, we changed the rewards structure. For every time-step there are two components:

1. A tracking error based signal, $p_e = ||\phi - \bar{\phi}||_4$ where ϕ and $\bar{\phi}$ represent the current and desired rates of angular velocity respectively
2. A motor thrust requirement $p_T = ||y - 0.35||$, where $y \in \mathbb{R}^4$ is the current motor thrust fractions. We use reward engineering to condition motor output to be maintained at approximately 35% for attitude control.

We cast these rewards as positive and bounded signals as $p^+(p) = \min(1, \max(0, 1-p))$ and compose the final reward, r as:

$$r = \sqrt{(p^+(p_e))\,(p^+(p_T))}$$

While just using $p^+(p_e)$ can allow for the training of minimally flight-worthy agents, we discovered that it was necessary to have the output levels increased to match the typical thrust required to hover. The base neuroflight pipeline described by Koch et al. [11] includes a control mixer which mixes thrust control with attitude control. While attitude is controllable with low motor actuation, the true motor levels during flight were typically higher - minimally at hover thrust of 35%. By conditioning the agents to learn with motor thrusts typically maintained around 35%, we ensured that the simulated dynamics more closely matched true-flight dynamics.

C.2 Comparing Agents

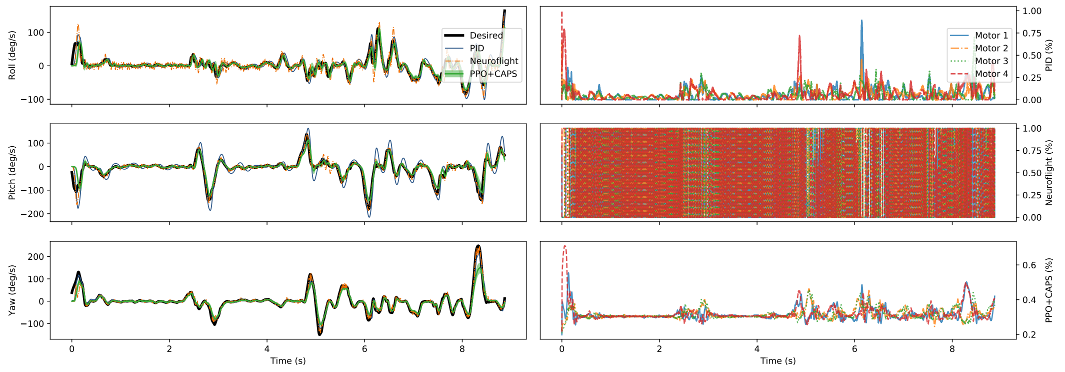


Figure 13: We compare the velocity tracking performance and simulated motor utilization for PID, Neuroflight and PPO trained with CAPS. Note that while all three controllers behave similarly in terms of tracking performance, PPO + CAPS is significantly smoother than the PPO agent trained by Neuroflight, despite having the same architecture.

C.3 Training progress

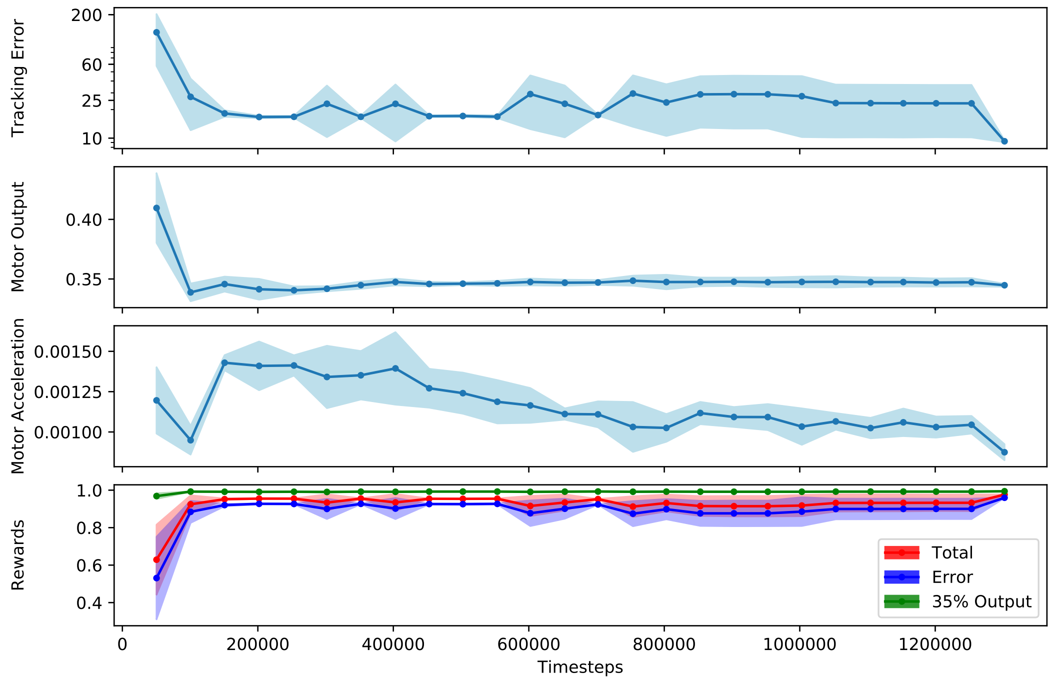


Figure 14: Training progress of PPO + CAPS. Training is halted when the mean absolute tracking error over roll, pitch and yaw falls below 10 deg/s in training, to prevent over-fitting. We also show the average motor acceleration to show that CAPS regularization does indeed reduce the rate of acceleration of the motors, consequently improving smoothness.

References

[1] Python noise library version 1.2.3. URL https://github.com/caseman/noise/tree/bb32991ab97e90882d0e46e578060717c5b90dc5.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[3] 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. *International Conference on Learning Representations*, 2016.

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning (ICML)*, 2018.

[5] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596, 2018.

[6] J. Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[7] 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.

[8] Openai gym rl zoo. URL https://github.com/araffin/rl-baselines-zoo/tree/2777c0376b2e96f146993b5b85a94ebaa0ea0b37/hyperparams.

[9] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines, 2018.

[10] W. Koch. Flight controller synthesis via deep reinforcement learning, 2019.

[11] W. Koch, R. Mancuso, and A. Bestavros. Neuroflight: Next generation flight control firmware. *CoRR*, abs/1901.06553, 2019. URL http://arxiv.org/abs/1901.06553.

[12] K. Perlin. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, page 287–296, New York, NY, USA, 1985. Association for Computing Machinery. ISBN 0897911660. doi:10.1145/325334.325247. URL https://doi.org/10.1145/325334.325247.