Appendix

A. SwaNNFlight System Transceiving Metrics

Using the Transceiving framework and hardware described in Section III, i.e MATEK-F722 controller equipped with ZigBee®-PRO radio-frequency modules, transmission and switching time for NNs were measured. The control networks used in this work use two hidden layers with 32 neurons each. We employ a transmission baudrate of 115200 to send NNs of 8mb, which takes 11 s to transfer between the ground station and the drone – receiving is handled in parallel to the flight control, so the drone is able to remain fully operational while waiting for a new network. When a network is received, switching to the new NN controller takes 134 ms. During flight, the drone also transmits state observations of 59 bytes back to the ground station, transmitting 244 observations per second.

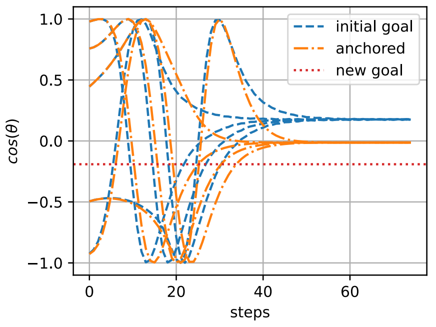
B. Hyper-parameter search

We conducted a hyper-parameter search and a correlation factor analysis using 4 different prominent RL algorithms – PPO, DDPG, SAC and TD3 – for the quadrotor attitude control problem mainly considered in this paper. The list of tested hyper-parameters and their corresponding correlations to the training reward are presented in Tables III, IV, V and VI respectively. Our hyper-parameter choices were informed by common values used to achieve the best performance across various benchmark Gym environments [31].

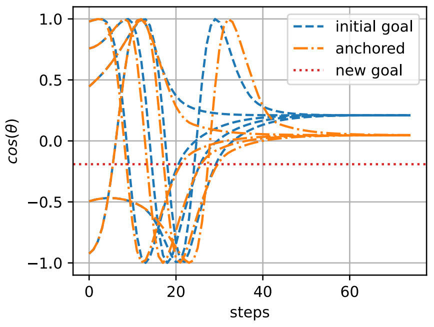
Table II meanwhile compares the training performances of each of the algorithms, as well as DDPG× which we introduce in this work, after 300000 training steps (corresponding to a quarter of our typical training time and the time by which DDPG× typically starts to plateau), but on a slightly modified version of the problem – where agents are only required to minimize tracking error, without worrying about other characteristics such as control smoothness. This, we found, was typically a good starting point for tuning algorithms' hyper-parameters and to determine if they are capable of addressing the underlying control problem. As the data demonstrates, all the tested algorithms perform significantly worse than DDPG×.

While the relative performance metrics appear similar across DDPG, PPO, SAC and TD3, there are critical differences in the quality of the behaviors being learned, which in turn impacts the ability of the algorithms to further improve agents' performances. As shown in Fig. 9, the tested TF2 implementation of PPO does not behave similarly to the modified OpenAI Baselines [12] TF1 version used in previous work. [7], [9], [10]. Where the PPO based on Baselines learns to track the control signals well, albeit with high variance in the motor actuation (since smoothness is not being trained for), the TF2 implementation saturates the motor commands at the maximum, thus effectively doing nothing to control the drone's attitude. This still generates some reward, since the training environment predominantly consists of smaller (but more frequently varying) control angular velocities. The problem with this, however, is that this behavior seems to be a local optima that is difficult for the algorithms to recover from to further improve performance.

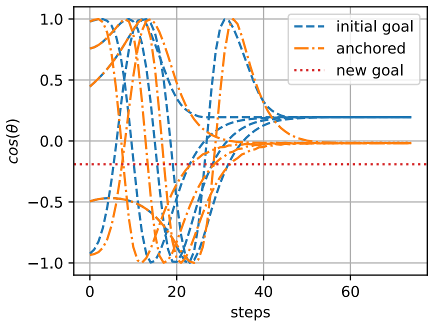
When considering alternatives to PPO, we experimented with Q-learning-based DDPG, SAC and TD3. As shown in Fig. 10, we observed that TD3 would often fall into similar trappings as PPO, with the motor outputs becoming saturated at maximum actuation. While DDPG was slower to learn (often performing worse than SAC), crucially, it did not saturate and would in fact consistently try to maintain control through more nuanced use of the motors – an extreme but illustrative case is shown in Fig. 10. Given our results, while SAC would have been preferable as a starting point based on the typical rewards achieved, the fact remained that training SAC was approximately 1.5 times slower than DDPG for relatively marginal gains, making it a less attractive option for iterating. DDPG was therefore deemed the best contender with which to experiment on our proposed multiplicative policy optimization. In our tests, agents which achieve an average reward of more 0.8 were typically responsive enough to control in real flight.



(a) DDPG



(b) SAC



(c) TD3

Fig. 8: Evolution of the angle ($sin(\theta)$) of an inverted pendulum over time for policies trained with anchor-critics tested on DDPG, SAC and TD3. There are 5 different test runs per goal, and the angle of the pendulum is initialized at random. All 3 anchored algorithms consistently learn to point to 0 rad, thus maintaining a good compromise between our initial goal and our new goal.

TABLE II: This table shows the performance of the different RL algorithms over the aforementioned environment. Training ends after invoking the environment 300,000 times and is tested 5 times for every hyperparameter sample. The reward value is then computed for every sample of hyperparameters of which there are at least 10 samples per algorithm. Step times are reported for a desktop running Ubuntu v 18.04 with an Intel Core i7-7700 CPU (no GPU optimization is used since training is primarily CPU limited).

Algorithm	DDPG×	DDPG	PPO (TF2)	SAC	TD3
Reward	0.89 ± 0.033	0.38 ± 0.13	0.36 ± 0.11	0.40 ± 0.12	0.37 ± 0.11
Max Reward	0.93	0.59	0.61	0.63	0.57
Step time (s)	9.6×10^{-3}	9.6×10^{-3}	3.6×10^{-3}	1.44×10^{-2}	1.2×10^{-2}

TABLE III: TF2 PPO paramater/reward correlations

Parameter	Correlation	Tested Values
Clip ratio	0.098	[0.0, 0.05, 0.1, 0.2]
γ	-0.100	[0.8, 0.9, 0.95, 0.99]
Policy learning rate	0.679	[1e-4, 5e-4, 1e-3, 3e-3, 5e-3]
Value learning rate	0.255	[1e-4, 5e-4, 1e-3, 3e-3, 5e-3]
lam	0.056	[0.9, 0.97, 0.99, 0.995]

TABLE IV: DDPG paramater/reward correlations

Parameter	Correlation	Tested Values
Replay buffer size	-0.156	[100 000, 500 000, 1000 000, 5000 000]
γ	-0.161	[0.8, 0.9, 0.95, 0.99]
Polyak	0.203	[0.5, 0.9, 0.95, 0.99, 0.995]
Policy learning rate	-0.211	[1e-4, 5e-4, 1e-3, 3e-3, 5e-3]
Critic learning rate	-0.676	[1e-4, 5e-4, 1e-3, 3e-3, 5e-3]
Batch size	-0.142	[50, 100, 200, 400]
Action noise	-0.691	[0.01, 0.05, 0.1, 0.2]

TABLE V: SAC paramater/reward correlations

Parameter	Correlation	Tested Values
Replay buffer size	-0.088	[100 000, 500 000, 1000 000, 5000 000]
γ	0.272	[0.8, 0.9, 0.95, 0.99]
Polyak	0.102	[0.5, 0.9, 0.95, 0.99, 0.995]
Learning Rate	-0.256	[1e-4, 5e-4, 1e-3, 3e-3, 5e-3]
α	0.356	[0.01, 0.05, 0.1, 0.2]
Batch size	0.866	[50, 100, 200, 400]

TABLE VI: TD3 paramater/reward correlations

Parameter	Correlation	Tested Values
Replay buffer size	0.516	[100 000, 500 000, 1000 000, 5000 000]
γ	0.479	[0.8, 0.9, 0.95, 0.99]
Polyak	0.007	[0.5, 0.9, 0.95, 0.99, 0.995]
Policy learning rate	-0.430	[1e-4, 5e-4, 1e-3, 3e-3, 5e-3]
Critic learning rate	0.023	[1e-4, 5e-4, 1e-3, 3e-3, 5e-3]
Batch size	0.118	[50, 100, 200, 400]
Act noise	0.313	[0.01, 0.05, 0.1, 0.2]

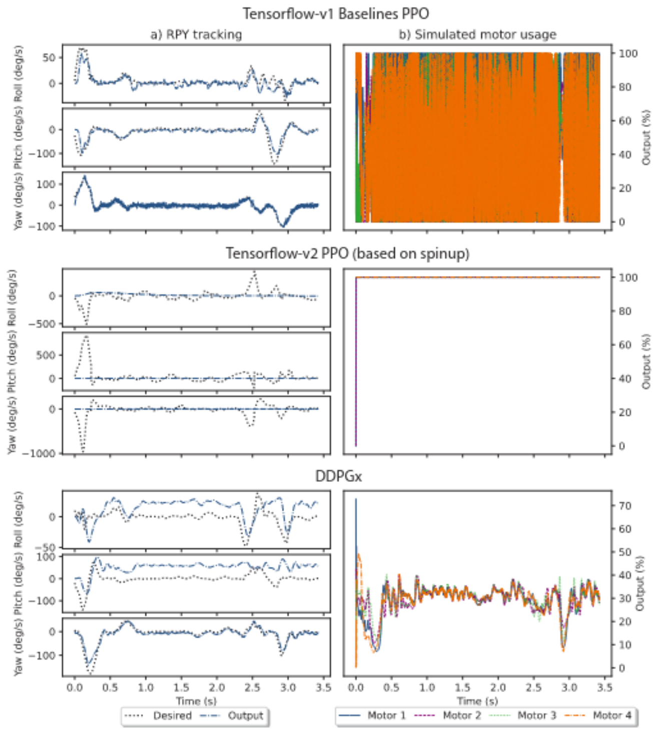


Fig. 9: A snapshot of PPO and DDPG× agents after 300,000 training steps acting in their training environment. Attitude control elements are split into Yaw, Pitch, and Roll in the left column. On the right are the outputs generated by the agents while attempting to follow the control inputs. Without smoothness considerations, within 300,000 steps, Baselines PPO performs very well on tracking while outputting actions with high variability (as is shown in previous works [10], [9]). Conversely, the TF2 implementation of PPO fails to learn useful behavior across hyper-parameters. DDPG×, which we introduced in this work, is able to reasonably follow the target while maintaining smooth outputs and continues to improve as training progresses.

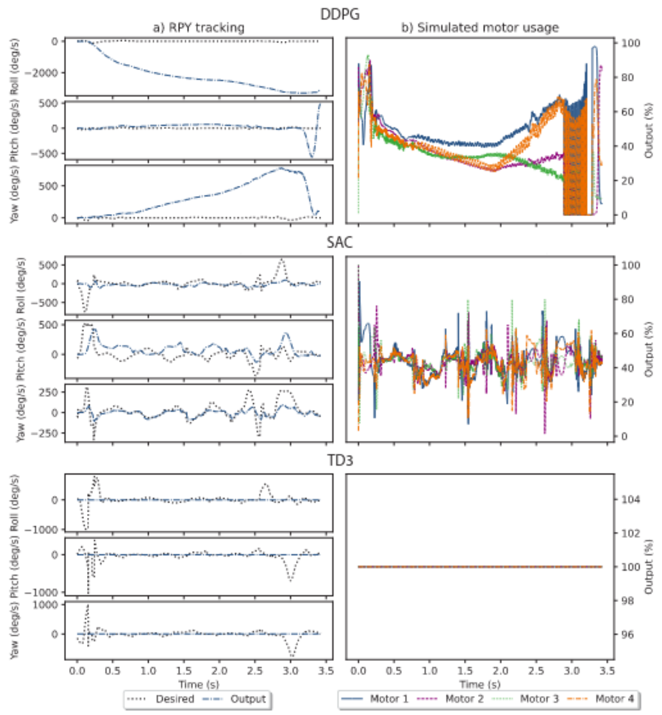


Fig. 10: For the three DDPG-based algorithms, we observe clear dissimilarities in the quality of the policies learned, based on input tracking and motor usage. Entropy regularization employed in SAC allows trained agents to achieve relatively reasonable performance on both tracking and motor usage, but the algorithm was noted to not improve further with more training, likely due to the same regularization that initially helps preventing large responses, thus preventing the algorithm from learning to track more aggressive maneuvers. TD3 behaves similarly to the TF2 PPO, in that it tends to saturate at various local optima, outputting extreme values. DDPG, on the other hand, demonstrates some tracking capabilities can varies its control output drastically when close to the setpoint, similar in behavior to Baselines PPO. Given these three candidate algorithms and their relative benefits and drawbacks, we chose to modify DDPG as it runs faster than SAC and was capable of achieving good tracking performance. While DDPG itself was not consistently reliable, DDPG×, with our proposed improvements, was repeatable and controllable.