Development of a reinforcement learning based controller for a VTOL drone

Christopher Narr
c.narr@tum.de

Oliver Hausdörfer
oliver.hausdoerfer@tum.de

***Abstract*—Developing controllers for Vertical Take-Off and Landing drones is difficult due to the complex underlying dynamics. The goal of this project is to develop a Reinforcement Learning based velocity-tracking controller for such a drone in simulation. The major contributions are to learn a controller for a drone with active wing flaps, to use information about the state history in the controller network and to incorporate gusty conditions in the simulation. As a main result, the tracking performance could be improved significantly compared to baseline by processing the last 0.5s of the state history with state dependent filters. Beyond tracking performance, flight efficiency and stability are improved as well by using this approach. Due to these benefits and ease of implementation of the network, we suggest to use similar designs in future works. The main reference papers are [1], [2] and [3].**

I. Experimental setup

Environment: The used drone is a Vertical Take-Off and Landing (VTOL) drone with a continuous 4D action space ($\boldsymbol{a} \in \mathbb{R}^4$), i.e. two thrusts and two active wing flaps. The task of the drone is to follow given 3D velocity trajectories. Algorithms to randomly sample these trajectories as well as random wind disturbances were implemented. Noise was added on the actuator and sensor signals to make the learned controller more robust (see Table V in appendix).

During training, the environments were terminated if one of the criteria in Table I were met or after 20 seconds. The termination criteria regarding the drones global z-position allowed for more stable training and was switched off during testing.

TABLE I
Simulation termination criteria

criteria	$v_{B,max}$	$a_{B,max}$	$\omega_{B,max}$	$\alpha_{B,max}$	z-pos.
value (SI)	10.0	1000.0	200.0	1000.0	-1

During training, the velocity trajectories and wind disturbances were randomly resampled on each reset of the environment. For testing, a test set with 100 fixed velocity trajectories and wind disturbances was used.

Learning Framework: We used the actor-critic PPO algorithm. The continuous 9D input state $\boldsymbol{s} \in \mathbb{R}^9$ consists of the 3D Euler angles α, β, γ, the rotational velocities in body frame $\boldsymbol{\omega}_B \in \mathbb{R}^3$ and the velocity tracking error in body frame $\boldsymbol{v_e} \in \mathbb{R}^3$. The velocity tracking error $\boldsymbol{v_e}$ is defined as the element-wise difference between the desired velocity $\boldsymbol{v_d} \in \mathbb{R}^3$ and the current velocity $\boldsymbol{v_t} \in \mathbb{R}^3$, both in body frame. The reward function consists of a constant stay alive reward r_{sa}, the tracking error cost $c_v = ||\boldsymbol{v_d} - \boldsymbol{v_t}||_1$, the cost for rotational velocities $c_\omega = ||\boldsymbol{\omega_B}||_1$ and the action cost $c_a = ||\boldsymbol{a}||_1$. The maximum achievable reward in our setting is 3200.

$$r = r_{sa} - w_v c_v - w_\omega c_\omega - w_a c_a \tag{1}$$

TABLE II
Reward function parameters

parameter	r_{sa}	w_v	w_ω	w_a
value	4.0	0.5	0.01	0.01

Neural Network Approximators: We trained different neural network architectures and compared their performance. Our baseline network is a simple fully connected network consisting of three layers with 128 neurons each and has the current drone state as input (Fig. 1).

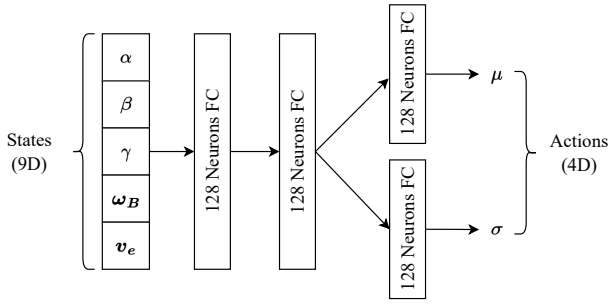

Fig. 1. Architecture of our baseline model consisting of fully connected feedforward layers. Shown is the Gaussian actor network. The current 9D state is used as input.

To incorporate the state history of the drone into the network we took three different approaches. First, we used the state history of *n* time steps for each state as input to a fully-connected network directly, which then consists of $9n$ inputs. For the other two variants we used CNN-type filters to process the state history before passing it to the fully connected layers. The first variant uses filters that are being shared by each state (classical CNN), whereas the second variant uses filters that are state dependent, i.e. they are not shared among the states (see Fig. 2, Fig. 3). We decided to use CNN filters instead of an RNN/LSTM architecture, because the time dependencies are expected to be short term only due to the highly nonlinear

dynamics of the system. Additionally, by analyzing the filter weights after training findings about the system can be derived.

The inputs to all networks are standardized using $\omega_{max} = 200\,\mathrm{rad\,s^{-1}}$ for the angular velocities, $v_{max} = 5\,\mathrm{m\,s^{-1}}$ for the translational velocities and π for the angles. For all layers we used ReLU as the activation function. The weights of the filters were initialized uniformly with $w_i = 1/(\text{length\_state\_history} * \text{number\_filters})$ to allow stable training.

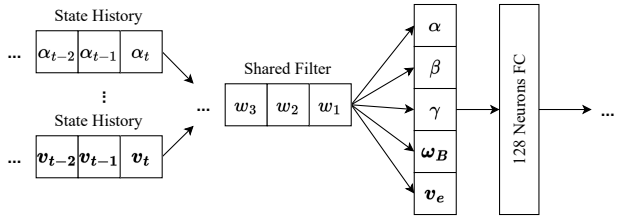

Fig. 2. Architecture that incorporates a history for each state and processes them with filters that are shared among all states (classical CNN.)

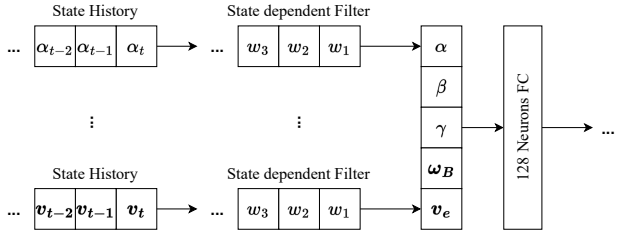

Fig. 3. Architecture that incorporates a history for each state and processes them with filters that dependent on the state, i.e. they are not shared among the states.

II. Results

Performance metrics are the average achieved reward, the average tracking error per time step and the number of early terminated environments on the test set. The best performance was achieved by using state dependent filters (Tab. III). We explain the superiority of using state dependent filters with the different dynamic properties of the drone in each direction, leading to different time dependencies. They arise, for instance, due to gravity and the geometrical asymmetries of the drone.

Furthermore, we observe that less environments terminate prematurely when using controllers that incorporate information about the state history. This indicates that even difficult trajectories can be followed more stable.

We also created a unitless metric to measure the energy consumed by all actuators during the test scenarios. For this, all actuator actions are summed up and divided by the total number of simulation time steps. We found that controllers using the state history were not only more accurate, but also consumed less total energy compared to baseline: 1.701 (4*9 state dependent filters) vs 1.965 (Baseline). This indicates that the controllers are more stable and efficient overall. We assume that the increased efficiency comes from better short-term planning due to the time awareness of the dynamics. We also observed the the oscillation in the actuators was reduced.

TABLE III
Performance comparison of network architectures

Network	Reward	Tracking error (m/s)	# early term. envs	# trainable params
FNN (Baseline)	2696	1.214	4	36.7k
FNN with past states	2764	0.862	2	80.5k
Shared filters (classical CNN)				
1 Filter → FNN	2664	1.231	2	36.7k
3 Filters → FNN	2531	1.586	4	41.1k
5 Filters → FNN	2691	1.158	3	46.1k
9 Filters → FNN	2714	1.138	2	55.5k
18 Filters → FNN	2820	0.899	1	76.6k
State dependent filters				
2*9 Filters → FNN	2924	0.62	0	39.1k
4*9 Filters → FNN	2951	0.56	0	45.1k

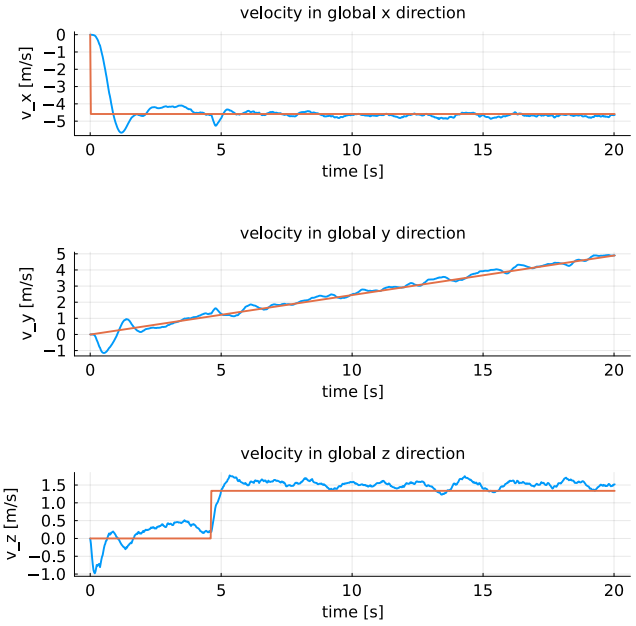

Fig. 4. An example target and true velocity trajectory of the drone in simulation using the network with two independent filters per state (2*9). Blue line: drone velocity, orange line: target velocity. Average velocity error per time step: $0.56\,\mathrm{m\,s^{-1}}$.

Further, we analyzed the weights of the filters after training. First, we trained a network using the last 100 time steps (2.5 s) as input. In Fig. 5 we observe that states close to the current state have higher weights and are thus more important than states further in the past. This confirms that mainly short term time dependencies are relevant. As a conclusion, we decided to only use the last 20 time steps (0.5 s) as inputs for all experiments. One such learned filter is shown in Fig. 6.

Additional experiments are summarized in Table IV. We found that using the wind speed as input did not improve

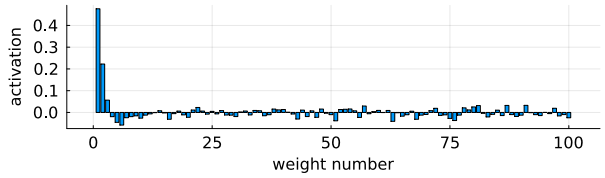

Fig. 5. Exemplary trained weights for a filter with a state history of 100 time steps (2.5 s). States further in the past are less relevant and have lower weights than states closer to the current state.

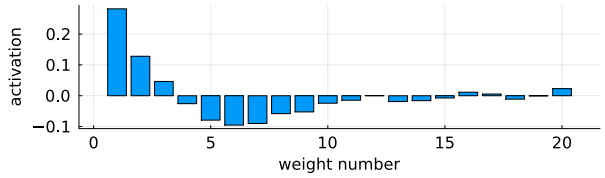

Fig. 6. Exemplary trained weights for a filter with a state history of 20 time steps (0.5 s). The trained weights don't appear to be random but follow a recognizable shape and might thus be physical meaningful.

the tracking performance. We argue that wind speed we used was not high enough to influence the drone performance significantly. For further experiments the wind speeds should be increased. Furthermore, we have tried making use of a two step training with a reduced stay alive reward $r_{sa} = 0.5$ for the second training. This method allowed us to reduce the tracking error. However, the number of early terminated environments increased, which indicates more unstable flights. The reason for the increase in instability could not be determined and we do not suggest to use a two step training in a similar manner to ours.

TABLE IV
ADDITIONAL EXPERIMENTS

Network	Reward	Tracking error (m/s)	# early term. envs
18 filters → FNN (New baseline)	2820	0.899	1
Add wind speed in [x,y,z] as input	2816	0.879	2
Two-step training	-	0.667	13

III. SUMMARY AND OUTLOOK

Within this project we evaluated different neural network architectures for the RL-based velocity control of a VTOL drone. Significant performance improvements were achieved by using networks that process the state history of the drone with state dependent filters. These networks additionally lead to more efficient control performance and stability in the flights, making them superior to standard FNNs. We observed that the dependency on state history information is only short term and thus used the last 0.5 s of each state. We conclude that the CNN-type filters we used are sufficient to model the time dependencies and using RNNs/LSTMs is not required.

Whether the learned filter weights from our simulations are transferable to the drone in the real world remains to be seen. Thus, further work should focus on transferring the results onto a real system.

Lastly, it would be interesting to see whether the results we obtained with our work are applicable to other control problems as well. For this, it might be interesting to implement controllers using information about the state history for other systems.

APPENDIX

TABLE V
DISTURBANCE AND NOISE PARAMETERS

parameter	value (SI units)
$v_{wind,max}$	5.0
σ_v	0.03
σ_ω	7e-3
σ_{rot}	2e-5
σ_{thrust}	0.02
σ_{flaps}	0.04

$v_{wind,max}$ - maximum mean velocity of the wind gusts. Sensor and actuator noise is sampled from a zero mean Gaussian distribution with the standard deviations σ_v, σ_ω, σ_{rot}, σ_{thrust} and σ_{flaps}.

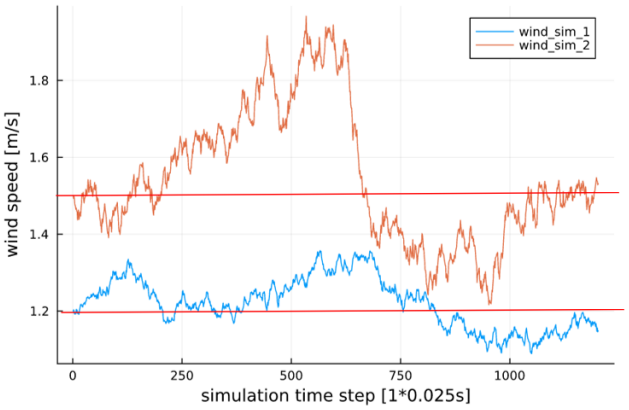

Fig. 7. Exemplary randomly generated wind disturbances with mean wind speed $1.5\,\mathrm{m\,s^{-1}}$ and $1.2\,\mathrm{m\,s^{-1}}$.

REFERENCES

[1] Xu, Jie and Du, Tao and Foshey, Michael and Li, Beichen and Zhu, Bo and Schulz, Adriana and Matusik, Wojciech, "Learning to fly: computational controller design for hybrid UAVs with reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.

[2] G. Holmes, M. Chowdhury, A. McKinnis, and S. Keshmiri, "Deep Learning Model-Agnostic Controller for VTOL Class UAS," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1520–1529, 2022.

[3] M. Chowdhury and S. Keshmiri, "Design and Flight Test Validation of an AI-Based Longitudinal Flight Controller for Fixed-wing UASs," in *2022 IEEE Aerospace Conference (AERO)*, pp. 1–12, 2022.