nature machine intelligence

Supplementary information

https://doi.org/10.1038/s42256-025-01048-0

Learning vision-based agile flight via differentiable physics

In the format provided by the authors and unedited

Table of Contents

Note 1 Calibration and system identification

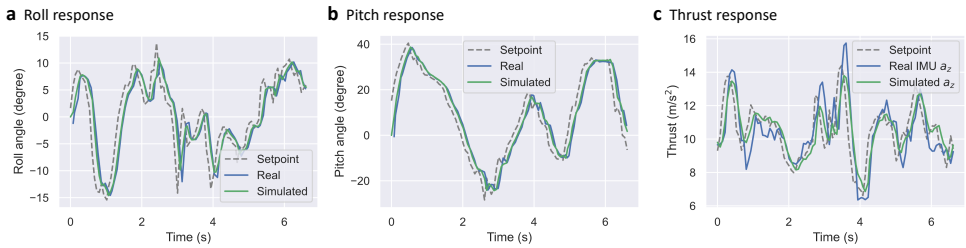
For the effective training of our quadrotor navigation framework, it's pivotal that our simulations mimic real world flight dynamics accurately. The primary goal of calibration is to identify the parameters of our differentiable physics simulation, ensuring that simulated dynamics mirror real world quadrotor responses. Calibration involves adjusting for control latency and air drag.

1.1 Control latency identification

Our simulation considers a fixed control latency and an exponential moving average for modeling the flight controller response. We fix the proportional attitude controller with an attitude control gain of 13. We collect 6 seconds of real world flight data and replay it with our latency model in simulation. We perform calibration experiments on roll and pitch response to measure the latency parameters λ, τ, yielding an estimate of $\lambda = 12, \tau = 1/15$. Supplementary Fig. 1**a**,**b** shows the simulated roll and pitch responses closely match the real world responses.

Measuring thrust in the real world is more challenging because the acceleration measurements are mixed with gravity, air drag, and non-linear thrust model. Nevertheless, we have verified that the delay parameters for roll and pitch are applicable for thrust as well. Results from Supplementary Fig. 1**c** shows that our simulation aligns closely with the real world thrust delay.

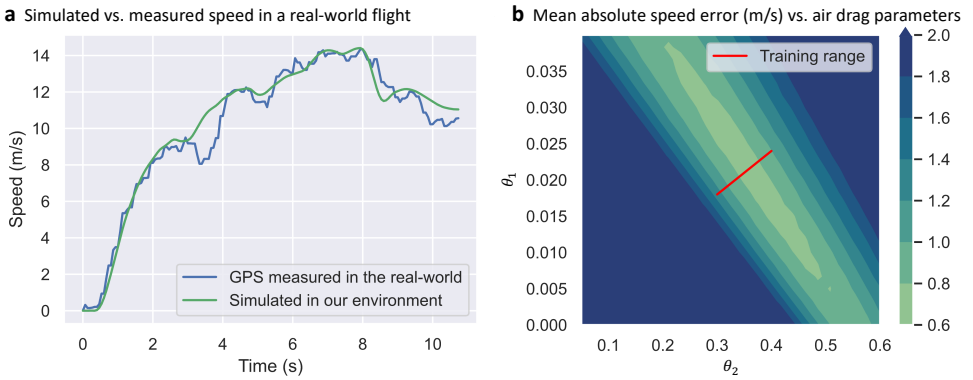
We have verified the consistency of this latency across several platforms, provided the attitude control gain remains uniform: a) our platform shown in Fig. 2**a** using BetaFlight; b) a larger 0.6 kg drone with 4-inch propellers also using BetaFlight; c) a 1.2 kg drone with 5-inch propellers running a PX4 controller; d) the AirSim [1] simulator; e) the Flightmare [2] simulator.



Supplementary Fig. 1: **Flight control response in simulation.** The simulation closely mirrors our real drone, demonstrating the accuracy of our calibration.

1.2 Air drag identification

In low-speed navigation, air drag is frequently overlooked. However, for high-speed sensorimotor flights, it's a critical factor. At speeds exceeding 9 m/s, our 3-inch quadrotor



Supplementary Fig. 2: **The simulated flight velocity vs. our real drone.** (**a**) We record actions from real world flights and replay it in simulation. The relatively small difference between the simulation and the real drone shows the fidelity of our air drag simulation. (**b**) We conduct a grid search for air drag parameters that minimize absolute speed error between simulation and real world flights.

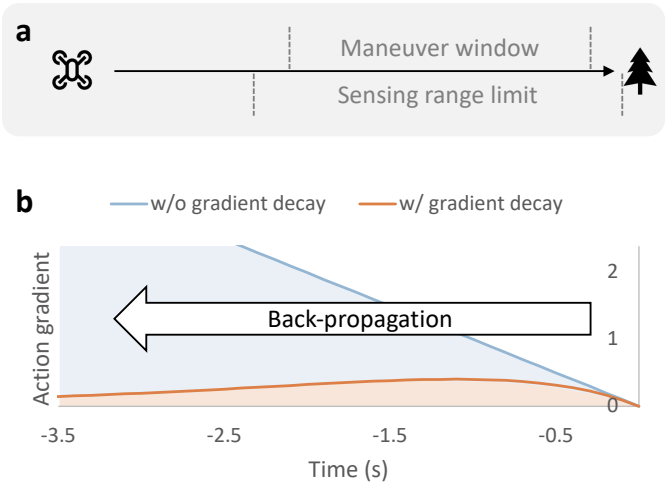
(refer to Fig. 2**a**) with propeller guards encounters considerable air resistance, measuring over $5\,\mathrm{m/s^2}$. With fixed agent actions, air drag largely dictates the flight speed. For localization-free flights, air drag plays a primary role in the implicit estimation of velocity.

Air drag varies depending on the platform. To calibrate it, we've employed a straightforward method. We record 1 minute of outdoor flight data, including the acceleration control commands and actual flight speeds, and replay these in simulation using the drag model $a_{drag} = -\theta_1\|v\|v - \theta_2 v$, where θ_1 and θ_2 are undetermined parameters. Then, we conduct a grid search for the parameters and compare the simulated speed to the GPS recorded speed (as shown in Supplementary Fig. 2**a**). The parameter set that yields the smallest speed discrepancy is adopted as the drag coefficients. To enhance model robustness, we train using a diverse set of drag coefficients, centered around the calibrated values (as depicted in Supplementary Fig. 2**b**).

Note 2 Temporal gradient decay

Temporal gradient decay is an essential technique to ensure stability and efficiency in training, especially for long-range sequences. This method manages the accumulation of gradients backpropagated through time, ensuring that the significance of distant obstacles vanishes over time. We first highlight the challenge posed by gradient accumulation (refer to Extended Data Fig. 2**a**,**b**) and subsequently demonstrate how temporal gradient decay addresses the issue of exploding gradients (see Extended Data Fig. 2**c**). Ultimately, we illustrate how the decayed gradient teaches the agents to maneuver at the optimal moment.

Extended Data Fig. 2 shows the graphical model of simulation spanning 4 timesteps, with the objective function being evaluated based on the position at t=4.



Supplementary Fig. 3: **Temporal gradient decay aligns the supervisory signals with the sensing range.** (**a**) A scenario where the agent approaches an obstacle. Graph (**b**) Gradient decay aligns training signals within the sensing range.

For simplicity, we take $\Delta_t = 1$ without loss of generality. Extended Data Fig. 2**b** shows the backward computation. Here, the positional gradient, denoted as g, is backpropagated through the entire computation graph. The numerical integration structure further leads to a growing gradient signal that ultimately accumulates to the model parameters. In contrast, when gradient decay is applied during temporal backpropagation, the gradients affecting the parameters result from the product of a linearly increasing term and an exponentially decaying term.

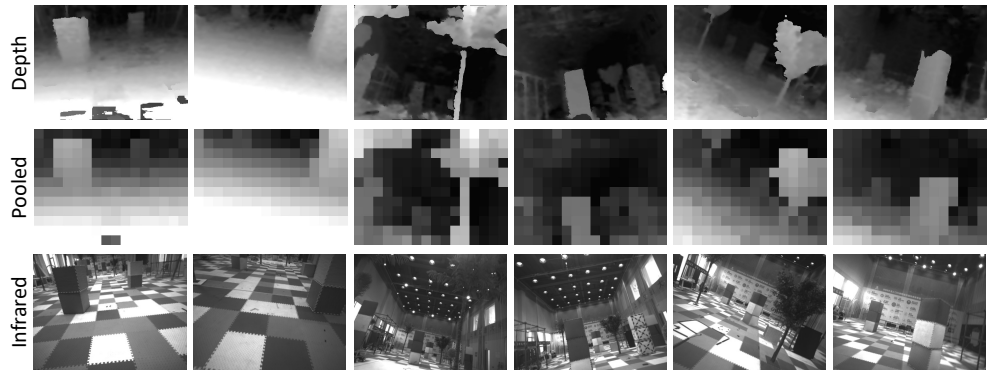
Supplementary Fig. 3 shows the advantage of temporal gradient decay in aligning supervisory signals with the quadrotor's sensing range. The blue line in Supplementary Fig. 3, representing the gradient without temporal gradient decay, displays an ever-increasing gradient. This misleads the agent with the impractical task of avoiding obstacles beyond its sensing range. The gradient, after the application of decay, initially rises and then diminishes to zero as it's backpropagated over time. This dynamic achieves a balance, addressing the inherent conflict that earlier maneuvers provide more significant benefits for obstacle avoidance, yet distant obstacles are often more challenging to perceive. Thus, the agent is directed to concentrate more on proximate obstacles that are immediate and more perceptible.

By prioritizing immediate and perceptible obstacles, the training process can converge faster as the training objective becomes more feasible, and the agent receives more relevant feedback. This method ultimately foresters a more robust navigation system.

Note 3 Depth Image Pre-Processing

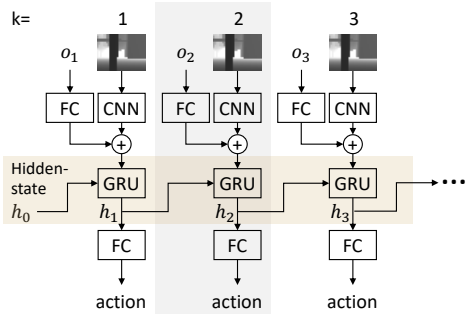
For real world deployment, the depth map from the D435 camera is first inverted, using $(\cdot)^{-1}$, and standardized. We then downsample the depth image to a resolution of 32×24 using nearest sampling. Following this, we apply a max-pooling with a 2×2 kernel. This operation captures the nearest point within each 2×2 grid. This results in a pooled depth image of size 16×12, which is used as input for the network. The original D435 depth image is captured in 16:9 mode and then cropped to 4:3 to avoid invalid pixels along the sides.

Supplementary Fig. 4 shows a sequence of inverted D435 depth images and the corresponding pooled results during a sample flight. This figure also includes the infrared grayscale image of the environment for reference.



Supplementary Fig. 4: Depth image sequence during a sample flight, along with the pooled depth images and infrared grayscale images. See Supplementary Video 8 for the full sequence.

Supplementary Figures



Supplementary Fig. 5: Temporal unfolded data flow of our recurrent neural network architecture. The gated recurrent unit (GRU) [3] maintains and updates a hidden state (h_t) over time as memory.

Supplementary Tables

Method	Model	Inference Time (ms)
Agile [4]	MobileNet v3 [5]	15.96
Bhattacharya et al. [6]	Custom ViT+LSTM	8.97
Ours	Custom CNN+GRU	0.44
Ours (onboard)	Custom CNN+GRU	3.74

Supplementary Table 1: Neural network inference times on an AMD Ryzen 2700X CPU. Onboard data is tested on an Allwinner H616 CPU. Times are averaged over 1000 samples.

Training Parameter	**Value**
optimizer	AdamW
learning rate	0.001
learning rate schedule	cosine decay
weight decay	0.01
batch size	64
number of timesteps N	150
Δt	1/15 s
gradient decay rate α	0.92
training iterations	50000

Supplementary Table 2: The hyperparameters used for training. We add a small perturbation on the simulation time step Δt during training.

Training Parameter	Value
optimizer	AdamW
weight decay	0.01
discount factor	0.99
GAE λ	0.95
number of timesteps N	150
Δt	1/15 s
clip range	0.2
entropy coefficient	0.001
parallel environments	256
training iterations	12,500
minibatch size	38,400
mini epochs	5

Supplementary Table 3: Hyperparameters for the Proximal Policy Optimization (PPO) Algorithm [7].

References

[1] Shah, S., Dey, D., Lovett, C., Kapoor, A.: AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In: Field and Service Robotics (2017). https://arxiv.org/abs/1705.05065

[2] Song, Y., Naji, S., Kaufmann, E., Loquercio, A., Scaramuzza, D.: Flightmare: A flexible quadrotor simulator. In: Conference on Robot Learning, pp. 1147–1157 (2021). PMLR

[3] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)

[4] Loquercio, A., Kaufmann, E., Ranftl, R., Müller, M., Koltun, V., Scaramuzza, D.: Learning high-speed flight in the wild. Science Robotics **6**(59), 5810 (2021). Publisher: American Association for the Advancement of Science

[5] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V., Adam, H.: Searching for MobileNetV3. arXiv preprint arXiv:1905.02244 (2019)

[6] Bhattacharya, A., Rao, N., Parikh, D., Kunapuli, P., Wu, Y., Tao, Y., Matni, N., Kumar, V.: Vision transformers for end-to-end vision-based quadrotor obstacle avoidance. arXiv preprint arXiv:2405.10391 (2024)

[7] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)