APPENDIX

APPENDIX I
HYPERPARAMETERS AND TRAINING DETAILS

We tested our method on three dynamic systems that are visualized in Figure 4: the CartPole balancing task, trajectory tracking with a quadrotor and point-goal navigation with a fixed-wing aircraft. Our framework is implemented in Pytorch, enabling the use of autograd for backpropagation through time. All source code is attached, and the parameters can be found as part of the source code in the folder `configs`. Example videos for all three applications are also attached in the supplementary material. In the following we provide details on training and evaluation.

A. CartPole

In the CartPole problem, a pole should be balanced on a cart by pushing the cart left and right, while maintaining low cart velocity. The time step is again set to $0.05s$. The state of the system can be described by position x and velocity $\dot{x}$ of the cart, as well as angle α and angular velocity $\dot{\alpha}$ of the pole. The "reference trajectory" is defined only by the target state, which is the upright position of the pole, $\alpha = 0$, $\dot{\alpha} = 0$ and $\dot{x} = 0$.

Policy network. Since the reference state is constant, it is sufficient for the policy to observe the state at each time step. Here, we input the raw state without normalization. It is passed through a five-layer MLP with 32, 64, 64, 32 and 10 neurons respectively. All layers including the output layer use tanh activation, in order to scale the actions to values between -1 (corresponding to $30N$ force to the left) and 1 (force of $30N$ pushing the cart to the right). As for the quadrotor and fixed-wing drone, the next 10 (1-dimensional) actions are predicted.

Loss function. To compute the loss with respect to a reference trajectory, we interpolate between the current state and the target state. Note that the intermediate states are often infeasible to reach; for example an increase of velocity might be required to reduce the pole angle. The interpolation where both velocity and angle decrease is thus not realistic. As before, a weighted MSE between the reference states and the actual states is computed, where the angle difference is weighted with a factor of 10, the cart velocity with factor 3, and the angular velocity with factor 1. The loss is minimized with SGD optimizer with learning rate 10^{-7}.

B. Quadrotor

Our implementation of a quadrotor environment is loosely based on the implementation provided at https://github.com/ngc92/quadgym (MIT license). However, our model of the quadrotor is a Pytorch implementation of the equations in the Flightmare simulator [38]. We also provide an interface to test our models in Flightmare, and a video of our model controlling a quadrotor in Flightmare is attached in the supplementary material. In all our experiments, we set the time between the discrete time steps to $0.1s$.
Furthermore, for training and testing we generate 10000 random polynomials of $10s$ length, where all trajectories are guaranteed to be feasible to track with the used platform. From this dataset, 1000 trajectories are left out as a test set to ensure that the policy generalizes to any given reference. The maximum desired velocity on such a reference trajectory is usually around $3 - 5m/s$, whereas the average velocity is $1 - 2m/s$.

Policy network. At each time step, the current state and the next reference states are given as input to the network. As the current state, we input the velocity in the world frame and in the body frame, the first two columns of the rotation matrix to describe the attitude (as recommended in [49]), and the angular velocity. For the reference, we input the next 10 desired positions *relative* to the current drone position, as well as the next 10 desired velocities (in the world frame). The state is first passed through a linear layer with 64 neurons (tanh activation), while the reference is processed with a 1D convolutional layer (20 filters, kernel size 3) to extract time-series features of the reference. The outputs are concatenated and fed through three layers of 64 neurons each with tanh activation. The output layer has 40 neurons to output 10 four-dimensional actions. Here, an action corresponds to the total thrust T and the desired body rates ω_{des} to be applied to the system. The network outputs are first normalized with a sigmoid activation and then rescaled to output a thrust between $2.21N$ and $17.31N$ (such that an output of 0.5 corresponds to $9.81N$) and body rates between -0.5 and 0.5.

Loss function. The loss (equation 3) is a weighted MSE between the reached states and the reference states. The weights are aligned to the ones used for the optimization-based MPC, namely a weight of 10 for the position loss, 1 for the velocity, 5 to regularize the predicted thrust command and 0.1 for the predicted body rates as well as the actualy angular velocity. Formally, these weights yields the following loss:

$$L = \sum_{k=1}^{10} 10 \cdot (x_{t+k,\pi} - x_{t+k,\nu})^2 + (\dot{x}_{t+k,\pi} - \dot{x}_{t+k,\nu})^2 + 5 \cdot (T_k - 0.5)^2 + 0.1 \cdot (\omega_{k,des} - 0.5)^2 + 0.1 \cdot \omega_{t+k} \quad (6)$$

where x_t is the position at time step t, $\dot{x}_t$ is the velocity, and the subscript π indicates the states reached with the policy while the subscript ν denotes the positions and velocities of the reference. T and ω_{des} correspond to the action after sigmoid activation but before rescaling (such that values lie between 0 and 1), and ω_t is the actual angular velocity of the system at each state. The loss is minimized with the Pytorch SGD optimizer with learning rate 10^{-5} and momentum 0.9.

Curriculum learning. As explained in section III, we use a curriculum learning strategy with a threshold τ_{div} on the allowed divergence from the reference trajectory. We set $\tau_{div} = 0.1m$ initially and increase it by $0.05m$ every 5 epochs, until reaching $2m$. Additionally, we start by training on slower reference trajectories (half the speed). Once the quadrotor is stable and tracks the full reference without hitting

τ_{div}, the speed is increased to 75% of the desired speed and τ_{div} is reset to $0.1m$. This is repeated to train at the full speed in the third iteration.

C. Fixed-wing drone

We implemented a realistic model of the dynamics based on the equations and parameters described in [39] and [50]. In our discrete-time formulation, we set $\delta t = 0.05s$.

Reference trajectory. In contrast to the reference trajectories for the quadrotor, the reference for the fixed-wing aircraft is only given implicitly with the target position. As an approximate reference, we train the policy to follow the linear trajectory towards the target point. In the following, the term "linear reference" will be used to refer to the straight line from the current position to the target point. At each time step, we compute the next 10 desired states as the positions on the linear reference while assuming constant velocity.

Policy network. Similar as for the quadrotor, the state and reference are pre-processed before being input to the network policy. The state is normalized by subtracting the mean and dividing by the standard deviation per variable in the state (mean and standard deviation are computed over a dataset of states encountered with a random policy). This normalized state together with the relative position of the 10-th desired state on the reference are passed to the policy network as inputs. Using all 10 reference states as input is redundant since they are only equally-distant positions on a line.
The state and the reference are each separately fed through a linear layer with 64 neurons and then concatenated. The feed-forward network then corresponds to the one used for the quadrotor training (three further layers with 64 neurons each and an output layer with 40 neurons). The output actions are also normalized with sigmoid activations and then scaled to represent thrust $T \in [0, 7]N$, elevator angle $a_1 \in [-20, 20]^\circ$, aileron angle $a_2 \in [-2.5, 2.5]^\circ$ and rudder angle $a_3 \in [-20, 20]^\circ$.

Loss function. As for the quadrotor, we align the loss function to the cost function of the online optimization model predictive control in the MPC baseline. The MSE between the reached positions and the target positions on the linear reference is minimized while regularizing the action, formally

$$L = \sum_{k=1}^{10} 10 \cdot (x_{t+k,\pi} - x_{t+k,\nu})^2 + 0.1 \cdot ((a_{k,1} - 0.5)^2 + (a_{k,2} - 0.5)^2 + (a_{k,3} - 0.5)^2) \ , \quad (7)$$

where t is the current time step and $x_{t+k,\pi}$ is the position of the aircraft after executing the k-th action $(T_k, a_{k,1}, a_{k,2}, a_{k,3})$, and $x_{t+k,\nu}$ is the corresponding reference state. The loss is minimized with an SGD optimizer with learning rate of 10^{-4} and momentum of 0.9.

Finally, the curriculum is initialized to allow a divergence of $4m$ from the linear reference and increased by $0.5m$ every epoch until reaching $20m$. Note that in contrast to the quadrotor, the model converges in few epochs.