JATIS Journal of Astronomical Telescopes, Instruments, and Systems

RESEARCH PAPER

Laboratory experiments of model-based reinforcement learning for adaptive optics control

Jalo Nousiainen,[a,b,*] Byron Engler,[c] Markus Kasper,[c] Chang Rajani,[d] Tapio Helin,[a] Cédric T. Heritier,[c,e,f] Sascha P. Quanz,[g] and Adrian M. Glauser[g]

[a]LUT University, Lappeenranta, Finland
[b]Aalto University, Department of Mathematics and Systems Analysis, Aalto, Finland
[c]European Southern Observatory, Garching bei München, Germany
[d]University of Helsinki, Helsinki, Finland
[e]DOTA, ONERA, Salon cedex Air, France
[f]Aix Marseille University, CNRS, CNES, LAM, Marseille, France
[g]ETH Zurich, Institute for Particle Physics and Astrophysics, Zurich, Switzerland

ABSTRACT. Direct imaging of Earth-like exoplanets is one of the most prominent scientific drivers of the next generation of ground-based telescopes. Typically, Earth-like exoplanets are located at small angular separations from their host stars, making their detection difficult. Consequently, the adaptive optics (AO) system's control algorithm must be carefully designed to distinguish the exoplanet from the residual light produced by the host star. A promising avenue of research to improve AO control builds on data-driven control methods, such as reinforcement learning (RL). RL is an active branch of the machine learning research field, where control of a system is learned through interaction with the environment. Thus, RL can be seen as an automated approach to AO control, where its usage is entirely a turnkey operation. In particular, model-based RL has been shown to cope with temporal and misregistration errors. Similarly, it has been demonstrated to adapt to nonlinear wavefront sensing while being efficient in training and execution. In this work, we implement and adapt an RL method called policy optimization for AO (PO4AO) to the GPU-based high-order adaptive optics testbench (GHOST) test bench at ESO headquarters, where we demonstrate a strong performance of the method in a laboratory environment. Our implementation allows the training to be performed parallel to inference, which is crucial for on-sky operation. In particular, we study the predictive and self-calibrating aspects of the method. The new implementation on GHOST running PyTorch introduces only around 700 μs of in addition to hardware, pipeline, and Python interface latency. We open-source well-documented code for the implementation and specify the requirements for the RTC pipeline. We also discuss the important hyperparameters of the method and how they affect the method. Further, the paper discusses the source of the latency and the possible paths for a lower latency implementation.

 [DOI: 10.1117/1.JATIS.10.1.019001]

Keywords: high contrast imaging; adaptive optics; reinforcement learning; system identification

Paper 23104G received Aug. 31, 2023; revised Dec. 21, 2023; accepted Dec. 28, 2023; published Feb. 22, 2024.

1 Introduction

High contrast imaging (HCI) utilizes a combination of extreme adaptive optics (XAO) and coronagraphy to generate images of faint sources near bright point sources, such as exoplanets near

*Address all correspondence to Jalo Nousiainen, jalo.nousiainen@aalto.fi

their host stars. Direct imaging of exoplanets has been largely limited to only a few dozen very young and luminous giant exoplanets using existing HCI instruments, as documented in studies such as Refs. 1–3. However, a greater number of planets could be directly imaged by enhancing the sensitivity in the vicinity of the host star, with the performance of the XAO system being the primary limiting factor in achieving such sensitivity.

In HCI, when imaging in close proximity to the star, the main performance limitations of a well-tuned adaptive optics (AO) system controlled with the common integrator controller are photon noise and temporal error, as noted in Ref. 4. The temporal delay error of AO systems controlled by standard methods arises from the integration of wavefront sensor detector data, detector readout, computation of the correction signal, and its application to the deformable mirror (DM). This delay amounts to at least two AO system operating cycles at the maximum camera framerate, where readout takes one entire frame, during which atmospheric turbulence has evolved and no longer matches the DM correction precisely.

There are two ways to mitigate the adverse effect of the temporal delay error for HCI: by increasing the operating frequency of the AO system or by implementing predictive control. The acceleration of the AO system can be accomplished, for example, by adding a second stage downstream from a classical first-stage AO system.[5] This second-stage system solely observes the residual from the first-stage AO system and can operate independently from the first-stage, employing DMs that can handle fast AO loops. One such example is the upgrade of SPHERE, which is referred to as SPHERE+,[6] which is expected to provide a considerable enhancement in raw point-spread function (PSF) contrast close to the star.

The other (not mutually exclusive) approach is to use a predictive control algorithm. A big part of the turbulence is presumably in frozen flow considering the millisecond timescale of AO control, and hence, a significant fraction of wavefront disturbances can be predicted.[7] Moreover, if the predictive control algorithm is fast enough, both strategies can be combined by operating the faster second stage with predictive control.

Besides the performance limitations induced by photon noise and temporal error, AO can suffer from dynamic modeling errors, such as misregistration,[8] optical gain effect for the Pyramid wavefront sensor (WFS).[9,10] Coping with these limitations usually requires external tuning and recalibration of a possible predictive control algorithm.

This paper presents a laboratory demonstration of a data-driven predictive control algorithm called the policy optimizations for AO (PO4AO)[11] implemented on a second stage AO system following a first stage running a classical integrator control. One of the main advantages of implementing fully data-driven control, such as PO4AO, is that it continuously learns a system model from the data rather than using a static calibration or synthweifhetic model. Consequently, it is less affected by pseudo-open-loop reconstruction errors, such as misregistration or the optical gain effect, as discussed in Refs. 11–13. Our contributions are two-fold: first, we thoroughly test the performance and robustness of PO4AO in a laboratory setup, detailed in Sec. 7.1, under different conditions. Second, we open-source Python-based implementation of the method that can be implemented in any AO system that runs Python-based controllers and has GPUs. We also discuss how the method can be tuned and further developed for different AO systems. The codes used in this paper are available on our GitHub repository [https://github.com/jnousi/PO4AO.git].

2 Related Work

PO4AO addresses the predictive control and reconstruction in the XAO control loop as a single reinforcement learning (RL) problem; hence, PO4AO is related to many aspects of XAO control, such as predictive control, optimal gain compensation, misregistration identification, reconstruction algorithms, and vibration canceling.

Remarkable progress has been achieved with various approaches to tackle the XAO control problem. These methods include the Kalman filter-based linear controllers,[14–20] sometimes combined with machine learning for system identification.[21] These methods rely on linear models for wavefront sensing and temporal evolution to obtain a state estimation of the system. Other methods focus on correcting temporal error and vary from spatio-temporal linear filters to filters operating on single modes, such as Fourier or Zernike modes.[22–27] The predictive filters are

obtained either from modeling or utilizing data analysis/machine learning. Some methods have also been tested on-sky (see, e.g., Refs. 21 and 28).

Moreover, machine learning methods utilizing neural networks (NNs) for predictive control have been studied in Refs. 29–34, where NNs show a lot potential, especially for AO systems with high number of degrees of freedom (DoF), and in noisy conditions. Lately, NNs, and the more modern deep NNs, have also been used for the wavefront reconstruction step (see, e.g., Refs. 35–38). The results indicate that NN reconstruction is less sensitive to non-linearity and increases the operational range of the pyramid WFS.

Finally, different NN-based RL approaches have been studied during the last years; see Refs. 39–41. PO4AO differs from other RL methods in AO literature by using so-called model-based RL instead of model-free RL (for a discussion on the difference between these methods, see Ref. 11). For interested readers, Fowler and Landman[42] provide a more thorough review of machine learning methods for wavefront control and phase prediction.

3 Classical Adaptive Optics Control and Baseline Controller

An AO system is commonly controlled with a linear integrator controller, referred to as the integrator. We consider it our reference method against the PO4AO as it is still widely used in AO. Integrator control in AO usually relies on the so-called interaction matrix mapping DM commands to WFS measurements

$$\Delta \boldsymbol{w}^t = D\boldsymbol{v}^t + \xi_t, \tag{1}$$

where $\Delta \boldsymbol{w}^t = (\delta w_1^t, \delta w_2^t, \ldots, \delta w_n^t)$ is the WFS data, $\boldsymbol{v}^t$ the DM commands and D is the interaction matrix, and ξ_t is the measurement noise typically composed of photon and detector noise. Once the interaction matrix is estimated, the inverse problem, i.e., reconstruction $\boldsymbol{v}^t$ given $\Delta \boldsymbol{w}^t$, needs to be considered. As D is generally not invertible, some regularization approach is needed. Here, we restrict ourselves to linear methods described by a reconstruction matrix C mapping WFS measurements to DM commands. As our regularization method, we project D to a smaller dimensional subspace spanned by the Karhunen–Loéve (KL) modal basis.[43] Each KL mode in the basis has a representation in terms of actuator voltages. This relation is fully determined by a transformation matrix B_m mapping DM actuator voltages to m first modal coefficients. The regularized reconstruction matrix is now defined by the Moore–Penrose pseudo-inverse

$$C_m = (DP_m)^\dagger, \tag{2}$$

where $P_m = B_m^\dagger B_m$ is a projection map to the KL basis. The number of modes m defines stability at the cost of resolution; smaller m results in lower noise amplification while producing a reconstruction with fewer modal basis functions (less detailed reconstruction). An optimal m balances the error produced by these two effects.

We use the leaky integrator as a baseline AO controller to which PO4AO is compared. At a given time step t, the WFS measures the residual wavefront. The leaky integrator then obtains the new control voltages $\tilde{\boldsymbol{v}}_t$ from

$$\tilde{\boldsymbol{v}}_t = l\tilde{\boldsymbol{v}}_{t-1} + gC_m \Delta \boldsymbol{w}_t, \tag{3}$$

where g is the integrator gain, typically fixed below a value of about 0.5 for a two-step delay system.[44] The DM saturation can cause a build-up of modes outside the control radius. Hence, introducing a leakage l typically chosen near one, e.g., 0.99, in the DM commands commonly used to remove those unseen modes and increase robustness.

4 Adaptive Optics as a Markov Decision Process

In this paper, we model AO control loop as a Markov decision process (MDP),[45] which is the de facto mathematical framework for sequential decision problems in RL. In AO control, instead of the full state of the system (exact DM shape, the full atmosphere profile, etc.), we only observe WFS data $\Delta \boldsymbol{w}_t$ that represents only a partial information of the whole system.[12] These kinds of processes are referred to as partially observed MDPs in the RL literature, and, in theory, optimal decisions (control) should consider all the past measurements observed (from the beginning of

operation). However, we expand the state space to include a history of WFS measurement and DM control voltages to guarantee approximately Markovian statistics and treat the process as an ordinary MDP, where optimal decisions can be taken directly from the previous state formulation (in our case a concatenation of past measurement and actions).[11,12,39,41]

Let us first identify an action $\boldsymbol{a}_t$ as the applied differential voltage satisfying

$$\tilde{\boldsymbol{v}}_t = l\tilde{\boldsymbol{v}}_{t-1} + \boldsymbol{a}_t, \tag{4}$$

where l is the leakage factor.

We now define the state $\boldsymbol{s}_t$ of the MDP as

$$\boldsymbol{s}_t = (\boldsymbol{o}_t, \boldsymbol{o}_{t-1}, \ldots, \boldsymbol{o}_{t-k}, \boldsymbol{a}_{t-1}, \boldsymbol{a}_{t-2}, \ldots, \boldsymbol{a}_{t-k}), \tag{5}$$

where $\boldsymbol{o}_t = C_m \Delta \boldsymbol{w}_t$ is the wavefront measurement projected to DM space and k the number of history frames used in state formulation. Hence, we assume that there exist Markovian transition dynamics $p(\boldsymbol{s}_t, \boldsymbol{a}_t) = \boldsymbol{s}_{t+1}$, where the only new element of $\boldsymbol{s}_{t+1}$ is the next observation $\boldsymbol{o}_{t+1}$. This transition model contains information on the time delay (i.e., which action affects which observation), misregistration and non-linearity errors (how actions are related to measurements), and atmospheric turbulence (how past information can be interpolated in the future). Further, in the following formulation of the control algorithm,[5] the initial reconstruction matrix C_m serves as preprocessing to observation and does not connect measurement to action directly (like in integrator control).

As the reward function of the MDP, we consider the negative Euclidian norm of the residual wavefront observed through the WFS with a regularization term that favors small actions, i.e.

$$\hat{r}_\omega(\boldsymbol{s}_t, \boldsymbol{s}_{t+1}, \boldsymbol{a}_t) = -\|\boldsymbol{o}_{t+1}\|^2 - \alpha\|\boldsymbol{a}_t\|^2, \tag{6}$$

where $\boldsymbol{o}_{t+1}$ is the first element of $\boldsymbol{s}_{t+1}$. The addition of the regularization term $\alpha\|\boldsymbol{a}_t\|^2$ effectively regularizes the control algorithm, making it less prone to saturation and oscillation, especially early in the training procedure.

5 Model-Based Policy Optimization

The key idea of PO4AO (for details, see Nousiainen et al. [11]) is to learn a non-linear control law that maps past telemetry to new DM commands from data collected from the AO loop and maximizes the reward. In RL terminology, this control law is referred to as the policy and will be formulated as a mapping from the current state $\boldsymbol{s}_t$ to the next action $\boldsymbol{a}_t$. Hence, the policy combines the reconstruction and control steps in AO (e.g., a least-squares modal reconstruction followed by integrator control).

In this work, the policy is constructed as a NN, and its parameters are derived indirectly via model-based policy optimization. More precisely, the method collects data to learn a dynamics model that is also represented by an NN and can be used to predict the subsequent state given the current state and an action. The dynamics model is then used to optimize the policy. Both NNs are fully convolutional NN (CNN) with three layers, see Fig. 1

The method first runs the so-called warm-up phase, where an initial data set is collected by injecting random control signals into the control system, followed by training the involved NN models. The warm-up phase aims to ensure rough estimates of policy and dynamic NN and, consequently, stabilize the training procedure in the beginning. After the warm-up phase, the method iterates the following three phases, from which phase one (1) is run in parallel to phases two (2) and three (3).

1. Running the policy: the method collects data by running the policy in the AO control loop for T timesteps (a single episode).
2. Improving the dynamics model: the dynamics model parameters are optimized via a supervised learning objective.
3. Improving the policy: the policy parameters are optimized by utilizing the dynamics model.

Let us now describe the PO4AO algorithm in more detail. The dynamics model $\hat{p}_\omega : (\boldsymbol{s}_t, \boldsymbol{a}_t) \mapsto \boldsymbol{s}_{t+1}$ parametrized by ω is expressed as an ensemble, i.e., a collection of deterministic CNNs, where ω represents the weights and biases of the networks. Moreover, the policy

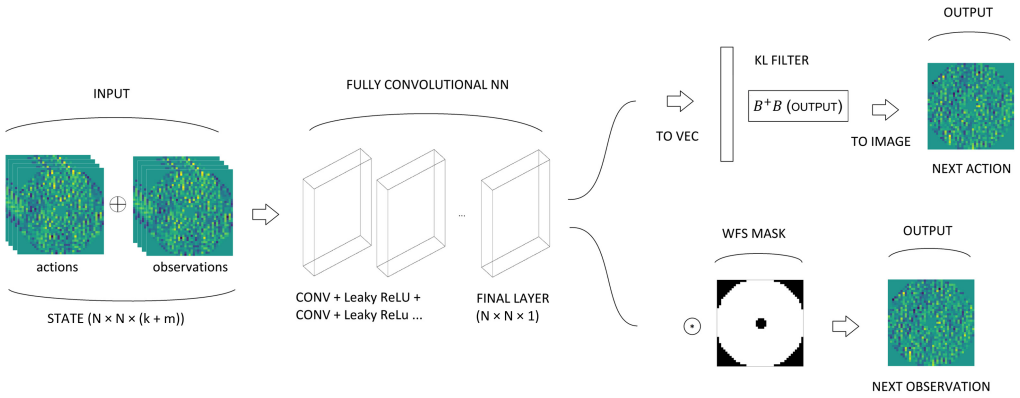

Fig. 1 NN architectures. Both NN, the dynamics, and policy take input tensor concatenations of past actions and observations. They also share the same fully convolutional structure in the first three layers. At the output layer, the policy model includes the KL-filtering scheme (upper right corner), and the dynamics model output is multiplied with the WFS mask (lower right corner). For the GHOST, the input and output images are 24×24 pixels (set by the DM).

mapping $\pi_\theta : \boldsymbol{s}_t \mapsto \boldsymbol{a}_t$ parametrized by θ is constructed as a fully CNN followed by a modal filter layer, i.e.

$$\pi_\theta(\boldsymbol{s}_t) = P_m F_\theta(\boldsymbol{s}_t), \tag{7}$$

where P_m is the projection map, and F_θ is a fully CNN, where the output is vectorized, and θ represents the weights and biases of the CNN.

In step 1, telemetry (tuples of $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ saved into a dataset $\mathcal{D}$) is collected by operating the AO control loop with the current (or initial) parametrization of the policy. In practice, PO4AO utilizes two datasets: one for the data collected during the warm-up and one for the most recent data (see details in Sec. 6).

Utilizing this data, in step 2, the dynamics model is trained, i.e., the parametrization is optimized by minimizing the squared difference between the true next states and the predictions according to

$$\sum_{\mathcal{D}} \|\boldsymbol{s}_{t+1} - \hat{p}_\omega(\boldsymbol{s}_t, \boldsymbol{a}_t)\|^2 = \sum_{\mathcal{D}} \|\boldsymbol{o}_{t+1} - \hat{\boldsymbol{o}}_{t+1}\|^2, \tag{8}$$

where $\boldsymbol{o}_{t+1}$ is obtained from the state $\boldsymbol{s}_{t+1}$ and $\hat{\boldsymbol{o}}_{t+1}$ is the observation predicted by $\hat{p}_\omega(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The parameters are optimized by the Adam algorithm.[46]

The objective of step 3 is to find policy parameters θ that maximize the expected reward within some pre-defined time horizon H given the dynamics of the environment (in our case, the approximate model $\hat{p}_\omega$), that is

$$\arg\max_\theta \sum_{\mathbf{s} \in \mathcal{D}} \sum_{t=1}^{H} \hat{r}_\omega(\tilde{\boldsymbol{s}}_t, \pi_\theta(\tilde{\boldsymbol{s}}_t)), \tag{9}$$

where H is so-called planning horizon and

$$\tilde{\mathbf{s}}_1 = \mathbf{s} \quad \text{and} \quad \tilde{\mathbf{s}}_{t+1} = \hat{p}_\omega(\tilde{\boldsymbol{s}}_t, \pi_\theta(\tilde{\boldsymbol{s}}_t)).$$

In practice, this is done by sampling from previously observed data points, computing the actions, and using the dynamics model to simulate the future. Moreover, we use the differentiability of the reward and backpropagate through the models.

We give separate pseudo codes for the warm-up phase and the two parallel processes, which are step one (1), i.e., running the policy, and steps two (2) and three (2), i.e., improving the dynamics model and improving the policy. Algorithms 1, 2, and 3 give a full pseudo-code for the procedures.

Algorithm 1 PO4AO warm up

Initialize policy and dynamics model parameters θ and ω randomly
Initialize gradient iteration length K, batch size $B < |\mathcal{D}|$ and planning horizon H
for number of warm-up episodes **do**
  for frame t to T **do**
    control with noisy integrator, i.e., $\boldsymbol{a}_t = g\Delta\boldsymbol{v}_t + \sigma x$, where x is Gaussian random noise and full command $\boldsymbol{v}_t = l\boldsymbol{v}_{t-1} + \boldsymbol{a}_t$. Record data $\{\mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t\}$ and append to $\mathcal{D}$
  end for
  reduce σ
end for
Fit dynamics by minimizing Eq. (8) w.r.t ω using Adam
Fit policy by maximizing Eq. (9)
Start parallel processes described in Algorithms 2 and 3.

Algorithm 2 PO4AO control thread

while observing **do**
  for frame t to T **do**
    wait for camera readout
    run policy $a_t = \pi_\theta(a_t|s_t)$, and send a_t to DM
    record data $\{\mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t\}$ append to $\mathcal{D}$
  end for
end while

6 PO4AO Implementation and Hyperparameters

PO4AO has a lot of free adjustable parameters; see Table 1. The subsections below discuss the specific choices and how they affect the method's performance. We arrange hyperparameters under four different subcategories.

6.1 Reinforcement Learning Parameters

These parameters set the frequency on which the policy NN is updated. The episode length (T in the pseudo codes) is the number of frames in an episode; for example, 500 frames on GPU-based high-order adaptive optics testbench (GHOST) running at 350 Hz is 1.4 s. A single training procedure is run during the episode (i.e., the dynamics and the policy optimization). After each episode, the policy is updated with the newly updated model. The episode length determines the maximum speed at which PO4AO can adapt to changing conditions.

The warm-up length determines how many episodes are run in the warm-up phase. For example, if the warm-up length is 20, the first 20 episodes are run with the noisy integrator. The Initial warm-up noise sets the maximum noise variance for the added noise component, and the Minimum warm-up noise sets the minimum. The noise is reduced linearly during the warm-up – starting from maximum and finishing at minimum. The training procedure is started after the warm-up phase. The noise levels should be adjusted considering the dynamic range of the DM and WFS. Initially, the control should be really noisy, but we should not saturate the mirror too much. We use 1% of the full range of the DM. The control should be close to the integrator performance level in the latest warm-up episodes. The progressive reduction of

Algorithm 3 PO4AO training thread

Initialize gradient iteration length K, batch size $B < |\mathcal{D}|$ and planning horizon H
while observing **do**
  Fit dynamics by minimizing Eq. (8) w.r.t ω using Adam
  for iteration $k = 1$ to K **do**
    Sample a mini-batch of $B < |\mathcal{D}|$ states $\{s_\tau\}$ from $\mathcal{D}$
    for each $\mathbf{s}_\tau$ in the mini batch **do**
      Set $\tilde{\mathbf{s}}_1^\tau = \mathbf{s}_\tau$
      for $t = 1$ to H **do**
        Predict $\boldsymbol{a}_t = \pi_\theta(\boldsymbol{s}_t)$
        Predict $\boldsymbol{s}_{t+1} = \hat{p}_\omega(\boldsymbol{s}_t, \boldsymbol{a}_t)$
        Calculate $R_t = \hat{r}_\omega(\boldsymbol{s}_t, \boldsymbol{a}_t)$
      end for
    end for
    Update θ by taking a gradient step according to $\nabla_\theta \sum_{t=\tau}^{\tau+H} R_t$ with Adam.
  end for
end while

the injected noise amplitudes in the warmup data covers a wide range of actions and states of the system, which is crucial knowledge for PO4AO to avoid instabilities when occasionally facing large wave-front residuals.

The loss function penalty parameter α defines the amount of regularization in the reward function Eq. (6). The default value $\alpha = 0.1$ provided enough regularization without affecting the performance.

6.2 Training Parameters

These parameters set the number of gradient steps in dynamics and policy optimization. After the warm-up phase, the loop is suspended, and the first training procedure is run on the data obtained for the warm-up. The parameter iterations after warm-up sets the number of gradient steps/iterations in this first training procedure. The suspension of the loop could be avoided by doing the warm-up training in parallel to warm-up control. However, suspension offers more flexible implementation for testing required warm-up training time, and the warm-up phase is usually avoided using a pre-trained model.

After the first training iteration, the loop is closed with policy, and the parallel training procedure is started. Parameters iterations during the episode dynamics and—policy parameters set the number of gradient steps on the parallel training thread. The latter parameters should be set so that the training procedure finishes during a single episode; for example, with an episode length of 1.4 s, the training procedure should take a maximum of 1.4 s. A good rule is to train the dynamics model more than the policy.

The mini batch size is the number of data points used to calculate a single gradient step. These data points are randomly sampled from the dataset (i.e., replay buffer or warm-up buffer).

6.3 Markov Decision Process Parameters

The MDP formulation outlined in Sec. 4 is specified by the parameters under the category MDP parameters. The number of history frames, k, decides the number of past measurements in the MDP formulation—the policy (controller) does not remember events outside this horizon. Consequently, periodic disturbances occurring at a rate longer than the history horizon are hard to predict.

Table 1 Table of adjustable PO4AO parameters. The values here were used during the experiments in the results section, and they are a good starting point for tuning the method into new instruments.

RL parameters		
Parameter	Value	Units
Episode length	500	Frames
Warm-up episodes	20	Episodes
Initial warm-up noise	2	% of the maximum stroke
Minimum warm-up noise	1	% of the maximum stroke
Reward penalty (α)	0.1	—
Training parameters		
Parameter	Value	Units
Iterations after warm-up dynamics	300	Gradient steps
Iterations after warm-up policy (K)	150	Gradient steps
Iterations during episode dynamics	30	Gradient steps
Iterations during episode policy (K)	15	Gradient steps
Mini batch size in training (B)	32	Data samples
MDP parameters		
Number of history frames	32	Frames
Planning horizon	4	Frames
Replay buffers		
Replay buffer size	20	Episodes
Warm-up buffer size	20	Episodes
Train warm-up percent	20	Percent (%)

Compared to a short history, the long history horizon has advantages. First, it considers more measurements and can effectively average the measurement noise. Second, a long history enables the method to predict low-order vibration. On the other hand, longer history increases the computational burden and hampers the training (the bigger the input, the more free parameters to train).

The choice of the number of history frames is not trivial and should be tuned according to the properties of the instrument. In Sec. 7.5, we run an experiment with different history lengths and compare the results.

The planning horizon, H, sets the future time window considered by the PO4AO (see Sec. 5). The loop latency drives the choice of this parameter, that is, the time delay. A good and stable choice of these parameters is typically 1 to 2 frames longer than the expected time delay of the system (see detailed discussion in Ref. 11).

6.4 Replay Buffers

PO4AO includes two buffers for saving data: the warm-up and replay buffer. The warm-up buffer saves the data recorded during the warm-up phase and stays the same after the warm-up phase. The warm-up buffer size is usually set to the length of the warm-up phase, that is, in our case, 20 episodes (10,000 frames). The newest data is added to the replay buffer, which keeps the latest

replay buffer size episodes in memory. The mini-batch sampled during the training is sampled from the warm-up buffer with the probability set by the train warm-up percentage and otherwise from the replay buffer. The warmup buffer data with the relatively large Gaussian noise added to the actions are needed to remind the policy of "bad" actions. Suppose the models are only trained with recent data. In that case, PO4AO will eventually forget about the actions outside the good control regime and, consequently, start to explore the region's bad actions again, leading to bad performance. We note here that the data in the bad region does not need to be very accurate—the PO4AO only has to remember that the good rewards are observed when the loop is behaving well.

The length of the replay buffer also affects the method's ability to adapt to changing conditions. A short replay buffer will react to changes in condition quickly but is prone to overfitting as the method trains on shorter data sequences. For our system and testing, a replay buffer length of 20 episodes was a good compromise between the two effects.

7 Experiments

Here, we present the results of several experiments performed with PO4AO on GHOST to explore its performance for high-level conditions relevant to operational on-sky AO. Specifically, we explore

1. The impact of the temporal delay on the performance by adding artificial extra delay (unknown to PO4AO) to the control loop.
2. The robustness and performance for low S/N
3. The ability to cope with misregistration
4. The effect of the history length (number of history frames in MDP parameters, Table 1) on performance.

In all experiments, the PO4AO is compared against an integrator whose gain is adjusted to minimize WFS residuals in all cases separately. Most other parameters listed in Table 1 (RL, training, replay buffers, and NN models) were kept constant. First, we introduce our lab setup in Secs. 7.1 and 7.2, then the individual experiments.

7.1 GPU-Based High-Order Adaptive Optics Testbench

The GHOST laboratory AO system[47] has been built to evaluate new AO control techniques, specifically predictive control, for the ELT planetary camera and spectrograph. Located at the ESO headquarters in Garching, Germany, the GHOST utilizes a simple single-source (single-mode fiber-coupled 770 nm SLED) on-axis setup equipped with a pyramid WFS and a Boston micromachines deformable mirror (DM-492). A programmable spatial light modulator (SLM, Meadowlark HSP1920-600-1300-HSP8) introduces turbulence with high spatial resolution.

The GHOST also splits the beam before the WFS to provide a "science channel" with a classical Lyot coronagraph ($4\,\lambda/D$ diameter mask and a circular Lyot stop undersized to 85%) and a CMOS camera Basler ACA2040-90 μm) with a sampling of $\sim$3 pixel/λ/D. Figure 2 shows the coronagraphic PSF limited by the bench aberrations (left), and the long-exposure coronagraphic PSF during replay of the numerically simulated first stage residuals by the SLM (right). The grid of satellite spots is produced by the actuator structure of the DM-492. The brightest of these spots is about $8e^{-4}$ of the intensity of the central PSF with the Lyot mask removed.

The real-time control (RTC) is built using commercial off-the-shelf server components and two Nvidia RTX Titan Graphics processing units (GPU)s. As the software solution, the GHOST makes use of the COSMIC RTC.[48] The COSMIC RTC is a platform developed for AO real-time control (AO RTC) and proposed for several future AO instruments.

The GHOST simulates a two-stage XAO system closely resembling the VLT/SPHERE+ setup. The initial control phase is conducted via simulation, utilizing a pyramid or SH-WFS (40×40 grid), effectively overseeing 800 modes. The atmosphere in the numeric simulation is sampled at twice the rate of the control loop. The residual wavefront error is subsequently

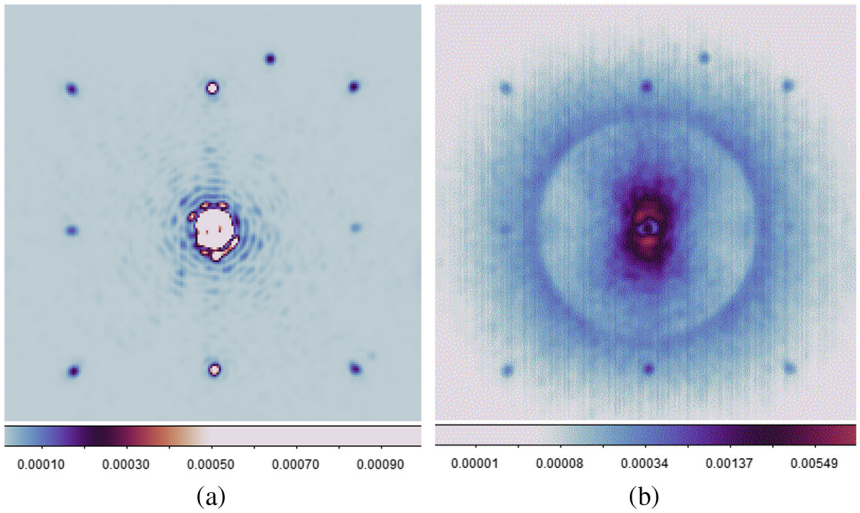

Fig. 2 GHOST coronagraphic PSFs. (a) The PSF without any turbulence, and the DM set to be flat. (b) The PSF with simulated 1-stage systems residual phase screens played on SLM, and a flat DM. The speckle at around 1 o'clock is a ghost in the system.

captured and applied to the setup via an SLM. The second stage control utilizes the actual hardware, a pyramid wavefront sensor coupled with the DM controlling 300 to 400 modes, at the SLM frequency (the second stage runs effectively twice the first stage frequency). We operate the SLM and DM at the maximum (stable) frequency of the SLM, 350 Hz, in all experiments.

7.1.1 *Implementation to GHOST and interfacing to COSMIC pipeline*

The Python script that runs the PO4AO is composed of two threads. One thread (called the control thread) controls the system with the trained policy NN by reading and writing to the shared memory buffers. The other thread (called the training thread) constantly trains the models, i.e., optimizes the dynamics and policy model parameters by running lines 6 to 17 in the Algorithm 3. Once a single episode is run, the newly updated policy parameters are imported to the control thread.

The PO4AO algorithm interfaces to the COSMIC pipeline through a shared memory buffer, see Fig. 3. The COSMIC pipeline reads the raw WFS images, preprocesses them to slopes, and adds the reference slopes; then, it multiplies the slopes with the reconstruction matrix, writes the resulting delta voltages to the shared memory buffer, and suspends the loop. The Python script reads the delta voltages from the shared memory and passes the data to PO4AO. The PO4AO keeps past residual voltages and actions applied in memory. The memory of past delta voltages and actions and the new delta voltages are passed to policy NN that outputs a two-dimensional image of voltages, the action, which is then written to DM command buffer where COSMIC catches them. The control thread also saves the observed delta voltages and actions fed (the orange boxes in the figure) to the replay buffers for the training.

7.2 Simulation Set-Up

To simulate a faster second-stage system using GHOST, we followed these steps. Firstly, we generated the residual phase screens numerically for the lab setup by using the Object-Oriented Python and AO simulation tool.[49] We simulated an 8-m telescope with a 41x41 DM and a PWFS, observing a natural guide star of magnitude 6.16. The time delay of the first stage was set to two frames. The atmospheric turbulence was a combination of nine frozen flow layers with Von Karman power spectra, with a Fried parameter of 15 cm at 550 nm wavelength. The simulation parameters can be found in Table 2. We controlled the simulated system using an integrator and recorded the residual turbulence after DM correction. This exact set of residual turbulence phase-screens is then replayed by the SLM for all our GHOST experiments.

Table 2 Simulations parameters. For full wind profiles of the simulated atmosphere, see the GitHub repository.

Numerically simulated first-stage		
Parameter	Value	Units
Telescope diameter	8	m
Obstruction ratio	0	Percent
Sampling frequency (atm)	2000	Hz
AO loop frequency (DM)	1000	Hz
NGS magnitude	6.16	—
WFS wavelength	0.79	μm
Actuators	41	Across the pupil
PWFS modulation	3	λ/D
KL modes	900	Modes
Integrator gain	0.5	—
GHOST (second-stage)		
Parameter	Value	Units
Sampling frequency (simulation)	2000	Hz
Sampling frequency (real-time)	350	Hz
Actuators	24	Across the pupil
PWFS modulation	4	λ/D
WFS and Science Cam wavelength	770	nm
Light source	6 and 187	10^3 camera counts/ WFS frame
Atmosphere parameters		
Fried parameter	15	cm @ 500 nm
Number of layers	9	—
Effective wind speeds	34	m/s
L_0 (m)	30	m

7.3 Time Delay Experiment

The AO system delay budget encompasses various factors that contribute to the overall delay. Initially, a minimum 1-frame delay cannot be avoided due to two primary sources: 0.5 frames result from frame integrations performed by the WFS camera, and another 0.5 frames arise from the sample and hold operations carried out by the DM. This 1-frame delay is a fundamental limitation and is encountered in systems such as GHOST as well.

When operating GHOST at a frame rate of 350 Hz, each frame corresponds to a duration of 2.86 milliseconds (ms). The camera readout time for a subwindow, which is ~70 ms (μs), along with the time taken by the COSMIC pipeline (ranging from 100 to 150 μs) and Python control (ranging from 600 to 800 μs, discussed in Sec. 7.7) and the rapid settling of the DM (less than 2 μs), are all relatively insignificant compared to the frame duration. Consequently, the total delay experienced is primarily comprised of the unavoidable 1-frame delay, along with additional

0.3 frames for the remaining processes, resulting in a delay of around 1.3 frames. Therefore, an extra frame should be added to the GHOST system to replicate the scenario encountered in systems like SPHERE.

When the SPHERE camera operates at its maximum frame rate, determined by the camera readout time (around 1380 Hz), an additional full frame delay is incurred, bringing the total delay to 2 frames. Additionally, smaller overheads such as real-time control (RTC) computation, communication, and DM settling contribute to the overall delay, typically amounting to a few hundred microseconds. When all these factors are combined, the total delay experienced by the SPHERE system reaches ∼2.4 frames.[50] However, if the system is run at a slower speed where the readout time is less than one frame, the total delay is reduced accordingly.

To study the method's predictive ability and demonstrate its ability to adapt to unknown time delays, we did the following test: we ran PO4AO (with the same hyperparameters) on three different temporal delays. We control these delays by adding an external buffer that suspends the DM commands for a given number of frames on the second stage only. First with zero frames of additional delay, second with one extra manually added frame of delay, and then with two (2) frames of additional delay. All test runs are compared against an integrator, which is separately tuned for all delays to give the best performance (lowest WFS residual variance).

We compare the performance of PO4AO and the integrator in two ways: by comparing the reward, that is, the residual variance of the WFS measurements, and then by comparing the science camera images. The light source is set relatively bright, resulting in around 187k camera counts/frame with a standard deviation of 214. Figure 4 plots the learning curves, the cumulative wavefront residual variance (i.e., negative reward), after each episode during the run. The first 20 episodes are the warm-up phase, where the integrator controls the system with added Gaussian noise on the control signal; after each warm-up episode, the noise is reduced linearly. For time delays, PO4AO outperforms the integrator after the warm-up of 20 episodes (10,000 frames), and the performance converges at around 80 episodes (40,000 frames). Converged PO4AO provides around a factor of three improvement in reconstructed wavefront variance for all delays.

Figure 5 shows the long exposure (8 s) science camera images of the integrator and converged PO4AO after the first 80 episodes. Figure 6 plots the related azimuthal average over intensities of the images, that is, the contrast. Along with the PO4AO and the integrator results, we plot the contrast without turbulence which is limited by the bench non-common path aberrations. We also plot the open-loop PSF contrast, where the SLM replays the numerically simulated first stage residuals. We observe a factor of 1.5 to 3 improvement in contrast (depending on angular separation and time delay).

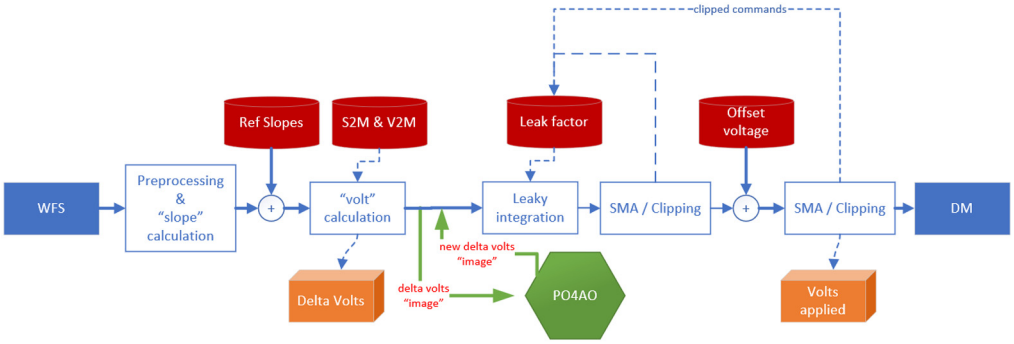

Fig. 3 PO4AO interface for RTC pipeline. COSMIC pipeline preprocesses the raw WFS data, projects it to DM-space with command matrix (using the modal basis matrices: S2M and V2M), then writes the "delta volts" to the shared memory buffer, and suspends the loop. Python interface (the green box) reads the shared memory buffer and passes the data to the PO4AO implementation. The PO4AO calculates the next command and saves the data (orange boxes), and the Python interface writes the command to shared memory, where COSMIC registers the command and passes it to the saturation management algorithm (SMA)/clipping stage.

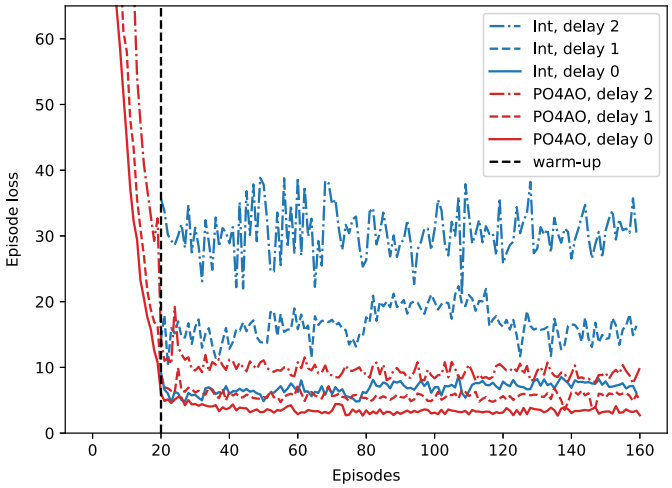

Fig. 4 Learning curves for time delay experiments. The red lines correspond to the performance of PO4AO during each episode, and the blue lines are for the integrator. A single episode is 500 frames. The gray dashed line marks the end of the integrator warm-up for PO4AO. In all cases, the PO4AO outperforms the integrator all ready after the warm-up period. The training is done parallel to control, so the 10 episodes correspond to ∼14 s in the figure.

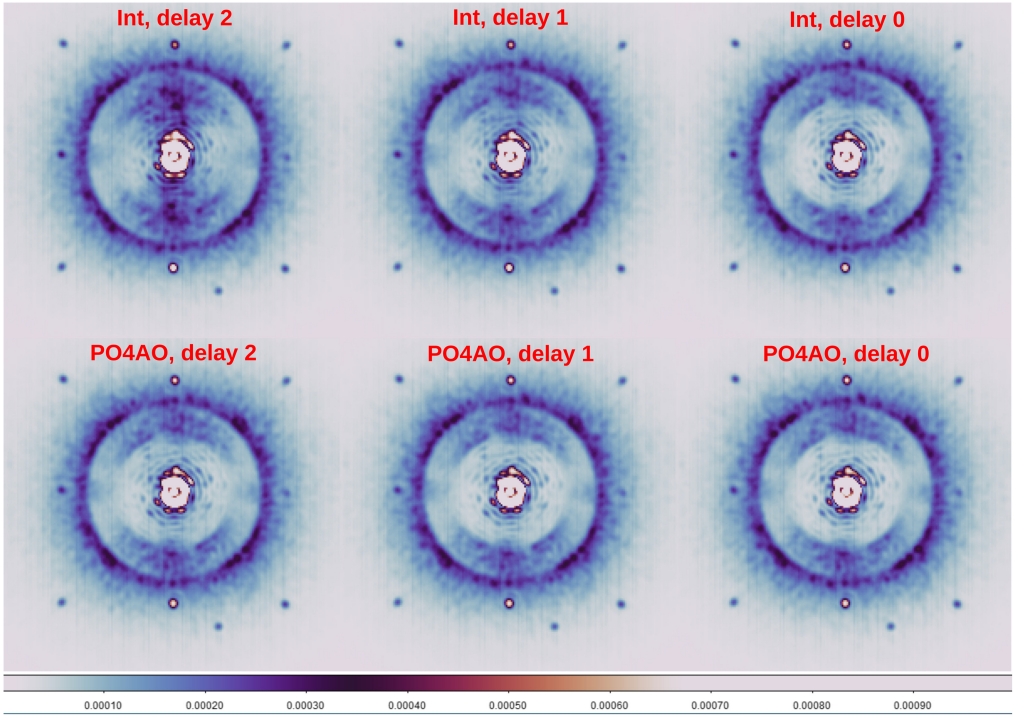

Fig. 5 PSFs on different additional control delays. The top row is for the integrator control, and the bottom is for PO4AO. The PO4AO and its hyper-parameters are exactly the same for all time delays – the time delay is learned from the interaction.

7.4 Low Flux Experiment

In this experiment, we study the performance of PO4AO under low flux. In the time-delay experiment, the internal light source was set relatively bright, resulting in a flux of 187,000 camera counts/frame. For the low flux experiment, we set the light source such that the flux was around 6000 camera counts/frame, while the detector noise was around 3000 camera counts/frame(S/N ≈2). Further, we fixed the additional control delay to 1 frame, resulting in a typical overall delay of slightly over two frames, and ran the algorithm for 140 episodes, including the warm-up.

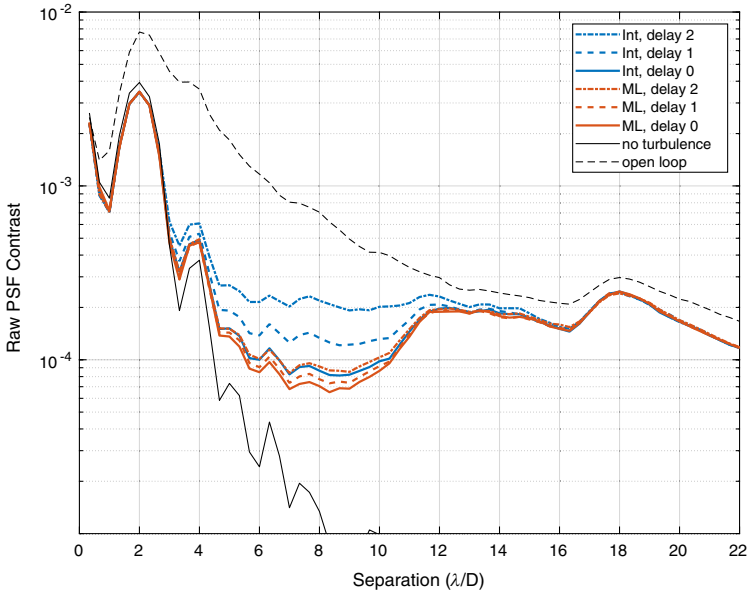

Fig. 6 The contrast with different time delays. The blue lines correspond to the azimuthal average of integrator images, and the red lines are the same for PO4AO. The line style indicates the length of the time delay. The solid black line is for flat DM, and the dashed black line is for flat DM (open loop) with first-stage residuals played on SLM. Note that around 11 λ/D is the correction area of the DM-492, and around 18 λ/D is the correction area of the numerically simulated first-stage DM. PO4A0 provides better contrast inside the second-stage control radius for all time delays.

Again, we record cumulative loss (negative reward) after each episode (Fig. 7) and the science camera PSF after 80 episodes (Fig. 8). The cumulative loss here is dominated by photon and sensor noise; hence, we plotted the cumulative reward when turbulence is not played. It represents the absolute lower limit for the performance. Figure 8 shows that PO4AO considerably reduces the photon flux inside the control radius. Consequently, PO4AO enables the imaging of fainter objects.

7.5 Vibration Reduction and Effect of MDP Formulation

This section studies the effect of the number of past telemetry frames (observations + actions) in the MDP formulation. We set the control delay to one extra frame and ran the same test but now with different history lengths. We note here that this test was run with a slower simulated wind profile (effective wind speed 26 m/s; the experiment could not be repeated with the original wind profile due to malfunctioning SLM); hence the lower rewards. Figure 9 shows the corresponding episode cumulative losses after the warm-up phase for different history lengths. As a general trend, the PO4AO corrections performance improves with the number of history frames

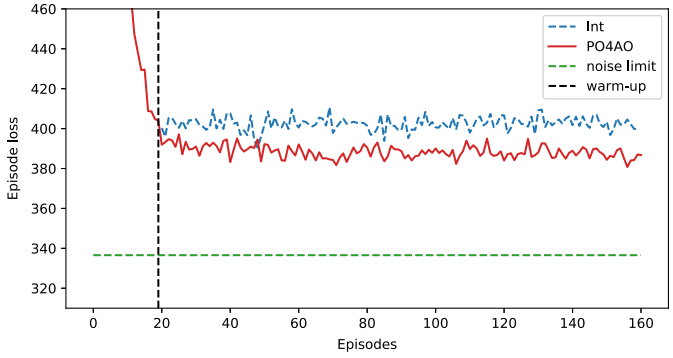

Fig. 7 Learning curve for the low flux experiment. The blue line is the cumulative reward after each episode for the integrator, and the red line is for PO4AO. The dashed green line is the reward after each episode when the turbulence was not played and the loop opened.

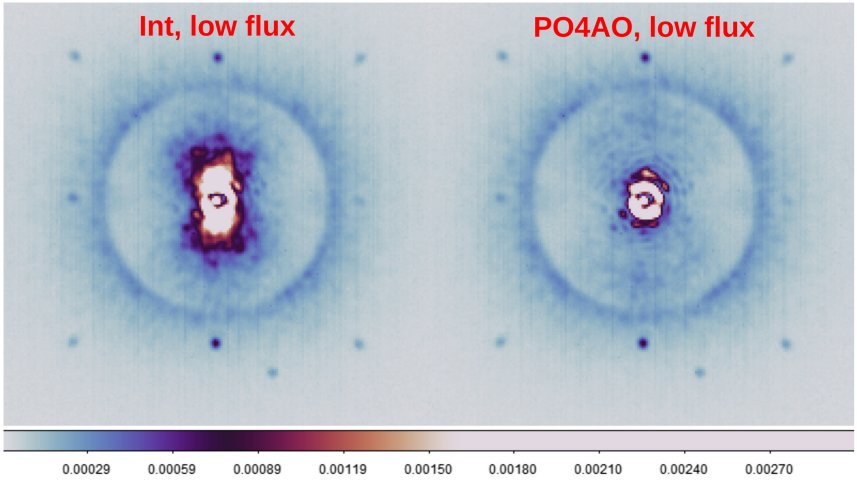

Fig. 8 PSFs during the low flux experiment. The left image is for the integrator, and the right is for the PO4AO.

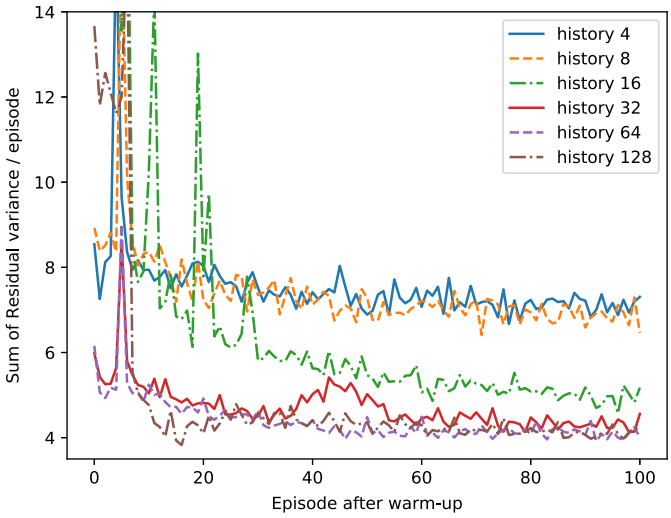

Fig. 9 Learning curves for different history lengths. We plot the cumulative reward after each episode for all history lengths after the warm-up phase. All history lengths have the same warm-up phase (not included in the plot). The longer the time length is, the better the performance.

considered. To better understand the increased predictive power with the number of frames, Fig. 10 shows the temporal power spectral densities of the mode #1 for the history length of 128 frames, history length of 4 frames, and the integrator. The mode #1 is tip mode in the direction of the dominant wind (the ground layer).

We observe that the PSD increases towards the Nyquist limit because of the intersampling signal of the first stage.[5] The 1st stage is numerically simulated with a frame rate that is half the one in the second stage; the 1st stage DM is only updated every second frame of the simulation. Hence, the location of the intersampling signal matches the Nyquist frequency of the second stage and creates this particular shape of the PSD.

Compared to the integrator, PO4AO dampens the residuals at mid-frequencies. However, like for linear controllers, we observe the Bode theorem-like behavior (waterbed effect); we observe amplification at higher frequencies. Some low-frequency residuals are transmitted through the leak.

In addition, we observed a vibration spike at 16 Hz. The spike is equally strong for the integrator and PO4AO with four history frames, but the PO4AO with 128 history frames dampens the spike. Also, PO4AO with 32 and 64 could dampen the spike to some extent but not all the way as the PO4AO with 128 history frames.

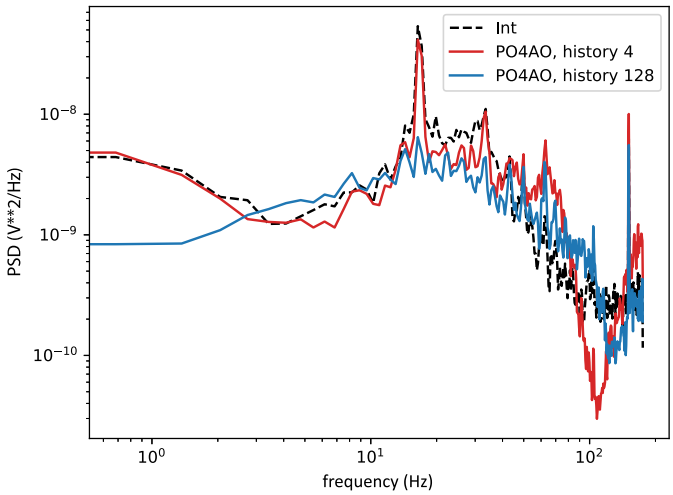

Fig. 10 Temporal PSD of mode #1 (tip) on different history lengths.

7.6 Misregistration Experiment

Finally, we ran an experiment to confirm the method's robustness for misregistration. We start with a pre-trained PO4AO and well-tuned integrator. The system is calibrated, the PO4AO is trained, and the integrator is tuned as before with no misregistration. Then we close the loop and start to introduce various degrees of misregistration. We introduce misregistration by shifting the DM off-axis manually while the loop is closed.

The whole experiment lasts 320 episodes. The first 80 episodes are run with no misregistration. We then shift the DM by 40 micrometers (μm) and let it run for 80 episodes; then at $\sim$160 episodes, we shift the DM an additional 40 μm (combining 80 μm), and finally, at 240 episodes, we shift the mirror by 40 μm (combing 120 μm) and let it run 80 episodes. The DM actuator spacing is 300 μm, meaning that 120 μm corresponds to a 40% shift. We repeat the same experiment with a well-tuned integrator. Finally, for reference, we re-calibrated (new interaction matrix and tuned gain) the system at 120 μm of misregistration and ran a well-tuned integrator with a re-calibrated reconstruction matrix (the number of modes is kept the same). The results of this experiment are shown in Fig. 11. PO4AO is able to obtain stability and performance with dynamic misregistration while the integrator gets unstable. However, we observed PO4AO some instability when we first moved the mirror (see the spike around episode (76)), but PO4AO automatically recovered from this. PO4AO also outperforms the re-calibrated integrator by a large margin.

7.7 Performance on RTC

This section discusses the latency introduced by the method. The total delay budget of the COSMIC pipeline is discussed previously in Sec. 7.3 and results in $\sim$150 μs (preprocessing of WFS data, projection to voltages, and clipping stage). In addition, there is a delay coming from the method's Python implementation. This includes all latency starting from when COSMIC writes the residual volts to the shared memory to the point when the Python interface writes a set of new commands to the shared memory, and the loop resumes.

More precisely, this latency is composed of the shared memory modification, the time required for converting vectors to images, and vice versa, as CNNs typically process images as input. Furthermore, the process of data collection adds to the overall delay, along with the delay caused by the update of the state memory of past telemetry frames. Finally, the delay budget also accounts for the time the policy NN takes for its forward pass, which is essential for control decisions and accounts for the majority of the total latency. Table 3 shows the latency of different components for the most common CNN (32 history steps) used in the experiment. Collectively, these factors contribute to the total latency experienced in the system's operation. Table 3 shows that the most significant amount of time (500 μs) is spent on the CNN forward pass. Saving data and updating the state information combines around 180 μs. The remaining $\approx$200 μs is spent on image-to-vector modifications.

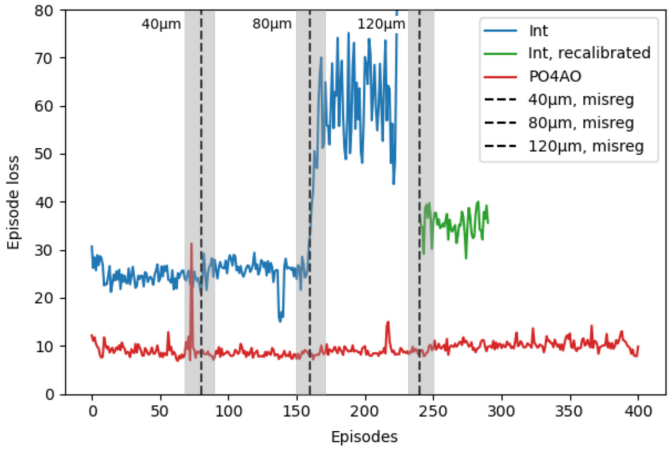

Fig. 11 The misregistration experiment. Here, we plot the cumulative loss over an episode during the misregistration experiment. The blue line is the well-tuned integrator calibrated with centered DM, and the green line is the well-tuned integrator calibrated with DM 120 microns off-axis. The red line is the PO4AO calibrated and pre-trained with centered DM. The dashed black lines indicate the moment when DM was manually shifted. Since the shifting was done manually, the gray areas around the line indicate the uncertainty of the exact moment.

Table 3 Latency terms of control thread.

Inference speed			
	CNN inference and jitter	Saving data	Update of the state
CNN (32 history)	532.86 + −8 μs	52.63 μs	128.38 μs

We further examine the latency by recording the end-to-end latency of CNN with different history lengths and PO4AO in the integrator mode (integrator running through the Python interface). By end-to-end latency, we mean the time from the WFS image arriving to the time the command is sent to the DM. Further, we measure the speed of the training procedure for each case given the training parameter (see Table 2). The training parameters were chosen such that the training procedure fit inside a single episode for each history length. These results are shown in Table 4. The integrator is around 500 μs (time of single CNN forward pass) faster, as expected. The longer state history does not affect the overall loop latency. The longer state history only includes more convolutional filters (in the first layers). Hence, these introduce mostly parallel

Table 4 Total latency of Python implementation.

Total latency				
	Past frames (k and m)	Total latency (μs)	Jitter std.	Tr. time/episode
Integrator	—	724	85	—
CNN	4	1205	60	0.78 s
CNN	8	1230	77	0.79 s
CNN	16	1208	57	0.80 s
CNN	32	1218	73	0.81 s
CNN	64	1219	73	0.91 s
CNN	128	1196	60	1.27 s

operations, and the GPU we used has plenty of memory headroom for the shallow architectures we used. However, the training time increases when more telemetry frames are added. This sets a limit to the episode length. Consequently, PO4AO, with fewer history frames, can adapt faster to misregistration and atmospheric wind conditions.

8 Conclusion and Discussion

To conclude, this paper demonstrates that PO4AO is a robust controller for a second-stage AO system in a lab simulation. RL is shown to mitigate several critical error terms in XAO control, such as misregistration, photon noise, and temporal error. Moreover, running PO4AO is a turnkey operation as the hyperparameters are tuned only when the method is implemented, and the method adapts automatically to changing conditions like noise level, misregistration, and wind profile. Extensive experiments on GHOST confirm that PO4AO can adapt to and mitigate these error terms on real hardware. In addition, we showed that PO4AO could also mitigate vibrations if it considers enough past telemetry frames.

However, like most deep RL methods, PO4AO is somewhat sensitive to the choice of hyperparameters. Tuning the parameters can take time, but the method performs robustly under all conditions once a good combination of hyperparameters is found. The method did not improve the system's stability to more degrees of freedom. We observed that the integrator was stable to ∼350 KL modes, and PO4AO did not enable us to control more KL modes robustly.

Additionally, we open-sourced the Python implementation of PO4AO, which is well-commented and can be easily adapted to numerical simulations or real hardware. The paper also discusses the hyperparameters of PO4AO and how they affect the method. The open-source implementation introduces an additional ∼700 μs to the pipeline latency, making it suitable for systems running lower than 1000 Hz (depending on the pipeline latency) that can be tested on-sky with this implementation. The implementation requires a minimum of two GPUs on the RTC, one for inference and one for training. The memory bottleneck for bigger NN structures (e.g., longer time length, more filters per layer, and more DoF in the input) is the training GPU RAM; the backpropagation requires the most GPU memory.

Various avenues exist to optimize the method further: First, critical code sections can be re-implemented by transitioning to lower-level programming languages, such as C, to benefit from direct memory access and optimized execution. Additionally, the NN could be implemented with NVIDIA TensorRT, which is a high-performance inference optimizer that delivers low latency and high throughput for deep learning inference applications. Second, optimizing memory handling through techniques like circular buffers can enhance data storage and retrieval efficiency, reducing computational load. Third, streamlining the pipeline to handle control voltage images instead of vectors can improve data processing and efficiency. Furthermore, the 500 μs forward pass is slower than expected for the shallow architecture we used. Our investigation indicates a CPU bottleneck somewhere. Once the code is optimized, we expect significant gains in latency. These optimizations could provide latency gains that would support loop speed up to a few kilohertz.

9 Appendix A: Implementation Details

This section discusses open-source Python implementation. The code consists of three Python files: po4ao.py, po4ao_models.py, po4ao_util.py, and po4ao_config.py. The po4ao.py script includes the main function that starts the warm-up phase and training procedure and closes the loop. The file includes the step-function that interfaces the Python code to the RTC pipeline. The PO4AO algorithm interfaces to the COSMIC pipeline through a so-called step-function. The step function can be found in the po4ao.py file. The step function is pipeline-specific and should be replaced if the method is adapted to different RTC pipelines or numeric simulations. The po4ao.py file also includes functions for the training thread, control thread, and warm-up phase. The file po4ao_models.py contains the NN models: the policy and the dynamics, and file po4ao_util.py includes utility functions, such as the implementation for replay buffers and shared memory optimizers for training. The configuration file po4ao_config.py includes all adjustable hyperparameters of the method.

The file po4ao_config.py contains few parameters beyond the PO4AO parameters discussed in Sec. 6. In the following, give a short explanation of them.

9.1 A.1 Parameters Under Subcategory the Integrator

If the parameter integrator is set to True, the PO4AO runs in integrator mode, where data is collected, and models are trained, but the control thread runs on integrator control. These parameters define the internal integrator controller of POAO and affect the PO4AO itself. The gain value gives the gain of the integrator during either the warm-up phase or in the integrator mode. The "leakages" and "number of modes" set the leakages and the DoF of the internal integrator and PO4AO. The number of modes should be set to a value where the integrator is stable on a reasonable gain value.

9.2 A.2 Parameters Under Subcategory the Neural Network Models

The NN parameters are for the architecture of the Policy and the dynamics NNs. The models are generic three-layer fully CNN without any up- or down-sampling and share the same architecture excluding the output layer.[11] The CNN filters/layer sets the number of convolutional filters on layers. Different NNs architectures can be easily implemented in the code by replacing the models in the po4ao_models.py- file.

9.3 A.3 Other Parameters

The control delay parameter (under the category "MDP parameters" in the code) sets the additional control delay. It decides how many frames a control signal is suspended on top of the natural loop latency. It is only for testing the behavior of PO4AO on different time delays. Obviously, for a real system, it is set to zero for optimized performance.

Code and Data Availability

The codes used in this paper are available on GitHub repository [https://github.com/jnousi/PO4AO.git]. The code is documented and annotated to help readers understand the methodology and reproduce the results.

We encourage readers to use the data and codes for their own research and to cite this paper as the source of the data. If you have any questions about the data or the codes, please do not hesitate to contact us.

Acknowledgments

The work of J.N. and T.H. was supported by the Academy of Finland (Grant Nos. 326961, 345720, and 353094). Part of this work was supported by an ETH Zurich Research Grant. S.P.Q. gratefully acknowledges the financial support from ETH Zurich. Part of this work has been carried out within the framework of the National Centre of Competence in Research PlanetS supported by the Swiss National Science Foundation (SNSF) (Grant Nos. 51NF40 182901 and 51NF40 205606). S.P.Q. and A.M.G acknowledge the support from the SNSF. The COSMIC is developed through a strategic partnership between LESIA at Observatoire de Paris, AITC at the Australian National University, Microgate, and CAS at Swinburne University of Technology.

References

1. C. Marois et al., "Images of a fourth planet orbiting hr 8799," *Nature* **468**(7327), 1080–1083 (2010).
2. A.-M. Lagrange et al., "A probable giant planet imaged in the β pictoris disk-VLT/NaCo deep l'-band imaging," *Astron. Astrophys.* **493**(2), L21–L25 (2009).
3. B. Macintosh et al., "Discovery and spectroscopy of the young Jovian planet 51 Eri b with the Gemini planet imager," *Science* **350**(6256), 64–67 (2015).
4. O. Guyon, "Limits of adaptive optics for high-contrast imaging," *Astrophys. J.* **629**(1), 592 (2005).
5. N. Cerpa-Urra et al., "Cascade adaptive optics: contrast performance analysis of a two-stage controller by numerical simulations," *J. Astron. Telesc. Instrum. Syst.* **8**(1), 019001 (2022).
6. A. Boccaletti et al., "Sphere+: imaging young Jupiters down to the snowline," arXiv:2003.05714 (2020).
7. L. Poyneer, M. van Dam, and J.-P. Véran, "Experimental verification of the frozen flow atmospheric turbulence assumption with use of astronomical adaptive optics telemetry," *J. Opt. Soc. Am. A* **26**(4), 833–846 (2009).

8. C. Heritier et al., "A new calibration strategy for adaptive telescopes with pyramid WFS," *Mon. Not. R. Astron. Soc.* **481**(2), 2829–2840 (2018).
9. V. Korkiakoski, C. Vérinaud, and M. Le Louarn, "Improving the performance of a pyramid wavefront sensor with modal sensitivity compensation," *Appl. Opt.* **47**(1), 79–87 (2008).
10. V. Deo et al., "A telescope-ready approach for modal compensation of pyramid wavefront sensor optical gain," *Astron. Astrophys.* **629**, A107 (2019).
11. J. Nousiainen et al., "Toward on-sky adaptive optics control using reinforcement learning-model-based policy optimization for adaptive optics," *Astron. Astrophys.* **664**, A71 (2022).
12. J. Nousiainen et al., "Adaptive optics control using model-based reinforcement learning," *Opt. Express* **29**(10), 15327–15344 (2021).
13. S. Y. Haffert et al., "Data-driven subspace predictive control: lab demonstration and future outlook," *Proc. SPIE* **11823**, 1182306 (2021).
14. C. Kulcsár et al., "Optimal control, observers and integrators in adaptive optics," *Opt. Express* **14**(17), 7464–7476 (2006).
15. R. N. Paschall and D. J. Anderson, "Linear quadratic Gaussian control of a deformable mirror adaptive optics system with time-delayed measurements," *Appl. Opt.* **32**(31), 6347–6358 (1993).
16. M. Gray and B. Le Roux, "Ensemble transform Kalman filter, a nonstationary control law for complex ao systems on ELTs: theoretical aspects and first simulations results," *Proc. SPIE* **8447**, 84471T (2012).
17. J.-M. Conan et al., "Are integral controllers adapted to the new era of ELT adaptive optics?," in *AO4ELT* (2011).
18. C. Correia et al., "Adapting optimal LQG methods to ELT-sized AO systems," in *1st AO4ELT Conf.-Adapt. Opt. for Extremely Large Telesc.*, EDP Sciences, p. 07003 (2010).
19. C. Correia et al., "On the optimal reconstruction and control of adaptive optical systems with mirror dynamics," *J. Opt. Soc. Am. A* **27**(2), 333–349 (2010).
20. C. M. Correia et al., "Modeling astronomical adaptive optics performance with temporally filtered Wiener reconstruction of slope data," *J. Opt. Soc. Am. A* **34**(10), 1877–1887 (2017).
21. B. Sinquin et al., "On-sky results for adaptive optics control with data-driven models on low-order modes," *Mon. Not. R. Astron. Soc.* **498**(3), 3228–3240 (2020).
22. O. Guyon and J. Males, "Adaptive optics predictive control with empirical orthogonal functions (EOFs)," arXiv:1707.00570 (2017).
23. L. A. Poyneer, B. A. Macintosh, and J.-P. Véran, "Fourier transform wavefront control with adaptive prediction of the atmosphere," *J. Opt. Soc. Am. A* **24**(9), 2645–2660 (2007).
24. C. Dessenne, P.-Y. Madec, and G. Rousset, "Optimization of a predictive controller for closed-loop adaptive optics," *Appl. Opt.* **37**(21), 4623–4633 (1998).
25. M. van Kooten, N. Doelman, and M. Kenworthy, *Performance of AO Predictive Control in the Presence of Non-Stationary Turbulence*, Instituto de Astrofisica de Canarias (2017).
26. M. van Kooten, N. Doelman, and M. Kenworthy, "Impact of time-variant turbulence behavior on prediction for adaptive optics systems," *J. Opt. Soc. Am. A* **36**(5), 731–740 (2019).
27. J. R. Males and O. Guyon, "Ground-based adaptive optics coronagraphic performance under closed-loop predictive control," *J. Astron. Telesc. Instrum. Syst.* **4**(1), 019001 (2018).
28. M. A. van Kooten et al., "Predictive wavefront control on Keck II adaptive optics bench: on-sky coronagraphic results," *J. Astron. Telesc. Instrum. Syst.* **8**(2), 029006 (2022).
29. R. Swanson et al., "Wavefront reconstruction and prediction with convolutional neural networks," *Proc. SPIE* **10703**, 107031F (2018).
30. Z. Sun et al., "A Bayesian regularized artificial neural network for adaptive optics forecasting," *Opt. Commun.* **382**, 519–527 (2017).
31. X. Liu, T. Morris, and C. Saunter, "Using long short-term memory for wavefront prediction in adaptive optics," *Lect. Notes Comput. Sci.* **11730**, 537–542 (2019).
32. A. P. Wong et al., "Predictive control for adaptive optics using neural networks," *J. Astron. Telesc. Instrum. Syst.* **7**(1), 019001 (2021).
33. R. Swanson et al., "Closed loop predictive control of adaptive optics systems with convolutional neural networks," *Mon. Not. R. Astron. Soc.* **503**(2), 2944–2954 (2021).
34. R. Hafeez et al., "Forecasting wavefront corrections in an adaptive optics system," *J. Astron. Telesc. Instrum. Syst.* **8**(2), 029003 (2022).
35. R. Landman and S. Y. Haffert, "Nonlinear wavefront reconstruction with convolutional neural networks for fourier-based wavefront sensors," *Opt. Express* **28**, 16644–16657 (2020).
36. A. P. Wong et al., "Nonlinear wave front reconstruction from a pyramid sensor using neural networks," *Publ. Astron. Soc. Pac.* **135**(1053), 114501 (2023).
37. F. Archinuk et al., "Mitigating the non-linearities in a pyramid wavefront sensor," arXiv:2305.09805 (2023).
38. Y. He et al., "Deep learning wavefront sensing method for Shack-Hartmann sensors with sparse subapertures," *Opt. Express* **29**(11), 17669–17682 (2021).

39. B. Pou et al., "Adaptive optics control with multi-agent model-free reinforcement learning," *Opt. Express* **30**, 2991–3015 (2022).
40. R. Landman et al., "Self-optimizing adaptive optics control with reinforcement learning," *Proc. SPIE* **11448**, 1144849 (2020).
41. R. Landman et al., "Self-optimizing adaptive optics control with reinforcement learning for high-contrast imaging," *J. Astron. Telesc. Instrum. Syst.* **7**(3), 039002 (2021).
42. J. Fowler and R. Landman, "Tempestas ex machina: a review of machine learning methods for wavefront control," *Proc. SPIE* **12680**, 126800E (2023).
43. E. Gendron, "Modal control optimization in an adaptive optics system," in *Eur. Southern Obs. Conf. and Workshop Proc.*, Vol. **48**, p. 187 (1994).
44. P.-Y. Madec, "Control techniques," in *Adaptive Optics in Astronomy*, F. Roddier, Ed., pp. 131–154, Cambridge University Press (1999).
45. R. Bellman, "A Markovian decision process," *J. Math. Mech.* **6**, 679–684 (1957).
46. D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR 2015)* (2015).
47. B. Engler et al., "The GPU-based high-order adaptive optics testbench," *Proc. SPIE* **12185**, 1218558 (2022).
48. F. Ferreira et al., "Hard real-time core software of the AO R TC cosmic platform: architecture and performance," *Proc. SPIE* **11448**, 1144815 (2020).
49. C. T. Heritier, "Object oriented python adaptive optics (OOPAO)," in *AO4ELT7 Proc.* (2023).
50. C. Petit et al., "SAXO: the extreme adaptive optics system of SPHERE (I) system overview and global laboratory performance," *J. Astron. Telesc. Instrum. Syst.* **2**(2), 025003 (2016).

Jalo Nousiainen is a postdoctoral researcher at LUT University. His research focuses on algorithm development for high-contrast instruments dedicated to exoplanet imaging. He has a background in applied mathematics, specifically in inverse problems, Bayesian statistics, and machine learning.

Byron Engler is a postdoctoral fellow at ESO, working in the field of adaptive optics.

Markus Kasper received his PhD from the University of Heidelberg in 1999, after which he joined ESO to work on adaptive optics, high-contrast imaging, and astronomical instrumentation projects. He was involved in many of ESO's AO projects, such as MACAO and NACO LGS, and was ESO's project leader for VLT-SPHERE. He was the principal investigator of the ELT/EPICS phase-A study (2008 to 2010, predecessor of PCS) and of NEAR, the mid-IR imaging experiment to search for habitable planets in Alpha Centauri in collaboration with the Breakthrough Initiatives. He is currently working towards kicking off the ELT-PCS planet imager project.

Biographies of the other authors are not available.