Supplemental Document

Optics EXPRESS

Adaptive optics control with multi-agent model-free reinforcement learning: supplement

B. Pou,[1,2,*] F. Ferreira,[3] E. Quinones,[1] D. Gratadour,[3,4] and M. Martin[2]

[1] *Barcelona Supercomputing Center (BSC), C. Jordi Girona, 29, 08034 Barcelona, Spain*
[2] *Computer Science Department, Universitat Politècnica de Catalunya (UPC), C. Jordi Girona, 31, 08034 Barcelona, Spain*
[3] *LESIA, Observatoire de Paris, Universite PSL, CNRS, Sorbonne Universite, Univ. Paris Diderot, Sorbonne Paris Cite, 5 place Jules Janssen, 92195 Meudon, France*
[4] *Research School of Astronomy and Astrophysics, Australian National University, Canberra, ACT 2611, Australia*
[*] *bartomeu.poumulet@bsc.es*

Supplement DOI: https://doi.org/10.6084/m9.figshare.17714879

Parent Article DOI: https://doi.org/10.1364/OE.444099

Adaptive Optics control with Multi-Agent Model-Free Reinforcement Learning: supplemental document

This supplementary material provides an additional explanation on the B_{tt} modes and Soft Actor Critic. Moreover, it shows the pseudocode of the training process and additional RL hyperparameters used in our simulations.

1. ADDITIONAL EXPLANATION ON THE B_{TT} MODAL BASIS

Let's consider the influence functions, IF_i, that denote how acting on actuator i will affect the DM shape. Any phase represented by the DM, Φ_{DM}, can be obtained through a linear combination of these influence functions:

$$\Phi_{DM} = \sum_i f_i IF_i. \tag{S1}$$

Because of coupling between actuators, this basis is not orthogonal and, on top of this, can reproduce piston. While piston is not seen by a SH-WFS and does not impact image quality, it has to be removed to avoid drifting towards actuators saturation. Additionally, producing tip and tilt with the DM is usually not practical since it requires a lot of stroke, especially for actuators at the edge of the pupil. In this case as well, these two modes are usually filtered out of the degrees of freedom of the DM and compensated using a dedicated subsystem (either using a flat mirror or the DM itself mounted on a dedicated tip-tilt mount).

The B_{tt} basis uses the influence functions to build another representation of possible DM shapes through orthonormal modes, spanning the full DM actuators space, but from which piston and tip-tilt are filtered out. The method is described in appendix A of [1] and recalled hereafter. Using the spatial covariance matrix of the actuators of Eq. (S1), one can retrieve the coefficients of the best fit for piston and tip-tilt on Eq. (S1) and create a set of generators that do not produce those modes. The generators matrix is then diagonalised and normalised using the corresponding eigenvalues to obtain a matrix B describing a set of orthonormal modes from which piston and tip-tilt have been removed. Finally, the tip-tilt modes are reintroduced in the basis by adding two extra columns to the matrix with a single non-zero value on the diagonal obtained as the inverse of μ, the eigenvalues of $IF_{tt}^t \dot{I}F_{tt}$ where IF_{tt} are the influence functions of tip-tilt. The final basis B_{tt} basis is expressed as:

$$B_{tt} = \begin{pmatrix} & & & 0 & 0 \\ & B & & \vdots & \vdots \\ & & & 0 & 0 \\ 0 & \dots & 0 & 1/\mu & 0 \\ 0 & \dots & 0 & 0 & 1/\mu \end{pmatrix}. \tag{S2}$$

The modes are normalised over the pupil surface, S, as $\frac{1}{S}\int B_{tt,i}^2 dS = 1\mu m^2$. With this basis the phase will computed as:

$$\Phi_{DM} = \sum_i v_i B_{tt,i}, \tag{S3}$$

where $B_{tt,i}$ are the vectors of the B_{tt} basis.

2. SOFT ACTOR CRITIC

Soft Actor Critic (SAC) [2], is considered one of the state-of-the-art methods in RL. SAC considers a stochastic policy, π, described as a probability distribution. We can introduce SAC by taking a closer look at its name.

1. The "soft" naming is given by the author's to its method because it uses a maximum entropy version of the return, J^{MaxEnt}, where a trade-off between reward and information entropy of the policy (denoted as $H(\pi)$) is introduced:

$$J^{MaxEnt}(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{t=T-1} \gamma^t (r_{t+1} + \beta H(\pi(\cdot|\boldsymbol{s}_t)))\right], \tag{S4}$$

   with the entropy of a policy defined as:

$$H(\pi(\cdot|\boldsymbol{s})) = \mathbb{E}_{\boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s})}\left[-\log(\pi(\boldsymbol{a}|\boldsymbol{s}))\right], \tag{S5}$$

   where the notation $\boldsymbol{a} \sim \pi$ indicates that the action, $\boldsymbol{a}$, has been sampled from the policy π and $\beta \geq 0$ is a hyperparameter that determines the strength of the entropy against the reward. Notice that a value of $\beta = 0$ returns the original definition of J. The entropy works as a regularisation in the learning procedure which increases the robustness with different initialisations and variations of hyperparameters which many other RL algorithms are culprit of.

2. Regarding actor critic, *actor* denotes the policy, π, which in SAC is stochastic, and *critic* denotes a function that predicts J^{MaxEnt} starting on a state $\boldsymbol{s}$, executing action $\boldsymbol{a}$, and then following the current policy afterwards denoted as $Q(\boldsymbol{s}, \boldsymbol{a})$. The idea of the critic is to guide the actor to policies that maximise Eq. (S4).

   In Deep RL, actor critic methods are functions parameterised with neural network weights, ϕ and θ respectively, henceforth we can write: $\pi_\phi(\boldsymbol{s})$ and $Q_\theta(\boldsymbol{s}, \boldsymbol{a})$. The neural network parameterisation of both actor and critic done via means of neural networks is a standard procedure in RL to overcome high dimensional state spaces.

   Consider a trajectory, τ, of samples $\tau = (\boldsymbol{s}_1, \boldsymbol{a}_1, r_2, \boldsymbol{s}_2, \boldsymbol{a}_2, r_3, ...)$ obtained by interacting with the environment. Minimising the following loss function $L(\theta)$, provided by comparing critic prediction $Q_\theta(\boldsymbol{s}, \boldsymbol{a})$ with the soft Bellman equation [2]

$$L(\theta) = \mathop{\mathbb{E}}_{\substack{\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}'\sim\tau,\\ \boldsymbol{a}'\sim\pi_\phi(\cdot|\boldsymbol{s}')}}\left[\frac{1}{2}\left(Q_\theta(\boldsymbol{s},\boldsymbol{a}) - \overbrace{(r + \gamma(Q_\theta(\boldsymbol{s}',\boldsymbol{a}') - \beta\log(\pi_\phi(\boldsymbol{a}'|\boldsymbol{s}'))))}^{\text{Soft Bellman Eq.}}\right)^2\right], \tag{S6}$$

   provides an iterative procedure to learn θ. Notice that $\boldsymbol{a}'$ is sampled given a $\boldsymbol{s}'$ obtained in the trajectory.

   In SAC, the actor represents a probability distribution which is usually represented as a normal distribution. The policy's neural network, π_ϕ, will output two values: a mean vector, $\boldsymbol{\mu}_\phi$, and a standard deviation vector $\boldsymbol{\sigma}_\phi$ dependant on parameters ϕ. This two values are the components of the Normal distribution $\mathcal{N}(\boldsymbol{\mu}_\phi, \boldsymbol{\sigma}^2_\phi)$. The actor should be learned in a similar iterative manner as the critic by minimising the difference between the current policy distribution and the exponential of the critic estimation which has been proved to lead to better policies [2] as:

$$L(\phi) = \mathbb{E}_{\boldsymbol{s}\sim\tau,\boldsymbol{a}\sim\pi_\phi(\cdot|\boldsymbol{s})}\left[\beta\log(\pi_\phi(\boldsymbol{a}|\boldsymbol{s})) - Q_\theta(\boldsymbol{s},\boldsymbol{a})\right]. \tag{S7}$$

   However, estimating the gradient of the loss above is intractable because we are sampling the actions with the same parameters that we are trying to update. To circumvent this problem, SAC uses the so-called reparameterisation trick: sampling an action with the policy parameters, ϕ, is rewritten as sampling from a deterministic function dependant on ϕ and a noise variable, $\boldsymbol{\epsilon}$, independent of ϕ, which is represented with the same distribution type as the one used to sample actions: a Normal distribution per action dimension with mean 0 and standard deviation 1, $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$. The sampled action per update becomes:

$$\hat{\boldsymbol{a}} = \boldsymbol{\mu}_\phi + (\boldsymbol{\sigma^2}_\phi \odot \mathcal{N}(0, I)), \tag{S8}$$

where $\odot$ indicates element-wise multiplication. Therefore, the loss expression ends up as:

$$L(\phi) = \mathbb{E}_{\boldsymbol{s}\sim\tau, \hat{\boldsymbol{a}}\sim f(\phi,\boldsymbol{\epsilon})} \left[\beta \log(\pi_\phi(\hat{\boldsymbol{a}}|\boldsymbol{s})) - Q_\theta(\boldsymbol{s}, \hat{\boldsymbol{a}})\right]. \tag{S9}$$

Each loss function is used as the starting point for the backpropagation algorithm for critic and actor respectively. Once the gradients are obtained from said algorithm, they are used to update the weights via Gradient Descent optimisation method.

Finally, to fully understand SAC it is important to explain a few tricks used for stabilisation purposes. First, the tuples $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s'})$ obtained while interacting with the environment, are saved in a memory buffer $\mathcal{D}$. Then, at update time, batches of this memory are used to update the weights from the actor and critic, hence now the trajectory in Eqs. (S6) and (S9) would be changed as sampling from $\mathcal{D}$. This is carried out because consecutive tuples are highly correlated and may lead to instability.

Second, as seen in Eq. (S6), the soft Bellman equation used to compare with the Q-function predictions depends on the same weights that are being updated θ which is another source of instability. To solve this, a copy of the original critic network with weights θ' is introduced with the name of target network and is used only on the critic term present in the Soft Bellman equation. At each update, weights θ will change and θ' will adopt a new value via Polyak averaging:

$$\theta' = \rho\theta + (1-\rho)\theta', \tag{S10}$$

where ρ is a hyperparameter to be tuned with usually low values to limit abrupt changes in the target network.

Third, on training time, the value of β is automatically tuned via optimisation where the entropy is constrained to a certain value, $\bar{H}$, in expectation (in [2] it is simply $\bar{H} = -Dim(\mathbb{A})$). This can be expressed as a loss in the following equation:

$$L(\beta) = \mathbb{E}_{\boldsymbol{s}\sim\tau, \boldsymbol{a}\sim\pi_\phi(\cdot|\boldsymbol{s})} \left[-\beta log \pi_\phi(\boldsymbol{a}|\boldsymbol{s}) - \bar{H}\right]. \tag{S11}$$

With this update, the additional hyperparameter β is eliminated.

Fourth, critic estimations are known to be overoptimistic [3] which may lead to wrong actions. The established solution to this problem is to use two Q-networks and select the minimum over both predictions. Consequently, SAC has a total of four Q-networks if we take into account that both online and target networks need two networks each to avoid the overestimation of the critic.

Finally, in SAC, the policy, π_ϕ, outputs two vectors: a mean vector, $\boldsymbol{\mu}_\phi$, and a standard deviation vector $\boldsymbol{\sigma}_\phi$ dependant on parameters ϕ. This two values are the parameters of a Normal distribution $\mathcal{N}(\boldsymbol{\mu}_\phi, \boldsymbol{\sigma}^2_\phi)$. While training, SAC samples from the Normal distribution to obtain the action i.e. $\boldsymbol{a}_t \sim \mathcal{N}(\boldsymbol{\mu}_\phi, \boldsymbol{\sigma^2}_\phi)$. This is done for exploratory purposes in order to discover actions that may lead to more reward in the future. However, in testing time, once the policy is trained, the action is simply the mean of the Normal distribution, $\boldsymbol{a}_t = \boldsymbol{\mu}_\phi$, as ineffective exploratory actions should be avoided to achieve maximum performance.

3. PSEUDOCODE OF THE TRAINING LOOP

Pseudocode S1 shows the training process of a MARL controller in the AO context. Due to time concerns, in our implementation, we have parallelised the update but not the action selection. In practice, the only part that may increase the delay is the action selection, which must be parallelised to the finest degree.

4. ADDITIONAL RL HYPERPARAMETERS

In Table S1, we show the hyperparameters used for SAC in all the AO experiments. Most of the hyperparameter values are default ones (as reported in [2]) as they usually work well for any problem. We experimented with the value of γ and found out that lower values of $\gamma \in [0, 0.5]$ lead to better performance. We recall here that γ is an hyperparameter that weighs future rewards and, in the AO case, the commands only have a short-term effect, therefore, a small value of γ is required for high performance.

Algorithm S1. MARL-AO training loop

Given a trained autoencoder, N policies $(\pi^1, ..., \pi^N)$, N empty memories $(\mathcal{D}^i, ..., \mathcal{D}^N)$ and a vector of $\boldsymbol{M^i}$ modes to be controlled per agent i.
for episode=1 to end **do**
  simulation.reset() # Resets simulation and DM shape
  for t=0 to T **do**
    raytrace() # Raytracing operation for the target image and WFS
    wfs.compute_image() # Computes WFS image based on last raytracing operation
    autoencoder.denoise_wfs_image()
    wfs.do_centroids()
    $\hat{\boldsymbol{c}}_t$ = integrator.do_control() # Computes current residual commands
    for agent i=1 to N in parallel **do**
      $\boldsymbol{s}_t^i$=create_state(i) # Creates state for agent i based on $\hat{\boldsymbol{c}}_t$ and previous $\hat{\boldsymbol{C}}$
      $\boldsymbol{a}_t^i \sim \pi^i(\cdot|\boldsymbol{s}_t^i)$
    Concatenate the actions into a joint action: $\boldsymbol{a}_t = (\boldsymbol{a}_t^1, ..., \boldsymbol{a}_t^N)$
    $\boldsymbol{C}_t = B_{tt} \cdot (\hat{\boldsymbol{C}}_{t-1} - g \cdot \hat{\boldsymbol{c}}_t + \boldsymbol{a}_t)$
    applyCommand($\boldsymbol{C}_t$) # Applies C simulating the delay in the DM
    calculate_psf_image()
    for agent i to N in parallel **do**
      $r_t^i = -\frac{\kappa}{|\boldsymbol{M^i}|} \sum_{m \in \boldsymbol{M^i}} (\hat{c}_t^m)^2$
    if $t \geq d$ **then**
      for agent i=1 to N in parallel **do**
        Add tuple $(\boldsymbol{s}^i = \boldsymbol{s}_{t-d}^i, \boldsymbol{a}^i = \boldsymbol{a}_{t-d}^i, \boldsymbol{s'}^{,i} = s_t^i, r^i = r_t^i)$ to memory $\mathcal{D}^i$
        Train SAC with mini-batches of tuples as reported in Section 2 and [2].

γ	0.1	**Memory size**	10^6
Num. hidden size	256	**Batch size**	256
Num. hidden layers	2	**Gradient descent optimiser**	ADAM
$\boldsymbol{\rho}$	0.005	$\boldsymbol{\beta}$	Automatic
Learning rate	0.0003	**Neural Network Layers**	Fully connected

Table S1. Soft Actor Critic hyperparameters.

REFERENCES

1. F. Ferreira, E. Gendron, G. Rousset, and D. Gratadour, "Numerical estimation of wavefront error breakdown in adaptive optics," Astron. & Astrophys. **616**, A102 (2018).
2. T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," arXiv preprint arXiv:1812.05905 (2018).
3. H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30 (2016), pp. 2094–2100.