Detect to Focus
Latent-Space Autofocusing System with Decentralized Hierarchical Multi-Agent Reinforcement Learning
Anikina, Anna; Rogov, Oleg Y.; Dylov, Dmitry V.

Published in:
IEEE Access

DOI:
10.1109/ACCESS.2023.3303844

Publication date:
2023

Document version
Publisher's PDF, also known as Version of record

Document license:
CC BY-NC-ND

Citation for published version (APA):
Detect to Focus: Latent-Space Autofocusing System With Decentralized Hierarchical Multi-Agent Reinforcement Learning

ANNA ANIKINA¹,², OLEG Y. ROGOV¹, AND DMITRY V. DYLOV¹, (Member, IEEE)

¹Skolkovo Institute of Science and Technology (Skoltech), 121205 Moscow, Russia
²Department of Computer Science, University of Copenhagen, 1172 Copenhagen, Denmark
Corresponding author: Dmitry V. Dylov (d.dylov@skoltech.ru)

This work was supported in part by the RFBR Fund under Grant 21-51-12012.

ABSTRACT State-of-the-art object detection models are frequently trained offline using available datasets, such as ImageNet: large and overly diverse data that are unbalanced and hard to cluster semantically. This kind of training drops the object detection performance should the change in illumination, in the environmental conditions (e.g., rain or dust), or in the lens positioning (out-of-focus blur) occur. We propose a simple way to intelligently control the camera and the lens focusing settings in such scenarios using DASHA, a Decentralized Autofocusing System with Hierarchical Agents. Our agents learn to focus on scenes in challenging environments, significantly enhancing the pattern recognition capacity beyond the popular detection models (YOLO, Faster R-CNN, and Retina are considered). At the same time, the decentralized training allows preserving the equipment from overheating. The algorithm relies on the latent representation of the camera’s stream and, thus, it is the first method to allow a completely no-reference imaging, where the system trains itself to auto-focus itself. The paper introduces a novel method for auto-tuning imaging equipment via hierarchical reinforcement learning. The technique involves the use of two interacting agents which independently manage the camera and lens settings, enabling optimal focus across different lighting situations. The unique aspect of this approach is its dependence on the latent feature vector of the real-time image scene for autofocusing, marking it as the first method of its kind to auto-tune a camera without necessitating reference or calibration data.

INDEX TERMS Artificial intelligence, neural networks, reinforcement learning, multi-agent systems, computer vision, photography, imaging, lenses.

I. INTRODUCTION
Modern computational photography is unimaginable without the image acquisition hardware and image analytics tandem [1]. The discipline absorbs the best of both worlds by embedding the state-of-the-art computer vision algorithms into the fastest image processing chips [2]. This tandem ultimately gave birth to the paradigm of 
ε-
photography [3], where a stream of photos could be processed and computationally analyzed on the fly to overcome a wide range of camera limitations. Both limitations, optical (e.g., the lens’ limitations) and electronic (e.g., the shot noise), can be eliminated by virtue of pre-trained artificial neural networks (ANN), embedded into the camera’s hardware. That enabled a plethora of photo enhancement options for the end-user, unimaginable with a single-exposure camera some 10 years ago. Such algorithmic ‘improvements’ of the camera’s hardware capacity include the higher dynamic range, the larger depth of focus, the broader color gamut, the wider/panoramic shooting, the night photography, and others.

The opposite direction of enrichment in this tandem has been inexplicably underestimated by the community until very recently [4]. In particular, the embedded algorithms rarely use the arsenal of image-improving hardware components within the camera to adjust/update themselves. The embedded models are typically pre-trained on offline datasets and, at best, use the recently proposed paradigm of online
learning [5] to update the pre-trained ANNs weights, yet, without venturing into the feedback dialogue with the camera hardware.

Recent rapid developments in imaging hardware have provided an opportunity for a new approach in lens autofocus and exposure control in the image with the aid of feedback-based control loops. Practical applications of automated camera calibrations span from on-the-fly segmentation of road scenes for autonomous vehicles [6] and smartphones [7], to lithography [8], to biomedical imaging [9], [10], becoming indispensable in these areas.

In this study, we were motivated by the recent advances in reinforcement learning (RL), one versatile candidate for filling the hardware feedback niche [11], [12], [13]. We aspired to check if we could train hardware in the imaging system to perform image adjustments live in order to improve the performance of popular embedded scene analysis models, such as Yolo [14], Faster R-CNN [15], or RetinaNet [16]. In short, we present a perception-inspired automatic focusing system that needs no reference and is supported by hierarchical RL.

The contribution of this study is in the following:

- To the best of our knowledge, this is the first no-reference autofocusing system based on the low-level representation of a live scene in the latent space.
- A new paradigm for passive lens autofocus by means of hierarchical reinforcement learning.
- The proposed approach is decentralized: both the camera and its lens are controlled by separate agents, preserving the hardware from overheating and guaranteeing functionality in adverse ambient conditions.

### II. RELATED WORK

#### A. AUTO-FOCUSING STATE-OF-THE-ART

To automatically focus on an object, the exact distance from the focal plane to the object should be determined, which could be done by active and passive autofocus (AF).

Active methods include the presence of additional (auxiliary) elements, such as, an ultrasonic locator [17], infrared LEDs [18] or lasers [19].

Passive AF works through the analysis of the image information captured by the system. Passive AF, in turn, can be broadly divided into contrast-based and phase-based detection approaches. Contrast-based AF is performed by determining the image sharpness. To find the lens position, this approach requires capturing a sequence of images with different focal distance and then calculating each image’s Focus Measure Value [20], [21], [22], [23], [24], [25], [26]. This sequence significantly affects the performance time. Phase detection methods represent the image by dividing it into the right and the left pixel parts on the camera sensor (the right phase image and the left phase image, respectively [27], [28]). After such splitting, the phase shift between the parts is computed, yielding the relative position between the object and the focal plane. Ultimately, the sign and the extent of the phase shift between the left and the right phase images is used to gauge the optimal lens position to align the focal and the object planes. Ideally, there is an exact correspondence between the phase shift and the travel distance of the lens. However, in practice, the phase shifts are very sensitive to the noise, making it challenging to find the optimal focal position [11], [29], [30].

Recent AF and autoexposure works have started leaning towards perception and object detection paradigms. For example, [31] features a system that synthetically renders a refocused video from a large-depth-of-focus video, using the upcoming video frames to deliver a context-aware autofocus in the current frame. Likewise, in [32], a center area of the image is partitioned, where the objects are most likely to be located, which is then followed by a battery-consuming frame stability estimation. The study in [33] reports an auto exposure control, optimized for a down-stream task, such as object detection, relying on the intersection-over-union loss. The authors of [34] optimized the shutter speed and the voltage gain on the sensor, relying on the offline dataset of 2.5k pictures in a simple scene with a limited number of known objects.

Despite the success of the classical methods, deep learning tools have been proactively engaged for performing the AF [35], including area-based depth maps [36], [37], [38], per-pixel depths from the multi-view stereo and the super-resolution [39] perspectives.

#### B. AUTOFOCUS WITH RL

Surprisingly few articles cover the reinforcement learning (RL) approach to the AF problem, with two functional implementations reported ([11] and [12]). These works use the reference value to evaluate the resulting image, subsequently imposing serious restrictions on hardware. Therefore, given a variable ambient environment the computations can be dramatically time-consuming and are unfavorable for the cutting-edge applications in computational photography. We describe these two papers in detail in the Supplement and use them for the comparison in Section V.
### C. EXPOSURE ADJUSTMENT WITH RL

Several publications [40], [41] report the use of RL for controlling the exposure time of a camera, all of which do not use a direct learning on the hardware. We also describe these methods in the Supplement.

### III. BACKGROUND: MULTI-AGENT RL & HIERARCHY

#### A. AGENTS AND OPTIMIZATION PROBLEM

Consider RL problem as a Markov Decision Process (MDP) formally represented in form of tuple \( (S, A, P, R, γ) \), where \( S \) is a finite set of states and \( s_t \in S \) denotes that agent at time \( t \) being in state \( s \); \( A \) is a set of actions and agent interacts with the environment by choosing action \( a_t \in A \) at time \( t \); \( P : S \times A \times S \rightarrow \mathbb{R} \) is the transition probability distribution describes the probability of arriving from \( s_{t-1} \) to \( s_t \) by choosing \( a_{t-1} \); \( R \) is a scalar reward function that controls the behavior of agent: the environment gives reward to agent for each action; and \( γ \in [0, 1] \) is the discount factor providing the importance of immediate reward versus future rewards.

Conventionally [42], [43], [44], two important concepts are introduced: the value and the state value functions. The state value function \( V^\pi \) is the cumulative sum of future rewards for the agent following the policy \( \pi \) from the state \( s \):

\[
V^\pi(s) = \mathbb{E} \left[ \sum_{i=t}^{H-1} γ^{i-t} r_i | s_t = s \right].
\]  

The state-action value function \( Q^\pi(s, a) \) defines the expected discounted sum of rewards \( \sum_{i=0}^{N} r_i \) from state \( s \) at time \( t \) to the \( H \) following the policy \( \pi \):

\[
Q^\pi(s, a) = \mathbb{E} \left[ \sum_{i=t}^{H} γ^{i-t} r_i | s_t = s, a_t = a \right].
\]

The \( V^\pi \) provides the best value following policy \( \pi \) for each of next states, while \( Q^\pi \) shows the effectiveness of actions that the agent chooses by following the policy \( \pi \) for the next states. The corresponding objective function one wants to maximize over the policy parameters \( \theta \) is:

\[
L(\theta) = \hat{E}_t \left[ \log \pi_\theta(a_t | s_t) \hat{A}_t \right] \approx \hat{E}_t \left[ \hat{r}_t \hat{A}_t \right],
\]

where \( \hat{r}_t \) denotes the empirical expectation over time steps, \( \hat{A}_t := Q^\pi_t(s, a) - V^\pi_t(s) \) is the estimated advantage at time \( t \), and \( \hat{r}_t = \frac{\pi(a_t | s_t)}{\pi_{\hat{\theta}}(a_t | s_t)} \) is the probability ratio under the new and old policies.

One of the most widespread optimization methods employed for RL is the Proximal Policy Optimization (PPO). To avoid an instability, typically penalizes large policy changes to stay within a small interval \([1 - \epsilon, 1 + \epsilon]\), where \( \epsilon \) is a hyperparameter, yielding the ultimate objective function needed for our work [43]:

\[
L(\theta) = \hat{E}_t \left[ \min \left( \hat{r}_t(\theta) \hat{A}_t, \text{clip} \left( \hat{r}_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right],
\]

where \( \text{clip} \) stands for the clipped objective restricting the values of \( \hat{r}_t(\theta) \).

In our study, PPO was chosen as the main optimization method, because it provides a universally robust performance in different environments and on different imaging hardware [43]. The comparison with other RL optimization approaches is beyond the scope of this work.

#### B. HIERARCHICAL RL

Hierarchical Reinforcement Learning (HRL) is the case of multi-agent RL and has multiple hierarchy layers of policies compared to the conventional single-agent RL. In the standard two-tier HRL, high-level agent \( A^H \) and low-level agent \( A^L \) are defined with corresponding policies \( \pi^H \) and \( \pi^L \). The higher-level policy sets goals for the lower-level policy, and the lower-level policy attempts to reach them. At the beginning of each episode, the higher-level policy receives the state (or observation) and forms a high-level action (or goal). Then, the lower-level policy receives an observation and a goal and forms a low-level action that affects the environment. The low-level agent receives a reward for each step; at the end of the episode, the high-level agent receives the final reward, and then the process is repeated.

Our approach will differ from the classic HRL. The high-level agent will establish a ‘favorable’ RL-environment for the low-level agent to act in: until the optimal parameters of the internal environment are reached, the low-level agent does not take control. The high-level agent is responsible for the camera parameters (the exposure time \( t_{ex} \)), while the low-level agent works with the lens: the task is to get the selected object in focus.

#### C. OBJECT DETECTION

Object detection is an essential part of the proposed autofocusing system. In this work, we rely on popular object detection models, such as Yolo [14], RetinaNet [16], and Faster R-CNN [15] in their intact (‘off-the-shelf’) variants. Their description and implementation details are given in the Supplement.

### IV. METHODS: DASHA

This section covers the method of controlling the camera’s exposure and focusing, summarized in Fig. 2, using the decentralized hierarchical reinforcement learning framework with two agents: \( A^H \) (high-level agent, Algorithm 1) and \( A^L \) (low-level agent, Algorithm 2). We begin by formally defining the state space, the action space, and the rewards for these agents.

#### A. STATE SPACE

We take advantage of deep auto-encoders, acquiring a low-dimensional feature space \( S \). The \( A^L \) observation space \( s_t \in S \) is a feature vector of size \([2048 \times 1]\) from the camera’s image \( I \), obtained using a ResNet-152 Encoder \( E(\cdot) \) pre-
Algorithm 1

1: \( A^H \) sends action \( a^H \)
2: Change \( t_{ex} \) in camera
3: Grab image \( I \)
4: Evaluate \( a^H \) of \( A^H \) by computing histogram
5: Form observation \( o^H \) from \( I \) using encoder \( E(\cdot) \) and histogram
6: Compute reward \( r^H \)
7: if \( (\text{histogram} < 10\% \text{ max (too dark) or histogram} > 90\% \text{ max (too bright)}) \) then
8: Episode completed with penalty for \( A^H \)
9: else
10: Pass action to \( A^L \) (Low-Level Agent) or successfully end episode (in case of Single Agent)
11: end if
12: return \((o^H, r^H, done)\)

trained [45] with ImageNet [46]:

\[
o^L = E(I).
\] (5)

The observation space of \( A^H \) is a tuple of features and the histogram data (values of the histogram \( \text{val} \) [10 \times 1] and bin edges \( \text{bin} \) [11 \times 1]):

\[
o^H = (E(I), \text{val}, \text{bin}).
\] (6)

### B. ACTION SPACE

Each policy has its own set of actions. The action space for the \( A^H \) is a discrete space of 146 values, where each value corresponds to its own exposure time \( t_{ex} \). This size of the array of the possible exposure time values allows for the non-linear nature of the exposure scale and empirically covers the entire range of needed exposure time values. Otherwise, the action space would have consisted of 5 million values, significantly increasing the training time. The action space for \( A^L \) is a multi-discrete space of 2 values: the coarse and the fine, enabling to tune the Focus Measure Value. The fine, enabling to tune the Focus Measure Value. The action leads to a successful detection of a bounding box of an object, but the image quality assessment yields a metric value insufficient to pass the required threshold, the agent receives a reduced penalty, as it is generally moving in the right direction. The closer to the focus, the smaller the penalty.

\[
A^H_{\text{max}} = 70.0.
\]

In our calibration experiments, we determined that these values correspond to the physical distance between the camera’s sensor and the lens of 36.4 and 24.0 mm, respectively.

### C. REWARD FUNCTION

The \( A^H \) gives the \( A^L \) an intrinsic reward for each step. The reward is defined by a piecewise function that takes into account different cases of agent behavior.

To properly construct the reward function, the following principles were followed:

- **Speed.** The faster the agents reach the desired state, the better. The more unsuccessful steps an agent needs, the greater is the penalty. We accelerate the training of the RL agents by the negative rewards, which force the end of the episode as soon as possible.
- **Encouragement for staying close to the focus.** When an action leads to a successful detection of a bounding box of an object, but the image quality assessment yields a metric value insufficient to pass the required threshold, the agent receives a reduced penalty, as it is generally moving in the right direction. The closer to the focus, the smaller the penalty.
- **The reward function convergence.** An upper bound is set for the reward that the agents can receive for their actions. This provides a clear graph of the agent’s convergence (to visualize the correct actions) and keeps the maximum reward within 1.

Given these principles, the rewards can be formally determined as follows.

For the agent \( A^H \),

\[
r^H = \begin{cases} 
+1, & \text{if } P \in [50, 150], \\
-0.01 \cdot B, & \text{if } P \in [25, 50) \cup (150, 175], \\
-1, & \text{otherwise}.
\end{cases}
\] (7)

In Eq. (7), \( P \) is the peak histogram value and \( B \) is a no-reference image quality metric obtained on the low-level representation of the image by Eq. (5). Each step, \( A^H \) relies on the brightness histogram of the image, with the piece-wise function in Eq. (7) being guided by the following rationale. If the peak histogram value \( P \) falls within the interval...
successful detection event occurred. Thus, the agent aims to being moved in the vicinity of that position where the first
is successful, the computation of the agent receives a penalty. In the case when the object detection
In Eq. (8), for each step when the object is not detected, the
For the agent \( A^L \),
reach the lowest possible value \( B \) while tending to converge to zero, which ultimately brings the lens to the optimal position.

**D. HIGH-LEVEL AGENT \( A^H \)**
This agent manages the built-in camera settings. The main role of this agent is to set up an ‘optimal environment’ for the \( A^L \). At the beginning of each episode, the camera and the lens have the initial parameters set up with their ‘factory’ values. These parameters do not always match the ambient environment optimally, resulting in a poorly exposed scene. Hence, the correct configuration of these parameters is needed so that a visible image can be obtained. The \( A^H \) sends an action \( a^H \) to the camera, which changes the exposure time. The resulting image from the camera \( I \) is analyzed for content by constructing a histogram. If the image is good, the control of the RL environment is transferred to the \( A^L \). If the image is unacceptable, the episode ends immediately with a penalty to the \( A^H \). This routine is summarized in Algorithm 1.

**E. LOW-LEVEL AGENT \( A^L \)**
The \( A^L \) adjusts the lens position for an in-focus image acquisition. As soon as the action of the agent \( a^L \) is commanded to the lens, a new image is acquired from the camera. This image \( I \) is then sent to the object detection model that returns the binary result: the object is successfully detected (in this case, the coordinates of the object are extracted) or the object is not detected. All these actions are repeated until the moment of successful detection or until the end of the episode. In the case of successful object detection, the resulting image is evaluated for the quality using the metric \( B \) and the reward \( r^L \) is received, as outlined in Algorithm 2.

**V. EXPERIMENTS**
For the experiments, we used the 2/3” Basler acA2000-50gm camera (GigE, CMV2000 CMOS, 50 frames per second at a 2 megapixel resolution). The access to the camera is established via the official Basler library pypylon. We also employed the Corning Varioptic C-C-39N0-250 Lens controlled through the cp210x board (Silicon Labs Ltd.) and a DLL-file provided by Corning. To implement RL and to operate both the camera and the lens, we used the rllib

\[
l^L = \begin{cases} 
-1, & \text{if detection fails,} \\
-0.01 \cdot B, & \text{otherwise.}
\end{cases}
\]  

(8)

In Eq. (8), for each step when the object is not detected, the agent receives a penalty. In the case when the object detection is successful, the computation of \( B \) is still needed in order to further tune the sharpness of the image while the lens is being moved in the vicinity of that position where the first successful detection event occurred. Thus, the agent aims to

**Algorithm 2 Low-Level Agent**

1: \( A^L \) sends action \( a^L \)
2: Change lens position
3: Acquire image \( I \)
4: if (Object is detected) then
5: Successful completion of episode
6: else
7: Send new action \( a^L \) until episode is done
8: end if
9: Form observation \( o^L \) from \( I \) using encoder \( E(\cdot) \)
10: Compute reward \( r^L \)
11: return \((o^L, r^L, \text{done})\)

**FIGURE 3.** Histogram comparison for the RL-images stack (purple) and the image obtained by a built-in algorithm embedded into the camera (green). Taken to demonstrate consistency of our single RL agent that performs automatic exposure tuning.

**FIGURE 4.** Learning curves of the competing autofocusing methods, demonstrating faster convergence of DASHA. Inset compares the three loss functions.
library [48] on OS Windows 10 64 bit, 16 GB RAM, CPU Core i7 3.6 GHz.

For training either full DASHA model or the agents separately, the PPO was performed with the following parameters: the number of workers 1, the entropy coefficient 0.01, the rollout fragment length 128, the train batch size 128, the SGD minibatch size 128, the learning rate $25 \times 10^{-5}$. We use pre-trained models, like YOLOv5 (600 object categories), with no modifications or re-training (‘off-the-shelf’ or ‘intact’). We detect 4K data points in 8 categories (cars, fruits, people, etc.), with at least 400 instances acquired in each category. Required staticity is set by AF speed (ours is the fastest in Table 1) in training RL; and by the lens response of the servomotors in inference (nearly instantaneous). We use bounding boxes found by the pre-trained models with detection certainty (Fig. 1 in the supplement).

**A. SINGLE AGENT TRAINING**

Prior to training the multi-agent RL, we trained single agents separately to gauge their stand-alone capacity to autofocus and to control the exposure. Besides enabling us to assess the ablation effect, this study also allows for proper comparison.
FIGURE 8. DASHA’s simultaneous optimization of the displacement of the lens $\Delta f$ (in mm) and of the exposure time $\Delta t_{ex}$ (in ms).

TABLE 1. Comparison of existing RL-based autofocus methods: key practical measurements.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IQA</td>
<td>RB</td>
<td>RB</td>
<td>NR</td>
</tr>
<tr>
<td>AF time (sec)</td>
<td>58</td>
<td>366</td>
<td>10</td>
</tr>
<tr>
<td>AF steps</td>
<td>12.5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Virtual pre-train</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
</tr>
<tr>
<td>DF</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
</tbody>
</table>

Note: IQA – Image Quality Assessment, RB – reference-based, NR – no-reference. AF time – average time to find focus, AF steps – the average number of steps to find focus. Virtual pre-train – a pre-trained agent before the online work with hardware, DF – deep image features (latent space).

get certain scenes in focus. Whereas, our model, once trained, required 2 steps on average to find the focus regardless of the arrangement of the objects (similarly averaged over 10 runs). Virtual pre-training is another feature in the benchmark [12] that we avoided altogether, because the Deep Image Features (the latent space) is a more efficient way to represent the distribution of data.

2) EXPOSURE-CONTROLLING AGENT ALONE

In this case, the agent only controls the $t_{ex}$ parameter of the camera, interacting with the environment according to Algorithm 1. Here, a variety of illumination conditions is used for training, ranging from a completely dark room ($E_v = 13$ lx) to a bright lighting ($E_v = 300$ lx). $E_v$ is adjusted every 30 min with an increment of $\delta E_v = 10$ lx in a looped cycle, to make the model learn the varying ambient conditions. The outcome can be compared with the ‘brute force’ embedded AutoExposure method from the Basler camera. We confirmed that the RL agent functions similarly to the embedded software without any exposure metering or without the phase difference elements (Fig. 3).

B. MULTI-AGENT TRAINING

In the full DASHA configuration, both agents are allowed to ‘learn’ and to perform together. The high-level agent controls the exposure time $t_{ex}$, and the low-level agent attempts to place the lens so that the image is in focus. The training occurred with an alternating object position (140 cm – 200 cm away from the camera) and the ambient illumination (from $E_v = 13$ lx to $E_v = 300$ lx altered in a loop), with each new change introduced every 30 minutes. On average, the training process takes about 12 hours, with typical learning curves shown in Fig. 4. Here, we used Yolo to detect objects because of its faster inference. An example of a training scene, iteratively improving with the adjustment of the exposure and the position of the lens, is shown in Fig. 5; while Fig. 7 shows a summary of a large-scale study of DASHA’s performance. The plots show $4 \times 10^3$ points in the PCA space, where each point corresponds to some image with a unique focusing distance.
VI. DISCUSSION

Notice the enhancement of the object detection capacity of the popular models by DASHA in Fig. 7, with the detection boundary moving rightwards in the plot when our agents are active (the boundary is highlighted by arrows). The ‘green’ points when the model is enabled (‘DASHA: ON’), corresponding to the images where the detection is successful, move along the distribution of the PCA data even for those images where the detection initially failed (compare to ‘DASHA: OFF’).

As such, this reports the aforementioned bi-directional enhancement of the capacity of the hardware-software tandem: the hardware ‘learns’ to adjust itself, using the outcome of the detection software; whereas, the detection models themselves begin to operate beyond their original limits (i.e., detecting dark and blurry scenes becomes easier). At different $t_{\text{ex}}$, the SOTA methods detect objects differently, which explains the importance of introducing an agent that controls $t_{\text{ex}}$ in the camera alongside the lens focusing agent.

Possible limitations: detection models can misclassify or miss objects, unless controlled by a targeted category search (Fig. 6). DASHA performs similarly to other passive AF methods in unusual and abstract scenes without detectable objects (like focusing on the sky or a blank wall; all passive methods would be bad at that).

Additionally, we performed a series of inference experiments for various ambient illuminations and various apertures of the camera (Fig. 8), confirming that the system is robust with regard to different depth of the field and ambient illumination conditions. Note how the system learned to compensate for the extra light entering the frame either by subtracting some ms of the exposure time or by proper re-focusing.

However, given that it relies on the latent representation, in some cases in Fig. 8, it decides to blur the image on purpose to make up for the missing light. These are the actions learned by DASHA which maximize the no-reference image quality. We speculate that the detection models at the core of our method are not that dependent on the image quality, being capable of functioning well even with a limited number of the crucial features pertinent to the detectable object. Remarkably, this trait is in no way an obstacle for the hardware to perform the autofocusing.

Visual examples of indoor and outdoor scenes, together with the parameters learnt by the camera and the lens, are presented in Fig. 9. The first row shows the initial images acquired by the camera at the beginning of the imaging process. The second row shows the same scenes marked as suitable according to the RL algorithm. These examples demonstrate how the over-/under-exposed and the blurry images are efficiently adjusted by DASHA, resulting in a successful object detection through the no-reference image quality assessment and the hardware tuning.

VII. CONCLUSION

In this work, we proposed a new approach to the problem of auto-tuning imaging equipment using hierarchical reinforcement learning. This methodology allows for correcting the improper focusing under various illumination conditions by employing two decentralized interacting agents that control the settings of the camera and the lens. The proposed way of autofocusing relies on the latent feature vector of the live image scene, being the first such method to auto-tune a camera without reference or calibration data.

The system proved efficient for detecting objects in the dark and the blurry initial states when three SOTA object detection methods originally failed to function. Being fast to learn (Fig. 4), our algorithm preserves the hardware from overheating, requires no active or phase-detection elements, and is functional for a range of camera apertures (i.e., robust to the depth of the field variation). Decentralized HRL...
reduces the load on the hardware during training, because it eliminates the need to engage the focusing servomotors until the electronic settings of the camera are adjusted to yield a well-exposed scene with sufficient object contrast.

One possible limitation of the current study is that we considered focusing only on the natural scenes (those that the models like Yolo, Retina, etc. are acquainted with). Standard domain adaptation and transfer learning methods should mitigate the problems of applying our method to objects of a completely different nature (e.g., cells in microscopy). Also, although our initial study partially explains the decisions made by DASHA (Fig. 8), an in-depth interpretability study is to be conducted, using such feature-based feedback solutions as Grad-CAM [50] or such targeted filtering techniques as GAF [51]. We envision simple integration of our framework with a wide range of consumer cameras and motorized lenses, enabling a widespread improvement both to the imaging hardware and to the embedded object detection models.

REFERENCES


[50] OLEG Y. ROGOV received the M.Sc. degree in physics from Lomonosov Moscow State University, Moscow, Russia, and the Ph.D. degree in mathematical modeling and physics from the Russian Academy of Sciences, Moscow. From 2015 to 2019, he was a Research Engineer with the Keldysh Institute of Applied Mathematics, Moscow. He is currently a Research Scientist with the Computational Imaging Group, Skolkovo Institute of Science and Technology (Skoltech). His current research interests include among others, evolutionary algorithms and deep learning, information fusion and decision making, and data science (fundamentals of imaging system design and performance evaluation, signal processing, detection, and estimation).

[51] ANNA ANIKINA received the bachelor’s degree in applied physics and mathematics from the Moscow Institute of Physics and Technology, Dolgoprudny, Russia, in 2018, and the M.Sc. degree in information science and technology from the Skolkovo Institute of Science and Technology (Skoltech), Moscow, Russia, in 2020. She is currently pursuing the Ph.D. degree with the University of Copenhagen, Denmark. Her research interests include machine learning and medical applications.

[52] DMITRY V. DYLOV (Member, IEEE) received the M.Sc. degree in applied physics and mathematics from the Moscow Institute of Physics and Technology, Moscow, Russia, in 2006, and the Ph.D. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2010. He is currently an Associate Professor and the Head of the Computational Imaging Group, Skolkovo Institute of Science and Technology (Skoltech), Moscow. His research interests include computational imaging, computer/medical vision, and the fundamental aspects of image formation.