

# Efficient Navigation of Colloidal Robots in an Unknown Environment via Deep Reinforcement Learning

Yuguang Yang, Michael A. Bevan, and Bo Li\*

Equipping micro-/nanoscale colloidal robots with artificial intelligence (AI) such that they can efficiently navigate in unknown complex environments can dramatically impact their use in emerging applications such as precision surgery and targeted nanodrug delivery. Herein, a model-free deep reinforcement learning algorithm is developed that trains colloidal robots to efficiently navigate in unknown environments with random obstacles. A deep neural network architecture is used that enables the colloidal robots to mimic animal navigation decision-making by directly processing raw sensor input and decomposing long-range navigations to short-range ones. The trained robot agents learn to make navigation decisions regarding both obstacle avoidance and travel time minimization, based solely on local sensory inputs without prior knowledge of the global environment. Such agents with biologically inspired mechanisms can acquire competitive navigation capabilities in large-scale, complex environments containing obstacles of diverse shapes, sizes, and configurations. Herein, the potential of AI to enable colloidal robots in extensive applications is illustrated.

## 1. Introduction

Self-propelled colloidal particles have recently demonstrated great promise as micro-/nanorobots capable of functioning in complex confined and crowded microenvironments.<sup>[1]</sup> In potential real-world applications (e.g., nanodrug delivery, precision surgery, environmental remediation, and machines<sup>[2–12]</sup>), micro-/nanorobots are confronted with navigation challenges, including long-distance travel (e.g., travel in tissue, soil, and vasculature), unknown or spatiotemporally changing environment abundant with obstacles and dead ends, and additional time

and fuel constraints. Beyond developing sophisticated micro-/nanorobot systems that have more efficient transport mechanisms and sensing capabilities,<sup>[4,13–15]</sup> efforts have also been directed toward developing better navigation strategies.<sup>[16–19]</sup> Examples include the application of a Markov decision process framework<sup>[20]</sup> and a variational Fermat's principle<sup>[21]</sup> to compute optimal navigation paths in mazes and flow fields. However, these methods generally require pre-existing knowledge of the entire environment, which is generally unavailable in the vast majority of expected applications, and even when available, the computational cost becomes prohibitive for large-scale navigation.


Animals and humans are capable of effortlessly navigating and exploring in unknown, complex, visually rich environments (e.g., cities, fields, mountains, sea,

space, etc.).<sup>[21]</sup> An adult in an unfamiliar city has little difficulty traversing city blocks and exploiting shortcuts when possible. In the context of colloidal robot navigation in an unknown obstacle field, here we aim to overcome navigation challenges by equipping colloidal robots with human-like decision capabilities, or artificial intelligence (AI), using deep reinforcement learning (DRL), which is a subset of AI algorithms inspired by the human learning and decision-making process. DRL has recently achieved remarkable performance, even superhuman performance, in various domains that require sequential decision-making, ranging from games<sup>[19,22]</sup> and robotics<sup>[23]</sup> in traditional computer science to novel applications such as drug design,<sup>[24]</sup> chemical reaction optimization,<sup>[25]</sup> and medical treatment.<sup>[26]</sup> The usage of deep neural networks in DRL, particularly the convolutional neural network,<sup>[27]</sup> has significantly expanded the success of traditional reinforcement learning to challenging visual domains like robotics and self-driving cars.<sup>[28,29]</sup>

In this study, we use a deep neural network architecture<sup>[30]</sup> (Figure 1a) that aims to minimally mimic the animal navigation decision-making system in two key aspects. First, we use convolutional neural layers,<sup>[29,31]</sup> which mimic the animal visual system<sup>[32]</sup> by directly processing high-dimensional raw sensory information (i.e., obstacle images). Raw sensory information has been shown to enable learning of efficient representations from high-dimensional observations and facilitate the generalization of experiences to new situations (i.e., unknown environments).<sup>[29,33]</sup> Second, we dynamically transform distant targets to local short-range proxy targets, which mimics animal navigation

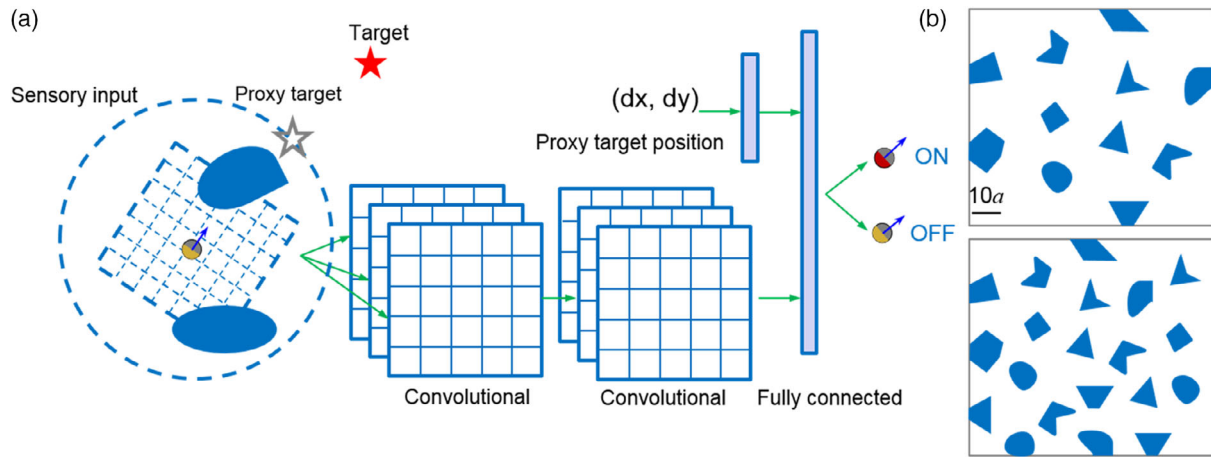
Dr. Y. Yang, Prof. B. Li  
Institute of Biomechanics and Medical Engineering  
Applied Mechanics Laboratory  
Department of Engineering Mechanics  
Tsinghua University  
Beijing 100084, China  
E-mail: libome@tsinghua.edu.cn

Dr. Y. Yang, Prof. M. A. Bevan  
Chemical & Biomolecular Engineering  
Johns Hopkins University  
Baltimore, MD 21218, USA

 The ORCID identification number(s) for the author(s) of this article can be found under <https://doi.org/10.1002/aisy.201900106>.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim. This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

DOI: 10.1002/aisy.201900106



**Figure 1.** a) The main body of neural network architecture used in our DRL algorithm. The details of the architecture and other units, such as nonlinearity, normalization, and pooling, are in Experimental Section. Two streams of sensory inputs are fed to the neural network, including a pixel image ( $30a \times 30a$ ) of the particle's neighborhood fed into a convolutional layer and the target's position fed into a fully connected layer. A distant target will be transformed to a local proxy target. The network will output two  $Q$  values associated with the ON and OFF actions. b) The two obstacle environments used to train the colloidal robot agent.

behaviors that decompose long-range goals into a series of shorter-range subgoals (on the scale of visual perception limit).<sup>[29,34]</sup>

We train the neural network (Figure 1a) via a model-free (i.e., data-driven) learning framework, which will allow our approach to be applicable to a broad range of colloidal robots exhibiting different dynamics.<sup>[35,36]</sup> Our DRL algorithm also contains critical enhancements to stabilize the learning process, as the strong stochasticity in colloidal robots arising from Brownian motion can make the training process unstable and divergent. We deploy the robot agent to navigate across diverse environment structures (e.g., different obstacle shapes, sizes, spacing, and target distances) (Figure 1b) and train the neural network directly from extensive navigation trajectory data, involving sequences of observations, self-propulsion decisions, subsequent observations, and reward signals. Ultimately, the trained agent learns to make self-propulsion decisions to circumvent obstacles and reach targets, accommodating the unique stochastic dynamics of the robot. Because our DRL algorithm only relies on local information, the agent is inherently trained to acquire short-range navigation capability; however, the architecture (Figure 1a) allows it to navigate long distances by implicitly decomposing them into a series of short-scale navigations.

In a series of navigation tests, we show in the following section that the trained agent can efficiently navigate in large, complex environments with obstacles of unknown shapes and arrangements. We demonstrate that the neural network can learn effective representations of observations and thus shed light on the successful navigation performance. Our results demonstrate a general framework to train an intelligent colloidal robot to master the rule of efficient navigation based on local visual information, in contrast to developing algorithms to compute navigation strategies on a case-by-case basis. Such an approach is promising to overcome the navigation challenges in emerging future applications of colloidal robots.

## 2. Model and Algorithm

### 2.1. Colloidal Robot Model

We model colloidal robots as Brownian-type self-propelled particles that can control their self-propulsion speed yet without direct control over their orientation. Although direct orientation control for some types of self-propelled particles has been realized using magnetic field,<sup>[4]</sup> we herein consider the more general and more challenging situation that particles cannot directly control their orientation and have to smartly exploit Brownian motion and their self-propulsion to realize efficient navigation. The equation of the model of such a colloidal robot confined on a plane is given by

$$\begin{aligned} \partial_t x &= \xi_x(t) + v \cos \theta \\ \partial_t y &= \xi_y(t) + v \sin \theta \\ \partial_t \theta &= \xi_\theta(t) \end{aligned} \quad (1)$$

where  $x$ ,  $y$ , and  $\theta$  denote the position and orientation,  $t$  is time, and  $v$  is propulsion speed taking binary values of 0 and  $v_{\max}$  as the control inputs. Brownian translational and rotational displacement processes  $\xi_x$ ,  $\xi_y$ , and  $\xi_\theta$  are zero-mean Gaussian noise processes with variances  $\langle \xi_x(t) \xi_x(t') \rangle = 2D_t \delta(t - t')$ ,  $\langle \xi_y(t) \xi_y(t') \rangle = 2D_t \delta(t - t')$ , and  $\langle \xi_\theta(t) \xi_\theta(t') \rangle = 2D_r \delta(t - t')$ , respectively, where  $D_t$  is the translational diffusivity and  $D_r$  is the rotational diffusivity. All lengths are normalized by particle radius  $a$ , and time is normalized by  $\tau = 1/D_r$ . The control update time is  $t_c = 0.1\tau$ , the integration time step  $\Delta t = 0.001\tau$ ,  $v_{\max} = 2a/t_c$ , and  $D_t = 1.33a^2 D_r$ . Notably, our DRL algorithm is model free and can be readily applied to other dynamic models controlled by specific physics.<sup>[15,37–39]</sup>

### 2.2. Deep Reinforcement Learning

We denote the particle state at time step  $n$  by  $s_n = (x_n, y_n, \theta_n)$ . The observation  $\phi(s_n)$  at  $s_n$  consists of the pixelated image of

the particle's neighborhood and the target position  $(x^t, y^t)$ , as shown in Figure 1a. We seek an optimal control policy,  $\pi^*$ , which maps an observation  $\phi(s_n)$  to a self-propulsion action (i.e., OFF or ON) to maximize the expected reward accumulated during a navigation process,  $\mathbb{E} \sum_{n=1}^{\infty} \gamma^n [R(s_n)]$ , where  $R$  is the one-step reward function and  $\gamma$  is set to 0.99 to encourage the agent to value rewards in the distant future. To seek an optimal policy minimizing arrival time,<sup>[40,41]</sup>  $R$  is set equal to 1 for all states that are within a threshold distance 2 to the target and 0 otherwise. The optimal control policy is obtained by training the neural network (Figure 1a) to approximate the optimal state-action value function (known as the  $Q^*$  function) given by

$$Q^*(\phi(s), v) = \mathbb{E}[R(s_1) + \gamma^1 R(s_2) + \gamma^2 R(s_3) + \dots | \phi(s_0)] \\ = \phi(s), v_0 = v, \pi^* \quad (2)$$

which is the expected sum of rewards along the process by following the optimal policy  $\pi^*$ , after observing  $\phi(s)$  and self-propelling with speed  $v$ . The neural network contains convolution neural layers to process sensory information of the particle neighborhood, represented by a  $W \times W$  binary image ( $W = 30$ ) and a fully connected layer to process the target's position in the particle's local coordinate system. Distant targets (distance  $> W$ ) are transformed into local proxy targets by projecting onto a circle of radius  $W$  centering on the particle (Figure 1a). The neural network finally outputs the two  $Q^*$  values associated with the ON and OFF actions.

The local neighborhood sensory input directly impacts the learned navigation strategies and the computational cost during the training stage. The sensory input is characterized by two parameters, vision field size  $W$  (i.e., the size of the binary image) and resolution. The vision field size determines the range the robot can perceive, and the resolution determines the minimum obstacle feature can be detected.  $W$  has to be greater than twice the largest obstacle size such that the agent can learn robust navigation strategies. This is because the agent makes self-propulsion decisions based solely on current observations without access to historical information (i.e., the agent is memoryless). A smaller  $W$  can cause ambiguity in the decision-making process when the agent is trying to circumvent large obstacles in a bounded environment (see Figure S1, Supporting Information, for illustrating examples). High resolution will increase the computation cost, whereas low resolution can cause delays as robots are unable to detect small trapping obstacle features. A reasonable criterion of resolution, characterized by pixel length scale  $U$ , is that  $U$  is comparable to the diffusion distance within unit characteristic time  $\sqrt{2D_t\tau} \approx 1.6a$  and  $U$  is smaller enough to detect trapping concave geometry features. We choose  $U = a$  in the present study.

We train the neural network through multiple episodes of navigation in free space and two obstacle-present environments (Figure 1b) (see Experimental Section for details). Each episode is initiated at a random starting position of the agent and random target; the episode ends when the agent arrives at the target or when the maximum control step is reached, whichever comes first. Extensive navigation data are necessary to enable the agent to learn robust navigation strategies in various scenarios (different obstacle shapes, sizes, spacing, and target locations). We use

the canonical deep Q-learning algorithm<sup>[21]</sup> to iteratively improve the estimate of  $Q^*$ , with several critical enhancements<sup>[34,42]</sup> to improve sample efficiency and the rate of convergence. With the estimated  $Q^*$  function, the optimal propulsion decision at a given observation  $\phi(s)$  is given by  $\pi^* = \operatorname{argmax}_v Q^*(\phi(s), v)$ .

### 3. Results and Discussion

#### 3.1. Free Space and Simple Obstacles

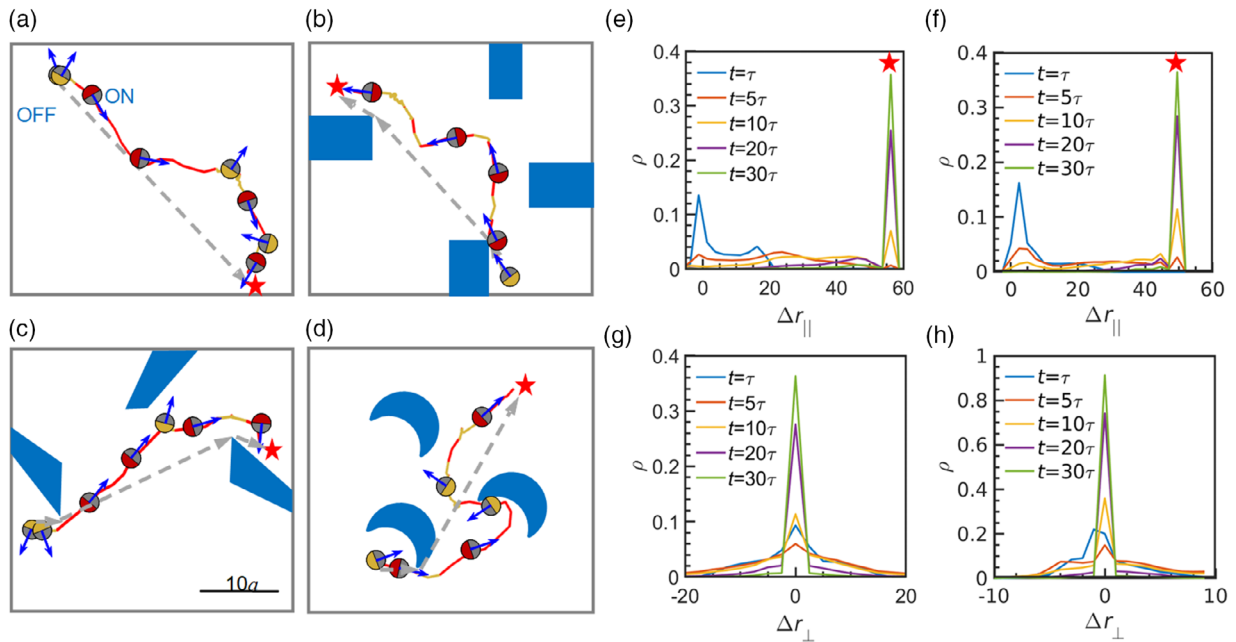
The trained DRL particle agent can efficiently navigate in free space and unknown bounded obstacle environments not observed during its training process (Figure 2a–d). In free space navigation (Figure 2a), the navigation strategy derived from the learned optimal  $Q^*$  function is similar to previous studies<sup>[18,43,44]</sup> and can be summarized approximately as

$$\pi^*(s) = \begin{cases} v_{\max}, & d_n \in [d_c, \infty) \\ v_{\max}, & d_n \in [0, d_c), \alpha_n \in [-\alpha_c, \alpha_c] \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

where  $d_n$  is the projection of the target-particle vector onto the orientation vector  $\mathbf{n} = (\cos\theta, \sin\theta)$ ,  $\alpha_n$  is the angle between target-particle distance vector and  $\mathbf{n}$ , and parameters  $d_c$  and  $\alpha_c$  are fitted to be  $\approx 0.4 v_{\max}\tau$  and  $\approx 30^\circ$ . Despite the inability to directly control orientations, the agent exploits Brownian rotation to realize desired reorientation. Intuitively, the particle agent has learned an “orientation timing” strategy to where it will propel itself when the target locates in front of the particle or otherwise waits for the favorable orientation to be sampled by Brownian rotation.

We further consider navigating in environments with obstacles of different shapes and arrangements unseen in training, including regular rectangular obstacles (Figure 2b), re-entrant rectangular obstacles (Figure 2c), and moon-shaped obstacles (Figure 2d). These obstacles, with different shapes, size, and spacing, are designed to block paths to targets or trap particles via concave geometries (obstacle–wall arrangements in Figure 2b,c or obstacle geometry itself in Figure 2d). Successful navigation trajectories indicate that the trained particle agent has learned navigation “knowledge” generalizing beyond the training environments (e.g., Figure 1b). Specifically, the agent learns to wait for desired favorable orientations such that it can propel itself forward without hitting convex blocking obstacles or getting trapped in dead ends (Figure 2b,c). The agent will also temporally propel itself away from the target to get out of concave traps if it accidentally gets trapped (Figure 2d). In general, the agent learns to avoid the obstacles and approximately follow the shortest geometric path toward the target, even when only local information is accessed.

We examine the distribution of displacements of the trained agent during the navigation processes in Figure 2a,b. In collecting the statistics, trajectories start at the same position but with randomly sampled initial orientations. The displacements are projected along and perpendicular to the shortest geometric path (represented by the direction vectors connecting from the initial position to the target), which are used to capture the navigation progress and the deviation from the ideal direct path, respectively. As shown in Figure 2e,f, for navigation in both free space



**Figure 2.** Navigation trajectories of a DRL trained agent in different bounded environments, including a) free space, b) rectangle obstacles, c) re-entrant rectangle obstacles, and d) moon-shaped obstacles. Targets are denoted by red stars. Trajectories are colored red when self-propulsion is turned ON and are colored gold when self-propulsion is turned OFF. Gray dash vectors are global shortest paths from the starting point to the target. e–h) The distributions of parallel and perpendicular components of displacements when navigating in (e,g) free space and in (f,h) rectangle-obstacle environment.

and rectangle-obstacle environments, the displacement distributions exhibit two modes at  $t = \tau$ : 1) a near mode located at 0 due to trajectories with unfavorable initial orientations waiting at the initial position for favorable orientations from stochastic rotation and 2) a far mode located at  $\approx 0.75 v_{\max} \tau$  due to trajectories with favorable initial orientation rapidly self-propelling. At longer times, the near mode continues to spread out, and the far mode propagates toward the target, indicating that the trained agent can readily propel itself to get closer to the target when favorable orientations and positions appear via Brownian motion. In addition, the perpendicular displacement (i.e., the vertical deviation to the ideal path) distributions exhibit a narrow peak around 0 but with tails extending to  $\approx v_{\max} \tau$  in free space and  $\approx 0.5 v_{\max} \tau$  in obstacle-present environments, indicating that the DRL trained agent can usually maintain a close distance to the ideal path, but occasional large deviation can also occur (Figure 2g,h).

### 3.2. Complex Obstacles

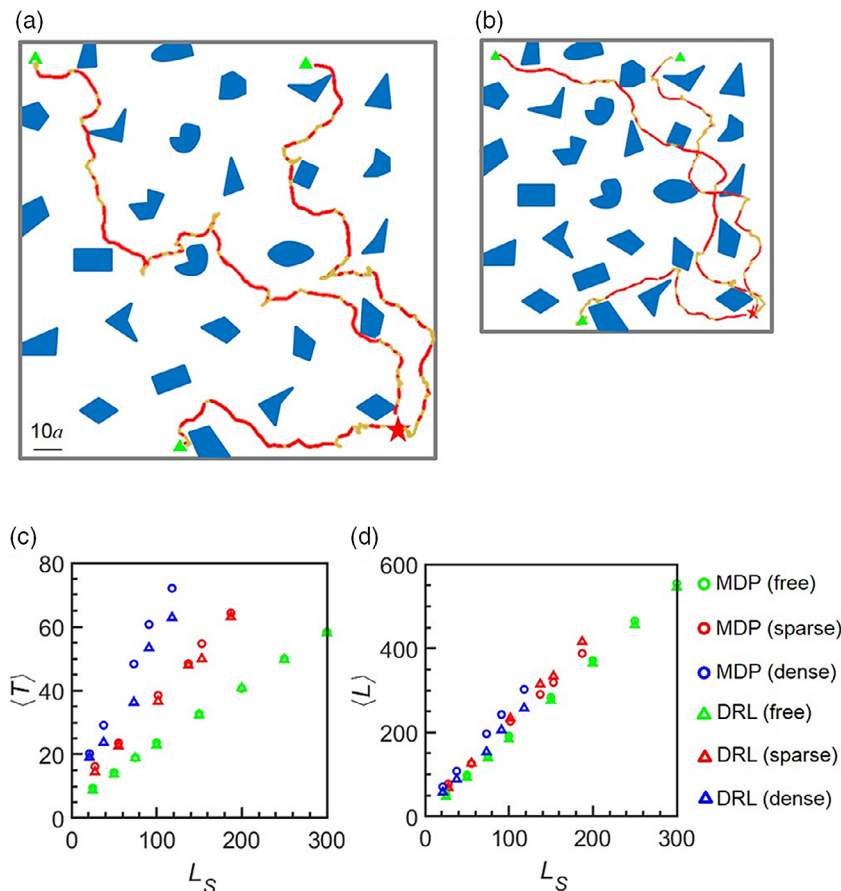
After understanding the navigation behavior of trained agents in free space and simple obstacle environments, we now analyze its navigation performance in larger and more complex unknown obstacle environments (Figure 3a,b). Both environments contain obstacles of the same shape, but the environment in Figure 3b is more crowded due to reduced space between obstacles. Representative trajectories show that the trained agent, starting at different locations, is capable of circumventing obstacles in the way and reaches the target. In a larger and more complex environment, the agent is repeatedly using the same navigation strategies as it does in the smaller and simpler environments

(Figure 2a–d) to “smartly” self-propel only at favorable orientation to avoid the obstacles and get out of traps.

We now quantify navigation performance across different length scales (Figure 3a,b) by benchmarking the navigation performance with a model-based Markov decision process (MDP) algorithm (see Supporting Information).<sup>[20,21]</sup> As a basic tool widely used in optimal sequential decision-making problems, MDP algorithm can compute the optimal navigation strategy using a discretized particle dynamic model. But the MDP algorithm cannot be applied to unknown environment navigation (as it requires prior knowledge of the environment) or large-scale navigation tasks due to prohibitive computational cost. The performance of DRL and MDP algorithms is first compared based on mean first arrival time  $\langle T \rangle$  (see Experimental Section for comparison setup). For trained agents navigating different distances in both free space and sparse-obstacle environments (Figure 3a,b), the DRL and MDP agents have similar mean arrival time (Figure 3c). Surprisingly, the DRL agent slightly outperforms MDP agent by 10% in the dense-obstacle environment (Figure 3d). This is probably because the DRL algorithm is optimizing on a continuous state space, whereas the MDP algorithm is optimizing on a discretized one; thus, in crowded environments that require frequent and careful steering, DRL agent is able to make more accurate decisions based on continuous state observation.

We further compare the traveled path length between DRL and MDP agents where the traveled path length of a trajectory is defined by  $L = \sum_{i=1}^{N-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$ , where  $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$  are particle positions observed at consecutive control update times. The traveled path length can provide information on how agents plan their route to get to the target. Since the MDP agent seeks a global optimal performance





**Figure 3.** a,b) Two unknown large-sized test environments with sparse obstacles (a) and dense obstacles (b) used for performance benchmark. The obstacles in environment (b) have the same shape as in (a) but with reduced space between obstacles. Solid curves are example trajectories of a DRL trained agent navigating from a different starting position (green triangles) to the target (red star). c,d) Mean first arrival time  $\langle T \rangle$  and mean traveled path length  $\langle L \rangle$  are compared for a DRL and an MDP trained agent navigating between different starting points and targets in free space and obstacle environments. Choices of starting points and target are given in Experimental Section and Figure S2, Supporting Information. These different navigations can be parameterized by the shortest geometric path length  $L_s$  from starting points to the target.

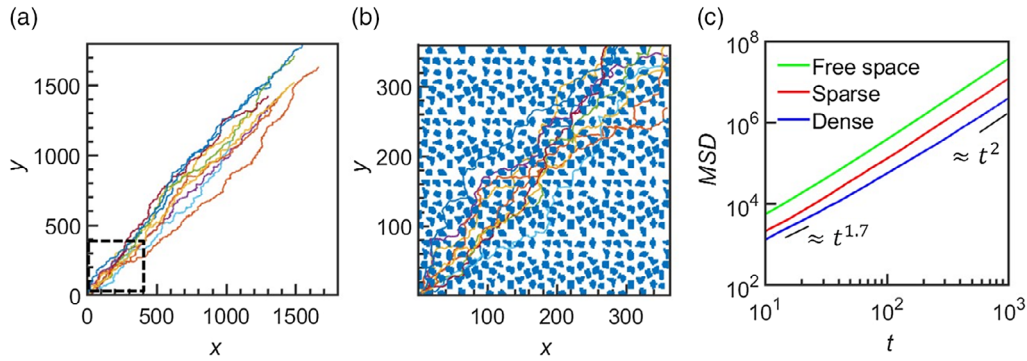
by planning the global shortest paths according to the pre-existing map,<sup>[21]</sup> a much larger path length, compared with the MDP agent, indicates that the DRL agent plans inferior paths to reach the target. Figure 3d shows that the MDP and DRL agents have comparable mean traveled path length  $\langle L \rangle$ , thereby indicating that the DRL agent can also plan efficient routes toward the target even based solely on local information.

The DRL agent's average travel speeds  $v = \langle L \rangle / \langle T \rangle$  are fitted to  $\approx 0.36 v_{\max}$  and  $\approx 0.22 v_{\max}$  for navigation in the sparse- and dense-obstacle environments, respectively, smaller than the fitting of  $\approx 0.5 v_{\max}$  in free space. Because circumventing obstacles requires frequent reorientation and thus more waiting for the desired orientation, the travel speeds will reduce with increasing crowdedness of the environment.

### 3.3. Infinitely Large-Obstacle Environments

To ultimately evaluate the navigation capability in infinitely large environments, we deploy trained agents to perform directed transport in free space and periodic environments containing

both sparse obstacles and dense obstacles (Figure 3a,b, also see Experimental Section). In all three cases, the trained agents can successfully travel along specified directions (i.e., horizontal direction in free space and  $45^\circ$  direction in obstacle environments). Particularly, even in the crowded dense-obstacle environment, the trained agents are able to circumvent all obstacles in the way and approximately maintain the ideal directed path (i.e., the radial path in the  $45^\circ$  direction) (Figure 4a,b). We further perform the mean squared displacement (MSD) analysis on the navigation trajectory (Figure 4c), which provides insights into the combined effect of control strategy and environment on the motion modes of the particle agent. At both shorter and longer time scales  $t$  compared with  $\tau$ , the MSD of the trained particle agent navigating in free space follows  $\text{MSD}(t) \approx t^2$ , as the navigation strategy produces intermittent directed transport. As a comparison, a constantly self-propelled particle will become an effective random walker at a longer time scale<sup>[45]</sup> and have  $\text{MSD}(t) \approx t$ . Surprisingly, for navigations in the obstacle environments, the DRL agent still displays approximately  $\text{MSD}(t) \approx t^2$  at a longer time scale, although at a smaller time scale it displays



**Figure 4.** a) Ten representative trajectories (lasting  $1000\tau$ ) during directed transport along  $45^\circ$  direction in a periodic dense obstacle environment (obstacles not shown). b) Magnified dashed region in (a) with obstacles shown. c) MSD for a DRL trained agent in free space, sparse-obstacle, and dense-obstacle environments.

$MSD(t) \approx t^{1.7}$ , where the smaller exponent arises from detours when circumventing obstacles. For the same reason, the transition from  $t^{1.7}$  to  $t^2$  occurs at a longer time for the environment with dense obstacles ( $\approx 100\tau$  vs  $\approx 20\tau$ ). In short, the trained agent can perform directed transport in obstacle-present environments as in obstacle-absent ones (i.e., without getting trapped), although at a reduced speed.

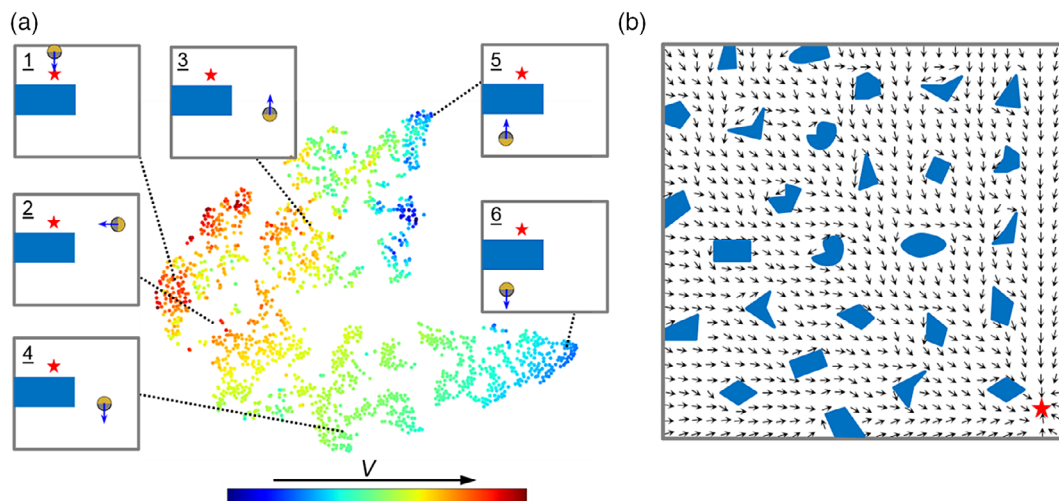
### 3.4. Model Analysis

Finally, we examine the representations learned from high-dimensional sensory inputs by the neural network to understand the successful performance in the previous navigation tasks. We consider an unknown environment with only one obstacle and apply the nonlinear dimension reduction t-distributed stochastic neighbor embedding (t-SNE) algorithm<sup>[21]</sup> to embed the learned representations in the last hidden layer into a 2D plane (Figure 5a). Each 2D point is colored by the state value defined by

$$V(s) = \max_v Q^*(\phi(s), v) = \mathbb{E} \left[ \sum_{n=1}^{\infty} \gamma^n R(s_n) | s_0 = s, \pi^* \right] \quad (4)$$

where  $V(s)$  can be interpreted as the expected maximum total reward a robot can achieve when it initializes with state  $s$  and follows the optimal navigation policy  $\pi^*$ . A particle state with higher  $V$  indicates that a particle starting at this state can arrive at the target faster than other states with lower  $V$ .

As shown in Figure 5a, high-dimensional sensory observations at different particle states are embedded in the 2D plane apparently based on the shortest path distance to the target location (in the horizontal direction) and the particle's orientation (in the vertical direction). In the left end, particle states with the shortest distance and favorable orientations are assigned with the highest state value (Figure 5a<sub>1</sub>), whereas particle states with a larger distance are assigned with lower state values (Figure 5a<sub>2</sub>). Particle states with intermediate range to the target are placed in the middle with intermediate state values and locate at the upper part if the particle favorably orients to the target (Figure 5a<sub>3</sub>), or



**Figure 5.** a) 2D t-SNE embedding of the representation in the last hidden layer of the neural network in a single-obstacle navigation task. Every point corresponds to a 2D representation of observations at admissible particle states  $(x, y, \theta)$  discretized with step size  $(1, 1, \pi/8)$ . Points are colored by state value  $V = \max_v Q^*(\phi(s), v)$  (Equation (4)). b) Implied effective navigation direction from the gradient of orientation averaging optimal state value (Equation (5)) with the target denoted by red star.

lower part if the colloidal particle orients opposite to it (Figure 5a<sub>4</sub>). For particles located the furthest from the target due to blockage of the obstacle, their observations are assigned to the lowest state value and placed on the rightmost end at either upper or lower wings (Figure 5a<sub>5</sub>,a<sub>6</sub>). While  $Q^*$  function assigns a lower value to states with unfavorable orientations for particles within an intermediate range, it assigns similar state values to distant particle states regardless of their orientations; this is because initial orientation does not add appreciable value to long-distance navigation. In short, as demonstrated in the toy example (Figure 5a), the reward signals have shaped the neural network to learn representations from observations that can distinguish whether one particle state is more favorable than the other via state values.

Another way to understand the learned navigation strategy is by visualizing the normalized gradient vector (see Supporting Information) of orientation averaging optimal state values  $V_{XY}(x, y)$  defined by

$$V_{XY}(x, y) = \frac{1}{8} \sum_{i=1}^8 V\left(x, y, \frac{i\pi}{4}\right) \quad (5)$$

Because the trained agent will take actions to move from low-value states (e.g., particle states with unfavorable orientation and far away from the target) to high-value states (e.g., particle states with favorable orientation and smaller distance to the target), the gradient of  $V_{XY}(x, y)$  represents the effective navigation direction that the trained agent wishes to move. In other words, if the agent orients along the effective navigation direction, it will self-propel; otherwise, it will wait for the orientation from Brownian rotation sampling. In Figure 5b, the effective navigation direction in the complex test environment clearly shows the agent's intent to circumvent all obstacles (even if they are not observed by the agent before) and follow paths directed toward the target, which is consistent with navigation trajectories shown in Figure 3a,b and Figure 4b.

## 4. Conclusion and Outlook

In summary, we have developed a bioinspired DRL methodology to tackle the challenge of efficient navigation of colloidal robots in an unfamiliar complex environment. By training the agent through extensive navigation trajectories in diverse navigation scenarios with varying obstacles shapes, size, spacing, and target distance, the agent is able to derive effective navigation strategies generalizing beyond the training environments. Using t-SNE, we show that the neural network can learn useful representations of visually rich, high-dimensional sensory input and use them to obtain generalizable navigation strategies.

Our DRL algorithm provides a general model-free framework that applies to navigation of colloidal robots with different dynamical models due to different actuation mechanisms,<sup>[4,46,47]</sup> translation-rotation hydrodynamic coupling,<sup>[48]</sup> etc. Our DRL architecture can also be used as a basic building block to construct more complex learning architectures for navigation tasks in more challenging environments, including incorporating additional visual channels for navigation in flow fields,<sup>[41]</sup> adding memory module<sup>[49]</sup> for navigation in nonstationary

environments with limited visibility, using 3D convolutional layers<sup>[50]</sup> for 3D navigation, extending to continuous control DRL<sup>[51]</sup> for high-precision localization tasks, and building hierarchical neural networks<sup>[52]</sup> for navigation in environments with multiple-scale obstacle features. Our algorithm can also be extended to a multiagent system<sup>[48,53,54]</sup> to control multiple robots to cooperate on tasks and assemble to nonequilibrium machines and devices<sup>[55]</sup> or applied as a general end-to-end controller for controlling stochastic colloidal assembly.<sup>[56]</sup> Ultimately, our DRL algorithm allows to be integrated with experimental systems as it can directly process raw sensor inputs of microrobot systems (e.g., microscope).<sup>[57]</sup> Possible data scarcity and noisy-state estimation issues in real experiments can be addressed by techniques such as imitation learning,<sup>[58]</sup> transfer learning,<sup>[17]</sup> and the usage of memory module.<sup>[59]</sup>

## 5. Experimental Section

### 5.1. Deep Convolution Neural Network Architecture

We approximated the  $Q^*(\phi(s), v)$  function using a neural network described as follows. The observation  $\phi(s)$  consisted of two streams of inputs. The first input was the pixel-level binary sensory input of a  $30a \times 30a$  neighborhood centering on the particle and aligned with its orientation (pixel width is  $a$ , 1 denotes the presence of obstacles or out of the boundary, and 0 vice versa). The second input was the target position in the local coordinate frame of the particle (note that distant targets will be transformed to local proxy targets, as we will discuss in Section 5.2.1). The neighborhood sensory input first entered a convolutional layer<sup>[29,60]</sup> consisting of 32 filters with kernel size  $2 \times 2$ , stride 1, and padding of 1, following a batch normalization layer,<sup>[61]</sup> a rectifier nonlinearity<sup>[62]</sup> (i.e.,  $\max(0, x)$ ), and a  $2 \times 2$  of maximum pooling layer.<sup>[32]</sup> The output then entered the second convolutional layer consisting of 64 filters and same kernel, stride, and padding as the previous layer, followed similarly by a batch normalization layer, a rectifier nonlinearity, and a maximum pooling layer. The local target coordinate first entered a fully connected layer consisting of 32 units followed by rectifier nonlinearity. Then the output from the target coordinate input and the sensory input merged and entered a fully connected layer of 64 units followed by rectifier nonlinearity. The output layer was a fully connected linear layer with two outputs associated with the binary actions ON and OFF.

### 5.2. Training Algorithm and Procedures

#### 5.2.1. Sensory Input Construction

For navigations in the obstacle environments, the environment maps were represented by a binary image (1 at obstacle region) with the pixel size of  $a$ . For each particle state  $s = (x, y, \theta)$ , we constructed the observation  $\phi(s)$ , which consisted of local neighborhood sensory input and target coordinate input in the following way:

- 1) The local neighborhood sensory input was obtained by first constructing a squared window of width  $W = 30a$  centering on the particle and aligned with its orientation and then extracted a

$30 \times 30$  binary matrix from the environment maps. Our image processing procedures included resizing the local neighborhood image to the desired resolution (i.e., the actual length per pixel) and thresholding (i.e., determining if a part of an obstacle is on the pixel). We used 50% gray level as the threshold. 2) For a given target position, we first transformed the target position to a local proxy target position in the following way: if the target's distance to the particle is greater than  $W = 30a$ , its local proxy target position is the projection of the original target on a circle of radius  $W$ ; otherwise, local proxy target is the original target itself. Local proxy target positions were further transformed to the local coordinate system of the particle before feeding into the neural network.

### 5.2.2. Obstacle Representation and Collision Dynamics

We directly converted environment maps to pixel images using an image processing software. Obstacle regions had a value 1, whereas free space regions had a value 0. Particle–obstacle collision dynamics was modeled in the following simplified way:

1) In every control time step, we evolved the particle state  $(x(t), y(t), \theta(t))$  using equation of motion (Equation (5)) for an interval of  $t_c$ . 2) If the new position of particle  $(x(t + t_c), y(t + t_c))$  from step (i), was in the obstacle region, we set its position to previous one  $(x(t), y(t))$ . But we needed to still update its orientation.

This approximation is reasonable because the particle–obstacle collision is not the dominant factor that affected the navigation process. Note that for nonspherical particles (e.g., rod shape), a better approximation is necessary since collision will affect particle orientation (see our previous study for more details<sup>[21]</sup>).

### 5.2.3. Training Algorithm

The algorithm we used to train the agent was the canonical deep Q-learning algorithm<sup>[63]</sup> plus enhancements from double Q-learning,<sup>[43]</sup> the hindsight experience replay,<sup>[44]</sup> and scheduled multistage learning. Hindsight experience  $(\phi(s_h), v_h, R(s_h), \phi(s_{h+1}))$  was created from ordinary experience  $(\phi(s_n), v_n, R(s_n), \phi(s_{n+1}))$  using the following procedures:  $\phi(s_h)$  and  $\phi(s_{h+1})$  were the observations at  $s_h$  and  $s_{h+1}$  but with the target set at particle position  $(x_{n+1}, y_{n+1})$  at step  $n+1$ ;  $R(s_h)$  was set to 1;  $v_h = v_n$ . Note that we only constructed hindsight experience from experience that did not collide with obstacles. The complete algorithm is presented later, with training parameters set according to Table 1. Our implementation of the algorithm is available at <https://github.com/yangyutu/DeepReinforcementLearning-PyTorch>

The enhancement of scheduled multistage learning consisted of the following steps:

1) At the beginning of each episode, initial particle states and target positions were randomly generated. To speed up the convergence, we used ideas from curriculum learning<sup>[64]</sup> and required that they were generated in such a way that their distance gradually increased from a small value. Then during the training process, the agent could learn the short-distance navigation before learning long-distance navigation. Mathematically, let  $D$  denote the distance between the generated initial particle position and target position, and we require  $D$  to satisfy

**Table 1.** Parameters used in training.

Parameter	Value
Training episode $N_E$	74 000 (first stage), 60 000 (second stage)
Minibatch size, $B$	64
Replay memory size, $N_M$	500 000
Target network update frequency, $C$	100
Discount factor, $\gamma$	0.99
Learning rate (Adam optimizer), $\alpha$	0.00025
Initial greedy exploration parameter	0.5
Final greedy exploration parameter	0.05
Greedy exploration parameter decay	2000
Initial target threshold, $T_s$	0.1
Final target threshold, $T_e$	1
Target threshold decay, $T_d$	2000
Max step in an episode, maxStep	500
Sensor window size, $W$	30
Hindsight experience replay frequency, $H$	1

$$D \leq S_m \times (T_e + (T_s - T_e) \exp(-\text{episode}/T_d)) \quad (6)$$

where episode is the episode number,  $S_m$  is the maximum of width and height of the training environment,  $T_s$  is the initial target threshold,  $T_e$  is the final target threshold, and  $T_d$  is the target threshold decay.

2) The whole training process consisted of two stages. The first stage lasted for 74 000 episodes, where the free space environment and the sparse-obstacle environment were randomly selected (0.2 vs 0.8 probability) to train the active particle. The second stage lasted for 60 000 episodes, where the free space environment, the sparse-obstacle environment, and the dense-obstacle environment were randomly selected (0.1 vs 0.4 vs 0.5 probability) to train the active particle. We empirically determined the training episode numbers in the first stage and the second stage by monitoring the running average reward plateaus. The typical training time on a desktop with a 1080Ti GPU was  $\approx 4$  days.

### Algorithm: Deep Q-learning with hindsight experience replay and double Q evaluation

Initialize replay memory  $M$  to capacity  $N_M$

Initialize action-value function  $Q$  with random weights  $\omega$

Initialize target action-value function  $Q'$  with weights  $\omega'$

**For** episode 1,  $N_E$  **do**

Randomly generate a particle state  $s_0$  and a target position.

Obtain initial observation  $\phi(s_1)$ .

**For**  $n = 1$ , maxStep **do**

With probability  $\epsilon$  select a random action; otherwise select  $v_n = \arg\max_v Q(\phi(v_n), v; \omega)$

Execute action  $v_n$  using simulation and get new particle state  $s_{n+1}$  and reward  $R(s_{n+1})$

Generate observation state  $\phi(s_{n+1})$  at state  $s_{n+1}$

Store transition  $(\phi(s_n), v_n, R(s_{n+1}), \phi(s_{n+1}))$  in  $M$ .



Store extra hindsight experience  $(\phi(s_h), v_h, R(s_{h+1}), \phi(s_{h+1}))$  in  $M$  every  $H$  step  
Sample random minibatch transitions  $(\phi(s_i), v_i, R(s_{i+1}), \phi(s_{i+1}))$  of size  $B$  from  $M$   
Set target value

$$y_i = \begin{cases} R(s_{i+1}), & \text{if } s_{i+1} \text{ arrives at target region} \\ R(s_{i+1}) + \gamma Q'(\phi(s_{i+1}), \arg \max_v Q(\phi(s_i), v)), & \text{otherwise} \end{cases}$$

Perform a gradient descent step on  $(y_i - Q(\phi(s_i), v_i))^2$  with respect to network parameters  $w$   
If arrive at the target, break the For loop.  
Every  $C$  steps reset  $\omega' = \omega$

**End For**

**End For**

Note that as we discussed in Section 5.2.1, distant targets were transformed to local proxy targets. The local proxy target position will change whenever the particle translates or rotates. This target shift problem will cause artifacts in the learned control policy. To remedy this undesired effect, when we executed the line “Store transition  $(\phi(s_n), v_n, R(s_{n+1}), \phi(s_{n+1}))$  in  $M$ ” in the algorithm, we needed to construct the observation  $\phi(s_{n+1})$  based on target position at the previous step  $n$  instead of the target position at step  $n+1$ .

### 5.3. Evaluation Performance

#### 5.3.1. Displacement Distribution Analysis

In the navigation example in free space (Figure 2a) and in the rectangle-obstacle environment (Figure 2b), we collected displacement statistics from 60 000 trajectories initialized at positions (5, 40) and (35, 30) and with random orientation. The targets were located at (40, 5) and (9, 5), respectively. Displacements at different observation times ( $t = 1\tau, 5\tau, 10\tau, 20\tau, 30\tau$ ) were projected along the unit vector connecting the initial position to the target position to get the parallel component  $r_{\parallel}$  and perpendicular component  $r_{\perp}$ . Particularly, displacements in the rectangle-obstacle environment were projected along the piecewise linear segment (which was the shortest geometric path) connecting the initial position to the target position to get the parallel component  $r_{\parallel}$  and perpendicular component  $r_{\perp}$ . Projections onto these piecewise linear segments were calculated using the MATLAB routine (<https://www.mathworks.com/matlabcentral/fileexchange/34869-distance2curve>).

#### 5.3.2. MSD Analysis

The MSDs for trajectories of the DRL agent and constant self-propelled agent were calculated from 300 trajectories. For navigation in free space, trajectories were initialized at a fixed location (0, 0) with random orientations and navigated toward a remote target set at (100 000, 0). For navigation in periodic obstacle environments (Figure 4), trajectories were initialized at a fixed location (5, 5) with random orientations and navigated toward a remote target set at (5760, 5760). For navigation in the obstacle environment in Figure 4a,b, we applied periodic boundary conditions on the simulation box without the hard wall. Note that we

applied the hard wall conditions in the simulation box when we performed other navigation tests.

#### 5.3.3. Mean First Passage Time and Mean Travel Path Length

The mean first passage time and mean travel path length for each navigation task were averaged from 300 trajectories. Trajectories were all initialized at different positions with random orientations. The shortest geometric paths between starting positions and the target position in obstacle environments were calculated using the fast-marching algorithm (see Figure S2, Supporting Information) (<https://www.mathworks.com/matlabcentral/fileexchange/6110-toolbox-fast-marching>).

## Supporting Information

Supporting Information is available from the Wiley Online Library or from the author.

## Acknowledgements

Support from National Natural Science Foundation of China (grant nos. 11961131005 and 11672161) is acknowledged.

## Conflict of Interest

The authors declare no conflict of interest.

## Keywords

artificial intelligence, colloidal robots, deep reinforcement learning, navigation

Received: September 9, 2019

Revised: September 28, 2019

Published online:

- [1] C. Bechinger, R. Di Leonardo, H. Löwen, C. Reichhardt, G. Volpe, G. Volpe, *Rev. Mod. Phys.* **2016**, *88*, 045006.
- [2] J. Li, I. Rozen, J. Wang, *ACS Nano* **2016**, *10*, 5619.
- [3] J. Li, B. E.-F. de Ávila, W. Gao, L. Zhang, J. Wang, *Sci. Robot.* **2017**, *2*, eaam6431.
- [4] T. E. Mallouk, A. Sen, *Sci. Am.* **2009**, *300*, 72.
- [5] L. Soler, V. Magdanz, V. M. Fomin, S. Sanchez, O. G. Schmidt, *ACS Nano* **2013**, *7*, 9611.
- [6] C. P. Goodrich, M. P. Brenner, *Proc. Natl. Acad. Sci.* **2017**, *114*, 257.
- [7] M. Xiao, X. Guo, M. Cheng, G. Ju, Y. Zhang, F. Shi, *Small* **2014**, *10*, 859.
- [8] P. L. Venugopalan, R. Sai, Y. Chandorkar, B. Basu, S. Shivashankar, A. Ghosh, *Nano Lett.* **2014**, *14*, 1968.
- [9] S. Li, Q. Jiang, S. Liu, Y. Zhang, Y. Tian, C. Song, J. Wang, Y. Zou, G. J. Anderson, J.-Y. Han, Y. Chang, Y. Liu, C. Zhang, L. Chen, G. Zhou, G. Nie, H. Yan, B. Ding, Y. Zhao, *Nat. Biotechnol.* **2018**, *36*, 258.
- [10] M. Yu, L. Xu, F. Tian, Q. Su, N. Zheng, Y. Yang, J. Wang, A. Wang, C. Zhu, S. Guo, X. Zhang, Y. Gan, X. Shi, H. Gao, *Nat. Commun.* **2018**, *9*, 2607.
- [11] B. Jurado-Sánchez, S. Sattayasamitsathit, W. Gao, L. Santos, Y. Fedorak, V. V. Singh, J. Orozco, M. Galarnyk, J. Wang, *Small* **2015**, *11*, 499.

- [12] M. Zarei, M. Zarei, *Small* **2018**, 14, 1800912.
- [13] K. K. Dey, X. Zhao, B. M. Tansi, W. J. Méndez-Ortiz, U. M. Córdova-Figueroa, R. Golestanian, A. Sen, *Nano Lett.* **2015**, 15, 8311.
- [14] M. Medina-Sánchez, L. Schwarz, A. K. Meyer, F. Hebenstreit, O. G. Schmidt, *Nano Lett.* **2015**, 16, 555.
- [15] S. Sánchez, L. Soler, J. Katuri, *Angew. Chem., Int. Ed.* **2015**, 54, 1414.
- [16] V. B. Koman, P. Liu, D. Kozawa, A. T. Liu, A. L. Cottrill, Y. Son, J. A. Lebron, M. S. Strano, *Nat. Nanotechnol.* **2018**, 13, 819.
- [17] Z. Wu, L. Li, Y. Yang, P. Hu, Y. Li, S.-Y. Yang, L. V. Wang, W. Gao, *Sci. Robot.* **2019**, 4, eaax0613.
- [18] D. F. B. Haeufle, T. Bäuerle, J. Steiner, L. Bremicker, S. Schmitt, C. Bechinger, *Phys. Rev. E* **2016**, 94, 012617.
- [19] B. Liebchen, H. Löwen, *Europhys. Lett.* **2019**, 127, 34003.
- [20] B. Qian, D. Montiel, A. Bregulla, F. Cichos, H. Yang, *Chem. Sci.* **2013**, 4, 1420.
- [21] Y. Yang, M. A. Bevan, *ACS Nano* **2018**, 12, 10712.
- [22] D. J. Aidley, *Animal Migration*, Cambridge University Press, Cambridge, UK **1981**.
- [23] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *Nature* **2016**, 529, 484.
- [24] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *Nature* **2017**, 550, 354.
- [25] S. Levine, C. Finn, T. Darrell, P. Abbeel, *J. Mach. Learn. Res.* **2016**, 17, 1334.
- [26] M. Popova, O. Isayev, A. Tropsha, *Sci. Adv.* **2018**, 4, eaap7885.
- [27] Z. Zhou, X. Li, R. N. Zare, *ACS Cent. Sci.* **2017**, 3, 1337.
- [28] M. Komorowski, L. A. Celi, O. Badawi, A. C. Gordon, A. A. Faisal, *Nat. Med.* **2018**, 24, 1716.
- [29] Y. LeCun, Y. Bengio, G. Hinton, *Nature* **2015**, 521, 436.
- [30] G. Kahn, A. Villafior, B. Ding, P. Abbeel, S. Levine, in *2018 IEEE Int. Conf. on Robotics and Automation (ICRA)*, IEEE, Brisbane, QLD **2018**, pp. 5129–5136.
- [31] A. E. Sallab, M. Abdou, E. Perot, S. Yogamani, *Electron. Imaging* **2017**, 2017, 70.
- [32] A. Krizhevsky, I. Sutskever, G. E. Hinton, in *Advances in Neural Information Processing Systems*, **2012**, pp. 1097–1105.
- [33] T. Serre, L. Wolf, T. Poggio, in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, IEEE, San Diego, CA **2005**, pp. 994–1000.
- [34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *Nature* **2015**, 518, 529.
- [35] D. M. McComb, S. M. Kajiura, *J. Exp. Biol.* **2008**, 211, 482.
- [36] L. Barberis, F. Peruani, *Phys. Rev. Lett.* **2016**, 117, 248001.
- [37] T. R. Kline, W. F. Paxton, T. E. Mallouk, A. Sen, *Angew. Chem. Int. Ed.* **2005**, 44, 744.
- [38] M. C. Marchetti, J.-F. Joanny, S. Ramaswamy, T. B. Liverpool, J. Prost, M. Rao, R. A. Simha, *Rev. Mod. Phys.* **2013**, 85, 1143.
- [39] J. M. Yeomans, D. O. Pushkin, H. Shum, *Eur. Phys. J. Spec. Top.* **2014**, 223, 1771.
- [40] J. L. Bitter, Y. Yang, G. Duncan, H. Fairbrother, M. A. Bevan, *Langmuir* **2017**, 33, 9034.
- [41] A. M. Brooks, S. Sabrina, K. J. Bishop, *Proc. Natl. Acad. Sci.* **2018**, 115, E1090.
- [42] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, Vol. 1, MIT Press, Cambridge, USA **1998**.
- [43] H. Van Hasselt, A. Guez, D. Silver, in *Thirtieth AAAI Conf. on Artificial Intelligence*, Phoenix, Arizona **2016**.
- [44] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, W. Zaremba, *Adv. Neural Inform. Process. Syst.* **2017**, 31, 5048.
- [45] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley, Hoboken, NJ **2014**.
- [46] J. R. Howse, R. A. Jones, A. J. Ryan, T. Gough, R. Vafabakhsh, R. Golestanian, *Phys. Rev. Lett.* **2007**, 99, 048102.
- [47] L. van der Maaten, G. Hinton, *J. Mach. Learn. Res.* **2008**, 9, 2579.
- [48] Y. Yang, M. A. Bevan, *J. Chem. Phys.* **2017**, 147, 054902.
- [49] M. Hausknecht, P. Stone, in *2015 AAAI Fall Symp. Series*, Arlington, VA **2015**.
- [50] D. Maturana, S. Scherer, in *2015 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, IEEE, Hamburg, Germany **2015**, pp. 922–928.
- [51] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, *US 15/217,758*, **2015**.
- [52] T. D. Kulkarni, K. Narasimhan, A. Saedi, J. Tenenbaum, *Adv. Neural Inform. Process. Syst.* **2016**, 30, 3675.
- [53] S. Colabrese, K. Gustavsson, A. Celani, L. Biferale, *Phys. Rev. Lett.* **2017**, 118, 158004.
- [54] H. Xie, M. Sun, X. Fan, Z. Lin, W. Chen, L. Wang, L. Dong, Q. He, *Sci. Robot.* **2019**, 4, eaav8006.
- [55] H. J. Charlesworth, M. S. Turner, *Proc. Natl. Acad. Sci.* **2019**, 116, 15362.
- [56] X. Tang, B. Rupp, Y. Yang, T. D. Edwards, M. A. Grover, M. A. Bevan, *ACS Nano* **2016**, 10, 6791.
- [57] Y. Yang, *Doctoral Dissertation, Johns Hopkins University* **2017**.
- [58] M. A. Bevan, D. M. Ford, M. A. Grover, B. Shapiro, D. Maroudas, Y. Yang, R. Thyagarajan, X. Tang, R. M. Sehgal, *J. Process Control* **2015**, 27, 64.
- [59] D. A. Pomerleau, *Neural Comput.* **1991**, 3, 88.
- [60] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, L. D. Jackel, *Adv. Neural Inform. Process. Syst.* **1989**, 2, 396.
- [61] S. Ioffe, C. Szegedy, in *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, International Conference on Machine Learning, Lille, France **2015**, pp. 448–456.
- [62] V. Nair, G. E. Hinton, in *Proc. of the 27th Int. Conf. on Machine Learning (ICML-10)*, International Conference on Machine Learning, Madison, WI **2010**, pp. 807–814.
- [63] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *Nature* **2015**, 518, 529.
- [64] Y. Bengio, J. Louradour, R. Collobert, J. Weston, in *Proc. of the 26th Annual Int. Conf. on Machine Learning*, ACM, Montreal, Quebec, Canada **2009**, pp. 41–48.