An online hyper-volume action bounding approach for accelerating the process of deep reinforcement learning from multiple controllers

Aflakian, Ali; Rastegarpanah, Alireza; Hathaway, Jamie; Stolkin, Rustam

DOI:
10.1002/rob.22355

Received: 2 May 2023 | Revised: 22 January 2024 | Accepted: 15 April 2024

DOI: 10.1002/rob.22355

RESEARCH ARTICLE

WILEY

An online hyper-volume action bounding approach for accelerating the process of deep reinforcement learning from multiple controllers

Ali Aflakian[1,2] | Alireza Rastegarpanah[1,2] | Jamie Hathaway[1,2] | Rustam Stolkin[1,2]

[1]Department of Metallurgy & Materials Science, University of Birmingham, Birmingham, UK

[2]The Faraday Institution, Quad One, Harwell Science and Innovation Campus, Didcot, UK

Correspondence
Alireza Rastegarpanah, Department of Metallurgy & Materials Science, University of Birmingham, Birmingham B15 2TT, UK.
Email: a.rastegarpanah@bham.ac.uk

Funding information
Faraday Institution; UKRI Horizon Europe Underwriting – Innovate UK

Abstract

This paper fuses ideas from reinforcement learning (RL), Learning from Demonstration (LfD), and Ensemble Learning into a single paradigm. Knowledge from a mixture of control algorithms (experts) are used to constrain the action space of the agent, enabling faster RL refining of a control policy, by avoiding unnecessary explorative actions. Domain-specific knowledge of each expert is exploited. However, the resulting policy is robust against errors of individual experts, since it is refined by a RL reward function without copying any particular demonstration. Our method has the potential to supplement existing RLfD methods when multiple algorithmic approaches are available to function as experts, specifically in tasks involving continuous action spaces. We illustrate our method in the context of a visual servoing (VS) task, in which a 7-DoF robot arm is controlled to maintain a desired pose relative to a target object. We explore four methods for bounding the actions of the RL agent during training. These methods include using a hypercube and convex hull with modified loss functions, ignoring actions outside the convex hull, and projecting actions onto the convex hull. We compare the training progress of each method using expert demonstrators, employing one expert demonstrator with the DAgger algorithm, and without using any demonstrators. Our experiments show that using the convex hull with a modified loss function not only accelerates learning but also provides the most optimal solution compared with other approaches. Furthermore, we demonstrate faster VS error convergence while maintaining higher manipulability of the arm, compared with classical image-based VS, position-based VS, and hybrid-decoupled VS.

KEYWORDS
imitation learning, multi-expert demonstrations, online learning, optimization technique, reinforcement learning

Ali Aflakian and Alireza Rastegarpanah should be considered joint first author.

1 | INTRODUCTION

Recent advances in deep learning and reinforcement learning (RL) research have enabled robots to handle increasingly challenging tasks (Hua et al., 2021). In a RL paradigm, an agent attempts to maximize expected reward by interacting with its environment. However, RL methods learn via considerable trial and error, making RL difficult to implement on a real robot (Vecerik et al., 2017). Additional challenges arise from the agent lacking a priori data or knowledge about its environment. To alleviate these problems, behavioral knowledge can be derived from expert demonstrations. A demonstrated action provides a starting point, which is further refined by RL. The demonstrated action essentially reduces the search space that must be explored by the RL agent, during optimization, to find an optimal policy (Zhu & Hu, 2018). This process is known as Reinforcement Learning from Demonstrations (RLfD) (Jing et al., 2020).

As an example application of RLfD, we consider a visual servoing (VS) task in which a robot arm must maintain its end-effector (EE) at a desired pose relative to a moving object observed by a wrist-mounted camera (eye-in-hand). The robot must learn a control policy that generalizes to handle arbitrary object motions, while exploiting feedback data comprising robot states and camera images. RL provides a theoretical way for learning such policies through exploration of the action space. However, the amount of exploration required has limited its implementation in real world applications. In this paper, we address this challenge by merging the demonstrations from multiple controllers and RL concepts into a single framework, that guides RL using demonstrations and feedback from several "expert" controllers.

There are two potential approaches for incorporating knowledge from expert demonstrations in RL: prior knowledge, comprising demonstrations before RL refinement; and online knowledge, in which case demonstrations are occasionally presented while the RL iterations are in progress (Ramírez et al., 2022). The online method can significantly enhance the learned policy's convergence toward an expected performance level while lowering the likelihood of distributional mismatch compared with the prior knowledge approach (Ross et al., 2011a). However, several challenges remain with the online use of demonstrations while the agent is learning. These include: the high cost of data collection; the inability to generalize to different scenarios; and the limitation of the agent's exploration to blindly following the demonstrator (Ho & Ermon, 2016; Krishnan et al., 2019; Takeda et al., 2007). To avoid unnecessary exploration while also overcoming the problem of the agent overly following the expert behavior, we present an online Action Optimizer for improving Reinforcement Learning from multi-Demonstrations (AORLD) (detailed in Section 2.2). Our approach is "online" in the sense that the agent action space would be modified in real-time during the training process (while the RL iterations are in progress). The proposed AORLD approach is generic, in that it can be applied to a wide variety of RL scenarios. However, to demonstrate and validate the method, we implement AORLD in the context of a VS task, detailed in Section 2.2. Based on the knowledge of several controllers, we explore four methods for bounding the actions of the RL agent during training.

The first method called AORLD-HL, involves constraining the agent's action space to an online convex hull generated from the collective knowledge of controllers. We achieve this by defining a convex hull that encapsulates the feasible action space based on the expertise of the controllers. Subsequently, we modify the loss function to penalize the agent for deviating from actions within this generated hypervolume. On the other hand, the second method, AORLD-CL, adopts a different approach by generating an online hypercube from the knowledge of experts to restrict the agent's action space. Similar to AORLD-HL, we adjust the loss function to discourage actions outside the boundaries of the hypercube. While these two methods share the common goal of limiting the agent's action space to improve training efficiency, they differ in how they define and enforce these limitations (convex hull and hypercube).

The third method involves filtering the actions suggested by the RL policy that lie outside the generated convex hull (AORLD-HF). We use the standard loss function for the RL agent. This method does not modify actions directly but instead filters out undesirable actions.

The fourth method involves projecting the actions outside the convex hull onto the convex hull (AORLD-HP). This method modifies the action space and ensures that the agent always takes actions within the convex hull. In Section 2, we provide detailed insights into the structures and algorithms of these four methods to explain the intricacies of each approach. Thereafter, we compare the performance of these methods to that of a standard RL algorithm without any bounding method.

The highlights of this paper are summarized as follows:

1. AORLD adaptively constrains the agent's action spaces by exploiting demonstrations from different "expert" controllers, improving training efficiency in terms of reducing the training time and improving the performance of the trained policy.
2. The AORLD approach is generic and can be incorporated into a wide variety of multiagent and other RLfD algorithms across various applications. Moreover, since AORLD does not directly conform to any single demonstration, it is robust against errors or imperfections in any individual demonstration.
3. AORLD mitigates the risk of the agent converging prematurely to suboptimal solutions, since, limiting the action space to a region where the likelihood of success is higher plays a crucial role in preventing RL agents from getting stuck in local optima.

The remainder of this study is structured as follows: Section 2 provides a discussion of related literature in RL and VS, followed by a detailed explanation of the proposed AORLD method. The process of training AORLD in a simulated environment is subsequently detailed. Section 3 introduces the simulation environment for a VS application. Section 4 presents the results of experiments, showing how the trained policy improves VS task performance. Section 5 provides concluding remarks, and also offers suggestions for extending AORLD to other applications, and combining it with other RLfD methods.

1.1 | Related work

Early RLfD algorithms are classified into three main groups: Behavior Cloning (BC), Generative Adversarial Imitation Learning (GAIL), and Inverse Reinforcement Learning (IRL) (Kumar et al., 2016). BC was developed based on direct policy learning, which enables the distribution of the state/action trajectory to match the demonstration given by a supervisor. The agent has no capability to respond to environmental changes (Takeda et al., 2007). Therefore, in the case of using a small number of samples, the trained BC policy has little capability to generalize to different scenarios. IRL was developed to tackle the problem of reward function design and is more adaptable to new situations (Krishnan et al., 2019). While BC and IRL methods gain experience from demonstrations, they have no capability to interact with experts during training to make the trained policy more optimized and robust. To enable the agent to better exploit the expert when optimizing a policy, the GAIL approaches were developed based on generative adversarial networks (Ho & Ermon, 2016). The GAIL approach is applied by making a comparison between generated and expert strategies, and converging them as closely as possible. However, the GAIL method is susceptible to convergence on local optima (Ho & Ermon, 2016). Also, BC methods suffer from data mismatch and compounding error issues. Consequently, the DAgger algorithm was developed to tackle this problem (Ross et al., 2011b). DAgger is an iterative policy learning method that employs online learning as a reduction in which the main classifier will be retrained on all states encountered by the learner at each iteration. In spite of this, the policy trained with DAgger will not be generalized to different scenarios, and the approach is limited to learning from the expert and cannot surpass its performance (Hester et al., 2018). Interactive imitation learning methods (e.g., HG-DAgger and ThriftyDAgger) are variants of DAgger, and they are also introduced to address some robustness issues of DAgger (Hoque et al., 2021; Kelly et al., 2019).

Typically, human demonstrations were employed for such interactive imitation techniques. Algorithms that can use other controllers as experts have been less well studied. Modern RLfD techniques incorporate aspects from imitation learning, and push the agent to replicate the demonstrated behaviors when feedback from the environment is scarce or even missing (Kang et al., 2018; Sun et al., 2018). They specifically reshape the reward function in RL by adding another term to encourage expert exploration. While rewarding expert-like activities might assist in minimizing unnecessary exploration, applying such rewards throughout the learning phase can be troublesome with imperfect demonstrations. There is no guarantee that limiting divergence from expert behavior will result in an improved agent policy (Yang et al., 2019). Unlike the aforementioned methods that aimed to replace existing learning from demonstration strategies, we propose a method that is complementary to established RLfD methods. We combined aspects of both RL and imitation learning in a novel online manner. In the selection of expert controllers, we strategically chose image-based visual servoing (IBVS) for its proficiency in image space (2D), position-based visual servoing (PBVS) for its excellence in workspace (3D), and hybrid decoupled visual servoing (HDVS) as a hybrid approach combining both 2D and 3D VS (Rastegarpanah et al., 2021a). This strategic selection ensures that the expert controllers demonstrate efficacy in different domains of VS. The practice of limiting the action space to a region where the likelihood of success is higher has been shown to play a pivotal role in preventing RL agents from becoming trapped in local optima (Yuan et al., 2023). Domain Randomization (DR) is utilized during the process of learning to adapt the trained policy to real world environments, and to make the policy robust in terms of noise, lighting variations, and also in the presence of random objects in the camera scene. We demonstrate a learning process that is not only greatly accelerated (compared with when there is no demonstrator), but the policy will also inherit the high performance of each expert technique and improve its behavior. Furthermore, using DR

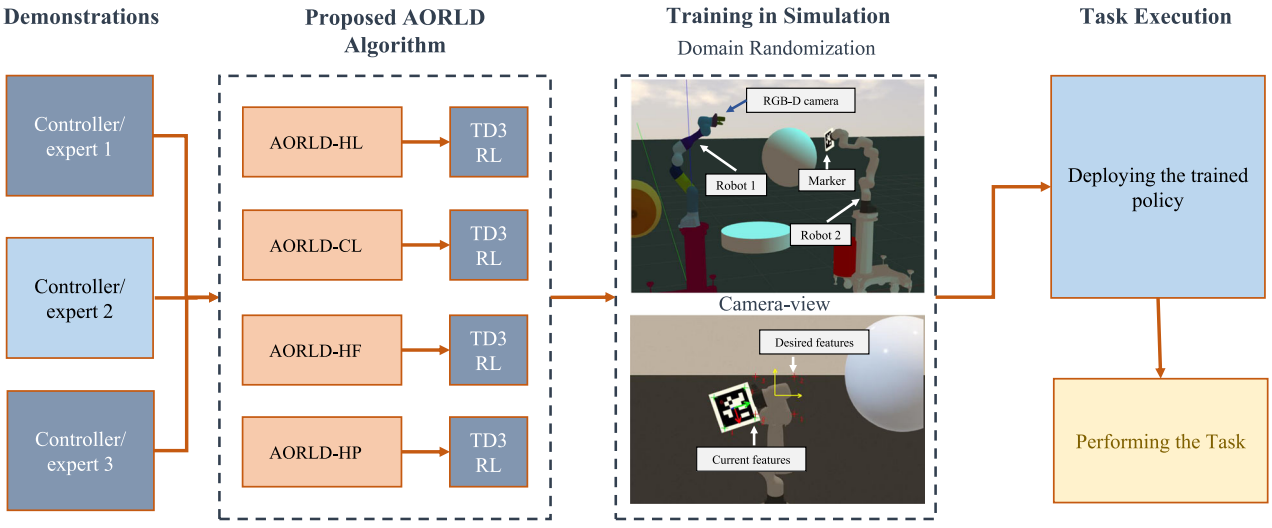

FIGURE 1 The outline of the proposed online action-optimizer combined with the reinforcement learning (RL) method for a visual servoing (VS) application. The Twin Delayed Deep Deterministic Policy Gradient (TD3) agent was trained by using the Domain Randomization (DR) method. The Action Optimizer for improving Reinforcement Learning from multi-Demonstrations (AORLD) method employs a combination of three different methods (position-based visual servoing, image-based visual servoing, and hybrid decoupled visual servoing) as demonstrators to accelerate training and enhance VS performance. Four different approaches have been used to constrain the action space of the agent and their results are compared together. [Color figure can be viewed at wileyonlinelibrary.com]

significantly enhances the generalizability of the policy to new environments, unlike other methods, such as DAgger and BC, which suffer in this area. Figure 1 outlines the AORLD method for a VS application.

2 | METHODOLOGY

2.1 | Proposed AORLD in RL

While a RL agent interacts with its environment, it receives a state s from a state space S and chooses an action a from an action space A, based on a policy $\pi(a|s)$. Mapping from state s to actions a ends up with a scalar reward r and next state s'. This mapping from state to actions is based on the environment model, rewards function $R(s, a)$, and state transition probability $T(s, a, s') = P(s'|s, a)$. In an episodic problem, this procedure is repeated until the agent achieves a terminal state and the agent aims to maximize cumulative rewards by optimizing its policy.

The RL algorithm used in this approach is TD3 (Twin Delayed Deep Deterministic Policy Gradient). TD3 is an effective off-policy actor-critic algorithm that uses delayed policy updates and target policy smoothing to improve stability and performance. The algorithm involves the use of two Q-functions, Q_{ϕ_1} and Q_{ϕ_2}, which are learned simultaneously by minimizing the mean square Bellman error (Bellman, 1957).

To form the Q-learning target in the TD3 RL algorithm, actions are generated based on the target policy, denoted as $\mu_{\theta_{\text{targ}}}$. However, clipped noise is added to each dimension of the action to enhance exploration. This means that the target action is obtained by adding clipped noise to the output of the target policy, and then clipping the result to ensure that it lies within the valid action range ($a_{\min} \le a \le a_{\max}$). Mathematically, the target actions can be expressed as (Fujimoto et al., 2018):

$$a'(s') = \text{clip}(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{\min}, a_{\max}) \quad (1)$$

where ϵ is a noise term drawn from a normal distribution with zero mean and standard deviation σ, and c is a constant that determines the amount of noise added to the action. The approach trains two Q-functions, labeled as Q_{ϕ_1} and Q_{ϕ_2}, simultaneously by regressing towards a single target value. The target value is computed by choosing the Q-function that gives the lower target value, which is expressed mathematically as:

$$y(r, s', d) = r + \gamma(1 - d)\min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')). \quad (2)$$

Here, r is the reward, s' is the next state, d is a binary indicator of whether the episode has ended, γ is the discount factor, and $a'(s')$ is the target action with clipped noise, as described earlier in (1).

Both Q-functions, Q_{ϕ_1} and Q_{ϕ_2}, are then trained by minimizing the squared difference between the predicted Q-value and the target value (Lillicrap et al., 2015).

Finally, the TD3 policy with the modified loss function is learned by maximizing Q_{ϕ_1}. The actor network, denoted as μ_θ, selects actions that maximize the Q-value estimated by Q_{ϕ_1}.

While we used TD3 in this case, the proposed method is readily applicable to different RL algorithms by modifying their target action limits. The overall procedure of the proposed optimization learning method is detailed in Algorithm 1. The algorithm takes candidate actions from multiple expert controllers as input and optimized actions for a given task, with respect to the observations, are the outputs of the trained policy. The AORLD approach has four different methods for constraining the actions generated by the RL algorithm: convex hull with modified loss function (AORLD-HL), hypercube with modified loss function (AORLD-CL), convex hull with filtering actions outside the hull (AORLD-HF), and convex hull with projecting actions outside the hull onto the convex hull (AORLD-HP). Each method modifies the TD3 algorithm in a specific way to constrain the generated actions. Algorithm 1 allows the user to choose one of the four methods (AORLD-HL, AORLD-CL, AORLD-HF, or AORLD-HP) for the entire training process. The step-by-step details for each action space limitation method are explicitly provided in dedicated algorithms (Algorithms 2, 3, 4, and 5, respectively).

Algorithm 1: AORLD approach

Inputs:
- Candidate actions from expert 1 (a_{ex1}), expert 2 (a_{ex2}), ..., expert n (a_{exn})

Outputs:
- Optimized actions a_t

Given:
- RL algorithm TD3
- The strategy for sampling goals from replay
- The reward function

```
Initialize actor and critic weights randomly;
Initialize a set of actions randomly;
Initialize replay buffer R;
while Controller stop criteria not achieved do
    for episode i = 1 to M do
        Sample g (goal) and initial s_0 (state);
        for t = 0 to T − 1 do
            if AORLD-HL then
                Use Algorithm 2 to calculate a_t;
            else if AORLD-CL then
                Use Algorithm 3 to calculate a_t;
            else if AORLD-HF then
                Use Algorithm 4 to calculate a_t;
            else if AORLD-HP then
                Use Algorithm 5 to calculate a_t;
            end
            Execute a_t and observe s_{t+1} (new state);
            r_t := r(s_t, a_t, g);
            Store (s_t|g, a_t, r_t, s_{t+1}|g) (transition) in R;
            Sample g' (additional goal) for replay
             G : S(currentepisode) ;
            for g' ∈ G do
                r' := r(s_t, a_t, g');
                Store (s_t|g', a_t, r', s_{t+1}|g') in R;
            end
            for t = 1 to N do
                Sample B (mini-batch) from the R
                 (replay buffer);
                Execute one step of optimization using
                 TD3 and B;
            end
        end
    end
end
```

Algorithm 2: AORLD-HL

Inputs:
- Candidate actions from expert 1 (a_{ex1}), expert 2 (a_{ex2}), ..., expert n (a_{exn})

Outputs:
- Candidate actions (a_t) inside the generated convex hull

for $j = 1$ *to* n **do**
  Generating convex hull around actions vectors A:
  $k = convhulln(A)$;
  Find the minimum and maximum vectors for each dimension:
  $a_{min} = \min(a(k(:),:),[],1)$;
  $a_{max} = \max(a(k(:),:),[],1)$;
  Modify the TD3 algorithm to use the new bounded action space:
  $a'(s') =$
  $\text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{min}, a_{max}\right)$
  Sample actions (a_t) inside the convex hull using TD3 policy:
  $\pi(s_t|g) \rightarrow a_t$;
end

Algorithm 3: AORLD-CL

Inputs:
- Candidate actions from expert 1 (a_{ex1}), expert 2 (a_{ex2}), ..., expert n (a_{exn})

Outputs:
- Candidate actions (a_t) inside the generated hyper cube

for $j = 1$ *to* n **do**
  Make action bounds:
  $a_{bound} = [\min(a_{ex1}[i], a_{ex2}[i], ..., a_{exn}[i]),$
  $\max(a_{ex1}[i], a_{ex2}[i], ..., a_{exn}[i])]$;
  Modify the TD3 algorithm to use the new bounded action space:
  $a'(s') =$
  $\text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{min}, a_{max}\right)$
  Sample actions (a_t) inside the convex hull using TD3 policy:
  $\pi(s_t|g) \rightarrow a_t$;
end

Algorithm 4: AORLD-HF

Inputs:
- Candidate actions from expert 1 (a_{ex1}), expert 2 (a_{ex2}), ..., expert n (a_{exn})

Outputs:
- Candidate actions (a_t) inside the generated convex hull

for $j = 1$ *to* n **do**
  Generating convex hull around actions vectors A:
  $k = convhulln(A)$;
  Filter actions outside the created bound:
  if *Inhull(k,a$_t$)==1* **then**
    Sample actions (a_t) inside the convex hull using TD3 policy:
    $\pi(s_t|g) \rightarrow a_t$;
  else
    Repeat the action generation
  end
end

Algorithm 5: AORLD-HP

Inputs:
- Candidate actions from expert 1 (a_{ex1}), expert 2 (a_{ex2}), ..., expert n (a_{exn})

Outputs:
- Candidate actions (a_t) inside the generated convex hull

for $j = 1$ *to* n **do**
  Generating convex hull around actions vectors A:
  $k = convhulln(A)$;
  if *Inhull(k,a)==1* **then**
    Sample action (a_t) inside the convex hull using TD3 policy:
    $\pi(s_t|g) \rightarrow a_t$;
  else
    Project the action vector onto the convex hull:
    for $l = 1 : size(k, 1)$ **do**
      $a_t = zeros(n, 1)$;
      $x = points(k(i,:),:)$;
      $u = x(1,:)'$;
      $v = x(2,:)' - u$;
      $proj = u + v * ((a - u)' * v)/norm(v)^2$;
      $a_t = a_t + proj$;
      $\pi(s_t|g) \rightarrow a_t$;
    end
  end
end

As illustrated in Figure 2, in each iteration of the training process, the AORLD approach is used to modify the target actions by one of the four abovementioned methods. The resulting action is then executed in the environment, and the observed states, taken actions, resulting rewards, and potential next states are stored in the replay buffer. Subsequently, these experiences stored in the replay buffer are used to augment the training data for the ongoing episode. The actions corresponding to each algorithm (2–5) represent the outputs of individual expert controllers. These outputs contribute to the fine-tuning of the agent's action space. The agent, in turn, employs each AORLD method to choose a singular candidate action from within the action space, which ultimately serves as the output action.

In the following, we will describe those four methods to constrain actions in detail and explain their respective pros and cons.

The AORLD-HL algorithm is introduced in Algorithm 2. To compute the $a_{\min}$ and $a_{\max}$ values in (1), the following procedure was adopted: data are collected from multiple controllers that generate candidate action vectors for each observation of the environment. Thereafter, the convex hull of action vectors is computed and the resulting convex hull used to define the feasible action space for the RL agent. Let k be the set of indices that define the convex hull of A, that is, k = convhulln(A). For each dimension i of A, minimum ($a_{\min}$) and maximum ($a_{\max}$) values over the vertices of the convex hull are computed as follows:

$$a_{\min,i} = \min_{j \in k} a_{j,i} \quad \text{and} \quad a_{\max,i} = \max_{j \in k} a_{j,i}, \tag{3}$$

where $a_{j,i}$ is the ith component of the jth vertex of the convex hull. It should be noted that $\pi(s_t|g)$ in Algorithm 2 is the policy that maps the

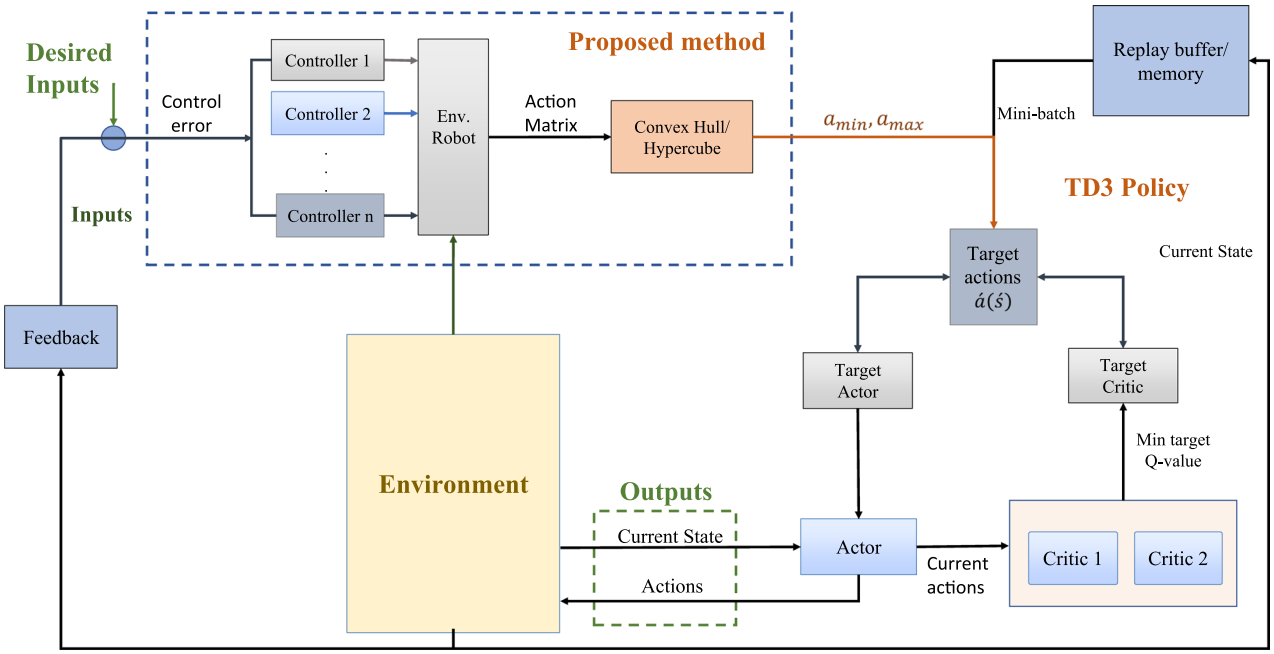

FIGURE 2 Structure of Action Optimizer for improving Reinforcement Learning from multi-Demonstrations (AORLD) integrated with TD3 RL. The proposed block diagram in this study takes in current and desired features extracted from the vision sensor as inputs. Then, during each episode, the knowledge from hybrid decoupled visual servoing, position-based visual servoing, and image-based visual servoing approaches is utilized to restrict the action space. The joint velocity actions are then applied to the training environment, and the average rewards are computed accordingly. [Color figure can be viewed at wileyonlinelibrary.com]

state s_t to an action a_t given the goal g, and $\rightarrow$ denotes the assignment of a_t to the output of the policy. The convex hull is a mathematical concept that defines the smallest convex set that contains all the given points in a higher dimensional space. In our case, the convex hull is a boundary that encloses most of the potentially desired action vectors. It should be mentioned that it requires at least $n + 1$ unique points in n-dimensional space to create a n-dimensional convex hull. The algorithm uses each controller prediction to have at least $n + 1$ data if there is not enough data to build the n-dimensional convex hull. Given the bounding convex hull, we can find the minimum and maximum values for each dimension by computing the minimum ($a_{\min}$) and maximum ($a_{\max}$) values of each coordinate of the vertices of the convex hull. During training, the convex hull would be periodically updated using a new set of action vectors. As explained before, this approach aims to adaptively constrain the RL agent to choose actions within the feasible action space defined by the convex hull. Limiting the actions of the RL agent to lie within the convex hull can potentially simplify the learning problem and make it easier for the agent to converge to a good policy. Moreover, by limiting the actions to a smaller region of the action space, the agent has fewer options to choose from and can more quickly learn which actions are likely to lead to good outcomes.

The AORLD-CL algorithm is established in Algorithm 3. In AORLD-CL, instead of generating a convex hull around the experts' outputs, a set of bounds for the action is defined based on the combination of demonstrators' data. These bounds represent the minimum and maximum values that each action can take. To create the lower limit of the i_{th} action, the minimum value of that action from all the demonstrators' data is used:

$$a_{\min,i} = \min(a_{ex1}[i], a_{ex2}[i], \ldots, a_{exn}[i]). \tag{4}$$

Similarly, the upper limit of the i_{th} action is created using the maximum value of that action from all the demonstrators' data:

$$a_{\max,i} = \max(a_{ex1}[i], a_{ex2}[i], \ldots, a_{exn}[i]). \tag{5}$$

This creates a hypercube in the action space that represents the feasible action space for the agent. Not to mention that the convex hull is the minimum bounding convex hypervolume that includes the actions from the controllers, which reduces the action and search space of the agent more than the hypercube. Convex hull also accounts for correlations between different actions, whereas the hypercube approach in AORLD-CL assumes each action dimension is independent. During training, the hypercube is updated periodically with the demonstrators' new set of action vectors. This ensures the feasible action space is updated as the agent learns from the demonstrators' data. The modified TD3 loss function in AORLD-CL enforces that the agent's predicted actions stay within these bounds. This simple approach to limiting the action space can be computationally less expensive than generating a convex hull and still helps to constrain the agent's actions to the feasible action space. Nevertheless, the agent's exploration space is not optimally condensed and is larger than when employing a convex hull.

The AORLD-HF algorithm (Algorithm 4) is a variant of the AORLD-HL algorithm that also generates a convex hull around expert actions but differs in how it filters actions. The algorithm takes candidate actions from multiple expert controllers as input and outputs candidate actions inside the generated convex hull. The algorithm 4 follows the steps below:

(I) Generate a convex hull around action vectors A using the *convhulln* function, (II) Find the minimum and maximum vectors for each dimension using the minimum and maximum functions, (III) For each action sample a_t, check if it is inside the generated convex hull using the Inhull function, (IV) If a_t is inside the hull, sample an action inside the convex hull using the TD3 policy, (V) If a_t is outside the hull, repeat the action generation.

Finally, AORLD-HP, introduced in Algorithm 5, differs from AORLD-HF in that AORLD-HF filters actions outside the generated bound; however, AORLD-HP uses a projection method explained in Algorithm 5. The projection method projects the candidate action onto the closest point on the boundary of the convex hull as follows: Let k be the set of indices of the convex hull points, *points* be the set of points on the convex hull, and a be the original action vector to be projected onto the convex hull. Let u be the starting point of a line segment on the convex hull, v be the direction of the line segment, and *proj* be the projection of a onto the line segment.

$$proj = u + v\frac{(a - u)^T \cdot v}{|v|^2}. \tag{6}$$

This approach ensures that the action is projected onto the nearest segment of the boundary, by computing the projections on all segments formed by the boundary and selecting the one with the lowest distance. Doing so allows the agent to stay within the desired action space. It should be mentioned that the hard constraint of modifying the loss function with new minimum and maximum actions in AORLD-HL method is more effective than filtering and projecting actions on the hull (AORLD-HF and AORLD-HP, respectively), as it enforces the action constraints strictly. However, the algorithm in AORLD-HL assumes that the candidate actions from the expert controllers are sufficient to define the action space, which may not always be the case in complex environments. We will discuss VS as an application to demonstrate our suggested approaches in the following subsection.

2.2 | VS

VS is a widely used technique in the field of robot vision that translates visual errors into actuator commands (Rastegarpanah et al., 2021b). However, conventional VS methods still face several limitations in terms of stability, convergence, and gain selection, as highlighted in (Sampedro et al., 2018). These issues are partly due to the challenges involved in calculating the image Jacobian (Interaction matrix), which can lead to singularities and local minima. Additionally, a lack of direct control over the robot joint velocities can also result in prominent issues, as the controller may not be aware of the limitations and performance of the robot (Chaumette, 1998).

The literature has discussed advanced techniques to circumvent the problems associated with conventional VS methods (Aflakian et al., 2023; Castelli et al., 2017; Chesi et al., 2004; Corke & Hutchinson, 2001; Gans & Hutchinson, 2007; Jin et al., 2021; massoud Farahmand et al., 2009; Rastegarpanah et al., 2021c). These techniques are aimed at developing more sophisticated and adaptive control strategies that can improve the stability, convergence, and gain selection of the system.

The proposed method in this paper involves combining the results of three different VS methods, namely IBVS, PBVS, and HDVS, to serve as a supervisor for the learner. The ultimate aim is to develop a policy that surpasses the performance of these supervisors. The paper provides an overview of the methodologies employed, described in detail in the following paragraphs. In IBVS, the features in the image space provide immediate feedback to the controller.

To link pixel velocity to camera velocity in IBVS, an image Jacobian matrix (L_i) is utilized (Hu et al., 2009). This matrix relates the camera velocity vector (v_{cam}) to the error in the image space (e_i) described in (Hu et al., 2009). The control law for IBVS is then calculated:

$$v_{\text{cam}} = -k_i L_i^+ e_i. \tag{7}$$

Here, k_i represents the controller gain, and L_i^+ denotes the pseudo-inverse representation of L_i. In PBVS, feedback is obtained from the reconstructed pose of the environment, which is computed using Euclidean algorithms and camera parameters (Malis et al., 1999). The control law for PBVS is given as follows:

$$v_{\text{cam}} = -k_p L_p^{-1} e_p. \tag{8}$$

The control gain is represented by k_p, and e_p refers to the 3D error of object position with respect to its desired position, measured in the task space. $L_p(t)$ is a 6×6 matrix, defined in (Hu et al., 2009).

The third approach utilized in our method is HDVS, which is explained in detail in Rastegarpanah et al. (2021a). HDVS utilizes both 2D information from image features and their estimated 3D poses. The control law of the HDVS method is obtained by simultaneously solving (9) and (10):

$$v_{xy} = L_{xy}^+\{-k_h e - L_r v_r\}, \tag{9}$$

$$v_r = L_{Pr}^+\{-k_h e_p - L_{Pxy} v_{xy}\}, \tag{10}$$

where $v_{xy} = (v_x, v_y)$, $v_r = (v_z, w_x, w_y, w_z)$, k_h is the controller gain, and L_{xy} and L_r are defined in (Rastegarpanah et al., 2021a). The data from (9) and (10) was then used to train a LoLiMOT neural network to produce a detailed prediction of the camera velocities from the feature errors. After obtaining the end-effector (EE) velocities $\left(v_{\text{cam}} = [v_{xy} : v_r]^T\right)$ in the HDVS method, the joint velocities ($\dot{q}$) can be computed using the following equation (Baerlocher & Boulic, 1998):

$$\dot{q} = J^{+\lambda} \xi_c^e v_{\text{cam}}^c, \tag{11}$$

where λ is damping factor (Baerlocher & Boulic, 1998). The AORLD approach utilizes expert demonstrations for learning decision-making policies and constraining the agent's action space, as described in Section 1.1, instead of allowing AI agents to learn solely through their own exploration. In the RL algorithm, the actions taken by the agent are defined as joint velocities, while the observations provided to the agent include image feature locations, camera pose, and robot Jacobian. Additionally, the reward earned by the agent is a composite of three distinct reward functions. The first reward function aims to drive the image feature errors to zero, which is expressed mathematically as r_1:

$$r_1 = -\sum_{i=1}^{4} \sqrt{(u_i - u_{id})^2 + (v_i - v_{id})^2}, \tag{12}$$

where (u, v) denotes the coordinates of a point in the camera view and (u_d, v_d) is the desired coordinates of that point. $i = 1$ to 4 is the number of features (in this case four features). To keep away from joint limits, the second reward function (r_2) is defined as follows (Franks et al., 2008):

$$r_2 = -\frac{1}{2n} \sum_{j=1}^{n} \left(\frac{q_j - \bar{q}_j}{q_{jM} - q_{jm}} \right)^2, \tag{13}$$

where $\bar{q}_j$ is center of j^{th} joint range. q_{jM} and q_{jm} are the maximum and minimum angles of the j^{th} joint respectively, and $n = 7$ is the number of joints. The last reward component is avoiding manipulator singularities and improving the robot manipulability (controllability), described in (Franks et al., 2008):

$$r_3 = \sqrt{\det(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}^T(\boldsymbol{q}))}, \tag{14}$$

where $\boldsymbol{J}$ is the robot Jacobian. The final reward function will be derived as:

$$r = -w_f r_1 - w_q r_2 - w_m r_3. \tag{15}$$

The terms w_f, w_q, and w_j are weighting factors that are manually adjusted. For each reward, values of $w_f = 10$, $w_q = 2$, and $w_j = 4$ were chosen to determine the weighting contribution. Therefore, in this case study, the controllers correspond to each visual servoing control law for IBVS, PBVS and HDVS, and the actions themselves correspond to robot's joint velocities. The output action of each algorithm (2–5) is hence also the joint velocities.

3 | EXPERIMENTAL SETUP

To train the policy, a simulation environment was developed using ROS/Gazebo. This approach is more efficient and cost-effective compared with real world experiments, especially for RL, because it involves extensive exploration of the environment. Additionally, using simulation helps prevent potential damage to the robot setup during training. The trained model with Domain Randomization (DR) can adapt to the real world environment because the real system is assumed to be one instance in a wide range of training variations. DR is a technique used for training models to work in various simulated settings with randomized properties (Tobin et al., 2017).

The TD3 algorithm was implemented as a ROS node, and Matlab Reinforcement Learning Toolbox (MATLAB, 2021) was used to train policies. The simulation environment included two Franka robot manipulators, one with an eye-in-hand configuration and the other with a tag marker attached to its end-effector to move the marker into various positions. An Intel RealSense depth camera D435i was utilized as a vision sensor. Parallel training was used to accelerate the learning process with the aid of Parallel Computing Matlab Toolbox, which involved deploying 12 workers to simulate the agent in the environment and transmit data back to the client. The simulation platform depicted in Figure 1 was used in the simulation environment in Gazebo and Figure 7 shows the real world environment used in this study.

The system used in this study had the following specifications: an NVIDIA GTX 1080Ti graphics card, an Intel(R) Core(TM) i7-10510U CPU with a base clock speed of 1.80 GHz and a turbo boost frequency of 4.9 GHz, and 16.0 GB of RAM.

4 | RESULTS AND DISCUSSION

4.1 | Simulation results

To evaluate the effectiveness of the proposed approach, six different agents were defined and trained, and their training progress was compared. The first method, called AORLD-HL, involved constraining the agent's action space to an online convex hull generated from controller knowledge and modifying the agent's loss function to penalize actions outside the generated hypervolume. The second method, AORLD-CL, involved generating an online hypercube to constrain the action space, and similarly modifying the loss function. The third method, AORLD-HF, involved filtering out actions suggested by the RL policy that lie outside the generated convex hull while using the standard loss function. The fourth method, AORLD-HP, involved projecting actions outside the convex hull onto the convex hull. The fifth approach involved implementing DAgger with HDVS as the demonstrator. Finally, the sixth policy was created using only RL without any demonstrator.

To make the policy robust to noise, calibration errors, and random objects in the scene, domain randomization was used. All six agents were trained for 25,000 episodes, with the initial position of the first robot randomized in each episode to generalize the trained policy. The agent restarted the episode if it met one of four criteria: (I) when the robot was close to joint limits, (II) when the features were very close to the image boundary, (III) when the robot's Jacobian manipulability was very small (less than 0.01), or (IV) when the

number of steps in each episode exceeded 400. The parameters for the RL algorithm in the training were specified in Table 1 for all the agents.

The agents in the experiment have learned to maximize the cumulative reward over time, as shown in Figure 3. Among the tested methods, the TD3 agent with the AORLD-HL algorithm achieved the highest average reward of approximately −220, followed by the agent with AORLD-CL with an average reward of around −250. These two methods are effective in ensuring that the agent's actions are valid, as they enforce hard constraints. However, they require modifications to the RL algorithm, including changes to the loss function and target action. Furthermore, it can be inferred from the data presented in Figure 3 that the AORLD-HL algorithm requires a smaller number of episodes to attain a satisfactory average reward compared with all other four methods. The less the agent must interact with the environment, the faster it will learn the task.

The AORLD-HF method, on the other hand, is simpler as it only allows the agent to choose valid actions without additional calculations. However, it resulted in a less effective agent, as it may limit the agent's ability to explore the state space and find optimal solutions. The average reward obtained by the agent in this method is approximately −300, as shown in Figure 3. Therefore, the first two methods are more effective as they allow the agent to explore the state space while staying within the feasible action space.

TABLE 1 Reinforcement learning (RL) and noise parameters employed in training.

RL parameters		Noise options	
Target smooth factor	1e−03	Mean	0
Learning rate	5e−04	Mean attraction constant	5
Sample time	2.5e−02	Variance decay rate	1e−05
Discount factor	0.95	Variance	0.5

The agent trained with AORLD-HP algorithm had an average reward of around −400, which is lower than the first two methods (Figure 3). This is because the projection method used in AORLD-HP may not always provide an accurate representation of the action space, especially in higher dimensions. Additionally, this method is computationally expensive since it requires projecting each action outside the hull onto the hull, and it may be less effective if the hull is irregularly shaped or difficult to calculate. The agent trained with DAgger demonstrates a noteworthy learning trajectory, reaching an average reward of approximately −400 within around 500 episodes. However, despite its initial rapid progress, it struggles to surpass an average reward of −350, hindering its ability to outperform the expert demonstrator. This emphasizes the need for a more flexible training approach, as provided by the proposed AORLD method. Moreover, as training progresses with DAgger, the agent begins to exhibit signs of overfitting behavior after approximately 8000 episodes. Finally, the agent without using any action constraints achieved an average reward of −600, indicating the effectiveness of the AORLD method.

In this study, we conducted a comparison between the performance of IBVS, PBVS, HDVS, and six trained policies using effective parameters. To obtain these parameters, we carried out 90 experiments with the robots starting from randomly selected initial positions, ensuring that all four features were visible in the image frame. The mean values of these parameters were then derived and reported in Figures 4–6.

Figure 4a compares the number of iterations taken to complete the VS task and the root mean square (RMSE) of 2D errors in the image space for different methods, including AORLD-HL, AORLD-CL, AORLD-HF, AORLD-HP, DAgger, IBVS, PBVS, HDVS, and the TD3. It

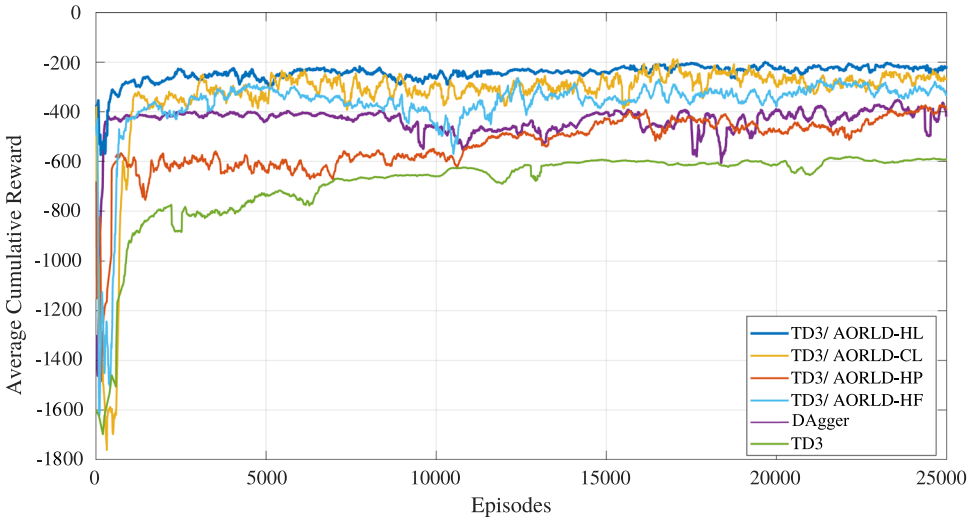

FIGURE 3 The graph illustrates the average reward per episode for each method during the training process. AORLD-HL and AORLD-CL exhibit faster learning and achieve higher average rewards. AORLD-HF, AORLD-HP, and the agent without action constraints show slower learning and lower average rewards. DAgger, demonstrates rapid learning but with a trade-off of lower average rewards compared with AORLD-HL and AORLD-CL. [Color figure can be viewed at wileyonlinelibrary.com]

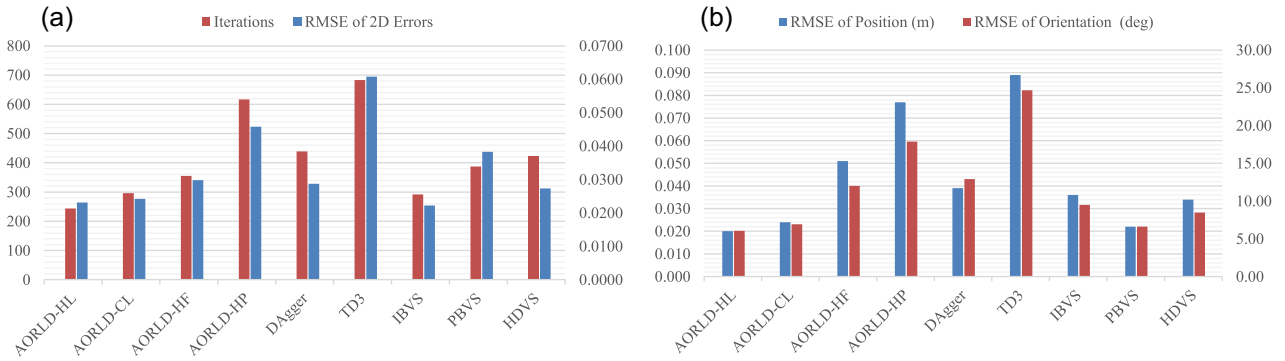

FIGURE 4 Comparison of visual servoing (VS) performance parameters in two-dimensional (2D) and 3D. All comparisons in these figures are based on the average of 10 trials for each method (overall 90) (a) Iterations are taken to complete the VS task and root mean square error (RMSE) of 2D errors in image space for nine different methods, (b) RMSE of position and orientation errors for nine different methods in 3D task space. [Color figure can be viewed at wileyonlinelibrary.com]

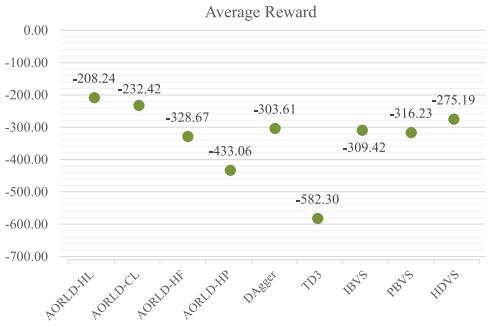

FIGURE 5 Comparison of Average Reward for Different Methods: This figure shows a comparison of the average reward obtained by different methods. AORLD-HL offers the highest average reward followed by AORLD-CL, hybrid decoupled visual servoing, DAgger, image-based visual servoing, position-based visual servoing, AORLD-HF, AORLD-HP, and TD3, indicating the better performance of AORLD-HL compared with other methods. [Color figure can be viewed at wileyonlinelibrary.com]

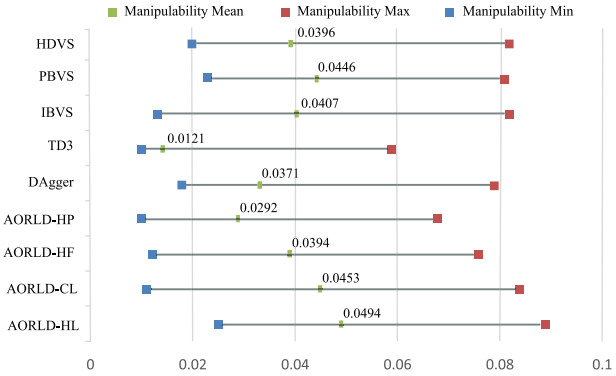

FIGURE 6 Comparison of Manipulability values and ranges for nine methods over 90 trials. [Color figure can be viewed at wileyonlinelibrary.com]

is evident from the figure that the RMSE of errors in AORLD-HL, AORLD-CL, and IBVS are the smallest values among all the other methods, and AORLD-HL performs faster (with fewer iterations) than other methods. To have an optimized performance in VS, both image space and robot space should be taken into account.

Figure 4b illustrates the RMSE of position and orientation for all methods in the 3D task space. From this figure, it is shown that the best performance in 3D task space is achieved by AORLD-HL and PBVS followed by AORLD-CL. This is because the RMSE of orientation and position is lower for AORLD-HL and PBVS compared with the other methods.

Additionally, it is worth noting that the TD3 RL method offers the worst performance in both 2D and 3D tasks compared with the other eight methods. This highlights the fact that the agent is more susceptible to getting stuck in local optima without using the knowledge of any other controllers or demonstrators. In other words, using the action space proposed by other controllers can significantly help the agent find an optimal solutions while avoiding unnecessary explorations.

Overall, the results presented in Figure 4a,b demonstrate that the agent using AORLD-HL is highly effective for both 2D and 3D tasks, while the TD3 without using data of any demonstrators performs poorly in comparison.

Figure 5, compared the mean average reward of different methods for the same 90 trials in Figure 4. The data in Figure 5 shows that AORLD-HL achieves the highest average reward of −208.24, outperforming all other methods. AORLD-CL comes in second place with an average reward of −232.42, followed by HDVS with an average reward of −275.19, DAgger with the average reward of −303.61, IBVS with an average reward of −309.42, PBVS with an average reward of −316.23, AORLD-HF with an average reward of −328.67, AORLD-HP with an average reward of −433.06, and finally TD3 with an average reward of −582.30. The higher average reward indicates the outperforming of AORLD-HL in 2D and 3D space compared with the other methods.

The manipulability of a robot is a key factor in evaluating its performance in task execution. In this study, we compare the manipulability of different methods for robot control, as shown in

TABLE 2 Advantages and disadvantages of Action Optimizer for improving Reinforcement Learning from multi-Demonstrations (AORLD) methods.

AORLD method	Advantages	Disadvantages
AORLD-HL	- Higher performance - Effective action space reduction - Enforces strict constraints - Fast training time	- Increase computational cost
AORLD-CL	- Simplicity in action space reduction - Lower computational cost - Enforces strict constraints	- Exploration space not optimally condensed - Assumes independence between action dimensions
AORLD-HF	- Simplicity, no direct action modification	- Does not enforce strict constraints - Limited reduction of action space - Slow training time
AORLD-HP	- Modification of action space - Enforces constraints	- Training process affected - Increase computational cost

Figure 6. The manipulability values and ranges are calculated by averaging the tracking performance of desired features for the same 90 trials as in Figure 4.

From the results presented in Figure 6, it is observed that the AORLD-HL method provides the highest mean manipulability value of 0.0494, followed by AORLD-CL with 0.0453, PBVS with 0.0446, IBVS with 0.0407, HDVS with 0.0396, AORLD-HF with 0.0394, DAgger with 0.0371 AORLD-HP with 0.0292, and TD3 with 0.0121. The higher manipulability values for AORLD-HL indicate its better performance compared with the other approaches.

Moreover, the results in Figure 6 suggest that the robot has better controllability with the AORLD-HL method while tracking the desired features compared with other methods. This is evident from the fact that AORLD-HL minimum manipulability exceeds that of other methods, showcasing a higher lower bound. Moreover, the maximum manipulability achieved by AORLD-HL surpasses that of other methods, indicating a superior upper bound. Additionally, the average manipulability of AORLD-HL is higher that of other methods, emphasizing its overall superiority in manipulability across the board, indicating that the AORLD-HL method provides better control over the motion of the robot.

Overall, from the date of Figures 4–6, the TD3 agent which is trained with AORLD-HL achieves the best overall performance over the image space and Cartesian space, also suggests the best controllability compared with other approaches. A comprehensive comparison of the four proposed AORLD methods in terms of their advantages and disadvantages is illustrated in Table 2.

4.2 | Real world results

In the previous sections, we presented the theoretical foundations and simulation-based evaluation of our AORLD method.

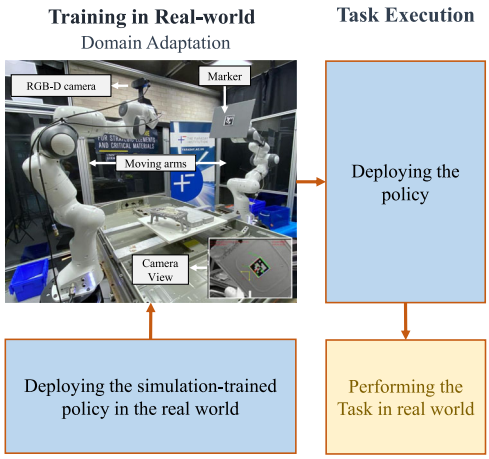

FIGURE 7 Transferring knowledge learned in the simulation to the real world using Domain Adaptation (Transfer Learning). [Color figure can be viewed at wileyonlinelibrary.com]

Results reveal the effectiveness of our proposed action bounding method in terms of accelerating the training process and enhancing cumulative rewards. Building upon this finding, we further investigated the practical applicability of AORLD in the real world. Our experiments began with the training of agents in a simulated environment, where AORLD-HL demonstrated superior performance compared with other alternative methods (Figures 3–6). However, transitioning directly from simulation to the real world is challenging due to the variations between these domains. To address this, alongside our use of Domain Randomization during the simulation, we employed a Domain Adaptation approach (Transfer learning) to further adapt our trained policies for the real world (Aflakian et al., 2023) (Figure 7). As illustrated in Figure 8, we initiated this adaptation

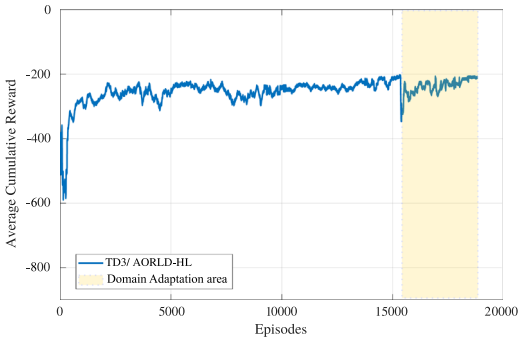

FIGURE 8 Training progress while leveraging AORLD-HL to constrain the action space and employing Transfer learning for real world adaptation. [Color figure can be viewed at wileyonlinelibrary.com]

phase by allowing the agent to continue training until it reached the specified reward threshold in the simulated environment. Once this threshold was achieved (in this case, −200), we transferred the trained policy to the real world, marked by the yellow area in Figure 8. This phase, which we refer to as domain adaptation, involved further training over 4000 additional episodes in the real-world environment. The drop in cumulative rewards during domain adaptation is a natural consequence of the environment shift (transitioning from a simulated domain to the real world). During this phase of domain adaptation, the agent needs time to recalibrate its policies to effectively operate in the real-world setting. This adjustment period can result in lower rewards until the agent successfully adapts and fine-tunes its behavior to accommodate the unique challenges posed by the real-world environment.

Table 3 presents a comparison of performance metrics among various VS techniques and the proposed AORLD-HL. These results stem from 40 experiments (10 experiments per method), each initiated from one of 10 randomly selected initial positions for the tracking. The selection criteria ensured that all four features remained within the camera's field of view.

As depicted in Table 3, AORLD-HL demonstrates superior performance in the image space. It exhibits the smallest RMSE and accomplishes the task with fewer iterations compared with other methods. Additionally, AORLD-HL and IBVS exhibit smaller ranges of feature errors, suggesting a reduced risk of losing track of the target object within the camera's field of view, a clear advantage over PBVS and HDVS. Additionally, the policy trained with AORLD-HL follows a shorter camera path on average (0.706 m) than IBVS (0.942 m), HDVS (0.917 m), and PBVS (0.772 m), underscoring a more optimized trajectory achieved through our proposed training technique.

Analyzing Table 3 further, the mean manipulability values for the same number of experiments and initial conditions are as follows: AORLD-HL (0.0521), IBVS (0.0407), PBVS (0.0446), and HDVS (0.0396). Consequently, AORLD-HL exhibits a higher average manipulability compared with the other methods, indicating better controllability when tracking desired features during VS tasks.

TABLE 3 The real world comparison of visual servoing (VS) performance.

Method	RMSE of 2D errors	Feature error range	RMSE of position (*m*)	RMSE of orientation (°)	Camera traveled distance (*m*)	Manipulability mean	Manipulability range	Iterations	Average reward
HDVS	0.0273	[−0.45, 0.49]	0.034	8.41	0.917	0.0396	[0.021, 0.081]	424	−275.19
PBVS	0.0383	[−0.44, 0.51]	**0.022**	**6.54**	0.722	0.0446	[0.024, 0.080]	387	−316.23
IBVS	0.0222	[**−0.36, 0.31**]	0.036	9.43	0.942	0.0407	[0.014, 0.081]	253	−309.42
AORLD-HL	**0.0209**	[−0.30, 0.34]	0.027	6.61	**0.706**	**0.0521**	[**0.031, 0.084**]	**189**	**−237.42**

Note: The bold value in each column indicates the best performance among all the compared methods.
Abbreviations: 2D, two dimensional; HDVS, hybrid decoupled visual servoing; IBVS, image-based visual servoing; PBVS, position-based visual servoing; RMSE, root mean square error.

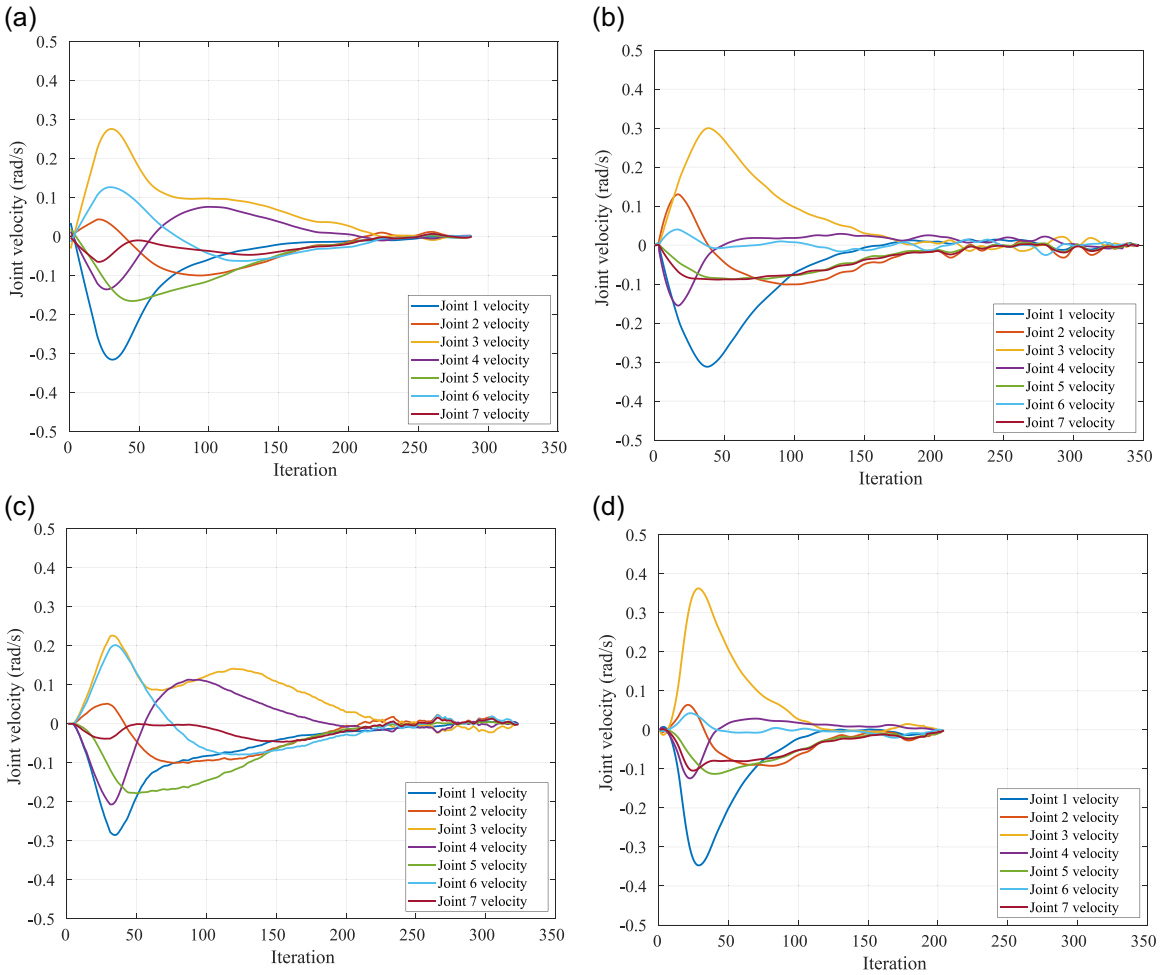

FIGURE 9 The real-world joint velocity comparison of different methods to perform one random trial (a) hybrid decoupled visual servoing, (b) position-based visual servoing, (c) image-based visual servoing, (d) AORLD-HL. [Color figure can be viewed at wileyonlinelibrary.com]

In summary, Table 3 highlights the superior performance (higher average reward) of AORLD-HL, trained with our proposed method, across both image space and Cartesian space.

Figure 9 illustrates the comparison of each controller's action vector for one random trial among different methods. This sample is part of the data set used to generate Table 3. The plots illustrate the joint velocity vector for each method during the task of tracking an object with a tag in the real world. It is evident from the plots that the agent trained with the combination of TD3 and AORLD-HL outperformed other methods, completing the tracking task in approximately 200 iterations.

5 | CONCLUSION

This paper proposed a learning-based online action-policy optimizer named AORLD. The AORLD technique intelligently limits the action space of the agent, based on demonstrated actions from an ensemble of several different supervisory experts. Thereafter, the agent explores further within this constrained action space, refining its policy to become increasingly optimal with respect to a reward function. The learning process is greatly accelerated, because the policy search space has been reduced by the expert demonstrations.

We demonstrated AORLD in the context of a standard VS task, with TD3 algorithms to train the policy. IBVS, PBVS, and HDVS were defined as a set of expert supervisors for AORLD. We proposed and compared four methods to bound actions online while training. We found using the convex hull with modified loss function (AORLD-HL) is the most effective method for improving the exploration-exploitation trade-off in RL. Our experimental results demonstrate the effectiveness of these methods in improving the average reward progress during training, compared with using no bounding methods. Moreover, the agent trained with AORLD-HL achieves better overall performance in terms of feature trajectories in the 2D image plane, and also robot trajectories in the 3D task space, while also achieving higher Jacobian and manipulability of the robot throughout its motions.

Overall, our study highlights the importance of incorporating prior knowledge into the training process of RL policies to improve their performance, particularly in challenging environments with

high-dimensional action spaces. The AORLD method can be used when there are multiple control methods available to serve as demonstrators (from two to arbitrarily many). AORLD finds a useful trade-off between these experts, while also incorporating the capabilities of RL to enable iterative optimizing of policies with respect to a reward function. The methods presented in this study provide a promising approach for addressing this challenge and can be applied in various RL applications. In future work, we aim to introduce haptic experts in our proposed optimization method to correct and improve the control signals from a human operator during tele-operation tasks. Furthermore, AORLD could be integrated with multiagent RL, to significantly reduce training episodes by intelligently limiting the exploration bounds of each agent.

ACKNOWLEDGMENTS

This work was supported in part by the project called "Research and Development of a Highly Automated and Safe Streamlined Process for Increase Lithium-ion Battery Repurposing and Recycling" (REBELION) under Grant 101104241 and in part by the UK Research and Innovation (UKRI) project "Reuse and Recycling of Lithium-Ion Batteries" (RELIB) under RELiB2 Grant FIRG005 and RELIB3 Grant FIRG057.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

Aflakian, A., Rastegarpanah, A. & Stolkin, R. (2023) Boosting performance of visual servoing using deep reinforcement learning from multiple demonstrations. *IEEE Access*, 11, 26512–26520.

Baerlocher, P. & Boulic, R. (1998) Task-priority formulations for the kinematic control of highly redundant articulated structures. In: *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No. 98CH36190)*. vol. 1. Victoria, BC, Canada, IEEE, pp. 323–329.

Bellman, R. (1957) *Dynamic programming*. Princeton University Press, Princeton, NJ.

Castelli, F., Michieletto, S., Ghidoni, S. & Pagello, E. (2017) A machine learning-based visual servoing approach for fast robot control in industrial setting. *International Journal of Advanced Robotic Systems*, 14(6), 1729881417738884.

Chaumette, F. (1998) Potential problems of stability and convergence in image-based and position-based visual servoing. In: *The confluence of vision and control*, Springer, pp. 66–78.

Chesi, G., Hashimoto, K., Prattichizzo, D. & Vicino, A. (2004) Keeping features in the field of view in eye-in-hand visual servoing: a switching approach. *IEEE Transactions on Robotics*, 20(5), 908–914.

Corke, P.I. & Hutchinson, S.A. (2001) A new partitioned approach to image-based visual servo control. *IEEE Transactions on Robotics and Automation*, 17(4), 507–515.

Franks, J., Huo, L. & Baron, L. (2008) The joint-limits and singularity avoidance in robotic welding. *Industrial Robot: An International Journal*, 35(5), 456–464.

Fujimoto, S., Hoof, H. & Meger, D. (2018) Addressing function approximation error in actor-critic methods. In: *International Conference on Machine Learning*. PMLR, pp. 1587–1596.

Gans, N.R. & Hutchinson, S.A. (2007) Stable visual servoing through hybrid switched-system control. *IEEE Transactions on Robotics*, 23(3), 530–540.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B. et al. (2018) Deep q-learning from demonstrations. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. vol. 32.

Ho, J. & Ermon, S. (2016) Generative adversarial imitation learning. *Advances in Neural Information Processing Systems*, 29, 4565–4573.

Hoque, R., Balakrishna, A., Novoseller, E., Wilcox, A., Brown, D.S. & Goldberg, K. (2021) Thriftydagger: budget-aware novelty and risk gating for interactive imitation learning. arXiv preprint arXiv:2109.08273.

Hu, G., Gans, N.R. & Dixon, W.E. (2009) *Adaptive visual servo control*. Florida: Citeseer. pp. 42-63.

Hua, J., Zeng, L., Li, G. & Ju, Z. (2021) Learning for a robot: deep reinforcement learning, imitation learning, transfer learning. *Sensors*, 21(4), 1278.

Jin, Z., Wu, J., Liu, A., Zhang, W. -A. & Yu, L. (2021) Policy-based deep reinforcement learning for visual servoing control of mobile robots with visibility constraints. *IEEE Transactions on Industrial Electronics*, 69(2), 1898–1908.

Jing, M., Ma, X., Huang, W., Sun, F., Yang, C., Fang, B. et al. (2020) Reinforcement learning from imperfect demonstrations under soft expert guidance. In: *Proceedings of the AAAI conference on artificial intelligence*. vol. 34. pp. 5109–5116.

Kang, B., Jie, Z. & Feng, J. (2018) Policy optimization with demonstrations. In: *International Conference on Machine Learning*. PMLR, pp. 2469–2478.

Kelly, M., Sidrane, C., Driggs-Campbell, K. & Kochenderfer, M.J. (2019) Hg-dagger: interactive imitation learning with human experts. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 8077–8083.

Krishnan, S., Garg, A., Liaw, R., Thananjeyan, B., Miller, L., Pokorny, F.T. et al. (2019) Swirl: a sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards. *The International Journal of Robotics Research*, 38(2–3), 126–145.

Kumar, V., Gupta, A., Todorov, E. & Levine, S. (2016) Learning dexterous manipulation policies from experience and imitation. arXiv preprint arXiv:1611.05095.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y. et al. (2015) Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

Malis, E., Chaumette, F. & Boudet, S. (1999) 2 1/2 d visual servoing. *IEEE Transactions on Robotics and Automation*, 15(2), 238–250.

Massoud Farahmand, A., Shademan, A., Jagersand, M. & Szepesvári, C. (2009) Model-based and model-free reinforcement learning for visual servoing. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE, pp. 2917–2924.

MATLAB (2021) *version 9.9.0 (R2020b)*. Natick, MA: The MathWorks Inc.

Ramírez, J., Yu, W. & Perrusquía, A. (2022) Model-free reinforcement learning from expert demonstrations: a survey. *Artificial Intelligence Review*, 55(4), 3213–3241.

Rastegarpanah, A., Aflakian, A. & Stolkin, R. (2021a) Improving the manipulability of a redundant arm using decoupled hybrid visual servoing. *Applied Sciences*, 11(23), 11566.

Rastegarpanah, A., Aflakian, A. & Stolkin, R. (2021b) Optimized hybrid decoupled visual servoing with supervised learning. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 236(2), 338–354.

Rastegarpanah, A., Hathaway, J. & Stolkin, R. (2021c) Vision-guided mpc for robotic path following using learned memory-augmented model. *Frontiers in Robotics and AI*, 8, 688275.

Ross, S., Gordon, G. & Bagnell, D. (2011a) A reduction of imitation learning and structured prediction to no-regret online learning. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence*

and Statistics. JMLR Workshop and Conference Proceedings, pp. 627–635.

Ross, S., Gordon, G.J. & Bagnell, J.A. (2011b) No-regret reductions for imitation learning and structured prediction. In: *In AISTATS*. Citeseer.

Sampedro, C., Rodriguez-Ramos, A., Gil, I., Mejias, L. & Campoy, P. (2018) Image-based visual servoing controller for multirotor aerial robots using deep reinforcement learning. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 979–986.

Sun, W., Bagnell, J.A. & Boots, B. (2018) Truncated horizon policy search: Combining reinforcement learning & imitation learning. arXiv preprint arXiv:1805.11240.

Takeda, T., Hirata, Y. & Kosuge, K. (2007) Dance step estimation method based on hmm for dance partner robot. *IEEE Transactions on Industrial Electronics*, 54(2), 699–706.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W. & Abbeel, P. (2017) Domain randomization for transferring deep neural networks from simulation to the real world. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 23–30.

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B. et al. (2017) Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. arXiv preprint arXiv: 1707.08817.

Yang, C., Ma, X., Huang, W., Sun, F., Liu, H., Huang, J. et al. (2019) Imitation learning from observations by minimizing inverse dynamics disagreement. *Advances in Neural Information Processing Systems*, 32.

Yuan, E., Cheng, S., Wang, L., Song, S. & Wu, F. (2023) Solving job shop scheduling problems via deep reinforcement learning. *Applied Soft Computing*, 143, 110436.

Zhu, Z. & Hu, H. (2018) Robot learning from demonstration in robotic assembly: a survey. *Robotics*, 7(2), 17.

How to cite this article: Aflakian, A., Rastegarpanah, A., Hathaway, J. & Stolkin, R. (2024) An online hyper-volume action bounding approach for accelerating the process of deep reinforcement learning from multiple controllers. *Journal of Field Robotics*, 1–15. https://doi.org/10.1002/rob.22355