An Online Hyper-volume Action Bounding Approach for Accelerating the Process of Deep Reinforcement Learning from Multiple Controllers

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

[1]University of Birmingham

May 3, 2023

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. 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 with and without using the expert demonstrators. Our experiments show that using the convex hull with a modified loss function significantly improves training progress. Furthermore, we demonstrate faster VS error convergence while maintaining higher manipulability of the arm, compared to classical image-based VS, position-based VS, and hybrid-decoupled VS.

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]

***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. 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 with and without using the expert demonstrators. Our experiments show that using the convex hull with a modified loss function significantly improves training progress. Furthermore, we demonstrate faster VS error convergence while maintaining higher manipulability of the arm, compared to classical image-based VS, position-based VS, and hybrid-decoupled VS.**

***Index Terms*—Reinforcement learning, Multi-expert demonstrations, Optimization technique, Online learning, Imitation learning.**

I. Introduction

Recent advances in deep learning and RL research have enabled robots to handle increasingly challenging tasks [1]. In a RL paradigm, an agent attempts to maximise 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 [2]. Additional challenges arise from the agent lacking a-priori data or knowledge about its environment. To alleviate these problems, behavioural 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 optimisation, to find an optimal policy [3]. This process is known as Reinforcement Learning from Demonstrations (RLfD) [4].

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 generalises to handle arbitrary object motions, while exploiting feedback data comprising robot states and camera images. Reinforcement Learning (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. Our approach is "online" in the sense that the agent action space would be modified in real-time during the training.

There are two potential approaches for incorporating knowledge from expert demonstrations in RL: prior knowledge, comprising demonstrations prior to RL refinement; and online knowledge, in which case demonstrations are occasionally presented while the RL iterations are in progress [5]. The online method can significantly enhance the learned policy's convergence towards an expected performance level while lowering the likelihood of distributional mismatch compared to the prior knowledge approach [6]. 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 generalise to different scenarios; and the limitation of the agent's exploration to blindly following the demonstrator [7]–[9]. To avoid unnecessary exploration while also overcoming the problem of the agent overly following the expert behaviour, we present an online Action Optimizer for improving Reinforcement Learning from multi-Demonstrations (AORLD) (detailed in Section II-B). 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 Visual Servoing (VS) task, detailed in Section II-B. Based on the knowledge of several controllers, we explore four methods for bounding the actions of the RL agent during training. The first method involves constraining the action space to an online convex hull generated from the knowledge of controllers (AORLD-HL). We modify the loss function to penalize the

1 Department of Metallurgy & Materials Science, University of Birmingham, Birmingham, B15 2TT.

2 The Faraday Institution, Quad One, Harwell Science and Innovation Campus, Didcot, OX11 0RA, UK.

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

* Corresponding author: Alireza Rastegarpanah, Email: a.rastegarpanah@bham.ac.uk

agent for taking actions outside the generated hypervolume. The second method involves generating an online hypercube from the knowledge of experts to constrain the action space (AORLD-CL). We again modify the loss function to penalize the agent for taking actions outside the 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. 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:

- AORLD adaptively constrains the agent's action spaces by exploiting demonstrations from different "expert" controllers, improving training efficiency.
- AORLD enables learning of policies that do not directly conform to any single demonstration, hence they are robust against errors or imperfections in any individual demonstration.
- The AORLD approach is generic, and can be incorporated into a wide variety of multi-agent and other RLfD algorithms, for many different applications.

The remainder of this study is structured as follows: Section II 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 III introduces the simulation environment for a VS application. Section IV presents the results of experiments, showing how the trained policy improves VS task performance. Section V provides concluding remarks, and also offers suggestions for extending AORLD to other applications, and combining it with other RLfD methods.

A. Related work

Early RLfD algorithms are classified into three main groups: Behavior Cloning (BC), Generative Adversarial Imitation Learning (GAIL), and Inverse Reinforcement Learning (IRL) [10]. 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 [7]. 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 [8]. 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 [9]. 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 minima [9]. Also, BC methods suffer from data mismatch and compounding error issues. Consequently, the DAgger algorithm was developed to tackle this problem [11]. 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 generalised to different scenarios, and the approach is limited to learning from the expert and cannot surpass its performance [12]. 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 [13], [14].

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 behaviours when feedback from the environment is scarce or even missing [15], [16]. They specifically reshape the reward function in RL by adding another term to encourage expert exploration. While rewarding expert-like activities might assist in minimising unnecessary exploration, applying such rewards throughout the learning phase can be troublesome with imperfect demonstrations. There is no guarantee that limiting divergence from expert behaviour will result in an improved agent policy [17]. Unlike the aforementioned methods that aimed to replace existing learning from demonstration strategies, we propose a method which is complementary to established RLfD methods. We combined aspects of both RL and imitation learning in a novel online manner. For the chosen application of VS, we use the joint velocities of three supervisory "experts": Image-Based VS (IBVS), Position-Based VS (PBVS), and Hybrid decoupled VS (HDVS) [18]. These are used in AORLD to constrain the action space of the agent and avoid redundant actions. 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. Subsequently, the policy was further trained in the real world to become more robust against real world environmental factors, the so-called Domain Adaptation (DA) approach. We demonstrate a learning process that is not only greatly accelerated (compared to when there is no demonstrator), but the policy will also inherit the high performance of each expert technique and improve its behaviour. Furthermore, using DR 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 visual servoing application.

II. Methodology

A. 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

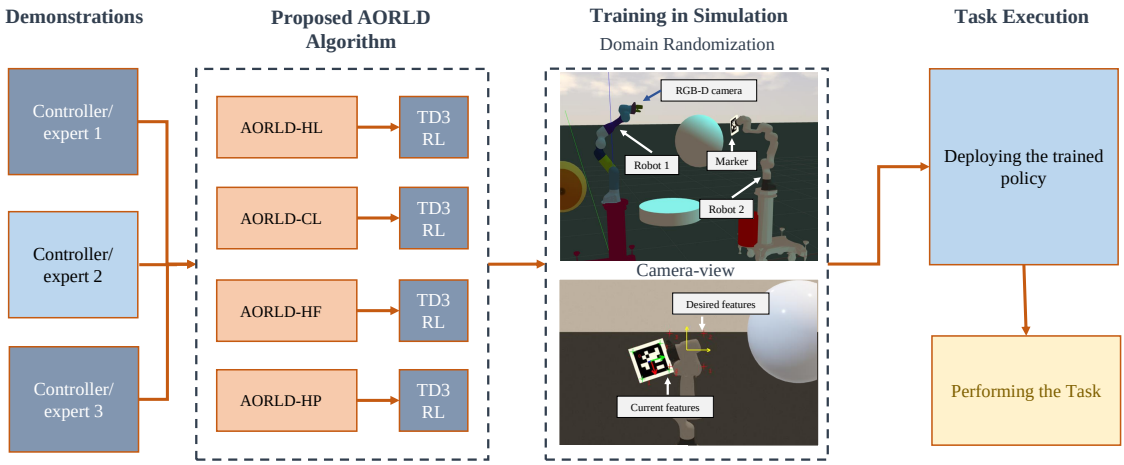

Fig. 1. The outline of the proposed online action-optimizer combined with the RL method for a VS application. The Twin Delayed Deep Deterministic Policy Gradient (TD3) agent was trained by using the Domain Randomization (DR) method. The AORLD method employs a combination of three different methods (PBVS, IBVS, and HDVS) 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.

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. During exploration, the agent is evaluated by the discounted sum of rewards (return) function, defined as follows:

$$R(s,a) = \sum_{k=0}^{\infty} \gamma^k r_k \tag{1}$$

where $\gamma \in (0,1]$ is the discount factor, and k is the number of episodes. The agent seeks to maximise the cumulative reward over time by maximising the anticipation of such long-term returns from each state.

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 [19]. The Bellman equation is a fundamental concept in reinforcement learning that helps predict the expected cumulative reward of being in a state and taking an action. To learn the Q-functions, TD3 uses the mean square Bellman error, which measures the difference between the predicted Q-value and the actual Q-value for a state-action pair. By using two Q-functions, TD3 aims to reduce the overestimation bias that can occur in single Q-function methods. The use of target policy smoothing, which adds noise to the actions selected by the actor, further enhances the stability and performance of the learned policy.

To form the Q-learning target in the TD3 reinforcement learning 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 ($a_{Low} \leq a \leq a_{High}$). Mathematically, the target actions can be expressed as [20]:

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{min}, a_{max}\right) \tag{2}$$

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.

Clipped double-Q learning is a method employed in the TD3 reinforcement learning algorithm to alleviate the overestimation issue in the Q-function. The approach trains two Q-functions, labelled 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:

$$y(r,s',d) = r + \gamma(1-d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')) \tag{3}$$

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 (2).

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 [21].

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}. Mathematically, the policy is learned by solving the following optimization problem:

$$\max_{\theta} \underset{s\sim\mathcal{D}}{\text{E}} \left[Q_{\phi_1}(s, \mu_\theta(s))\right] \tag{4}$$

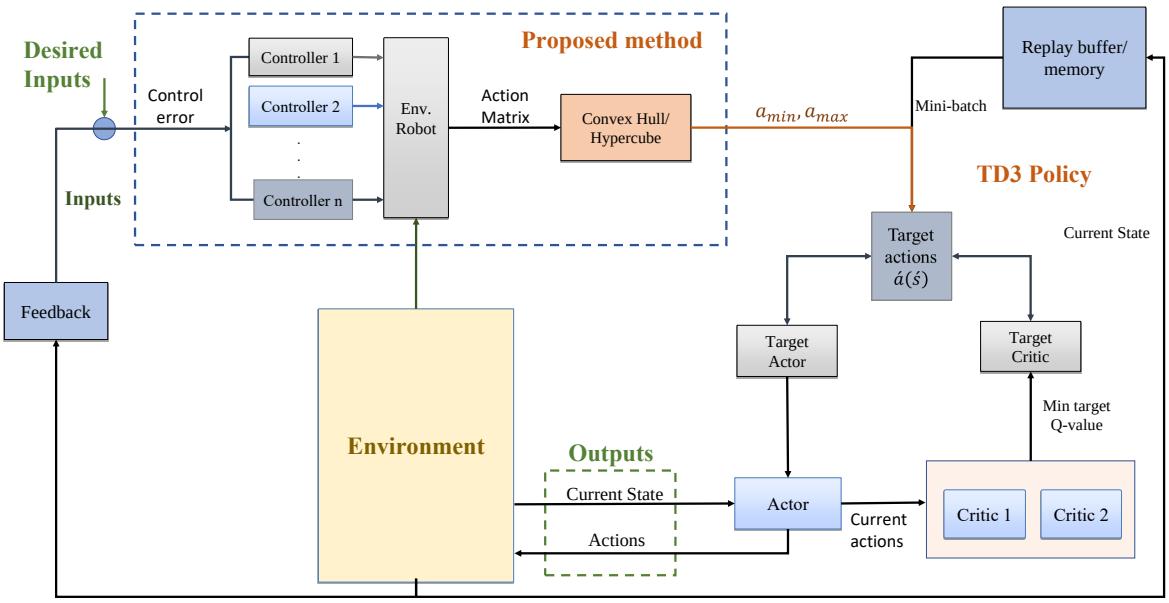

Fig. 2. Structure of AORLD integrated with TD3 RL. The proposed block diagram in this study takes in current and desired features extracted from the vision sensor as inputs. Then, during each episode, the knowledge from HDVS, PBVS, and IBVS approaches is utilized to restrict the action space. The joint velocity actions are then applied to the training environment, and the average rewards are computed accordingly.

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 RL algorithm used in this approach is TD3 (Twin Delayed Deep Deterministic Policy Gradient). 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.

In each iteration of the training process, the AORLD approach samples a goal and an initial state and generates an action using the RL algorithm. The generated action is then constrained by one of the four abovementioned methods, depending on the chosen method. The resulting action is executed in the environment, and the resulting state and reward are stored in a replay buffer.

The AORLD approach then samples additional goals from the replay buffer and generates actions for these goals. The resulting states and rewards are also stored in the replay buffer. The replay buffer is then sampled to create mini-batches for optimizing the policy using TD3.

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 (2), 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 reinforcement learning agent. Let k be the set of indices that define the convex hull of A, i.e., $k = \text{convhulln}(A)$. 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:

$$a_{min,i} = \min_{j \in k} a_{j,i} \quad \text{and} \quad a_{max,i} = \max_{j \in k} a_{j,i} \tag{5}$$

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 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

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 $g' \in G$ **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

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:

$$a_{min,i} = min(a_{ex1}[i], a_{ex2}[i], ..., a_{exn}[i] \tag{6}$$

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

$$a_{max,i} = max(a_{ex1}[i], a_{ex2}[i], ..., a_{exn}[i] \tag{7}$$

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.

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

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

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

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.

$$proj = u + v\frac{(a-u)^T \cdot v}{|v|^2} \tag{8}$$

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 test our suggested approaches in the following subsection.

B. Visual servoing

VS is a widely used technique in the field of robot vision that translates visual errors into actuator commands [22]. However, conventional VS methods still face several limitations in terms of stability, convergence, and gain selection, as

highlighted in [23]. 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 [24].

The literature has discussed advanced techniques to circumvent the problems associated with conventional VS methods [25]–[32]. 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 visual servoing methods, namely image-based visual servoing (IBVS), position-based visual servoing (PBVS), and hybrid decoupled visual servoing (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.

In order to link pixel velocity to camera velocity in image-based visual servoing (IBVS), an image Jacobian matrix (L_i) is utilized [33]. This matrix relates the camera velocity vector (v_{cam}) to the difference (e_i) between the current and desired position of the i^{th} feature. The control law for IBVS is then calculated [33]:

$$v_{cam} = -k_i L_i^{+} e_i \tag{9}$$

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 [34]. The control law for PBVS is given as follows:

$$v_{cam} = -k_p L_p^{-1} e_p \tag{10}$$

The control gain is represented by k_p, and e_p refers to the difference between the current and desired camera position, measured in the task space. $L_p(t)$ is a 6×6 Euclidean matrix to do the camera 3D position estimation; as defined in [34].

The third approach utilized in our method is HDVS, which is explained in detail in [18]. 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 (11) and (12):

$$v_{xy} = L_{xy}^{+} \{-k_h e - L_r v_r\} \tag{11}$$

$$v_r = L_{Pr}^{+} \{-k_h e_p - L_{Pxy} v_{xy}\} \tag{12}$$

where $v_{xy} = [v_x v_y]^T$, $v_r = [v_z w_x w_y w_z]^T$, k_h is the controller gain, and L_{xy} and L_r are defined in [18]. The data from (11) and (12) 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 ($v_{cam} = [v_{xy} : v_r]^T$) in the HDVS method, the joint velocities ($\dot{q}$) can be computed using the following equation [35]:

$$\dot{q} = J^{+\lambda} \xi_c^e v_{cam}^c \tag{13}$$

where λ is damping factor [35]. The AORLD approach utilizes expert demonstrations for learning decision-making policies and constraining the agent's action space, as described in Section I-A, 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:

$$r_1 = -\sum_{i=1}^{4} \sqrt{(u_i - u_{id})^2 + (v_i - v_{id})^2} \tag{14}$$

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). In order to keep away from joint limits, the second reward function (r_2) is defined as follows [36]:

$$r_2 = -\frac{1}{2n} \sum_{j=1}^{n} \left(\frac{q_j - \bar{q}_j}{q_{jM} - q_{jm}} \right)^2 \tag{15}$$

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 [36]:

$$r_3 = \sqrt{\det\left(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}^T(\boldsymbol{q})\right)} \tag{16}$$

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

$$r = -w_f r_1 - w_q r_2 - w_m \vec{r}_3 \tag{17}$$

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. In the AORLD approach, the robot attains the target joint velocities using the IBVS, PBVS, and HDVS at the start of each step. Next, the algorithm 1 is used to define a set of action bounds (a_{bound}) by merging the results obtained from these three demonstrators.

III. Experimental Setup

To train the policy, a simulation environment was developed using ROS/Gazebo. This approach is more efficient and cost-effective compared to 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 [37].

The TD3 algorithm was implemented as a ROS node, and Matlab Reinforcement Learning Toolbox [38] 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.

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 1.80GHz and a turbo boost frequency of 4.9GHz, and 16.0 GB of RAM.

IV. Results and Discussion

To evaluate the effectiveness of the proposed approach, five 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. Finally, the fifth 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 five 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 I 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 Fig. 3 that the AORLD-HL algorithm requires a smaller number of episodes to attain a satisfactory average reward compared to all other four methods. The less the agent must interact with the environment, the faster it will learn the task.

TABLE I
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 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 Fig. 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.

The agent trained with AORLD-HP algorithm had an average reward of around -400, which is lower than the first two methods (Fig. 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. 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 five trained policies using effective parameters. To obtain these parameters, we carried out 80 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 Figs. 4,5, and 6.

Fig. 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 eight different methods, including AORLD-HL, AORLD-CL, AORLD-HF, AORLD-HP, IBVS, PBVS, HDVS, and the TD3. It 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 optimised performance in VS, both image space and robot space should be taken into account.

Fig. 4b illustrates the RMSE of position and orientation for all eight 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

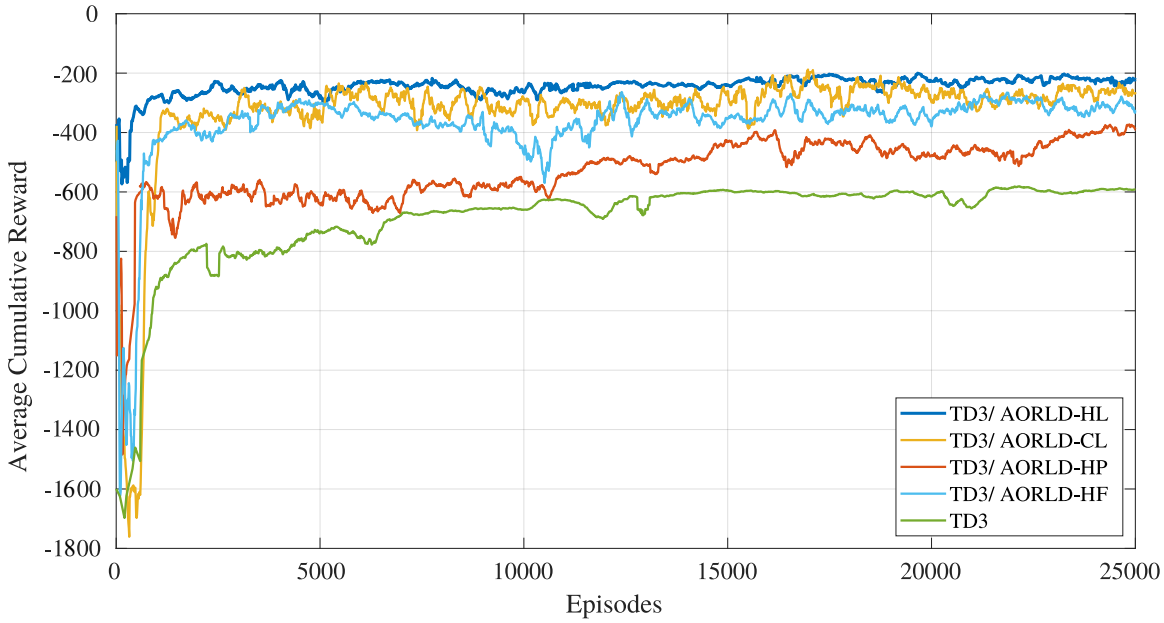

Fig. 3. Comparison of training progress across 5 different methods for action space constraint in reinforcement learning. The graph shows the average reward per episode for each method over the course of training. AORLD-HL and AORLD-CL show faster learning and higher average rewards, possibly due to the hard constraint on actions. AORLD-HF, AORLD-HP, and the agent without action constraints show slower learning and lower average rewards.

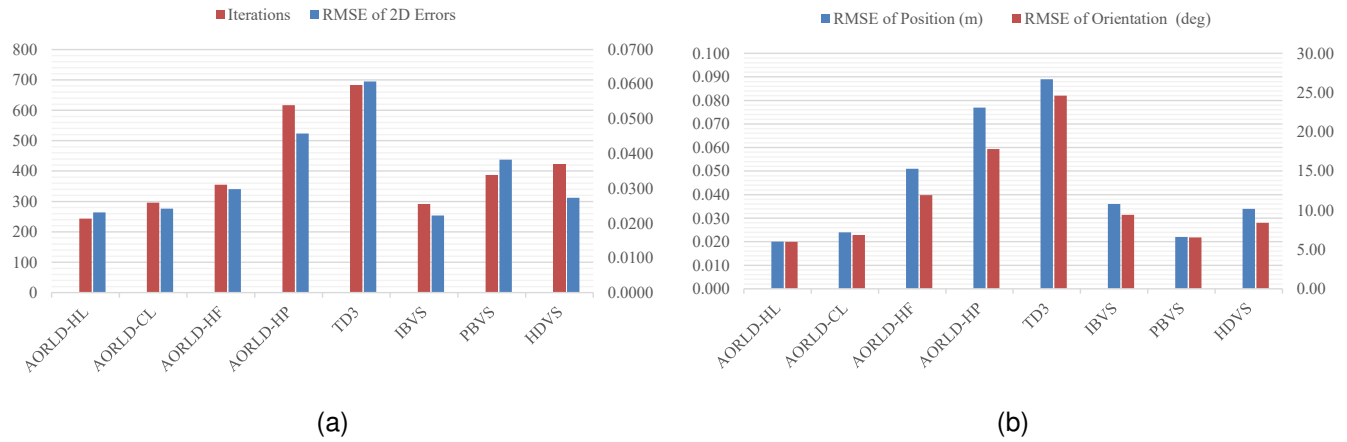

Fig. 4. Comparison of VS performance parameters in 2D and 3D. (a) Iterations are taken to complete the VS task and RMSE of 2D errors in image space for eight different methods. (b) RMSE of position and orientation errors for eight different methods in 3D task space. All comparisons in these figures are based on the average of 10 trials for each method (overall 80).

because the RMSE of orientation and position is lower for AORLD-HL and PBVS compared to the other 6 methods.

Additionally, it is worth noting that the TD3 RL method offers the worst performance in both 2D and 3D tasks compared to the other 7 methods. This highlights the fact that the agent is more susceptible to getting stuck in local minima 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 to find its optimal solutions while avoiding unnecessary explorations.

Overall, the results presented in Figs. 4a and 4b 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.

Fig. 5, compared the mean average reward of different methods for the same 80 trials in Fig. 4. The data in Fig. 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, 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 to the other methods.

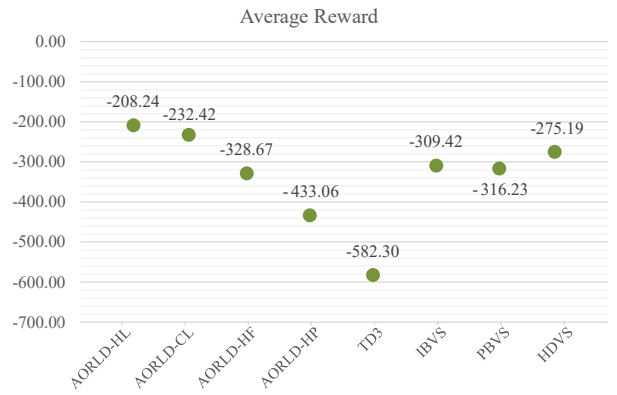

Fig. 5. Comparison of Average Reward for Different Methods: This figure shows a comparison of the average reward obtained by different methods. AORLD-HL offers the highest average reward followed by AORLD-CL, HDVS, IBVS, PBVS, AORLD-HF, AORLD-HP, and TD3, indicating the better performance of AORLD-HL compared to other methods.

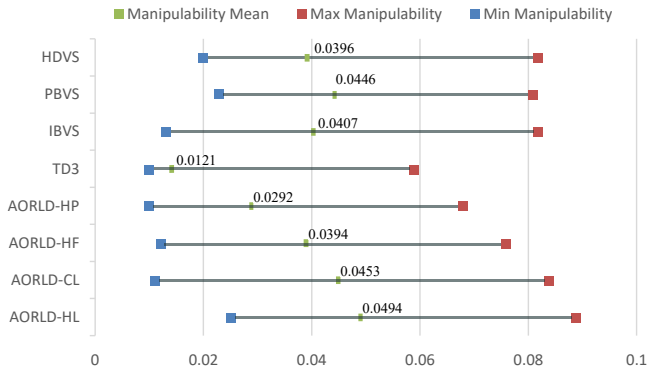

Fig. 6. Comparison of Manipulability values and ranges for eight methods over 80 trials.

The manipulability of a robot is a key factor in evaluating its performance in task execution. In this study, we compare the manipulability of eight different methods for robot control, as shown in Fig. 6. The manipulability values and ranges are calculated by averaging the tracking performance of desired features for the same 80 trials as in Fig. 4.

From the results presented in Fig. 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, AORLD-HP with 0.0292, and TD3 with 0.0121. The higher manipulability values for AORLD-HL indicate its better performance compared to the other approaches.

Moreover, the results in Fig. 6 suggest that the robot has better controllability with the AORLD-HL method while tracking the desired features compared to other methods. This is evident from the fact that the lower and higher manipulability bound in agent-3 from the manipulability range is bigger than the other seven methods, indicating that the AORLD-HL method provides better control over the motion of the robot.

Overall, from the date of Fig. 4, 5 and 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 to other approaches. In our earlier work [32], we integrated our method with domain adaptation to enable the agent to train in the real world setting as well as simulation. However, the present study aims to compare and identify the most effective ways to constrain the action space and facilitate the exploration of potential action vectors by the agent in simulation. In future works, we plan to develop our methods with domain adaptation further and implement them on real robots.

V. 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 to 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 optimising 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 multi-agent RL, to significantly reduce training episodes by intelligently limiting the exploration bounds of each agent.

Acknowledgment

This research was conducted as part of the project called “Reuse and Recycling of Lithium-Ion Batteries” (RELIB). This work was supported by the Faraday Institution [grant number FIRG005].

REFERENCES

[1] J. Hua, L. Zeng, G. Li, and Z. Ju, "Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning," *Sensors*, vol. 21, no. 4, p. 1278, 2021.

[2] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[3] Z. Zhu and H. Hu, "Robot learning from demonstration in robotic assembly: A survey," *Robotics*, vol. 7, no. 2, p. 17, 2018.

[4] M. Jing, X. Ma, W. Huang, F. Sun, C. Yang, B. Fang, and H. Liu, "Reinforcement learning from imperfect demonstrations under soft expert guidance," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 5109–5116.

[5] J. Ramírez, W. Yu, and A. Perrusquía, "Model-free reinforcement learning from expert demonstrations: a survey," *Artificial Intelligence Review*, vol. 55, no. 4, pp. 3213–3241, 2022.

[6] S. Ross, G. Gordon, and D. Bagnell, "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, 2011, pp. 627–635.

[7] T. Takeda, Y. Hirata, and K. Kosuge, "Dance step estimation method based on hmm for dance partner robot," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 2, pp. 699–706, 2007.

[8] S. Krishnan, A. Garg, R. Liaw, B. Thananjeyan, L. Miller, F. T. Pokorny, and K. Goldberg, "Swirl: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards," *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 126–145, 2019.

[9] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, pp. 4565–4573, 2016.

[10] V. Kumar, A. Gupta, E. Todorov, and S. Levine, "Learning dexterous manipulation policies from experience and imitation," *arXiv preprint arXiv:1611.05095*, 2016.

[11] S. Ross, G. J. Gordon, and J. A. Bagnell, "No-regret reductions for imitation learning and structured prediction," in *In AISTATS*. Citeseer, 2011.

[12] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, *et al.*, "Deep q-learning from demonstrations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

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

[14] R. Hoque, A. Balakrishna, E. Novoseller, A. Wilcox, D. S. Brown, and K. Goldberg, "Thriftydagger: Budget-aware novelty and risk gating for interactive imitation learning," *arXiv preprint arXiv:2109.08273*, 2021.

[15] W. Sun, J. A. Bagnell, and B. Boots, "Truncated horizon policy search: Combining reinforcement learning & imitation learning," *arXiv preprint arXiv:1805.11240*, 2018.

[16] B. Kang, Z. Jie, and J. Feng, "Policy optimization with demonstrations," in *International conference on machine learning*. PMLR, 2018, pp. 2469–2478.

[17] C. Yang, X. Ma, W. Huang, F. Sun, H. Liu, J. Huang, and C. Gan, "Imitation learning from observations by minimizing inverse dynamics disagreement," *Advances in neural information processing systems*, vol. 32, 2019.

[18] A. Rastegarpanah, A. Aflakian, and R. Stolkin, "Improving the manipulability of a redundant arm using decoupled hybrid visual servoing," *Applied Sciences*, vol. 11, no. 23, p. 11566, 2021.

[19] R. Bellman, "Dynamic programming, princeton univ," *Press Princeton, New Jersey*, 1957.

[20] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[22] A. Rastegarpanah, A. Aflakian, and R. Stolkin, "Optimized hybrid decoupled visual servoing with supervised learning," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, p. 09596518211028379, 2021.

[23] C. Sampedro, A. Rodriguez-Ramos, I. Gil, L. Mejias, and P. Campoy, "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, 2018, pp. 979–986.

[24] F. Chaumette, "Potential problems of stability and convergence in image-based and position-based visual servoing," in *The confluence of vision and control*. Springer, 1998, pp. 66–78.

[25] G. Chesi, K. Hashimoto, D. Prattichizzo, and A. Vicino, "Keeping features in the field of view in eye-in-hand visual servoing: A switching approach," *IEEE Transactions on Robotics*, vol. 20, no. 5, pp. 908–914, 2004.

[26] P. I. Corke and S. A. Hutchinson, "A new partitioned approach to image-based visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, pp. 507–515, 2001.

[27] N. R. Gans and S. A. Hutchinson, "Stable visual servoing through hybrid switched-system control," *IEEE Transactions on Robotics*, vol. 23, no. 3, pp. 530–540, 2007.

[28] A. Rastegarpanah, J. Hathaway, and R. Stolkin, "Vision-guided mpc for robotic path following using learned memory-augmented model," *Frontiers in Robotics and AI*, vol. 8, 2021.

[29] F. Castelli, S. Michieletto, S. Ghidoni, and E. Pagello, "A machine learning-based visual servoing approach for fast robot control in industrial setting," *International Journal of Advanced Robotic Systems*, vol. 14, no. 6, p. 1729881417738884, 2017.

[30] Z. Jin, J. Wu, A. Liu, W.-A. Zhang, and L. Yu, "Policy-based deep reinforcement learning for visual servoing control of mobile robots with visibility constraints," *IEEE Transactions on Industrial Electronics*, 2021.

[31] A. massoud Farahmand, A. Shademan, M. Jagersand, and C. Szepesvári, "Model-based and model-free reinforcement learning for visual servoing," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 2917–2924.

[32] A. Aflakian, A. Rastegarpanah, and R. Stolkin, "Boosting performance of visual servoing using deep reinforcement learning from multiple demonstrations," *IEEE Access*, 2023.

[33] G. Hu, N. R. Gans, and W. E. Dixon, "Adaptive visual servo control." 2009.

[34] E. Malis, F. Chaumette, and S. Boudet, "2 1/2 d visual servoing," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 2, pp. 238–250, 1999.

[35] P. Baerlocher and R. Boulic, "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. IEEE, 1998, pp. 323–329.

[36] J. Franks, L. Huo, and L. Baron, "The joint-limits and singularity avoidance in robotic welding," *Industrial Robot: An International Journal*, 2008.

[37] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "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, 2017, pp. 23–30.

[38] MATLAB, *version 9.9.0 (R2020b)*. Natick, Massachusetts: The MathWorks Inc., 2021.