Robotic approach trajectory using Reinforcement Learning with Dual Quaternions

Daniel Frau-Alfaro
AUtomatics, RObotics, and Artificial Vision Lab
University Institute for Computer Research
San Vicente, Spain
0009-0000-4098-3783

Santiago T. Puente
AUtomatics, RObotics, and Artificial Vision Lab
University Institute for Computer Research
San Vicente, Spain
0000-0002-6175-600X

Ignacio de Loyola Páez-Ubieta
University Institute for Engineering Research
University Miguel Hernández
Elche, Spain
0000-0001-9901-7264

Edison Velasco-Sánchez
AUtomatics, RObotics, and Artificial Vision Lab
University Institute for Computer Research
San Vicente, Spain
0000-0003-2837-2001

2024 7th Iberian Robotics Conference (ROBOT) | 979-8-3503-7636-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ROBOT61475.2024.10796878

***Abstract*—Manipulation tasks in robotics usually involve two phases: an approach to the object and the grasp itself. The first action allows the robot to reach a certain pose in space that is likely to allow the object to be manipulated. Reinforcement Learning (RL) techniques allow a policy to be learned through experience given a set of states and actions, so this is a powerful tool for developing controllers for specific tasks such as positioning the robot in a particular point in space. However, when manipulating an object, orientation is as relevant as position. For this reason, a method of RL for positioning the robot's end effector in a suitable position and orientation for manipulation in simulation is presented. This approach models the problem of computing the distance for the reward function using dual quaternions parameterisation, an element that can represent the pose and attitude of a rigid body in Euclidean space in a compact way without having to apply any constraints.**

***Index Terms*—robotics, manipulation, reinforcement learning, dual quaternion**

I. Introduction

One of the main tasks when working with robotic manipulators is the handling of objects. This task involves different phases, as it is performed, namely approximation to the target, grasping by means of a tool or a robotic hand, and movement with the grasped object. Hence, one of the main goals of the Manipulation Task is to make it generalisable for a large amount of different objects and materials, so that the algorithms do not depend on the objects being manipulated.

In order to achieve this goal, automatic learning techniques have become a well-established solution for tackling this problem. [1] In particular, Reinforcement Learning (RL) produces solutions suitable for manipulation tasks, thus avoiding the arduous task of labelling samples, which is a necessary step in supervised models. RL allows an agent to interact with an environment with the objective of gathering experience in a specific task via reward modelling. Consequently, a controller for a high-level activity can be trained automatically. [2], [3]

In this work, a RL agent that performs manipulation tasks using only visual information from a simulated environment is proposed. Specifically, we aim to achieve two different goals:

- Reach task: it consists of reaching a specific pose with an orientation in space. This is achieved by using a dual quaternion formulation, which avoids additional constraints on reward modelling. Other representations, such as Euler angles with euclidean vectors need adaptations in order to depict the transformations between frames because they represent different magnitudes; translation and rotation. In addition, Euler angles present different issues when representing moving bodies in space, as in the case of Gimbal Lock. Dual quaternions allow this problem to be solved thanks to a compact representation.
- Manipulation task: implies the manipulation of the object. Using the RL agent trained in the reach task, the final pose is evaluated using classical control techniques. Once the robot has reached the episode's end, the gripper is closed until it detects contact and the object is moved into a home configuration. The manipulation is considered successful if it reaches home without falling.

The main contributions of this work are the proposal of a RL agent to perform manipulation tasks modelled as a reach-type one, using only visual information of the environment, as well as the simultaneous translation and orientation positioning through the use of a dual quaternion formulation for reward computation during training.

This paper is organised as follows: Section II reviews the state-of-the-art work on RL with robotic arms, Section III shows the methodology used in this paper, with RL and dual quaternion formulation, Section IV explains the setup used, its modelling and the experiments conducted, Section V shows the results obtained after performing the tests on the trained

The research work was supported by grant PID2021-122685OB-I00 funded by MICIU/AEI/10.13039/501100011033 and by ERDF/EU, as well as by grant Grand PRE2019-088069 funded by MICIU/AEI/10.13039/501100011033 and ESF Investing in your future.

Authorized licensed use limited to: UNIVERSIDAD DE ALICANTE . Downloaded on January 09,2025 at 12:01:15 UTC from IEEE Xplore. Restrictions apply.

agents, and Section VI concludes the paper with the final reflections together with the proposed future work.

II. Previous works

When applying RL to robotics, the reach task presents a challenge to the designed RL agents due to the complexity involved and the high dimensionality of the states, particularly when images are included. In [4], [5] agents are trained to reach an objective point in space by using only their Euclidean position. In [6], it is demonstrated that RL allows soft robots with complex models to learn reach actions using Artificial Neural Networks (ANN) as controllers. These works only employ the final position as the objective for the robot to reach, leaving out of the process the end effector orientation which in most applications can not be ignored.

Other works fulfil the aforementioned Reach Task along with the orientation one. In [7], Curriculum Learning (CL) [8] is used to progressively teach an agent how to perform it. In [9], [10] a framework is presented to achieve position and orientation in space with precision. However, they tend to impose restrictions on the movement, resulting in the loss of Degrees of Freedom (DOF). Alternatively, in [11], [12] the Reach Task has been developed for high-dimensional spaces involving six or more DOF. In these papers the orientation is included in the objective for the RL agent. Nevertheless, the employed Euler formulation needs to impose restrictions in order to avoid interpolation issues, along with adjustments for the position and orientation magnitudes because they present different units. The proposed approach with dual quaternions does not require these limitations.

For the Manipulation Task, RL agents are often considered as an end-to-end model that learns how to perform the task completely during training. In [13], [14], they use assistive techniques that help reducing the training time and transferring the controller to real environments. In contrast, [15], [16] use other techniques, such as the usage of human demonstrations from real world tasks, with the objective of learning more dexterous skills than a simple manipulation, such as spinning a pen or using a tool. End-to-end models tend to be hard to train as they require heavy resources. In this work, the agent trained in a simple Reach Task is used to conduct a Manipulation Task without any further training.

III. Methodology

In this section, basic and used RL techniques are introduced, as well as dual quaternion formulation to represent 6 DOF transformations and distance computing.

A. Reinforcement Learning Techniques

A RL approach is used in this paper to create a task-oriented agent that allows the robot to be controlled in order to perform Reach Tasks. A review of the basic elements of this type of algorithms and the main agent used in the experiments [17] [18], known as Soft Actor-Critic, are presented in the following lines.

1) Basic elements in RL: These methods are based on an agent that interacts with an environment in order to gather experience on how to develop certain tasks according to a reward function. These tasks consist of a sequence of actions generated according to a certain policy.

The RL approach models the environment as a Markov Decision Process (MDP) defined by the tuple $\mathcal{M} = [\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}]$, where $\mathcal{S}$ are the states the agent sees as it interacts with the environment, $\mathcal{A}$ are the actions the agent takes according to a state in $\mathcal{S}$ and its policy $\underline{\pi}$, $\mathcal{R}$ is the reward associated with taking an action in $\mathcal{A}$ given a state in $\mathcal{S}$, and $\mathcal{T}$ is the transition rule indicating how the environment changes according to $\mathcal{M} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ [15].

The goal of the agent is to maximise the cumulative reward or return at the end of an episode following (1).

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t \tag{1}$$

Where $R(\tau)$ is the return obtained following the sequence of steps τ, T is the number of steps employed to reach the end of an episode and γ is the discount factor for future rewards.

The Bellman equations are used to estimate the value of the states the agent visits and the actions it takes. Equation (2), also known as Value-State function, estimates the value of the transition from one state to another, while (3), also known as Action-State function, introduces the application of certain actions to the transition.

$$V(s_t) = \underset{s_{t+1}, a_t \sim \underline{\pi}}{\mathrm{E}} [r(s_t, a_t) + \gamma V(s_{t+1})] \tag{2}$$

Where $V(s_t)$ is the value of the state s_t and $r(s_t, a_t)$ is the reward function. The symbol E indicates that the values from the Bellman equations are estimations from the optimal functions.

$$Q(s_t, a_t) = \underset{s_{t+1}}{\mathrm{E}} \left[r(s_t, a_t) + \gamma \underset{a_t \sim \underline{\pi}}{\mathrm{E}} [Q(s_{t+1}, a_{t+1})] \right] \tag{3}$$

Where $Q(s_t, a_t)$ is the values of the tuple formed by the state s_t and the action a_t.

2) Soft Actor-Critic: Soft Actor-Critic (SAC) is a Deep Reinforcement Learning (DRL) algorithm belonging to the Actor-Critic family of methods, which aims to combine both Policy Gradient and Q-learning approaches.

The policy is optimised using a simultaneously optimised value function [19]. In this way, the Actor represents the policy that produces the actions that are evaluated by the Critic, which tries to estimate the Value function at the same time. Normally, SAC optimises two Critics at the same time. This is due to the tendency of the Actor-Critic methods to overestimate its actions, so the minimum of the two is used in the update functions.

This method is also off-policy, meaning that it uses a buffer $\mathcal{B}$ to store previous transitions. This way, every few steps, a tuple of $[s_t, a_t, r_t, s_{t+1}]$ is sampled from it so as to update the networks from both the Actor and Critic.

Authorized licensed use limited to: UNIVERSIDAD DE ALICANTE . Downloaded on January 09,2025 at 12:01:15 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: Soft Actor-Critic [17]

```
1  Input: initial policy parameters $\theta$, Q-function parameters $\phi_1, \phi_2$,
     empty replay buffer $\mathcal{B}$
2  Set target parameters equal to main parameters
     $\phi_{\text{targ},1} \leftarrow \phi_1,\ \phi_{\text{targ},2} \leftarrow \phi_2$
3  while not converge do
4      Observe state $s$ and select action $a \sim \underline{\pi}_\theta(\cdot|s)$
5      Execute a in the environment
6      Observe next state $s'$, reward $r$, and done signal $d$ to indicate
         whether $s'$ is terminal
7      Store $(s,a,r,s',d)$ in replay buffer $\mathcal{B}$ If $s'$ is terminal, reset
         environment state.
8      if it's is to update then
9          for j in range (however many update) do
10             Randomly sample a batch of transitions,
                 $b = (s,a,r,s',d)$ from $\mathcal{B}$
11             Compute targets for the Q-function:
12             // Equation (7)
13             Update Q-functions by one step of gradient descent
                 using
14
15             $\nabla_{\phi_i} \frac{1}{|B|} \sum_{\{s,a,r,s',d\} \in B} (Q_{\phi,i}(s,a) - y(r,s',d)))^2$
16               for  $i = 1,2$
17
18             Update policy by one step of gradient ascent using
19
20             $\nabla_\theta \frac{1}{|B|} \sum \left( \min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \text{log} \underline{\pi}_\theta(\tilde{a}_\theta(s)|s) \right)$,
21
22             where $a_\theta(s)$ is a sample from $\underline{\pi}_\theta(\cdot|s)$ which is
                 differentiable wrt $\theta$ via the reparametrization trick.
23
24             Update target networks with
25
26             $\phi_{targ,i} \leftarrow \rho\phi_{targ,i} + (1-\rho)\phi_i$
27         end
28     end
29 end
```

This paper uses the continuous version of the SAC method, where the Actor outputs two values $\mu_\theta(s_t)$ and $\sigma_\theta(s_t)$, which are used to parameterise a normal Gaussian distribution $\xi \sim \mathcal{N}(0, I)$ in order to apply gradient descent to the network. Using this initial distribution, SAC introduces entropy so as to mitigate the exploration-exploitation trade-off. Thus, the goal of the agent is not only to maximise the return, but also to maximise the entropy in the output, as shown in (4).

$$R_{\text{SAC}}(\tau) = \sum_{t=0}^{T} \underset{(s_t,a_t)\sim\underline{\pi}}{\text{E}} \left[\gamma^t r(s_t, a_t) + \alpha \text{H}(\underline{\pi}(\cdot|s_t)) \right] \quad (4)$$

Where $\text{H}(\underline{\pi}(\cdot|s_t))$ is the entropy calculated on the output distribution, which is regulated by an α parameter, which can be set as a trainable factor.

Both Actor and Critic are represented by a ANN, so it is necessary to define loss functions for both of them. In order to optimise the Actor network, (5) is used.

$$\begin{aligned} L(\theta) = \min_{i=1,2} Q_{\phi_i}(s_t, \hat{a}_\theta(s_t)) - \alpha \text{log}(\hat{a}_\theta(s_t)|s_t) \\ \hat{a}_\theta(s_t, \xi) = \tanh(\mu_\theta(s_t) + \sigma_\theta(s_t) \odot \xi) \end{aligned} \quad (5)$$

Where $\hat{a}_\theta$ represents the parameterised output distribution.

On the other hand, each of the two Critic ANN is updated according to (6) as its loss function, which represents the mean squared error between the current estimation and a target one, represented by a previous copy of the Critic estimating the values for the next states.

$$L(\phi_i, \mathcal{B}) = \text{E}\left[(Q_{\phi_i}(s_t, a_t) - y(r_t, s_{t+1}, d))^2 \right] \quad (6)$$

$$\begin{aligned} y(r_t, s_{t+1}, d) = r_t + \gamma(1-d)(\min_{i=1,2} Q_{\phi_{targ_i}}(s_{t+1}, \hat{a}_{t+1}) \\ - \alpha \text{log}(\underline{\pi}(a_{t+1}|s_{t+1}))) \end{aligned} \quad (7)$$

Where $Q_{\phi_{targ}}(s_{t+1}, \hat{a}_{t+1})$ is a target network for one of the critics.

The complete algorithm used to train SAC is shown in Algorithm 1. It shows the whole process of gaining experience and updating the weights of each network after a certain number of steps.

B. Simple Quaternions

Simple quaternions are an extension of complex numbers $\mathbf{q} \in \mathbb{H}$. They are expressed as $\mathbf{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$, with $q_0, q_1, q_2, q_3 \in \mathbb{R}$ and the imaginary units satisfy $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = -1$. Other representations are commonly used, such as the vectorial form $\mathbf{q} = (u, \tilde{\mathbf{v}}) = (q_0, q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k})$. Some basics operations are listed in Table I considering the quaternions in the vectorial convention.

Operation	Formulation
Real part	$\text{Re}(\mathbf{q}_1) = u_1$
Imaginary Part	$\text{Im}(\mathbf{q}_1) = \tilde{\mathbf{v}}_1$
Addition	$\mathbf{q}_1 + \mathbf{q}_2 = (u_1 + u_2, \tilde{\mathbf{v}}_1 + \tilde{\mathbf{v}}_2)$
Multiplication	$\mathbf{q}_1 \cdot \mathbf{q}_2 = (u_1u_2 - \mathbf{v_1}\mathbf{v_2}, u_1\mathbf{v_1} + u_2\mathbf{v_2} + \mathbf{v_1} \times \mathbf{v_2})$
Conjugate	$\mathbf{q}_1^* = (u_1, -\mathbf{v}_1)$
Module	$\lvert\lvert\mathbf{q}_1\rvert\rvert = \mathbf{q}_1 \cdot \mathbf{q}_1^*$

TABLE I: Basic operations with simple quaternions given a pair of them $\mathbf{q}_1$ and $\mathbf{q}_2$.

Simple quaternions are commonly used in order to represent rotations in space. To this end, it is necessary that the quaternion $\mathbf{q}$ fulfils the unitary property $||\mathbf{q}|| = 1$.

C. Dual Quaternions

Dual quaternions are an extension of dual numbers $\hat{\mathbf{q}} \in \mathcal{H}$, which allow to represent translations as well as rotations in space using a compact formulation defined as $\hat{\mathbf{q}} = \mathbf{q_r} + \epsilon\mathbf{q_d}$, with $\mathbf{q_r},\ \mathbf{q_d} \in \mathbb{H}$ on the form $\mathbf{q} = [q_0, q_1, q_2, q_3]$ [20] [21] [22], where $\mathbf{q_r}$ is the Primary part, $\mathbf{q_d}$ is the Dual part, ϵ represent the dual operator which denotes dual numbers and satisfies $\epsilon^2 = 0, \epsilon \neq 0$. Some basic operations with dual quaternions are listed in Table II.

They are often used to represent whole frames in Euclidean space, expressing both a translation and a rotation with fewer parameters than a homogeneous transformation matrix, for instance. Since dual quaternions are an extension of quaternions,

Authorized licensed use limited to: UNIVERSIDAD DE ALICANTE . Downloaded on January 09,2025 at 12:01:15 UTC from IEEE Xplore. Restrictions apply.

Operation	Formulation
Primary part	$\mathcal{P}(\hat{\mathbf{q}}_1) = \mathbf{q}_{\mathbf{r}1}$
Dual Part	$\mathcal{D}(\hat{\mathbf{q}}_1) = \mathbf{q}_{\mathbf{d}1}$
Addition	$\hat{\mathbf{q}}_1 + \hat{\mathbf{q}}_2 = \mathbf{q}_{\mathbf{r}1} + \mathbf{q}_{\mathbf{r}2} + \epsilon(\mathbf{q}_{\mathbf{d}1} + \mathbf{q}_{\mathbf{d}2})$
Multiplication	$\hat{\mathbf{q}}_1 \otimes \hat{\mathbf{q}}_2 = \mathbf{q}_{\mathbf{r}1} \cdot \mathbf{q}_{\mathbf{r}2} + \epsilon(\mathbf{q}_{\mathbf{r}1} \cdot \mathbf{q}_{\mathbf{d}2} + \mathbf{q}_{\mathbf{d}1} \cdot \mathbf{q}_{\mathbf{r}2})$
Conjugate	$\hat{\mathbf{q}}_1^* = \mathbf{q}_{\mathbf{r}1}^* + \epsilon \mathbf{q}_{\mathbf{d}1}^*$
Module	$\|\|\hat{\mathbf{q}}_1\|\| = \hat{\mathbf{q}}_1 \otimes \hat{\mathbf{q}}_1^*$
Difference	$\hat{\mathbf{q}}_{\text{diff}} = \hat{\mathbf{q}}_1^* \otimes \hat{\mathbf{q}}_2$

TABLE II: Basic operations with dual quaternions given a pair of them $\hat{\mathbf{q}}_1$ and $\hat{\mathbf{q}}_2$.

they are composed of two simple quaternions: one representing the translation and another to the rotation. Thus, a 6 DOF transformation is represented by a translation vector $\mathbf{t} \in \mathbb{R}^3$ as the Dual component, expressed as a simple quaternion with zero real part $\mathbf{q}_{\text{t}} = 0 + \text{t}_x\boldsymbol{i} + \text{t}_y\boldsymbol{j} + \text{t}_z\boldsymbol{k}$. The rotation quaternion constitutes the Dual part $\mathbf{q}_r$, having as a result a dual quaternion transformation according to (8).

$$\hat{\mathbf{q}} = \mathbf{q}_{\mathbf{r}} + \epsilon\left(\frac{1}{2}\mathbf{q}_{\mathbf{r}} \cdot \mathbf{q}_{\mathbf{t}}\right) \tag{8}$$

IV. Experimentation

In this section the hardware setup, as well as the environment modeling and the training method used to optimize the neural policies of the agent is presented.

A. Hardware setup

The robot used in this project is the UR5e, an object manipulation robot arm with 6 DOF. For this reason it is modelled using the Denavit-Hartenberg (DH) formulation for open kinematic chains [23]. This modelling allows the calculation of both direct and inverse kinematics of the robot. The former provides the position of the robot's end effector given the values of the manipulator's joints, while the latter obtains the joint references to bring the robot to a given pose in Euclidean space. These computations are performed using homogeneous transformation matrices that represent the frames for each joint [24].

On the other hand, the tool used in this project is the Robotiq 3f, an adaptive gripper with three fingers that allows to perform form closure gripping. It has an internal algorithm that allows it to adapt to the object being manipulated, although in this work it is used as a simple gripper that only closes and opens without considering other functionalities.

In terms of visual feedback, three Intel® RealSense™ D435i are placed in the simulated environment. The first is on the robot's wrist, the second is in front of the robot and the third remains on one side of the environment. In this way, all the planes of Euclidean space are covered.

Figure 1 shows the complete setup in a simulated environment, together with the images from the cameras of the environment. The robot is placed on the table to reach the placed objects.

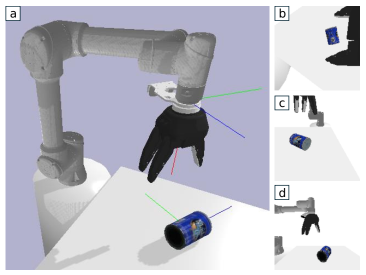

Fig. 1: Simulated environment. (a) Experimental environment of the UR5e robotic arm and the Robotiq 3f gripper. Camera views of the environment. (b) wrist camera, (c) front camera, (d) side camera.

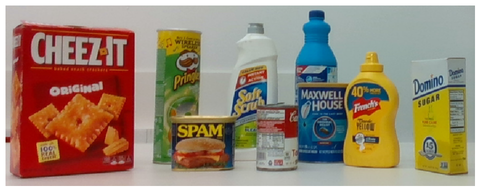

Fig. 2: Object subset from YCB dataset.

The objects used in this work are those from the YCB object dataset [25], which provide a wide variety of shapes and colours for visual recognition and grasping tasks. The subset used in this work is shown in Figure 2.

B. Environment modeling

The modelling of the environment is a crucial aspect of the design of a RL agent. It involves defining the states that the agent will use to compute its actions, which must also be specified. The reward function is also defined and tuned in order to guide the agent towards the desired behaviour.

As for the states, the position of the robot's end effector is used together with images obtained from the cameras. These consist of depth and greyscale images of the environment, both normalised in the range $[0, 1]$ and concatenated as a two-channel image, to speed up computation and save buffer space during training.

The actions are modelled as positional increments in the Euclidean space of the robot's tool. The values obtained by the policy are normalised between $[-1, 1]$ and then escalated and added to each direction of the current position of the end effector. Inverse kinematics is used to convert the Cartesian coordinates of the target into joint references.

The reward function is designed to reduce the distance between the robot's end effector and the object poses, both obtained from the simulation. For this reason, dual quaternions are used to represent both frames; the object $\hat{\mathbf{q}}_{obj} = \mathbf{q}_{\mathbf{r}obj} + \epsilon\mathbf{q}_{\mathbf{d}obj}$ and the wrist of the robot $\hat{\mathbf{q}}_w = \mathbf{q}_{\mathbf{r}w} + \epsilon\mathbf{q}_{\mathbf{d}w}$. At each time step, the difference between two positions $\hat{\mathbf{q}}_t$ and $\hat{\mathbf{q}}_{t-1}$ is computed using the operations given in Section III-C, resulting in the dual quaternion $\hat{\mathbf{q}}_{\text{diff}} = \mathbf{q}_{\mathbf{r}\text{diff}} + \epsilon\mathbf{q}_{\mathbf{d}\text{diff}}$. This way the reward function is formulated in 9.

Authorized licensed use limited to: UNIVERSIDAD DE ALICANTE . Downloaded on January 09,2025 at 12:01:15 UTC from IEEE Xplore. Restrictions apply.

$$r(t) = \begin{cases} \eta \tanh\left(\frac{1}{\lambda_r \theta_t + \lambda_d d_t}\right) & \theta_t < \theta_{t-1} \wedge d_t < d_{t-1} \\ -\eta \tanh\left(\frac{1}{\lambda_r \theta_t + \lambda_d d_t}\right) & \theta_t \geq \theta_{t-1} \vee d_t \geq d_{t-1} \end{cases} \tag{9}$$

Where $\theta_t = 2 \text{ atan}\left(\frac{||\text{Im}(\mathcal{P}(\hat{\mathbf{q}}_{\text{diff}}))||}{\text{Re}(\mathcal{P}(\hat{\mathbf{q}}_{\text{diff}}))}\right)$ is the angle represented by the rotational part of the dual quaternion and $d_t = ||\mathbf{q}^*_{\mathbf{r}\text{diff}} \cdot \mathbf{q}_{\mathbf{d}\text{diff}} \cdot \mathbf{q}_{\mathbf{r}\text{diff}}||_2$ is the module of the imaginary part of the difference, rotated to be in global coordinates. For all experiments $\eta = 3$, $\lambda_r = \pi^{-1}$ and $\lambda_d = 3.5$. The hiperbolic functions aim to limit the distance between $[-\eta, \eta]$.

The objective of this functions is to encourage the agent to take actions that reduce the distance between the end effector's and object's frame, $\hat{\mathbf{q}}_w$ and $\hat{\mathbf{q}}_{obj}$ respectively. If the robot moves the Primary and Dual part of its frame in the direction of the object, the reward will be positive. In any other case, it will be negative. In addition, the reward is inversely proportional to the distance from the object in the timestamp t. For instance, if the robot is far from the objective and takes an action that leads it farther, the agent will get a negative reward. That penalisation will be smaller than when taking the same action from a closer distance. The same occurs when taking positive actions; movements that reduce the distance to the object will produce greater rewards the closer the frames are.

C. Training method

The approach used to train the agent is similar to that of the CL. First, an agent is pre-trained with only one object in the environment, as this allows it to learn the task more quickly. Then the policy is trained again using the weights learned for a single object, but using a subset of the YCB dataset, including the former. All the training and experiments are conducted in simulation, concretely in PyBullet.

After a certain number of steps, the model is saved and 50 validation episodes are run in which the actions are not sampled. Instead, they are obtained deterministically directly from the mean of the output distribution.

V. Results and Discussion

The aim of these experiments is to show that using dual quaternions when computing the reward of a RL agent can lead to better policies than using Euler angular distances with the XYZ convention and without movement restrictions. Both tasks are considered for evaluation; reach and manipulation.

A. Reach task

The Reach Task is evaluated using the dual quaternion distances presented in 9, which serve as a measure of the accumulated reward during an episode. This dual quaternion distance has no measurement units, although it is normalised between 0 and 1. In the case of Euler angles, the norm between the position and rotation difference of the object and the robot's wrist is used to compute the distance.

As it can be seen in Figure 3, the agent trained with the dual quaternion reward obtains larger rewards during training than the one trained with linear distance and Euler angles. Dual quaternions allow the distance between states to be represented in a more compact way than with the linear option and unambiguously. This leads to better representation of rotations, as the agent is often correctly positioned but misoriented in the case of the Euler agent.

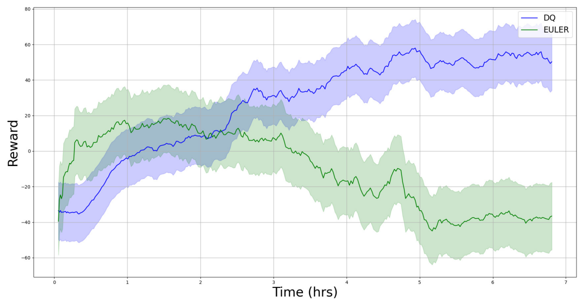

Fig. 3: Reward accumulated for the agents trained with reward functions formulated in Dual Quaternions (DQ) and Euler with linear distance (EULER). The results are obtained combining 3 complete training processes for each case.

	Linear distance (m)	Angle distance (rad)
Linear reward agent	0.71	1.45
Dual Quaternion reward agent	**0.11**	**0.80**

TABLE III: Error obtained between the robot's end effector and the object poses at the end of an episode.

Table III shows the error committed by the best agent trained with each reward function taking into account that the error is expressed as the distance between frames at the end of an episode. The dual quaternion option allows to reduce the linear distance between the poses, as well as the difference between the Euler angles. The results are obtained by averaging the error of 50 evaluation episodes.

B. Manipulation task

The Manipulation Task includes an additional step: once the robot's end effector has reached a final state in terms of pose, the gripper is closed until contact is detected, assuming it is in a suitable state. After this action, the robot moves the object to the initial position of the sequence. If the object is manipulated without falling, it is considered a success.

This grasping technique depends to a large extent on the position reached by the robot at the end of the episode. Therefore, the agent that can reach a more precise position will be able to manipulate the object better. For this reason, the one trained with the dual quaternion reward function achieves higher results in these manipulation tasks, as shown in Figure 4, where the agents try to grasp a random object from the YCB dataset during 50 episodes.

Authorized licensed use limited to: UNIVERSIDAD DE ALICANTE . Downloaded on January 09,2025 at 12:01:15 UTC from IEEE Xplore. Restrictions apply.

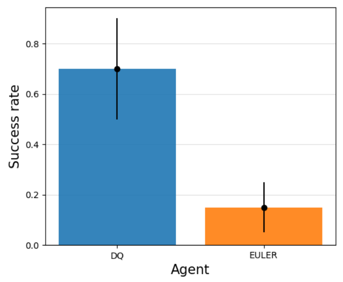

Fig. 4: Success rate of both agents in the Manipulation Task with DQ reward and EULER reward.

VI. Conclusion

In this work, an agent trained using RL techniques has been proposed to solve reach techniques that generate trajectories for the robotic arm. The reward has been designed using a dual quaternion parametrization in order to avoid defining excessive restrictions that would make the formulation hard to understand. The experiments show promising results when this representation is used against others, such as Euler. Moreover, dual quaternion representations allow to interpolate and express frames in space without worrying about singularities.

Furthermore, by tackling the Reach Task and then applying classical control techniques to manipulate the objects, we can significantly reduce the training time and the computational resources used for this process.

Nevertheless, this work has a lot of potential. Dual quaternions are used in many tasks other than reaching, such as pushing or completing manipulations, which can be explored following this research. In addition, this formulation is planned to be extended into real world environments so as to test the performance of the proposed method, as well to check the sim2real gap (if exists) on this scenario.

References

[1] S. Nahavandi, R. Alizadehsani, D. Nahavandi, C. P. Lim, K. Kelly, and F. Bello, "Machine learning meets advanced robotic manipulation," *Information Fusion*, pp. 1–27, 2024. doi: 10.1016/j.inffus.2023.102221

[2] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, "A survey on deep reinforcement learning algorithms for robotic manipulation," *Sensors*, pp. 1–35, 2023. doi: 10.3390/s23073762

[3] Y. Qin, B. Huang, Z.-H. Yin, H. Su, and X. Wang, "Dexpoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation," in *Proceedings of The 6th Conference on Robot Learning*. PMLR, 2023, pp. 594–605. [Online]. Available: https://proceedings.mlr.press/v205/qin23a.html

[4] T. Lindner, A. Milecki, and D. Wyrwał, "Positioning of the robotic arm using different reinforcement learning algorithms," *International Journal of Control, Automation and Systems*, pp. 1661–1676, 2021. doi: 10.1007/s12555-020-0069-6

[5] A. Lobbezoo and H.-J. Kwon, "Simulated and real robotic reach, grasp, and pick-and-place using combined reinforcement learning and traditional controls," *Robotics*, p. 12, 2023. doi: 10.3390/robotics12010012

[6] R. Morimoto, S. Nishikawa, R. Niiyama, and Y. Kuniyoshi, "Model-free reinforcement learning with ensemble for a soft continuum robot arm," in *2021 IEEE 4th International Conference on Soft Robotics (RoboSoft)*, 2021, pp. 141–148. doi: 10.1109/RoboSoft51838.2021.9479340

[7] S. Luo, H. Kasaei, and L. Schomaker, "Accelerating reinforcement learning for reaching using continuous curriculum learning," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8. doi: 10.1109/IJCNN48605.2020.9207427

[8] A. Bassich and D. Kudenko, "Continuous curriculum learning for reinforcement learning," in *Proceedings of the 2nd Scaling-Up Reinforcement Learning (SURL) Workshop. IJCAI*, 2019. doi: 10.48550/arXiv.2003.04960

[9] P. Shukla, H. Kumar, and G. C. Nandi, "Robotic grasp manipulation using evolutionary computing and deep reinforcement learning," *Intelligent Service Robotics*, pp. 61–77, 2021. doi: 10.1007/s11370-020-00342-7

[10] P. Aumjaud, D. McAuliffe, F. J. Rodríguez-Lera, and P. Cardiff, "Reinforcement learning experiments and benchmark for solving robotic reaching tasks," in *Advances in Physical Agents II*, vol. 1285. Springer International Publishing, 2021, pp. 318–331. doi: 10.1007/978-3-030-62579-5_22

[11] A. Iriondo, E. Lazkano, A. Ansuategi, A. Rivera, I. Lluvia, and C. Tubío, "Learning positioning policies for mobile manipulation operations with deep reinforcement learning," *International journal of machine learning and cybernetics*, pp. 3003–3023, 2023. doi: 10.1007/s13042-023-01815-8

[12] R. Srivastava, R. Lima, R. Sah, and K. Das, "Deep reinforcement learning based control of rotation floating space robots for proximity operations in pybullet," in *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2023, pp. 1224–1229. doi: 10.1109/SMC53992.2023.10394028

[13] D. Zhou, R. Jia, H. Yao, and M. Xie, "Robotic arm motion planning based on residual reinforcement learning," in *2021 13th International Conference on Computer and Automation Engineering (ICCAE)*, 2021, pp. 89–94. doi: 10.1109/ICCAE51876.2021.9426160

[14] A. A. Shahid, D. Piga, F. Braghin, and L. Roveda, "Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning," *Autonomous Robots*, pp. 483–498, 2022. doi: 10.1007/s10514-022-10034-z

[15] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," *Robotics: Science and Systems (RSS)*, pp. 1–9, 2018. doi: 10.15607/RSS.2018.XIV.049

[16] J. Pitz, L. Röstel, L. Sievers, and B. Bäuml, "Dextrous tactile in-hand manipulation using a modular reinforcement learning architecture," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 1852–1858. doi: 10.1109/ICRA48891.2023.1016075

[17] J. Achiam and P. Abbeel, "Openai: Spinning up in deep rl," 2018. [Online]. Available: https://spinningup.openai.com/en/latest/

[18] T. Simonini and O. Sanseviero, "The hugging face deep reinforcement learning class," 2023. [Online]. Available: https://github.com/huggingface/deep-rl-class

[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 1861–1870. doi: 10.48550/arXiv.1801.01290

[20] B. Kenwright, "A beginners guide to dual-quaternions: What they are, how they work, and how to use them for 3d character hierarchies," *20th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pp. 15–24, 2012. [Online]. Available: http://wscg.zcu.cz/WSCG2012/!_2012-Journal-Full-1.pdf

[21] F. Thomas, "Approaching dual quaternions from matrix algebra," *IEEE Transactions on Robotics*, pp. 1037–1048, 2014. doi: 10.1109/TRO.2014.2341312

[22] Y.-B. Jia, "Dual quaternions," *Iowa State University: Ames, IA, USA*, pp. 1–15, 2013. [Online]. Available: https://faculty.sites.iastate.edu/jia/files/inline-files/dual-quaternion.pdf

[23] P. I. Corke, "A simple and systematic approach to assigning denavit–hartenberg parameters," *IEEE Transactions on Robotics*, pp. 590–594, 2007. doi: 10.1109/TRO.2007.896765

[24] U. Robots, "Dh parameters for calculations of kinematics and dynamics," 2024. [Online]. Available: https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/

[25] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Yale-cmu-berkeley dataset for robotic manipulation research," *The International Journal of Robotics Research*, pp. 261–268, 2017. doi: 10.1177/0278364917700714

Authorized licensed use limited to: UNIVERSIDAD DE ALICANTE . Downloaded on January 09,2025 at 12:01:15 UTC from IEEE Xplore. Restrictions apply.