Motion Mappings for Continuous Bilateral Teleoperation

Xiao Gao[1,2], João Silvério[2], Emmanuel Pignat[2], Sylvain Calinon[2], Miao Li[1] and Xiaohui Xiao[1,3]

Abstract—Mapping operator motions to a robot is a key problem in teleoperation. Due to differences between workspaces, such as object locations, it is particularly challenging to derive smooth motion mappings that fulfill different goals (e.g. picking objects with different poses on the two sides or passing through key points). Indeed, most state-of-the-art methods rely on mode switches, leading to a discontinuous, low-transparency experience. In this paper, we propose a unified formulation for position, orientation and velocity mappings based on the poses of objects of interest in the operator and robot workspaces. We apply it in the context of bilateral teleoperation. Two possible implementations to achieve the proposed mappings are studied: an iterative approach based on locally-weighted translations and rotations, and a neural network approach. Evaluations are conducted both in simulation and using two torque-controlled Franka Emika Panda robots. Our results show that, despite longer training times, the neural network approach provides faster mapping evaluations and lower interaction forces for the operator, which are crucial for continuous, real-time teleoperation.

I. INTRODUCTION

Driven by the need to perform tasks in remote environments, teleoperation has emerged as a central robotics paradigm with applications ranging from deep sea [1] to outer space [2]. In this paper, we propose an object-centered motion mapping framework for continuous bilateral teleoperation. Our main objective is to address the problem of generating smooth, goal-directed robot trajectories when discrepancies exist between operator and remote robot workspaces. This problem, illustrated in Fig. 1 with a valve turning example, remains largely unaddressed among the state-of-the-art teleoperation frameworks.

The logical steps between naive joint space mappings and object-centered representations can be seen in Fig. 2. Direct joint space mappings (Fig. 2(a)) impose strong embodiment constraints, hence most teleoperation frameworks rely on task space representations as in Fig. 2(b). Despite that, even the most simple differences between local and remote environments can lead the remote robot to fail as in Fig. 2(c).
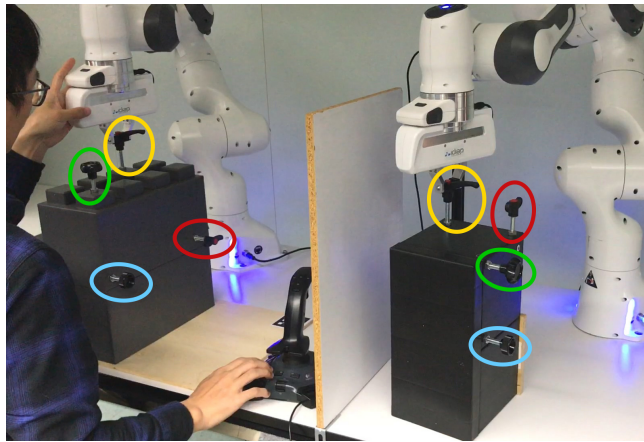
Fig. 1: Valve turning experiment. The task consists of an operator kinesthetically guiding the left (local) robot to rotate four valves (in any preferred order) and the right (remote) robot executing the same behavior on its side, despite different valve poses. We consider four different valves on each workspace, where each pair has different poses with respect to the robot base (see colored circles). A vertical board prevents the user from relying on visual feedback to complete the task, emulating the realistic teleoperation of a remote robot.

This favours object-centered approaches (Fig. 2(d)) that are better but still tend to scale poorly with the number of objects (Fig. 2(e)). A common approach to this issue is to suspend the communication between the haptic device and the remote robot, relocate the haptic device and continue the task based on visual feedback [3], [4] (Section II will provide an overview). Nonetheless, the discontinuous manipulation leads to low efficiency and transparency, especially in tasks that involve multiple objects. In this paper we propose an approach that locally reshapes the task space to ensure that the robot adapts to points of interest that can differ on the two sides (Fig. 2(f)). Particularly, our main contribution is an object-centered formulation for task space motion mapping between the local and remote robot, with:

1) adaptability to different object poses between local and robot workspaces (either real or virtual);
2) invertible and continuously differentiable position and orientation mappings;
3) bilateral teleoperation capabilities, by relying on impedance control both for haptic guidance and compliant interaction of the robot with the environment.

Our approach permits the operator to focus on its local setup (see Fig. 1), driving the input device to locally execute the task, with the remote robot performing it synchronously despite different object poses. We use a bilateral teleoperation setup with two Franka Emika Panda robots, where one acts as haptic device (on the operator side) and the other as a remote robot (operating on the targeted environment). We
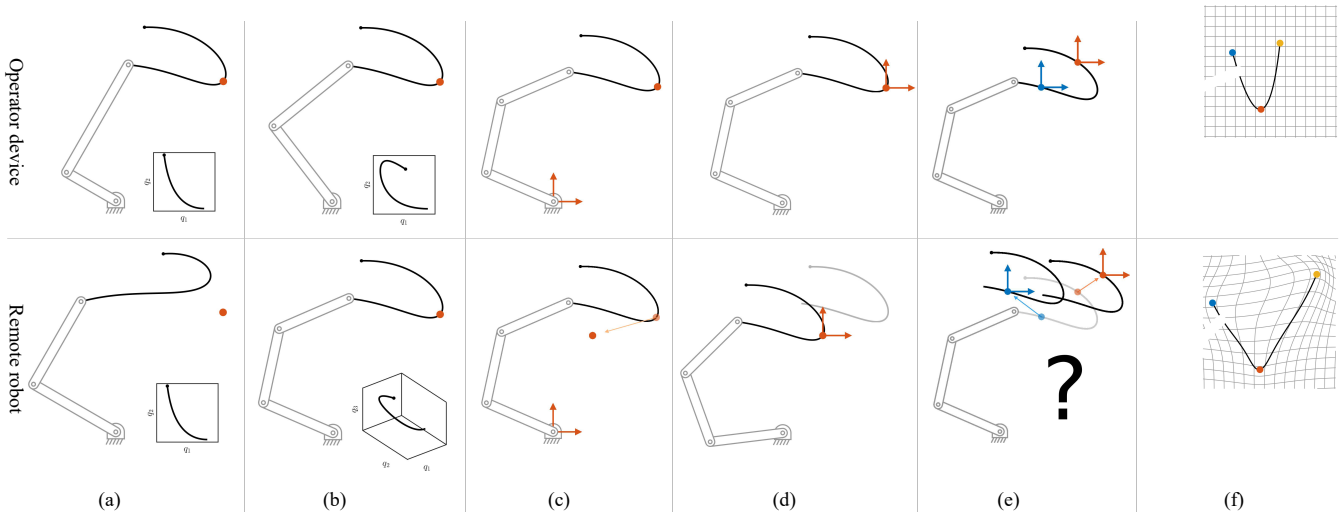
Fig. 2: **Top row:** Operator side (here referred to as *local environment*. **Bottom row:** Robot side (potentially a remote location). **(a)** Teleoperating a robot to pass through a point in space (orange) using a mapping in *joint space* does not allow to handle kinematic differences (e.g., different link lengths or number of joints) between the teleoperating device and the remote robot. **(b)** A direct mapping in *task space* can overcome kinematic differences. **(c)** However, it is not robust to differences in the environment (e.g. different object locations between workspaces). **(d)** A possible solution is to formulate the tracking problem with a coordinate system attached to the object instead of the robot base. **(e)** The question is: how to extend this strategy to multiple objects, while ensuring continuity, smoothness and precision? **(f)** Our proposed solution consists of finding a continuous and differentiable mapping (also called a *diffeomorphism*) that considers multiple objects (or other relevant landmarks in the workspace).

bring forward two possible implementations to achieve the proposed mappings: an iterative approach (Section IV) and a neural network (NN) approach (Section V). Experimental results (Section VII) show that the latter is more advantageous due to faster mapping evaluations.

II. RELATED WORK

As the operator and remote robot workspaces are often different, a workspace motion mapping is required to compute the trajectory of the remote robot from the operator signals. The most common solution is position mapping [3] by copying or scaling up/down the pose displacement [4]. However, it requires the user to suspend the teleoperation and relocate the haptic device due to physical constraints, which results in a heavy workload and long time cost. To avoid this problem, rate control [5] is used by setting the velocity command of the remote robot proportional to the pose displacement of the local robot. But with this approach, it is hard to get accurate positioning. Workspace drift control [3], [6] can automatically center the physical workspace of the device towards the region of interest of the user. In [7], [8], [9], two methods, rate control for coarse motion and a variable scaled position mapping for accurate positioning, are combined to realize efficient and accurate teleoperation. However, for sequential teleoperation tasks, these methods still require a manual switching between the mapping modes, hence the user cannot perform the teleoperation continuously. Moreover, they often heavily rely on visual feedback, which comes with additional limitations such as inferring 3D pose corrections from 2D video feedback or the need to stream video data, which can hinder applications where communication bandwidth is limited.

A relevant line of work relies on shared control approaches, which have been proposed, generally, as a means to reduce mental workload [10] and improve performance

[11]. Trajectory distributions are used as virtual fixtures in [12], which can constrain the robots to stay close to certain objects. Along these lines, Raiola *et al.* [13] propose probabilistic virtual fixtures to constrain the end-effector motion towards given objects or paths. However, this approach requires demonstrations (as in [12]), and it is not straightforward to generalize the virtual fixtures to arbitrary positions when the workspaces change. In the spirit of learning from demonstration, Zeestraten *et al.* [14] propose an object-centered strategy for shared control, where the remote robot corrects the orientation of the end-effector to align with an object. Havoutis *et al.* [15] exploit a similar concept while considering multiple coordinate systems associated with different objects to resolve differences between local and remote environment configurations. With respect to this line of solutions, our approach does not rely on demonstrations and can ensure that moving towards an object on one side is replicated on the other.

Our solution takes inspiration from [16] to formulate the problem as that of computing a diffeomorphic mapping between workspaces. We begin by directly extending the iterative method proposed in [16] to fulfill our goals (Section IV). But, as we show in Section VII, despite fast training times, the time for updating the desired robot trajectory can be prohibitively high for real-time teleoperation. We build on this solution, and propose an approach based on a neural network that considerably alleviates the time constraints (Section V). Additionally, we go beyond classical unilateral teleoperation and consider the more general case of bilateral teleoperation [17]. Indeed, we show that our approach is robust to haptic feedback originating from the remote side, which is leveraged to perceive how the teleoperated robot interacts with the environment. For the sake of stability, we rely on an impedance control implementation of bilateral

teleoperation [18], as opposed to using direct force feedback.

III. PROBLEM STATEMENT AND DEFINITIONS

In this section we propose a set of fundamental constraints that need to be fulfilled for robust bilateral teleoperation with different object poses. We then formalize them mathematically. Finally, orientation conversions and error definitions (which will be used in optimization) are introduced.

A. Constraints for bilateral teleoperation with different object poses

We consider an arbitrary number of objects N that are present in both workspaces following a one-to-one correspondence, as shown in Fig. 3. The positions and orientations of the objects are represented as $\boldsymbol{A} = \{\boldsymbol{x}_i, \boldsymbol{q}_i\}_{i=1}^N$ and $\boldsymbol{B} = \{\boldsymbol{x}_i', \boldsymbol{q}_i'\}_{i=1}^N$ respectively, where $\boldsymbol{x} \in \mathbb{R}^3$, $\boldsymbol{q} \in \mathcal{S}^3$. We consider cases where $N \geq 2$. The current poses of the left and right robot end-effectors are described as $(\boldsymbol{x}, \boldsymbol{q})$ and $(\boldsymbol{x}', \boldsymbol{q}')$ respectively. Our goal is to build a mapping $(\boldsymbol{x}, \boldsymbol{q}) \Leftrightarrow (\boldsymbol{x}', \boldsymbol{q}')$ between the two workspaces, where the following constraints apply:

1) When the left robot approaches the i-th object, with $\boldsymbol{x} \to \boldsymbol{x}_i$, the right robot should also approach the corresponding object, with $\boldsymbol{x}' \to \boldsymbol{x}_i'$.
2) Relative orientation should be the same when manipulating objects, which means that if $\boldsymbol{x} = \boldsymbol{x}_i$, $\Delta\boldsymbol{q}_i = \bar{\boldsymbol{q}}_i * \boldsymbol{q} = \bar{\boldsymbol{q}}_i' * \boldsymbol{q}'$ where $\Delta\boldsymbol{q}_i$ is the orientation of the robot end-effector with respect to the i-th object[1].
3) Position and orientation mappings should be invertible and continuously differentiable.
4) The gradient of the position mapping should be small enough so that the robot motion stays within safe velocity limits.

The first two constraints are aimed at ensuring object-directed motions on both sides with proper relative orientations. The goal of the third constraint is to guarantee smooth control commands. Firstly, the continuity of pose mappings ensures that a continuous trajectory of the local robot results in a continuous trajectory for the remote robot. Secondly, ensuring that pose mappings are continuously differentiable also guarantees that the resulting velocities are continuous. And thirdly, invertibility of the mappings is crucial in impedance-based bilateral teleoperation, since motions on the remote side should be uniquely mapped from and to the motion of the local side. Finally, the last constraint sets a limit on the magnitude of the position mapping gradient to limit velocities and ensure safe operation.

B. Mapping formulation

Note that the constraints for position and orientation are formulated differently, even though they share the same goal of reaching a given object with the same relative pose on both sides. This is due to the fact that, when object orientations are known, applying an orientation offset close to the object is enough to guarantee correct relative

---

[1]Here, $*$ denotes the product between two quaternions and $\bar{(\cdot)}$ denotes quaternion conjugation.
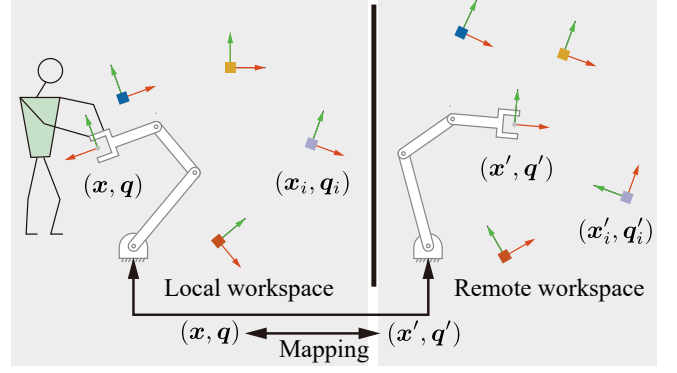

Fig. 3: A schematic of bilateral teleoperation. **Left:** the local space built for teleoperation. **Right:** the remote workspace. Colored squares with arrows represent the poses of 4 objects to be manipulated. Our approach builds a mapping between the end-effector poses of local and remote robots.

orientations. Consequently, we can treat the position and orientation mapping problems separately by focusing on the position first. We formulate the position mapping problem as finding an invertible and continuously differentiable function, or mapping, between $\boldsymbol{x}$ and $\boldsymbol{x}'$. The forward and backward mappings $\boldsymbol{\Phi}_x$, $\boldsymbol{\Phi}_x^{-1}$, are defined as

$$\boldsymbol{x}' = \boldsymbol{\Phi}_x(\boldsymbol{x}), \quad \boldsymbol{x} = \boldsymbol{\Phi}_x^{-1}(\boldsymbol{x}'). \tag{1}$$

Since, by constraint 2), the orientation offset depends on the current robot position, we propose to write the orientation mapping directly as a function of position, with

$$\boldsymbol{q}' = \boldsymbol{\Phi}_q(\boldsymbol{x}, \boldsymbol{q}) = \boldsymbol{g}(\boldsymbol{x}) * \boldsymbol{q}, \quad \boldsymbol{q} = \bar{\boldsymbol{g}}(\boldsymbol{x}) * \boldsymbol{q}'. \tag{2}$$

Here, $\boldsymbol{\Phi}_x$ and $\boldsymbol{\Phi}_q$ are invertible and continuously differentiable position and orientation mapping functions, and $\boldsymbol{g}(\boldsymbol{x}) \in \mathcal{S}^3$ is the difference between two end-effector orientations as a unit quaternion and as a function of position.

From the proposed mappings (1), (2), velocity mappings are obtained trivially by computing partial derivatives:

$$\dot{\boldsymbol{x}}' = \frac{\partial\boldsymbol{\Phi}_x(\boldsymbol{x})}{\partial\boldsymbol{x}}\dot{\boldsymbol{x}}, \quad \dot{\boldsymbol{x}} = \left(\frac{\partial\boldsymbol{\Phi}_x(\boldsymbol{x}')}{\partial\boldsymbol{x}'}\right)^{-1}\dot{\boldsymbol{x}}', \tag{3a}$$

$$\dot{\boldsymbol{q}}' = \frac{\partial\boldsymbol{g}(\boldsymbol{x})}{\partial\boldsymbol{x}}\dot{\boldsymbol{x}} * \boldsymbol{q} + \boldsymbol{g}(\boldsymbol{x}) * \dot{\boldsymbol{q}}, \tag{3b}$$

$$\dot{\boldsymbol{q}} = \frac{\partial\bar{\boldsymbol{g}}(\boldsymbol{x})}{\partial\boldsymbol{x}}\dot{\boldsymbol{x}} * \boldsymbol{q}' + \bar{\boldsymbol{g}}(\boldsymbol{x}) * \dot{\boldsymbol{q}}', \tag{3c}$$

with resulting angular velocity computed as $\boldsymbol{\omega} = 2\dot{\boldsymbol{q}}*\bar{\boldsymbol{q}}$. Note that, as suggested by constraint 4), the velocity mappings are scaled by the gradient of the position mapping.

Equations (1)–(3) lay out the foundation of our approach. As we shall see in the following sections, continuity and continuous differentiability of the proposed mappings come naturally from the techniques we choose to represent $\boldsymbol{\Phi}_x$ and $\boldsymbol{\Phi}_q$. Note that, for the orientation mapping in (2), there is a linear relationship between $\boldsymbol{q}'$ and $\boldsymbol{q}$. Thus invertibility is already fulfilled. We only need the function $\boldsymbol{g}(\boldsymbol{x})$ to be continuously differentiable. Invertibility of the position mapping, however, requires an additional formal constraint to be fulfilled. According to the inverse function theorem, a continuously differentiable function $\boldsymbol{\Phi}_x(\boldsymbol{x})$ is invertible if $\det\left(\frac{\partial\boldsymbol{\Phi}_x(\boldsymbol{x})}{\partial\boldsymbol{x}}\right) > 0, \forall\boldsymbol{x} \in \mathbb{R}^3$. With this in mind, we will

impose constraints on the used representations such that invertibility is ensured.

C. Orientation conversions

In our approach we rely on the unit quaternion representation of orientation due to its compactness and easy composition of orientations. Nonetheless, for some operations, we will need to represent orientations as a 3-dimensional Euclidean vector. For this, we here recall the definition of the logarithmic function $\log: \mathcal{S}^3 \mapsto \mathbb{R}^3$ to convert unit quaternions accordingly [19]:

$$\boldsymbol{r} = \log(\boldsymbol{q}) = \begin{cases} \arccos^*(q_v)\dfrac{\boldsymbol{q}_u}{\|\boldsymbol{q}_u\|}, & \text{if } \|\boldsymbol{q}_u\| \neq 0 \\ [0,0,0]^T, & \text{otherwise} \end{cases}, \quad (4)$$

where $\boldsymbol{q} = [q_v, \boldsymbol{q}_u^\top]^\top \in \mathcal{S}^3$, and $q_u \in \mathbb{R}$, $\boldsymbol{q}_v \in \mathbb{R}^3$ refer to the real and vector parts of the quaternion. arccos* is a modified version of the arc-cosine function, see [19] for details. When required during training and reproduction, unit quaternions are retrieved using the exponential function:

$$\exp(\boldsymbol{r}) = \begin{cases} \left[\cos(\|\boldsymbol{r}\|), \sin(\|\boldsymbol{r}\|)\dfrac{\boldsymbol{r}^\top}{\|\boldsymbol{r}\|}\right]^\top, & \text{if } \|\boldsymbol{r}\| \neq 0 \\ [1,0,0,0]^\top, & \text{otherwise.} \end{cases}$$
$$(5)$$

D. Error definitions

Based on constraint 1) and 2), the mapping errors should be small enough to fulfill task accuracy requirements. Position and orientation errors of the mappings are defined as

$$F_p(\Phi_x(\boldsymbol{A}), \boldsymbol{B}) = \frac{1}{N}\sum_{i=1}^{N}\|\Phi_x(\boldsymbol{x}_i) - \boldsymbol{x}_i'\|, \quad (6a)$$

$$F_q(\Phi_q(\boldsymbol{A}), \boldsymbol{B}) = \frac{1}{N}\sum_{i=1}^{N}d(\Phi_q(\boldsymbol{q}_i), \boldsymbol{q}_i'), \quad (6b)$$

$$\text{with} \quad d(\boldsymbol{q}_a, \boldsymbol{q}_b) = \|\log(\boldsymbol{q}_a * \bar{\boldsymbol{q}}_b)\| = \arccos(|\boldsymbol{q}_a^\top \boldsymbol{q}_b|), \quad (6c)$$

where the position distance function F_p is the Euclidean distance and orientation distance function F_q is a quaternion metric.

IV. MAPPING BY ITERATIVE METHOD

A promising way to build the mappings (1)–(2) is to use a diffeomorphism, which is an invertible and differentiable mapping. For this, we extend the diffeomorphic matching method in [16] by including orientations. In this section, we compute $\boldsymbol{\Phi}_x(\boldsymbol{x})$, $\boldsymbol{\Phi}_q(\boldsymbol{x})$ as a composition of K locally weighted translations and rotations $\phi_1(\boldsymbol{x}),\ldots,\phi_K(\boldsymbol{x})$ and $\varphi_1(\boldsymbol{x}),\ldots,\varphi_K(\boldsymbol{x})$, respectively. And we explain how to find their optimal parameters such that the properties defined in Section III are respected.

A. Locally-weighted translations and rotations

Following [16], we consider *locally weighted translations*

$$\phi(\boldsymbol{x}) = \boldsymbol{x} + k_\phi(\boldsymbol{x})\boldsymbol{v}_1, \quad (7a)$$

$$\text{with} \quad k_\phi(\boldsymbol{x}) = e^{-\rho_1^2\|\boldsymbol{x}-\boldsymbol{c}_1\|^2}, \quad (7b)$$

where $\boldsymbol{v}_1 \in \mathbb{R}^3$ is the direction of the translation and $k_\phi(\cdot)$ is a radial basis function (RBF) kernel with center $\boldsymbol{c}_1$ and width ρ_1. If $\det\left(\frac{\partial\phi(\boldsymbol{x})}{\partial\boldsymbol{x}}\right) > 0$, $\forall(\boldsymbol{x}, \boldsymbol{v}_1) \in \mathbb{R}^3 \times \mathbb{R}^3$, then $\phi(\boldsymbol{x})$ is a diffeomorphism. A sufficient condition is $0 < \rho_1 < \rho_{\max}(\boldsymbol{v}_1) = \frac{1}{\sqrt{2}\|\boldsymbol{v}_1\|}e^{\frac{1}{2}}$ (see [16] for the proof).

For the orientation mapping, *locally weighted rotation* is applied as

$$\varphi(\boldsymbol{q}) = \boldsymbol{v}_2^{k_\varphi(\boldsymbol{x})} * \boldsymbol{q}, \quad (8a)$$

$$\text{with} \quad k_\varphi(\boldsymbol{x}) = e^{-\rho_2^2\|\boldsymbol{x}-\boldsymbol{c}_2\|^2}. \quad (8b)$$

Here, $\boldsymbol{v}_2 \in \mathcal{S}^3$ is a unit quaternion representing a rotation direction. $\boldsymbol{v}_2^{k_\varphi(\boldsymbol{x})}$ refers to spherical linear interpolation, which we implement using the functions (4)–(5), between the identity quaternion and $\boldsymbol{v}_2$ with the weight $k_\varphi(\boldsymbol{x}) \in (0, 1]$. $k_\varphi(\cdot)$ is an RBF kernel with center $\boldsymbol{c}_2$ and width ρ_2. This is analogous to the term $k_\phi(\boldsymbol{x})\boldsymbol{v}_1$ in (7a), but in orientation.

B. Algorithm description

Following [16], we minimize the distances (6a)–(6b) using an iterative method, with a fixed number of K iterations. The algorithm we use is shown in Algorithm 1, which includes the extension to orientation. It employs three hyperparameters $\{\mu, \beta_1, \beta_2\}$ that regulate the degree of distortion in the resulting map. A complete description is beyond the scope of this paper, but the reader can refer to [16] for details. Note that β_2 is not defined in [16]; it has the same effect as β_1 but for orientation. The output $\boldsymbol{\rho}, \boldsymbol{C}, \boldsymbol{V}$ are the sets of parameters of the locally weighted translations/rotations in the final mappings Φ_x and Φ_q. To calculate these parameters, $\boldsymbol{Z} = \{\boldsymbol{Z}_x, \boldsymbol{Z}_q\}$ is initialized by $\boldsymbol{A}$ and the following steps are run at each iteration:

1) Find the index $m \in \{1,\ldots,N\}$ where $\boldsymbol{z}_i$ has the max position distance with $\boldsymbol{b}_i$, and store the translation center as $\boldsymbol{c}_{1,j}$ and direction as $\boldsymbol{v}_{1,j}$;
2) Solve a bound-constrained minimization problem of $F_p(\phi(\boldsymbol{Z}), \boldsymbol{B})$ under $\rho_1 \in (0, \mu\rho_{\max}(\boldsymbol{v}_1))$;
3) Update the position part $\boldsymbol{Z}_x$ by (7);
4) Find the index n where $\tilde{\boldsymbol{q}}_i$ and $\boldsymbol{q}_i'$ has the max orientation distance with (6c). Store the point as $\boldsymbol{c}_{2,j}$ for the center of the RBF in (8b);
5) Use the spherical linear interpolation to generate the quaternion offset $\boldsymbol{v}_{2,j}$;
6) Solve a bound-constrained minimization problem of $F_q(\varphi(\boldsymbol{Z}), \boldsymbol{B})$ under $\rho_2 > 0$;
7) Update orientation part $\boldsymbol{Z}_q$ by (8).

In the j-th iteration step, we have

$$\boldsymbol{Z}_x = \phi_j(\boldsymbol{Z}_x), \quad (9a)$$

$$\boldsymbol{Z}_q = \varphi_j(\boldsymbol{Z}) = \boldsymbol{v}_{2,j}^{k_{\varphi,j}(\boldsymbol{Z}_x)} * \boldsymbol{Z}_q. \quad (9b)$$

with a final mapping given by

$$\boldsymbol{\Phi}_x(\boldsymbol{x}) = \phi_K \circ \cdots \circ \phi_2 \circ \phi_1(\boldsymbol{x}), \tag{10a}$$

$$\boldsymbol{\Phi}_q(\boldsymbol{x}, \boldsymbol{q}) = \boldsymbol{\varphi}_K \circ \cdots \circ \boldsymbol{\varphi}_2 \circ \boldsymbol{\varphi}_1 = \boldsymbol{g}(\boldsymbol{x}) * \boldsymbol{q}, \tag{10b}$$

where $\circ$ refers to function composition, and the associative law of quaternions is adopted to simplify $\boldsymbol{\Phi}_q$. So the forward mapping has the same form as (1)–(2). Backward mapping can be computed with Newton's method. According to the continuity of composite functions and chain rule, the pose mappings (10) are continuously differentiable.

Algorithm 1: Position and orientation mapping

Input: $\boldsymbol{A} = \{\boldsymbol{x}_i, \boldsymbol{q}_i\}_{i=1}^N$,
$\qquad \boldsymbol{B} = \{\boldsymbol{x}_i', \boldsymbol{q}_i'\}_{i=1}^N$,
Parameters: $K \in \mathbb{N}^+, 0 < \mu < 1, 0 < \beta_1, \beta_2 \le 1$,
Output: $\boldsymbol{\rho} = \{\rho_{1,j}, \rho_{2,j}\}_{j=1}^N$,
$\qquad \boldsymbol{C} = \{\boldsymbol{c}_{1,j}, \boldsymbol{c}_{2,j}\}_{j=1}^N$,
$\qquad \boldsymbol{V} = \{\boldsymbol{v}_{1,j}, \boldsymbol{v}_{2,j}\}_{j=1}^N$,

1 **Initialize** $\boldsymbol{Z} = \{\boldsymbol{Z}_x, \boldsymbol{Z}_q\} = \{\boldsymbol{z}_i, \tilde{\boldsymbol{q}}_i\}_{i=1}^N, \boldsymbol{Z} := \boldsymbol{A}$,
$\quad$ **for** $j = 1$ to K **do**
2 $\quad\quad$ $m := \underset{i \in [1,\dots,N]}{\arg\max}(\|\boldsymbol{z}_i - \boldsymbol{x}_i'\|)$
3 $\quad\quad$ $\boldsymbol{c}_{1,j} := \boldsymbol{z}_m$
4 $\quad\quad$ $\boldsymbol{v}_{1,j} := \beta_1(\boldsymbol{x}_m' - \boldsymbol{c}_{1,j})$
5 $\quad\quad$ $\rho_{1,j} := \underset{\rho_1 \in (0, \mu\rho_{\max}(\boldsymbol{v}_{1,j}))}{\arg\min}(F_p(\boldsymbol{\phi}(\boldsymbol{Z}_x), \boldsymbol{B}))$
6 $\quad\quad$ $\boldsymbol{Z}_x := \phi_j(\boldsymbol{Z}_x)$
7 $\quad\quad$ $n := \underset{i \in [1,\dots,N]}{\arg\max}(d(\tilde{\boldsymbol{q}}_i, \boldsymbol{q}_i'))$
8 $\quad\quad$ $\boldsymbol{c}_{2,j} := \boldsymbol{z}_n$
9 $\quad\quad$ $\boldsymbol{v}_{2,j} := (\tilde{\boldsymbol{q}}_n * \bar{\boldsymbol{q}}_n')^{\beta_2}$
10 $\quad\quad$ $\rho_{2,j} := \underset{\rho_2 > 0}{\arg\min}(F_q(\boldsymbol{\varphi}(\boldsymbol{Z}), \boldsymbol{B}))$
11 $\quad\quad$ $\boldsymbol{Z}_q := \varphi_j(\boldsymbol{Z}_q)$
12 **end**

V. NEURAL-NETWORK MAPPING

In this section, a NN-based approach for finding $\boldsymbol{\Phi}_x(\boldsymbol{x})$ and $\boldsymbol{\Phi}_q(\boldsymbol{x})$ is proposed. Particularly, we leverage the use of normalizing flows [20], another popular approach for building invertible mappings. We adopt two NNs for position and orientation mappings: a real-valued non-volume preserving (Real NVP) model [21] and a fully connected NN, respectively. Then the two NNs are combined together for the motion mapping.

A. Position mapping

We adopt the idea of normalizing flows, which use a composition of bijective transformations to build a mapping between an *observed space* and a *latent space*. Here we will consider these as the local and remote workspaces in the teleoperation setup and apply the mapping with K transformations as (10a).

Following the structure of Real NVP [21], several coupling layers are built and each layer is a simple bijection as an affine transformation. Given a 3-dimensional input $\boldsymbol{x}$ and

$d = 1$ or $d = 2$, the output $\boldsymbol{x}'$ of an affine coupling layer follows

$$\begin{cases} \boldsymbol{x}_{1:d}' = \boldsymbol{x}_{1:d}, \\ \boldsymbol{x}_{d+1:3}' = \boldsymbol{x}_{d+1:3} \odot \exp(s(\boldsymbol{x}_{1:d})) + t(\boldsymbol{x}_{1:d}), \end{cases} \tag{11}$$

where s and t are convolutional networks for scale and translation. $\odot$ refers to element-wise product. Backward propagation is given by

$$\begin{cases} \boldsymbol{x}_{1:d} = \boldsymbol{x}_{1:d}', \\ \boldsymbol{x}_{d+1:3} = (\boldsymbol{x}_{d+1:3}' - t(\boldsymbol{x}_{1:d})) \odot \exp(-s(\boldsymbol{x}_{1:d})). \end{cases} \tag{12}$$

The Jacobian of the forward transformation (11) is given by

$$\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \Phi_x(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \boldsymbol{I}_d & \boldsymbol{0} \\ \frac{\partial \boldsymbol{x}_{d+1:3}'}{\partial \boldsymbol{x}_{1:d}} & \text{diag}(\exp(s(\boldsymbol{x}_{1:d}))) \end{bmatrix}, \tag{13}$$

which is triangular and the diagonal elements are positive ($\boldsymbol{I}_d$ is an identity matrix). Thus $\det\left(\frac{\partial \Phi_x(\boldsymbol{x})}{\partial \boldsymbol{x}}\right) > 0, \forall \boldsymbol{x} \in \mathbb{R}^3$ and the mapping is invertible, as discussed in Section III. Similarly to the proof of the iterative method, we know the NN-based mapping is also continuously differentiable.

The cost function to train the network is designed as

$$Q_1 = F_p(\Phi_x(\boldsymbol{A}), \boldsymbol{B}) + \lambda_1 \frac{1}{N_1} \sum_{i=1}^{N_1} |\boldsymbol{J}(\boldsymbol{x}_i^s) - \alpha\boldsymbol{I}_3|, \tag{14}$$

where the second term is a penalty for keeping the scaling factor of the mapping to fulfill the velocity constraint 4), by making the Jacobian matrix $\boldsymbol{J}(\boldsymbol{x}_i^s)$ approach the identity matrix $\boldsymbol{I}_3$ with a scaling variable $\alpha > 0$. Additionally, $\boldsymbol{x}_i^s \sim \mathcal{N}(\boldsymbol{\mu}_a, 3\boldsymbol{\Sigma}_a)$ are N_1 end-effector position samples on the local side, where $\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a$ are computed from the distribution of positions in $\boldsymbol{A}$. The scalar λ_1 is the weight of the penalty term in the cost.

B. Orientation mapping

A fully connected NN is designed to approximate $\boldsymbol{g}(\boldsymbol{x})$ in (2). It is trained in Euclidean space by the logarithmic function of quaternions as (4). We use a four-layer NN with 24 units per layer and *tanh* as the activation function to build the mapping $\boldsymbol{x} \mapsto \boldsymbol{r}$, where $\boldsymbol{r} = \log(\boldsymbol{q}' * \bar{\boldsymbol{q}})$ and $\boldsymbol{g}(\boldsymbol{x}) = \exp(\boldsymbol{r})$. By the constraint 2), the observed data used for training are $\{\boldsymbol{x}_i \mapsto \log(\boldsymbol{q}_i' * \bar{\boldsymbol{q}}_i)\}_{i=1}^N$.

The cost function of the orientation mapping is written as

$$Q_2 = \frac{1}{N} \sum_{i=1}^N d(\boldsymbol{g}(\boldsymbol{x}_i), \boldsymbol{q}_i' * \bar{\boldsymbol{q}}_i) + \lambda_2 \frac{1}{N_1} \sum_{j=1}^{N_1} F_q(\hat{g}(\boldsymbol{x}_j^s), \boldsymbol{q}_I)$$
$$+ \lambda_3 \sum_n \|\boldsymbol{w}_n\|^2, \tag{15}$$

where $\boldsymbol{r}_i$ is the output of the NN when $\boldsymbol{x} = \boldsymbol{x}_i$, and the first term is the orientation distance. The second term is used to keep the orientation offset close to the identity quaternion. Finally, the third term is the L2 regularization term to reduce overfitting, and $\boldsymbol{w}_n$ are the weights of the NN. The scalars λ_2, λ_3 weigh the contribution of the last two terms. Since we use fully connected layers and *tanh* for activation function, $\boldsymbol{g}(\boldsymbol{x})$ is a continuously differentiable function.

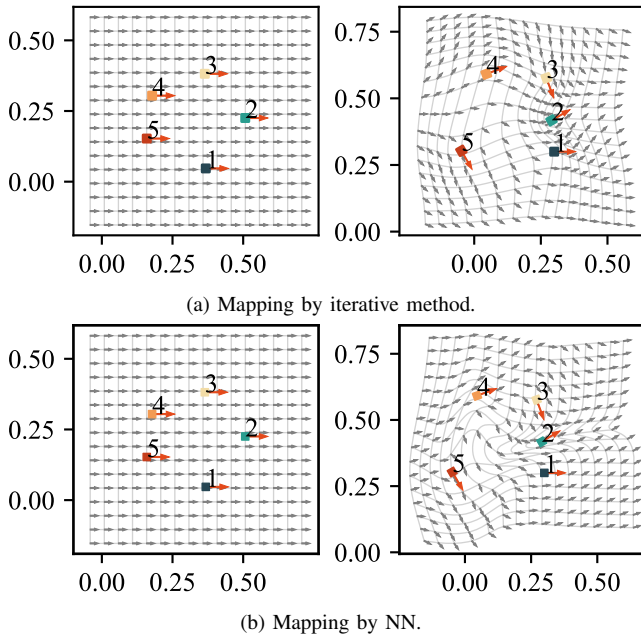(a) Mapping by iterative method.

(b) Mapping by NN.

Fig. 4: Position and orientation mappings with the two methods. Colored squares represent the objects with corresponding numbers, and the arrows show the object orientations. For better visualization, on the **left side**, orientations of colored points are set as identity quaternions. Grid points are equally spaced, also with identity quaternions. On the **right side**, we see new poses of the colored objects which are illustrative of a robot workspace that differs from the local workspace. Positions and orientations of grid points are smoothly distorted by the learned mappings.

C. Combined network for pose mapping

The two proposed NNs are combined together and trained using the Adam optimizer. The combined cost function is

$$Q = Q_1 + \lambda_4 Q_2. \qquad (16)$$

The observed data, sample input-output pairs are set as $\{x_i \mapsto x_i', \log(q_i' * \bar{q}_i)\}_{i=1}^N$ for training the combined NN. $\lambda = \{\lambda_i\}_{i=1}^4$ are the weight factors chosen empirically. The velocity mappings are computed trivially based on the partial derivatives of the NN and (3).

VI. TOY EXAMPLE IN 2D

In this section, we compare the approaches from Sections IV–V in a 2D scenario with 5 objects (Fig. 4). For the iterative method, parameters are set as $K = 100$, $\mu = 0.3$, $\beta_1 = 0.5$, $\beta_2 = 1$. For the NN method, $\lambda = [0.02, 3, 0.1, 9], \alpha = 1$, $N_1 = 200$ and we used 5×10^4 training loops.

The results of estimating diffeomorphic mappings on a 2D example are shown in Fig. 4. Grid points are plotted to visualize how the mappings distort the robot workspace. To get a clear visualization of orientation mapping, orientations of local workspace are simplified as identity quaternions (depicted as horizontal arrows). As only the orientation differences between the corresponding objects of two workspaces matter in (2), this simplification is appropriate. Arrows are plotted to represent the x-axis direction. We observe that the grid is reshaped smoothly by the proposed mappings. For the orientation mapping, the red arrows have almost the same directions as the nearby grey arrows, indicating that

TABLE I: Comparison of simulation results.

	Iterative method	Mapping by NN
Training time (s)	**0.268**	4.68×10^3
Forward estimation (ms)	6.71	**1.83**
Backward estimation (ms)	20.4	**1.93**
Velocity estimation (ms)	43.4	**1.79**
Position error (mm)	$\mathbf{8 \times 10^{-15}}$	0.1
Orientation error (rad)	$\mathbf{3.8 \times 10^{-6}}$	1.6×10^{-2}

the computed mapping is correct and smooth around the points of interest. Far from the colored points, the orientation approaches the identity quaternion, and the neighboring region is close to a squared shape, which means that the Jacobian of position mapping is close to the identity matrix and it is a local mapping. These observations apply to both methods (Figs 4a, 4b).

Table I shows a comparison of the time cost and estimation error. The iterative method takes a much shorter time for training than NN, but a longer time for forward and backward estimation. Because of using Newton's method to compute the backward mapping, it is more time consuming than the forward mapping, which is also true for the velocity mapping. These preliminary results in 2D suggest that the time constraints can affect the performance of real-time robot control. On the other hand, the NN method shows a very efficient ability of estimation. For accuracy, the iterative method provides almost zero position and orientation errors, while the NN-based method shows larger errors. Depending on the tasks, higher accuracy can be achieved with more training loops. All tests were run in a system of a 2.8Ghz Intel Core i7-7700HQ CPU (4 cores), 8 GB memory.

VII. ROBOT EXPERIMENTS

Valve-turning experiments were carried out to validate the two mapping methods in a realistic scenario, as shown in Fig. 1. Two torque-controlled Panda robots were used. The left robot is kinesthetically guided by the user, while the right one executes the task in its surrounding environment by tracking reference poses generated by the proposed mappings. Four valves are attached to a block in both local and remote sides. The valve poses with respect to the robot bases differ between the two setups. A joystick is used to command the closing and opening of the grippers. A camera is mounted on the table to locate the blocks, and consequently the valves, by using two Aruco makers (the user does not use these video streams for teleoperation). The transformations between the valves and the markers, as well as between the camera and the robots, are calibrated in advance.

A. Implementation

Cartesian impedance control [18] is used on both robots, so that the user can guide the left robot and obtain haptic feedback from the interaction between the right robot and the environment. The control law for both robots is $\tau = M(\theta)\ddot{\theta} + g(\theta) + c(\theta, \dot{\theta}) + J(\theta)^\top f$, with $f = \begin{bmatrix} K_p(x_d - x) + K_v(\dot{x}_d - \dot{x}) \\ K_{pr} \log(q_d * \bar{q}) + K_{vr}(w_d - w) \end{bmatrix}$, where $\theta \in \mathbb{R}^7$
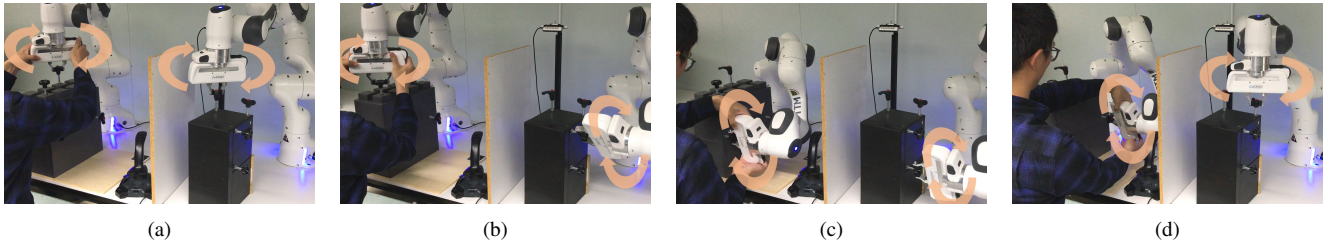
(a)　　　　　　　(b)　　　　　　　(c)　　　　　　　(d)

Fig. 5: Snapshots of the valve turning experiment. *(a)-(d)* show different valve poses on both workspaces and how the operator turns them without visual feedback. The accompanying video, also available at `http://sites.google.com/view/bilateralmaps`, further shows this aspect.

denote joint positions, $M(\boldsymbol{\theta}) \in \mathbb{R}^{7\times 7}$ is the mass matrix, $\boldsymbol{c}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \in \mathbb{R}^7$ is the Coriolis and centrifugal torque $\boldsymbol{J}(\boldsymbol{\theta}) \in \mathbb{R}^{6\times 7}$ is the Jacobian matrix, and $\boldsymbol{\tau}$, $\boldsymbol{g}(\boldsymbol{\theta}) \in \mathbb{R}^7$ are respectively control and gravitational joint torques. $\boldsymbol{f} \in \mathbb{R}^6$ is the desired Cartesian force. $\boldsymbol{x}, \dot{\boldsymbol{x}} \in \mathbb{R}^3$ are the Cartesian position and velocity. $\boldsymbol{q} \in \mathcal{S}^3$ is the unit quaternion and $\boldsymbol{w} \in \mathbb{R}^6$ the angular velocity of the end-effector. $(\cdot)_d$ denotes the desired value, which is computed by the mappings in (1)–(3). $\boldsymbol{K}_{(\cdot)}$ represents stiffness and damping gains (for position, orientation and velocity) of the Cartesian impedance controller. The parameters of $\boldsymbol{k}$ can be different for the two robots. We introduce a scalar scaling factor to decrease the gains on the left robot such that interaction forces are lower and the robot is easier to guide by the operator. In this case, $\boldsymbol{K}_p = 600\boldsymbol{I}_3$, $\boldsymbol{K}_v = 20\boldsymbol{I}_3$, $\boldsymbol{K}_{pr} = 30\boldsymbol{I}_3$, $\boldsymbol{K}_{vr} = 2\boldsymbol{I}_3$. The scaling-down factor was set as 0.1. The motor torque commands are sent at 1 kHz, and the reference trajectories are updated at 100 Hz. Parameters of the two methods are the same as Section VI.

B. Experimental results

The goal was to rotate four valves by $90°$ in sequence. Figure 5 shows snapshots of a reproduction. On the left side, valves were turned by human guidance one by one, and the right robot performed the same task synchronously. The two robot 3D trajectories in one trial are plotted in Fig. 6. Similarly to the 2D case, cube grids and their respective distorted version are used to show the task mapping. Although the valves on the two sides had different positions and orientations, both methods could deal with the differences, as the obtained mappings are smooth and generate trajectories that can adapt to the valve poses on both sides.

A traditional method of direct position mapping [3] was used here as baseline to evaluate our two methods. The vertical board and valves of the left side were removed for the direct position mapping, so the user can directly watch the robot workspace on the right, which emulates a typical approach where video streams are used. Ten trials were held for each method. Table II shows the results of the three methods. The iterative method takes almost the same task duration as the NN method to turn all the valves. However, it shows higher interaction forces applied by the operator. One plausible explanation is that the forward and backward estimation of the iterative method takes longer to compute than the NN method (see Table I), so the reference trajectories cannot be updated at a high enough rate. Both of them show better performance than the direct position

(a) Mapping by iterative method
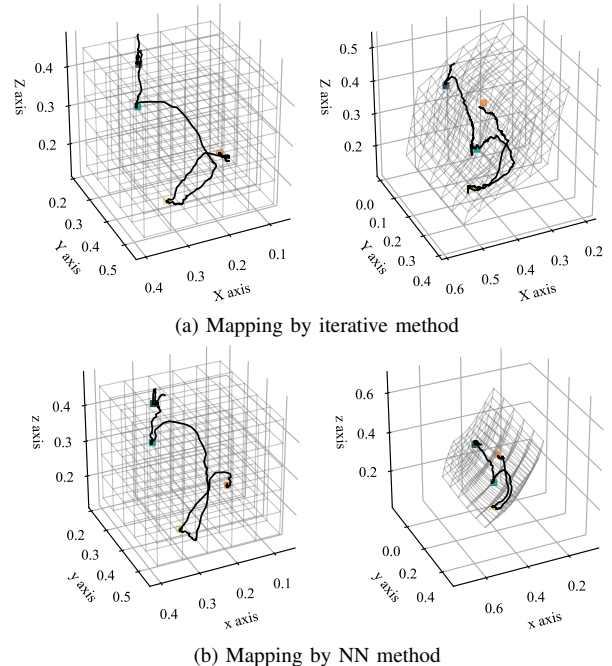

(b) Mapping by NN method

Fig. 6: 3D workspace mapping with the two methods. The left plots show the trajectories of the left robot end-effector, and the right plots show the remote robot. Colored squares represent the position of the valves. Grid points are equally spaced on the left side. The distorted grid lines on the right side represent the corresponding mapped workspace.

TABLE II: Comparison of experimental results.

	Iterative method	NN method	Direct pos. mapping
Task duration (s)	101 ± 18	$\mathbf{96 \pm 16}$	195 ± 26.5
Interaction force of the left robot (N)	1.12 ± 0.21	$\mathbf{0.81 \pm 0.14}$	1.05 ± 0.15

method.

The NN method shows excellent ability for efficient forward and backward estimation, as shown by the low estimation times in Table I, which results in a better transparency experienced by the operator. One limitation is that the training time can be substantially higher than the iterative method for a totally new scenario. A potential way to reduce this limitation is to refine the NN models from pre-trained models. For the iterative method, learning is very fast, but the backward estimation and velocity mapping take a longer time, bringing a sensation of higher stiffness while guiding the robot.

VIII. DISCUSSION

In this section, we discuss the limitations and additional considerations for the proposed approaches. In the valve

turning experiment, only the grasping poses are given as inputs to the mapping methods but one can potentially add further constraints. For example, viapoints can be added between two objects to bring additional geometric features or virtual guide priors, such as following a straight line between objects. If an object cannot be simplified as a single point for manipulation, more feature points can be added to describe objects. We have observed in practice that the number of points does not have a strong effect on the performance of the mapping methods.

While the approach targets teleoperation in highly unstructured environments, it should be noted that, when feasible, it is good practice to design the local and remote workspaces to be as similar as possible. Prior knowledge about the remote workspace, such as the proximity between objects, should be exploited in order to keep the Jacobian of Φ_x close to the scaled identity, which will reduce both the distortions of the workspace and the involved velocity commands.

Two impedance controllers were adopted here for bilateral teleoperation, providing indirect force feedback. Alternatively, other control methods can be used to implement bilateral teleoperation, such as direct force feedback. This should be done by taking into account that for some controllers, maintaining the stability of the closed-loop system in contact tasks might be difficult.

Finally, due to the higher training times, the neural network approach might be prohibitively slow for cases in which the objects of interest move, which would require the network to be re-trained on-the-fly. We plan to further investigate this research challenge in future work. We also plan to extend the approach to the context of virtual fixtures.

IX. Conclusion

We proposed a motion mapping approach for continuous bilateral teleoperation. The novel idea of using diffeomorphic mappings for this purpose is introduced, with two possible implementations being compared, one based on an extension of the iterative approach from [16] and an original one based on neural networks. We show that both can realize continuous bilateral teleoperation and generate an object-centered mapping, which is used to handle the differences between local and remote environment configurations. A 2D simulation has first been presented to visualize both position and orientation mappings. We showed that the NN-based solution provides faster estimation (below 2ms) but needs a long time for training, while the iterative method can be trained in under 0.3s. We then validated both solutions in a valve-turning experiment using two torque-controlled Franka Emika Panda robots, that we contrasted to a direct position mapping between the two robots baseline [3]. Our methods show better performance than direct position mapping. The NN-based implementation shows lower interaction forces on the local side due to faster mapping function evaluations.

References

[1] R. R. Murphy, K. L. Dreger, S. Newsome, J. Rodocker, E. Steimle, T. Kimura, K. Makabe, F. Matsuno, S. Tadokoro, and K. Kon, "Use of remotely operated marine vehicles at Minamisanriku and Rikuzentakata Japan for disaster recovery," in *IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 19–25, November 2011.

[2] N. Y. Lii, Z. Chen, B. Pleintinger, C. H. Borst, G. Hirzinger, and A. Schiele, "Toward understanding the effects of visual- and force-feedback on robotic hand grasping performance for space teleoperation," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pp. 3745–3752, October 2010.

[3] F. Conti and O. Khatib, "Spanning large workspaces using small haptic devices," in *First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics Conference*, pp. 183–188, IEEE, 2005.

[4] H. Kobayashi and H. Nakamura, "A scaled teleoperation," in *Proceedings IEEE International Workshop on Robot and Human Communication (RO-MAN)*, pp. 269–274, 1992.

[5] S. E. Salcudean, M. Zhu, W.-H. Zhu, and K. Hashtrudi-Zaad, "Transparent bilateral teleoperation under position and rate control," *The International Journal of Robotics Research*, vol. 19, no. 12, pp. 1185–1202, 2000.

[6] Y. Lin, H. Zhao, Y. Zhang, and H. Ding, "A Data-Driven Motion Mapping Method for Space Teleoperation of Kinematically Dissimilar Master/Slave Robots," *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 862–867, 2018.

[7] L. Liu, G. Liu, Y. Zhang, and D. Wang, "A modified motion mapping method for haptic device based space teleoperation," in *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, pp. 449–453, 2014.

[8] L. Guanyang, G. Xuda, L. Lingzhi, and W. Yan, "Haptic based teleoperation with master-slave motion mapping and haptic rendering for space exploration," *Chinese Journal of Aeronautics*, vol. 32, no. 3, pp. 723–736, 2019.

[9] M. Mamdouh, A. A. Ramadan, and A. A. Abo-Ismail, "Evaluation of a proposed workspace spanning technique for small haptic device based manipulator teleoperation," in *Intelligent Robotics and Applications*, pp. 161–170, Springer Berlin Heidelberg, 2012.

[10] H. Boessenkool, D. A. Abbink, C. J. M. Heemskerk, F. C. T. van der Helm, and J. G. W. Wildenbeest, "A task-specific analysis of the benefit of haptic shared control during telemanipulation," *IEEE Transactions on Haptics*, vol. 6, no. 1, pp. 2–12, 2013.

[11] J. J. Abbott, P. Marayong, and A. M. Okamura, "Haptic virtual fixtures for robot-assisted manipulation," in *Robotics Research*, pp. 49–64, Springer Berlin Heidelberg, 2007.

[12] M. Ewerton, G. Maeda, D. Koert, Z. Kolev, M. Takahashi, and J. Peters, "Reinforcement learning of trajectory distributions: Applications in assisted teleoperation and motion planning," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pp. 4294–4300, November 2019.

[13] G. Raiola, X. Lamy, and F. Stulp, "Co-manipulation with multiple probabilistic virtual guides," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pp. 7–13, Sep. 2015.

[14] M. J. A. Zeestraten, I. Havoutis, and S. Calinon, "Programming by demonstration for shared control with an application in teleoperation," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, pp. 1848–1855, July 2018.

[15] I. Havoutis and S. Calinon, "Learning from demonstration for semi-autonomous teleoperation," *Autonomous Robots*, vol. 43, no. 3, pp. 713–726, 2019.

[16] N. Perrin and P. Schlehuber-Caissier, "Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems," *Systems and Control Letters*, vol. 96, pp. 51–59, 2016.

[17] P. F. Hokayem and M. W. Spong, "Bilateral teleoperation: An historical survey," *Automatica*, vol. 42, no. 12, pp. 2035–2057, 2006.

[18] H. C. Cho and J. H. Park, "Stable bilateral teleoperation under a time delay using a robust impedance control," *Mechatronics*, vol. 15, no. 5, pp. 611–625, 2005.

[19] M. J. A. Zeestraten, I. Havoutis, J. Silvério, S. Calinon, and D. G. Caldwell, "An approach for imitation learning on Riemannian manifolds," *IEEE Robotics and Automation Letters (RA-L)*, vol. 2, no. 3, pp. 1240–1247, 2017.

[20] I. Kobyzev, S. Prince, and M. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

[21] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," *arXiv*, vol. abs/1605.08803, 2016.