IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 12 November 2024, accepted 5 December 2024, date of publication 31 December 2024, date of current version 7 January 2025.

Digital Object Identifier 10.1109/ACCESS.2024.3524946

RESEARCH ARTICLE

Safely Imitating Predictive Control Policies for Real-Time Human-Aware Manipulator Motion Planning: A Dataset Aggregation Approach

AIGERIM NURBAYEVA, (Member, IEEE), AND MATTEO RUBAGOTTI, (Senior Member, IEEE)
Department of Robotics, School of Engineering and Digital Sciences, Nazarbayev University, 010000 Astana, Kazakhstan

Corresponding author: Matteo Rubagotti (matteo.rubagotti@nu.edu.kz)

This research was funded by Nazarbayev University under Faculty Development Competitive Research Grants program for 2024-2026, grant no. 201223FD8808 (PI: Matteo Rubagotti).

ABSTRACT This paper proposes a dataset-aggregation approach for imitating a nonlinear model predictive control law via deep neural networks, to safely allow a robot manipulator to share its workspace with a human operator. As the robot approaches the human, its speed is gradually reduced using the ''speed and separation monitoring'' framework. Specific time-varying upper bounds are explicitly imposed on the control input generated by the deep neural network through a ''safety filter'' based on real-time numerical optimization. The proposed method is experimentally tested on a UR5 manipulator, comparing the performance of different neural network structures and types of training. As a result, it is shown that the dataset-aggregation approach provides better performance with respect to a ''naive'' approach to training, and that the presence of the safety filter is indeed needed to avoid the violation of the speed-and-separation-monitoring constraints.

INDEX TERMS Industrial robotics, physical human-robot interaction, model predictive control, neural networks.

I. INTRODUCTION

Workspace sharing in physical human-robot interaction (pHRI) is an important area of research in manufacturing, with the potential to revolutionize the industry. In fact, some works already refer to the ''Industry 5.0'' framework, focusing on further developing cooperation between humans and robots by implementing advanced technologies to expand human capabilities [1]. By combining the strengths of humans and robots, manufacturers can increase efficiency, flexibility, safety and profitability [2].

A fundamental aspect of pHRI applications is to ensure safety, i.e., to prevent accidents that could injure humans. One of the most widely used methods to ensure safety is *speed and separation monitoring* (SSM), which helps prevent collisions and ensure safe collaboration in pHRI tasks [3], [4], [5], [6] according to the ISO/TS 15066 safety standard. In SSM, the maximum allowable speed of the robot is calculated according to the distance between the robot and the human; as this distance decreases, the robot automatically slows down or stops, depending on the situation. Usually, SSM is imposed on a fixed robot path, but in some cases the robot motion is redefined in real time, with the main aim of increasing productivity while still ensuring safety (see, e.g., [7], [8], [9], [10], [11], [12], [13]).

The periodic recalculation of updated robot trajectories – for example, calculated via model predictive control (MPC) in [9], [11], and [12] solving a numerical optimization problem at each sampling instant – allows the robot motion to immediately adapt to the current human location while satisfying safety constraints, thus potentially improving productivity compared to using a fixed-path approach. This is shown, for instance, in [11], in which an MPC-based control scheme achieved superior performance as compared to a fixed-path SSM implementation, when experimentally tested on a Kinova Gen3 robot. However, using MPC requires carrying out computations that necessarily cause delays in the implementation of the trajectory; this leads to performance degradation proportional to the computation time. This problem of MPC implementations in systems with fast dynamics has been studied beyond the field of robotics, and a possible solution is to approximate the MPC law via

The associate editor coordinating the review of this manuscript and approving it for publication was Aysegul Ucar.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

neural networks (see, e.g., the seminal paper by Parisini and Zoppoli [14]), using the approach known as ''imitation learning'' [15].

To cite some examples, in [16], the use of long short-term memory (LSTM) deep neural networks (DNNs) was proposed for approximating the MPC control variable. The authors of [17] proposed using multi-stage nonlinear MPC (NMPC) to generate data pairs that were then used to learn a robust NMPC policy via DNNs. The authors of [18] trained a DNN to imitate a hybrid MPC controller for domestic heating systems, while imitation learning of MPC laws via DNNs was applied to resonant power converters in [19]. In [20], a learning-based multistage MPC framework was proposed for systems with time-varying plant-model mismatch, and Gaussian processes were employed to learn plant-model mismatches online. Approaches in which the agent is implemented via a DNN are often referred to as *deep imitation learning* (see, e.g., [21], [22], [23]).

This paper proposes an NMPC deep imitation learning framework for human-robot workspace sharing based on the concept of *dataset aggregation*. This concept was formalized in [24] by introducing the so-called *DAgger* algorithm. In a ''naive'' approach to imitation learning (such as that applied in [25] for the same type of problem), one would record observation-action pairs by running the policy to be imitated (namely, the *expert*), and then provide these pairs to the imitation learning algorithm to train the DNN. However, this might lead the trained policy, when being tested, to acquire observations that are quite different from those acquired when training the expert policy, thus resulting in poor performance. Instead, DAgger iteratively trains a policy starting with observation-action pairs generated by the expert, but this dataset is continuously increased (hence the expression ''dataset aggregation'') by adding new pairs in which the observation is generated by running the trained policy and the action is generated by the expert for each of these observations. Although this approach solves the problem of the naive approach, it has the drawback of requiring the expert to provide its actions, which is difficult when imitating human agents (e.g., drivers for an autonomous driving task). However, in our case the expert consists of an NMPC law that can be easily obtained for a given value of robot and human positions, which constitute the observation in our problem.

An NMPC algorithm can easily be formulated to impose constraints on the evolution of the robot joint positions. When the NMPC law is approximated using a DNN, however, it might happen that these constraints (including the guarantee of safety via SSM) are violated. This can occur especially if the human operator moves differently from what is observed in the training set.

A. CONTRIBUTIONS

The first contribution of this paper is the application of a dataset-aggregation approach to train different DNNs to imitate the SSM-based NMPC policy for robot motion planning developed in [11], rather than using a naive approach as was done in [25].

The second contribution is the introduction of a simple safety filter, i.e., a method to enforce the satisfaction of the SSM constraints when running the DNNs. This is achieved by solving a numerical optimization problem online after obtaining the DNN output. The optimization problem aims to determine a control action that is as close as possible to the DNN output, but which also satisfies the SSM constraints.

The third and last contribution is the design and experimental verification of the proposed strategy on a case study. The URsim industrial simulator, together with the human motion dataset introduced in [26], was used to simulate a UR5 manipulator and train the policy implemented via DNN for multiple human subjects. The resulting performance was experimentally verified on an actual UR5 manipulator, simulating different types of human motion from [26].

B. PAPER ORGANIZATION

The paper is organized as follows. The main concepts of the NMPC-based real-time motion planning scheme are presented in Section II, together with the problem formulation. Section III describes the details of the above-mentioned imitation learning methods, specifically regarding the DAgger-based training and the design of safety filters. Case study, neural network analysis and related experimental results are discussed in Sections IV, V and VI, respectively. Conclusions and future work are finally outlined in Section VII.

II. PROBLEM FORMULATION

In the following, a summary of the NMPC law that implements the expert policy in our work is presented. A few variations were made with respect to the approach of [11] – for example, by imposing limits only on the robot joint speeds and not on the joint positions as well – but the overall NMPC strategy can be identified with that proposed in [11]. After presenting the expert policy, a general formulation of our imitation learning problem is provided.

A. SUMMARY OF THE EXPERT POLICY

1) ROBOT AND HUMAN MODELING

The NMPC strategy proposed in [11] aims to steer the manipulator configuration, represented by the joint positions $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$, to a *goal configuration* $\boldsymbol{\theta}_f$. NMPC does not generate joint torques, but rather determines references for the joint speeds. In the employed simplified system dynamics, these references are assumed to coincide with the actual joint speeds, and are referred to as $\boldsymbol{\omega} \in \mathbb{R}^{n_\theta}$.

Using a fixed Cartesian reference $O - xyz$, the portions of space occupied by human operator and robot manipulator are over-approximated by a union of spheres. Each sphere has a constant radius and is centered at a given *test point*,

which is fixed at a specific location (e.g., at the left elbow of the human), and moves together with it.

The human body is over-approximated by n_h spheres, each of which has center $\boldsymbol{p}_{h,j} \in \mathbb{R}^3$ and radius $R_{h,j}$, given $j \in \mathbb{N}_h \triangleq \mathbb{N}_{[1,n_h]}$ (in general, in this paper the sequence of integer numbers from $n_1 \in \mathbb{N}$ to $n_2 \in \mathbb{N}$ included is expressed as $\mathbb{N}_{[n_1,n_2]}$). For the sake of compactness, the information on the location of all human test points is given by $\mathcal{H} \triangleq \{\boldsymbol{p}_{h,j}\}$, $i \in \mathbb{N}_h$.

On the other hand, the space occupancy of the manipulator is represented by n_r spheres, with centers $\boldsymbol{p}_{r,i} \in \mathbb{R}^3$ and radii $R_{r,i}$, given $i \in \mathbb{N}_r \triangleq \mathbb{N}_{[1,n_r]}$. In order to formulate the NMPC problem, one has to provide the locations of the robot test points $\boldsymbol{p}_{r,i}$ and of their velocity vectors $\mathbf{v}_i \in \mathbb{R}^3$ in the Cartesian space as functions of the joint variables $\boldsymbol{\theta}$ and $\boldsymbol{\omega}$. This can be done via forward and differential kinematics, respectively.

2) CONSTRAINTS OF THE NMPC PROBLEM

The first constraint imposed in the NMPC problem consists of limiting the joint speeds. This constraint can be expressed as $\boldsymbol{\omega} \in \Omega$, where $\Omega \subseteq \mathbb{R}^{n_\theta}$ is a hyper-rectangle that includes the origin in its interior.

Specific constraints can be defined to avoid fixed obstacles, such as the table on which the manipulator is placed. We summarize all these constraints on the location of the robot test points as $\boldsymbol{p}_{r,i} \in \mathcal{P}_i$, for $i \in \mathbb{N}_{n_r}$, where $\mathcal{P}_i \subseteq \mathbb{R}^3$ is a suitably-defined set.

The SSM constraints are set following the formulation of [3], to ensure that the human can never be in contact with a moving manipulator. To achieve this, the robot speed is decreased based on its distance with the human to ensure that, if the human moves with the maximum possible speed $\bar{v}_h \in \mathbb{R}_{>0}$ towards the robot, the robot can stop before a contact occurs, given the maximum deceleration $\bar{a}_i \in \mathbb{R}_{>0}$ achievable for each robot test point, the precision $\epsilon_s \in \mathbb{R}_{>0}$ with which the human test points can be detected by the employed motion capture system, and the time of detection and reaction T_{dr} of the robot to the human motion (i.e., the time interval needed to acquire the human location and calculate an appropriate control action). After defining $v_i \triangleq \|\mathbf{v}_i\|$, $i \in \mathbb{N}_r$ (where $\|\cdot\|$ is the Euclidean norm) and $d_{ij} \triangleq \|\boldsymbol{p}_{r,i} - \boldsymbol{p}_{h,j}\|$, $i,j \in \mathbb{N}_r \times \mathbb{N}_h$, one can give a mathematical formulation to the SSM constraints, as follows:

$$\bar{v}_h \left(T_{dr} + \frac{v_i}{\bar{a}_i} \right) + v_i T_{dr} + \frac{v_i^2}{2\bar{a}_i} + \epsilon_s \leq d_{ih}, \ i \in \mathbb{N}_r, \tag{1}$$

in which $d_{ih} \triangleq \min_{j \in \mathbb{N}_h} \{ d_{ij} - (R_{r,i} + R_{h,j}) \}$. The reader is referred to [3] for the explanation of the meaning of each term in (1). As explained in [11], the exact imposition of (1) leads to numerical issues in the NMPC problem, and therefore we impose a more conservative condition, which can be written as $v_i^2 \leq \zeta_i$ for all $i \in \mathbb{N}_r$, with

$$\zeta_i \triangleq \min_{j \in \mathbb{N}_h} \alpha_i^2 \left(d_{ij}^2 - \left(R_{r,i} + R_{h,j} + \bar{d}_i \right)^2 \right), \tag{2}$$

where $\alpha_i \in \mathbb{R}_{>0}$ and $\bar{d}_i \in \mathbb{R}_{>0}$, $i \in \mathbb{N}_r$, are tuning parameters, defined to ensure that $v_i^2 \leq \zeta_i$ implies (1) for all $i \in \mathbb{N}_{n_r}$.

3) LONG-TERM NMPC LAW

As explained in [11], two NMPC controllers are implemented to build the overall NMPC scheme. The *long-term NMPC* (L-NMPC) determines a plan for the robot motion, and passes it as references to the *short-term NMPC* (S-NMPC), which generates the joint speed references for the joint controllers with a shorter sampling time.

After introducing the L-NMPC sampling time T_ℓ and the discrete-time index $t \in \mathbb{N}$, the L-NMPC algorithm utilizes the simplified dynamical model

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + T_\ell \boldsymbol{\omega}_t \tag{3}$$

for predicting the time evolution of $\boldsymbol{\theta}_t$ along a time interval equal to N sampling intervals, with N referred to as *prediction horizon*. In this time interval, we refer to a generic sequence of joint positions, joint speeds and provided human positions as $\boldsymbol{\omega}_s \triangleq \{\boldsymbol{\omega}_0, \boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_{N-1}\}$, $\boldsymbol{\theta}_s \triangleq \{\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N\}$ and $\mathcal{H}_s \triangleq \{\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_N\}$, respectively, where $t = 0$ is the current time instant. When a previously-recorded human motion can be used – for example when training the DNNs developed in this paper – then the actual value of $\mathcal{H}_s$ can be provided to the NMPC controller. In a real-time application, if a reliable prediction of the human motion is not available, then one can generate $\mathcal{H}_s$ by assuming that the human will remain at the currently measured position during the whole prediction horizon, as implemented in [11].

The task of the L-NMPC routine is to find the optimal sequences $\boldsymbol{\omega}_s^* \triangleq \{\boldsymbol{\omega}_0^*, \boldsymbol{\omega}_1^*, \ldots, \boldsymbol{\omega}_{N-1}^*\}$ and $\boldsymbol{\theta}_s^* \triangleq \{\boldsymbol{\theta}_0^*, \boldsymbol{\theta}_1^*, \ldots, \boldsymbol{\theta}_N^*\}$ minimizing the cost function $J(\boldsymbol{\omega}_s, \boldsymbol{\theta}_s, \mathcal{H}_s) \triangleq \sum_{t=0}^{N-1} \ell_t$, with

$$\ell_t \triangleq (\boldsymbol{\theta}_t - \boldsymbol{\theta}_f)^T \boldsymbol{Q} (\boldsymbol{\theta}_t - \boldsymbol{\theta}_f) + \boldsymbol{\omega}_t^T \boldsymbol{R} \boldsymbol{\omega}_t + \gamma \varphi^2(\boldsymbol{\theta}_t, \mathcal{H}_t) \tag{4}$$

where $\boldsymbol{Q} \in \mathbb{R}^{n_\theta \times n_\theta}$ and, $\boldsymbol{R} \in \mathbb{R}^{n_\theta \times n_\theta}$ are symmetric positive definite matrices and $\gamma \in \mathbb{R}_{>0}$. Minimizing the cost function leads to reducing the distance $\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_f\|$ to the goal configuration, while moderating the robot speed $\|\boldsymbol{\omega}_t\|$, and maintaining the robot end-effector relatively far from the human through a suitable function $\varphi(\boldsymbol{\theta}_t, \mathcal{H}_t)$, for the detailed explanation of which we refer the reader to [11, Sec. IV.A].

The above-mentioned optimal sequences are determined by solving, at each sampling time, the following optimal control problem, where the initial state $\boldsymbol{\theta}_0$ coincides with the currently measured vector of joint positions:

$$\begin{aligned}
&(\boldsymbol{\omega}_s^*, \ \boldsymbol{\theta}_s^*) \\
&\quad = \arg\min_{\boldsymbol{\omega}_s, \boldsymbol{\theta}_s} \ J(\boldsymbol{\omega}_s, \boldsymbol{\theta}_s, \mathcal{H}_s) && \text{(5a)} \\
&\qquad \text{s.t. } \boldsymbol{\theta}_N = \boldsymbol{\theta}_f && \text{(5b)} \\
&\qquad\quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + T_\ell \boldsymbol{\omega}_t, \ t \in \mathbb{N}_{[0,N-1]} && \text{(5c)} \\
&\qquad\quad \boldsymbol{\omega}_t \in \Omega, \ t \in \mathbb{N}_{[0,N-1]} && \text{(5d)} \\
&\qquad\quad \boldsymbol{p}_{r,i} \in \mathcal{P}_i, \ i \in \mathbb{N}_r, \ t \in \mathbb{N}_{[0,N-1]}, && \text{(5e)} \\
&\qquad\quad v_i^2 \leq \zeta_i, \ i \in \mathbb{N}_r, \ t \in \mathbb{N}_{[0,N-1]}. && \text{(5f)}
\end{aligned}$$

Only the first computed control action is applied to the robot, and a new optimal control problem is solved after one sampling time, when a new measurement of the positions of robot and human are available. This approach is known as *receding horizon*, as the time horizon is shifted forward as new information is obtained about the system. This allows the MPC law to adapt to changing conditions and disturbances in real time, and to redefine the control actions accordingly.

4) SHORT-TERM NMPC LAW

In order to generate a prediction of the robot motion that can reach the goal configuration $\boldsymbol{\theta}_s$, while limiting the value of N, a relatively large value of T_ℓ is typically used. In order to increase the frequency with which the robot motion is re-planned, and to reduce the detection and reaction time T_{dr} used to formulate the SSM constraints (1), the optimal sequence $\boldsymbol{\theta}_s^*$ is passed to the above-mentioned S-NMPC as a reference to be tracked. S-NMPC solves an optimal control problem similar to (5), typically using the same prediction horizon N, but with a smaller sampling time $T_\sigma \ll T_\ell$. More specifically, the first term in ℓ_t (see, (4)) is modified by substituting the element of $\boldsymbol{\theta}_s^*$ to be tracked instead of $\boldsymbol{\theta}_f$; (5b) is removed; (5c) is redefined as $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + T_\sigma \boldsymbol{\omega}_t$ for $t \in \mathbb{N}_{[0,N-1]}$; finally, (5d)-(5f) are maintained unchanged. Again using the receding-horizon approach, the first element of the sequence of joint speeds generated by S-NMPC is passed to the low-level joint controllers of the robot.

B. THE DEEP IMITATION LEARNING PROBLEM

The first element of the sequence of joint speeds generated via S-NMPC constitutes, from an imitation learning standpoint, the action generated by the expert at time t (with sampling time T_σ) corresponding to a measured robot state $\boldsymbol{\theta}_t$ and human position $\mathcal{H}_t$. This action is referred to in the following as $a^{NMPC}(s)$, where

$$s \triangleq \{\boldsymbol{\theta}, \mathcal{H}\} \tag{6}$$

is the *state* of the imitation learning problem. The problem considered in our work is the same of [25]: to generate, using the dataset aggregation approach, a deterministic policy $a^{DNN}(s)$ via DNN to imitate $a^{NMPC}(s)$, with $a^{DNN}(s)$ satisfying the SSM constraints.

Strictly speaking, due to the structure of the employed hierarchical NMPC scheme, the first element of the sequence of joint speeds from S-NMPC depends not only on the current value of s, but also on the reference value from L-NMPC. However, learning this second dependence would not benefit the resulting realization of $a^{DNN}(s)$, as in an ideal case the NMPC law should only be based on s. Therefore, we consider the variation of the NMPC law due to the L-NMPC reference (for a given s) as noise.

III. DNN TRAINING AND SAFETY FILTER

As already mentioned in Section II-A3, the NMPC routine used for training the DNN can rely on pre-recorded human position data, and can therefore plan the robot trajectory assuming to know a-priori the time evolution $\mathcal{H}_s$ of the human motion. For a given set $\mathcal{S}$ containing n_s realizations of the state – namely, $s_j, j = 1, \ldots, n_s$ – and corresponding sets of DNN and NMPC actions – respectively, $\mathcal{A}^{DNN}$ and $\mathcal{A}^{NMPC}$ – a quadratic loss function can be calculated as

$$L = \sum_{j=1}^{n_s} \left(a^{DNN}(s_j) - a^{NMPC}(s_j) \right)^2 . \tag{7}$$

For a given number of training episodes N_e, the DNN policy at episode i, namely $a_i^{DNN}(s)$ (which belongs to set Π of all possible policies implementable by the given DNN), is updated by first obtaining the corresponding loss function, namely L_i, and then running one step of back-propagation to update the DNN to implement $a_{i+1}^{DNN}(s)$. At each step, policy $a_i^{DNN}(s)$ is also validated on a different set of states not included in $\mathcal{S}$, and in the end, the policy that minimizes L on this set is chosen as the optimal policy, namely $a_*^{DNN}(s)$. Though the described procedure holds in general, different methods can be used to generate the set of states $\mathcal{S}$, as explained in the following.

A. DNN TRAINING METHODS

The most intuitive way of training the DNN can be referred to as "naive" or "offline". This training strategy is described in Algorithm 1. In it, $\mathcal{S}$ is defined using the NMPC policy before starting training the DNN (hence the term "offline") and, at each episode i, the DNN generates actions corresponding to states in $\mathcal{S}$. Although more than satisfactory results were obtained in [25] using this training technique, its drawback is that the distribution of states encountered by the DNN during testing will be in general different from that encountered by the expert, which could lead to poor performance or even instability. In the remainder of the paper, we refer to the resulting DNN as O-DNN, where the "O" stands for "offline-trained".

Algorithm 1 Offline Training: O-DNN

```
Generate S, A^NMPC via NMPC
Set a_0^DNN(s) to any policy in Π
for i ← 0 to N_e do
    Run policy a_i^DNN on S to get A^DNN
    Calculate L_i(A^NMPC, A^DNN)
    Determine a_{i+1}^DNN(s) via back-propagation
end for
Return best policy a_*^DNN(s) based on validation.
```

To overcome this issue in a general imitation learning setting, Ross et al. in [24] proposed the so-called DAgger algorithm, which iteratively trains the agent online (i.e., new states are generated using the current policy $a_i^{DNN}(s)$) on a continuously aggregating dataset. In our work, we employ a version of DAgger in which states generated via NMPC are still used in the first N_0 episodes, to avoid employing a completely "untrained" DNN. This is to avoid generating several data points that violate the NMPC constraints, as these

states would not be useful for training (indeed, NMPC would not be able to find a solution in those states). With reference to Algorithm 2, sets $\mathcal{S}$, $\mathcal{A}^{DNN}$ and $\mathcal{A}^{NMPC}$ are first initialized as empty sets. Then, for the first N_0 episodes (initialization) new NMPC-generated data $\mathcal{S}_i$ and $\mathcal{A}_i^{NMPC}$ are aggregated to $\mathcal{S}$ and $\mathcal{A}^{NMPC}$, respectively. The DNN policy $a_i^{DNN}(s)$ is then executed on the resulting set $\mathcal{S}$ to generate the corresponding set of actions $\mathcal{A}^{DNN}$. The loss function L_i at the i-th episode is calculated and used to generate the new policy $a_{i+1}^{DNN}(s)$. After initialization, a similar dataset aggregation procedure is still used until the last episode N_e, but now the new states $\mathcal{S}_i$ are always generated via DNN rather than via NMPC. This should allow for a better exploration of the state space as compared to Algorithm 1. The resulting DNN that implements the optimal policy with this training procedure will be referred to in the following as DA-DNN, where "DA" stands for "dataset aggregation".

Algorithm 2 Dataset Aggregation: DA-DNN

$\mathcal{S} \leftarrow \emptyset$, $\mathcal{A}^{DNN} \leftarrow \emptyset$, $\mathcal{A}^{NMPC} \leftarrow \emptyset$
Set $a_0^{DNN}(s)$ to any policy in Π
for $i \leftarrow 0$ to N_0 **do**
  Generate $\mathcal{S}_i$, $\mathcal{A}_i^{NMPC}$ via NMPC
  Aggregate datasets: $\mathcal{S} \leftarrow \mathcal{S} \bigcup \mathcal{S}_i$
  Aggregate datasets: $\mathcal{A}^{NMPC} \leftarrow \mathcal{A}^{NMPC} \bigcup \mathcal{A}_i^{NMPC}$
  Run policy $a_i^{DNN}(s)$ on $\mathcal{S}$ to get new $\mathcal{A}^{DNN}$
  Calculate $L_i(\mathcal{A}^{NMPC}, \mathcal{A}^{DNN})$
  Determine $a_{i+1}^{DNN}(s)$ via back-propagation
end for
for $i \leftarrow N_0 + 1$ to N_e **do**
  Generate $\mathcal{S}_i$ via π_i
  Run NMPC on $\mathcal{S}_i$ to get $\mathcal{A}_i^{NMPC}$
  Aggregate datasets: $\mathcal{S} \leftarrow \mathcal{S} \bigcup \mathcal{S}_i$
  Aggregate datasets: $\mathcal{A}^{NMPC} \leftarrow \mathcal{A}^{NMPC} \bigcup \mathcal{A}_i^{NMPC}$
  Run policy $a_i^{DNN}(s)$ on $\mathcal{S}$ to get new $\mathcal{A}^{DNN}$
  Calculate $L_i(\mathcal{A}^{NMPC}, \mathcal{A}^{DNN})$
  Determine $a_{i+1}^{DNN}(s)$ via back-propagation
end for
Return best policy $a_*^{DNN}(s)$ based on validation.

The DNN policy trained with Algorithm 1 can immediately rely on a much larger set of states $\mathcal{S}$ as compared to Algorithm 2, in which this set is initially empty and increases at each episode. To better be able to compare the performance of the offline method with that of the DA-DNN, we propose to run the offline method using a dataset-aggregation approach. The resulting procedure, illustrated in Algorithm 3, consists of the practice of extending the initialization phase of Algorithm 2 until the last episode N_e. We refer to the resulting DNN as ODA-DNN, where "ODA" stands for "offline dataset aggregation".

In order to condense the information provided as input to the DNN (thus reducing the number of DNN parameters to be trained), the encoder part of an autoencoder [27] can be used prior to the DNN input layer to process $\mathcal{H}$,

Algorithm 3 Offline Dataset Aggregation: ODA-DNN

$\mathcal{S} \leftarrow \emptyset$, $\mathcal{A}^{DNN} \leftarrow \emptyset$, $\mathcal{A}^{NMPC} \leftarrow \emptyset$
Set $a_0^{DNN}(s)$ to any policy in Π
for $i \leftarrow 0$ to N_e **do**
  Generate $\mathcal{S}_i$, $\mathcal{A}_i^{NMPC}$ via NMPC
  Aggregate datasets: $\mathcal{S} \leftarrow \mathcal{S} \bigcup \mathcal{S}_i$
  Aggregate datasets: $\mathcal{A}^{NMPC} \leftarrow \mathcal{A}^{NMPC} \bigcup \mathcal{A}_i^{NMPC}$
  Run policy $a_i^{DNN}(s)$ on $\mathcal{S}$ to get new $\mathcal{A}^{DNN}$
  Calculate $L_i(\mathcal{A}^{NMPC}, \mathcal{A}^{DNN})$
  Determine $a_{i+1}^{DNN}(s)$ via back-propagation
end for
Return best policy $a_*^{DNN}(s)$ based on validation.

generating a vector $\mathcal{H}_E$ of reduced size (this idea was already applied in [25]). More precisely, a symmetrical autoencoder is defined with $\mathcal{H}$ as input, and trained to reproduce this input through a latent space expressing $\mathcal{H}_E$. The decoder (which reconstructs $\mathcal{H}$ given $\mathcal{H}_E$) is removed, and only the encoder (which generates $\mathcal{H}_E$ given $\mathcal{H}$) is inserted before our DNN. In order to indicate the presence of the encoder in the above-mentioned algorithms, the letter "E" will be introduced in their acronyms: for example, the version of DA-DNN with encoder will be referred to as DA-E-DNN.

B. DEFINITION OF THE SAFETY FILTER

Approximating the DNN algorithms might lead to violating the constraints that were imposed when generating the NMPC law, which might be particularly problematic when the SSM conditions are violated, as this would hinder safety. As a possible solution, one can avoid directly applying the joint velocity references (namely, $\boldsymbol{\omega}$) determined by $a_*^{DNN}(s)$ when actually testing the algorithm. Instead of setting $\boldsymbol{\omega} = a_*^{DNN}(s)$, one can calculate

$$\boldsymbol{\omega} = \arg \min_{a \in \mathcal{A}_{SSM}(s)} \left(a - a_*^{DNN}(s)\right)^2 \tag{8}$$

by numerically solving, at each sampling time, a nonlinear program in which the decision variable a belongs to set $\mathcal{A}_{SSM}(s)$ of actions that satisfy the SSM constraints for that specific state. If $a_*^{DNN}(s)$ satisfies the SSM constraints, then one would obtain $\boldsymbol{\omega} = a_*^{DNN}(s)$, which implies $\left(a - a_*^{DNN}(s)\right)^2 = 0$. In the worst-case scenario, imposing $\boldsymbol{\omega} = 0$ would always guarantee safety from the SSM point of view. This approach, to which we can refer as *safety filter*, resembles the *predictive safety filter* approach proposed in [28] and [29]. However, while an actual NMPC problem is solved to determine the action when using a predictive safety filter, our approach solves a simpler optimization problem – without prediction – only aimed at determining the current action. The nonlinear program to be solved is much simpler than that solved by NMPC, leading to relatively low computation times. On the downside, our safety filter cannot guarantee closed-loop stability, which can instead be achieved using the predictive safety filter. When incorporating safety filters in the implementation of

the above-mentioned algorithms, the letter "S" is introduced in the acronyms: for example, DA-E-DNN with safety filter becomes DA-SE-DNN.

IV. CASE STUDY

In the considered case study, all the above-described DNN-based controllers were designed and implemented for a Universal Robots UR5 manipulator with six degrees of freedom (i.e., the robot configuration is described by $n_\theta = 6$ joint angles, grouped in vector $\boldsymbol{\theta} \in \mathbb{R}^6$). The task of the robot was to reach the following two different target configurations in sequence:

$$\boldsymbol{\theta}_f = \begin{bmatrix} 0.0 & -2.3 & -1.1 & -1.2 & -1.2 & 0.5 \end{bmatrix}' \triangleq \boldsymbol{\theta}_A,$$

and

$$\boldsymbol{\theta}_f = \begin{bmatrix} 3.0 & -1.6 & -1.7 & -1.7 & -1.7 & 1.0 \end{bmatrix}' \triangleq \boldsymbol{\theta}_B,$$

where all angles are expressed in radians. A total of $n_r = 7$ test points were located on the robot frame, and a (virtual) sphere with a constant radius was defined as centered at each test point, such that the union of the spheres covered the whole manipulator. The radii were set as $R_{r,1} = R_{r,2} = R_{r,3} = 0.15$ m, $R_{r,4} = R_{r,5} = 0.08$ m, $R_{r,6} = 0.12$ m and $R_{r,7} = 0.1$ m. The robot base was located on a table with a height of 1.2 m; to avoid the table, it was imposed in the NMPC problem that none of the robot spheres should enter into contact with the table surface. This condition defined sets $\mathcal{P}_i$ introduced in Section II-A2 by imposing that the z-coordinate of $\boldsymbol{p}_{r,i}$, $i \in \mathbb{N}_r$, remained above the table level of a distance greater than the corresponding $R_{r,i}$. In addition to this obstacle avoidance constraint, upper bounds of joint speeds were imposed by defining set Ω introduced in Section II-A2 as $\Omega = \left\{\boldsymbol{\omega} \in \mathbb{R}^6 : \|\boldsymbol{\omega}\|_\infty \leq 1 \text{ rad/s}\right\}$. To obtain the speed and location of the above-mentioned robot test points, the differential and forward kinematics of the robot manipulator were expressed using homogeneous transformation matrices based on the Denavit-Hartenberg parameters already utilized in [11].

As for the human operator's space occupancy, only the upper part of the human body was considered, as the robot frame was imposed to always remain above the table. Similarly to the robot, the upper human body was covered by a number of spheres (precisely, $n_h = 14$) centered at different locations (test points) of torso, head and arms. The location and radii of the spheres coincide with those used in [11], to which we refer the reader. These points constitute vector $\mathcal{H}$: as each of them is made of three Cartesian coordinates, we have that $\mathcal{H} \in \mathbb{R}^{42}$. The SSM constraints were imposed in the NMPC problem for each robot sphere, to guarantee to be able to reach zero speed when a possible collision with a human sphere would happen, assuming that $\bar{v}_h = 2$ m/s (as in [3]), $\epsilon_s = 10^{-3}$ m (in line with the precision of state-of-the-art optical motion capture systems [30]) and $T_{dr} = 100$ ms, based on the fact that T_σ was set equal to 50 ms. After estimating suitable values of $\bar{a}_i$ for each joint, the parameters of the SSM constraints imposed in the NMPC controllers (see, (2)) were set as $\alpha_1 = 2.79$ s^{-1}, $\bar{d}_1 = 0.15$ m $\alpha_2 = 1.95$ s^{-1}, $\bar{d}_2 = 0.15$ m, $\alpha_3 = 1.00$ s^{-1}, $\bar{d}_3 = 0.15$ m, $\alpha_4 = 0.80$ s^{-1}, $\bar{d}_4 = 0.15$ m, $\alpha_5 = 0.65$ s^{-1}, $\bar{d}_5 = 0.17$ m, $\alpha_6 = 0.45$ s^{-1}, $\bar{d}_6 = 0.94$ m, $\alpha_7 = 0.35$ s^{-1} and $\bar{d}_7 = 1.77$ m, following the approach described in [25]. The human test points were obtained from the motion in the dataset introduced in [26], in which human subjects would walk around a room and complete tasks such as screwing bolts at different heights, untying a knot, and replacing loads with different weights on shelves.

To define the NMPC controllers, we set $T_\ell = 500$ ms, $T_\sigma = 50$ ms and $N = 10$. The terms ℓ in equation (4) in the NMPC cost function were defined with $\boldsymbol{Q} = 10 \cdot \boldsymbol{I}_{6\times6}$, $\boldsymbol{R} = \boldsymbol{I}_{6\times6}$, with $\boldsymbol{I}$ being an identity matrix. In the same equation we set $\gamma = 500$, and we refer the reader to [11, Sec. 14.A] for the detailed definition of function $\varphi(\cdot,\cdot)$.

The motion of 13 different human subjects in the above-mentioned dataset was used. Each subject repeated a sequence of 3 different motions for 5 times. The 13 motion sequences were divided as follows: 10 for training, 2 for validation, and 1 for the testing process. Considering that a sampling of the human motion was used to align with the 50 ms sampling time of the NMPC/DNN controllers, a total number of $1.35 \cdot 10^6$ data points were used for all 13 human subjects.

The same DNN structure was used for all considered training methods (i.e., O-DNN, DA-DNN and ODA-DNN, with their variants with safety filter and/or encoder, amounting to a total of 12 methods). It consisted of an input layer with 48 neurons acquiring state $s \in \mathbb{R}^{48}$ (6 neurons for robot joint positions $\boldsymbol{\theta}$ and 42 for the positions of 14 center of spheres on the human upper body in the Cartesian space). When encoders were used, these generated a new vector $\mathcal{H}_E \in \mathbb{R}^{24}$ from $\mathcal{H}$. The output layer consisted of 6 neurons, one for each component of $\boldsymbol{\omega}$. The sizes of the hidden layers were defined as shown in Fig. 1. The number of hidden layers and of neurons per layer were determined via trial and error, training DNNs with different structures and observing which structures would lead to better performance.

A hyperbolic tangent function was used as an activation function to automatically satisfy the above-mentioned constraint on joint speeds. All proposed algorithms were designed based on the PyTorch Python library. The Adam optimizer, with a learning rate equal to 10^{-3}, was used for training. Also, dropout [31], with a rate of 25% per hidden layer, was employed during the training phase to prevent overfitting. For each method, four DNNs were trained, and specifically: two for learning to imitate NMPC with $\boldsymbol{\theta}_A$ as goal configuration, one of which was active in a neighborhood of $\boldsymbol{\theta}_A$ (specifically, for $\boldsymbol{\theta}$ such that $\|\boldsymbol{\theta} - \boldsymbol{\theta}_A\|_2 \leq 0.18$ rad) and the other one outside this neighborhood; the other two for learning to imitate NMPC with $\boldsymbol{\theta}_B$ as goal configuration, one of which was active in a neighborhood of $\boldsymbol{\theta}_B$ (specifically, for $\boldsymbol{\theta}$ such that $\|\boldsymbol{\theta} - \boldsymbol{\theta}_B\|_2 \leq 0.18$ rad), and

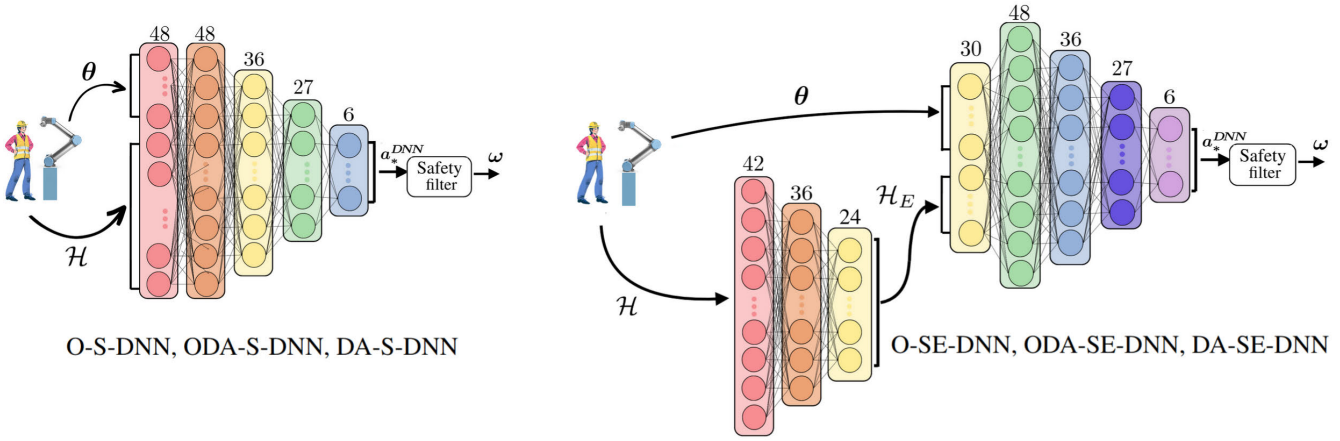

FIGURE 1. Schematics of the DNNs designed in the case study. O-S-DNN, ODA-S-DNN and DA-S-DNN share the same structure, and the same holds for O-SE-DNN, ODA-SE-DNN and DA-SE-DNN. For the cases when the encoder is present, it is represented as a separate neural network. The same structure applies to the cases without safety filters, in which case ω would be taken equal to the DNN output a_*^{DNN}.

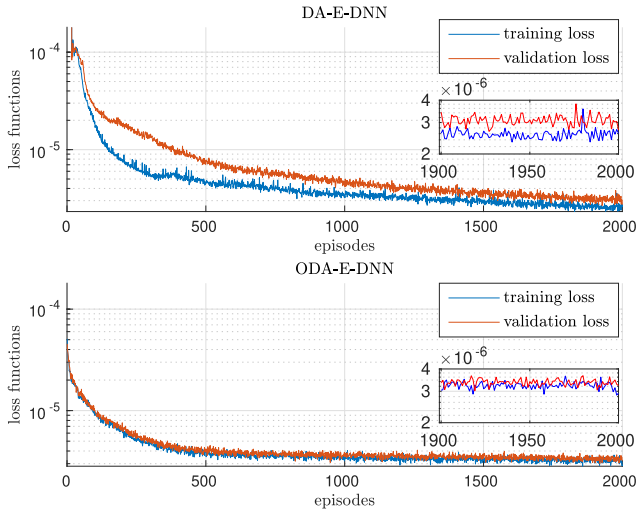

FIGURE 2. Learning curves for DA-E-DNN and ODA-E-DNN, employing a semi-logarithmic scale. For the sake of simplicity, we only show the case when $\theta_f = \theta_B$ and $\|\theta - \theta_B\|_2 \geq 0.18$ rad.

the other one outside this neighborhood. The use of different DNNs close and far from the goal point, as in [25], had the role of reducing steady-state errors in the robot configuration.

V. NEURAL NETWORK TRAINING AND ANALYSIS

All methods were trained for a total of $N_e = 2000$ episodes, whereas the initialization of DA-based methods consisted of the first $N_0 = 30$ episodes. As an example, the learning curves for ODA-E-DNN and DA-E-DNN are shown in Fig. 2. We do not show learning curves for O-DNN as the available amount of data per episode is different than in ODA-E-DNN and DA-E-DNN, and a comparison would not be possible: indeed, the ability of making this comparison was the main reason for introducing ODA-based algorithms. Both methods were trained on data from the above-mentioned training set and, in parallel, run on the validation set. As can be noticed in Fig. 2, ODA-E-DNN initially converged faster than DA-E-DNN, probably because DA-E-DNN had to explore a larger portion of the environment. On the other hand, both training and validation values of the loss function for DA-E-DNN converged to lower values than their counterparts for ODA-E-DNN, as can be noticed in the zoomed portions of Fig. 2. In particular, considering the average of the values of the last 100 episodes – to compensate for the presence of oscillations in the loss function – ODA-E-DNN presented a value of the training (respectively, validation) loss 12.4% higher (respectively, 18.5% higher) than that of DA-E-DNN. This confirms the ability of dataset aggregation to improve the performance of the naive approach to imitation learning.

To assess the quality of the trained DNNs, a visualization analysis was conducted based on the concepts described in [32], and the results are shown in Fig. 3. For the sake of brevity, we restricted our analysis to the DA-E-DNN case with θ_B as goal configuration. This DNN showed, together with its version with safety filter, the best performance in terms of experimental results (see Section VI).

First, the so-called *saliency map* was created to highlight the importance of different features. A saliency map is a visualization tool that shows which input values are most important in the decision-making process by calculating gradients [32]. As shown at the top of Fig. 3, the saliency map includes 30 features: 6 for joint positions and 24 for encoded human poses (coherently with the size of the input layers of DNNs with encoders, as shown in Fig. 1). For this specific combination of initial configuration, final configuration and human motion, the position of the fifth joint seems to be more relevant than the others, and the same can be said of certain components of the encoded human position (for instance, input 13). This, however, could vary considerably when a different human motion is used for training. In any case all input parameters, joint positions and human positions, contribute to the performance of the DNN. None of them can be excluded from the input parameters, as each one carries valuable information to the model. For example, excluding

an encoded human position or joint position of the robot could reduce the ability of the model to solve unseen inputs effectively.

To describe how neuron outputs are spread across the three hidden layers, normalized activation distributions are shown in the center of Fig. 3. The pattern of the outputs of the input data that flows through the layers can provide information on how well the layer is working. Activations of the model are spread around zero with some variation, which means that the network was trained well [33].

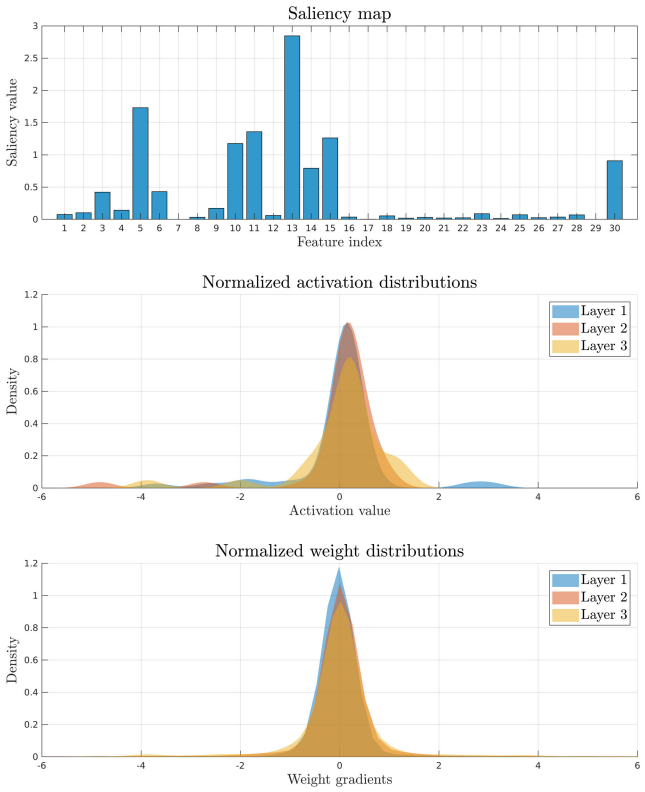

FIGURE 3. Visualization of the DA-E-DNN model for $\theta_f = \theta_B$. Saliency map (top), normalized histogram of activation values (center), and normalized histogram of weight gradients (bottom), all were tested on a sample test input.

Another way to identify if the model is trained well is to analyze the normalized weight distribution (bottom of Fig. 3). Balanced weights that are centered around zero allow unbiased and flexible learning. The normalized weight distribution graph indicates that the DNN is well-trained and unbiased.

VI. EXPERIMENTAL RESULTS AND DISCUSSION

All aforementioned algorithms were implemented on a laptop with an i7-9750H CPU, 16 GB RAM, and NVIDIA GeForce GTX 1660 Ti to complete experiments with the real UR5 robot. The NMPC law was generated via acados [34] and implemented as a C++ node in Robot Operating System (ROS). Meanwhile, DNN algorithms were trained with the PyTorch python library on a PC with an Intel Xeon CPU, 16GB RAM, and NVIDIA GeForce GTX 1080Ti. The implementation of the safety filter was realized in CasADi, an open-source toolkit designed for nonlinear optimization and algorithmic differentiation [35]. The URsim industrial simulator was utilized during training instead of the real robot, so as to complete the training process faster and avoid risking to damage the robot, especially when exploring the state space using the DNN-generated laws with DA-based algorithms.

The joint speed reference $\boldsymbol{\omega}$ generated by NMPC or by any of the DNN-based methods described above were sent from the laptop to the UR5 through the ROS package called *Universal Robot driver* every 50 ms. The algorithms were tested on 50 separate portions of the test set. Compared to the human subjects used for training, the human subject motion used for testing was on average 10 cm closer to the robot, so as to evaluate how the robot would react to a partially different human behavior.

The algorithms were compared with each other in terms of the NMPC cost value J_{exp} (evaluated by summing the terms ℓ in (4) for all sampling instants until task completion), and the time T_c required to complete the task (moving from $\boldsymbol{\theta}_A$ to $\boldsymbol{\theta}_B$ and then back to $\boldsymbol{\theta}_A$). We also considered the maximum/average computation times of the controllers, denoted, respectively, as $\bar{\tau}$ and $\hat{\tau}$.

Table 1 summarizes the results of the experiments, with the values in bold representing the best result – either among methods with or without safety filter – for each considered measure. The best values of J_{exp} and T_c were both achieved by DA-E-DNN among methods without safety filter and by DA-SE-DNN among methods with safety filter. For most methods, the presence of the encoder increased the computation time and improved J_{exp} (probably due to the more compact representation of information in $\mathcal{H}_E$ compared to $\mathcal{H}$, and the consequent reduction in the number of DNN parameters), whereas T_s was not influenced by it. DA-based approaches always achieved a better performance in terms of T_s compared to the corresponding O-based or ODA-based methods; the same was true for J_{exp}, with the exception of methods without encoder and without safety filter. This

TABLE 1. Average values of the considered measures for 50 experiments.

	J_{exp}	T_c (s)	$\bar{\tau}$ (ms)	$\hat{\tau}$ (ms)
NMPC	6257.2	18.9	119	43.8
O-DNN	6078.5	17.9	2.30	0.78
ODA-DNN	5680.8	17.1	**1.90**	**0.73**
DA-DNN	5850.5	16.5	3.00	0.81
O-E-DNN	5793.9	18.2	6.50	1.35
ODA-E-DNN	5429.7	17.5	6.20	1.30
DA-E-DNN	**5378.7**	**15.3**	29.4	1.40
O-S-DNN*	6913.6	19.8	2.30	0.70
ODA-S-DNN*	7254.0	19.4	3.60	0.67
DA-S-DNN	5875.2	16.8	**2.20**	**0.72**
O-SE-DNN*	6973.8	20.4	29.7	1.55
ODA-SE-DNN*	6793.6	19.2	31.1	1.60
DA-SE-DNN	**5722.8**	**16.0**	32.4	1.06

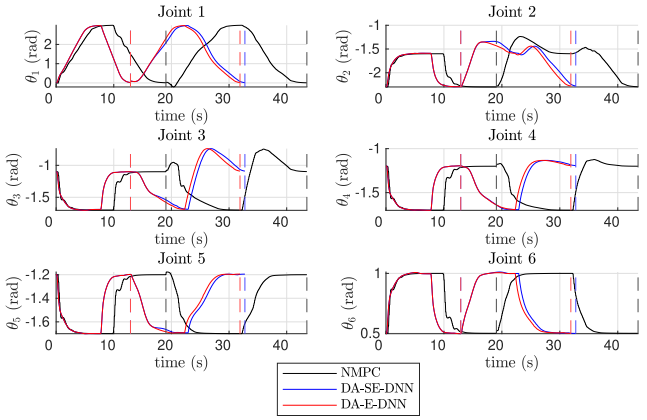

FIGURE 4. Time evolution of joint positions θ_i for NMPC, DA-E-DNN and DA-SE-DNN. The vertical dashed lines denote the time instants at which the joint position depicted by the particular color accomplishes the process from θ_A to θ_B and back.

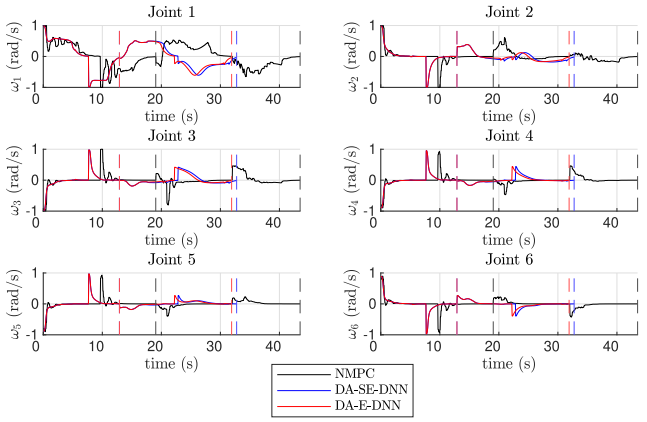

FIGURE 5. Time evolution of joint speeds ω_i for NMPC, DA-E-DNN and DA-SE-DNN, similarly to the joint positions illustrated in Fig. 4. The vertical dashed lines denote the time instants at which the joint position depicted by the particular color accomplishes the process from θ_A to θ_B and back.

confirms that the proposed approach outperforms the naive (offline) approach proposed in [25]. The presence of the safety filter introduced a perturbation to the actual DNN output to enforce SSM constraints; as the enforcement of these constraints limited the robot speed, the value of T_c obtained with DA-SE-DNN was, as one would expect, higher than that obtained with DA-E-DNN. J_{exp}, instead, would be decreased by this speed reduction on the one hand (as terms $\boldsymbol{\omega}_t^T \boldsymbol{R} \boldsymbol{\omega}_t$ in (4) would be lower) and would be increased by the resulting slower convergence to the goal configuration (via terms $\left(\boldsymbol{\theta}_t - \boldsymbol{\theta}_f\right)^T \boldsymbol{Q} \left(\boldsymbol{\theta}_t - \boldsymbol{\theta}_f\right)$ in (4)); the results in Table 1 show that this second effect was more pronounced, as J_{exp} obtained with DA-SE-DNN was higher than that obtained with DA-E-DNN.

It is important to mention that a number of methods, for at least a subset of the experimental trials, could not reach the goal configuration: these methods are reported with an asterisk in Table 1, and are all comprising a safety filter. The reason for this phenomenon can be the fact that, when using the safety filter, the robot reached configurations in portions of the state space never experienced during training, which in

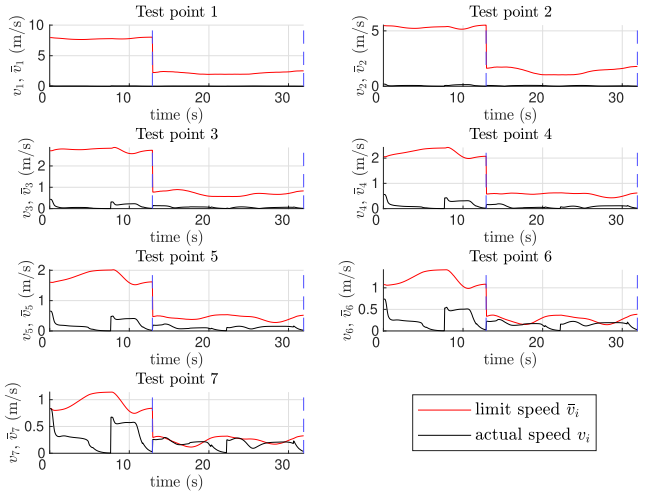

FIGURE 6. Time evolution of linear velocities of robot test points v_i for the DA-E-DNN method, together with a corresponding limit speed $\bar{v}_i$, for the robot motions shown in Figs. 4 and 5. Task completion is indicated by the vertical, dashed blue lines.

turn led the DNN to generate meaningless action values. The problem occurred only for O-based and ODA-based methods, for which a better exploration of the workspace could not be obtained (as all state trajectories were generated by the NMPC expert). On the other hand, DA-based methods did not suffer from this problem.

Even if, up to this point, DA-E-DNN seems to be the best method, the absence of the safety filter can possibly lead to the violation of SSM constraints and hinder safety. To better understand this issue, we further analyze results for DA-E-DNN and DA-SE-DNN, comparing them with the NMPC expert that is imitated (but which cannot rely on perfect human predictions, as when generating training data, when implemented in reality). Time evolutions of the robot joint angles and speeds for these methods are reported and compared, respectively, in Figs. 4 and 5. These figures illustrate two robot motions from $\boldsymbol{\theta}_A$ to $\boldsymbol{\theta}_B$ and back. The depicted dashed lines separate two different robot motions, first with the human working far from the robot, and then with the human moving close to the robot. Figure 4 confirms the results of Table 1 in terms of T_c: both DNN-based methods achieved faster motions than NMPC, with DA-E-DNN outperforming DA-SE-DNN. It can be seen from Fig. 5 that the range of speed values is very similar for all three methods, which means that all DNNs were trained suitably to mimic the NMPC behavior.

To ascertain safety, the time evolution of the speed of the test point on which the SSM constraint is imposed is shown in Figs. 6 and 7 for DA-E-DNN and DA-SE-DNN, respectively, together with the corresponding SSM upper bounds. As can be seen in Fig. 6, DA-E-DNN violates the SSM constraints for test points 6 and 7 during the second motion, when the human operator comes closer to the robot. Specifically, the SSM constraints are violated during 33.42% of the total time. On the other hand, DA-SE-DNN satisfies the constraints due to the presence of the safety filter.

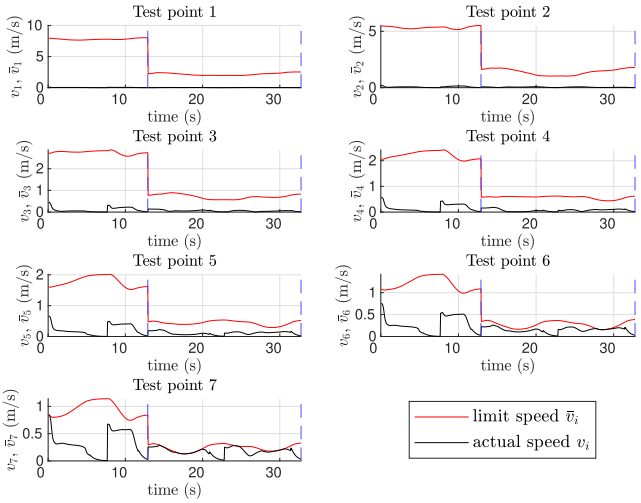

FIGURE 7. Time evolution of linear velocities of robot test points v_i for the DA-SE-DNN method, together with a corresponding limit speed $\bar{v}_i$, for the robot motions shown in Figs. 4 and 5. Task completion is indicated by the vertical, dashed blue lines.

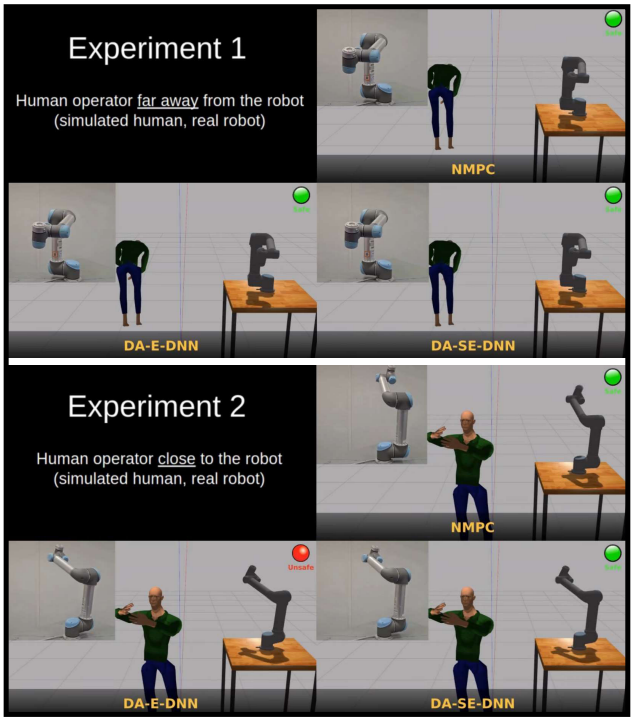

FIGURE 8. Frames from the video provided as supplementary material, representing the motion of the real robot (Universal Robots UR5) for the considered three algorithms, and a simulation via the Gazebo software that reproduces the same motion of the robot in the experimental setup, together with the position of the moving human that is passed in real time to the algorithm that determines the references for the robot joint speeds.

We also assessed the robustness of the safety filter by translating the sequence of human movements such that the operator would stay much closer to the robot as compared to the previously-described test set. This depicts a situation never experienced during training, in which a considerable loss of performance is expected. The results, restricted to the cases of NMPC and the two best-performing DNNs for the cases with and without safety filters, can be seen in Table 2. Compared to Table 1, one can notice that both J_{exp} and T_c increased considerably, due to the closer presence of the human operator. This situation also led to a violation of the SSM constraints in 40.67% of the time for DA-E-DNN, which is higher than in the previous dataset. However, both NMPC and DA-SE-DNN succeeded in never violating the SSM constraints. This shows that, even if a loss of performance has to be expected when the human motion considerably differs from the training data, safety is ensured by the safety filter in any case.

TABLE 2. Values of the considered measures for an experiment to test the robustness of the safety filter.

	J_{exp}	T_c (s)	SSM violation
NMPC	13534	21.9	0.00%
DA-E-DNN	10752	16.2	40.67%
DA-SE-DNN	12953	18.6	0.00%

A video is provided as supplementary material, showing the motion of the real robot and of the simulated human using the Gazebo software for the results reported in Figs. 4-7. Specific frames of the video are also shown in Fig. 8. In the upper part of Fig. 8, the human operator is relatively far from the robot, whereas in the lower portion of the figure the human is close to the robot. In this second case, the figure shows a time frame when the DA-E-DNN algorithm is violating the SSM constraint (notice the red light in the upper-right corner of the DA-E-DNN image, indicating violation of the SSM constraints, in lieu of the green light of the other images, indicating compliance with SSM). In conclusion, DA-SE-DNN appears to be the method offering the best trade-off between performance and safety.

VII. CONCLUSION AND FUTURE WORK

The paper proposed a set of methods for deep imitation learning of NMPC laws for safe pHRI via SSM. Compared to previous work, the main contributions consist of the use of dataset aggregation-based training and of the use of safety filters. Experimental results showed that the DA-SE-DNN method, employing an encoder (to condense the information provided as input to the DNN), aggregation-based training and a safety filter, provided the best trade-off between safety and performance.

Future work will be devoted to the implementation of the proposed imitation learning framework in more complex scenarios, for example considering an actual production line in a factory, and using different collaborative manipulators. We will also explore the design of new learning-based motion planning strategies for the same safe pHRI problem, based on deep reinforcement learning.

REFERENCES

[1] P. K. R. Maddikunta, Q.-V. Pham, B. Prabadevi, N. Deepa, K. Dev, T. R. Gadekallu, R. Ruby, and M. Liyanage, "Industry 5.0: A survey on enabling technologies and potential applications," *J. Ind. Inf. Integr.*, vol. 26, Mar. 2022, Art. no. 100257.

[2] S. Haddadin, "Physical human–robot interaction," in *Springer Handbook of Robotics*. Cham, Switzerland: Springer, 2016, pp. 1835–1874.
[3] J. A. Marvel, "Performance metrics of speed and separation monitoring in shared workspaces," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 2, pp. 405–414, Apr. 2013.
[4] *Robots Robotic Devices—Collaborative Robots*, Standard ISO-TS 15066, Int. Org. for Standardization, Geneva, Switzerland, 2016.
[5] M. J. Rosenstrauch, T. J. Pannen, and J. Krüger, "Human robot collaboration–using Kinect v2 for ISO/TS 15066 speed and separation monitoring," *Proc. CIRP*, vol. 76, pp. 183–186, Jan. 2018.
[6] E. Kim, R. Kirschner, Y. Yamada, and S. Okamoto, "Estimating probability of human hand intrusion for speed and separation monitoring using interference theory," *Robot. Comput.-Integr. Manuf.*, vol. 61, Feb. 2020, Art. no. 101819.
[7] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, and B. Matthias, "Safety in human–robot collaborative manufacturing environments: Metrics and control," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 882–893, Apr. 2016.
[8] A. M. Zanchettin, P. Rocco, S. Chiappa, and R. Rossi, "Towards an optimal avoidance strategy for collaborative robots," *Robot. Comput.-Integr. Manuf.*, vol. 59, pp. 47–55, Oct. 2019.
[9] P. Zheng, P.-B. Wieber, and O. Aycard, "Online optimal motion generation with guaranteed safety in shared workspace," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 9210–9215.
[10] P. Glogowski, A. Böhmer, A. Hypki, and B. Kuhlenkötter, "Robot speed adaption in multiple trajectory planning and integration in a simulation tool for human–robot interaction," *J. Intell. Robotic Syst.*, vol. 102, no. 1, pp. 1–20, May 2021.
[11] A. Oleinikov, S. Kusdavletov, A. Shintemirov, and M. Rubagotti, "Safety-aware nonlinear model predictive control for physical human–robot interaction," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 5665–5672, Jul. 2021.
[12] M. Eckhoff, R. J. Kirschner, E. Kern, S. Abdolshah, and S. Haddadin, "An MPC framework for planning safe & trustworthy robot motions," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 4737–4742.
[13] A. Oleinikov, S. Soltan, Z. Balgabekova, A. Bemporad, and M. Rubagotti, "Scenario-based model predictive control with probabilistic human predictions for human–robot coexistence," *Control Eng. Pract.*, vol. 142, Jan. 2024, Art. no. 105769.
[14] T. Parisini and R. Zoppoli, "A receding-horizon regulator for nonlinear systems and a neural approximation," *Automatica*, vol. 31, no. 10, pp. 1443–1451, Oct. 1995.
[15] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Comput. Surveys*, vol. 50, no. 2, pp. 1–35, Mar. 2018.
[16] S. S. P. Kumar, A. Tulsyan, B. Gopaluni, and P. Loewen, "A deep learning architecture for predictive control," in *Proc. IFAC Int. Symp. Adv. Control Chem. Processes*, Jan. 2018, pp. 506–511.
[17] S. Lucia and B. Karg, "A deep learning-based approach to robust nonlinear model predictive control," in *Proc. IFAC Conf. Nonlinear Model Predictive Control*, Jan. 2018, pp. 511–516.
[18] Y. Löhr, M. Mönnigmann, M. Klauco, and M. Kalúz, "Mimicking predictive control with neural networks in domestic heating systems," in *Proc. 22nd Int. Conf. Process Control*, Jun. 2019, pp. 19–24.
[19] S. Lucia, D. Navarro, B. Karg, H. Sarnago, and Ó. Lucía, "Deep learning-based model predictive control for resonant power converters," *IEEE Trans. Ind. Informat.*, vol. 17, no. 1, pp. 409–420, Jan. 2021.
[20] A. D. Bonzanini, J. A. Paulson, G. Makrygiorgos, and A. Mesbah, "Fast approximate learning-based multistage nonlinear model predictive control using Gaussian processes and deep neural networks," *Comput. Chem. Eng.*, vol. 145, Feb. 2021, Art. no. 107174.
[21] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 5628–5635.
[22] P. M. Kebria, A. Khosravi, S. M. Salaken, and S. Nahavandi, "Deep imitation learning for autonomous vehicles based on convolutional neural networks," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 1, pp. 82–95, Jan. 2020.
[23] D. Seita, A. Ganapathi, R. Hoque, M. Hwang, E. Cen, A. K. Tanwani, A. Balakrishna, B. Thananjeyan, J. Ichnowski, N. Jamali, K. Yamane, S. Iba, J. Canny, and K. Goldberg, "Deep imitation learning of sequential fabric smoothing from an algorithmic supervisor," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 9651–9658.
[24] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. Int. Conf. Artif. Intell. Statist.*, Jan. 2010, pp. 627–635.
[25] A. Nurbayeva, A. Shintemirov, and M. Rubagotti, "Deep imitation learning of nonlinear model predictive control laws for safe physical human–robot interaction," *IEEE Trans. Ind. Informat.*, vol. 17, no. 9, pp. 8384–8395, Oct. 2022.
[26] P. Maurice, A. Malaisé, C. Amiot, N. Paris, G.-J. Richard, O. Rochel, and S. Ivaldi, "Human movement and ergonomics: An industry-oriented dataset for collaborative robotics," *Int. J. Robot. Res.*, vol. 38, no. 14, pp. 1529–1537, Dec. 2019.
[27] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction," *Neurocomputing*, vol. 184, pp. 232–242, Apr. 2016.
[28] K. P. Wabersich and M. N. Zeilinger, "A predictive safety filter for learning-based control of constrained nonlinear dynamical systems," *Automatica*, vol. 129, Jul. 2021, Art. no. 109597.
[29] B. Tearle, K. P. Wabersich, A. Carron, and M. N. Zeilinger, "A predictive safety filter for learning-based racing control," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7635–7642, Oct. 2021.
[30] M. Topley and J. G. Richards, "A comparison of currently available optoelectronic motion capture systems," *J. Biomechanics*, vol. 106, Jun. 2020, Art. no. 109820.
[31] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.
[32] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2013, *arXiv:1312.6034*.
[33] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, Mar. 2010, pp. 249–256.
[34] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. V. Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados—A modular open-source framework for fast embedded optimal control," *Math. Program. Comput.*, vol. 14, no. 1, pp. 147–183, Oct. 2021.
[35] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, Mar. 2019.

AIGERIM NURBAYEVA (Member, IEEE) received the B.Sc. degree in radio engineering, electronics, and telecommunications from L. N. Gumilyov Eurasian National University, Astana, Kazakhstan, in 2017, and the M.Sc. degree in robotics from Nazarbayev University, Astana, in 2020, where she is currently pursuing the Ph.D. degree in robotics engineering. Her research interests include nonlinear model predictive control, motion tracking systems, neural networks, and deep imitation/reinforcement learning.

MATTEO RUBAGOTTI (Senior Member, IEEE) received the Ph.D. degree in electronics, computer science, and electrical engineering from the University of Pavia, Pavia, Italy, in 2010. He held postdoctoral positions with the University of Trento, Trento, Italy, and IMT Institute for Advanced Studies, Lucca, Italy; and faculty positions with the University of Leicester, Leicester, U.K., and Nazarbayev University, Astana, Kazakhstan, where he is currently a Professor of robotics. His research interests include model predictive control, sliding mode control, imitation/reinforcement learning, and their applications to robotics. He is a member of the conference editorial boards of the IEEE Control System Society and of the European Control Association. He was a Subject Editor of *International Journal of Robust and Nonlinear Control*, from 2020 to 2024.