Constraint-aware learning of policies by demonstration

Article

The International Journal of Robotics Research
2018, Vol. 37(13-14) 1673–1689

DOI: 10.1177/0278364918784354
journals.sagepub.com/home/ijr

Constraint-aware learning of policies by demonstration

Leopoldo Armesto[1], João Moura[2,3], Vladimir Ivan[3], Mustafa Suphi Erden[2], Antonio Sala[4], and Sethu Vijayakumar[3]

Abstract

Many practical tasks in robotic systems, such as cleaning windows, writing, or grasping, are inherently constrained. Learning policies subject to constraints is a challenging problem. In this paper, we propose a method of constraint-aware learning *that solves the policy learning problem using redundant robots that execute a policy that is acting in the null space of a constraint. In particular, we are interested in generalizing learned null-space policies across constraints that were not known during the training. We split the combined problem of learning constraints and policies into two: first estimating the constraint, and then estimating a null-space policy using the remaining degrees of freedom. For a linear parametrization, we provide a closed-form solution of the problem. We also define a metric for comparing the similarity of estimated constraints, which is useful to pre-process the trajectories recorded in the demonstrations. We have validated our method by learning a wiping task from human demonstration on flat surfaces and reproducing it on an unknown curved surface using a force- or torque-based controller to achieve tool alignment. We show that, despite the differences between the training and validation scenarios, we learn a policy that still provides the desired wiping motion.*

Keywords

Direct policy learning, constrained motion, null-space policy, force/torque application

1. Introduction

When performing a given task in an unfamiliar environment, human beings easily adapt the skills or previously learned motions to novel situations and environments. For instance, the operator in Figure 1 wipes the front panels of the train by employing a small set of motions and skills that generalize to different train geometries and positioning (Moura and Erden, 2017). However, current robotic systems often require computationally expensive replanning and precise scans of the new environment to reproduce a given task (Pastor et al., 2011; Shiller, 2015). In addition to this, movement in complex, high degree of freedom manipulation systems often contains a high level of redundancy. The degrees of freedom available to perform a task are usually higher than what is necessary to execute that task. This allows a certain flexibility in finding an appropriate solution, so that this redundancy may be resolved according to some strategy that achieves a secondary objective, while the primary task is not affected. Such approaches to redundancy resolution are employed by human beings (Cruse and Brüwer, 1987), as well as other redundant systems, such as (humanoid) robots (D'Souza et al., 2001).

The redundancy resolution may also be interpreted as a form of hierarchical task decomposition, in which the complete space of available movement is split into a task-space component and a null-space component. For instance, one might consider a primary task, such as reaching or trajectory tracking, and a lower-priority task as a secondary goal, such as avoiding joint limits (Gienger et al., 2005), self-collisions (Sugiura et al., 2006), or kinematic singularities (Yoshikawa, 1985). This notion is particularly evident when considering motions modulated by external or environmental constraints. For instance, in the wiping task of Figure 1, the tool is constrained by the window surface; the primary task is to keep the tool aligned and in contact,

[1]Instituto de Diseño y Fabricación, Universitat Politécnica de València, Spain
[2]Institute of Sensors, Signals and Systems, Heriot-Watt University, UK
[3]Institute of Action, Perception, and Behaviour, University of Edinburgh, UK
[4]Instituto U. de Automática e Inf. Industrial, Universitat Politécnica de València, Spain

Corresponding author:
Leopoldo Armesto, Instituto de Diseño y Fabricación, Universitat Politécnica de València, C/Camino de Vera s/n, 46019, Valencia, Spain.
Email: larmesto@idf.upv.es

Fig. 1. Manual cleaning of an electric train. Willesden depot, London (2016).

and the secondary task is to provide surface coverage while maintaining a comfortable arm position. Several variants of this hierarchical approach to redundancy have been used in robotics (Khatib et al., 2008). This core concept has been applied to task sequencing (Mansard and Chaumette, 2007), task prioritization (Baerlocher and Boulic, 2004), and hierarchical quadratic programming (Escande et al., 2014; Herzog et al., 2015). These methods minimize a cost function subject to known constraints. However, they suffer from the curse of dimensionality and are typically unsuitable for real-time applications in high dimensions.

To circumvent this problem, one might attempt to learn a policy, a mapping from states to actions, that encodes behavior consistent with the set of constraints, instead of continuously calculating constraint-consistent actions. This mapping can be learned from data captured during demonstrations, consisting of human or robot motions. This approach falls under the category of imitation learning or learning by demonstration (Argall et al., 2009). One straightforward way to learn behaviors from this is through direct policy learning (DPL) (Alissandrakis et al., 2007; Calinon and Billard, 2007; Schaal et al., 2003). For instance, Gams et al. (2014) proposes to use a modification of dynamic movement primitives (Ijspeert et al., 2003) so that limits are considered at velocity and acceleration levels to tune the interaction forces of a robotic system with an object. Although DPL is well known and widely used, other approaches related to the problem of learning by demonstration involve learning a "filtered" trajectory over the demonstrations and combine operational and configuration tasks within a probabilistic framework. In particular, Calinon (2016) and Hussein et al. (2015) propose to use a Gaussian mixture model or Gaussian mixture regression to learn a parametrized trajectory with known tasks constraints, while Paraschos et al. (2017) propose that learning the prioritization of tasks can also enable the estimation of "soft" constraints and a prioritization between them.

In this paper, the problem of learning by demonstration will be understood as an action mapping in a DPL context (Alissandrakis et al., 2007; Schaal et al., 2003); however, it is well known that this method suffers from poor generalization (Argall et al., 2009) under varying unknown constraints. On the contrary, constraint-aware learning, in which the task or constraint is learned first and a null-space policy common to all tasks is learned separately using conventional methods has been shown to provide significant improvements (Armesto et al., 2017a,b; Lin et al., 2015; Towell et al., 2010). The idea behind constraint-aware methods is that the raw input data can be projected onto the null space of the task or constraint once it has been learned. We can then use other learning methods for the unconstrained policy, which is assumed to be the same across all demonstrations (Lin et al., 2015). Such an approach falls under the categorization of "hard" constraint methods (Paraschos et al., 2017). Lin et al. (2015). Lin et al. (2015) present a method for estimating the null-space projection matrix. The main drawback of their approach is that the estimation is performed by solving a non-convex optimization problem using a spherical representation. This often leads to long computation times and decreased performance (Armesto et al., 2017a). In this paper, we present a closed-form solution of this problem.

The results presented in this paper are, indeed, an extension of (Armesto et al., 2017b), in which we provide a more detailed explanation and justification of the proposed method. In particular, we consider a DPL problem, which might be difficult to learn, by making a reasonable separation into two subproblems: learning the constraint and learning the null-space policy, where both subproblems have closed-form solutions with linear parametrization. This improvement allows us to estimate null-space projection matrixes from data of different tasks, which can be used for learning a null-space policy by observing multiple projections of such a policy. Howard and Vijayakumar (2007) later use this estimate to learn the null-space policy. One of the key differences between our approach and that presented by Lin et al. (2015) is that in this paper we propose learning the constraint equation by minimizing the error in the task-space, while Lin et al. (2015) perform the minimization on an error defined in the null space. Secondly, Lin et al. (2015) impose the assumption of having access to the null space, while here we can deal with data containing both task and null-space components. In addition to this, we split the raw observation into task and null-space components in a more efficient way than the method proposed by Towell et al. (2010). Lin et al. (2017) also efficiently split the learning method into task and null-space components, but for lower-dimensional systems, unlike our method. To estimate the null-space policy, we propose to use locally weighted models (Atkeson et al., 1997); however, the method used to model such a policy is not that relevant and other well-known approaches in DPL might also be used (Calinon, 2016; Hussein et al., 2015; Ijspeert et al., 2003). We show that the learned policy can then be executed *online* by using a force-sensor-based task to align to an arbitrary surface.

The contributions of this paper are:

1. We formulate the constrained learning problem as a joint optimization over both constraint and policy parameters. Since this is a difficult problem to solve in practice, we then propose an alternate formulation, which splits this optimization into two subproblems, which we solve sequentially.
2. We formulate a closed-form solution of these subproblems by making them linear in their respective parameters.
3. We extend the theoretical work of hierarchically constrained optimization presented by Escande et al. (2014) and adapt it for the domain of constraint-aware learning from demonstration.
4. We develop a metric for computing the similarity of estimated constraints.
5. We show that our framework can employ generic models to represent the constraints and policies with no prior knowledge. We then show how application-specific knowledge can be exploited by using domain-specific regressors with physical meaning.
6. We validate our method through experiments by learning a circular wiping policy from human demonstrations on planar surfaces.
7. We define a surface alignment task using a force sensor, allowing us to perform wiping on curved surfaces based on the previously learned policy.

2. Preliminaries and problem statement

In many robotics applications, we can decompose the motion policy into a hierarchy of sub-policies. For instance, in such applications as welding, ironing, wiping, writing, etc., we can split the overall policy into a primary task of maintaining the contact with the working surface, and a secondary task of tracing a specific trajectory along the surface. Additionally, we might even specify a third task of avoiding joint limits, or minimizing deviations from a comfortable pose. In this case, a task from a higher level in the hierarchy acts as a constraint on the lower-level policies. In learning from demonstration, we assume that we have been given demonstrations of the kind of motion that we want to describe by a mathematical model.

Let us assume a system with control input $\mathbf{u}(t) \in \mathbb{R}^q$, which is subject to the following Pfaffian constraint

$$\mathbf{A}(t)\,\mathbf{u}(t) = \mathbf{b}(t) \tag{1}$$

where $\mathbf{A}(t) \in \mathbb{R}^{s\times q}$ is a full row-rank Pfaffian constraint matrix and $\mathbf{b}(t) \in \mathbb{R}^s$ is denoted as the primary-task policy. When $s < q$, there exists a null-space policy $\boldsymbol{\pi}(t) \in \mathbb{R}^q$, such that the control action in equation (1) can be obtained as

$$\mathbf{u}(t) = \mathbf{A}(t)^{\dagger}\,\mathbf{b}(t) + \mathbf{N}(t)\,\boldsymbol{\pi}(t) \tag{2}$$

where $\mathbf{N}(t) := \mathbf{I} - \mathbf{A}(t)^{\dagger}\,\mathbf{A}(t)$ is a projection matrix of the right null space of $\mathbf{A}(t)$ and † denotes the Moore–Penrose pseudoinverse. The control action can be divided into task ${}^{\text{ts}}\mathbf{u}(t)$ and null-space components ${}^{\text{ns}}\mathbf{u}(t)$

$${}^{\text{ts}}\mathbf{u}(t) := \mathbf{A}(t)^{\dagger}\,\mathbf{b}(t) \tag{3}$$

$${}^{\text{ns}}\mathbf{u}(t) := \mathbf{N}(t)\,\boldsymbol{\pi}(t) \tag{4}$$

Note that, by definition, ${}^{\text{ts}}\mathbf{u}(t) \perp {}^{\text{ns}}\mathbf{u}(t)$, and the primary-task ($\mathbf{b}(t)$) and null-space ($\boldsymbol{\pi}(t)$) policies form the full control action. Actually, as $\mathbf{N}(t)\,\mathbf{A}(t)^{\dagger} = \mathbf{0}$ and $\mathbf{N}(t)^2 = \mathbf{N}(t)$, from equation (2) we can assert that

$${}^{\text{ns}}\mathbf{u}(t) = \mathbf{N}(t)\,\mathbf{u}(t) \tag{5}$$

In a DPL context, an action mapping involves searching for optimal parameters $\mathbf{w}_u$ (Schaal et al., 2003; Alissandrakis et al., 2007)

$$\mathbf{w}_u^* := \underset{\mathbf{w}_u}{\operatorname{argmin}} \,\|\mathbf{u}(t) - \mathcal{U}(\mathbf{w}_u, t)\|^2 \tag{6}$$

where the norm is defined, given $\alpha : [0, T] \mapsto \mathbb{R}^n$, as

$$\|\alpha(t)\| := \sqrt{\frac{1}{T}\int_0^T \alpha^{\top}(t)\,\alpha(t)\;\mathrm{d}t}$$

and $\mathcal{U}(\mathbf{w}_u, t)$ is a suitably parametrized function approximator, with $\mathbf{w}_u$ the adjustable parameters, i.e. parameters from a policy model $\mathcal{U}(\mathbf{w}_u, t)$, which could be solved using any optimization procedure. In most cases, choosing a linear-in-parameter approximator will allow computationally simpler learning algorithms to be employed, as discussed later on.

Calinon and Billard (2007) proposed to solve this problem using a method they called DPL. This is a special case of our formulation that ignores the task hierarchy, pursuing a direct optimization of equation (6).

However, such a direct approach cannot "distinguish" between the actions needed to achieve the primary task (constraint satisfaction) and the actions needed to carry out the secondary task (for instance, tracing a trajectory in the constraint space). Furthermore, the primary task can often be achieved using inverse kinematics or reactively using force or visual feedback. This shifts the focus onto learning a policy that is aware of the acting constraint imposed by the primary task. This idea is implicit in the separation into task-space and null-space components in equations (3) and (4). The inability of DPL to separate such components in the learning process motivates the main goal of this paper: setting up a suitable parametrization and formulating an optimization problem such that the aforementioned constraint-aware learning is possible.

In a generic scenario, the problem is solved using a training dataset with samples of states and controls. However, the dataset might contain data coming from a mixture of different constraints, tasks, and policies. Our problem statement encompasses all these possible situations if the dataset is extended by encoding the different constraints and tasks onto pieces of time-varying information. This additional information classifies the training dataset according to the

relevant task or constraint information, to produce an estimate of the null-space policy $\boldsymbol{\pi}(t)$ consistent with such classification. Obviously, in a specific implementation, this idea may require the data to be annotated using a set of subindices referring to a specific constraint, task, or demonstration. However, with no loss of generality, we omit such indexing to avoid cluttering the notation until it is required for the implementation in the example sections. If such training data classification is not known, it might be inferred directly from the data, as shown in the last example of this paper.

3. Constraint-aware policy learning

When learning the policy $\boldsymbol{\pi}(t)$ from constrained data, we assume that matrixes $\mathbf{A}(t)$, $\mathbf{b}(t)$, and $\mathbf{N}(t)$ are not known. However, they will be given or estimated from sensor data when executing the learned policy, i.e. estimating surface parameters using computer vision and aligning the end effector with the surface. This is useful, because the learned policy can be projected onto the estimated constraint.

Given this assumption, the aim is to learn a null-space policy $\boldsymbol{\pi}(t)$ from constrained data for a given set of demonstrations, so that they can be reproduced under different constraints in real operation using sensor-based data. In that sense, for a given $\mathbf{u}(t)$, we want to estimate the constraint matrix $\mathbf{A}(t)$ and task $\mathbf{b}(t)$. Later on, by combining data from all demonstrations, we will learn $\boldsymbol{\pi}(t)$.

Learning is usually carried out by parametrized approximators, which are continuous functions. For instance, universal function approximators, such as the ones proposed by Hornik et al. (1989) provide progressively more accurate approximations as the number of parameters, neurons, etc., increases. Let us assume that the set of constraint matrix, task, and null-space policy can be parametrized as

$$\mathbf{A}(t) := \mathbf{A}(\mathbf{w}, t) \tag{7}$$

$$\mathbf{b}(t) := \mathbf{b}(\mathbf{w}, t) \tag{8}$$

$$\boldsymbol{\pi}(t) := \boldsymbol{\pi}(\mathbf{w}, \mathbf{w}_\pi, t) \tag{9}$$

where we can consider the parameters of the DPL problem in equation (6) to be $\mathbf{w}_u := (\mathbf{w}, \mathbf{w}_\pi)$ in equations (7) to (9).

Note that the actual explicit expression of equations (7) to (9) would depend on some problem-dependent information available at time t. In most cases, this will be the robot state, but there might also be other task or constraint-dependent information, as discussed in the previous section. For instance, from the robot's forward kinematics, expressed as $\mathbf{f}(\mathbf{x}(t), \mathbf{w}, t) = \mathbf{0}$, the Pfaffian constraint is derived as

$$\mathbf{A}(\mathbf{x}(t), \mathbf{w}, t) := \frac{\partial \mathbf{f}}{\partial \mathbf{x}(t)}(\mathbf{x}(t), \mathbf{w}, t)$$

and

$$\mathbf{b}(\mathbf{x}(t), \mathbf{w}, t) := \frac{\partial \mathbf{f}}{\partial t}(\mathbf{x}(t), \mathbf{w}, t)$$

with $\mathbf{u}(t) \equiv \dot{\mathbf{x}}(t)$. If $\mathbf{f}$ is time-invariant (no explicit dependence on t), then $\mathbf{b}(t) \equiv \mathbf{0}$. This particular case is, indeed, common in many situations, if we assume that the demonstration trajectories lie on a fixed "surface".

From this task or null-space parametrization, given a set of demonstrations, the DPL can be reformulated as the "*constraint-aware* policy learning" (CAPL) problem of minimizing equation (6) with the parametrization

$$\mathcal{U}(\mathbf{w}_u, t) := \mathbf{A}(\mathbf{w}, t)^\dagger \mathbf{b}(\mathbf{w}, t) + \mathbf{N}(\mathbf{w}, t)\, \boldsymbol{\pi}(\mathbf{w}, \mathbf{w}_\pi, t) \tag{10}$$

i.e., minimizing

$$J(\mathbf{w}_u) := \left\| \mathbf{u}(t) - \mathbf{A}(\mathbf{w}, t)^\dagger \mathbf{b}(\mathbf{w}, t) - \mathbf{N}(\mathbf{w}, t)\, \boldsymbol{\pi}(\mathbf{w}, \mathbf{w}_\pi, t) \right\|^2 \tag{11}$$

Of course, the DPL cost (equation (6)) may be directly optimized with a suitable parametrization. However, the assumption that the demonstrations are provided under the previously discussed constraints suggests that equation (10) might be a better parametrization than a generic "constraint-unaware" parametrization of $\mathcal{U}$. For instance, if we consider a state-dependent policy, $\mathcal{U}(\mathbf{x}, \mathbf{w}_u)$, a set of training demonstrations might have different actions for the same state under different constraints (data inconsistency); see the example in Section 6.1, where intersecting circles in different orientations illustrate such a case. In this situation, constraint-unaware DPL would try to "average" the actions for a state, whereas the constraint-aware learning method would involve correctly learning a different action for each constraint; see the details in Section 6.1.

In practice, this problem is difficult to solve because, even if the approximators (equation (7) to (9)) were linear in their parameters, the presence of pseudoinverses in equation (11) introduces a complex relation with respect to $\mathbf{w}$. However, under mild assumptions, we can reasonably approximate the original cost function by splitting it into two simpler optimization problems. Indeed, we recall that, for any orthogonal matrix Ω, we have $\|\mathbf{e}\| = \|\Omega\mathbf{e}\|$. This will inspire an orthogonal change of coordinates, yielding a factored expression of $J(\mathbf{w}_u)$ but keeping the same optimal parameter values.

Lemma 1. *If rows of* $\mathbf{A}(\mathbf{w}, t)$ *are orthonormal, i.e., if* $\mathbf{A}(\mathbf{w}, t)$ *is a semi-orthogonal matrix,*[1] *then we can express equation* (11) *as*

$$J(\mathbf{w}, \mathbf{w}_\pi) = J_1(\mathbf{w}) + J_2(\mathbf{w}, \mathbf{w}_\pi) \tag{12}$$

where

$$J_1(\mathbf{w}) := \|\mathbf{A}(\mathbf{w}, t)\, \mathbf{u}(t) - \mathbf{b}(\mathbf{w}, t)\|^2 \tag{13}$$

$$J_2(\mathbf{w}, \mathbf{w}_\pi) := \|\mathbf{N}(\mathbf{w}, t)\, (\mathbf{u}(t) - \boldsymbol{\pi}(\mathbf{w}, \mathbf{w}_\pi, t))\|^2 \tag{14}$$

The proof of this lemma can be found in Appendix A.

3.1. Sequential optimization

We can approximately solve the constraint-parametrized learning problem sequentially, first minimizing $J_1(\mathbf{w})$ by searching for optimal parameters $\mathbf{w}^*$ and then fixing these parameters while minimizing $J_2(\mathbf{w}^*, \mathbf{w}_\pi)$ over $\mathbf{w}_\pi$. The approximation comes from the fact that $\mathbf{w}$ and $\mathbf{w}_\pi$ are computed in sequence, even though J_2 also depends on $\mathbf{w}$. Thus, if the solution of the sequential minimization makes both J_1 and J_2 small (say, compared with $\|\mathbf{u}(t)\|$), then we have a good solution for the original J. However, if the value of J_2 were large, a joint optimization of equation (11) might obtain better results (albeit with the mentioned computational drawbacks). Nevertheless, this might also indicate that richer functions approximations are needed; this would certainly be the case if $J_1(\mathbf{w}^*)$ were large as, evidently, the optimal value of J will be always larger than $J_1(\mathbf{w}^*)$, since $J_2 \geq 0$.

The advantage of this approach is that $J_2(\mathbf{w}^*, \mathbf{w}_\pi)$ can be minimized using the standard least-squares method if we use a linear parametrization of $\boldsymbol{\pi}(\mathbf{w}, \mathbf{w}_\pi)$ with respect to $\mathbf{w}_\pi$. Additionally, regarding $J_1(\mathbf{w})$, parameters $\mathbf{w}^*$ can be computed in closed form using a generalized eigenvalue method, as we will show now.

3.2. Closed-form constraint estimation

In this section, we define a method for solving the minimization of equation (13). Hence, it will allow us to estimate the constraint matrix and the associated null-space projection matrix, which will be used to split the action observations into task-space and null-space components.

Note that equation (13) only depends on parameters $\mathbf{w}$. We can compute these parameters from the demonstrated data. If we express $\mathbf{A}(\mathbf{w}, t)$ and $\mathbf{b}(\mathbf{w}, t)$ as a linear combination of regressors,[2] they could be defined, at any time, as

$$\mathbf{A}(\mathbf{w}, t) := \mathbf{w}_A \boldsymbol{\Phi}_A(t) \tag{15}$$

$$\mathbf{b}(\mathbf{w}, t) := \mathbf{w}_b \boldsymbol{\Phi}_b(t) \tag{16}$$

where $\mathbf{w}_A \in \mathbb{R}^{s \times w_A}$ and $\mathbf{w}_b \in \mathbb{R}^{s \times w_b}$ are constant matrixes composed of parameters to be learned, $\mathbf{w} := (\mathbf{w}_A, \mathbf{w}_b)$. $\boldsymbol{\Phi}_A(t) \in \mathbb{R}^{w_A \times n}$, and $\boldsymbol{\Phi}_b(t) \in \mathbb{R}^{w_b}$ are some regressors that can be evaluated from information of the demonstrated motion at time t, e.g. the state $\mathbf{x}(t)$, the end-effector position computed from this state, or any other arbitrary function. This information may, for instance, describe some task information, as we discuss later.

Let us, for convenience, define

$$\begin{aligned}\boldsymbol{\Delta}(\mathbf{w}, t) &= \mathbf{A}(\mathbf{w}, t)\,\mathbf{u}(t) - \mathbf{b}(\mathbf{w}, t) \\ &= [\mathbf{w}_A \ \mathbf{w}_b] \begin{bmatrix} \boldsymbol{\Phi}_A(t)\,\mathbf{u}(t) \\ -\boldsymbol{\Phi}_b(t) \end{bmatrix} \\ &= \mathbf{w}\mathcal{H}(t)\end{aligned} \tag{17}$$

where $\mathcal{H}(t)$ comprises all the regressors (multiplied by the control inputs in the case of those from $\mathbf{A}(\mathbf{w}, t)$) into a single matrix.

We now want to compute the parameters $\mathbf{w}^*$ that will fit the regressors to the demonstrated data using a least-squares technique. The solution can be computed via the generalized eigenvalues and eigenvectors, as described in Lemma 2.

Lemma 2. *Consider the problem of minimizing $J := \boldsymbol{\theta}^\top \boldsymbol{R} \boldsymbol{\theta}$ subject to $\boldsymbol{\theta}^\top \boldsymbol{Q} \boldsymbol{\theta} = 1$, with $\boldsymbol{R}$ and $\boldsymbol{Q}$ symmetric. The optimal value of J is λ, where λ is the minimum generalized eigenvalue of the linear matrix pencil $\lambda \boldsymbol{Q} - \boldsymbol{R}$. The minimizer $\boldsymbol{\theta}$ must be a generalized eigenvector corresponding to eigenvalue λ.*

The proof of this lemma appears in Appendix B.

Now, recall that a "demonstration" will be a set of controls at different time instants from, say $t = 0$ to $t = T$. Thus, the (Euclidean) norm in $J_1(\mathbf{w})$ will actually be expressed as

$$\mathbf{w} \left(\int_0^T \mathcal{H}(t)\,\mathcal{H}^\top(t)\ \mathrm{d}t \right) \mathbf{w}^\top \tag{18}$$

subject to

$$1 = \mathbf{w} \begin{bmatrix} \boldsymbol{\Phi}_A(t)\,\boldsymbol{\Phi}_A(t)^\top & 0 \\ 0 & 0 \end{bmatrix} \mathbf{w}^\top \qquad \forall\, t \in [0, T] \tag{19}$$

However, depending on the chosen parametrization, this constraint might be difficult to satisfy. Therefore, we propose to approximate $J_1(\mathbf{w})$ with an average unit-norm constraint, i.e., enforcing

$$\mathbf{w} \begin{bmatrix} \frac{1}{T} \int_0^T \boldsymbol{\Phi}_A(t)\,\boldsymbol{\Phi}_A(t)^\top\,\mathrm{d}t & 0 \\ 0 & 0 \end{bmatrix} \mathbf{w}^\top = 1 \tag{20}$$

Now, applying Lemma 2, we can obtain the optimal values for $\mathbf{w}$ by minimizing equation (18) subject to the constraint (equation (20)). To do this, we compute $\mathbf{R} = \int_0^T \mathcal{H}(t)\,\mathcal{H}^\top(t)\,\mathrm{d}t$ from equation (18) and a rank-deficient $\mathbf{Q}$ from equation (20).

Note that, in practice, these integrals would be evaluated via a sum of the available data samples, i.e., if we have N uniformly sampled data points, we can arrange the $\mathbf{H}$ matrix as

$$\mathbf{H} := \begin{bmatrix} \boldsymbol{\Phi}_A(t_1)\,\mathbf{u}(t_1) & \boldsymbol{\Phi}_A(t_2)\,\mathbf{u}(t_2) & \dots & \boldsymbol{\Phi}_A(t_N)\,\mathbf{u}(t_N) \\ \boldsymbol{\Phi}_b(t_1) & \boldsymbol{\Phi}_b(t_2) & \dots & \boldsymbol{\Phi}_b(t_N) \end{bmatrix} \tag{21}$$

where $\mathbf{u}(t_1), \mathbf{u}(t_2), \dots, \mathbf{u}(t_N)$ are the raw observations of the action from the demonstration, with $t_1 = 0$, $t_N = T$. Then, the integral in equation (18) would be evaluated as $\frac{1}{N}\mathbf{H}\mathbf{H}^\top$ and an analogous approach would be taken for equation (20).

In theory, if several constraints are fulfilled with no error, then the (generalized) eigenvalue zero would have a multidimensional subspace of eigenvectors; thus, an orthogonal

basis of such eigenvectors would form the rows of $\mathbf{w}_A$ and $\mathbf{w}_b$. However, in practice, such a situation might not occur with noisy demonstrations so the smaller eigenvalues should be interpreted as being zero. This is a common practice in the "total-least-squares" and "principal-components" techniques discussed in Zhang (2017), to which this proposal is related.

Note that, in this noisy case, the overall result of this first phase of the learning methodology is a matrix $\mathbf{w}^*$ of parameters associated to low eigenvalues, which fulfills $\mathbf{A}(\mathbf{x}(t), \mathbf{w}^*, t)\,\mathbf{u}(t) \approx \mathbf{b}(\mathbf{x}(t), \mathbf{w}^*, t)$. Once the parameters have been learned, we can compute a modified task vector $\tilde{\mathbf{b}}(t) := \mathbf{A}(\mathbf{x}, \mathbf{w}^*, t)\,\mathbf{u}(t)$ such that the Pfaffian constraint is fulfilled *exactly*.

If $\mathbf{b}(t) = \mathbf{0}$, it can be shown (details omitted for brevity) that the problem reduces to removing the rows of $\mathbf{H}$ related to $\mathbf{\Phi}_b$ in equation (21), and computing the smallest singular values or vectors of $\mathbf{\Upsilon}^{-1}\mathbf{H}$, where $\mathbf{\Upsilon}$ is a scaling matrix, such that

$$\mathbf{\Upsilon}\mathbf{\Upsilon}^\top := \frac{1}{T}\int_0^T \mathbf{\Phi}_A(t)\,\mathbf{\Phi}_A(t)^\top \,\mathrm{d}t$$

generalizing the work of Armesto et al. (2017b). Actually, if the number of constraints is known in advance, it is easy to discriminate between situations where there is an over- or under-parametrization by computing the number of significantly smaller eigenvalues. Thus, if the number of significant eigenvalues is smaller than the expected number of constraints, it implies that there is an under-parametrization and more regressors should be added. On the contrary, if the number of significantly smaller eigenvalues is greater than the number of expected constraints, either the problem is over-parametrized or the data fulfill more constraints than originally assumed.

3.3. Learning the null-space policy

At this stage, once the minimization of $J_1(\mathbf{w})$ has been carried out and $\mathbf{w}^*$ is available, each data point in the dataset can be split into its null-space and task-space components, as

$$^{\mathrm{ns}}\mathbf{u}(\mathbf{w}^*, t) := \mathbf{N}(\mathbf{w}^*, t)\,\mathbf{u}(t) \tag{22}$$

$$^{\mathrm{ts}}\mathbf{u}(\mathbf{w}^*, t) := \mathbf{u}(t) - {}^{\mathrm{ns}}\mathbf{u}(\mathbf{w}^*, t) \tag{23}$$

This can be interpreted as an estimate of the "true" null-space and task-space components (equations (3) and (4)), if the relevant eigenvalues are close to zero. Note also that $\mathbf{A}(\mathbf{w}^*, t)^{\mathrm{ns}}\,\mathbf{u}(\mathbf{w}^*, t) = \mathbf{0}$ and $\mathbf{A}(\mathbf{w}^*, t)^{\mathrm{ts}}\,\mathbf{u}(\mathbf{w}^*, t) = \tilde{\mathbf{b}}(t)$.

Now, we can estimate the optimal value of $\mathbf{w}_\pi$ from equation (14) evaluated at $\mathbf{w}^*$

$$J_2(\mathbf{w}^*, \mathbf{w}_\pi) = \left\| {}^{\mathrm{ns}}\mathbf{u}(\mathbf{w}^*, t) - \mathbf{N}(\mathbf{w}^*, t)\,\boldsymbol{\pi}(\mathbf{w}^*, \mathbf{w}_\pi, t) \right\|^2 \tag{24}$$

Since $\boldsymbol{\pi}(\mathbf{w}^*, \mathbf{w}_\pi, t)$ is linear in parameters $\mathbf{w}_\pi$, this corresponds to a standard least-squares problem.

3.4. Learning with locally weighted models

There are several ways in which we can model the policy $\boldsymbol{\pi}(\mathbf{w}^*, \mathbf{w}_\pi, t)$. Let us consider a very generic state-feedback policy $\boldsymbol{\pi}(\mathbf{x}(t), \mathbf{w}_\pi, t)$. This simple model has also been adopted by Howard et al. (2009), Lin et al. (2015), and Towell et al. (2010). Indeed, the robot configuration $\mathbf{x}(t)$ will often encode essential features of the constraint. For instance, if we assume that all demonstrations keep a constant *orientation* of the end effector with respect to the normal vector of the constraint surface, then the normal of the surface will be represented in some features of the robot's state (we exploit this particular constraint later in this paper). We can implicitly replace dependence on $\mathbf{w}^*$ with the dependence on $\mathbf{x}(t)$ in applications where closed-loop feedback in the primary task will ensure that the position or orientation constraints are maintained during real-time operation.

Based on this idea, $\boldsymbol{\pi}(\mathbf{x}(t), \mathbf{w}_\pi, t)$ will be defined as a weighted combination of M local models, as

$$\boldsymbol{\pi}(\mathbf{x}(t), \mathbf{w}_\pi, t) := \frac{\sum_{m=1}^{M} \rho_m(\mathbf{x}(t))\,\boldsymbol{\pi}_m(\mathbf{x}(t), \mathbf{w}_{\pi,m}, t)}{\sum_{m=1}^{M} \rho_m(\mathbf{x}(t))} \tag{25}$$

where each local model m is parametrized by a corresponding $\mathbf{w}_{\pi,m}$ with $\mathbf{w}_\pi := (\mathbf{w}_{\pi,1}, \ldots, \mathbf{w}_{\pi,M})$, and $\rho_m(\mathbf{x}(t)) := \mathrm{e}^{-\frac{1}{2}(\mathbf{x}(t)-\mathbf{c}_m)^\top \mathbf{D}_m^{-1}(\mathbf{x}(t)-\mathbf{c}_m)}$ is the importance weight of each state observation according to the distance from a Gaussian receptive field, with center $\mathbf{c}_m$ and variance $\mathbf{D}_m$ (a diagonal matrix). The centers and variances of the receptive fields can be obtained from data, for instance, by running the k-means algorithm presented by Kanungo et al. (2002).

For each local model, we use a regressor vector $\mathbf{\Psi}(\mathbf{x}(t), t)$ with linear parameters, as

$$\boldsymbol{\pi}_m(\mathbf{x}(t), \mathbf{w}_{\pi,m}, t) := \mathbf{\Psi}(\mathbf{x}(t), t)\,\mathbf{w}_{\pi,m} \tag{26}$$

where $\mathbf{w}_{\pi,m}$, for $m = 1, \ldots, M$ are the weight vectors to be learned. If the receptive fields $\rho_m(\mathbf{x}(t))$ are dense enough and the constraint is time-independent, the local regressors $\mathbf{\Psi}(\mathbf{x}(t), t)$ may be chosen as simple linear functions $\mathbf{\Psi}(\mathbf{x}(t), t) := [\mathbf{x}(t)^\top\ 1]$. Indeed, the nonlinearity will be handled by mixing local linear models, as studied by Atkeson et al. (1997). The described regressor choice can now be inserted into equation (24) to form the associated least-squares problem. In a more general case, we would consider a policy $\boldsymbol{\pi}(\mathbf{x}(t), \mathbf{w}, \mathbf{w}_\pi, t)$ that depends on the parameters of the primary task. The regressors may then also depend on these parameters $\mathbf{\Psi}(\mathbf{x}(t), \mathbf{w}^*, t)$.

Actually, note that the local-model structure can, too, be used to form the regressors for $\mathbf{A}(\mathbf{x}(t), \mathbf{w}, t)$ and $\mathbf{b}(\mathbf{x}(t), \mathbf{w}, t)$, which could model a nonlinear constraint in the same way.

If we have prior information about the policy we are attempting to learn, we can choose specific regressors if we believe that they will represent the task better. This may improve accuracy and reduce the number of parameters,

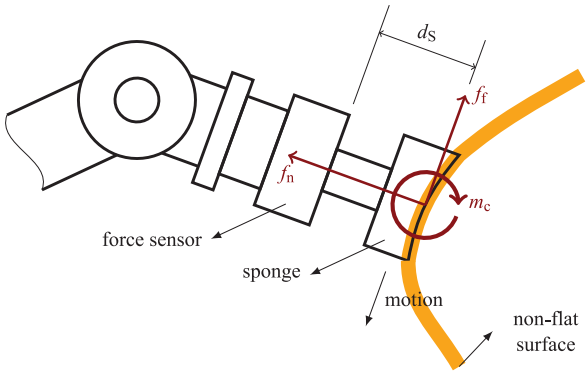

Fig. 2. Robot performing constrained task on curved surface. The robot uses a force sensor and a soft material (sponge) mounted at the end effector as a tool. The interaction of the wiping tool and the surface causes a friction force f_f, a normal force f_n, and a contact torque m_c, where the arrows indicate the direction in which the values f_x and f_z are measured. The task is to align the tool with the surface normal, by minimizing the contact torque m_c, and maintain contact by controlling the normal force f_n.

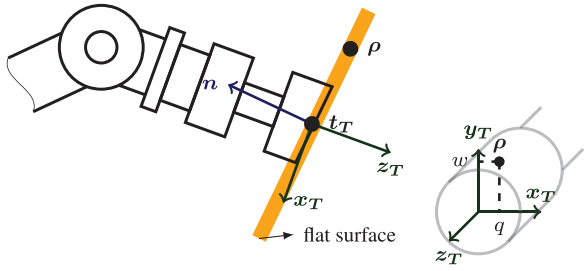

Fig. 3. Two-dimensional illustration of a robot performing the demonstrated motion on a flat surface. $\boldsymbol{\rho}$ is a point on the $\boldsymbol{x_T}$-$\boldsymbol{x_T}$ plane used as a center of the wiping motion performed in the null space of the surface alignment task.

compared with other options. For instance, a task involving tracing end-effector trajectories may use the rows of the end-effector Jacobian as regressors. The choice of suitable regressors is application-dependent. In our case study, we discuss the selection of regressors suitable for learning policies that are constrained to a planar surface.

4. Learning planar-constrained policies

Defining the appropriate set of regressors can be difficult without prior knowledge about the application. In this section, we propose to exploit the prior knowledge of the application by using Jacobians of the end effector as the main regressors for learning both the constraint and the null-space policy. This will allow us to define exact models for tasks demonstrated on planar surfaces. However, once the model has been trained, the policy can be executed on non-planar surfaces as long as we can guarantee that the end effector will stay aligned with the surface (e.g. by using force feedback). This parametrization is useful for applications where the robot is constrained by a surface on which the task is being performed, such as wiping, dusting, sweeping, scratching, or writing. In all these examples, a constraint could be defined in terms of minimizing the distance from the surface and the misalignment between the surface normal and the orientation of the robot's tool (see Figure 2). The null space of this task would be any motion of the robot's tool on the surface, i.e., with speed of movements tangential to the surface.

4.1. Learning the primary task and the constraint

Let us consider a robot with some tool at its end, whose position in three-dimensional space will be denoted $p_T(x(t))$, and a reference frame attached to the tool, denoted by the vectors $\mathbf{x}_T$, $\mathbf{y}_T$, and $\mathbf{z}_T$. We consider a training scenario where the reference surface is flat and static, as shown in Figure 3. The normal to the surface $\mathbf{n}$ does not change with time and the primary-task error can be defined using the distance of the tool from the surface and the tool's misalignment, as

$$\mathbf{e}(\mathbf{x}(t)) := \begin{bmatrix} \mathbf{n}^\top(\mathbf{t}_T(\mathbf{x}(t)) - \mathbf{p}) \\ \mathbf{n}^\top \mathbf{x}_T(\mathbf{x}(t)) \\ \mathbf{n}^\top \mathbf{y}_T(\mathbf{x}(t)) \end{bmatrix} \tag{27}$$

where $\mathbf{p}$ is any arbitrarily chosen point on the surface.

In differential kinematics (Siciliano et al., 2009), the state of a robot can be described by the joint velocity, $\dot{\mathbf{x}}(t)$, and its relation with respect to the velocity vector (error) of a task, $\dot{\mathbf{e}}(\mathbf{x}(t))$

$$\dot{\mathbf{e}}(\mathbf{x}(t)) = \mathbf{J}(\mathbf{x}(t))\,\dot{\mathbf{x}}(t) \tag{28}$$

where $\mathbf{J}(\mathbf{x}(t)) = \partial \mathbf{e}(\mathbf{x}(t))/\partial \mathbf{x}(t)$ is the analytical Jacobian of the task. We substitute $\mathbf{A}(\mathbf{x}(t)) \equiv \mathbf{J}(\mathbf{x}(t))$ and $\mathbf{u}(t) = \dot{\mathbf{x}}(t)$ in equation (1). If we assume that the demonstrator crafts $\mathbf{u}(t)$ such that it pursues some surface approximation and alignment task, with a certain target "closed-loop dynamics" if the initial error is not zero, i.e., $\dot{\mathbf{e}}(\mathbf{x}(t)) = \mathbf{g}(\mathbf{e}(\mathbf{x}(t)))$, then the associated Pfaffian constraint would be $\mathbf{A}(\mathbf{x}(t))\,\mathbf{u}(t) = \mathbf{g}(\mathbf{e}(\mathbf{x}(t)))$; hence, in this particular problem, $\mathbf{b}(\mathbf{x}) = \mathbf{g}(\mathbf{e}(\mathbf{x}(t)))$ becomes the dynamics of the implicit alignment controller, ensuring that the error converges to zero.

From equations (28) and (27), we can select the following regressors

$$\boldsymbol{\Phi}_A(\mathbf{x}(t)) := \mathbf{J}_T(\mathbf{x}(t)) \equiv \begin{bmatrix} \frac{\partial \mathbf{p_T}(\mathbf{x}(t))}{\partial \mathbf{x}(t)} \\ \frac{\partial \mathbf{x_T}(\mathbf{x}(t))}{\partial \mathbf{x}(t)} \\ \frac{\partial \mathbf{y_T}(\mathbf{x}(t))}{\partial \mathbf{x}(t)} \end{bmatrix} \tag{29}$$

$$\boldsymbol{\Phi}_b(\mathbf{x}(t)) := \begin{bmatrix} \mathbf{p_T}(\mathbf{x}(t))^\top & \mathbf{x_T}(\mathbf{x}(t))^\top & \mathbf{y_T}(\mathbf{x}(t))^\top & 1 \end{bmatrix} \tag{30}$$

where the primary-task controller will attempt to achieve a linear time-invariant stable closed loop, so the position and alignment error converge to zero. as required by the primary

task. Indeed, note that

$$\mathbf{J}(\mathbf{x}(t)) = \begin{pmatrix} \mathbf{n}^\top & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{n}^\top & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{n}^\top \end{pmatrix} \mathbf{J}_{\mathrm{T}}(\mathbf{x}(t)) \tag{31}$$

and thus the choice of $\boldsymbol{\Phi}_A(\mathbf{x}(t))$ is justified as, in an ideal scenario, the ground truth $\mathbf{A}(\mathbf{x}(t))$ can indeed be expressed as the linear-in-parameter expression (equation (31)), as long as the parameters $\mathbf{w}_A$ are allowed to adjust elements of the block-diagonal matrix in equation (31) containing the normal vector.

In theory, the regressors for $\boldsymbol{\Phi}_A(\mathbf{x}(t))$ should be correct, as long as the demonstrations are always constrained to the surface and the task is to minimize misalignment error. However, the regressors $\boldsymbol{\Phi}_b(\mathbf{x}(t))$ might be insufficient because the human operator might not have used a linear controller for alignment. Note also that measurement noise and small varying distances from the surface during the demonstration will, in general, make it impossible for the approximator errors J_1 and J_2 in equations (13) and (14) to become exactly zero. As earlier noted, from the analysis of the singular values, since the dimension of $\mathbf{e}(\mathbf{x}(t))$ is known, we can clearly identify situations where extra parametrization is needed if the number of eigenvalues that are significantly smaller than the other eigenvalues is less than three.

Regarding regressors $\boldsymbol{\Phi}_b(\mathbf{x}(t))$, in realistic applications, learning primary-task controllers is not usually of relevance since ensuring contact and alignment with the surface can be achieved via sensory feedback. This means that the recommendation for practical applications would be to provide demonstrations with an initial configuration already on the surface (or trimming the prior samples of the actual demonstration data) and assuming $\mathbf{b}(\mathbf{x}(t)) = \mathbf{0}$. This assumption has computational benefits, reducing the generalized eigenvalue computations to faster ordinary eigenvalue computations, as discussed earlier.

4.2. Learning the null-space policy

We will now propose a specialized structure for the null-space policy $\boldsymbol{\pi}(t)$, based on the Jacobian specific to the planar-constrained task under consideration. As already discussed, incorporating problem-dependent information when building the regressors instead of generic universal-approximator black-box regressors will allow us to improve accuracy and decrease the number of parameters.

Recall that the primary task attempts to align the tool orientation with the surface (constraining two degrees of freedom) and maintain the contact (constraining one more degree of freedom). This implies a task that constrains a total of three of the degrees of freedom of the robot. We can reasonably assume that any motion along the surface will be part of the null space of the primary task, with the remaining degrees of freedom available. We can now choose a suitable parametrization of the null-space policy $\boldsymbol{\pi}(\mathbf{x}(t), t)$.

Since the tool's orientation is constrained by the primary task, only the position trajectory $p_{\mathrm{T}}(\mathbf{x}(t))$ is relevant for the null-space policy. We choose an arbitrary reference frame $(\boldsymbol{\xi}_x(\mathbf{n}), \boldsymbol{\xi}_y(\mathbf{n}))$ on the surface orthogonal to the normal $\mathbf{n}$, and we define a modified tool Jacobian

$$\mathbf{J}^n(\mathbf{x}(t)) = \begin{bmatrix} \boldsymbol{\xi}_x(\mathbf{n})^\top & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\xi}_y(\mathbf{n})^\top & \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{J}_{\mathrm{T}}(\mathbf{x}(t)) \tag{32}$$

which computes the tool's speed relative to this reference frame. The estimated parameters $\mathbf{w}_A$ will not, in general, coincide with the block-diagonal expression arising from equation (31); nonetheless, if the orientation error during the demonstration is reasonably small, the surface normal would be close to the actual tool's $\mathbf{z}_{\mathrm{T}}(\mathbf{x}(t))$ vector. So we will neglect this error and propose the parametrization

$$\mathbf{J}^z(\mathbf{x}(t)) = \begin{bmatrix} \boldsymbol{\xi}_x(\mathbf{z}_{\mathrm{T}}(\mathbf{x}(t)))^\top & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\xi}_y(\mathbf{z}_{\mathrm{T}}(\mathbf{x}(t)))^\top & \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{J}_{\mathrm{T}}(\mathbf{x}(t)) \tag{33}$$

which will, basically, be coincident with equation (32) unless heavy misalignment has occurred during the demonstration. With this assumption, we will define the tool speed in the coordinate system of the plane as

$$\tilde{\boldsymbol{\kappa}}(t) := \mathbf{J}^z(\mathbf{x}(t))\,\mathbf{u}(t) \tag{34}$$

Note that these tool Jacobians consider only the tool's position and not its orientation. Additionally, $\tilde{\boldsymbol{\kappa}}(t)$ does not depend on the parameters of the policy, which means that it can be computed directly from the demonstrated data.

If the robot has five degrees of freedom, this choice of $\tilde{\boldsymbol{\kappa}}(t)$ (two degrees of freedom) and $\boldsymbol{\Phi}_A(\mathbf{x}(t))$ (three degrees of freedom) will ensure that the solution for $\mathbf{u}(t)$ is unique. However, if the robot has more than five degrees of freedom, there will be remaining redundant degrees of freedom that $\tilde{\boldsymbol{\kappa}}(t)$ will not model.

To account for this redundancy, we complement the secondary task description $\mathbf{J}^z(\mathbf{x}(t))$, with additional independent rows describing the Jacobians of quantities $\boldsymbol{\eta}(\mathbf{x}(t), t)$ related to the application by setting $\tilde{\boldsymbol{\gamma}}(t) := \dot{\boldsymbol{\eta}}(\mathbf{x}(t), t) = \mathbf{J}^\eta(\mathbf{x}(t))\,\mathbf{u}(t)$. Ideally, these additional quantities would have a physical meaning, such as tool or elbow speeds, or other posture-related velocities that a human expert can identify as relevant to the task. Given this parametrization of the policy, let us consider the expression

$$\mathbf{Q}(\mathbf{x}(t), \mathbf{w}^*, t)\,\mathbf{u}(t) := \begin{pmatrix} \mathbf{A}(\mathbf{x}(t), \mathbf{w}^*, t) \\ \mathbf{J}^z(\mathbf{x}(t)) \\ \mathbf{J}^\eta(\mathbf{x}(t)) \end{pmatrix} \mathbf{u}(t) = \begin{pmatrix} \tilde{\mathbf{b}}(t) \\ \tilde{\boldsymbol{\kappa}}(t) \\ \tilde{\boldsymbol{\gamma}}(t) \end{pmatrix} \tag{35}$$

where matrix $\mathbf{T}(\mathbf{x}(t), \mathbf{w}^*, t)$ is square and invertible.

Let us denote $\mathbf{T}^{-1}(\mathbf{x}(t), \mathbf{w}^*, t) \equiv [\mathbf{E}_b\ \mathbf{E}_\kappa\ \mathbf{E}_\gamma]$, suitably partitioning the columns of $\mathbf{T}^{-1}(\mathbf{x}(t), \mathbf{w}^*, t)$ compatible with the dimensions of $\tilde{\mathbf{b}}(t)$, $\tilde{\boldsymbol{\kappa}}(t)$, $\tilde{\boldsymbol{\gamma}}(t)$. Let us also consider suitable function approximators, where $\boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa, t)$ is a parametrized approximator of $\tilde{\boldsymbol{\kappa}}(t)$ and, similarly, $\boldsymbol{\gamma}(\mathbf{w}^*, \mathbf{w}_\gamma, t)$ is a parametrized approximator of $\tilde{\boldsymbol{\gamma}}(t)$.

Lemma 3. *The minimization of J_2 in equation* (24) *is equivalent to solving the following least-squares problem*

$$\mathbf{w}_\pi^* := \arg\min_{\mathbf{w}_\kappa, \mathbf{w}_\gamma} \left\| \left(\mathbf{E}_\kappa \mathbf{E}_\gamma\right) \begin{pmatrix} \boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa, t) \\ \boldsymbol{\gamma}(\mathbf{w}^*, \mathbf{w}_\gamma, t) \end{pmatrix} - \mathbf{N}\mathbf{u}(t) + (\mathbf{E}_b - \mathbf{A}^\dagger)\,\tilde{\mathbf{b}}(t) \right\|^2 \quad (36)$$

such that once the optimal values of $\mathbf{w}_\pi^* := (\mathbf{w}_\kappa^*, \mathbf{w}_\gamma^*)$ have been obtained, the null-space policy is defined as

$$\boldsymbol{\pi}(\mathbf{w}^*, \mathbf{w}_\pi^*, t) := (\mathbf{E}_b - \mathbf{A}^\dagger)\,\mathbf{b}(\mathbf{w}^*, t) + \mathbf{E}_\kappa \boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa^*, t) + \mathbf{E}_\gamma \boldsymbol{\gamma}(\mathbf{w}^*, \mathbf{w}_\gamma^*, t) \quad (37)$$

The proof of this lemma appears in Appendix C.

Note that, in the case where $\mathbf{b}(t)$ can be assumed to be zero, the actual expression for $\boldsymbol{\pi}(\mathbf{w}^*, \mathbf{w}_\pi^*, t)$ is

$$\boldsymbol{\pi}(\mathbf{w}^*, \mathbf{w}_\pi^*, t) := \mathbf{E}_\kappa \boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa^*, t) + \mathbf{E}_\gamma \boldsymbol{\gamma}(\mathbf{w}^*, \mathbf{w}_\gamma^*, t) \quad (38)$$

If we choose to parametrize the regressors $\boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa, t) := \boldsymbol{\Phi}_\kappa(\mathbf{w}^*, t)\,\mathbf{w}_\kappa$ and $\boldsymbol{\gamma}(\mathbf{w}^*, \mathbf{w}_\gamma, t) := \boldsymbol{\Phi}_\gamma(\mathbf{w}^*, t)\,\mathbf{w}_\gamma$ linearly, the solution to equation (36) can be solved using the standard linear least-squares method. These regressors may, too, be used to set up locally weighted models, as discussed earlier, and the learning problem will still remain a least-squares one. Recall that, in most cases, the actual parametrizations will incorporate state-dependent terms in the regressors. Let us now propose such parametrizations.

4.2.1. Suitable regressors for $\boldsymbol{\kappa}$. Recall that the components of $\boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa, t)$ have the interpretation of speeds over the constraint plane. Thus, if we know beforehand that the demonstrated curves are the result of some differential equations, this knowledge can be used to construct state-dependent regressors $\boldsymbol{\kappa}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}_\kappa, t)$.

For instance, let us assume that the robot is tracking a curve $f(\boldsymbol{\nu}) = 0$, where $\boldsymbol{\nu} := (\nu_x, \nu_y)$ are the two-dimensional coordinates of the end effector on the constraint plane. The policy will encode a motion along the curve (perpendicular to the gradient of $f(\boldsymbol{\nu})$) and a motion toward the curve (proportional to the gradient of $f(\boldsymbol{\nu})$), as

$$\dot{\boldsymbol{\nu}} = \begin{pmatrix} -\frac{\partial f}{\partial \nu_y} \\ \frac{\partial f}{\partial \nu_x} \end{pmatrix} w_\mathrm{t} - \begin{pmatrix} \frac{\partial f}{\partial \nu_x} \\ \frac{\partial f}{\partial \nu_y} \end{pmatrix} f(\boldsymbol{\nu})\, w_\mathrm{r} \quad (39)$$

with, say, constant tangential speed w_t and feedback proportional gain w_r. Then the explicit representation for f and its gradient will suggest some regressors for which there exist a "ground-truth" value for the coefficients (if the demonstration actually tracked such a curve). These regressors can be seen as a type of dynamic motion primitives for curves (Ijspeert et al., 2003).

As an example, if $f(\boldsymbol{\nu})$ were a circle $(\nu_x - c_x)^2 + (\nu_y - c_y)^2 - r^2 = 0$, the expression for $\dot{\boldsymbol{\nu}}$ would be a third polynomial in ν_x and ν_y. We therefore place the respective monomials appearing in equation (39) in the regressors for $\boldsymbol{\kappa}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}_\kappa, t)$.

4.2.2. Suitable regressors for $\boldsymbol{\gamma}$. The quantities $\boldsymbol{\gamma}(\mathbf{w}^*, \mathbf{w}_\gamma, t)$ represent redundant degrees of freedom. We will assume that there is a "comfortable" pose $\boldsymbol{\eta}^\mathrm{ref}$, such that a controller $\dot{\boldsymbol{\eta}} = \mathbf{K}_\eta(\boldsymbol{\eta}^\mathrm{ref} - \boldsymbol{\eta}(\mathbf{x}(t)))$ is approximately used in the demonstrations. We then define the regressors as the affine expression

$$\boldsymbol{\gamma}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}_\gamma, t) = \begin{bmatrix} -\mathbf{K}_\eta & \mathbf{K}_\eta & \boldsymbol{\eta}^\mathrm{ref} \end{bmatrix} \begin{pmatrix} \boldsymbol{\eta}(\mathbf{x}(t)) \\ 1 \end{pmatrix}$$

so that the matrix $\begin{bmatrix} -\mathbf{K}_\eta & \mathbf{K}_\eta & \boldsymbol{\eta}^\mathrm{ref} \end{bmatrix}$ would be the "ground-truth" parameter $\mathbf{w}_\gamma$ that we intend to learn from demonstration.

Additionally, we can exploit the locally weighted models on top of each of these regressors to model more complex policies. We have used these models in our experiments to demonstrate that they are suitable for the modeling wiping task.

5. Task generalization using force sensor

We show the utility of learning surface-constrained policies through generalization to a novel task. In many scenarios, such as in the train-cleaning application (Figure 1), it might be hard to obtain a precise model of the surface, owing to outdoor lighting conditions, different surface materials, and the surface dimensions. Thus, in practical applications, the constraint surface might not be known. Therefore, we aim to redefine the surface alignment task using, for instance, a force or torque sensor.

To guarantee the alignment between the robot end effector and the curved surface, the robot must exert some contact force on the surface and adjust the end-effector orientation to be perpendicular to that surface. As shown in Figure 2, this alignment corresponds to having the end-effector local z axis collinear with the surface normal and the end-effector local x and y axes tangent to the surface. As illustrated in Figure 2, this alignment corresponds to having minimal torque around the local x and y axes at the contact point, and having the contact force applied along the local z axis. Therefore, we can define an alignment task error as

$$\mathbf{e}_F(\mathbf{x}(t)) := \begin{bmatrix} f_\mathrm{c} - f_z \\ -m_x \\ -m_y \end{bmatrix} \quad (40)$$

where f_z is the z component of the contact force, f_c is the desired contact force, and m_x and m_z are the x and y components of the contact torque relative to the tool axis. By attaching a force or torque sensor at the tip of the end effector, we can measure the contact wrench (force and torque); by minimizing $\mathbf{e}_F$, the robot end effector will align with the contact surface.

In this scenario, the Jacobian of this error with respect to the tool frame is defined as $\mathbf{J}_F(\mathbf{x}) \in \mathbb{R}^{3\times 7}$

$$\mathbf{J}_F(\mathbf{x}(t)) = \begin{bmatrix} \mathbf{z}_\mathrm{T}^\top & \mathbf{0}^\top \\ \mathbf{0}^\top & \mathbf{x}_\mathrm{T}^\top \\ \mathbf{0}^\top & \mathbf{y}_\mathrm{T}^\top \end{bmatrix} \bar{\mathbf{J}}_\mathrm{T}(\mathbf{x}(t)) \quad (41)$$

where $\bar{\mathbf{J}}_{\mathrm{T}}(\mathbf{x}(t)) \in \mathbb{R}^{6\times 7}$ represents a standard geometric robot Jacobian.

Remark 1. *We intentionally used a "different" Jacobian* (*equation* (40)) *for real-time operation (based on sensor information). This Jacobian replaces the purely geometric choice* (*equation* (27)) *during learning in order to show the generalization capabilities to a new primary constraint or control law. Additionally, this allows us to re-project the learned planar path* $\boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa^*, t)$ *onto the constraint defined around the normal* $\mathbf{z}_{\mathrm{T}}$*, even if it is not constant.*

Our task is derived from a controller trying to achieve the closed-loop dynamics $\dot{\mathbf{e}}_F(\mathbf{x}(t)) = -\mathbf{K}_{\mathrm{P}}\mathbf{e}_F(\mathbf{x}(t))$, which can be expressed as the primary-task constraint $\mathbf{J}_f(\mathbf{x}(t))\,\mathbf{u}(t) = -\mathbf{K}_{\mathrm{P}}\mathbf{e}_F(\mathbf{x}(t))$.

It is important to remark that the error vector (equation (40)) used in wiping a non-flat surface with force feedback is different from the error used during the demonstration on the flat surfaces (equation (27)). Despite this, the learned policy $\boldsymbol{\pi}(\mathbf{w}^*, \mathbf{w}_\pi^*, t)$, and also the low-dimensional policies $\boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa^*, t)$ and $\boldsymbol{\gamma}(\mathbf{w}^*, \mathbf{w}_\gamma^*, t)$, can be projected using the new projection matrix, without affecting the primary sensor-based task. The basic idea is that we can transfer the policy to a new set of constraints $\mathbf{A}_{\mathrm{o}}(\mathbf{x}(t), t)$, $\mathbf{b}_{\mathrm{o}}(\mathbf{x}(t), t)$ at run-time. By substituting these sensor-based constraints into equation (35) we get

$$\mathbf{T}_{\mathrm{o}}(\mathbf{x}(t), t)\,\mathbf{u}(t) := \begin{pmatrix} \mathbf{b}_{\mathrm{o}}(\mathbf{x}(t), t) \\ \boldsymbol{\kappa}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}_\kappa^*, t) \\ \boldsymbol{\gamma}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}_\kappa^*, t) \end{pmatrix} \tag{42}$$

with $\mathbf{T}_{\mathrm{o}}(\mathbf{x}(t), t) := \begin{pmatrix} \mathbf{A}_{\mathrm{o}}(\mathbf{x}(t), t) \\ \mathbf{J}^{\varepsilon}(\mathbf{x}(t)) \\ \mathbf{J}^{\eta}(\mathbf{x}(t)) \end{pmatrix}$

In real-time operation, $\mathbf{T}_{\mathrm{o}}(\mathbf{x}(t), t)$ is known; therefore, the state-feedback controller will be given by

$$\mathbf{u}(\mathbf{x}(t), t) = \mathbf{T}_{\mathrm{o}}^{-1}(\mathbf{x}(t), t) \begin{pmatrix} \mathbf{b}_{\mathrm{o}}(\mathbf{x}(t), t) \\ \boldsymbol{\kappa}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}_\kappa^*, t) \\ \boldsymbol{\gamma}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}_\kappa^*, t) \end{pmatrix} \tag{43}$$

In this particular case of force-sensor feedback, we use the following regressors

$$\begin{aligned} \mathbf{A}_{\mathrm{o}}(\mathbf{x}(t), t) &:= \mathbf{J}_F(\mathbf{x}(t)) \\ \mathbf{b}_{\mathrm{o}}(\mathbf{x}(t), t) &:= -\mathbf{K}_{\mathrm{P}}\mathbf{e}_F(\mathbf{x}(t)) \end{aligned} \tag{44}$$

6. Examples

6.1. Learning a circular policy of a particle in the Cartesian space

We first introduce a simple example that contrasts our CAPL with a DPL, illustrating the problems arising from data inconsistency.

Consider a particle moving in a three-dimensional Cartesian space at constant speed—the norm of the velocity vector—and at constant distance of 1 m from the origin. When restricting the motion of this particle to a plane intersecting the origin, the resulting trajectory is a circumference centered at the origin. Our aim is to learn this circular motion for any plane intersecting the origin, provided a set of trajectories of the particle constrained to different planes. We captured two demonstration trajectories of this particle when constrained to move in two planes with an inclination of $\pm 60^\circ$ with the y axis, as shown in Figure 4.

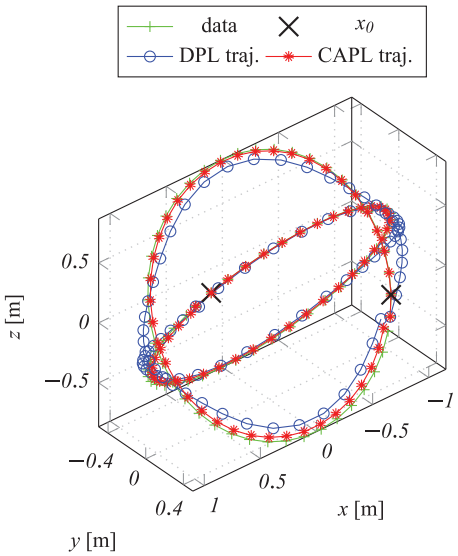

Fig. 4. Two circular trajectories of a three-dimensional particle moving in two different planes. Plot of the training data and the result of policy execution learned through DPL and CAPL, starting at the same initial position $\mathbf{x}_0$ and subject to the same planar constraints. The training circles are centered at the origin with an inclination of $\pm 60^\circ$ with respect to the y axis.

For this problem, we define the state $\mathbf{x}(t) \in \mathbb{R}^3$ as the vector of the Cartesian position of the particle and the action $\mathbf{u}(t) \in \mathbb{R}^3$ as the particle velocity. Each sub-dataset has 500 data points that correspond to a full revolution with a duration of 5 s—in Figure 4, we plot the trajectories using one fifth of the total number of training samples.

6.1.1. CAPL. Given that the constraint is independent of the state space, we define the regressors for the constraint matrix $\mathbf{A}(t)$ as a constant matrix $\boldsymbol{\Phi}_A(t) = \mathbf{I}_{3\times 3} \in \mathbb{R}^{3\times 3}$, where $\mathbf{I}_{3\times 3}$ is the identity matrix. Moreover, in each demonstration, the particle never leaves the constraint plane $\mathbf{b}(t) = \mathbf{0}$, corresponding to the case where there is only a null-space component of the actions and no task component. Given the noiseless training data, the estimated constraint parameters $\mathbf{w}_{A_1} = \begin{bmatrix} 0.0 & -0.866 & 0.5 \end{bmatrix}$ and $\mathbf{w}_{A_2} = \begin{bmatrix} 0.0 & 0.866 & 0.5 \end{bmatrix}$ exactly match the normals of the planes used in the generation of the training data. Having estimated the constraint matrix $\mathbf{A}(t)$, we can compute the estimated null-space projection matrix $\mathbf{N}(t)$ and then compute the null-space component of the training actions using equation (5) for each constraint. For the unconstrained policy, we used a linear

policy suited for this particular problem

$$\boldsymbol{\pi}(\mathbf{x}) := \begin{bmatrix} \mathbf{x}^\top & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{x}^\top & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{x}^\top \end{bmatrix} \cdot \mathbf{w}_\pi \tag{45}$$

6.1.2. DPL. For this method, we first used the same policy function (equation (45)).

Let us now compare the performance of both approaches. The DPL is biased because of the inconsistent data at intersection points. For this particular example, different actions $\mathbf{u} = \begin{bmatrix} 0 & 0.5 & 0.855 \end{bmatrix}^\top$ for the first constraint and $\mathbf{u} = \begin{bmatrix} 0 & 0.5 & -0.855 \end{bmatrix}^\top$ for the second constraint at point $\mathbf{x} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^\top$ appear in the training data. With a single regressor, the biased DPL affects all state space, producing incorrect trajectories even when trying to replicate the trained demonstrations (not shown in Figure 4, to avoid cluttering). To improve the fit at training data, we tuned 20 locally weighted regressors distributed across the training set via the k-means algorithm (Kanungo et al., 2002). However, the DPL fundamental problem at the intersection points cannot be overcome (see Figure 4, showing policy execution). Conversely, our CAPL produces the correct actions at intersection points, once projected over the constraint.

6.2. Learning a wiping policy

We have reproduced conditions outlined in Armesto et al. (2017a) to simulate a kinematic seven-degrees-of-freedom Kuka LBR IIWA R800 robot. We have generated a circular wiping motion together with a joint limit avoidance policy affecting the first, third, and seventh joints of the robot, as described in Armesto et al. (2017a). Both these ground-truth policies have an influence on the motion in the null space of the primary task that aligns the tool with the wiping of a planar surface. A single trajectory of a wiping motion with a duration of 1.5 s sampled at 0.01 s intervals has been collected with a randomly oriented planar surface placed to be perpendicular to the tool of the robot. The robot's end effector is therefore initially aligned with the surface and in contact with the surface. This ensures that alignment errors are initially close to zero. We can therefore use the singular-value decomposition approach to estimate the constraint parameters. During the data generation, we have artificially added Gaussian noise with a standard deviation of 5% of the joint's physical range on each joint's velocity, to emulate collection of noisy (non-perfect) data from a human operator.

The proposed method provides an estimate of $\mathbf{A}(\mathbf{w}^*, t)$, which generates a value of $J_1 = 0.32 \times 10^{-3}$ and $J_2 = 2.83 \times 10^{-2}$, while replacing the learned policy in the original DPL-like cost index (equation (6)) provides a cost of $J = 2.85 \times 10^{-2}$.

Figure 5 shows the (simulated) measured velocities in the tool's plane $\tilde{\boldsymbol{\kappa}}(t)$ compared with the execution of a parametrized version of $\boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}^*_\kappa, t)$, which exploits our proposed polynomial regressors for the circular trajectories. In Figure 6, we show the equivalent result used for the joint limit avoidance for the redundant joints ($\tilde{\boldsymbol{\gamma}}$ versus $\boldsymbol{\gamma}$). In addition to this, in Figure 7, we depict the simulated ground-truth policy (unknown to the learner, of course) and we overlay the trajectories computed using the estimated policy. In all cases, we can see that both the ground-truth values and measured values contain a noise as a consequence of a noisy wiping motion, while the reproduced estimated policies provide a filtered version of the correct values.

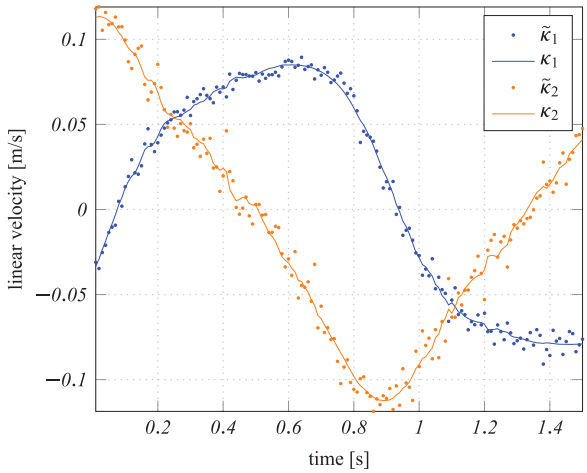

Fig. 5. Planar wiping policy estimation. Estimated planar wiping policy $\boldsymbol{\kappa}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}^*_\kappa, t)$ (continuous lines) and ground-truth planar wiping trajectory $\tilde{\boldsymbol{\kappa}}(t)$ (dots). $\tilde{\boldsymbol{\kappa}}(t)$ is noisy (to emulate non-perfect data from a human operator) and is not available during training.

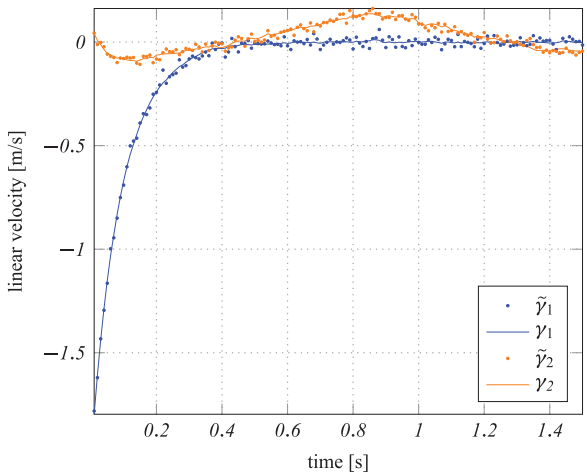

Fig. 6. Joint limit avoidance policy estimation. Estimated joint limit avoidance policy $\boldsymbol{\gamma}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}^*_\gamma, t)$ (continuous lines) and ground-truth joint limit avoidance trajectory $\tilde{\boldsymbol{\gamma}}(t)$ (dots). $\tilde{\boldsymbol{\gamma}}(t)$ is noisy (to emulate non-perfect data from a human operator) and is not available during training.

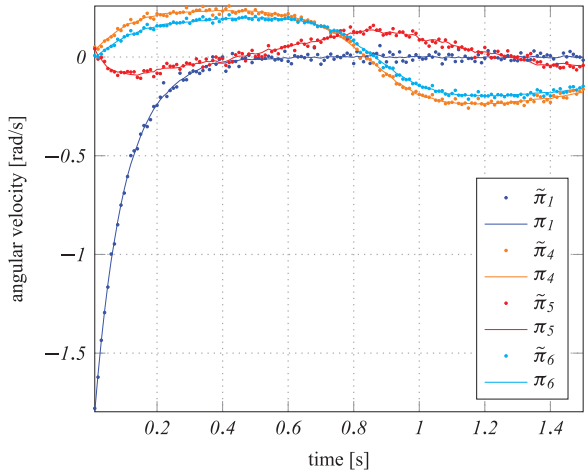

Fig. 7. Unconstrained policy estimation. Estimated unconstrained policy $\boldsymbol{\pi}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}_\pi^*, (t)$ (continuous lines) and ground-truth unconstrained trajectory $\tilde{\boldsymbol{\pi}}(t)$ (dots). $\boldsymbol{\pi}(t)$ is noisy because $\tilde{\boldsymbol{\kappa}}(t)$ and $\tilde{\boldsymbol{\gamma}}(t)$ were noisy too and is not available during training. Policies corresponding to joints 2, 3, and 7 are not shown, to avoid cluttering.

7. Experimental setup

7.1. Testing with real data and a force sensor

In our experiments, we use the seven-degrees-of-freedom Kuka LWR3 robot with an ATI industrial automation Gamma force and torque sensor attached at the end effector, as shown in Figure 8. The force sensor retrieves a six-dimensional wrench vector expressed in the sensor frame. Therefore, we compute the torque at the contact point by transforming the wrench through a distance d_S toward the contact area. We estimated this distance empirically by pressing the tool against surfaces at different angles. The robot is velocity controlled and, therefore, the minimization of the force-based main task error (equation (40)) is achieved by admittance control. This means that the robot compensates for the end-effector position and orientation according to the wrench feedback. To accommodate this motion when in contact with a rigid surface, we introduce a compliant material at the end-effector tip (such as a sponge). This added compliance introduces some dynamic behavior to the system, such as vibrations, which are suitably damped by adding a derivative component to the proportional controller suggested in the previous section.

We recorded a dataset of wiping trajectories demonstrated by a human being, as shown in Figure 9. The dataset contains 12 trajectories, each on a surface at a different orientation (four of which are shown in Figure 10). Each demonstration involved several circles with the tool of the robot, giving approximately 2000 data points[3] (using a sampling rate of 100 Hz). The demonstrated data were only minimally cropped to ensure that data contained only poses where the tool was in contact with the surface and moving along the demonstrated trajectory.

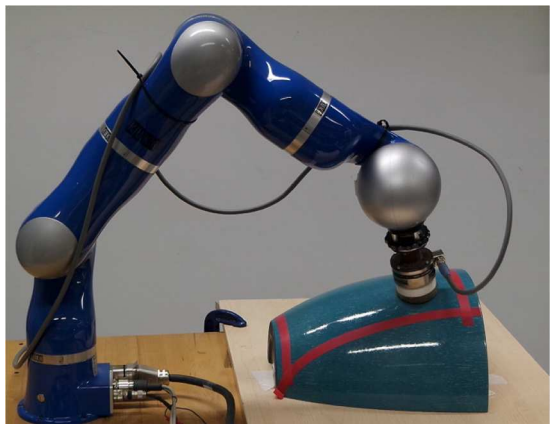

Fig. 8. Kuka LWR 3 robotic arm, equipped with a force and torque sensor, wiping a curved surface.

Fig. 9. Demonstration of a circular wiping trajectory on a flat surface. The demonstration was repeated on 12 surfaces of different orientations.

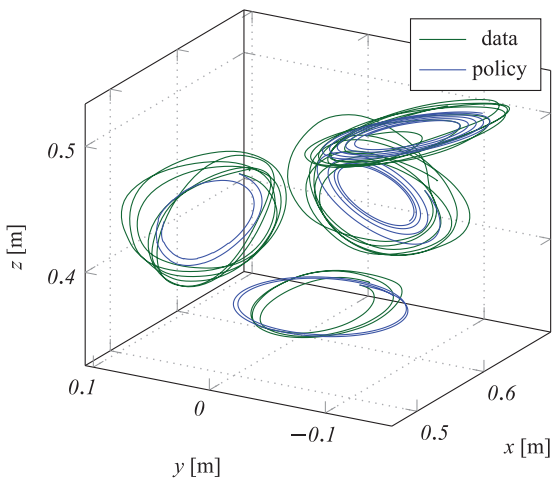

Fig. 10. Learning by demonstration. Four of the twelve wiping trajectories from human demonstration (green), and closed-loop policy validation using the respective flat surface orientation and initial position (blue).

Table 1. Costs, J, J_1, and J_2, for the four experimental demonstrations shown in Figure 10.

Demonstration	J	J_1	J_2
1	0.0206	0.81×10^{-6}	0.0199
2	0.0445	2.36×10^{-6}	0.0431
4	0.0319	2.39×10^{-6}	0.0302
7	0.0199	4.43×10^{-6}	0.0175

We used this dataset to first learn the different constraint matrixes, by parametrizing them as a linear combination of regressors and functions of state, and by applying the estimation method described in Section 3.2. The regressors for each constraint matrix are these from equation (29). For the policy $\boldsymbol{\pi}$, we used 25 locally weighted models with the same regressors used by Armesto et al. (2017b). The resulting policy was then stored and used, in a closed loop, together with the force-based surface alignment task described in the previous section.

Figure 10 shows the robot's end-effector trajectory, corresponding to the execution of the estimated null-space policy for the same constraint (surface inclination) of the demonstrations, as well as the respective end-effector position corresponding to the data. Table 1 shows the result of computing the costs J, J_1, and J_2, according to equations (11), (13), and (14), respectively. The figure shows that the locally weighted model has learned that there is a "common" circular wiping motion across the different demonstrations.

Furthermore, we have also validated the learned policy on a non-flat surface, as shown in Figure 8, demonstrating that the policy, trained from human demonstrations on flat surfaces, generalizes to both flat and curved surfaces. The resulting wiping motion is depicted in Figure 11. Note that we have demonstrated the wiping motion exclusively on flat surfaces; therefore, this shows two aspects of generalization: (I) from a surface alignment task to a force alignment task and (II) from flat surfaces to a curved surface. See Armesto et al. (2017c) for video recordings of the policy generalization to a curved surface. In many practical cases, training with flat surfaces will be easier for the demonstrator (for instance, to align the tool properly with the surface), resulting in a dataset with demonstrations in which $\mathbf{A}(\mathbf{x})\,\mathbf{u} \approx \mathbf{0}$, and consequently reducing the amount of error in the task policy.

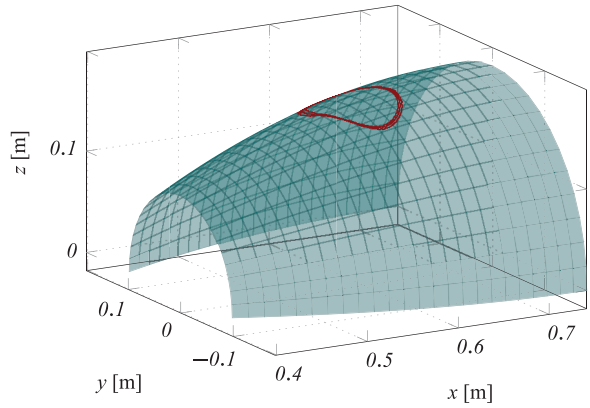

Fig. 11. A wiping policy has been trained from human demonstrations on flat surfaces (without using the force sensor); the policy generalizes to non-flat surfaces using a force-sensor-based task to align the tool dynamically.

7.2. Constraint similarity analysis

In all experiments so far, we have assumed that the demonstrator provides a set of sub-datasets $\{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_\nu\}$, each of which contains samples of pairs of raw observations, which encapsulate a sufficiently diverge set of tasks and constraints, allowing us to uncover the underlying policy common to all demonstrations that, therefore, can be generalized to different constraints. To estimate the unconstrained policy, we need demonstrations from different constraints (Howard and Vijayakumar, 2007). As a consequence, a typical dataset will contain a sequence of demonstrations, which will be classified as sub-datasets.

The aim now is to analyse how similar or distinct these sub-datasets are from one another, regarding the estimated underlying constraint, by using the same cost metric proposed for the constraint estimation. Moreover, we consider this analysis for the case of a single full dataset containing data originating from different constraints, to help us in identifying the transition regions. The experiments in this section are meant to provide an additional analysis of the training data, highlighting the difference between data obtained for an unconstrained motion or a motion subject to the same constraint and data collected under different constraints.

To compare the sub-datasets, we simply compute the cost J_1 from equation (13) for the sub-dataset l using the parameters $\hat{\mathbf{w}}_k$ estimated with the sub-dataset k, as

$$J_{1,k,l} = \left\| \mathbf{A}(\hat{\mathbf{w}}_k, t_l)\,\mathbf{u}(t_l) - \mathbf{b}(\hat{\mathbf{w}}_k, t_l) \right\|^2 \qquad (46)$$

where we index the time t_l to emphasize that the data is coming from dataset l. The value of $J_{1,k,l}$ will be low for $k = l$ and high otherwise, according to the assumption that each sub-dataset was subjected to different constraints. When different constraints intersect in some region of the space, i.e., the underlying constraints are similar to one another, this cost should be low, reflecting this constraint similarity.

For the experimental data used in the previous subsection, we manually selected the ν sub-datasets. This preprocessing step separates the full dataset into the sub-datasets. Figure 12 shows the Cartesian positions of the

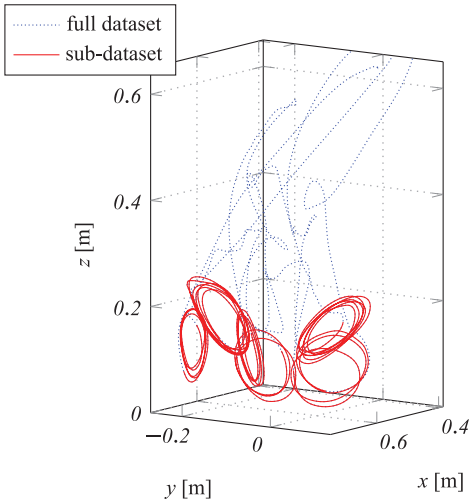

Fig. 12. Kuka lightweight robotic arm end effector. Cartesian positions for a full unseparated dataset (blue), subject to different constraints in the form of flat surface inclinations. Overlapping are the manually separated sub-datasets, showing that a full unprocessed dataset contains transition regions with data points that are discarded before the learning process.

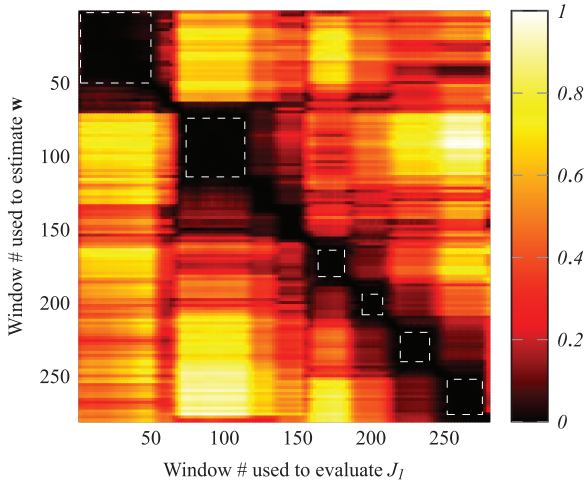

Fig. 13. Normalized $J_{1,k,l}$ cost for window l using estimated parameters from window k. Each window contains 400 consecutive data points from the full unseparated dataset, differing from the preceding window by 50 data points.

Kuka's end effector for the full dataset (blue) and, overlapping, the corresponding manually separated sub-datasets (red).

This manual separation was achieved by visually inspecting the data and selecting the initial and final indices of the data points for each sub-dataset. However, for larger full datasets this figure might become cluttered, making it difficult even to verify that some demonstrations correspond to very similar constraints. This suggests that we could use J_1 to split the sub-datasets.

Given an unprocessed dataset, we must conduct a similarity analysis for groups of data points, regardless of whether they correspond to the same constraint or not. One approach is to select a set of consecutive data points that represent a window within the full dataset. We then compute the parameters for that window k. We shift window k across the dataset by some increment smaller then the size of the window, creating a window $k+1$ (the size refers to the number of consecutive data points). If the parameters estimated in this new window produce a small J_1, then this suggests that the data covered by these two windows is subjected to the same constraint.

By repeating this process for the full dataset, we then obtain a matrix such as the one shown in Figure 13. This matrix corresponds to the data shown in Figure 12. We have empirically chosen a window size of 400 samples (corresponding to 8 s for a sampling frequency of 50 Hz) and increments of 50 samples (1 s). In Figure 13, we also overlap boxes showing the manual separation provided by the expert. There are at least two groups of windows (around indices 120 and 150) that could be confused with demonstrations, given that they produce squares of small J_1 in the matrix. Even though these two groups of samples are not true demonstrations, the cost J_1 indicates that the data belonging to those two groups are consistent with some constraint, which is sufficiently well modeled by the chosen combination of regressors. For instance, if those samples correspond to a moment in time where the robot was static while changing the flat table orientation between demonstrations, then it makes sense to say that those data points are consistent with the same constraint, e.g. the same configuration of the robot. This metric can be further combined with other application specific metrics. Data points where the robot is static may be removed using pre-processing if necessary. Alternatively, a tactile sensor could be used to detect when the end tool is in contact with the surface, etc.

8. Conclusion

This paper presents a new method for learning, from demonstration, policies that lie in the null space of a primary task, i.e. subject to some constraint. We introduce the term "constraint-aware policy learning" as a reformulation of the direct policy learning method, where the policy appropriately parametrizes the constraint. Additionally, we discuss the conditions for which this "constraint-aware policy learning" can be split into two optimization problems: constraint estimation preceded by null-space policy estimation.

The main advantage of this approach, compared with classic direct policy learning, is its ability to learn a policy consistent with the constraint. To demonstrate this point, we used different tasks and constraints in our experimental demonstration with the real Kuka lightweight arm. In this case, while recording the training data, the human demonstrator provides the task, whereas in the validation stage we use a force-based task to adapt and align the tool to an unknown surface.

While the null-space policy can be parametrized with locally weighted models, as discussed in this paper, or any other more generic functions, in the example of learning a

wiping motion we choose to take advantage of our knowledge of this specific task by incorporating more specialized regressors. This decreases the number of parameters that the algorithm must learn, decreasing the required number of demonstrations. Certainly, a clever choice of regressors can—as in our case—greatly improve the results or even turn the learning exercise into a trivial problem. However, what this framework provides is a way of encapsulating all the specifics and domain knowledge in the chosen regressors, and not in the learning algorithm itself.

Moreover, we consider the case of a null-space policy that, instead of having the full dimension of the system actions, can be decomposed, by assumption, into a set of lower-dimensional policies. For this case, we propose an alternative reformulation for estimating these lower-dimensional policies, provided the respective regressors are supplied.

However, to learn a generalizable null-space policy, we must somehow guarantee that the training datasets provide enough variability of constraints. We provide a means of comparing the datasets, regarding their underlying constraint, by using the same metric used in the constraint estimation. This involves building a similarity matrix by computing the estimation residual of a sub-dataset, using the estimated parameters from the other sub-datasets. Furthermore, besides allowing us to identify similar constraints between different sub-datasets, this similarity matrix allows us to identify different constraints within the same dataset, by running the same metric but over different windows of data. This can be a valuable tool for helping to identify the beginning and end of a demonstration.

In our future work, we intend to exploit more challenging application domains, as well as learning constrained tasks for dynamic systems. We would also like to integrate the constraint-aware learning framework with other policy learning methods to guarantee some desired properties for the null-space policy.

Funding

The author(s) disclosed receipt of the following financial support for the research, auth/orship, and/or publication of this article: This work was supported by the Spanish Ministry of Economy and the European Union (grant number DPI2016-81002-R (AEI/FEDER, UE)), the European Union Horizon 2020, as part of the project Memory of Motion - MEMMO (project ID 780684), and the Engineering and Physical Sciences Research Council, UK, as part of the Robotics and AI hub in Future AI and Robotics for Space - FAIR-SPACE (grant number EP/R026092/1), and as part of the Centre for Doctoral Training in Robotics and Autonomous Systems at Heriot-Watt University and the University of Edinburgh (grant numbers EP/L016834/1 and EP/J015040/1).

Notes

1. As the constraint $\mathbf{A}(\mathbf{w},t)\,\mathbf{u}(t) = \mathbf{b}(\mathbf{w},t)$ can be equivalently expressed as $\mathbf{R}^\top(\mathbf{w},t)\,\mathbf{A}(\mathbf{w},t)\,\mathbf{u}(t) = \mathbf{R}^\top(\mathbf{w},t)\,\mathbf{b}(\mathbf{w},t)$ for any invertible matrix $\mathbf{R}(\mathbf{w},t)$, there is no loss of generality in assuming that the Pfaffian constraint (equation (1)) in the problem statement involves matrix $\mathbf{A}(\mathbf{w},t)$ having orthonormal rows. In a particular case, $\mathbf{R}(\mathbf{w},t)$ can be obtained from the (economy size) *QR* decomposition of $\mathbf{A}(\mathbf{w},t)^\top$: $\mathbf{A}(\mathbf{w},t)^\top = \mathbf{Q}(\mathbf{w},t)\,\mathbf{R}(\mathbf{w},t)$, with $\mathbf{Q}(\mathbf{w},t)$ orthogonal, results in $\mathbf{R}(\mathbf{w},t)^\top\,\mathbf{A}(\mathbf{w},t) = \mathbf{Q}(\mathbf{w},t)^\top$.
2. With the appropriate regressors, any complex function can be expressed as a linear combination of parameters, such as recursive best first searches, neural networks (Haykin, 1998), or locally weighted linear models (Schaal and Atkeson, 1998).
3. The data included the joint state $\mathbf{x}(t)$ and the joint commands $\mathbf{u}(t)$, obtained by differentiating the joint states.

References

Alissandrakis A, Nehaniv CL, and Dautenhahn K (2007) Correspondence mapping induced state and action metrics for robotic imitation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37(2): 299–307.

Argall BD, Chernova S, Veloso M, et al. (2009) A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5): 469–483.

Armesto L, Bosga J, Ivan V, et al. (2017a) Efficient learning of constraints and generic null space policies. In: *International conference on robotics and automation*, Singapore, 29 May–3 June 2017, pp. 1520–1526. Piscataway, NJ: IEEE.

Armesto L, Ivan V, Moura J, et al. (2017b) Learning constrained generalizable policies by demonstration. In: *Robotics: Science and systems* (eds. N Amato, S Srinivasa, Nora A, et al.), Cambridge, MA, USA, 12–16 July 2017. Cambridge, MA: Robotics: Science and System XIII.

Armesto L, Ivan V, Moura J, et al. (2017c) Efficient learning of constraints and generic null space policies. Available at: https://www.youtube.com/watch?v=2n0yW1yI524 (accessed 19th June 2018).

Atkeson CG, Moore AW, and Schaal S (1997) Locally weighted learning for control. *Artificial Intelligence Review* 11(1–5): 75–113.

Baerlocher P and Boulic R (2004) An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer* 20(6): 402–417.

Calinon S (2016) A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics* 9(1): 1–29.

Calinon S and Billard A (2007) Incremental learning of gestures by imitation in a humanoid robot. In: *ACM/IEEE international conference on human–robot interaction*, Arlington, VA, USA, 10–12 March 2007, pp. 255–262. New York, NY: ACM.

Cruse H and Brüwer M (1987) The human arm as a redundant manipulator: The control of path and joint angles. *Biological Cybernetics* 57(1–2): 137–144.

D'Souza A, Vijayakumar S, and Schaal S (2001) Learning inverse kinematics. In: *IEEE/RSJ international conference on intelligent robots and systems*, Maui, HI, USA, 29 October–3 November 2001, vol. 1., pp. 298–303. Piscataway, NJ: IEEE.

Escande A, Mansard N, and Wieber PB (2014) Hierarchical quadratic programming: Fast online humanoid–robot motion generation. *The International Journal of Robotics Research* 33(7): 1006–1028.

Gams A, Nemec B, Ijspeert AJ, et al. (2014) Coupling movement primitives: Interaction with the environment and bimanual tasks. *IEEE Transactions on Robotics* 30(4): 816–830.

Gienger M, Janssen H, and Goerick C (2005) Task-oriented whole body motion for humanoid robots. In: *IEEE-RAS international conference on humanoid robots*, Tsukuba, Japan, 5 December 2005, pp. 238–244. Piscataway, NJ: IEEE.

Haykin SS (1998) *Neural Networks: A Comprehensive Foundation* (2nd edition). Upper Saddle River, NJ: Prentice Hall PTR.

Herzog A, Rotella N, Mason S, et al. (2015) Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid. *Autonomous Robots* 40(3): 473–491.

Hornik K, Stinchcombe M, and White H (1989) Multilayer feedforward networks are universal approximators. *Neural Networks* 2(5): 359–366.

Howard M and Vijayakumar S (2007) Reconstructing null-space policies subject to dynamic task constraints in redundant manipulators. In: *Workshop on robotics and mathematics (RoboMat)*, Coimbra, Portugal, 17–19 September 2007. Edinburgh, UK: Informatics Report Series EDI-INF-RR-1200.

Howard M, Klanke S, Gienger M, et al. (2009) A novel method for learning policies from variable constraint data. *Autonomous Robots* 27(2): 105–121.

Hussein M, Mohammed Y, and Ali SA (2015) Learning from demonstration using variational Bayesian inference. In: Ali M, Kwon Y, Lee CH, et al. (eds.) *Current Approaches in Applied Artificial Intelligence. IEA/AIE 2015* (*Lecture Notes in Computer Science*, vol. 9101). Cham: Springer, pp. 371–381.

Ijspeert A, Nakanishi J, and Schaal S (2003) Learning attractor landscapes for learning motor primitives. In: *15th international conference on neural information processing systems* (eds. S Becker, S Thrun, and K Obermayer), Vancouver, British Columbia, Canada 8–13 December 2003, pp. 1547–1554. Cambridge, MA: MIT Press.

Kanungo T, Mount DM, Netanyahu NS, et al. (2002) An efficient *k*-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24(7): 881–892.

Khatib O, Sentis L, and Park JH (2008) A unified framework for whole-body humanoid robot control with multiple constraints and contacts. In: Bruyninckx H, Přeučil L, and Kulich M (eds.) *European Robotics Symposium 2008* (*Springer Tracts in Advanced Robotics*, vol. 44). Berlin: Springer, pp. 303–312.

Lin HC, Howard M, and Vijayakumar S (2015) Learning null space projections. In: *IEEE international conference on robotics and automation*, Seattle, WA, USA, 26–30 May 2015, pp. 2613–2619. Piscataway, NJ: IEEE.

Lin HC, Ray P, and Howard M (2017) Learning task constraints in operational space formulation. In: *IEEE international conference on robotics and automation*, Singapore, 29 May–3 June 2017, pp. 309–315. Piscataway, NJ: IEEE.

Mansard N and Chaumette F (2007) Task sequencing for high-level sensor-based control. *IEEE Transactions on Robotics* 23(1): 60–72.

Moura J and Erden MS (2017) Formulation of a control and path planning approach for a cab front cleaning robot. In: *Procedia CIRP* 59: 67–71.

Paraschos A, Lioutikov R, Peters J, et al. (2017) Probabilistic prioritization of movement primitives. *IEEE Robotics and Automation Letters* 2(4): 2294–2301.

Pastor P, Righetti L, Kalakrishnan M, et al. (2011) Online movement adaptation based on previous sensor experiences. In: *IEEE/RSJ international conference on intelligent robots and systems*, San Francisco, CA, USA, 25–30 September 2011, pp. 365–371. Piscataway, NJ: IEEE.

Schaal S and Atkeson CG (1998) Constructive incremental learning from only local information. *Neural Computation* 10(8): 2047–2084.

Schaal S, Ijspeert A, and Billard A (2003) Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society B: Biological Sciences* 358(1431): 537–547.

Shiller Z (2015) Off-line and on-line trajectory planning. In: G Carbone and F Gomez-Bravo (eds.), *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches* (*Mechanisms and Machine Science*, vol. 29). Cham: Springer International Publishing, pp. 29–62.

Siciliano B, Sciavicco L, Villani L, et al. (2009) *Differential Kinematics and Statics*. London: Springer, pp. 105–160.

Sugiura H, Gienger M, Janssen H, et al. (2006) Real-time self collision avoidance for humanoids by means of nullspace criteria and task intervals. In: *IEEE-RAS international conference on humanoid robots*, Genova, Italy, 4–6 December 2006, pp. 575–580. Piscataway, NJ: IEEE.

Towell C, Howard M, and Vijayakumar S (2010) Learning nullspace policies. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, Taipei, Taiwan, 18–22 October 2010, pp. 241–248. Piscataway, NJ: IEEE.

Yoshikawa T (1985) Manipulability of robotic mechanisms. *The International Journal of Robotics Research* 4(2): 3–9.

Zhang XD (2017) *Matrix Analysis and Applications*. Cambridge: Cambridge University Press.

Appendix A Proof of Lemma 1

Let us define the the transformation matrix

$$\Omega(\mathbf{w},t) := \begin{bmatrix} \mathbf{A}(\mathbf{w},t) \\ \text{null}[\mathbf{A}(\mathbf{w},t)]^\top \end{bmatrix}$$

This matrix is orthogonal because the matrix $\text{null}[\mathbf{A}(\mathbf{w},t)]^\top$ is an orthogonal basis of the left null space of $\mathbf{A}(\mathbf{w},t)^\top$. i.e., the set of row vectors $\mathcal{N}_A(\mathbf{w},t) := \{\boldsymbol{\psi} \in \mathbb{R}^{1\times q} : \psi\mathbf{A}(\mathbf{w},t)^\top = \mathbf{0}\}$.

Indeed, by orthogonality $\mathbf{A}(\mathbf{w},t)^\dagger = \mathbf{A}(\mathbf{w},t)^\top$, so $\mathbf{A}(\mathbf{w},t)\,\mathbf{A}(\mathbf{w},t)^\top\mathbf{b}(\mathbf{w},t) = \mathbf{b}(\mathbf{w},t)$, yielding $J_1(\mathbf{w})$. Also, $\mathbf{A}(\mathbf{w},t)\,\mathbf{N}(\mathbf{w},t) = \mathbf{0}$ because $\mathbf{N}(\mathbf{w})$ projects on the right null space of $\mathbf{A}(\mathbf{w},t)$. Finally, $J_2(\mathbf{w},\mathbf{w}_\pi)$ is the norm of a column vector belonging to the right null space of $\mathbf{A}(\mathbf{w},t)$ (and this null space is actually the transpose of $\mathcal{N}_A(\mathbf{w},t)$). This norm is equal to the sum of the squares of its coordinates in an orthogonal basis, and these coordinates are what the left multiplication by $\text{null}[\mathbf{A}(\mathbf{w},t)]^\top$ does.

Appendix B Proof of Lemma 2

To prove Lemma 2, we introduce a Lagrange multiplier λ and construct the following augmented cost function

$$\mathbf{L} = \boldsymbol{\theta}^\top\mathbf{R}\boldsymbol{\theta} + \lambda(\boldsymbol{\theta}^\top\mathbf{Q}\boldsymbol{\theta} - 1) \tag{47}$$

which we differentiate to get

$$0 = \frac{\partial\mathbf{L}}{\partial\boldsymbol{\theta}} = 2\mathbf{R}\boldsymbol{\theta} - 2\lambda\mathbf{Q}\boldsymbol{\theta} \;\Rightarrow\; \mathbf{R}\boldsymbol{\theta} = \lambda\mathbf{Q}\boldsymbol{\theta} \tag{48}$$

Thus, λ must be a generalized eigenvalue and $\boldsymbol{\theta}$ a generalized eigenvector of the matrix pencil $\lambda\mathbf{Q} - \mathbf{R}$, i.e., λ is a solution of $\det(\mathbf{R} - \lambda\mathbf{Q}) = \mathbf{0}$, and $\boldsymbol{\theta}$ is a vector in $\text{null}(\mathbf{R} - \lambda\mathbf{Q})$.

The resulting eigenvectors for different eigenvalues are $\mathbf{Q}$-orthogonal: if $\mathbf{Rx} = \lambda\mathbf{Qx}$ and $\mathbf{Ry} = \mu\mathbf{Qy}$, we have

$$\lambda(\mathbf{Qx})^\top \mathbf{y} = \lambda\mathbf{x}^\top\mathbf{Qy} = \mathbf{x}^\top\mathbf{Ry} = \mu\mathbf{x}^\top\mathbf{Qy} \tag{49}$$

Thus, $(\lambda - \mu)\mathbf{x}^\top\mathbf{Qy} = 0$ so, if $\lambda \neq \mu$, also implies $\mathbf{x}^\top\mathbf{Qy} = 0$. Of course, if an eigenvalue gives rise to a multidimensional subspace of eigenvectors, we can always build a $\mathbf{Q}$-orthogonal basis of it.

Appendix C Proof of Lemma 3

From equation (35), control actions can be expressed as

$$\mathbf{u}(t) = \mathbf{E}_b\tilde{\mathbf{b}}(t) + \mathbf{E}_\kappa\tilde{\boldsymbol{\kappa}}(t) + \mathbf{E}_\gamma\tilde{\boldsymbol{\gamma}}(t) \tag{50}$$

Now, we can express the estimated null-space component (equation (22)) as

$$\begin{aligned}{}^{\text{ns}}\mathbf{u}(\mathbf{w}^*, t) &= \mathbf{u}(t) - \mathbf{A}^\dagger\tilde{\mathbf{b}}(t) \\ &= (\mathbf{E}_b - \mathbf{A}^\dagger)\tilde{\mathbf{b}}(t) + \mathbf{E}_\kappa\tilde{\boldsymbol{\kappa}}(t) + \mathbf{E}_\gamma\tilde{\boldsymbol{\gamma}}(t)\end{aligned} \tag{51}$$

Indeed, we can prove that the right-hand side of equation (51) lies in the null space of the primary-task matrix $\mathbf{A}(\mathbf{x}(t), \mathbf{w}^*, t)$, i.e., that the following identity is verified

$$\mathbf{A}\left((\mathbf{E}_b - \mathbf{A}^\dagger)\tilde{\mathbf{b}}(t) + \mathbf{E}_\kappa\tilde{\boldsymbol{\kappa}}(t) + \mathbf{E}_\gamma\tilde{\boldsymbol{\gamma}}(t)\right) = \mathbf{0} \tag{52}$$

because $\mathbf{AT}^{-1} = \begin{bmatrix}\mathbf{I} & \mathbf{0} & \mathbf{0}\end{bmatrix}$, $\mathbf{J}^z\mathbf{T}^{-1} = \begin{bmatrix}\mathbf{0} & \mathbf{I} & \mathbf{0}\end{bmatrix}$ and $\mathbf{J}^\gamma\mathbf{T}^{-1} = \begin{bmatrix}\mathbf{0} & \mathbf{0} & \mathbf{I}\end{bmatrix}$; hence, we can assert that $\mathbf{AE}_b = \mathbf{I}$ and therefore $\mathbf{A}(\mathbf{E}_b - \mathbf{A}^\dagger) = \mathbf{0}$, $\mathbf{AE}_\kappa = \mathbf{0}$, and $\mathbf{AE}_\gamma = \mathbf{0}$.

Thus, to build the null-space terms in equation (24), inspired in equation (51), we can define the parameters $\mathbf{w}_\pi := (\mathbf{w}_\kappa, \mathbf{w}_\gamma)$ and set up the expression

$$\begin{aligned}\boldsymbol{\pi}(\mathbf{w}^*, \mathbf{w}_\pi, t) :=& (\mathbf{E}_b - \mathbf{A}^\dagger)\tilde{\mathbf{b}}(t) \\ &+ \mathbf{E}_\kappa\boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa, t) + \mathbf{E}_\gamma\boldsymbol{\gamma}(\mathbf{w}^*, \mathbf{w}_\gamma, t)\end{aligned} \tag{53}$$

Note that $\tilde{b}(t)$ is used to ensure that equation (1) is satisfied with equality. Indeed, note that $\mathbf{N}(\mathbf{x}(t), \mathbf{w}^*, t)\boldsymbol{\pi}(\mathbf{w}^*, \mathbf{w}_\pi, t) = \boldsymbol{\pi}(\mathbf{w}^*, \mathbf{w}_\pi, t)$. Thus, we have proven that minimizing J_2 in equation (24) is equivalent to solving the following equation in a least-squares sense

$$\begin{aligned}\mathbf{0} =& \\ \mathbf{Nu}(t) - \begin{pmatrix}\mathbf{E}_\kappa & \mathbf{E}_\gamma\end{pmatrix}&\begin{pmatrix}\boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa, t) \\ \boldsymbol{\gamma}(\mathbf{w}^*, \mathbf{w}_\gamma, t)\end{pmatrix} - (\mathbf{E}_b - \mathbf{A}^\dagger)\tilde{\mathbf{b}}(t)\end{aligned} \tag{54}$$

Once the optimal values for $\mathbf{w}_\kappa^*$ and $\mathbf{w}_\gamma^*$ have been obtained, incorporating the parameters $\mathbf{w}^*$ of the identified constraints from J_1, we can express $\mathcal{U}(\mathbf{w}_u^*)$ as in equation (10) from the identity arising from the matrix inversion, analogous to equation (50) but using the estimated parameter values

$$\begin{aligned}\mathcal{U}(\mathbf{w}_u^*, t) =& \\ \mathbf{E}_b\mathbf{b}(\mathbf{w}^*, t) &+ \mathbf{E}_\kappa\boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa^*, t) + \mathbf{E}_\gamma\boldsymbol{\gamma}(\mathbf{w}^*, \mathbf{w}_\gamma^*, t)\end{aligned} \tag{55}$$

with $\mathbf{w}_u^* = (\mathbf{w}^*, \mathbf{w}_\kappa^*, \mathbf{w}_\gamma^*)$ and defining the estimated null-space policy as

$$\begin{aligned}\boldsymbol{\pi}(\mathbf{w}^*, \mathbf{w}_\pi^*, t) :=& (\mathbf{E}_b - \mathbf{A}^\dagger)\mathbf{b}(\mathbf{w}^*, t) \\ &+ \mathbf{E}_\kappa\boldsymbol{\kappa}(\mathbf{w}^*, \mathbf{w}_\kappa^*, t) + \mathbf{E}_\gamma\boldsymbol{\gamma}(\mathbf{w}^*, \mathbf{w}_\gamma^*, t)\end{aligned} \tag{56}$$