Constraint-aware learning of policies by demonstration

Citation for published version:

Digital Object Identifier (DOI):
10.1177/0278364918784354

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
International Journal of Robotics Research

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Constraint-aware Learning of Policies by Demonstration

Leopoldo Armesto\textsuperscript{1} and João Moura\textsuperscript{2,3} and Vladimir Ivan\textsuperscript{3} and Mustafa Suphi Erden\textsuperscript{2} and Antonio Sala\textsuperscript{4} and Sethu Vijayakumar\textsuperscript{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 constraint-aware learning method that solves the policy learning problem on redundant robots which execute a policy that is acting in the null-space of a constraint. In particular, we are interested in generalizing learnt null-space policies across constraints that were not known during the training. We split the combined problem of learning constraints and policies into: 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/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.

Introduction
When performing a given task in an unfamiliar environment, humans easily adapt the skills or previously learned motions to novel situations and environments. For instance, the operator in Fig. 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 Shiller (2015); Pastor et al. (2011). 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 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 humans Cruse and Brüwer (1987) as well as other redundant systems such as (humanoid) robots D’Souza et al. (2001).

The redundancy resolution may be also 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 to be a secondary goal such as avoiding joint limits Gienger et al. (2005), self-collisions Sugiiura 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 Fig. 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 Politècnica de València, C/Camino de Vera s/n, 46019, Valencia, Spain
2 Institute of Sensors, Signals and Systems, Heriot-Watt University, Mountbatten Building, EH14 4AS, Edinburgh, UK
3 Institute of Action, Perception, and Behaviour, University of Edinburgh, Crichton Street 10, EH8 9AB, Edinburgh, UK
4 Instituto U. de Automática e Inf. Industrial, Universitat Politècnica de València, C/Camino de Vera s/n, 46019, Valencia, Spain

Corresponding author:
Leopoldo Armesto
Email: larmesto@idf.upv.es

Figure 1. Manual cleaning of an electric train. Willesden depot, London (2016)
and the secondary task is to provide surface coverage while maintaining 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 prioritisation 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 behaviour 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 behaviours from this is through direct policy learning (DPL) Schaal et al. (2003); Alissandrakis et al. (2007); Calinon and Billard (2007). 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 learn a “filtered” trajectory over the demonstrations and combine operational and configuration tasks within a probabilistic framework. In particular, Hussein et al. (2015); Calinon (2016) propose to use GMM/GMR to learn a parametrized trajectory with known tasks constraints, while in Paraschos et al. (2017), authors propose to learn the prioritization of tasks can also be used to estimate “soft” constraints and a prioritization among them.

In this paper the problem of learning by demonstration will be understood as an action mapping in a DPL context Schaal et al. (2003); Alissandrakis et al. (2007), however it is well known that it suffers from poor generalisation Argall et al. (2009) under varying unknown constraints. On the contrary, constraint-aware learning, in which the task/constraint is learnt first and a null-space policy common for all tasks is learnt separately using conventional methods has been shown to provide a significant improvements Towell et al. (2010); Lin et al. (2015). The idea behind constraint-aware methods, is that the raw input data can be projected onto the null-space of the task/constraint once it’s been learnt. We can then use other learning methods for the unconstrained policy, which is assumed to be the same across all demonstrations as in Lin et al. (2015). Such approach falls under the categorization of “hard” constraint methods Paraschos et al. (2017). In Lin et al. (2015), the authors presented a method for estimating the null-space projection matrix. The main drawback of that 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. (2017c), where we provide more detailed explanation and justification of the proposed method. In particular, we relate a DPL problem, which might be difficult to learn, with a reasonable separation into two subproblems: learning the constraint and learning the null-space policy, where both of them have closed-form solutions with linear parameterization. This improvement allows us to estimate null-space projection matrices from data of different tasks which can be used for learning a null-space policy by observing multiple projections of such policy. Howard and Vijayakumar (2007a) later use that estimation to learn the null-space policy. One of the key differences between our approach and the one presented in Lin et al. (2015) is that in this paper we propose learning the constraint equation by minimizing the error in the task-space, while in Lin et al. (2015) the minimization was performed on an error defined in the null-space. Secondly, in Lin et al. (2015) the assumption of having access to the null-space was imposed, 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 in Towell et al. (2010). Lin et al. (2017) also efficiently splits the learning method into task and null-space components, but does it for lower dimensional systems unlike our method. In order to estimate the null-space policy, we propose to use locally weighted models Akeson et al. (1997), however, the method used to model such policy is not that relevant and other well-known approaches in DPL might be used too Ijspeert et al. (2003); Hussein et al. (2015); Calinon (2016). We show that the learnt policy can be then 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 optimisation 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 optimisation into two sub-problems that 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 optimisation presented in 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 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 which allows us to perform wiping on curved surfaces based on the previously learned policy.
Preliminaries and Problem Statement

In many robotics applications, we can decompose the motion policy into a hierarchy of sub-policies. For instance, in applications such as welding, ironing, wiping, writing, etc., we can split the overall policy in 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 this kind of motion that we want to describe by a mathematical model.

Let’s assume a system with control input \( u(t) \in \mathbb{R}^q \) is subject to the following Pfaffian constraint:

\[
A(t)u(t) = b(t),
\]

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

\[
u(t) = A(t)^\dagger b(t) + N(t)\pi(t),
\]

where \( N(t) := I - A(t)^\dagger A(t) \) is a projection matrix of the right null-space of \( A(t) \) and \( \dagger \) denotes Moore-Penrose pseudoinverse. From the above, the control action can be divided into task \( t_s u(t) \) and null-space components \( n_s u(t) \):

\[
t_s u(t) := A(t)^\dagger b(t)
\]

\[
n_s u(t) := N(t)\pi(t).
\]

Note that by definition \( t_s u(t) \perp n_s u(t) \), and the primary task \( (b(t)) \) and null-space \( \pi(t) \) policies end up forming the full control action. Actually, as \( N(t)A(t)^\dagger = 0 \) and \( N(t)^2 = N(t) \), from (2) we can assert that:

\[
n_s u(t) = N(t)u(t).
\]

In a DPL context, an action mapping requires searching for optimal parameters \( w_u \). Schaal et al. (2003); Alissandrakis et al. (2007):

\[
w_u := \text{argmin}_{w_u} \|u(t) - U(w_u, t)\|^2,
\]

where the norm is defined, given \( \alpha : [0, T] \rightarrow \mathbb{R}^n \), as \( \|\alpha(t)\| := \sqrt{\int_0^T \alpha^T(t)\alpha(t) dt} \), and \( U(w_u, t) \) is a suitably parametrised function approximator, being \( w_u \) the adjustable parameters i.e.: parameters from a policy model \( U(w_u, t) \) which could be solved via any optimization procedure. In most cases, choosing a linear-in-parameter approximator will allow to employ computationally simpler learning algorithms, 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 (6).

However, such 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 the task-space and null-space components in (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 parameterisation and to formulate an optimisation problem such that the above mentioned 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/constraint information, to produce an estimation of the null-space policy \( \pi(t) \) consistent with such classification. Obviously, in a specific implementation, the above idea may require annotating the data 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’s required for the implementation in the example sections. If such training data classification is not known, it might be inferred from the data itself, as shown in the last example of this paper.

Constraint-aware Policy Learning

When learning the policy \( \pi(t) \) from constrained data, we assume that matrices \( A(t), b(t) \) and \( N(t) \) 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 this surface. This is useful, because the learned policy can be projected onto the estimated constraint.

Given the above assumption, the aim is to learn a null-space policy \( \pi(t) \) 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 \( u(t) \), we want to estimate the constraint matrix \( A(t) \) and task \( b(t) \). Later on, by combining data from all demonstrations, we will learn \( \pi(t) \).

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

\[
A(t) := A(w, t)
\]

\[
b(t) := b(w, t)
\]

\[
\pi(t) := \pi(w, \pi, t),
\]

where, we can consider the parameters of the DPL problem in (6) to be \( w_u := (w, \pi) \) in (7)-(9).
Note that, the actual explicit expression of (7)-(9) would depend on some problem-dependent information available at time $t$. In most of the cases, this will be the robot state, but also other task or constraint-dependent information as discussed in the previous section. For instance, from the robot’s forward kinematics expressed as $f(x(t), w, t) = 0$, the Pfaffian constraint is derived as $A(x(t), w, t) := \frac{\partial f}{\partial x}(x(t), w, t)$ and $b(x(t), w, t) := \frac{\partial f}{\partial t}(x(t), w, t)$ with $u(t) \equiv \dot{x}(t)$. If $f$ is time-invariant (no explicit dependence on $t$), then $b(t) \equiv 0$. This particular case is, indeed, common in many situations if we assume that the demonstration trajectories lie on a fixed “surface”.

From the above task/null-space parameterisation, given a set of demonstrations, the DPL can be reformulated as the “constraint-aware policy learning” (CAPL) problem of minimising (6) with the parameterisation

$$U(w_u, t) := A(w, t)^T b(w, t) + N(w, t) \pi(w, w_\pi, t). \tag{10}$$

i.e., minimising:

$$J(w_u) := \| u(t) - A(w, t)^T b(w, t) - N(w, t) \pi(w, w_\pi, t) \|^2. \tag{11}$$

Of course, the DPL cost (6) may be directly optimised with a suitable parameterisation. However, the assumption that the demonstrations are provided under the above-discussed constraints suggests that (10) may a better parameterisation than a generic “constraint-unaware” parameterisation of $U$. For instance, if we considered a state-dependent policy, $U(x, w_u)$, a set of training demonstrations might have different actions for the same state under different constraints (data inconsistency), see example on page 9 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 constraint-aware learning would correctly learn a different action for each constraint, see details on the referred example.

In practice, this problem is difficult to solve because, even if the approximators (7)-(9) were linear in parameters, the presence of pseudo-inverses in (11) introduces a complex relation w.r.t. to $w$. However, under mild assumptions, we can reasonably approximate the original cost function by splitting it in two simpler optimization problems. Indeed, we recall that, for any orthogonal matrix $T$, we have $\|e\| = \|Te\|$. This will inspire an orthogonal change of coordinates yielding a factored expression of $J(w_u)$ but keeping the same optimal parameter values.

**Lemma 1.** If rows of $A(w, t)$ are orthonormal, i.e.: if $A(w, t)$ is a semi-orthogonal matrix, then, we can express (11) as:

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

where:

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

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

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

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

The advantage of this approach is that $J_2(w^*, w_\pi)$ can be minimised using standard least squares if we use a linear parameterisation of $\pi(w, w_\pi)$ with respect to $w_\pi$. Additionally, regarding $J_1(w)$, parameters $w^*$ can be computed in closed form using a generalised eigenvalue method as we will show now.

**Closed-form Constraint Estimation.**

In this section, we define a method for solving the minimisation of (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 the task-space and null-space components.

Note that equation (13) only depends on parameters $w$. We can compute these parameters from the demonstrated data. If we express $A(w, t)$ and $b(w, t)$ as a linear combination of regressors$^1$, they could be defined, at any time, as:

$$A(w, t) := w_A \Phi_A(t) \tag{15}$$

$$b(w, t) := w_b \Phi_b(t), \tag{16}$$

where $w_A \in \mathbb{R}^{s \times w_A}$ and $w_b \in \mathbb{R}^{s \times w_b}$ are constant matrices composed of parameters to be learnt $w := (w_A, w_b)$, and $\Phi_A(t) \in \mathbb{R}^{w_A \times s}$, $\Phi_b(t) \in \mathbb{R}^{w_b \times s}$ are some regressors that can be evaluated from information of the demonstrated motion at time $t$, e.g. the state $x(t)$, 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.

$^1$As the constraint $A(w, t)u(t) = b(w, t)$ can be equivalently expressed as $R^T(w, t)A(w, t)u(t) = R^T(w, t)b(w, t)$ for any invertible matrix $R(w, t)$, there is no loss of generality in assuming that the Pfaffian constraint (1) in the problem statement involves matrix $A(w, t)$ having orthonormal rows. In a particular case, $R(w, t)$ can be obtained from the (economy size) QR decomposition of $A(w, t)^T$:

$$A(w, t)^T = Q(w, t)R(w, t)$$

where $Q(w, t)$ is orthogonal, and $R(w, t)$ is orthonormal results in $R(w, t)^T A(w, t) = Q(w, t)^T$.

$^2$With the appropriate regressors, any complex function can be expressed as a linear combination of parameters, such as RBF’s Neural-Networks Haykin and Network (2004) or Locally Weighted Linear Models Schaal and Alkesson (1998).
Let us, for convenience, define:
\[
\Delta(w, t) = A(w, t)u(t) - b(w, t)
\]
\[
= w_At - b(t)
\]
\[
= wH(t),
\]
(17)
where \(H(t)\) comprises all the regressors (multiplied by the control inputs) in the case of those from \(A(w, t)\) into a single matrix.

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

**Lemma 2. Consider the problem of minimising \(J := \theta^T R \theta\) subject to \(\theta^T Q \theta = 1\), with \(R\) and \(Q\) symmetric. The optimal value of \(J\) is \(\lambda\), where \(\lambda\) is the minimum generalised eigenvalue of the linear matrix pencil \(\lambda Q - R\). The minimiser \(\theta\) must be a generalised eigenvector corresponding to eigenvalue \(\lambda\).**

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 \(J_1(w)\) will actually be expressed as:
\[
w \left( \int_0^T H(t)H^T(t) dt \right) w^T,
\]
subject to:
\[
1 = w \left[ \Phi_A(t)\Phi_A(t)^T \right]_{00} = 0 \forall t \in [0, T].
\]
(19)

However, depending on the chosen parameterisation, the above constraint might be difficult to satisfy. Therefore, we propose to approximate \(J_1(w)\) with an average unit-norm constraint, i.e., enforcing:
\[
w \left[ \frac{1}{T} \int_0^T \Phi_A(t)\Phi_A(t)^T dt \right]_{00} = 0 \quad w^T = 1.
\]
(20)

Now, applying Lemma 2, we can obtain the optimal values for \(w\) by minimising (18) subject to the constraint (20). In order to do that, we compute \(R = \int_0^T H(t)H^T(t) dt\) from (18) and a rank-deficient \(Q\) from (20).

Note that, in practice, the above 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 \(H\) matrix as follows:
\[
H := \begin{bmatrix} \Phi_A(t_1)u(t_1) & \Phi_A(t_2)u(t_2) & \ldots & \Phi_A(t_N)u(t_N) \\ \Phi_b(t_1) & \Phi_b(t_2) & \ldots & \Phi_b(t_N) \end{bmatrix},
\]
(21)
where \(u(t_1), u(t_2), \ldots, u(t_N)\) are the raw observations of the action from the demonstration, with \(t_1 = 0, t_N = T\). Then, the integral in (18) would be evaluated as \(\frac{1}{N} HH^T\) and an analogous approach would be taken for (20).

In theory if several constraints are fulfilled with no error, then the (generalised) eigenvalue zero would have a multi-dimensional subspace of eigenvectors; thus, an orthogonal basis of such eigenvectors would form the rows of \(w_A\) and \(w_b\). However, in practice, such 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”/“principal components” techniques discussed in Zhang (2017), which the above proposal is related to.

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

If \(b(t) = 0\), then it can be shown (details omitted for brevity) that the problem reduces to removing the rows of \(H\) related to \(\Phi_b\) in (21), and computing the smallest singular values/vectors of \(YY^T\), being \(Y\) a scaling matrix such that \(YY^T := \frac{1}{T} \int_0^T \Phi_A(t)\Phi_A(t)^T dt\) generalising Armesto et al. (2017c). Actually, if the number of constraints is known in advance, it is easy to discriminate situations where there’s an over/under-parameterisation 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-parametrised or data fulfills more constraints than originally assumed.

### Learning the Null-space Policy

At this stage, once the minimisation of \(J_1(w)\) has been carried out and \(w^*\) is available, each data-point in the dataset can be splitted into its null-space and task-space components as:
\[
"u(w^*, t) := N(w^*, t)u(t)
\]
\[
it_u(w^*, t) := u(t) - "u(w^*, t).
\]
(22)
(23)

This can be interpreted as an estimation of the “true” null-space and task-space components (3) and (4) if the relevant eigenvalues are close to zero. Note, also that \(A(w^*, t)\)\("u(w^*, t), t = 0, A(w^*, t)\)\(nit_u(w^*, t) = b(t)\).

Now, we can estimate the optimal value of \(w_\pi\) from (14) evaluated at \(w^*\):
\[
J_2(w^*, w_\pi) = \|"u(w^*, t) - N(w^*, t)\pi(w^*, w_\pi, t)\|^2.
\]
(24)

Since \(\pi(w^*, w_\pi, t)\) is linear in parameters \(w_\pi\), this corresponds to an standard least squares problem.

### Learning with Locally Weighted Models

There are several ways we can model the policy \(\pi(w^*, w_\pi, t)\). Let us consider a very generic state-feedback policy \(\pi(x(t), w_\pi, t)\). This simple model has also been adopted by Lin et al. (2015); Howard et al. (2009); Towell et al. (2010). Indeed, the robot configuration \(x(t)\) 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 paper). We can implicitly replace dependence on \( w^* \) by the dependence on \( x(t) \) in applications where closed-loop feedback in the primary task will ensure that the above position/orientation constraints are maintained during real-time operation.

Based on the above idea, \( \pi(x(t), w_{\pi}, t) \) will be defined as a weighted combination of \( M \) local models as follows:

\[
\pi(x(t), w_{\pi}, t) := \frac{\sum_{m=1}^{M} \rho_m(x(t)) \pi_m(x(t), w_{\pi,m}, t)}{\sum_{m=1}^{M} \rho_m(x(t))}
\]

(25)

where each local model \( m \) is parametrised by a corresponding \( w_{\pi,m} \) with \( w_{\pi} := \{ w_{\pi,1}, \ldots, w_{\pi,M} \} \), and \( \rho_m(x(t)) := e^{-\frac{1}{2}(x(t) - c_m)\Sigma_m^{-1}(x(t) - c_m)} \) is the importance weight of each state observation according to the distance to a Gaussian receptive field, with centre \( c_m \) and variance \( D_m \) (a diagonal matrix). The centres 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 \( \Psi(x(t), t) \) with linear parameters as follows:

\[
\pi_m(x(t), w_{\pi,m}, t) := \Psi(x(t), t)w_{\pi,m},
\]

(26)

where \( w_{\pi,m} \), for \( m = 1, \ldots, M \) are the weight vectors to be learned. If the receptive fields \( \rho_m(x(t)) \) are dense enough and the constraint is time-independent, the local regressors \( \Psi(x(t), t) \) may be chosen as simple linear functions \( \Psi(x(t), t) := [x(t)^\top] \). 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 (24) to form the associated least squares problem. In a more general case, we would consider a policy \( \pi(x(t), w, w_{\pi}, t) \) that depends on the parameters of the primary task. The regressors may then also depend on these parameters \( \Psi(x(t), w^*, t) \).

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

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 compared to 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.

**Learning Planar-Constrained Policies**

Defining the appropriate set of regressors can be a difficult task 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 parameterisation 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, writing, etc. In all these examples, a constraint could be defined in terms of minimizing the distance to the surface and the misalignment between the surface normal and the orientation of the robot’s tool (see Figure 3). 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.

**Learning the Primary Task and the Constraint**

Let us consider a robot with some tool at its end, whose position in 3D space will be denoted as \( x_T \), and a reference frame attached to the tool denoted by the vectors \( x_T, y_T \) and \( z_T \). We consider a training scenarios where the reference surface is flat and static as shown in Figure 2. The normal to the surface \( n \) does not change with time and the primary task error can be defined using the distance of the tool to the surface and the tool’s misalignment:

\[
e(x(t)) := \begin{bmatrix} n^\top(t_T(x(t)) - p) \\
 n^\top x_T(x(t)) \\
 n^\top y_T(x(t)) \end{bmatrix},
\]

(27)

where \( p \) 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 velocities, \( \dot{x}(t) \), and its relation with respect to the velocity vector (error) of a task, \( \dot{e}(x(t)) \):

\[
\dot{e}(x(t)) = J(x(t))\dot{x}(t),
\]

(28)

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

![Figure 2. A two dimensional illustration of a robot performing the demonstrated motion on a flat surface. \( p \) is a point on the \( x_T-y_T \) plane used as a centre of the wiping motion performed in the null-space of the surface alignment task.](image-url)
From (28) and (27) we can select the following regressors:

\[
\Phi_A(x(t)) := J_T(x(t)) \equiv \begin{bmatrix} \frac{\partial x_T(x(t))}{\partial x(t)} \\ \frac{\partial y_T(x(t))}{\partial x(t)} \\ \frac{\partial z_T(x(t))}{\partial x(t)} \end{bmatrix},
\]

(29)

\[
\Phi_b(x(t)) := \begin{bmatrix} t_T(x(t))^T x_T(x(t))^T y_T(x(t))^T \end{bmatrix},
\]

(30)

where the above primary task controller would 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:

\[
J(x(t)) = \begin{bmatrix} u^T & 0 & 0 \\ 0 & n^T & 0 \\ 0 & 0 & n^T \end{bmatrix} J_T(x(t)) \quad (31)
\]

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

In theory, the regressors for \( \Phi_A(x(t)) \) 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 \( \Phi_b(x(t)) \) 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 to the surface during the demonstration will, in general, make it impossible for the approximator errors \( J_1 \) and \( J_2 \) in (13) and (14) to become exactly zero. As earlier commented, from the analysis of the singular values, as the dimension of \( \sigma(x(t)) \) is known we can clearly identify situations where extra parametrisation is needed if the number of eigenvalues which are significantly smaller compared to the rest of the them is less than three.

Regarding regressors \( \Phi_b(x(t)) \), in realistic applications, learning primary task controllers isn’t 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 \( b(x(t)) = 0 \). This assumption has computational benefits, reducing the generalised eigenvalue computations to faster ordinary eigenvalue ones, as discussed earlier.

Learning the Null-space Policy

We will now propose a specialized structure for the null-space policy \( \pi(t) \) based on the Jacobian specific to the of 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 DoF) and maintain the contact (constraining one more DoF). This implies a task which constrains a total of three of the DoF 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 parametrisation of the null-space policy \( \pi(x(t), t) \).

Since the tool’s orientation is constrained by the primary task, only the position trajectory \( T_T(x(t)) \) is relevant for the null-space policy. We choose and arbitrary reference frame \( (\xi_x(n), \xi_y(n)) \) on the surface orthogonal to the normal \( n \), and we define a modified tool’s Jacobian:

\[
J^v(x(t)) = \begin{bmatrix} \xi_x(n)^T & 0 & 0 \\ \xi_y(n)^T & 0 & 0 \end{bmatrix} J_T(x(t)) \quad (32)
\]

which computes the tool’s speed relative to this reference frame. The estimated parameters \( w_A \) will not, in general, coincide with the block-diagonal expression arising from (31), nonetheless, if the orientation error during the demonstration is reasonably small, the surface normal would be close to the actual tool’s \( z_T(x(t)) \) vector. So, we will neglect this error and propose the parametrisation:

\[
J^z(x(t)) = \begin{bmatrix} \xi_x(z_T(x(t)))^T & 0 & 0 \\ \xi_y(z_T(x(t)))^T & 0 & 0 \end{bmatrix} J_T(x(t)) \quad (33)
\]

which will be, basically, coincident with (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:

\[
\kappa(t) := J^z(x(t)) u(t). \quad (34)
\]

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

If the robot had 5 DoF, this choice of \( \kappa(t) \) (2 DoF) and \( \Phi_A(x(t)) \) (3 DoF) would ensure that the solution for \( u(t) \) is unique. However, if the robot has more than 5 DoF, there would be remaining redundant degrees of freedom that \( \kappa(t) \) won’t model.

To account for this redundancy, we complement the secondary task description \( J^s(x(t)) \) with additional independent rows describing the Jacobians of quantities \( \eta(x(t), t) \) related to the application by setting \( \tilde{\gamma}(t) := \tilde{\eta}(x(t), t) = J^0(x(t)) u(t) \). 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 parameterisation of the policy, let’s consider the expression:

\[
T(x(t), w^*, t) u(t) := \begin{bmatrix} A(x(t), w^*, t) \\ J^x(x(t)) \\ J^y(x(t)) \end{bmatrix} u(t) = \begin{bmatrix} b(t) \\ \kappa(t) \\ \gamma(t) \end{bmatrix}
\]

(35)

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

Let us denote \( T^{-1}(x(t), w^*, t) \equiv [E_6, E_{w}, E_3] \) suitably partitioning of the columns of \( T^{-1}(x(t), w^*, t) \) compatible with the dimensions of \( b(t), \kappa(t), \gamma(t) \). Let us also consider suitable function approximators where \( \kappa(w^*, w_e, t) \) is a parameterised approximator of \( \kappa(t) \) and, similarly, \( \gamma(w^*, w_e, \gamma, t) \) is a parameterised approximator of \( \gamma(t) \).
Lemma 3. The minimization of $J_2$ in (24) is equivalent to solving the following least-squares problem:

$$w^*_\kappa := \arg \min_{w^*_\kappa, w^*_\gamma} \left\| (E_{\kappa} - E_{\kappa} \gamma(w^*_\kappa, t)) - N \mu(t) + (E_0 - A^l) \hat{b}(t) \right\|^2,$$

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

$$\pi(w^*, w^*_\kappa, t) := (E_0 - A^l) \hat{b}(w^*, t) + E_\kappa \gamma(w^*, w^*_\gamma, t).$$

The proof of the above lemma appears in Appendix C.

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

$$\pi(w^*, w^*_\kappa, t) := E_\kappa \kappa(w^*, w^*_\kappa, t) + E_\gamma \gamma(w^*, w^*_\gamma, t).$$

If we choose to linearly parametrize the regressors $\kappa(w^*, w^*_\kappa, t) := \Phi_\kappa(w^*, w^*_\kappa, t)$ and $\gamma(w^*, w^*_\gamma, t) := \Phi_\gamma(w^*, w^*_\gamma, t)$, the solution to (36) can be solved with standard linear least squares. These regressors may, too, be used to set up the locally weighted models, as discussed earlier, and the learning problem will still keep being 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 parameterisations.

Suitable regressors for $\kappa$. Recall that the components of $\kappa(w^*, w^*_\kappa, t)$ 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 $\kappa(x(t), w^*, w^*_\kappa, t)$.

For instance, let us assume the robot is tracking a curve $f(\nu) = 0$, where $\nu := (\nu_x, \nu_y)$ 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 $f(\nu)$) and a motion towards the curve (proportional to the gradient of $f(\nu)$), as follows:

$$\dot{\nu} = \left( \frac{\partial f}{\partial \nu_x} \right) \nu_x - \left( \frac{\partial f}{\partial \nu_y} \right) \nu_y = \left( \frac{\partial f}{\partial \nu_x} \right) w_x - \left( \frac{\partial f}{\partial \nu_y} \right) w_y$$

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 curve). This regressors can be seen as a type of dynamic motion primitives for curves Ijspeert et al. (2003).

As an example, if $f(\nu)$ were a circle $(\nu_x - c_x)^2 + (\nu_y - c_y)^2 - r^2 = 0$, the expression for $\dot{\nu}$ would be a 3rd polynomial in $\nu_x$ and $\nu_y$. We therefore place the respective monomials appearing in (39) in the regressors for $\kappa(x(t), w^*, w^*_\kappa, t)$.

Suitable regressors for $\gamma$. The quantities $\gamma(w^*, w^*_\gamma, t)$ represent redundant degrees of freedom. We will assume that there is a “comfortable” pose $\eta^{ref}$ such that a controller $\tilde{\eta} = K_\eta(\eta^{ref} - \eta(x(t)))$ was approximately used in the demonstrations. We then define the regressors as the affine expression $\gamma(x(t), w^*, w^*_\gamma, t) = [-K_\eta(K_\eta^{ref}) \tilde{\eta}(x(t))]$ so that the matrix $[-K_\eta K_\eta^{ref}]$ would be the “ground-truth” parameter $w_\gamma$ 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 demonstrate that they are suitable for the modelling wiping task.

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 due to outdoors lighting conditions, different surface materials, and its dimensions. Thus, in practical applications, the constraint surface may not be known. Therefore, we aim to redefine the surface alignment task using, for instance, a force/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 3 this alignment corresponds to having the end-effector local $z$ axis collinear with the surface normal and the the end-effector local $x$ and $y$ axis are tangent to the surface. As illustrated in Figure 3, this alignment corresponds to having the minimal torque around the local $x$ and $y$ axis at the contact point, and having the contact force applied along the local $z$ axis. Therefore, we can define an alignment task error as

$$e_F(x(t)) := \begin{bmatrix} f_x - f_z \\ -m_x \\ -m_y \end{bmatrix},$$

where $f_x$ is the $x$ component of the contact force and $f_z$ is the desired contact force, and $m_x$, and $m_y$ are the $x$ and $y$ components of the contact torque relative to the tool axis. By attaching a force/torque sensor at the tip of the end-effector, we can measure the contact wrench (force and torque), and by minimizing $e_F$ the robot end-effector will align with the contact surface.

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

$$J_F(x(t)) = \begin{bmatrix} x_T^T \\ 0^T \end{bmatrix} J_T(x(t))$$

where $J_T(x(t)) \in \mathbb{R}^{6 \times 7}$ represents a standard geometric robot Jacobian.

Remark 1. We intentionally used a “different” Jacobian (40) for the real-time operation (based on sensor information). This Jacobian replaces the purely geometric choice (27) during learning in order to show the generalisation.
capabilities to a new primary constraint/control law. Additionally, this allows us to re-project learnt planar path \( \kappa(\mathbf{w}^*, \mathbf{w}^*_c, t) \) onto the constraint defined around the normal \( \mathbf{z}_T \) even if its not constant.

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

It is important to remark that the error vector (40) used in wiping a non-flat surface with force feedback is different from the error used during the demonstration on the flat surfaces (27). Despite of that, the learnt policy \( \pi(\mathbf{w}^*, \mathbf{w}^*_c, t) \) and also low-dimensional policies \( \kappa(\mathbf{w}^*, \mathbf{w}^*_c, t) \) and \( \gamma(\mathbf{w}^*, \mathbf{w}^*_c, t) \), 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 \( \mathbf{A}_o(\mathbf{x}(t), t), \mathbf{b}_o(\mathbf{x}(t), t) \) at runtime. By substituting these sensor-based constraint into (35) we get:

\[
\begin{align*}
\mathbf{T}_o(\mathbf{x}(t), t)\mathbf{u}(t) := & \begin{pmatrix}
\mathbf{b}_o(\mathbf{x}(t), t) \\
\mathbf{\kappa}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}^*_c, t) \\
\mathbf{\gamma}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}^*_c, t)
\end{pmatrix},
\end{align*}
\]

with \( \mathbf{T}_o(\mathbf{x}(t), t) := \begin{pmatrix}
\mathbf{A}_o(\mathbf{x}(t), t) \\
\mathbf{J}_F(\mathbf{x}(t))
\end{pmatrix} \).

In real-time operation, \( \mathbf{T}_o(\mathbf{x}(t), t) \) is known, therefore the state-feedback controller will be given by:

\[
\mathbf{u}(\mathbf{x}(t), t) = \mathbf{T}_o^{-1}(\mathbf{x}(t), t) \begin{pmatrix}
\mathbf{b}_o(\mathbf{x}(t), t) \\
\mathbf{\kappa}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}^*_c, t) \\
\mathbf{\gamma}(\mathbf{x}(t), \mathbf{w}^*, \mathbf{w}^*_c, t)
\end{pmatrix},
\]

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

\[
\begin{align*}
\mathbf{A}_o(\mathbf{x}(t), t) := & \mathbf{J}_F(\mathbf{x}(t)), \\
\mathbf{b}_o(\mathbf{x}(t), t) := & -\mathbf{K}_P \mathbf{e}_F(\mathbf{x}(t))
\end{align*}
\]

### Examples

**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 –norm of the velocity vector– and at constant distance of 1 meter from the origin. When restricting the motion of this particle to a plane intersecting the origin, the resulting trajectory is a circumference centred 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 ±60° with the y axis, as shown in Figure 4.

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 actions \( \mathbf{u}(t) \in \mathbb{R}^3 \) as the particle velocity. Each sub-dataset has 500 data points that correspond to a full revolution with duration of 5 seconds - in Figure 4 we plot the trajectories using one fifth of the total number of training samples.

**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 \( \mathbf{\Phi}(t) = \mathbf{I}_{3 \times 3} \in \mathbb{R}^{3 \times 3} \), where \( \mathbf{I}_{3 \times 3} \) is the identity matrix. Moreover, in each demonstration, the particle never leaves the constraint plane \( \mathbf{b}(t) = 0 \), 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 \( \mathbf{w}_{A_1} = [0.0 - 0.866 0.5] \) and \( \mathbf{w}_{A_2} = [0.0 0.866 0.5] \) 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 (5) for each constraint. For the unconstrained policy we used a linear policy suited for this particular problem:

\[
\pi(\mathbf{x}) := \begin{bmatrix}
\mathbf{x}^\top & 0 & 0 \\
0 & \mathbf{x}^\top & 0 \\
0 & 0 & \mathbf{x}^\top
\end{bmatrix} \cdot \mathbf{w}_x .
\] (45)

**DPL.** For the Direct Policy Learning we first used the same policy function (45).

Let us now compare the performance of both approaches. DPL learning is biased because of the inconsistent data at intersection points. For this particular example different actions \( \mathbf{u} = [0 \ 0.5 \ 0.855]^T \) for the first constraint and \( \mathbf{u} = [0 \ 0.5 \ -0.855]^T \) for the second constraint at point \( \mathbf{x} = [1 \ 0 \ 0]^T \) appear in the training data. With a single regressor, the biased DPL learning affects all state space producing incorrect trajectories even when trying to replicate the trained demonstrations (not shown in Figure 4 to avoid cluttering). In order to improve the fit at training data, we tuned 20 locally weighted regressors distributed across the training set via K-means algorithm Kanungo et al. (2002). However, the DPL fundamental problem at the intersection points can not be overcome (see Figure 4 showing policy.

---

**Figure 3.** A two dimensional illustration of a robot performing a constrained task on a 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 \( f \), a normal force \( f_n \), and a contact torque \( m_c \), where the arrows indicate the direction in which the values \( f \) and \( f_n \) 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.
execution). On the other hand, our CAPL produces the correct actions at intersection points, once projected over the constraint.

Learning a Wiping Policy

We have reproduced conditions in Armesto et al. (2017a) to simulate a kinematic 7-DOF KUKA LBR IWA R800 robot. We have generated a circular wiping motion together with a joint limit avoidance policy affecting the 1st, 3rd and 7th joints of the robot as described in Armesto et al. (2017a). Both of 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 a planar surface. A single trajectory of a wiping motion with a duration of 1.5 sec sampled at 0.01 sec 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 alignment errors are initially close to zero. We can therefore use the SVD approach to estimate the constraint parameters. During the data generation, we have artificially added a Gaussian noise with a standard deviation of 5% of the joint’s physical range on each of the joints’ velocity, to emulate collection of noisy (non-perfect) data from a human operator.

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

Figure 5 shows the (simulated) measured velocities in the tool’s plane $\kappa(t)$ compared with the execution of a parametrized version of $\kappa(w^*, w^*_w, t)$ which exploits our proposed polynomial regressors for the circular trajectories. In Figure 6 we shown the equivalent result used for the joint limit avoidance for the redundant joints ($\gamma$ versus $\gamma$). 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.

Experimental Setup

Testing with Real Data and a Force Sensor

In our experiments, we use the 7 DoF KUKA LWR3 robot with an ATI industrial automation Gamma F/T sensor attached at the end effector, as shown in Figure 10. The force sensor retrieves a 6 dimensional wrench vector expressed in the sensor frame. Therefore, we compute the torque at the contact point by transforming the wrench by distance $d_S$ towards 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 (40) is achieved by admittance control. This means that the robot compensates the end-effector position and orientation according the wrench feedback. In order 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 behaviour on 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, as shown in Figure 8. The dataset contains 12 trajectories, each on a surface of a different orientation (four of them shown in Figure 9). Each demonstration involved several circles with the tool of the robot, containing approximately 2000 data points1 (using a sampling rate of 100Hz). The demonstrated data was only minimally cropped to ensure the data contained only poses where the tool was in contact with the surface and moving along the demonstrated trajectory.

We used this dataset to first learn the different constraint matrices, by parametrising them as a linear combination of regressors, functions of state, and by applying the estimation method described in Section Closed-form Constraint Estimation. The regressors for each constraint matrix are the ones from expression (29). For the policy \( \pi \) we used 25 Locally Weighted Models with the same regressors used in Armesto et al. (2017c). The resulting policy was then stored and used, in closed-loop, together with the force-based surface alignment task described in the previous section.

Figure 9 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 equations (11), (13), and (14), respectively. The figure shows that the locally-weighted model has learnt that there is a “common” circular wiping motion across the different demonstrations.

Furthermore, we have also validated the learnt policy on a non-flat surface, as shown in Figure 10, 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. (2017b) 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 properly align the tool with the surface), resulting in a dataset with demonstrations in which \( A(x)u \approx 0 \), and consequently reducing the amount of error in the task policy.

<table>
<thead>
<tr>
<th>Dem.</th>
<th>( J )</th>
<th>( J_1 )</th>
<th>( J_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0206</td>
<td>0.81 ( \cdot ) 10^{-6}</td>
<td>0.0199</td>
</tr>
<tr>
<td>2</td>
<td>0.0445</td>
<td>2.36 ( \cdot ) 10^{-6}</td>
<td>0.0431</td>
</tr>
<tr>
<td>4</td>
<td>0.0319</td>
<td>2.39 ( \cdot ) 10^{-6}</td>
<td>0.0302</td>
</tr>
<tr>
<td>7</td>
<td>0.0199</td>
<td>4.43 ( \cdot ) 10^{-6}</td>
<td>0.0175</td>
</tr>
</tbody>
</table>

**Constraint Similarity Analysis**

In all experiments so far, we assumed that the demonstrator provides a set of sub-datasets \( \{X_1, X_2, \ldots, X_r\} \), each of which contains samples of pairs of raw observations, that encapsulate a sufficiently diverse set of task/constraints, allowing us to uncover the underlying policy common to all demonstrations that, therefore, can be generalized to

---

1The data included the joint state \( x(t) \) and the joint commands \( u(t) \), obtained by differentiating the joint states.
different constraints. In order to estimate the unconstrained policy, we need demonstrations from different constraints Howard and Vijayakumar (2007b). As a consequence, a typical data-set will contain a sequence of demonstrations which will be classified as sub-datasets.

The aim now is to analyse how similar/distinct are these sub-datasets 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 originated by different constraints, in order to help us identifying the transition regions. The experiments in this section are meant to provide an additional analysis to 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.

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

$$ J_{1,k,l} = \| A(\hat{w}_k, t_l) u(t_l) - b(\hat{w}_k, t_l) \|^2. $$

where we indexed 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 previous sub-section, we manually selected the $\nu$ sub-datasets. This pre-processing step separates the full dataset into the sub-datasets. Figure 12 shows the Cartesian positions of the Kuka’s end-effector for the full dataset (blue) and, overlapping, the corresponding sub-datasets manually separated (red).

This manual separation was done 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 to verify even 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 perform 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 which 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 than the size of the window creating a window \( k + 1 \) (size refers to the number of consecutive data points). If the parameters estimated in this new window produce low \( J_1 \), then it 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 in Figure 13. This matrix corresponds to the data shown in Figure 12. We have empirically chosen the window size of 400 samples (corresponding to 8 seconds for a sampling frequency of 50 Hz) and increments of 50 samples (1 second). In Figure 13 we also overlap boxes showing the manual separation provided by the expert. There are also at least two groups of windows (around indices 120 and 150) that could be confused with demonstrations given that they produce squares of low \( J_1 \) 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 is consistent with some constraint, which is sufficiently well modelled 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.

**Conclusion**

This paper presented 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 the 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 to 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 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 specialised regressors. This decreased the number of parameters the algorithm has to 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 in 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 considered the case of a null-space policy that, instead of having the full dimension of the system actions, can be decomposed, by assumption, in a set of lower dimensional policies. For this case, we proposed an alternative reformulation for estimating this lower dimensional policies, provided the respective regressors.

However, in order to learn a generalisable 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 consists in 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 windows of data. This can be a valuable tool for helping identifying 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.
Appendix A - Proof of Lemma 1

Let us define the the transformation matrix $T(w, t) := \left[ A(w, t) \right]^\top$. This matrix is orthogonal because the matrix $null[ A(w, t) ]^\top$ is an orthogonal basis of the left null-space of $A(w, t)$, i.e., the set of row vectors $N_A(w, t) := \{ \psi \in \mathbb{R}^{1 \times d} : \psi \bar{A}(w, t)^\top = 0 \}$.

Indeed, by orthogonality $A(w, t)^\top A(w, t) = I$, so $A(w, t)^\top b(w, t) = b(w, t)$, yielding $J_2(w)$. Also, $A(w, t)N(w, t) = 0$ because $N(w)$ projects on the right null-space of $A(w, t)$. Last, $J_2(w, w_e)$ is the norm of a column vector belonging to the right null-space of $A(w, t)$ (and this null-space is actually the transpose of $N_A(w, t)$). 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 $null[ A(w, t) ]^\top$ does.

Appendix B - Proof of Lemma 2

To prove Lemma 2, we introduce a Lagrange multiplier $\lambda$ and construct the following augmented cost function:

$$ L = \theta^\top R \theta + \lambda(\theta^\top Q \theta - 1), \quad (47) $$

which we differentiate to get:

$$ 0 = \frac{\partial L}{\partial \theta} = 2R \theta - 2\lambda Q \theta \Rightarrow \theta = \lambda Q \theta. \quad (48) $$

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

The resulting eigenvectors for different eigenvalues are Q-orthogonal: if $R x = \lambda Q x$ and $R y = \mu Q y$, we have

$$ \lambda(Q x)^\top y = \lambda x^\top Q y = x^\top R y = \mu x^\top Q y. \quad (49) $$

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

Appendix C - Proof of Lemma 3

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

$$ u(t) = E_0 \tilde{b}(t) + E_\kappa \tilde{\kappa}(t) + E_\gamma \tilde{\gamma}(t), \quad (50) $$

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

$$ n^e u(w^*, t) = u(t) - A^\dagger \tilde{b}(t) $$

$$ = (E_0 - A^\dagger) \tilde{b}(t) + E_\kappa \tilde{\kappa}(t) + E_\gamma \tilde{\gamma}(t), \quad (51) $$

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

$$ A \left( (E_0 - A^\dagger) \tilde{b}(t) + E_\kappa \tilde{\kappa}(t) + E_\gamma \tilde{\gamma}(t) \right) = 0, \quad (52) $$

because $AT^{-1} = [I \ 0 \ 0]$, $J^T T^{-1} = [0 \ 1 \ 0]$ and $J^T T^{-1} = [0 \ 0 \ I]$, hence, we can assert that $AE_0 = I$ and therefore $A(E_0 - A^\dagger) = 0$, and $AE_\kappa = 0$, $AE_\gamma = 0$.

Thus, in order to build the null-space terms in (24), inspired in (51), we can define the parameters $w_e := (w_e, w_e)$ and set up the expression:

$$ \pi(w^*, w_e, t) := (E_0 - A^\dagger) b(t) + E_\kappa \gamma(w^*, w_e, t) + E_\gamma \gamma(w^*, w_e, t) \quad (53) $$

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

$$ 0 = Nu(t) - (E_\kappa \ E_\gamma) \left( \kappa \gamma(w^*, w_e, t) \right) - (E_0 - A^\dagger)\tilde{b}(t), \quad (54) $$

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

$$ \mathcal{U}(w^*_u, t) := E_0 b(w^*, t) + E_\kappa \gamma(w^*, w^*_e, t) + E_\gamma \gamma(w^*, w^*_e, t), \quad (55) $$

with $w^*_u = (w^*, w^*_e, w^*_e)$ and defining the estimated null-space policy as:

$$ \pi(w^*, w^*_e, t) := (E_0 - A^\dagger) b(w^*, t) + E_\kappa \gamma(w^*, w^*_e, t) + E_\gamma \gamma(w^*, w^*_e, t). \quad (56) $$

Acknowledgements

The authors are grateful to the financial support of Spanish Ministry of Economy and European Union, grant DPI2016-81002-R (AEI/FEDER, UE) and the Engineering and Physical Sciences Research Council (EPSRC), as part of the CDT in Robotics and Autonomous Systems at Heriot-Watt University and The University of Edinburgh, under the grants EP/L016834/1, EP/J015040/1.

References


Matthew Howard and Sethu Vijayakumar. Reconstructing null-space policies subject to dynamic task constraints in redundant manipulators. 2007b.


