2017 IEEE International Conference on Robotics and Automation (ICRA)
Singapore, May 29 - June 3, 2017

Learning Task Constraints in Operational Space Formulation

Hsiu-Chin Lin, Prabhakar Ray, and Matthew Howard

***Abstract*— Many human skills can be described in terms of performing a set of prioritised tasks. While a number of tools have become available that recover the underlying control policy from constrained movements, few have explicitly considered learning how constraints should be imposed in order to perform the control policy. In this paper, a method for learning the self-imposed constraints present in movement observations is proposed. The problem is formulated into the operational space control framework, where the goal is to estimate the constraint matrix and its null space projection that decompose the task space and any redundant degrees of freedom. The proposed method requires no prior knowledge about either the dimensionality of the constraints nor the underlying control policies. The techniques are evaluated on a simulated three degree-of-freedom arm and on the AR10 humanoid hand.**

I. Introduction

Many human skills can be described in terms of performing synchronous, prioritised tasks. For example, when operating the remote control of an electrical device (see Fig. 1), one must simultaneously manage the control of gripping, button pressing and orienting the transmitting end of the controller toward the electrical device's receiver. Additional, secondary control may also be involved in regulating degrees of freedom not directly involved in the task, for example, maintaining a comfortable posture or avoiding joint limits. Isolating the different components of such behaviour from data—for instance, extracting the button pressing policy—is no trivial task, especially considering that it involves the simultaneous execution of multiple control policies in different subspaces of configuration space, and that these may come into conflict or impinge on one another in their execution.

To manage this complexity, a general framework for the control of redundancy is operational space control [1]. This formulation enables the composition of joint-space movements through the selection of a set of prioritised task space constraints. Applications include defining different task controllers for multiple end-effectors [2], [3], avoiding obstacles [4], and balancing humanoid robots [5]. From the perspective of *designing* robot behaviours, if the model of the robot and the required task are precisely known, operational-space control can be applied with relative ease, provided that the Jacobian of the system and the inertia matrices are accurate.

However, the reverse of this—that is, analysing synchronous behaviours from observations—is a much harder problem. In this setting, it may not be clear which dimensions should be constrained or how redundancy is resolved from direct observations. A potential solution is to take examples from human demonstrations and attempt to learn a control policy that somehow captures the behaviours [6], [7], [8]. This includes learning the operational space control law [9], or recovering the underlying redundancy from the constrained data [10], [11].

H. Lin (h.lin@bham.ac.uk) is at the School of Informatics, University of Edinburgh, UK. P. Ray and M. Howard (matthew.j.howard@kcl.ac.uk) are at the Dept. of Informatics, King's College London, UK.

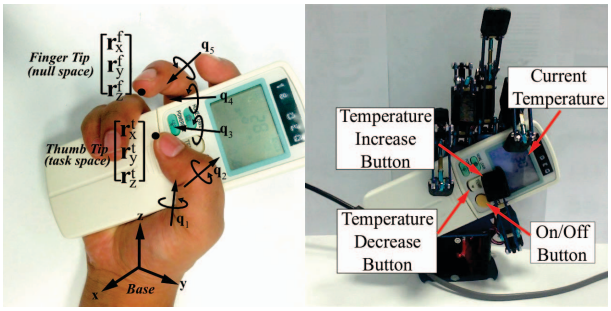

Fig. 1: Handling a remote control for an air conditioning unit. The task space for pressing the temperature control button is defined as the thumb tip position $(r_x^t, r_y^t, r_z^t)^\top$ (the task), while the control of other fingers (redundant dimensions) is handled by other, secondary control policies (*e.g.,* for gripping or maintaining a comfort posture).

However, to date, such methods have been limited in that they learn *motions related to the task* but do not provide a model of the *task space itself*. The ability to do so would transform our ability to generalise behaviours, since having it would allow us to replace the demonstrated task policy with a new policy for new situations. In the remote control example, if one could learn from observations of pressing one of the buttons that the task space for button pressing in general involves control of the thumb tip, then one can easily define new policies in which the thumb presses other buttons instead (*e.g.,* by adjusting the policy attractor points).

In this paper, a method for directly learning the self-imposed constraints present in movement observations is proposed. The problem is formulated as an operational space control problem, where the goal is to *estimate the constraint matrix that defines the task space*. The proposed method requires no prior information about either the dimensionality of the constraints, nor the policy underlying the observed movement. The techniques are evaluated on a simulated three degree-of-freedom arm and on the AR10 humanoid hand.

II. Problem Definition

Based on the principles of analytical dynamics [12], it is assumed that the systems under consideration are subject to a set of $\mathcal{S}$-dimensional ($\mathcal{S} \leq \mathcal{Q}$) constraints

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

where $\mathbf{x} \in \mathbb{R}^{\mathcal{P}}$ represents state, $\mathbf{u} \in \mathbb{R}^{\mathcal{Q}}$ represents the action, and $\mathbf{b} \in \mathbb{R}^{\mathcal{S}}$ is the *task space policy* describing the

underlying task to be accomplished. $\mathbf{A}(\mathbf{x}) \in \mathbb{R}^{\mathcal{S}\times\mathcal{Q}}$ is the *constraint matrix*, that projects the task space policy onto the relevant part of the control space. Inverting (1), results in the relation

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

where $\mathbf{A}^{\dagger}$ is the Moore-Penrose pseudo-inverse of $\mathbf{A}$,

$$\mathbf{N}(\mathbf{x}) := \mathbf{I} - \mathbf{A}^{\dagger}(\mathbf{x})\mathbf{A}(\mathbf{x}) \in \mathbb{R}^{\mathcal{Q}\times\mathcal{Q}} \tag{3}$$

is the projection matrix and $\mathbf{I} \in \mathbb{R}^{\mathcal{Q}\times\mathcal{Q}}$ is the identity matrix. The projection matrix $\mathbf{N}$ projects the null space policy $\boldsymbol{\pi}$ onto the null space of $\mathbf{A}$ (which in general, has non-linear dependence on state).

In the context of operational space control, the constraint matrix $\mathbf{A}$ defines the operational/task space, since it projects control actions into the space corresponding to key task variables. For example, consider the kinematic control of a task such as holding a glass of water without spilling it. The task space control objective is to maintain the orientation, so $\mathbf{b}(\mathbf{x})$ may correspond to a stabilising point attractor in hand orientation space with a stationary point corresponding to the glass upright position. Defining $\mathbf{A}(\mathbf{x})$ as the Jacobian mapping from joint space to end-effector orientation, and applying (2), actions due to $\mathbf{b}(\mathbf{x})$ in task space will be mapped into joint space for execution.

Now, if we wish to generalise this task to, for example, pouring the water out of the glass *the same task space applies*, however, *the task space policy* $\mathbf{b}(\mathbf{x})$ *must change*. This is straightforward if the correct $\mathbf{A}$ is known (*e.g.,* one might adjust the stationary point of $\mathbf{b}(\mathbf{x})$). However, in general, the decomposition of $\mathbf{A}$, $\mathbf{b}$, $\mathbf{N}$ and $\boldsymbol{\pi}$ are not directly observable making this kind of generalisation difficult. It is this problem that the present paper addresses.

In [13], it was first demonstrated that $\mathbf{A}$ (equivalently, $\mathbf{N}$) can be estimated for the special case of $\mathbf{A}\mathbf{u} = \mathbf{0}$. The approach proposed here extends that work to estimate $\mathbf{A}$ for the generic case described by (1) (with $\mathbf{b} \neq \mathbf{0}$). This is the first time this has been shown for problems in the full operational space formulation (1)-(2).

III. Method

The proposed method works on data given as $\mathcal{N}$ pairs of observed states $\mathbf{x}_n$ and actions $\mathbf{u}_n$. It is assumed that (i) the observations follow the formulation in (2), (ii) the task space policy $\mathbf{b}$ varies across observations, (iii) the null space policy $\boldsymbol{\pi}$ is the same across observations, (iv) neither $\mathbf{A}$, $\mathbf{N}$, $\mathbf{b}$ nor $\boldsymbol{\pi}$ are explicitly known for any given observation. The aim is to form an estimate of the task space constraint matrix $\mathbf{A}$.

A. Learning the null space component

The key to the proposed approach is to use the properties of $\mathbf{N}$ as related to $\mathbf{A}$ through (3). From (2), two orthogonal components of the movement can be identified, namely

$$\mathbf{u}^{\mathrm{ts}}(\mathbf{x}) := \mathbf{A}^{\dagger}(\mathbf{x})\mathbf{b}(\mathbf{x}), \tag{4}$$

referred to here as the *task space component*, and

$$\mathbf{u}^{\mathrm{ns}}(\mathbf{x}) := \mathbf{N}(\mathbf{x})\boldsymbol{\pi}(\mathbf{x}), \tag{5}$$

the *null space component*. The first step of the proposed approach is to extract an estimate of the null space component (5) from the raw observations.

From [11], an estimate $\tilde{\mathbf{u}}^{\mathrm{ns}}(\mathbf{x})$ can be found by minimising

$$E[\tilde{\mathbf{u}}^{\mathrm{ns}}] = \sum_{n=1}^{\mathcal{N}} ||\tilde{\mathbf{P}}_n \mathbf{u}_n - \tilde{\mathbf{u}}^{\mathrm{ns}}_n||^2 \tag{6}$$

where $\tilde{\mathbf{u}}^{\mathrm{ns}}_n := \tilde{\mathbf{u}}^{\mathrm{ns}}(\mathbf{x}_n)$ and $\tilde{\mathbf{P}}_n := \tilde{\mathbf{u}}^{\mathrm{ns}}_n \tilde{\mathbf{u}}^{\mathrm{ns}\,\top}_n / ||\tilde{\mathbf{u}}^{\mathrm{ns}}_n||^2$. This exploits the identity $\mathbf{P}\mathbf{u} = \mathbf{P}(\mathbf{u}^{\mathrm{ts}} + \mathbf{u}^{\mathrm{ns}}) = \mathbf{u}^{\mathrm{ns}}$, see [11] for details.

Having an estimate of the null space term $\tilde{\mathbf{u}}^{\mathrm{ns}}$, and knowing that the data follows the relationship (2), allows several properties of this relation to be used to form the estimate of $\mathbf{A}$.

B. Learning the projection matrix

Since $\mathbf{u}^{\mathrm{ns}}$ is the projection of $\boldsymbol{\pi}$ onto the image space of $\mathbf{N}$ (see (5)), and by the indempotence of $\mathbf{N}$,

$$\mathbf{N}\mathbf{u}^{\mathrm{ns}} = \mathbf{u}^{\mathrm{ns}} \tag{7}$$

must hold [13]. This means that an estimate $\tilde{\mathbf{N}}$ may be formed by minimising

$$E[\tilde{\mathbf{N}}] = \sum_{n=1}^{\mathcal{N}} ||\tilde{\mathbf{u}}^{\mathrm{ns}}_n - \tilde{\mathbf{N}}_n \tilde{\mathbf{u}}^{\mathrm{ns}}_n||^2 \tag{8}$$

where $\tilde{\mathbf{N}}_n := \tilde{\mathbf{N}}(\mathbf{x}_n)$. Fig. 2a-2b shows a visualisation of this objective function. In Fig. 2a, an example data point is plotted where $\mathbf{A}$ is a vector parallel to the z-axis, its null space is the xy-plane, the null space component $\mathbf{u}^{\mathrm{ns}}$ is parallel to the y-axis, and the task space component $\mathbf{u}^{\mathrm{ts}}$ is parallel to the z-axis. The objective function (8) aims at minimising the distance between $\mathbf{u}^{\mathrm{ns}}$ and its projection onto the null space of $\mathbf{A}$ (green plane), illustrated as the red dashed line.

Second, as noted above, $\mathbf{u}^{\mathrm{ts}}$ and $\mathbf{u}^{\mathrm{ns}}$ are orthogonal (*i.e.,* $\mathbf{u}^{\mathrm{ts}\,\top}\mathbf{u}^{\mathrm{ns}} = 0$) by definition and so the true projection matrix must also satisfy $\mathbf{N}\mathbf{u}^{\mathrm{ts}} = \mathbf{0}$. Using this insight, an alternative is to seek an estimate $\tilde{\mathbf{N}}$ that minimises

$$E[\tilde{\mathbf{N}}] = \sum_{n=1}^{\mathcal{N}} ||\tilde{\mathbf{N}}_n \tilde{\mathbf{u}}^{\mathrm{ts}}_n||^2 \tag{9}$$

where $\tilde{\mathbf{u}}^{\mathrm{ts}}_n := \tilde{\mathbf{u}}^{\mathrm{ts}}(\mathbf{x}_n)$. A visualisation of (9) for the example data point is shown in Fig. 2c, where now the blue dashed line indicates the distance minimised.

Since both (8) and (9) contain information about the projection matrix, the third alternative—proposed here—is to minimise the sum of two, namely,

$$E[\tilde{\mathbf{N}}] = \sum_{n=1}^{\mathcal{N}} ||\,\tilde{\mathbf{u}}^{\mathrm{ns}}_n - \tilde{\mathbf{N}}_n\,\tilde{\mathbf{u}}^{\mathrm{ns}}_n\,||^2 + ||\tilde{\mathbf{N}}_n\,\tilde{\mathbf{u}}^{\mathrm{ts}}_n\,||^2 \tag{10}$$

as illustrated in Fig. 2d. While this incurs a slight increase in computational cost (due to the need to evaluate the two terms instead of one), it has important benefits in ensuring the learnt $\tilde{\mathbf{A}}$ has *correct rank*, that are missed if (8) or (9) are used in isolation. In other words (10) helps ensuring the learnt constraints have the *correct dimensionality*, as shall be elucidated in the following section.

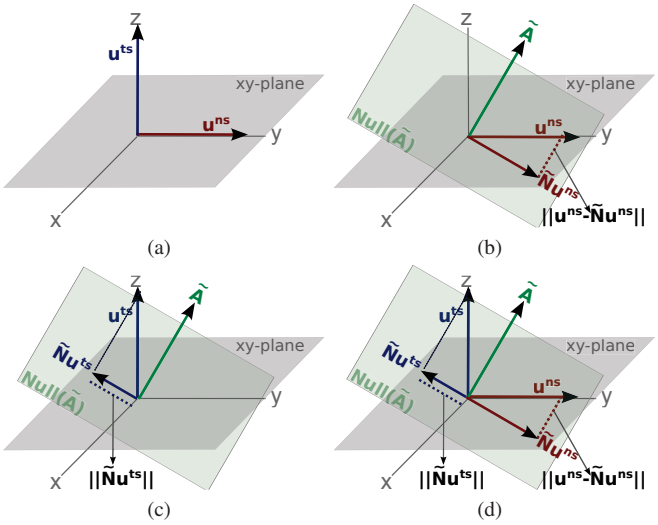

Fig. 2: Visualisation of the objective functions. (a) Example data point with task and null space components plotted. (b) The objective function (8) minimises $||\mathbf{u}^{\text{ns}} - \tilde{\mathbf{N}}\mathbf{u}^{\text{ns}}||$ (red dashed line). (c) Alternately, (9) minimises $||\tilde{\mathbf{N}}\mathbf{u}^{\text{ts}}||$ (blue dashed line). (d) It is proposed to use (10), which minimises the sum of these distances.

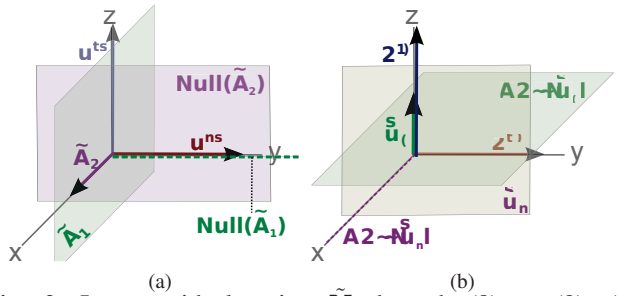

Fig. 3: Issues with learning $\tilde{\mathbf{N}}$ through (8) or (9). (a) Using (8), both $\tilde{\mathbf{A}}_1$ (xz-plane) and $\tilde{\mathbf{A}}_2$ (x-axis) are candidate solutions. Note that, $\tilde{\mathbf{A}}_2$ is incorrect since $\tilde{\mathbf{N}}_2\mathbf{u}^{\text{ts}} = \mathbf{u}^{\text{ts}}$, but this is not evident from the error measure. (b) Using (9) instead, $\tilde{\mathbf{A}}_1$ (on the z-axis) and $\tilde{\mathbf{A}}_2$ (the yz-plane) are candidate solutions. However, $\tilde{\mathbf{A}}_2$ is wrong because $\tilde{\mathbf{N}}_2\mathbf{u}^{\text{ns}} = \mathbf{0}$.

C. Ensuring Correct Rank

Consider again the example data point illustrated in Fig. 2a, with $\mathbf{u}^{\text{ts}}$ (blue) parallel to the z-axis and $\mathbf{u}^{\text{ns}}$ (red) parallel to the y-axis. Minimising (8) or (9) in isolation for this data point has subtly different results.

1) Minimising (8): Recall that (8) relies on finding $\tilde{\mathbf{N}}$ such that $\mathbf{u}^{\text{ns}}$ lies in the image space of $\tilde{\mathbf{N}}$, or $\mathbf{u}^{\text{ns}} \in Image(\tilde{\mathbf{N}})$. However, the problem is that the superset of $\mathbf{N}$ also minimises this error measure. Let $\tilde{\mathbf{N}}'$ be a superset of $\tilde{\mathbf{N}}$ such that $\tilde{\mathbf{N}} \subset \tilde{\mathbf{N}}'$. Since $\mathbf{u}^{\text{ns}} \in Image(\tilde{\mathbf{N}})$ and $\tilde{\mathbf{N}} \subset \tilde{\mathbf{N}}'$, then it is also true that $\mathbf{u}^{\text{ns}} \in Image(\tilde{\mathbf{N}}')$. In other words, the rank of $\tilde{\mathbf{N}}$ can be *overestimated* by minimising (8).

This is visualised in Fig. 3a. There, two candidate solutions are (i) $\tilde{\mathbf{A}}_1$ (xz-plane), with the y-axis as the null space, and (ii) $\tilde{\mathbf{A}}_2$ (x-axis) with the yz-plane as null space. Note that both minimise (8) exactly, so without prior knowledge of the task-space nor dimensionality $\mathcal{S}$, it is hard to decide which one is the better estimate. If the data contains rich enough variations in $\mathbf{u}^{\text{ns}}$ such that it spans $\mathbb{R}^{\mathcal{Q}-\mathcal{S}}$, then the $\tilde{\mathbf{N}}$ with the lowest rank, can heuristically be chosen, but this may be hard to verify in real world problems. Meanwhile, note that $\tilde{\mathbf{A}}_2$ is not a correct solution since $\tilde{\mathbf{N}}_2\mathbf{u}^{\text{ts}} = \mathbf{u}^{\text{ts}}$.

2) Minimising (9): The converse problem arises by optimising (9) instead. By definition, there exists a solution $\tilde{\mathbf{N}}$ in $\mathbb{R}^{\mathcal{Q}-\mathcal{S}}$ such that $\tilde{\mathbf{N}}\mathbf{u}^{\text{ts}} = \mathbf{0}$. However, note also that there also exist subspaces of $\tilde{\mathbf{N}}$ that satisfy this condition. Specifically, (9) seeks a projection matrix $\tilde{\mathbf{N}}$ such that its image space is orthogonal to $\mathbf{u}^{\text{ts}}$.

The rank of the true solution $\mathbf{N}$ is $\mathcal{Q} - \mathcal{S}$, which implies that the image space of $\mathbf{N}$ can be described by $\mathcal{Q}-\mathcal{S}$ linearly independent row vectors. Note that, if we find a $\tilde{\mathbf{N}}$ such that $\tilde{\mathbf{N}}$ is orthogonal to $\mathbf{u}^{\text{ts}}$, then each row vector of $\tilde{\mathbf{N}}$, and their linear combinations, are also orthogonal to $\mathbf{u}^{\text{ts}}$.

The issue is shown graphically in Fig. 3b. There, candidate solutions $\tilde{\mathbf{A}}_1$ (z-axis) and $\tilde{\mathbf{A}}_2$ (yz-plane) both minimise (9). However, $\tilde{\mathbf{A}}_2$ is wrong because $\tilde{\mathbf{N}}_2\,\mathbf{u}^{\text{ns}} = \mathbf{0}$. The risk here is *underestimating* the rank of $\tilde{\mathbf{N}}$. If it is known that $\mathbf{u}^{\text{ts}}$ spans the task-space, then the $\tilde{\mathbf{N}}$ with the highest rank (or $\tilde{\mathbf{A}}$ with the lowest rank) can be chosen, but this again relies on a heuristic choice.

3) Minimising (10): If the observations are rich enough such that $\mathbf{u}^{\text{ts}}$ spans $\mathbb{R}^{\mathcal{S}}$ and $\mathbf{u}^{\text{ns}}$ spans $\mathbb{R}^{\mathcal{Q}-\mathcal{S}}$, there is a projection matrix $\mathbf{N} \in \mathbb{R}^{(\mathcal{Q}-\mathcal{S})\times\mathcal{Q}}$ such that $\mathbf{N}\mathbf{u}^{\text{ns}} = \mathbf{u}^{\text{ns}}$ and $\mathbf{N}\mathbf{u}^{\text{ts}} = \mathbf{0}$. From the preceding analysis, if $\tilde{\mathbf{N}}\mathbf{u}^{\text{ns}} = \mathbf{u}^{\text{ns}}$ is satisfied, a projection matrix $\tilde{\mathbf{N}}$ such that $\mathbf{N} \subseteq \tilde{\mathbf{N}}$ has been found. Likewise, if $\tilde{\mathbf{N}}\mathbf{u}^{\text{ts}} = \mathbf{0}$, then it is also true that that $\tilde{\mathbf{N}} \subseteq \mathbf{N}$. If both conditions are met, $\tilde{\mathbf{N}} \subseteq \mathbf{N} \subseteq \tilde{\mathbf{N}}$, and the only possibility is $\tilde{\mathbf{N}} = \mathbf{N}$. Therefore, (10) can be applied to ensure that our estimated $\tilde{\mathbf{N}}$ has the correct rank.

D. Algorithms for learning $\mathbf{A}$

Based on the above considerations, in the following, two different algorithms for learning $\mathbf{A}$ are defined.

The first considers a simplified case, where a set of candidate task spaces are given that may appear as rows of $\mathbf{A}$, and the aim is to select which of the candidates best describe the data. For instance, in the glass holding/pouring example described in §II, candidate task spaces may include the hand orientation and its Cartesian position, and it is up to the algorithm to determine that the orientation is key.

The second concerns the more general case, in which no prior knowledge is assumed, so the rows of $\mathbf{A}$ must be estimated with a non-linear function approximator, see §III-D.2.

1) Learning $\mathbf{A}$ with candidate rows: Consider the case where the constraint matrix can be written as

$$\mathbf{A}_n = \mathbf{\Lambda}\mathbf{\Phi}_n \tag{11}$$

where $\mathbf{\Phi}_n := \mathbf{\Phi}(\mathbf{x}_n)$ is a feature matrix whose rows are candidates for the rows that occur in the true $\mathbf{A}$. In the pouring example (see §II), for instance, one may choose $\mathbf{\Phi}(\mathbf{x}) = \mathbf{J}(\mathbf{x})$, where $\mathbf{J}$ is the Jacobian mapping from joint space to end-effector space. $\mathbf{\Lambda} \in \mathbb{R}^{\mathcal{S}\times\mathcal{S}}$ is a selection matrix specifying which of these represent valid constraints (*i.e.,* $\mathbf{\Lambda}_{s,s} = 1$ if the s^{th} row of $\mathbf{\Phi}$ is contained in $\mathbf{A}$) and which should be discarded $\mathbf{\Lambda}_{s,s} = 0$).

Algorithm 1 Learning $\mathbf{A}$ with candidate features

Input:
$\mathbf{x}$: observed states
$\mathbf{\Phi}$: feature matrix containing candidate rows of $\mathbf{A}$
$\mathbf{u}$: observed actions
Output:
$\tilde{\mathbf{\Lambda}}$: the estimated selection matrix
1: Set $\tilde{\mathbf{\Lambda}} \leftarrow \emptyset$ and $s \leftarrow 1$
2: Learn $\boldsymbol{\lambda}_s^*$ by minimising (12)
3: **while** Adding $\boldsymbol{\lambda}_s^*$ to $\tilde{\mathbf{\Lambda}}$ does not increase (10) **do**
4: Set $\tilde{\mathbf{\Lambda}} \leftarrow [\boldsymbol{\lambda}_1^*, \cdots, \boldsymbol{\lambda}_s^*]^\top$ and $s \leftarrow s+1$
5: Learn $\boldsymbol{\lambda}_s^*$ by (12) such that $\boldsymbol{\lambda}_s^* \perp \boldsymbol{\lambda}_j^* \ \forall j < s$
6: **end while**
7: Return $\tilde{\mathbf{\Lambda}}$

From [13], the objective function in (8) can be written as $||\tilde{\mathbf{u}}_n^{\text{ns}} - \tilde{\mathbf{N}}_n \tilde{\mathbf{u}}_n^{\text{ns}}||^2 = (\tilde{\mathbf{u}}_n^{\text{ns}})^\top \tilde{\mathbf{A}}_n^\dagger \tilde{\mathbf{A}}_n \tilde{\mathbf{u}}_n^{\text{ns}}$. Substituting $\mathbf{A}_n = \mathbf{\Lambda}\mathbf{\Phi}_n$, (8) can be written as

$$E[\tilde{\mathbf{N}}] = \sum_{n=1}^{\mathcal{N}} (\tilde{\mathbf{u}}_n^{\text{ns}})^\top (\mathbf{\Lambda}\mathbf{\Phi}_n)^\dagger (\mathbf{\Lambda}\mathbf{\Phi}_n) \tilde{\mathbf{u}}_n^{\text{ns}}. \tag{12}$$

Choosing $\mathbf{\Lambda}$ such that it is described by a set of $\mathcal{S}$ orthonormal vectors $\mathbf{\Lambda} = \left[\boldsymbol{\lambda}_1^\top, \boldsymbol{\lambda}_2^\top, \ldots, \boldsymbol{\lambda}_{\mathcal{S}}^\top\right]^\top$ where $\boldsymbol{\lambda}_s \in \mathbb{R}^{\mathcal{S}}$ corresponds to the s^{th} constraint and $\boldsymbol{\lambda}_i \perp \boldsymbol{\lambda}_j$ for $i \neq j$, the optimal $\mathbf{\Lambda}$ can be formed by iteratively searching the choice of $\boldsymbol{\lambda}_s$ that minimises (12).

Following [13], an unit vector $\hat{\mathbf{a}} = (a_1, a_2, \cdots, a_{\mathcal{Q}})$ with an arbitrary dimension $\mathcal{Q}$ can be represented by $\mathcal{Q}-1$ parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \cdots, \theta_{\mathcal{Q}-1})^\top$ where

$$\begin{aligned} a_1 &= \cos\theta_1, \ a_2 = \sin\theta_1 \cos\theta_2, \\ a_3 &= \sin\theta_1 \sin\theta_2 \cos\theta_3 \ldots \\ a_{\mathcal{Q}-1} &= \prod_{q=1}^{\mathcal{Q}-2} \sin\theta_q \cos\theta_{\mathcal{Q}-1}, \ a_{\mathcal{Q}} = \prod_{q=1}^{\mathcal{Q}-1} \sin\theta_q \end{aligned} \tag{13}$$

Using the formulation above, each $\boldsymbol{\lambda}_s$ can be represented by parameters $\boldsymbol{\theta}_s \in \mathbb{R}^{\mathcal{S}-1}$. To form an estimate of $\boldsymbol{\lambda}_s$, we model $\boldsymbol{\theta}_s(\mathbf{x}_n) = \mathbf{W}_s^\top \boldsymbol{\beta}(\mathbf{x}_n)$ where $\mathbf{W}_s$ is a matrix of weights, and $\boldsymbol{\beta}(\mathbf{x}_n) \in \mathbb{R}^{\mathcal{S}}$ is a vector of $\mathcal{S}$ kernel functions. In the experiments reported in this paper, Gaussian radial basis functions (RBFs) $\beta_m(\mathbf{x}_n) = \frac{K(\mathbf{x}_n - \mathbf{c}_m)}{\sum_{i=1}^{\mathcal{M}} K(\mathbf{x}_n - \mathbf{c}_i)}$ are used, where $K(.)$ denotes the Gaussian kernel and $\mathbf{c}_m$ are $\mathcal{S}$ predetermined centres chosen according to k-means.

Note that, estimating $\boldsymbol{\lambda}_s$ is a non-linear least squares problem, which cannot easily be solved in closed form. In the evaluations, the Levenberg-Marquardt algorithm [14], a numerical optimisation technique, is used to find the optimal $\boldsymbol{\lambda}_s$ that minimises (12). In this, $\boldsymbol{\lambda}_{s+1}$ is added only if it does not reduce the fit under (10). The process is summarised in Algorithm 1.

2) Learning $\mathbf{A}$ in absence of prior knowledge: If no prior knowledge about the rows of $\mathbf{A}$ is available, an alternative is to estimate the constraints directly. In this, it is assumed that $\mathbf{A}_n$ is formed from a set of $\mathcal{S}$ unit vectors $\mathbf{A}_n = [\mathbf{a}_1(\mathbf{x}_n)^\top, \mathbf{a}_2(\mathbf{x}_n)^\top, ..., \mathbf{a}_{\mathcal{S}}(\mathbf{x}_n)^\top]^\top$ where $\mathbf{a}_s$ corresponds to the s^{th} constraint and $\mathbf{a}_i \perp \mathbf{a}_j$ for all $i \neq j$. Similar to the approach for learning $\boldsymbol{\lambda}_s$, an iterative approach is taken,

Algorithm 2 Learning $\mathbf{A}$ with a function approximator

Input:
$\mathbf{x}$: observed states
$\mathbf{u}$: observed actions
Output:
$\tilde{\mathbf{A}}$: the estimated constraint matrix
1: Decompose $\mathbf{u}$ into $\tilde{\mathbf{u}}^{\text{ts}}$ and $\tilde{\mathbf{u}}^{\text{ns}}$ by (6)
2: Set $\tilde{\mathbf{A}} \leftarrow \emptyset$ and $s \leftarrow 1$
3: Learn $\mathbf{a}_1^*$ by minimising (8)
4: **while** Adding $\mathbf{a}_s^*$ to $\tilde{\mathbf{A}}$ does not increase (10) **do**
5: Set $\tilde{\mathbf{A}} \leftarrow [\mathbf{a}_1^*, \cdots, \mathbf{a}_s^*]^\top$ and $s \leftarrow s+1$
6: Learn $\mathbf{a}_s^*$ by (8) such that $\mathbf{a}_s^* \perp \mathbf{a}_j^* \ \forall j < s$
7: **end while**
8: Return $\tilde{\mathbf{A}}$

where the s^{th} constraint vector $\mathbf{a}_s$ is learnt by optimising (8), and $\mathbf{a}_s$ is added only if it does not reduce the fit under (10). The process is summarised in Algorithm 2.

E. Evaluation Criteria

For testing the performance of learning, the following evaluation criteria may be defined.

1) Normalised Projected Policy Error: This error measure measures the difference between the policy subject to the true constraints, and that of the policy subject to the estimated constraints. Formally, the *normalised projected policy error* (NPPE) can be defined as

$$E_{PPE} = \frac{1}{\mathcal{N}\sigma_{\mathbf{u}}^2} \sum_{n=1}^{\mathcal{N}} ||\mathbf{N}_n \boldsymbol{\pi}_n - \tilde{\mathbf{N}}_n \boldsymbol{\pi}_n||^2 \tag{14}$$

where $\mathcal{N}$ is the number of data points, $\boldsymbol{\pi}_n$ are samples of the policy, and $\mathbf{N}$ and $\tilde{\mathbf{N}}$ are the true and learnt projection matrices, respectively. The error is normalised by the variance of the observations $\sigma_{\mathbf{u}}^2$.

2) Normalised Projected Observation Error: To evaluate the fit of $\tilde{\mathbf{N}}$ under the objective function (10), the *normalised projected observation error* (NPOE) may be used, defined as

$$E_{POE} = \frac{1}{\mathcal{N}\sigma_{\mathbf{u}}^2} \sum_{n=1}^{\mathcal{N}} ||\mathbf{u}_n^{\text{ns}} - \tilde{\mathbf{N}}_n \mathbf{u}_n^{\text{ns}}||^2 + ||\tilde{\mathbf{N}}_n \mathbf{u}_n^{\text{ts}}||^2. \tag{15}$$

The latter only reaches zero if the model exactly satisfies (10).

3) Normalised Null Space Component Error: Since the quality of the fit depends on the accuracy of $\tilde{\mathbf{u}}^{\text{ns}}$, it is also illustrative to look at the *normalised null space component error* (NNCE),

$$E_{NCE} = \frac{1}{\mathcal{N}\sigma_{\mathbf{u}}^2} \sum_{n=1}^{\mathcal{N}} ||\mathbf{u}_n^{\text{ns}} - \tilde{\mathbf{u}}_n^{\text{ns}}||^2. \tag{16}$$

This measures the distance between the true and the learnt null space components $\mathbf{u}^{\text{ns}}$ and $\tilde{\mathbf{u}}^{\text{ns}}$, respectively.

IV. Evaluation

In this section, the proposed approach is evaluated for extracting the task space from systems of policies controlled hierarchically according to the operational space formulation.

$\boldsymbol{\pi}$	NNCE	NPPE	NPOE
Linear	$\sim 10^{-7}$	$\sim 10^{-9}$	$\sim 10^{-9}$
Limit-cycle	0.08 ± 0.02	0.001 ± 0.002	0.001 ± 0.002
Sinusoidal	5.26 ± 4.46	0.011 ± 0.017	0.014 ± 0.021

TABLE I: NNCE, NPPE, and NPOE (mean±s.d.)$\times 10^{-2}$ over 50 trials when learning from data from different null space policies.

A. Toy Example

In this experiment, the proposed approach is applied to the problem of learning the constraint matrix from data from a simple, two-dimensional system with a one-dimensional task space. The aim is to find out if the task space can be correctly identified in face of 'distractions' caused by having different policies executing movements in the null space.

The task space is defined by the constraint matrix $\mathbf{A} = \mathbf{a} \in \mathbb{R}^{1\times 2}$, meaning that task space movements occur in the direction of the unit vector $\hat{\mathbf{a}}$. To ensure the robustness of the results, the latter is drawn uniform-randomly $\theta \sim \mathcal{U}(0, \pi]\, rad$ at the start of each experiment.

For simplicity, the task space policy is a linear point attractor $\mathbf{b}(\mathbf{x}) = \beta^{ts}(r^* - r)$ where r defines the position in task space, r^* is the target point and $\beta^{ts} = 0.1$ is a scaling factor. To simulate variable tasks performed in the same task space, the task space target was drawn randomly $r^* \sim \mathcal{U}[-2, 2]$ for each data point.

Running simultaneously with the task space policy, a secondary policy is executed in the null space of the constraints. In the following, three different null space policies $\boldsymbol{\pi}$ are considered, namely (i) a *linear policy*, $\boldsymbol{\pi} = -\mathbf{L}\bar{\mathbf{x}}$ where $\bar{\mathbf{x}} := (\mathbf{x}^\top, 1)^\top$ and $\mathbf{L} = ((2, 4, 0)^\top, (1, 3, -1)^\top)^\top$, (ii) a *limit-cycle policy*, $\dot{\rho} = \rho(\rho_0 - \rho^2)$ with radius $\rho_0 = 0.75\, m$, angular velocity $\dot{\phi} = 1\, rad/s$, where ρ and ϕ are the polar representation of the state, *i.e.,* $\mathbf{x} = (\rho\cos\phi, \rho\sin\phi)^\top$, and (iii) a *sinusoidal policy*, $\boldsymbol{\pi} = (\cos z_1 \cos z_2, -\sin z_1 \sin z_2)^\top$ with $z_1 = \pi x_1$ and $z_2 = \pi(x_2 + \frac{1}{2})$. The training data consists of 150 data points, drawn randomly across the space $(\mathbf{x})_i \sim \mathcal{U}(-1, 1)$, $i \in \{1, 2\}$. For testing, a further 150 data points are used, generated through the same procedure.

For learning, the null space component is modelled as $\tilde{\mathbf{u}}^{\text{ns}} = \mathbf{W}_{ns}^\top \boldsymbol{\beta}(\mathbf{x})$ where $\boldsymbol{\beta}$ is a vector of $\mathcal{M} = 16$ Gaussian RBFs arranged on a 4×4 grid, with widths set according to distance between the basis functions. The parameters $\mathbf{W}_{ns}$ are learnt through minimisation of the objective function (6).

The resulting $\tilde{\mathbf{u}}^{\text{ns}}$ is used for learning the constraint matrix $\tilde{\mathbf{A}}$. For this, each row of $\tilde{\mathbf{A}}$ is represented by another parametric model $\boldsymbol{\theta}_s(\mathbf{x}_n) = \mathbf{W}_{\mathbf{a},s}^\top \boldsymbol{\beta}(\mathbf{x}_n)$ where $\boldsymbol{\beta}$ is $\mathcal{M} = 16$ RBFs and $\mathbf{W}_{\mathbf{a},s}$ are learnt through the method outline in §III-D.2. The experiment is repeated 50 times.

Table I summarises NPPE, NPOE, and NNCE ((14)-(16)) when learning from data containing the different null space policies. Looking at the NPPE and NPOE, it can be seen that a good approximation of $\mathbf{A}$ is learnt in all cases with normalised errors $< 10^{-4}$.

To further characterise the performance of the proposed approach, the experiment was repeated while varying the size of the input data for the limit-cycle policy for $5 < \mathcal{N} < 250$ data points. The results (in log scale) over 50 trials are plotted in Fig. 4 (left). It can be seen that the NPPE, NPOE, and NNCE rapidly decrease as the number of input data

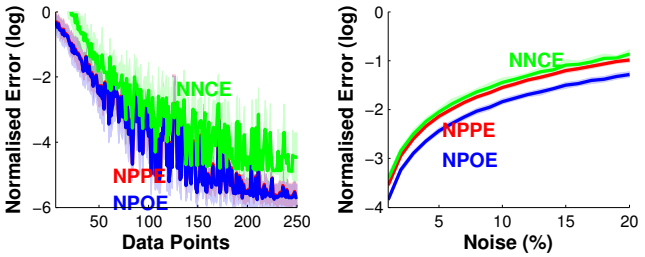

Fig. 4: NNCE, NPPE, and NPOE for increasing number of data points (left) and increasing noise levels in the observed $\mathbf{u}$ (right). Curves are mean±s.d. over 50 trials.

increases. This is to be expected, since larger data sets contain richer variations in $\mathbf{u}_n^{\text{ts}}$ resulting in more accurate estimates of $\tilde{\mathbf{u}}^{\text{ns}}$ and $\tilde{\mathbf{A}}$ (equivalently, $\tilde{\mathbf{N}}$). Note that, even with a relatively small data set ($\mathcal{N} < 50$), the error is still low ($\approx 10^{-3}$).

To assess the effect of noise, the experiment was repeated with different levels of noise in the training data. For this, the limit-cycle policy data was contaminated with Gaussian noise, the scale of which varied to match up to 20% of the variance of the data. The resulting NNCE, NPPE, and NPOE follows the noise level, as plotted in Fig. 4 (right). However, the error is still relatively low (NPPE $< 10^{-2}$), even when the noise is as high as 5% of the variance of the data.

B. Three Link Planar Arm

The goal of the second experiment is to assess the performance of the proposed approach for more complex, non-linear constraints. For this, constrained motion data from a kinematic simulation of a planar three link arm is used.

The set up is as follows. The state and action spaces of the arm are described by the joint angles $\mathbf{x} := \mathbf{q} \in \mathbb{R}^3$ and the joint velocities $\mathbf{u} := \dot{\mathbf{q}} \in \mathbb{R}^3$. The task space is described by $\mathbf{r} = (r_x, r_z, r_\psi)^\top$ where r_x and r_z specify the end-effector position and r_ψ its orientation.

Joint space motion is recorded as the arm performs tasks in different end-effector spaces. As discussed in §III-D.1, the task space constraint matrix at state $\mathbf{x}_n$ is described as

$$\mathbf{A}_n = \boldsymbol{\Lambda}\boldsymbol{\Phi}_n \tag{17}$$

where $\boldsymbol{\Phi}_n = \mathbf{J}(\mathbf{x}_n) \in \mathbb{R}^{3\times 3}$, the manipulator Jacobian, and $\boldsymbol{\Lambda} \in \mathbb{R}^{3\times 3}$ is the selection matrix specifying the coordinates to be controlled in the task. In the following, the task is performed in the subspaces of end-effector space defined by three different constraint systems:

1) $\boldsymbol{\Lambda}_{(x,z)} = ((1, 0, 0)^\top, (0, 1, 0)^\top, (0, 0, 0)^\top)^\top$,
2) $\boldsymbol{\Lambda}_{(x,\psi)} = ((1, 0, 0)^\top, (0, 0, 0)^\top, (0, 0, 1)^\top)^\top$, and
3) $\boldsymbol{\Lambda}_{(z,\psi)} = ((0, 0, 0)^\top, (0, 1, 0)^\top, (0, 0, 1)^\top)^\top$.

Choosing $\boldsymbol{\Lambda}_{(x,z)}$, for example, means that the the end-effector position coordinates (x, z) are controlled by the task space policy $\mathbf{b}$, while the the orientation is allowed to follow the null space policy $\boldsymbol{\pi}$.

In this experiment, the task space policy is a linear policy tracking a task space target $\mathbf{r}^*$, that is drawn uniform randomly ($r_x^* \sim \mathcal{U}[-1, 1]$, $r_z^* \sim \mathcal{U}[0, 2]$, $r_\psi^* \sim \mathcal{U}[0, \pi]$). Acting simultaneously, but in the null space, is a second policy $\boldsymbol{\pi} = -\mathbf{L}(\mathbf{q} - \mathbf{q}^*)$, where $\mathbf{q}^* = \mathbf{0}$ represents a 'comfort posture' away from joint limits and $\mathbf{L} = \mathbf{I} \in \mathbb{R}^{3\times 3}$.

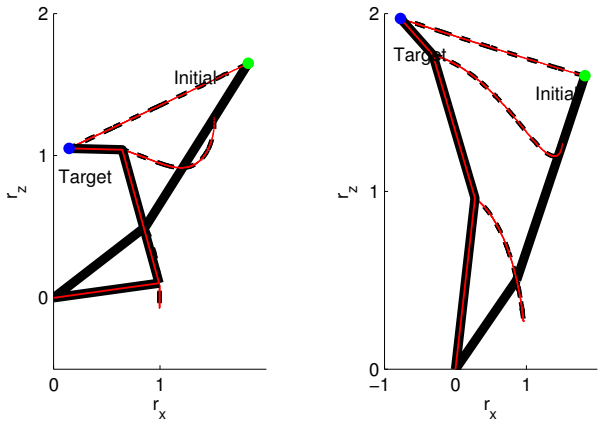

Fig. 5: Example trajectories generated under the true $\mathbf{A}$ (black) and the learnt $\tilde{\mathbf{A}}$ (red), using the policy $\mathbf{b}$ from the training data (left), and a new, previously unseen task space policy $\mathbf{b}'$ (right).

	Constraint	NNCE	Method	NPPE	NPOE
1	$\mathbf{\Lambda}_{x,z}$	0.32 ± 1.07	Learn $\mathbf{a}$	3.54 ± 2.29	0.06 ± 0.04
2			Learn $\boldsymbol{\lambda}$	0.30 ± 0.44	0.04 ± 0.04
3	$\mathbf{\Lambda}_{x,\psi}$	3.89 ± 1.10	Learn $\mathbf{a}$	7.79 ± 12.0	2.14 ± 6.48
4			Learn $\boldsymbol{\lambda}$	0.32 ± 1.42	2.53 ± 1.11
5	$\mathbf{\Lambda}_{z,\psi}$	0.61 ± 0.60	Learn $\mathbf{a}$	2.80 ± 2.59	0.43 ± 0.24
6			Learn $\boldsymbol{\lambda}$	0.24 ± 0.51	0.14 ± 0.25

TABLE II: NNCE, NPPE, and NPOE (mean$\pm$ s.d.)$\times 10^{-2}$ for each constraint over 50 trials.

Arm motion data is collected in the form of trajectories, each from a uniform random start state $q_1 \sim \mathcal{U}[0^\circ, 10^\circ]$, $q_2 \sim \mathcal{U}[90^\circ, 100^\circ]$, $q_3 \sim \mathcal{U}[0^\circ, 10^\circ]$. As training data, 50 trajectories each of length 50 steps are generated for each choice of $\mathbf{\Lambda}$. Following the same procedure, another 50 trajectories are also generated as unseen testing data. Fig. 5 (left) shows an example trajectory with constraint $\mathbf{\Lambda}_{x,z}$ and task space target $\mathbf{r}^* = (1, 0)^\top$.

For learning, the null space component $\tilde{\mathbf{u}}^{\text{ns}}$ is estimated in form of a parametric model $\tilde{\mathbf{u}}^{\text{ns}} = \mathbf{W}_{ns}^\top \boldsymbol{\beta}(\mathbf{x})$, where $\boldsymbol{\beta}$ is a vector of $\mathcal{M} = 100$ Gaussian RBFs. In this experiment, the centres are chosen according to k-means and the widths are taken as the mean distance between the centres. The parameters $\mathbf{W}_{ns}$ are learnt by minimising (6).

The learnt $\tilde{\mathbf{u}}^{\text{ns}}$ is then used to learn the constraint through the methods outlined in §III-D.1 and §III-D.2. For learning $\tilde{\mathbf{\Lambda}}$, each row of $\tilde{\mathbf{\Lambda}}$ is modelled as $\boldsymbol{\theta}_{\lambda,s}(\mathbf{x}_n) = \mathbf{W}_{\lambda,s}^\top \boldsymbol{\beta}(\mathbf{x}_n)$ where $\boldsymbol{\beta}(\mathbf{x}_n)$ consists of $\mathcal{M} = 16$ Gaussian RBFs and $\mathbf{W}_{\lambda,s}$ is estimated through minimising (12). For learning $\tilde{\mathbf{A}}$, the constraint vector is represented by another parametric model $\boldsymbol{\theta}_{\mathbf{a},s}(\mathbf{x}_n) = \mathbf{W}_{\mathbf{a},s}^\top \boldsymbol{\beta}(\mathbf{x}_n)$ where $\mathcal{M} = 50$ and the weights $\mathbf{W}_{\mathbf{a},s}$ are learnt by optimising (8).

Looking at the NPPE and NPOE in Table II, a good approximation of the constraint matrix is learnt for each of the constraints, with errors of order 10^{-2} or lower in all cases. In the case that the Jacobian $\mathbf{J}(\mathbf{x})$ is assumed known *a priori* and only the selection matrix $\mathbf{\Lambda}$ is learnt (rows 2, 4 and 6) the errors are an order of magnitude lower, reflecting the fact that the algorithm exploits this prior knowledge. Nevertheless, in absence of this information (rows 1, 3 and 5) the algorithm is still able to learn a good approximation of the non-linear, state-dependent constraint matrix (17).

To further examine the accuracy, in Fig. 5 (left) the trajectories generated with the learnt model (red) are overlaid on those generated with the ground truth, for the three task spaces, using the same task and null space policies. As can be seen, the trajectories generated with the learnt constraints match the true trajectories extremely well.

As discussed in §II, one of the strengths of the proposed framework is that, once the constraints have been estimated, new controllers can be applied in the same task space to generalise behaviour to new situations. To test this, a new task space policy $\mathbf{b}'$ with target $\mathbf{r}' = (-1, 2)^\top$ was used to generate a trajectory under (i) the learnt constraints (*i.e.,* $\tilde{\mathbf{A}}^\dagger \mathbf{b}' + \tilde{\mathbf{N}}\boldsymbol{\pi}$) and (ii) the true constraints (*i.e.,* $\mathbf{A}^\dagger \mathbf{b}' + \mathbf{N}\boldsymbol{\pi}$). The trajectory is shown in Fig. 5 (right), where the former (red) is overlaid on the latter (black). A close match is seen between the predicted behaviour under the true and learnt constraints, indicating good generalisation in predicting behaviour that is not present in the training data.

C. AR10 Humanoid Hand

The final experiment aims to verify the proposed approach on a physical robotic system, for a real world task. The experimental scenario chosen is the manual operation of the remote control of an air conditioning unit with the AR10 Robotic Hand (Fig. 1) [15].

As discussed in §I, due to the large configuration space associated with dexterous hands (the AR10 has a total of 10 degrees of freedom), demonstration data may contain movement features from many different control policies running concurrently (*e.g.,* gripping, button pressing, transmitter orientation, posture adjustments for comfort). For simplicity, here, a restricted form of this problem is considered, whereby the task space part of the movement corresponds to pressing the temperature control button with the thumb, while the null space movements consist of moving the middle finger to reach a comfort posture. The aim is to extract the task relevant part of the movement (thumb movement) in face of the distracting secondary finger movements, and from this determine the task space (thumb tip position) to enable generalisation to other button pressing movements.

The experimental setup is illustrated in Fig. 6. The state $\mathbf{x} := \mathbf{q} \in \mathbb{R}^4$ consists of the joint angles of the thumb and the middle finger of the robot, and the command $\mathbf{u} := \dot{\mathbf{q}} \in \mathbb{R}^4$ is the joint velocity vector. As a ground truth, the task space is defined as the thumb tip position, corresponding to constraint matrix $\mathbf{A}(\mathbf{x}) = \mathbf{\Lambda}\mathbf{\Phi}(\mathbf{x}) \in \mathbb{R}^{3\times 4}$ where $\mathbf{\Phi}(\mathbf{x}) := \mathbf{J}(\mathbf{x}) \in \mathbb{R}^{6\times 4}$ is the Jacobian that relates the joint velocities to the tip velocities. of the thumb and the middle finger, and $\mathbf{\Lambda} \in \mathbb{R}^{3\times 6}$ is the selection matrix that selects the rows of $\mathbf{J}$ corresponding to the x, y, z position of the thumb tip.

As training data, 100 trajectories are collected, in which the thumb is used manipulate the remote control. The initial position for each trajectory is drawn randomly within its joint limit $q_1 \sim \mathcal{U}[-115^\circ, -45^\circ]$, $q_2 \sim \mathcal{U}[20^\circ, 120^\circ]$, $q_3 \sim \mathcal{U}[0^\circ, 80^\circ]$, $q_4 \sim \mathcal{U}[-100^\circ, 60^\circ]$.

For the task-space policy, the task-space target $\mathbf{r}^*$ is set so that the tip of the thumb presses a random position on the remote control, with the speed randomly varied in each

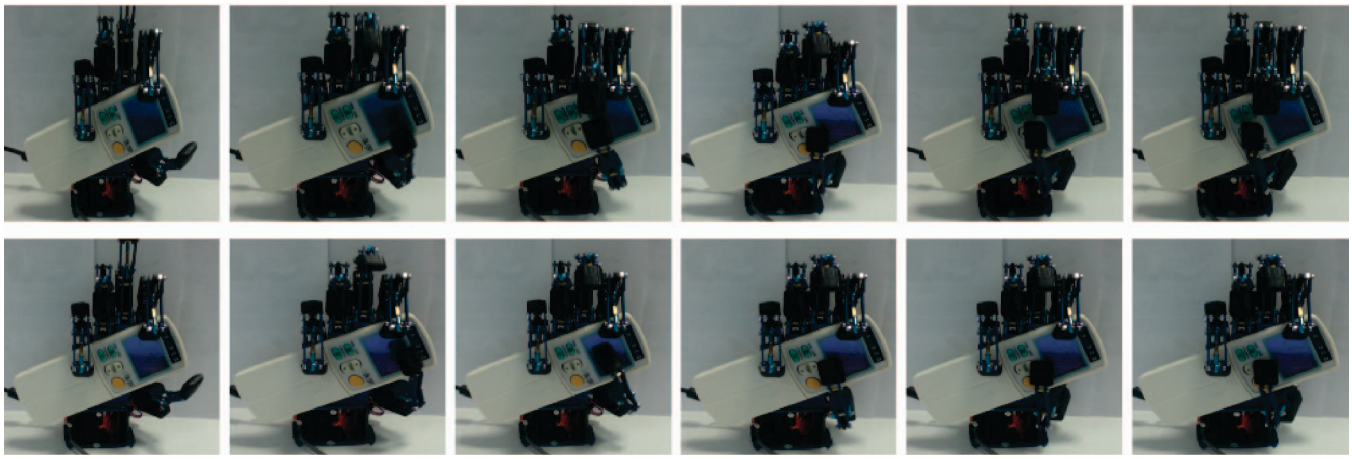

Fig. 6: AR10 hand experiment. The task space is learnt from demonstrations containing thumb movement to decrease the temperature (top row). In testing, the behaviour is generalised to enable temperature increase, by applying a new task space policy using the learnt task space, corresponding to pressing a different button (bottom row).

trajectory. Simultaneously, the null space policy moves the middle finger to a comfort posture, in which the finger rests on the top of the remote control. The procedure is repeated to collect 100 trajectories of test data.

For learning, the null space component is represented by $\tilde{\mathbf{u}}^{\text{ns}} = \mathbf{W}_{ns}^{\top}\boldsymbol{\beta}(\mathbf{x})$ where $\boldsymbol{\beta}$ consists of $\mathcal{M} = 50$ Gaussian RBFs, and $\mathbf{W}_{ns}$ is a matrix of weights. The setup of the RBFs is similar to the last experiment where the centres are chosen according to k-means, and the widths are taken as the mean distance between the centres. The parameters $\mathbf{W}_{ns}$ are learnt by minimising (6).

Following the method proposed in §III-D.2, the resulting $\tilde{\mathbf{u}}^{\text{ns}}$ is used to form the task constraint. For learning $\tilde{\mathbf{A}}$, each row in $\tilde{\mathbf{A}}$ is represented by parametric model $\boldsymbol{\theta}_s(\mathbf{x}_n) = \mathbf{W}_{\mathbf{a},s}^{\top}\boldsymbol{\beta}(\mathbf{x}_n)$ where $\mathcal{M} = 50$ and the weights $\mathbf{W}_{\mathbf{a},s}$ are learnt by optimising (8).

The learnt constraint matrix is then applied for generalisation by replacing the original task space policy with a new, unseen one, in which the task space target corresponds to the thumb tip hitting another button to increase the temperature. An example of the outcome is shown in Fig. 6 and in the supplementary video.

V. Conclusion

In this paper, a method for learning the self-imposed constraints from movement observations is proposed. The problem is formulated into an operational space control framework, and the aim is to estimate the constraint matrix that define the task space of a movement. The proposed method can approximate these matrices in the absence of any prior knowledge of the dimensionality of the constraints or the underlying movement policies.

The effectiveness of the approach has been demonstrated on simulated data with different dimensionality, and with different degrees of non-linearity. The method has also been validated on the AR10 Robotic Hand performing manual operation of the remote control of an air conditioning unit.

Future research includes extending the proposed method on robots with higher degree of freedom and improving the efficiency through iterative learning approaches.

Acknowledgement

This paper was partially supported by the European Commission, within the CogIMon project in the Horizon 2020 Work Programme (ICT-23-2014, grant agreement 644727), and partially supported by the EPSRC Grant Agreement No. EP/P511171/1 and EP/P010202/1

References

[1] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE J. Robotics & Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[2] M. Gienger, H. Janssen, and C. Goerick, "Task-oriented whole body motion for humanoid robots," *2005 5th IEEE-RAS Int. Conf. Humanoid Robots*, pp. 238–244, 2005.

[3] O. Khatib, L. Sentis, J. Park, and J. Warren, "Whole-body dynamic behavior and control of human-like robots," *Int. J. Humanoid Robotics*, pp. 29–43, 2004.

[4] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," *Int. J. Robotics Research*, vol. 27, no. 11-12, pp. 1295–1307, 2008.

[5] L. Sentis and O. Khatib, "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives," *Int. J. Humanoid Robotics*, vol. 2, no. 4, pp. 505–518, 2005.

[6] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, *Robot programming by demonstration*. MIT Press, 2007.

[7] J. Craig, P. Hsu, and S. Sastry, "Adaptive control of mechanical manipulators," *Int. J. Robotics Research*, vol. 6, no. 2, pp. 16–28, 1987.

[8] M. Kawato, "Feedback-error-learning neural network for supervised motor learning," in *Advanced Neural Computers*, R. Eckmiller, Ed. Elsevier, 1990, pp. 365–372.

[9] J. Peters and S. Schaal, "Learning to control in operational space," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 197–212, 2008.

[10] M. Howard, S. Klanke, M. Gienger, C. Goerick, and S. Vijayakumar, "A novel method for learning policies from variable constraint data," *Autonomous Robots*, pp. 105–121, 2009.

[11] C. Towell, M. Howard, and S. Vijayakumar, "Learning nullspace policies," in *IEEE Int. Conf. Intelligent Robots and Systems*, 2010, pp. 241–248.

[12] F. E. Udwadia and R. E. Kalaba, *Analytical dynamics: a new approach*. Cambridge University Press, 2007.

[13] H.-C. Lin, M. Howard, and S. Vijayakumar, "Learning null space projections," in *IEEE Int. Conf. on Robotics and Automation*, 2015, pp. 2613–2619.

[14] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*. Springer, 1978, pp. 105–116.

[15] Active8robots, "AR10 Robot Hand: 10 degree of freedom robot hand with position feedback," http://www.active8robots.com/wp-content/uploads/AR10-Datasheet-rev4.pdf.