ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1. Available from http://www.i6doc.com/en/.

Learning Null Space Projections Fast

Jeevan Manavalan[1] and Matthew Howard[1]*

1- King's College London - Department of Informatics
Strand, Kings College, Strand, London WC2R 2LS - United Kingdom

Abstract. Typically robot interactions with the environment may involve some type of constraint which impedes the motion of the system. This paper proposes an approach to learn kinematic constraints from observed movements. Our method derives the null space projection of a kinematically constrained system using gradient descent. Moreover, we compare this method to the existing brute force-based approach for learning constraints on data sets of different dimensionality, to demonstrate how it can learn constraints from data sets of a much higher dimensionality.

1 Introduction

Robot interactions with the environment tend to involve some type of constraint which may have an impeding factor to the success of a task, for example, when considering environmental constraints, we can see how the surface of a table limits movement of a hand during a wiping task (Fig. 1) [1].
On the other hand, complex interactions can also be derived from self-imposed constraints, such as pouring water from a bottle, as the hand's orientation is restricted such that the water is poured into the glass. Essentially, robotic manipulators are expected to produce a set of joint-space movements which comply with the constraint [3, 4].

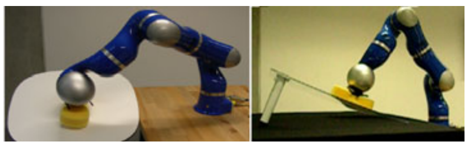
Fig. 1: System restrictions on different environmental constraints [2].

One approach to learning constraints is to focus on extracting relating information from restricted motion [2], particularly where the constraint plays a critical role, such as planning a path through the manipulation of obstacles [5] or maintaining grip on a remote while switching between different buttons [1].

There are numerous approaches to learning constraints from the environment, including vision based systems [6], force/torque and tactile sensors [7] or a combination of both [8]. Recently, systems learn kinematic constraints using proprioception as in [9, 10] and movement observation as proposed by [1].

This paper proposes an alternative approach to [1] to learn the kinematic constraints from observed movements and, in doing so, extends the approach envisioned originally in [2]. Our method derives the null space projection of a kinematically constrained system using gradient descent. Moreover, we compare this method to [2] for learning constraints on data sets of different dimensionality to demonstrate: 1) how it can predict policies faster when dealing with higher dimensional problems, and 2) that it can learn constrained policies from data of a much higher dimensionality, with a smaller demand for computational resources.

*This work was partially supported by the EPSRC Grant Agreement No. EP/P510804/1 and EP/P010202/1

2 Problem Definition

This paper considers the problem of finding a constraint from observed motion data, such as an object or obstacle in the path of a motion that restricts said movement of a system (e.g. a manipulator). We consider systems that are constrained through a set of $\mathcal{S}$-dimensional ($\mathcal{S} \leq \mathcal{Q}$) constraints of the form

$$\mathbf{A}(\mathbf{x},t)\mathbf{y}(\mathbf{x},t) = \mathbf{0} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^{\mathcal{P}}$ represents state, $\mathbf{y} \in \mathbb{R}^{\mathcal{Q}}$ represents the action, t is time, and $\mathbf{A}(\mathbf{x},t) \in \mathbb{R}^{\mathcal{S}\times\mathcal{Q}}$ is a matrix describing the constraints. *The goal is to estimate the constraints described in* $\mathbf{A}$ *only given the observed actions* $\mathbf{y}$.

The relation between $\mathbf{A}$ and $\mathbf{y}$ can be observed by inverting (1), resulting in

$$\mathbf{y}(\mathbf{x},t) = \mathbf{N}(\mathbf{x},t)\boldsymbol{\pi}(\mathbf{x}) \tag{2}$$

where

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

is the null space projection matrix that projects the null space policy $\boldsymbol{\pi}(\mathbf{x})$ onto the null space of $\mathbf{A}$. Here, $\mathbf{I} \in \mathbb{R}^{\mathcal{Q}\times\mathcal{Q}}$ denotes the identity matrix and $\mathbf{A}^{\dagger} = \mathbf{A}^{\top}(\mathbf{A}\mathbf{A}^{\top})^{-1}$ is the Moore-Penrose pseudo-inverse of $\mathbf{A}$.

3 Method

The proposed approach requires no information regarding the dimensionality of the constraint $\mathbf{A}$ or the underlying control policy $\boldsymbol{\pi}$, it only requires the observed motion data $\mathbf{y}$ to learn the constraint matrix $\mathbf{A}$.

The key to the proposed approach is to use properties of the projection matrix $\mathbf{N}$ in order to find $\mathbf{A}$ [2]. By definition $\mathbf{y} = \mathbf{N}\boldsymbol{\pi}$, so $\mathbf{y}$ is the vector $\boldsymbol{\pi}$ projected onto the image space of $\mathbf{N}$. However, it is also the case that *the projection of* $\mathbf{y}$ *also lies in this image space*, i.e.,

$$\mathbf{N}\mathbf{y} = \mathbf{y}. \tag{4}$$

Based on this insight, $\mathbf{N}$ can be approximated by seeking an estimate *such that this condition holds.*

Specifically, given samples $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{\mathcal{N}}$ it is proposed to form an estimate $\tilde{\mathbf{N}}$ that minimises the difference between $\tilde{\mathbf{N}}\mathbf{y}$ and the raw observations $\mathbf{y}$, i.e.,

$$E = \frac{1}{2}\sum_{n=1}^{\mathcal{N}} ||\mathbf{y}_n - \tilde{\mathbf{N}}\mathbf{y}_n||^2. \tag{5}$$

Using (3), the nth term of (5) can be written

$$||\mathbf{y}_n - (\mathbf{I} - \tilde{\mathbf{A}}^{\dagger}\tilde{\mathbf{A}})\mathbf{y}_n||^2 = ||\mathbf{y}_n - \mathbf{y}_n + \tilde{\mathbf{A}}^{\dagger}\tilde{\mathbf{A}}\mathbf{y}_n||^2 = ||\tilde{\mathbf{A}}^{\dagger}\tilde{\mathbf{A}}\mathbf{y}_n||^2. \tag{6}$$

where $\tilde{\mathbf{A}} \in \mathbb{R}^{\mathcal{S}\times\mathcal{Q}}$ is an estimate of the constraint matrix $\mathbf{A}$. Expanding the norm $||\tilde{\mathbf{A}}^{\dagger}\tilde{\mathbf{A}}\mathbf{y}_n||^2 = \mathbf{y}_n^{\top}(\tilde{\mathbf{A}}^{\dagger}\tilde{\mathbf{A}})^{\top}\tilde{\mathbf{A}}^{\dagger}\tilde{\mathbf{A}}\mathbf{y}_n$, and using the identities $(\mathbf{A}^{\dagger}\mathbf{A})^{\top} = \mathbf{A}^{\dagger}\mathbf{A}$ and $\mathbf{A}^{\dagger}\mathbf{A}\mathbf{A}^{\dagger} = \mathbf{A}^{\dagger}$, (5) can be expressed in simplified form [2]

$$E = \frac{1}{2}\sum_{n=1}^{\mathcal{N}} \mathbf{y}_n^{\top}\tilde{\mathbf{A}}^{\dagger}\tilde{\mathbf{A}}\mathbf{y}_n. \tag{7}$$

Defining the data matrix $\mathbf{Y} = (\mathbf{y}_1, \cdots, \mathbf{y}_{\mathcal{N}}) \in \mathbb{R}^{\mathcal{Q}\times\mathcal{N}}$, this can be rewritten

$$E = \frac{1}{2}\mathrm{Tr}[\mathbf{Y}^{\top}\tilde{\mathbf{A}}^{\dagger}\tilde{\mathbf{A}}\mathbf{Y}]. \tag{8}$$

ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1. Available from http://www.i6doc.com/en/.

The constraint projection matrix can therefore be estimated by seeking an estimate $\tilde{\mathbf{A}}$ that minimises (8). Note that, through use of the latter, *no prior knowledge of the underlying* $\boldsymbol{\pi}$*, nor of the true projection matrix* $\mathbf{N}$ *is required.*

3.1 Representation of $\tilde{\mathbf{A}}$

So far, no specific assumptions have been made on the form of the estimated matrix $\tilde{\mathbf{A}}$. Following [2], an appropriate structure of this matrix is outlined, that ensures that the estimate is interpretable in terms of the constraints.

Specifically, $\tilde{\mathbf{A}}$ is assumed to be formed from a set of $\mathcal{S}$ orthonormal vectors

$$\tilde{\mathbf{A}} = \left[\boldsymbol{\alpha}_1^\top\, \boldsymbol{\alpha}_2^\top\, \cdots\, \boldsymbol{\alpha}_{\mathcal{S}}^\top\right]^\top \tag{9}$$

where $\boldsymbol{\alpha}_s = (\alpha_{s,1}, \alpha_{s,2}, ..., \alpha_{s,\mathcal{Q}})$ corresponds to the sth constraint in the observations. The latter are represented by a total of $\mathcal{P} = \mathcal{S}(2\mathcal{Q} - \mathcal{S} - 1)/2$ parameters[1] $\boldsymbol{\theta} = (\theta_1, \theta_2, \cdots, \theta_{\mathcal{P}})^\top$.

Since $\tilde{\mathbf{A}}$ consists of orthonormal row vectors, $(\tilde{\mathbf{A}}\tilde{\mathbf{A}}^\top)^{-1} = \mathbf{I}^{-1} = \mathbf{I}$. So the pseudoinverse here simplifies to $\tilde{\mathbf{A}}^\dagger = \tilde{\mathbf{A}}^\top$. Substituting into (8) yields

$$E = \frac{1}{2}\mathrm{Tr}[\mathbf{Y}^\top \tilde{\mathbf{A}}^\top \tilde{\mathbf{A}} \mathbf{Y}]. \tag{10}$$

To visualise a simple case of the equations defined, we can consider a 1D constraint in a system with two degrees of freedom where $\mathbf{A} = \hat{\boldsymbol{\alpha}} \in \mathbb{R}^{1\times 2}$ and $\hat{\boldsymbol{\alpha}}$ is a unit vector representation of $\boldsymbol{\alpha}$ (Fig. 2) [2]. Figure 2 shows how the $\boldsymbol{\theta}$ corresponds to different estimates of the constraint $\hat{\boldsymbol{\alpha}}$ as well as its associated null-space $\mathbf{N}$. $\hat{\boldsymbol{\alpha}}$ is simply a unit vector with the form $\hat{\boldsymbol{\alpha}} = (\cos\theta, \sin\theta)$ [2].

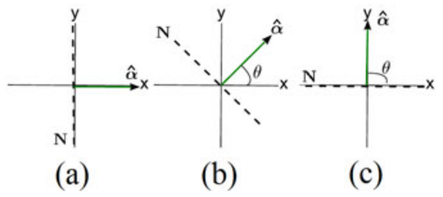

Fig. 2: Three 1D unit vector constraints $\hat{\boldsymbol{\alpha}}$ where (a) $\theta = 0°$, (b) $\theta = 45°$, and (c) $\theta = 90°$ (reproduced from [2]).

3.2 Gradient Descent

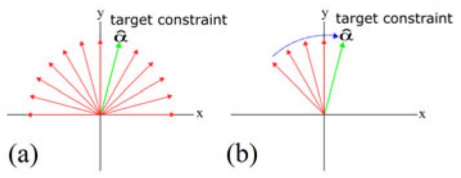

Fig. 3: The search space for (a) brute force and (b) gradient descent.

Although the brute force method and gradient descent approach work from the same insights, the main deviation and thereby main contribution lies in how the search for every $\tilde{\boldsymbol{\alpha}}$ is conducted as demonstrated in Fig 3 below. The former method Fig 3(a) performs an exhaustive search by generating a set of $\mathcal{L}$ sample $\boldsymbol{\theta}$s and finding the one with the lowest error (i.e. the target constraint) from all possible solutions. While, the latter approach finds the optimal $\boldsymbol{\theta}$ by first initialising a random $\boldsymbol{\theta}$ which lies between 0 and π (thus

[1]Since each $\boldsymbol{\alpha}_s$ is orthonormal to all preceding vectors $\boldsymbol{\alpha}_1, \cdots \boldsymbol{\alpha}_{s-1}$, the number of parameters required to describe the sth vector is $\mathcal{Q} - s$. The total number of parameters required to describe $\tilde{\mathbf{A}}$ then is $\mathcal{Q} - 1 + \mathcal{Q} - 2 + \cdots + \mathcal{Q} - \mathcal{S} = \mathcal{S}\mathcal{Q} - \sum_{s=1}^{\mathcal{S}} s = \mathcal{S}(2\mathcal{Q} - \mathcal{S} - 1)/2$.

covering the entire search space, including between π and 2π which are identical [2]), and iteratively moving towards the negative of the gradient in fixed step sizes λ.

This iterative approach to find the optimal $\boldsymbol{\theta}$ can be expressed as:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \lambda \boldsymbol{\nabla}_{\boldsymbol{\theta}} E. \tag{11}$$

Applying the chain rule

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} E = \frac{\partial E}{\partial \tilde{\mathbf{A}}} \frac{\partial \tilde{\mathbf{A}}}{\partial \boldsymbol{\theta}} = \tilde{\mathbf{A}} \mathbf{Y} \mathbf{Y}^\top \frac{\partial \tilde{\mathbf{A}}}{\partial \boldsymbol{\theta}} \tag{12}$$

where the identity $\frac{\partial}{\partial \mathbf{A}} \mathrm{Tr}[\mathbf{B}^\top \mathbf{A}^\top \mathbf{A} \mathbf{B}] = 2\mathbf{A}\mathbf{B}\mathbf{B}^\top$ is used.

One important aspect of this approach is having k number of randomly distributed starting $\boldsymbol{\theta}$s, which can run in parallel with a condition to stop the search if one of the $\boldsymbol{\theta}$s reaches the constraint sooner, saving both time and memory demand through fewer overall iterations. Multiple θs reduce the chance of getting stuck in a minimum, differing from the desired constraint. It is also necessary to set a limit to the number of iterations j. Otherwise the time required to reach the minimum cannot be precisely predicted, resulting in vastly varying running times. [2] describes the learning procedure using the brute force method. [2]

4 Experiment: Simulated Problem

This experiment aims to assess whether the gradient descent method learns null space projections from constrained data faster than the brute force approach in [2], and identify the strengths and weaknesses of the method by evaluating its performance under varying conditions.

Following [2], a linear and non-linear ground truth null space policy type ($\boldsymbol{\pi}$) are considered. These policies are designed to simulate real world manipulators of varying degrees of freedom, and are tested on problems where $Q = 1, \ldots, 9$:

1. a linear policy: $\boldsymbol{\pi}(\mathbf{x}) = -\boldsymbol{L}(\mathbf{x} - \mathbf{x}^*)$ where $\boldsymbol{L}$ is a $Q \times Q$ positive definite gain matrix and $\mathbf{x}^*$ is the target state.
2. a sinusoidal policy:

$$\boldsymbol{\pi}_i(\mathbf{x}) = \begin{cases} \pi i \sin x_1 ... \sin x_{Q-1} & \text{for } i \text{ odd} \\ \pi i \cos x_1 ... \cos x_{Q-1} & \text{for } i \text{ even.} \end{cases}$$

To evaluate the performance of the gradient descent approach, the following parameters are used. A fixed step λ size, the number of iterations as $j = 100$ and $k = 20$ initial $\boldsymbol{\theta}$s drawn uniform randomly are used. Following [2], the training and testing data sets each contain 50 data points which are drawn uniform-randomly across the space $(\mathbf{x})_i \sim \mathcal{U}(-2, 2), i \in \{1, 2\}$ and are subject to a 1D constraint $\boldsymbol{A} = \hat{\boldsymbol{\alpha}} \in \mathbb{R}^{1 \times Q}$, in the direction of the unit vector $\hat{\boldsymbol{\alpha}}$. The constraint is generated uniform-randomly, $\boldsymbol{\theta} \sim \mathcal{U}(0, \pi)$ *rad.*

The experiment is repeated 50 times and the average NPPE (defined in [2]) and time taken to learn the constraint are evaluated. The experiment is repeated with the brute force method on the same data.

[2]An implementation of the algorithm described in this section can be downloaded from github.com/jeevanmanavalan/constraint-learning

4.1 PPE of Gradient Descent and Brute Force

Figure 4 below shows the average PPE (defined in [2]) obtained for brute force and gradient descent. These methods are used to learn a 1D constraint from data ranging between 2-9 dimensions. The solid (blue) and dashed (yellow) lines in

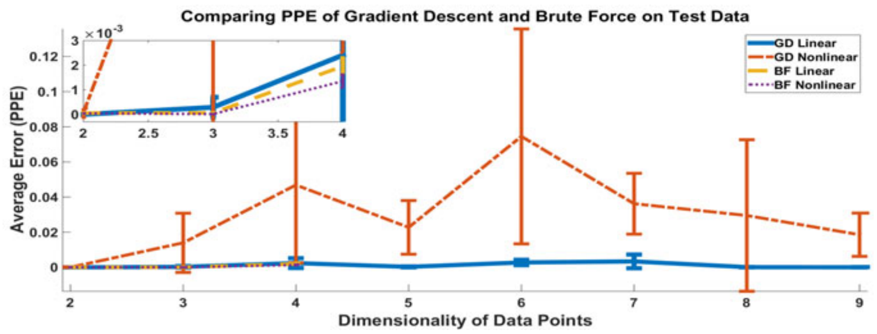

Fig. 4: PPE (mean±std) of Gradient Descent and Brute Force for 2-9 Dimensions over 50 trials. Inset showing same data for 2-4 dimensions

Figure 4 show good predictions from linear data with both gradient descent and brute force respectively (with errors around $\sim 10^{-3}$ shown clearly in the inset within the Figure). Gradient descent can learn constraints of up to 9D data, while brute force is limited to 4D data, since generating a search space for 5D data requires over 30GB of memory using Matlab. The significant difference in memory resource requirements demonstrates gradient descent's superiority with linear control policies. Although this method performs with a lower accuracy when dealing with non-linear data, the errors ranging between 0.01 and 0.08 remain below 0.1%. Both the rate and standard deviation of errors in gradient descent-based trials, relating to problems with local minimums and increasing search space sizes, can be addressed by increasing j or k. There is no mentionable decline in the brute force's performance of either control policy up to 4D data. [3]

4.2 Time Taken for Gradient Descent and Brute Force

Another important factor for practical applications is the time these methods take when handling data of varying dimensionality. Figure 5 shows the average time taken for learning the constraint using the aforementioned methods. The specification of the computer used is an Intel Core i5-4690 CPU with 4 CPU's running at 3.5GHz and 8GB of memory. No background programs were running during testing. In relation to 2D and 3D data, brute force learns fastest, requiring on average a fraction of a second. However, with 4D data brute force takes 9 times longer than its competitor due to a large search space. By contrast, gradient descent with 9D data and using $j = 100$ takes maximum 5 seconds under restricted conditions. As explained in §4.1, good predictions can be made especially in the case of handling a linear control policy.

[3] The data used in these evaluations can be downloaded from https://doi.org/10.6084/m9.figshare.4645456

5 Conclusion and Future Work

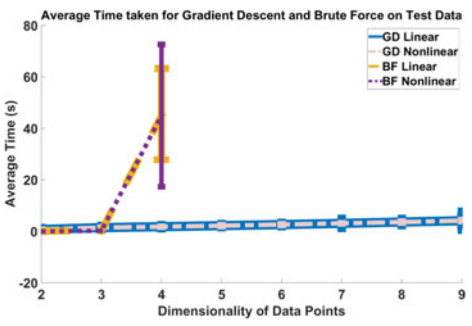

Fig. 5: Time taken (mean±std) for gradient descent and brute force over 50 trials with std scaled 30 times.

This paper evaluates the performance of the gradient descent-based learning method, extending [2] to show how this method can be used to estimate the null space projection matrix of a kinematically constrained system in the absence of any prior knowledge regarding the underlying control policy. In doing so, it compares the proposed method to the existing method in [1]. The findings reveal that this method is significantly faster than brute force when handling 4D data or higher using a linear policy. There is a comparable yet insignificant degradation in the gradient descent's performance of non-linear data, although it can handle higher dimensions of up to and including 9D data. By contrast, brute force is limited to 4D data.

Future work entails optimising the gradient descent approach to improve how it handles non-linear data and to test its application in the real world and on memory demand, thereby extending it to higher degrees of freedom.

References

[1] H.C. Lin, P. Ray, and M. Howard. Learning task constraints in operational space formulation. *IEEE Int. Conf. on Robotics and Automation (In press, preprint available: https://arxiv.org/pdf/1607.07611.pdf)*, 2017.

[2] H.C. Lin, M. Howard, and S. Vijayakumar. Learning null space projections. *IEEE Int. Conf. on Robotics and Automation*, pages 2613–2619, May 2015.

[3] M. Howard, S. Klanke, M. Gienger, C. Goerick, and S. Vijayakumar. Robust constraint-consistent learning. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 4629–4636, 2009.

[4] M. Howard, S. Klanke, M. Gienger, C. Goerick, and S. Vijayakumar. A novel method for learning policies from variable constraint data. *Autonomous Robots*, 27(2):105–121, 2009.

[5] M. Stilman and J. Kuffner. Planning among movable obstacles with artificial constraints. *Int. J. Robotics Research*, 27(11-12):1295–1307, 2008.

[6] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *Experimental Robotics*, pages 477–491. Springer, 2014.

[7] R. Fernandez, A.S. Vazquez, I. Payo, and A. Adan. A comparison of tactile sensors for in-hand object location. Hindawi Publishing Corporation, 2016.

[8] P. Gil, C.M. Mateo, A. Delgado, and F. Torres. Visual/tactile sensing to monitor grasps with robot-hand for planar elastic objects. In *Int. Symp. Robotics*, 2016.

[9] V. Ortenzi, H.C. Lin, M. Azad, J.A. Kuo, R. Stolkin, and M. Mistry. Estimation of contact constraints using kinematics. In *IEEE-RAS Int. Conf. on Humanoid Robots.*

[10] C. Gehring, C.D. Bellicoso, S. Coros, M. Bloesch, P. Fankhauser, M. Hutter, and R. Siegwart. Dynamic trotting on slopes for quadrupedal robots. In *IEEE/RSJ IEEE Int. Conf. on Intelligent Robots and Systems*, 2015.