Knowledge Transfer for Learning Robot Models via Local Procrustes Analysis

Ndivhuwo Makondo[1] Benjamin Rosman[2] Osamu Hasegawa[1]

***Abstract*— Learning of robot kinematic and dynamic models from data has attracted much interest recently as an alternative to manually defined models. However, the amount of data required to learn these models becomes large when the number of degrees of freedom increases and collecting it can be a time-intensive process. We employ transfer learning techniques in order to speed up learning of robot models, by using additional data obtained from other robots. We propose a method for approximating non-linear mappings between manifolds, which we call Local Procrustes Analysis (LPA), by adopting and extending the linear Procrustes Analysis method. Experimental results indicate that the proposed method offers an accurate transfer of data and significantly improves learning of the forward kinematics model. Furthermore, it allows learning a global mapping between two robots that can be used to successfully transfer trajectories.**

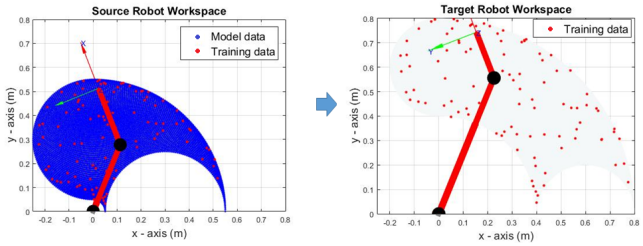


Fig. 1: Knowledge transfer process. The source robot has collected sufficient data (blue) to learn its models, whereas the target robot has only sparse data (red). The mapping function is learned using the corresponding training data (red) from both robots. The light gray background in the target space is the workspace of the target robot to be inferred. The dark cylinders are the joints – with the base at (0,0) – connecting the two red links.

I. INTRODUCTION

Learning robot models from data is typically employed as an alternative to manual programming when the physical parameters of the robot are unknown or inaccurate. Unknown non-linearities can be taken into account as the model is estimated directly from measured data, while they are typically neglected by the standard physics-based modeling techniques [1]. Furthermore, modern robot systems are complex to model manually and are becoming increasing more diverse and widespread. As a result, learning of robot kinematic and dynamic models has attracted much interest and has been used successfully in recent years [1]–[3]. However, this requires a large amount of data as the number of degrees of freedom (DoF) increases.

Collecting data samples to learn these models can be a time-intensive process. This problem becomes considerably worse when we have multiple robots, each with different structural properties (link dimensions, DoFs, etc.), that have to go through the same laborious process so as to learn models. If at least one robot (source) has collected sufficient data to learn its models it would be helpful to leverage this so as to reduce the need and effort to collect data for every other robot (target). In this case, knowledge transfer between robots would be desirable.

This research was supported by Core Research for Evolutional Science and Technology (CREST) program of Japan Science and Technology Agency (JST).

[1]Ndivhuwo Makondo and Osamu Hasegawa are with the Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology, Yokohama, Japan. makondo.n.aa@m.titech.ac.jp, oh@haselab.info

[2]Benjamin Rosman is with the Mobile Intelligent Autonomous Systems, Council for Scientific and Industrial Research, Pretoria, South Africa; and also with the School of Computer Science and Applied Mathematics, University of the Witwatersrand, South Africa. brosman@csir.co.za

In this study, we investigate how to facilitate transfer of kinematic and dynamic models between robots, in order to improve the learning speed of these internal models, by using data collected from other robots. We consider a scenario where a source robot has collected sufficient data to learn an accurate kinematic (or dynamic) model, whereas a target robot has only a small set of training data, insufficient for model learning, as shown in Fig. 1. Additionally, the source robot might have learned several tasks and the target robot is expected to learn these from the source robot. We are interested in utilizing the "knowledge" gained by the source robot when learning its internal models and tasks in order to improve the learning speed of the internal models and the tasks of the target robot.

Knowledge transfer in machine learning has been studied under the concept of transfer learning [4]. The core idea is that experience gained in learning to perform a task in one domain can help improve learning performance of the same or a similar task in a different but related domain. In robotics it has mainly been used in a reinforcement learning (RL) context – where knowledge from source tasks is used to learn target tasks faster than if transfer was not used [5] – and has recently been adopted for robot models [6]. We also adopt transfer learning techniques in this work to facilitate knowledge transfer for robot models.

The motivation behind transfer learning is the ability of humans to retain and use knowledge acquired previously to solve new tasks faster. Humans do not only learn from their own past experiences but also from those of others by means of knowledge sharing. For example, pupils use knowledge gained from their teachers to enhance their skills in order to solve problems. This intuition motivates our work.

The remainder of this paper is organized as follows: a brief survey of robot model learning is given in Section I-A.

Then, we present a general introduction to transfer learning in Section I-B, also highlighting related work. Section II lays the framework of our approach and introduces an algorithm adopted in this work, and in Section III we detail our proposed algorithm. In Section IV we show experiments where our proposed algorithm improves robot model learning and compare to an existing algorithm, while conclusions and future work are discussed in Section V.

A. Robot Model Learning

Robot model learning is the process whereby robots automatically generate internal models based on information which is extracted from the data streams available to the robot [1]. Based on these models, robots are able to interact with and influence the environment around them. Typically considered models are forward kinematics [7], inverse kinematics [2], [8], inverse dynamics [3], [7], [9] and operational space control [10], [11]. These models are typically learned by employing well known regression techniques; with an exception of inverse kinematics, which is an ill-posed problem and requires specific solutions. For more information about robot model learning, the reader is referred to a survey [1].

In order for these models to be effectively used much data needs to be acquired. This process can be expensive, but a number of techniques for dealing with this exist, including goal babbling [12]. Typically in kinematics modeling the models need to be defined everywhere in the robot domain. In contrast, models for inverse dynamics and operational space control may be only defined in the domain of the task to be learned and that task may not necessarily cover the entire robot domain. Thus, less data is required for these compared to forward kinematics. In this paper, we primarily focus on speeding up learning of models that are defined in the entire domain of a robot such as forward and inverse kinematics, but our method is applicable to the other models mentioned above.

B. Transfer Learning

Transfer learning is applied in cases where the training and test data are drawn from different feature spaces and different distributions [4]. This is because standard machine learning techniques would typically require new training data when the distribution changes, which can be expensive to collect. In this case, learning can be improved by supplementing the new dataset with additional datasets from the same or different domains. Transfer learning requires a source task associated with the source domain and a target task associated with the target domain. In our case of robot learning, the domain could refer to the joint space and the end-effector workspace of the robot, and the task could be one of the models to be learned, e.g., inverse kinematics. Several approaches have been developed including instance-based transfer learning [13], feature-based transfer learning [14], parameter sharing [9] and relational-knowledge-transfer [15]. Transfer learning has been widely applied [4], with applications including text categorization [16], Naive Bayes classification [17], boosting [13] and WiFi localization [14].

There have been only a few attempts to apply transfer learning for robot models. Bocsi et al. [6] recently proposed a method to transfer robot models using the Procrustes Analysis algorithm (PA) [18]. Their goal is to transfer a task from the source robot to the target robot in order to speed up learning of the target task, by using additional data from the source task. This approach is similar to that of Pan et al. [14] in that they use dimensionality reduction to find a common latent feature space of the source and target data. They then find a linear transformation between the source and target in this latent space. This creates a bridge through which data can be transferred between the two data spaces.

In this paper we adopt and extend the same framework. However, instead of transferring knowledge about a single task [6], we are interested in learning a global mapping between the source robot domain and the target robot domain, similar to the Heterogeneous Domain Adaptation (HDA) framework for classification problems [19]. This will allow all tasks that are learned in the source domain to be automatically transferred to the target domain without having to learn an independent mapping for each task. We show in our experiments that applying a linear mapping function is limited and we thus propose an extension to the Procrustes Analysis algorithm, which we call Local Procrustes Analysis (LPA), to approximate non-linear mappings.

II. Knowledge Transfer in Robot Learning

The aim of our work is to transfer a model from a source robot to a target robot in order to improve the learning process of the target tasks. We assume the source and target datasets, $\boldsymbol{D}^s = \{\boldsymbol{\theta}_i, \boldsymbol{x}_i\}_{i=1}^{N_s}$ and $\boldsymbol{D}^t = \{\boldsymbol{\theta}_i, \boldsymbol{x}_i\}_{i=1}^{N_t}$, are samples from the source and target domains $\boldsymbol{\chi}^s \in \mathbb{R}^{d_s+m_s}$ and $\boldsymbol{\chi}^t \in \mathbb{R}^{d_t+m_t}$, respectively; where N_s and N_t are the sizes of the respective datasets, d_i is the number of DoFs of robot i and m_i is the corresponding number of variables representing the end-effector pose. In each case, the vector $\boldsymbol{\theta}$ contains the joint angles and the vector $\boldsymbol{x}$ contains the end-effector pose variables. For simplicity we present our work for kinematic models, and specifically forward kinematics. However, our method can be applied to the other models described above. For models that track a specific trajectory such as inverse dynamics and operational space control, the datasets $\boldsymbol{D}^s$ and $\boldsymbol{D}^t$ represent the data collected in the domain of the task and not the robot domains. Our objective is to find a mapping f that enables us to map points from the source domain $\boldsymbol{\chi}^s$ to the target domain $\boldsymbol{\chi}^t$, i.e., $f : \boldsymbol{\chi}^s \mapsto \boldsymbol{\chi}^t$, using the samples in $\boldsymbol{D}^s$ and $\boldsymbol{D}^t$.

We consider a case where the source robot has collected sufficient data to learn its models whereas the target robot has not, i.e., $N_t \ll N_s$, and we assume that we know correspondences between a subset of N_t points of the source data and all of the target data. In practice, this is not always possible and some optimal matching of points is required[1]. In general, the dimensions of $\boldsymbol{D}^s$ and $\boldsymbol{D}^t$ may not match

[1]This is related to the correspondence problem often encountered in the programming by demonstration (PbD) framework [20].

due to the robots having different DoFs and different end–effector workspace dimensions, i.e., $d_s \neq d_t$ and $m_s \neq m_t$. To overcome this problem, Bosci et al. [6] use dimensionality reduction to find a low dimensional representation of the datasets $\boldsymbol{D}^s$ and $\boldsymbol{D}^t$ with the same dimensionality and then compute the linear mapping in this latent space. In our work we assume that, if required, the datasets have already been mapped into this latent space. Our goal is then to find a non-linear mapping between the two datasets. We now present the Procrustes Analysis algorithm [6], [18] for modeling linear mappings, which we adapt in Section III to build our non-linear mapping method.

A. Procrustes Analysis

In the Procrustes Analysis method, the goal is to learn a mapping f that maps points between the source manifold $\boldsymbol{Z}^s$ and the target manifold $\boldsymbol{Z}^t$, where $\boldsymbol{Z}^s \subset \boldsymbol{\chi}^s$ and $\boldsymbol{Z}^t \subset \boldsymbol{\chi}^t$. The manifolds are assumed to be of the same dimension and contain the same number of paired elements, i.e., $\boldsymbol{z}_i^s$ matches $\boldsymbol{z}_i^t$ for $i = 1, \ldots, N_t$, and $\boldsymbol{Z}^s = \{\boldsymbol{z}_i^s\}_i^{N_t}$ and $\boldsymbol{Z}^t = \{\boldsymbol{z}_i^t\}_i^{N_t}$. This method was developed by Wang and Mahadevan [18] and was adopted by Bocsi et al. [6] for robot models. We present it here as originally described [6] and then introduce our extension to non-linear mappings in the next section.

First, the data is centered by subtracting the mean and then it is whitened by dividing by the standard deviations as follows,

$$\boldsymbol{s} = \boldsymbol{B}^s(\boldsymbol{z}^s - \boldsymbol{\omega}^s), \tag{1}$$

$$\boldsymbol{t} = \boldsymbol{B}^t(\boldsymbol{z}^t - \boldsymbol{\omega}^t), \tag{2}$$

where $\boldsymbol{s} \in \boldsymbol{M}^s$ and $\boldsymbol{t} \in \boldsymbol{M}^t$ are the standardized elements. The values $\boldsymbol{\omega}^s = \mathbb{E}\{\boldsymbol{Z}^s\}$ and $\boldsymbol{\omega}^t = \mathbb{E}\{\boldsymbol{Z}^t\}$ are the means of the data, where $\mathbb{E}\{\cdot\}$ denotes the expectation operator. Matrices $\boldsymbol{B}^s$ and $\boldsymbol{B}^t$ can be obtained using the Singular Value Decomposition (SVD) of the covariance matrices of $\boldsymbol{Z}^s$ and $\boldsymbol{Z}^t$ respectively, and are such that the data $\boldsymbol{M}^s$ and $\boldsymbol{M}^t$ are whitened.

The manifold alignment function is modeled as a linear mapping $f : \boldsymbol{M}^s \mapsto \boldsymbol{M}^t$, with $f(\boldsymbol{s}) = \boldsymbol{A}\boldsymbol{s}$, where $\boldsymbol{A}^{J\times J}$ is a transformation matrix with J as the dimension of the manifolds. Wang and Mahadevan [18] minimized $\|\boldsymbol{T} - f(\boldsymbol{S})\|_F$ where $\boldsymbol{S}$ and $\boldsymbol{T}$ are matrices formed from the data of $\boldsymbol{M}^s$ and $\boldsymbol{M}^t$, and $\|\cdot\|_F$ is the Frobenius norm. Bocsi et al. [6] minimize the expected loss of the transformation instead. We find the parameters $\boldsymbol{A}$ such that the expectation of the error of the transformation $L(\boldsymbol{A})$ is minimized, i.e.,

$$\boldsymbol{A} = \arg\min_r L(\boldsymbol{A}) \tag{3}$$

with

$$\begin{aligned} L(\boldsymbol{A}) &= \mathbb{E}\{(\boldsymbol{t} - \boldsymbol{A}\boldsymbol{s})^T(\boldsymbol{t} - \boldsymbol{A}\boldsymbol{s})\} \\ &= tr(\boldsymbol{\Sigma}_{tt} - 2\boldsymbol{A}^T\boldsymbol{\Sigma}_{ts} + \boldsymbol{A}^T\boldsymbol{\Sigma}_{ss}\boldsymbol{A}) \end{aligned} \tag{4}$$

where $\boldsymbol{\Sigma}_{ss}$, $\boldsymbol{\Sigma}_{tt}$ and $\boldsymbol{\Sigma}_{ts}$ are covariance matrices and L is a loss function. The minimization can be performed by setting the derivative of $L(\boldsymbol{A})$ to zero. After differentiation we get $0 = -2\boldsymbol{\Sigma}_{ts} + 2\boldsymbol{A}^T\boldsymbol{\Sigma}_{ss}$, giving

$$\boldsymbol{A} = \boldsymbol{\Sigma}_{ss}^{-1}\boldsymbol{\Sigma}_{ts}. \tag{5}$$

A new point $\boldsymbol{s}_\star = \boldsymbol{B}^s(\boldsymbol{z}_\star^s - \boldsymbol{\omega}^s)$ in the source manifold can then be mapped using $\hat{\boldsymbol{z}}_\star^t = \boldsymbol{B}^{t^{-1}}\boldsymbol{A}\boldsymbol{s}_\star + \boldsymbol{\omega}^t$, where $\hat{\boldsymbol{z}}_\star^t$ is the transferred point. This method works well for manifolds that are linear transformations of each other. In the next section, we detail our proposed method that extends to non-linear manifold mapping.

III. Local Procrustes Analysis

Our method, LPA, was inspired by locally weighted methods [3], [7], [21]. The core idea of these methods is that a non-linear function is approximated by weighting the outputs of locally linear functions. In practice, the data is clustered online, such that each cluster is associated with a linear model, and the prediction of a query point is a weighted sum of the outputs of the M linear models that are closest to the query point. The weights are computed using a similarity measure between clusters and the query point based on the Gaussian kernel [7]. The same idea could be applied with local Gaussian Processes (GP) [3].

In this paper, we approximate a non-linear mapping between two manifolds by using locally linear mappings. We apply the Procrustes Analysis method presented in Section II-A to model each of our linear mappings. To this end, we cluster the data into K clusters and compute the mapping for each cluster. Algorithm 1 shows the LPA process. The symbols will be discussed throughout the explanation in the next section. In order to obtain meaningful clustering information, we reduce the dimensionality of the datasets to some latent space wherein the variance of the data is maximized. This is because our data may be redundant, e.g., the end-effector workspace of a robot arm is related to the joint space through kinematic equations. In our experiments Principal Component Analysis (PCA) proved sufficient but any dimensionality reduction technique can be used, as long as it allows new points to be mapped onto the latent space.

A. Clustering and Mapping

The assumptions we make in LPA are that the domains are locally continuous and smooth. This means that points in the same local neighborhood in the source domain are mapped to the same neighborhood in the target domain, and that they share the same locally linear mapping. Our goal then is to represent the two datasets $\boldsymbol{D^s}$ and $\boldsymbol{D^t}$ by a mixture of K

Algorithm 1 Local Procrustes Analysis

IN: Training set $\boldsymbol{Z}^s$ and $\boldsymbol{Z}^t$
$(\mathcal{Z}^s, \mathcal{Z}^t, f_{pca}) \Leftarrow$ PCA($\boldsymbol{Z}^s$, $\boldsymbol{Z}^t$)
($\boldsymbol{\Pi}$, K) $\Leftarrow$ initEM($\mathcal{Z}^s$, $\mathcal{Z}^t$) {see Section III-B}
$\boldsymbol{\Pi} \Leftarrow$ fitGMM($\mathcal{Z}^s$, $\boldsymbol{\Pi}$, K)
for each cluster $k \in [1, K]$ **do**
    Compute $\boldsymbol{B}_k^s$, $\boldsymbol{B}_k^t$, $\boldsymbol{\omega}_k^s$ and $\boldsymbol{\omega}_k^t$ from (1) and (2)
    Compute $\boldsymbol{A}_k$ from (5) {see Section II-A}
end for
$\Theta = \{\boldsymbol{A}_k, \boldsymbol{B}_k^s, \boldsymbol{B}_k^t, \boldsymbol{\omega}_k^s, \boldsymbol{\omega}_k^t\}$
OUT: $\{\boldsymbol{\Pi}, \Theta, f_{pca}\}$

regions where corresponding points in the datasets map to the same region, as shown in Fig. 2. We select N_t corresponding points, $\boldsymbol{Z}^s \subset \boldsymbol{D}^s$ and $\boldsymbol{Z}^t \subset \boldsymbol{D}^t$, in both datasets as our training data. We employ Gaussian Mixture Modeling (GMM), where the Gaussian mixtures correspond to the local regions, trained using the Expectation–Maximization (EM) algorithm, because it enables us to naturally interpolate the output of local mappings using component responsibilities as weights.

A GMM is represented by three parameters: the mixing coefficients $\boldsymbol{\pi}_k$, the mean vectors $\boldsymbol{\mu}_k$ and the covariance matrices $\boldsymbol{\Sigma}_k$. The total probability density is then defined as a superposition of K Gaussian densities of the form

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \boldsymbol{\pi}_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{6}$$

and the components' responsibilities are defined as

$$\gamma_k = \frac{\boldsymbol{\pi}_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \boldsymbol{\pi}_j \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}, \tag{7}$$

where $\mathcal{N}$ is a multivariate normal distribution. γ_k can be viewed as the responsibility that component k takes for explaining the point $\boldsymbol{x}$.

We fit a GMM to the source domain in a latent space defined by the function f_{pca} and cluster the the projected data $\mathcal{Z}^s$ of $\boldsymbol{Z}^s$ by assigning points to components with the highest responsibilities. We use this clustering information to cluster data in the target domain ($\mathcal{Z}^t$ projected from $\boldsymbol{Z}^t$), i.e., points in the target domain that correspond to points in the same cluster in the source domain are clustered together. This means that the clustering is only done on the source domain and then transferred to the target domain. When training a GMM using the EM algorithm, we need to determine the number K of components and initialize the parameters $\boldsymbol{\Pi} = \{\boldsymbol{\pi}_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$. This is done by the function *initEM* in step 3 of Algorithm 1 and will be discussed in detail in the next section.

We then use this information to compute mappings for each cluster in the original dimensions. We represent mappings of the set of clusters by the parameter $\Theta = \{\boldsymbol{A}_k, \boldsymbol{B}_k^s, \boldsymbol{B}_k^t, \boldsymbol{\omega}_k^s, \boldsymbol{\omega}_k^t\}$ where the transformation matrix $\boldsymbol{A}_k$ is computed from (5), matrices $\boldsymbol{B}_k^s$ and $\boldsymbol{B}_k^t$ are computed from (1) and (2), and mean vectors $\boldsymbol{\omega}_k^s$ and $\boldsymbol{\omega}_k^t$ are also computed from (1) and (2).

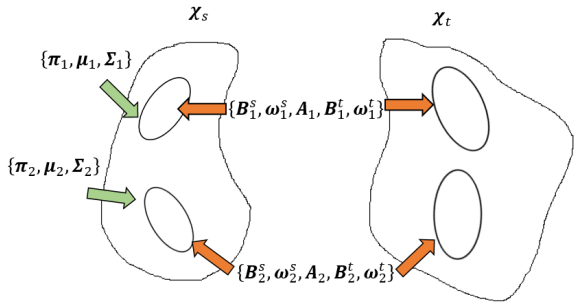


Fig. 2: Illustration of LPA for $K = 2$. The orange arrows correspond to mappings between Gaussian kernels in the two spaces and the green arrows correspond to the kernels themselves.

B. Initialization of the GMM

The performance of the EM algorithm is dependent on the initialization of its parameters. There are many ways of initializing the EM algorithm and we do not put any restrictions on the choice of method to employ. Here we present our own initialization method that automatically estimates the number K of Gaussian components required.

Algorithm 2 shows the pseudo-code of our initialization method. It employs a hierarchical clustering scheme that starts with one cluster and splits it into two if some conditions are not met. This is repeated for all clusters at each iteration until the conditions are satisfied. The datasets $\mathcal{Z}^s$ and $\mathcal{Z}^t$ are projected from their corresponding datasets in the original dimension in step 2 of Algorithm 1. The vector $\boldsymbol{h}$ contains indices h_n where $h_n = k$ indicates that $\hat{\boldsymbol{z}}_n^s \in \mathcal{Z}^s$ and $\hat{\boldsymbol{z}}_n^t \in \mathcal{Z}^t$ belong to cluster k ($1 \leq n \leq N_t$). c_k is the mapping error of a cluster and is defined as follows

$$c_k = \frac{1}{N_k} \sqrt{\sum_{n=1}^{N_k} \| \boldsymbol{B}_k^{t\,-1} \boldsymbol{A}_k \boldsymbol{s}_n + \boldsymbol{\omega}_k^t - \hat{\boldsymbol{z}}_n^t \|^2}, \tag{8}$$

where N_k is the number of points in the cluster, s_n is computed from $\hat{\boldsymbol{z}}_n^s$ using (1), $\hat{\boldsymbol{z}}_n^s$ and $\hat{\boldsymbol{z}}_n^t$ are the latent points in the cluster that are in correspondence with each other in the source domain and target domain respectively. If this mapping error is greater than some threshold c_{min}, the cluster is split into two using the k–$means$ algorithm in step 7, and if the number of points in the resulting clusters, N_{C_1} and N_{C_2}, are less than some threshold N_{min}, the cluster is not split.

Algorithm 2 Initialize GMM

```
1:  IN: Training sets Z^s, Z^t
2:  Initialize index vector h to ones and K = 1
3:  while not terminated do
4:      for each cluster k ∈ [1, K] do
5:          Compute c_k on data belonging to cluster k
6:          if c_k > c_min then
7:              Split cluster C_k into two (C_{k,1} and C_{k,2})
8:              if N_{C_{k,1}} ≥ N_min and N_{C_{k,2}} ≥ N_min then
9:                  Update index vector h
10:             end if
11:         end if
12:     end for
13:     Update number of clusters K
14: end while
15: OUT: K, h
```

The threshold c_{min} reflects the error that one is willing to tolerate and in our experiments we set this to 0.001 m. If one can tolerate large errors and one cluster is sufficient, our algorithm is equivalent to the original PA. The threshold N_{min} ensures that in cases where we do not have enough training data small clusters are not split. This value is set to be at least the dimension of the original datasets, i.e., $N_{min} \geq d_s + m_s$. If a cluster is split the index vector $\boldsymbol{h}$

is updated accordingly to reflect this. Finally, the number of clusters K is updated according to how many clusters were split. This procedure terminates when there are no further clusters to split.

The output of this procedure is the number of clusters and the assignment of points to clusters represented by the index vector $\boldsymbol{h}$. In practice, we run the initialization procedure multiple times and choose parameters with the highest log-likelihood estimate. This is performed by the function *initEM* in step 3 of Algorithm 1. This result is then used in function *fitGMM* in step 4 of Algorithm 1 to train the GMM. This is repeated until the EM algorithm converges.

C. Prediction

Once the parameters $\{\boldsymbol{\Pi}, \Theta, f_{pca}\}$ of our model have been learned, we can use the model to transfer points from the source domain to the target domain. For a point $\boldsymbol{d}^s_\star \in \boldsymbol{D}^s$ in the source domain to be transferred to the target domain, we map it onto the latent space by f_{pca} to obtain point $\boldsymbol{p}_\star$ and compute each Gaussian component's weights using (7),

$$\gamma_k = \frac{\boldsymbol{\pi}_k \mathcal{N}(\boldsymbol{p}_\star \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \boldsymbol{\pi}_j \mathcal{N}(\boldsymbol{p}_\star \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{9}$$

Some components far from the query point will have zero responsibilities and therefore will have no effect on the mapping. To speed up the process for cases with many clusters, one can first find $M < K$ nearest components to the query point $\boldsymbol{p}_\star$ and only use those in (9). Finally, we map the query point $\boldsymbol{d}^s_\star$ in the original dimensions as follows

$$\boldsymbol{d}^t_\star = \sum_{j=1}^{K} \gamma_k (\boldsymbol{B}_k^{t\,-1} \boldsymbol{A}_k \boldsymbol{z}^s_{k_\star} + \boldsymbol{\omega}^t_k), \tag{10}$$

where $\boldsymbol{z}^s_{k_\star} = \boldsymbol{B}^s_k(\boldsymbol{d}^s_\star - \boldsymbol{\omega}^s_k)$ from (1) and K can be replaced by M when the M nearest components are used.

IV. Experiments

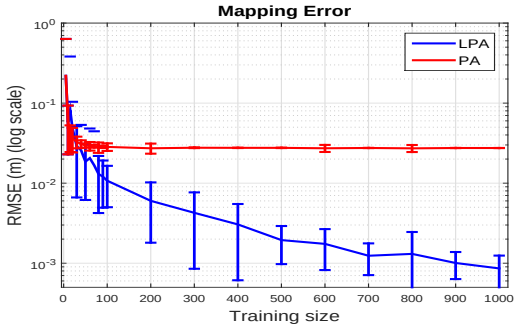
We conducted experiments on 2 DoF and 3 DoF robots to evaluate the models transferred with our method, and compare against the linear PA [6]. We aim to evaluate the performance of our mapping function and so use robots with the same number of DoFs, although in general this need not be the case. Robots were simulated using the Robotics Toolbox [22] in order to generate the data.

A. 2-Link Planar Robots

In this experiment we evaluate the effectiveness of our mapping algorithm on 2 DoF robots. Note that the visualization is done in a 2D end-effector workspace and the actual datasets are 4 dimensional, i.e., $d_s = d_t = 2$ and $m_s = m_t = 2$. We compute a mapping between two 2-link planar robots, from a small set of corresponding points, for speeding up learning of forward kinematics on the target robot. For both robots, the range of the first joint is $\{0, \frac{\pi}{2}\}$ radians and for the second joint is $\{0, \pi\}$ radians. We performed two experiments: in experiment 1, the source robot has link lengths of 0.3 m and 0.25 m, respectively, and the target robot has link lengths of 0.6 m and 0.2 m, respectively; and in experiment 2, we varied the link lengths of the target robot while keeping the links of the source robot fixed. We find correspondences in the joint space by pairing together points from both robots that correspond to the same joint space position. We set the dimension of the latent space for clustering to be 3 and $N_{min} = 7$ for the two experiments. Fig. 4 shows that the end-effector workspaces of the source robot (Fig. 4a) and the target robot (Fig. 4b) for experiment 1 are not linearly transformable.

We use the LWPR [7] algorithm – one of the state-of-the-art algorithms in robot model learning [3] – to learn forward kinematics of the target robot as the ground-truth model from 10 000 uniformly generated points (RMSE = 0.0061 m). Fig. 3 shows the performance comparison of the linear method and our non-linear method for different training data sizes for experiment 1. All results are averaged over 100 runs. This experiment shows that LPA offers a more accurate transfer of data and, given enough training data, no further learning for the target robot is required to obtain an accurate model. We achieve convergence to the ground-truth model with about 200 data points. Without transfer, this would not have happened until well over 1000 points. Also, although the linear method converges slightly faster, it does so to a much poorer model.

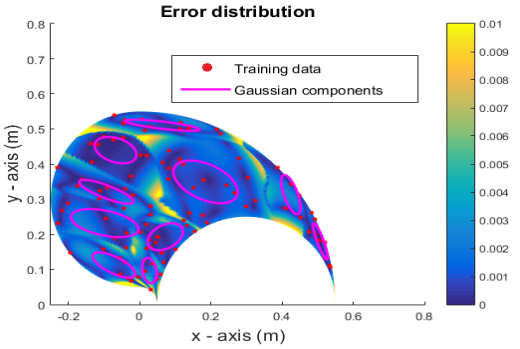
Fig. 4 shows the error distribution in the source workspace and the target workspace for an instance with 100 training points. It also shows the Gaussian components fitted to the training data, with the ellipses indicating one standard



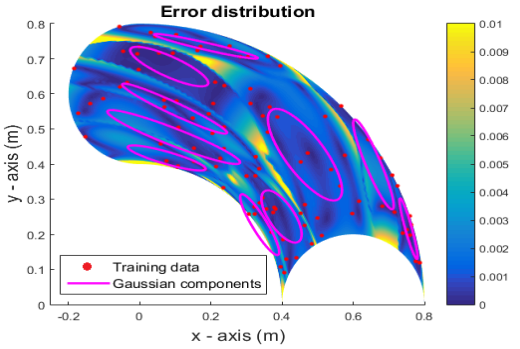
(a) Mapping error.

(b) Model error.

Fig. 3: Mapping and model errors as functions of training data size. (a) The error in the mapping is measured by the RMSE of the transferred points on the ground-truth data. (b) The error in the transferred model is measured by testing 10 000 random samples from the target robot on the different forward kinematics models.



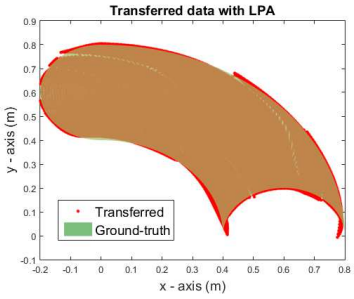
(a) Source workspace.



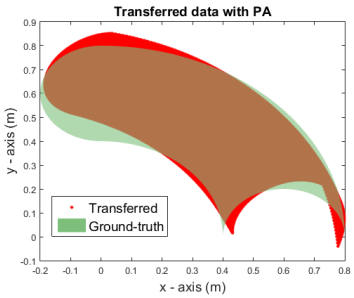
(b) Target workspace.

Fig. 4: Error distributions. The scale of the color bar is meters and for visualization purposes, errors above 1 cm are capped at 1 cm. The ellipses indicate one standard deviation of each Gaussian component.

deviation of each component. Large errors occur in regions where there are either no points or the required transformation is extremely non-linear. In both cases, adding more training data would reduce the mapping errors. Fig. 5 shows

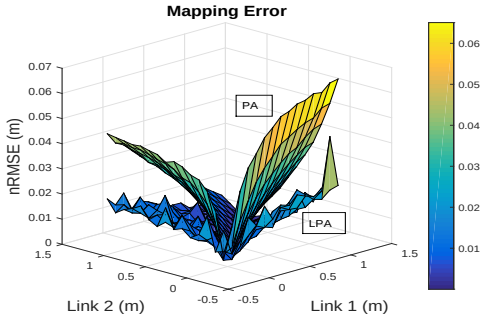


(a) Local Procrustes Analysis.

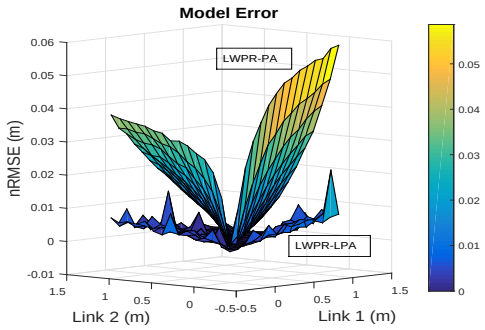


(b) Procrustes Analysis.

Fig. 5: Modeled workspace regions using LPA and PA.



(a) Mapping error.



(b) Model error.

Fig. 6: Mapping and model errors as functions of difference in robot links.

the transferred data of LPA and PA for a training set of 100 points, superimposed over the ground-truth data, to highlight the resulting structures of the transferred data for both methods.

Fig. 6 shows the results for experiment 2. We incremented each link by 0.1 m from 0.1 to 1.5 m and again used a training set of 100 points. The x- and y-axes are the difference between the robots' link 1 and link 2, respectively. We observe that both methods have a zero error along a line in the x-y plane. This occurs when the ratio between the links is the same for both robots, i.e., $\frac{L_1^s}{L_2^s} = \frac{L_1^t}{L_2^t}$ where L_i^s and L_i^t indicate the i-th links of the source robot and target robot, respectively. In this case, the underlying manifold of the target robot is a scaled version of the source robot and therefore the underlying relationship is linear. As this ratio changes the error of the linear mapping increases, while that of LPA stays relatively low. Furthermore, LPA still manages to improve the learning of the target model when the difference of the robots is significant and most of the model errors are under 1 cm. Note that errors are high when the ratios are very dissimilar but these can be reduced with more training data.

B. Forward Kinematics of 3 DoF Robots

In this experiment we evaluate the effectiveness of our mapping algorithm on 3 DoF robots. The datasets consists of 3 joint variables, i.e., $d_s = d_t = 3$, and 3 end-effector variables, i.e., $m_s = m_t = 3$ and the datasets are thus 6 dimensional. In this experiment, clustering the data in the original 6 dimensional space proved sufficient and we set $N_{min} = 10$. We map data between two 3 DoF robots with

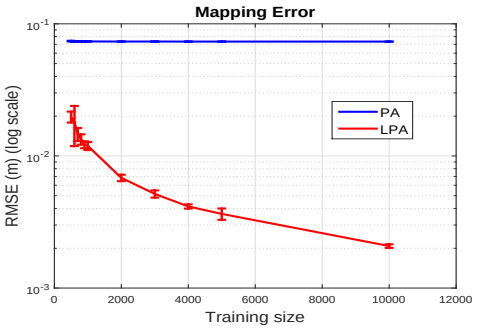
two links for speeding up learning forward kinematics on the target robot as before. The links' lengths of both robots are the same as in the previous experiment and the joint limits are as follows: $\theta_1 \in \{\frac{-\pi}{2}, \frac{\pi}{2}\}$ and $\theta_2, \theta_3 \in \{0, \frac{\pi}{2}\}$.

We compute the mappings for different training data sizes, ranging from 500 to 10 000 and the size of the ground-truth is 100 000. Results are averaged over 10 runs. As shown in Fig. 7a, the error of our method decreases significantly with more training data, confirming that it successfully captures the non-linear relationship. The linear method converges to a suboptimal transformation and cannot improve when given more data. Fig. 7b shows the success rate of the mapping functions, which is the percentage of points with total error less than 1 cm. This shows the number of points which were accurately mapped according to this threshold.

Transferring using LPA slightly improves learning of the forward kinematics model on the target robot using LWPR, as shown in Fig. 8a. The linear method fails to improve learning of the model because a high percentage of points are inaccurately mapped. Fig. 8b shows the success rate of the models, which is the percentage of test points with error less than 1 cm.

C. Trajectory Tracking Transfer

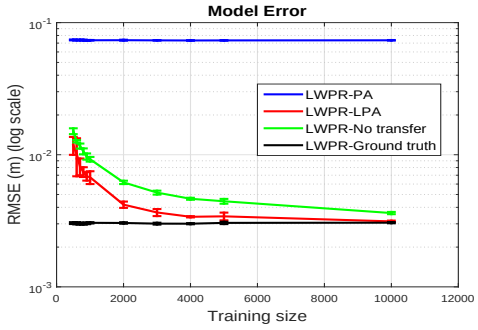
In this experiment, we also used two 3 DoF robots with two links as in the previous experiment. The links of the source robot are both 0.4 m and those of the target robot are 0.3 and 0.6 m. The joint limits of both robots are as follows: $\theta_1 \in \{\frac{-\pi}{2}, \frac{\pi}{2}\}$, $\theta_2 \in \{0, \frac{\pi}{2}\}$ and $\theta_3 \in \{0, \frac{-\pi}{2}\}$. The models of the source and target robots are shown in Fig. 9 and Fig. 10, respectively, at the same end-effector position.



(a) Mapping mean error.

(b) Mapping success rate.

Fig. 7: Mapping performance for varying training data sizes.



(a) Model error.

(b) Model success rate.

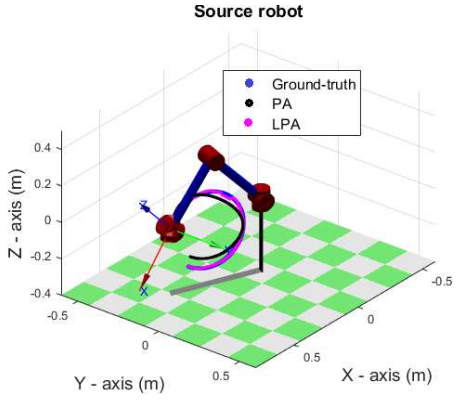
Fig. 8: Model performance for varying training data sizes.

The goal of the experiment was to transfer a trajectory from the source robot to the target robot accurately. We used a numerical inverse kinematics method to collect training data in the workspace common to both robots and then we used the data to learn a global mapping between the two robots offline, using both linear and non-linear methods. We then tracked the trajectory – the blue line in both Fig. 9 and 10 – with the source robot in order to collect the joints data corresponding to the workspace trajectory, and transfer it to the target robot – black for the linear method and magenta for our non-linear method. Our method accurately transfers the trajectory to the target robot whereas the linear method, as expected, is not as accurate.

V. Discussion

Model learning has become prevalent in robot programming. We are interested in reducing the time spent on

Fig. 9: 3D workspace visualization of the trajectories.



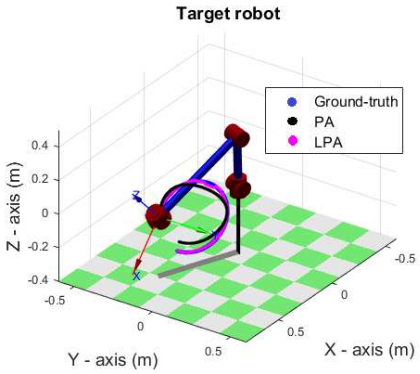


Fig. 10: 3D workspace visualization of the trajectories.

learning models of the kinematics of a target robot, by transferring from a well-understood model from a source robot. We have shown that the transformation needed to transfer data from one robot space to another in general is not linear and our experiments show that we can improve transfer learning for kinematic models, from a small training set, by using non-linear mappings. We have also shown that given enough data our proposed method can accurately transfer models such that no further learning of the target task is required. Also, our method can learn a global mapping between robots; this mapping can then be used to transfer trajectories between the robots. Although our method worked well for kinematic models and can be easily extended for dynamic models, future work will explore extensions for transferring dynamic models. The amount of training data required in the 3 DoF case grew compared to the 2 DoF case and we expect it grow fast with the number of dimensions. However, we expect our method to be more data efficient than other methods. This will also be examined in future work. Combining this approach with goal babbling for high dimensional cases is another avenue worth exploring.

The limitation of our algorithm is that it requires correspondences between the datasets to be provided. An algorithm that automatically finds correspondences would remove the need for human supervision in the data collection stage. Also, the current version works in a batch mode and this can be restrictive in high dimensions. Modifying this to run online and incrementally will enable users to build local mappings on the fly in regions of interest and update them as more data is acquired. The probabilistic nature of our algorithm allows us to find regions with low confidence and we can use this to provide more data in an active learning fashion. Moreover, we can take actions against points with low probabilities to avoid negative transfer of knowledge. We plan to implement these features in the near future. We will also focus on combining our method with suitable dimensionality reduction techniques in order to apply it to robots with different numbers of DoFs.

ACKNOWLEDGMENT

Ndivhuwo Makondo wishes to thank Michihisa Hiratsuka, Yoshihiro Nakamura and Daiki Kimura of O. Hasegawa Lab for their helpful suggestions. The authors wish to thank the reviewers for their invaluable comments.

REFERENCES

[1] D. Nguyen-Tuong, and J. Peters, Model Learning in Robotics: a Survey, Cognitive Processing, 2011.
[2] A. D'Souza, S. Vijayakumar, and S. Schaal, Learning inverse kinematics, IEEE International Conference on Intelligent Robots and Systems (IROS 2001), Vol 1 , pp. 298-303.
[3] D. Nguyen-Tuong, M. Seeger, and J. Peters, Model Learning with Local Gaussian Process Regression, Advanced Robotics, no. 15, pp. 2015-2034 (2009).
[4] S. J. Pan, and Q. Yang, A survey on transfer learning, Knowledge and Data Engineering, IEEE Transactions on, 2010, Vol, 10, pp. 1345–1359.
[5] M. E. Taylor, and P. Stone, Transfer Learning for Reinforcement Learning Domains: A Survey, Journal of Machine Learning Research, 2009, Vol. 10, pp. 1633–1685.
[6] B. Bocsi, L. Csato, and J. Peters, Alignment-based Transfer Learning for Robot Models, in Proc. of the IEEE International Joint Conference on Neural Networks (IJCNN), 2013, pp. 1-7.
[7] S. Vijayakumar, A. D'Souza and S. Schaal, Incremental Online Learning in High Dimensions, Neural Computation, vol. 17, no. 12, pp. 2602-2634 (2005).
[8] B. Bocsi, D. Nguyen-Tuong, L. Csato, B. Schoelkopf, and J. Peters, Learning Inverse Kinematics with Structured Prediction, in Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS), pages 698-703, San Francisco, USA, 2011.
[9] K. M. Chai, S. Vijayakumar, S. Klanke, and C. Williams, Multi-task Gaussian Process Learning of Robot Inverse Dynamics, Advances in Neural Information Processing Systems 21, 2009, pp. 265-272.
[10] B. Bocsi, P. Hennig, L. Csato, and J. Peters, Learning Tracking Control with Forward Models, in Proc. of the IEEE International Conference on Robotics and Automation (ICRA), pp. 259-264, St. Paul, MN, USA, 2012.
[11] C. Salan, V. Padois, and O. Sigaud, Learning Forward Models for the Operational Space Control of Redundant Robots, From Motor Learning to Interaction Learning in Robots, Studies in Computational Intelligence, Vol 264, 2010, pp. 169-192, Springer Berlin Heidelberg.
[12] M. Rolf, and J. J. Steil, Efficient exploratory learning of inverse kinematics on a bionic elephant trunk, IEEE Trans. Neural Networks and Learning Systems , vol. 25, no. 6, pp. 1147-1160, 2014.
[13] W. Dai, Q. Yang, G. R. Xue, and Y. Yu, Boosting for Transfer Learning, in Proc. of the 24th International Conference on Machine Learning (ICML), 2007, pp. 193-200, Corvalis, Oregon, USA.
[14] S. J. Pan, J. T. Kwok, and Q. Yang, Transfer Learning via Dimensionality Reduction, in Proc. of the 23rd National Conference on Artificial Intelligence, Vol. 2, 2008, pp. 677-682, Chicago, Illinois.
[15] H. Zhuo, Q. Yang, and D. H. Hu, Transferring knowledge from another domain for learning action models, in Proc. of 10th Pacific Rim International Conference on Artificial Intelligence, 2008.
[16] R. Raina, and A. Y. Ng and D. Koller, Constructing Informative Priors Using Transfer Learning, in Proc. of the 23rd International Conference on Machine Learning (ICML), 2006, pp. 713-820, Pittsburgh, Pennsylvania, USA.
[17] W. Dai, G. R. Xue, Q. Yang, and Y. Yu, Transferring Naive Bayes Classifiers for Text Classification, in Proc. of the 22Nd National Conference on Artificial Intelligence (AAAI), 2007, pp. 540-545, Vancouver, British Columbia, Canada.
[18] C. Wang, and S. Mahadevan, Manifold alignment using Procrustes analysis, in Proc. of the 25th international conference on Machine learning, 2008, pp. 1120–1127.
[19] J. T. Zhou, I. Tsang, S. J. Pan, and M. Tan, Heterogeneous Domain Adaptation for Multiple Classes, in Proc. of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS) 2014, Reykjavik.
[20] B. Argall, S. Chernova, M. Veloso, and B Browning, A survey of robot learning from demonstration, Robotics and Autonomous Systems, 2009, 57(5), pp 469-483.
[21] C. G. Atkeson, A. W. Moore, and S. Schaal, Locally weighted learning for control, Artificial Intelligence Review, 1997, Vol. 11, pp. 75–113.
[22] P.I. Corke, Robotics, Vision & Control, Springer 2011, ISBN 978-3-642-20143-1.