Imitation Learning and Transferring of Human Movement and Hand Grabbing to Adapt to Environment Changes

Stephan Al-Zubi and Gerald Sommer

Cognitive Systems, Christian Albrechts University, Kiel, Germany,
{sa, gs}@ks.informatik.uni-kiel.de

Abstract. We propose a model for learning the articulated motion of human arm and hand grabbing. The goal is to generate plausible trajectories of joints that mimic the human movement using deformation information. The trajectories are then mapped to a constraint space. These constraints can be the space of start and end configuration of the human body and task-specific constraints such as avoiding an obstacle, picking up and putting down objects. Such a model can be used to develop humanoid robots that move in a human-like way in reaction to diverse changes in their environment and as a priori model for motion tracking. The model proposed to accomplish this uses a combination of principal component analysis (PCA) and a special type of a topological map called the dynamic cell structure (DCS) network. Experiments on arm and hand movements show that this model is able to successfully generalize movement using a few training samples for free movement, obstacle avoidance and grabbing objects. We also introduce a method to map the learned human movement to a robot with different geometry using reinforcement learning and show some results.

1 Introduction

Human motion is characterized as being smooth, efficient and adaptive to the state of the environment. In recent years a lot of work has been done in the fields of robotics and computer animation to capture, analyze and synthesize this movement with different purposes [1–3]. In robotics there has been a large body of research concerning humanoid robots. These robots are designed to have a one to one mapping to the joints of the human body but are still less flexible. The ultimate goal is to develop a humanoid robot that is able to react and move in its environment like a human being. So far the work that has been done is concerned with learning single gestures like drumming or pole balancing which involves restricted movements primitives in a simple environment or a preprogrammed movement sequence like a dance. An example where more adaptivity is needed would be a humanoid tennis robot which, given its current position and pose and the trajectory of the incoming ball, is able to move in a human-like way to intercept it. This idea enables us to categorize human movement learning from simple to complex as follows: (A) Imitate a simple gesture, (B) learn a sequence of gestures to form a more complex movement, (C) generalize movement over the range allowed by the human body, and (D) learn different classes of movement specialized for specific tasks (e.g. grasping, pulling, etc.).

This paper introduces two small applications for learning movement of type (C) and (D). The learning components of the proposed model are not by themselves new. Our contribution is presenting a supervised learning algorithm which learns to imitate human movement that is specifically more adaptive to constraints and tasks than other models. This also has the potential to be used for motion tracking where more diverse changes in movement occur. We will call the state of the environment and the body which affects the movement as constraint space. This may be as simple as object positions which we must reach or avoid, a target body pose or more complex attributes such as the object's orientation and size when grabbing it. The first case we present is generating realistic trajectories of a simple kinematic chain representing a human arm. These trajectories are adapted to a constraint space which consists of start and end positions of the arm as shown in fig. 4. The second case demonstrates how the learning algorithm can be adapted to the specific task of avoiding an obstacle where the position of the obstacle varies. The third case demonstrates how hand grabbing can be adapted to different object sizes and orientations.

The model accomplishes this by aligning trajectories. A trajectory is the sequence of body poses which change in time from the start to the end of a movement. Aligning trajectories is done by scaling and rotation transforms in angular space which minimizes the distance between similar poses between trajectories. After alignment we can analyze their deformation modes which describe the principal variations of the shape of trajectories. The constraint space is mapped to these deformation modes using a topological map.

In addition to generating adaptive human movement, We also introduce in this paper a reinforcement learning algorithm which enables us to transfer the learned human movement to a robot with a different embodiment by using reinforcement learning. This is done by learning similar trajectories of the human hand and the robot's end effector. A reward function measures this similarity as a mixture of constraint fitting in the robot's workspace and the similarity of the trajectory shape to the human's. The reinforcement learning algorithm explores trajectory space using a spectral representation of trajectories in order to reduce the state space dimensionality. A special type of DCS networks called QDCS is used for learning.

The Combination of adaptive movement learning and movement transfer enables a complete movement learning and transfer system architecture. This consists of two neural networks (NN) as shown in fig. 1. The first network reconstructs human movement and the second transforms this movement to the robot space using reinforcement learning.

Next, we describe an overview of the work done related to movement learning and transferring and compare them with the proposed model. After that the adaptive movement algorithm will be presented followed by the transfer algorithm and then experimental results.

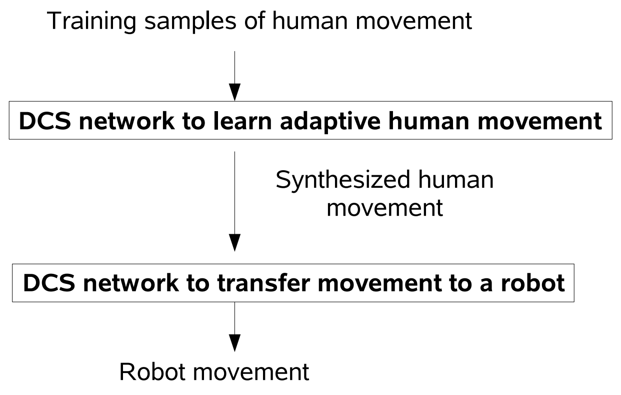
Training samples of human movement
DCS network to learn adaptive human movement
Synthesized human movement
DCS network to transfer movement to a robot
Robot movement

Fig. 1. Architecture for learning movement and transferring it to a robot.

2 State of the art

There are two representations for movements: pose based and trajectory based. We will describe next pose based methods.

Generative models of motion have been used in [2, 1] in which a nonlinear dimensionality reducing method called Scaled Gaussian Latent Variable Model (SGPLVM) is used on training samples in pose space to learn a nonlinear latent space which represents the probability distribution of each pose. Such a likelihood function was used as a prior for tracking in [1] and finding more natural poses for computer animation in [2] that satisfy constraints such as that the hand has to touch some points in space. Another example of using a generative model for tracking is [4] in which a Bayesian formulation is used to define a probability distribution of a pose in a given time frame as a function of the previous poses and current image measurements. This prior model acts as a constraint which enables a robust tracking algorithm for monocular images of a walking motion. Another approach using Bayesian priors and nonlinear dimension reduction is used in [5] for tracking.

After reviewing pose probabilistic methods, we describe in the following trajectory based methods. Schaal [3] has contributed to the field of learning movement for humanoid robots. He describes complex movements as a set of movement primitives (DMP). From these a nonlinear dynamic system of equations are defined that generate complex movement trajectories. He described a reinforcement learning algorithm that can efficiently optimize the parameters (weights) of DMPs to learn to imitate a human in a high dimensional space. He demonstrated his learning algorithm for applications like drumming and a tennis swing.

To go beyond a gesture imitation, in [6] a model for segmenting and morphing complex movement sequences was proposed. The complex movement sequence is divided into subsequences at points where one of the joints reaches zero velocity. Dynamic programming is used to match different subsequences in which some of these key movement features are missing. Matched movement segments are then combined with each other to build a morphable motion trajectory by calculating spatial and temporal displacement between them. For example, morphable movements are able to naturally represent movement transitions between different people performing martial arts with different styles.

Another aspect of motion adaptation and morphing with respect to constraints comes from computer graphics on the topic of re-targeting. As an example, Gleicher [7] proposed a nonlinear optimization method to re-target a movement sequence from one character to another with an identical structure but different segment lengths. The problem is to satisfy both the physical constraints and the smoothness of movement. Physical constraints are contact with other objects like holding the box.

The closest work to the model presented in this paper is done by Banarer [8]. He described a method for learning movement adaptive to start and end positions. His idea is to use a topological map called Dynamic Cell Structure (DCS) network [9]. The DCS network learns the space of valid arm configurations. The shortest path of valid configurations between the start and end positions represents the learned movement. He demonstrated his algorithm to learn a single gesture and also obstacle avoidance for a single fixed obstacle.

3 Contribution

The main difference between pose based methods and our approach is that instead of learning the probability distribution in pose space, we model the variation in trajectory space (each trajectory being a sequence of poses). This representation enables us to generate trajectories that vary as a function of environmental constraints and to find a more compact representation of variations than allowed by pdfs in pose space alone. Pose pdfs would model large variations in trajectories as a widely spread distribution which makes it difficult to trace the sequence of legal poses that satisfy the constraints the human actually makes without some external reference like motion sequence data.

Our approach models movement variation as a function of the constraint space. However, style based inverse kinematics as in [2] selects the most likely poses that satisfy these constraints. This works well as long as the pose constraints do not deviate much from the training data. This may be suitable for animation applications but our goal here is to represent realistic trajectories adapted to constraints without any explicit modeling. Banarer [8] uses also a pose based method and the model he proposed does not generalize well because as new paths are learned between new start and end positions, the DCS network grows very quickly and cannot cope with the curse of dimensionality. Our DCS network generalizes over trajectory space not poses enabling more adaptivity.

Gleicher [7] defines an explicit adaptation model which is suitable to generate a visually appealing movement but requires fine tuning by the animator because it may appear unrealistic. This is because it explicitly morphs movement using a prior model rather than learning how it varies in reality as done in [2].

In the case of Schaal [3], we see that DMPs although flexible are not designed to handle large variations in trajectory space. This is because reinforcement learning adapts to a specific target human trajectory.

Morphable movements [6] define explicitly the transition function between two or more movements without considering the constraint space. Our method can learn the nonlinear mapping between constraint space and movements by training from many samples. The variation of a movement class is learned and not explicitly pre-defined.

To sum up, we have a trajectory based learning model which learns the mapping between constraints and movements. The movement can be more adaptive and generalizable over constraint space. It learns movements from samples and avoids explicit modeling which may generate unrealistic trajectories.

4 Learning Model

After describing the problem, this section will develop the concept for learning movement and then it describes how this model is implemented.

In order to develop a system which is able to generalize movement, a number of reductions have to be made to the high dimensional space of start - end configurations or any other environmental constraints. This reduction is done in two steps. The first step is to learn the mechanics of movement itself and the second is to learn how movement changes with start - end configuration. The mechanics of movement are called *intrinsic features*. The changes of intrinsic feature with respect to relative position are called *extrinsic features*. The intrinsic features describe movement primitives that are characteristic for the human being. These features are the following:

1. The acceleration and velocity of joints as they move through space. For example a movement generally begins by a joint accelerating at the start then decelerating as it nears its end position.
2. The nonlinear path taken by the joints to reach their destination. This is what characterizes smooth human movement. Otherwise the movement will look rigid similar to inverse kinematics used by robots. This nonlinearity is not only seen in simple movements like moving from point a to b but also it can be seen in more complex movements for which it is necessary like obstacle avoidance and walking.
3. The coordination of joints with respect to each other in time. This means that joints co-deform in space and time working together to achieve their goal.

After modeling intrinsic features, extrinsic features can be characterized as the variation of intrinsic feature in the space of all possible start and end positions of the joints and any environmental constraints such as obstacle positions. Extrinsic features describe:

1. The range of freedom of joints.

2. The movement changes with respect to rotation and scale. As an example, we can consider how the movement changes when we draw the letter A in different directions or with different sizes.

The difference between intrinsic and extrinsic features that characterizes movement enables the formulation of a learning model. This model consists of two parts: The first part is responsible for learning intrinsic features which uses principal component analysis (PCA). It is applied on the aligned trajectories of the joints to reduce the dimensionality. The second part models the extrinsic features using a special type of an adaptive topological map called the dynamic cell structure (DCS) network. The DCS learns the nonlinear mapping from the extrinsic features to intrinsic features that are used to construct the correct movement that satisfies these extrinsic features.

In the following subsections we will take a detailed look at these mechanisms.

4.1 Intrinsic features using PCA

The algorithm which will be used to extract intrinsic features consists of the following steps: (1) Interpolation, (2) Sampling, (3) Conversion to orientation angles, (4) Alignment and (5) Principal component analysis. The main step is alignment in which trajectories traced by joints in space are aligned to each other to eliminate differences due to rotation and scale exposing the mechanics of movement which can then be analyzed by PCA. In the following paragraphs each step of the algorithm will be explained in detail as well as the reasons behind it. As an example, we will assume throughout this paper that there is a kinematic chain of 2 joints: shoulder and elbow. Each joint has 2 degrees of freedom (ϕ, θ) which represent the direction of the corresponding limb in spherical coordinates.

To perform statistical analysis, we record several samples of motion sequences. In each motion sequence the 3D positions of the joints are recorded with their time. Let us define the position measurements of joints of a movement sequence (k) as $\{(x_{i,j,k}, y_{i,j,k}, z_{i,j,k}, t_{i,j,k})\}$ where (x, y, z) is the 3D position of the joint, t is the time in milliseconds from the start of the motion. The index i is the position (frame), j specifies the marker and k specifies the the movement sequence.

The first step is to interpolate between the points of each movement sequence. Usually a 2nd degree B-spline is sufficient to obtain a good interpolation. We end up with a set of parametric curves $\{\mathbf{p}_k(t)\}$ for each motion sequence k where $\mathbf{p}_k(t)$ returns the position vector of all the joints at time t.

The second step is to sample each $\mathbf{p}_k(t)$ at equal time intervals from the start of the sequence $t = 0$ to its end $t = t_{end_k}$. Let n be the number of samples then we form a vector of positions $\mathbf{v}_k = [\mathbf{p}_{1,k}, \mathbf{p}_{2,k} \ldots \mathbf{p}_{n,k}]$ where $\mathbf{p}_{i,k} = \mathbf{p}_k((\frac{i-1}{n-1})t_{end_k})$. This regular sampling at equal time intervals enables us to represent trajectories with a fixed length vector which facilitates statistical analysis on a population of such vectors. This vector form also represents implicitly the acceleration and velocity of these paths through variability of distances between points. This is the reason why time was used as the variable in the parametric curves. In cases where motion is more complex and consists of many curves, this method automatically samples more points of high curvature than points of lower curvature. This places comparable corner points of complex paths near each other and thus does not normally necessitate more complex registration techniques to align the curves as long as the trajectories have up to 4-5 corners.

The third step is to convert the Euclidean coordinates $\mathbf{v}_k$ to direction angles in spherical coordinates $\mathbf{S}_k = [\mathbf{s}_{1,k}, \mathbf{s}_{2,k}, \dots \mathbf{s}_{n,k}]$ where $\mathbf{s}_{i,k}$ is the vector of direction angles for all the joints. This means that a given joint j in a kinematic chain with some position $\mathbf{p}_j = (x_j, y_j, z_j)$ is attached to its parent with a position $\mathbf{p}_{j-1} = (x_{j-1}, y_{j-1}, z_{j-1})$ and since the distance D between them is constant we need only to represent the direction of joint j with respect to joint $j-1$. This can be done by taking the relative coordinates $\Delta\mathbf{p} = \mathbf{p}_j - \mathbf{p}_{j-1}$ and then convert $\Delta\mathbf{p}$ to spherical coordinates $(\rho_j, \phi_j, \theta_j)$ where $\rho_j = D$ is constant. After that, we take only the direction angles (ϕ_j, θ_j) as the position of the joint. For a kinematic chain of m joints we will need $m-1$ such direction angles because the first joint is the basis of the chain therefore has no direction.

The forth step is to align the paths taken by all the joints with respect to each other. This alignment makes paths comparable with each other in the sense that all extrinsic features are eliminated leaving only the deformations of the path set from the mean. In order to accomplish this, we first convert every direction point (ϕ_j, θ_j) of a joint j to a 3D unit vector $\hat{\mathbf{u}}_j = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$ that corresponds to the direction of the joint at some point in time. With this representation we can imagine that a moving joint traces a path on a unit sphere $\hat{\mathbf{u}}_j(t)$ as the paths shown in fig. 2. Given a kinematic chain of m joints moving together, we can represent these moving joints as $m-1$ moving unit vectors $\mathbf{U}(t) = [\hat{\mathbf{u}}_1(t), \dots \hat{\mathbf{u}}_j(t), \dots \hat{\mathbf{u}}_{m-1}(t)]$. $\mathbf{S}_k$ samples the path at equal time intervals $t_0, \dots, t_{n_k}$ that correspond to $\mathbf{U}_i = \mathbf{U}(t_i), i = 0 \dots t_{n_k}$. This enables us to represent the path as a matrix of direction vectors $\mathbf{W} = [\mathbf{U}_0, \dots \mathbf{U}_{n_k}]'$. The reason why we use direction vector representation $\mathbf{W}$ is because it facilitates alignment of paths with each other by minimizing their distances over rotation and scale transforms. To accomplish this, we define a distance measure between two paths $\mathbf{W}_1, \mathbf{W}_2$ as the mean distance between corresponding direction vectors of both paths (i.e. corresponding joints j at the same time ti)

$$pathdist(\mathbf{W}_1, \mathbf{W}_2) = \frac{1}{|\mathbf{W}_1|} \sum_{\forall(\hat{\mathbf{u}}_{i,j} \epsilon \mathbf{W}_1, \hat{\mathbf{v}}_{i,j} \epsilon \mathbf{W}_2)} dist(\hat{\mathbf{u}}_{i,j}, \hat{\mathbf{v}}_{i,j}) \tag{1}$$

where the distance between two direction vectors is simply the angle between

$$dist(\hat{\mathbf{u}}, \hat{\mathbf{v}}) = cos^{-1}(\hat{\mathbf{u}} \cdot \hat{\mathbf{v}}) \tag{2}$$

Two transforms are used to minimize the distances between the paths:

1. Rotation (R): When we multiply each direction vector in the path with a rotation matrix R, we can rotate the whole path as shown in fig. 2.
2. Scaling: We can scale the angles between path points. This assumption is made by the observation that movements are scalable. For example, when

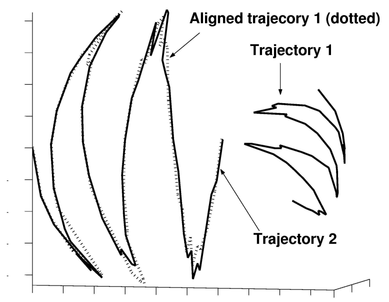

Fig. 2. Alignment of two trajectories by scale and rotation. The trajectories are a sequence of direction vectors tracing curves on a unit sphere.

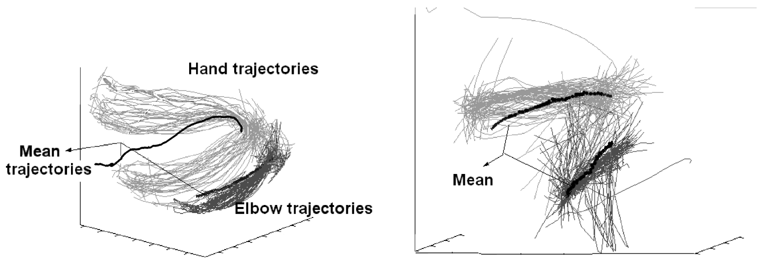

Fig. 3. Example of aligning a training set of trajectories represented as direction vectors tracing curves on a unit sphere. Left before alignment and right after. We see how the hand trajectories cluster together and the mean becomes smoother.

we draw a letter smaller and then bigger, we basically move in the same way but with larger angles as shown in fig. 2. This is of course a simplifying assumption that is not exactly true but it helps us later to fit start-end positions of variable arc-length distances. To scale the path we simply choose a reference direction vector on the path and then shift each direction vector in the direction of the arc length between them by multiplying the arc length θ with a scale factor s as depicted in fig. 2.

Minimizing distances can be done by simple gradient descent over the scale parameter s and rotation angles around the three axes $\theta_x, \theta_y, \theta_z$ defining a rotation matrix $R = R_x(\theta_x)R_y(\theta_y)R_z(\theta_z)$. When extending this algorithm to align more than two paths, we can do that by computing the mean path $\overline{\mathbf{W}}$ and then fitting all the sample paths $\{\mathbf{W}_1, \ldots, \mathbf{W}_p\}$ to the the mean. We repeat this cycle until the mean path $\overline{\mathbf{W}}$ converges. The mean path is initialized by an arbitrary sample $\overline{\mathbf{W}} = \mathbf{W}_j$. It is computed from aligned samples in each iteration step

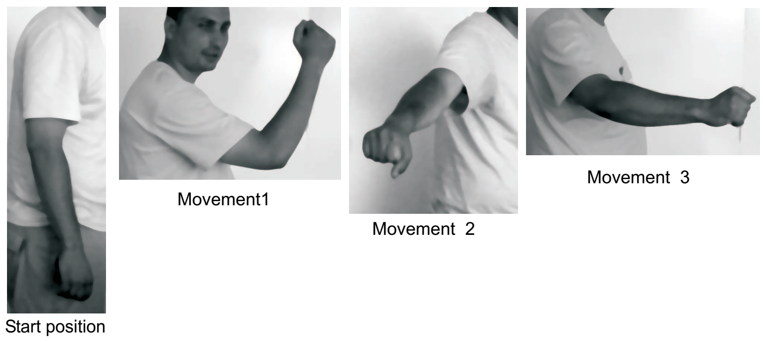

Fig. 4. Three movements of the arm that all begin with the same start position (left image), the rest are end positions.

by summing the corresponding direction vectors from all the sample paths and then normalizing the sum $\overline{\mathbf{W}} = \frac{\sum_i \mathbf{w}_i}{|\sum_i \mathbf{w}_i|}$. An example of aligning is in fig. 3.

The fifth step is to convert the aligned trajectories back to the angular representation and form a data matrix of all p aligned motion sequences $X = [\mathbf{S}_1^T \dots \mathbf{S}_k^T \dots \mathbf{S}_p^T]^T$. Principal component analysis is applied on X yielding latent vectors $\mathbf{\Psi} = [\psi_1, \psi_2, \dots, \psi_n]$. Only the first q components are used where q is chosen such that the components cover a large percentage of the data $\mathbf{\Psi}_q = [\psi_1, \psi_2, \dots, \psi_q]$. Any point in eigenspace can be then converted to the nearest plausible data sample using the following equation

$$\mathbf{S} = \overline{\mathbf{S}} + \mathbf{\Psi}_q \mathbf{b} \tag{3}$$

where $\overline{\mathbf{S}} = \frac{1}{p}\sum_{k=1}^{p} \mathbf{S}_k$ and $\mathbf{b}$ is the column vector of an eigenpoint.

The inverse transform from eigenspace to trajectories is approximated by

$$\mathbf{b} = \mathbf{\Psi}_q'(\mathbf{S} - \overline{\mathbf{S}}) \tag{4}$$

The latent coordinates $\mathbf{b}$ represent the linear combination of deformations from the average paths taken by the joints. An example of that can be seen in fig. 5. In this example, the thick lines represent the mean path and the others represent ± 3 standard deviations in the direction of each eigenvector which are called modes. The first mode represents the twisting of the hand's path around the elbow and shoulder. The second mode shows the coordination of angles when moving the hand and elbow together. The third mode represent the bulginess of the path taken by the hand and shoulder around the middle. We see that these deformation modes have meaningful mechanical interpretations.

4.2 Extrinsic features using DCS

PCA performs a linear transform (i.e. rotation and projection in (3)) which maps the trajectory space into the eigenspace. The mapping between constraint space

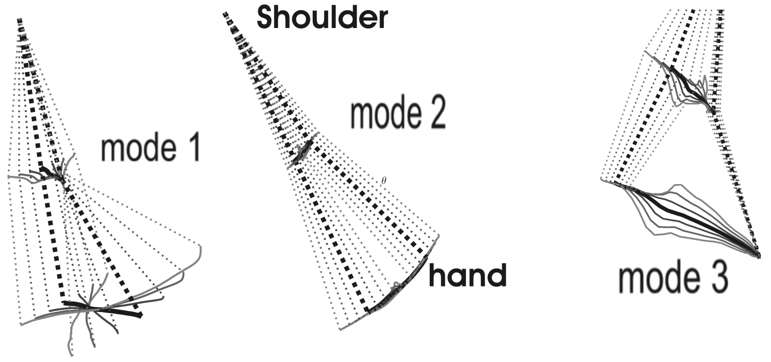

Fig. 5. The first three variation modes of a kinematic chain representing the shoulder, elbow and hand constructed in 3D space. The middle thick line is the mean trajectory and the others represent $\pm1, \pm2, \pm3$ standard deviations along each eigenvector.

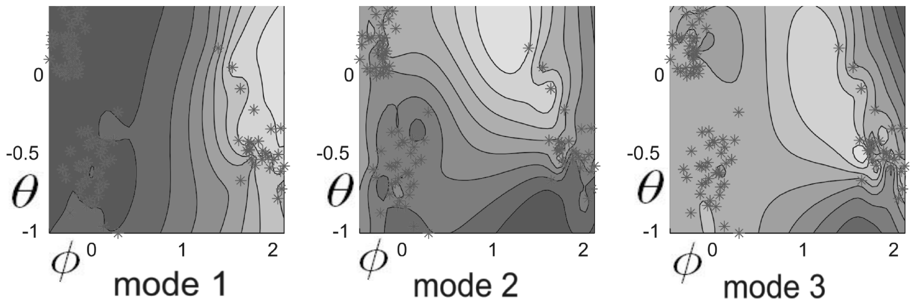

Fig. 6. Distribution of eigenvalues (bright regions represent maxima) in the angular space of the end position of the hand.

and eigenspace is generally nonlinear. To learn this mapping we use a special type of self organizing maps called Dynamic Cell Structure which is a hybrid between radial basis networks and topologically preserving maps [9]. DCS networks have many advantages: They have a simple structure which makes it easy to interpret results, they adapt efficiently to training data and they can cope with changing distributions. They consist of neurons that are connected to each other locally by a graph distributed over the input space. These neurons also have radial basis functions which are Gaussian functions used to interpolate between these neighbors. The DCS network adapts to the nonlinear distribution by growing dynamically to fit the samples until some error measure is minimized. When a DCS network is trained, the output $\mathbf{b}_{DCS}(\mathbf{x})$ which is a point in eigenspace can be computed by summing the activations of the best matching neuron (i.e. closest) to the input vector $\mathbf{x}$ representing a point in constraint space and the local neighbors to which it is connected by an edge which is defined by the function $A_p(\mathbf{x})$. The output is defined as

$$\mathbf{b}_{DCS}(\mathbf{x}) = f_P^{nrbf}(\mathbf{x}) = \frac{\sum_{i \in A_p(\mathbf{x})} \mathbf{b}_i h(\| \mathbf{x} - \mathbf{c}_i \| / \sigma_i)}{\sum_{j \in A_p(\mathbf{x})} h(\| \mathbf{x} - \mathbf{c}_j \| / \sigma_j)}, \tag{5}$$

where $\mathbf{c}_i$ is the receptive center of the neuron i, $\mathbf{b}_i$ represents a point in eigenspace which is the output of neuron i, h is the Gaussian kernel and σ_i is the width of the kernel at neuron i.

The combination of DCS to learn nonlinear mapping and PCA to reduce dimension enables us to reconstruct trajectories from $\mathbf{b}(\mathbf{x})$ using (3) which are then fitted to the constraint space by using scale and rotation transformations. For example, a constructed trajectory is fitted to a start and end position.

When using the network to generate new motion paths, the start-end positions $\mathbf{\Theta}$ are given to the network. It returns the deformation modes $\mathbf{b}$ of the given start-end position. We must use $\mathbf{b}$ to reconstruct the path between the start and positions given by $\mathbf{\Theta}$. This is accomplished by converting $\mathbf{b}$ to angular representation $\mathbf{S}$ given by (3). $\mathbf{S}$ is converted to direction vector representation $\mathbf{W}$. We take the start and end positions of $\mathbf{W}$ and find the best rotation and scale transform that fits it to $\mathbf{\Theta}$ using the same method shown in the previous section. The resulting path represents the reconstruction that contains both intrinsic and extrinsic features from the learning model.

5 Learning Model for Transferring Movement to a robot

After building a model which can generate human movement as a function of constraint space, we will introduce in this section a learning algorithm to transfer this movement to the robot. The main problem here is how to transfer human movement to a manipulator with a different geometry and degrees of freedom. There are many solutions proposed in literature which basically fall into two categories: The first class looks only at the effects of actions on the environment [10]. If the effects of goal sequences are the same on the environment then the robot movement is considered equivalent to the humans'. The other category defines some ad hoc function which measures the similarity between the robot and human movement [6]. This function can also be a mixture of degree of goal satisfaction and pose similarity. In this paper we will solve this problem by using a reinforcement learning approach rather than explicitly defining the function. This approach will mix the two mapping categories. Specifically, we will use similarities of trajectories between the end effector and the hand. This simplifies the problem and enables us to define a meaningful intuitive mapping for any manipulation task involving manipulation of objects. The reward function r will be a weighted mixture of similarities of trajectories and constraint satisfaction as follows

$$r(\mathbf{u}, \mathbf{v}, C) = \alpha_1 f_1(\mathbf{u}, \mathbf{v}) + \alpha_2 f_2(\mathbf{v}, C) \tag{6}$$

where $\mathbf{u}$ is the trajectory of the human hand, $\mathbf{v}$ is the trajectory of the robot end effector. C is the constraints to be satisfied in the robots workspace. α_1, α_2 are wieghts, f_1 measures the similarity between the shapes of trajectories and f_2 mesures constraint satisfaction in the robot space.

This approach is similar to programming by demonstration (POD) in [11, 12] where a human teaches the robot to move in a similar way to avoid obstacles for example. The reinforcement learning approach suffers from the drawback that when the action space has a high dimensionality, the solution will not converge in a reasonable time. For this reason we will use a frequency representation of trajectories where only the first few components are used. Specifically, we choose a discrete cosine transform representation of trajectories which uses only real numbers. A trajectory is sampled at N points in 3D space equidistant in time $\mathbf{p}_n, n = 1 \dots N$. This transform is

$$\mathbf{a}_k = w_k \sum_{n=1}^{N} \mathbf{p}_n cos(\frac{\pi(2n-1)(k-1)}{2N}), k = 1, \dots, N, \tag{7}$$

$$w_k = \{ ^{\frac{1}{\sqrt{N}}, k=1}_{\sqrt{\frac{2}{N}}, 2 \leq k \leq N} \tag{8}$$

The reinforcement learning represents trajectories through the parameters $\mathbf{a}_k, k = 1...m$ that where m is some small dimension. The robot trajectory is reconstructed using inverse discrete cosine transform applied on a zeros padded vector of length N: $(\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m, 0, \dots, 0)$. This reduces the action space dimension enabling a fast converging solution. This is the advantage of using a spectral representation of trajectories for reinforcement learning. The state is represented as the vector $(\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m, \mathbf{C}) = (\mathbf{\Theta}, \mathbf{C})$ where $\mathbf{C}$ is the constraint that is satisfied by the trajectory that is reconstructed from $\mathbf{\Theta}$. The action applied on some state is some small increment $\Delta\mathbf{\Theta}$ added to $\mathbf{\Theta}$ to get a new modified trajectory.

The reinforcement learning begins exploring the states of trajectories that optimize the reward function by incrementally modifying the trajectory parameters for some initial constraint $\mathbf{C}_0$. Once the optimal trajectory is found, new optimal trajectories for a new constraint $\mathbf{C}_1$ close to $\mathbf{C}_0$ is learned. This will exploit the optimal trajectory learned for $\mathbf{C}_0$ as a prior to learn quickly the optimal trajectory of $\mathbf{C}_1$. The process continues for all constraints in constraint space that are needed to be learned. When fully trained, we can use the QDCS network to find the best robot trajectory that satisfies a given constraint.

A continuous version Q-learning is used to learn the optimal trajectory. In it we replace the discrete Q table in normal Q learning with a DCS network that learns the Q values online as demonstrated in [9]. The QDCS network uses online learning to update its values. The algorithm is depicted in figure 7.

```
Initialize the QDCS network arbitrarily.
For each constraint $\mathbf{C}$ do
   Initialize the start state $\mathbf{\Theta}$
   Repeat
      choose an action $\Delta\mathbf{\Theta}$, observe $\mathbf{\Theta}' = \mathbf{\Theta} + \Delta\mathbf{\Theta}$ and reward $r$
      update online $QDCS(\mathbf{C}, \mathbf{\Theta}, \Delta\mathbf{\Theta})$ by adding:
         $\alpha[r + \gamma max_{\Delta\mathbf{\Theta}'} QDCS(\mathbf{C}, \mathbf{\Theta}', \Delta\mathbf{\Theta}') - QDCS(\mathbf{C}, \mathbf{\Theta}, \Delta\mathbf{\Theta})]$
      $\mathbf{\Theta} \leftarrow \mathbf{\Theta}'$
   Until $\mathbf{\Theta}$ converges to a good solution
```

Fig. 7. Learning optimal trajectories using QDCS

6 Experiments

In order to record arm movements, a marker-based stereo tracker was developed in which two cameras track the 3D position of three markers placed at the shoulder, elbow and hand at a rate of 8 frames per second. This was used to record trajectory samples. Two experiments were conducted to show two learning cases: moving between two positions and avoiding an obstacle.

The first experiment demonstrates that our learning model reconstructs the nonlinear trajectories in the space of start-end positions. A set of 100 measurements were made for an arm movement consisting of three joints. The movements had the same start position but different end positions as shown in Fig. 4.

The first three eigenvalues have a smooth nonlinear unimodal distribution with respect to the start-end space as shown in Fig. 6. The first component explained 72% of the training samples, the second 11% and the third 3%.

The performance of the DCS network was first tested by a k-fold cross validation on randomized 100 samples. This was repeated for $k = 10$ runs. In each run the DCS network was trained and the number of neurons varied between 6 to 11. The average distance between the DCS-trajectory and the data sample was 3.9° and the standard deviation was 2.1°. This shows that the DCS network was able to generalize well using only a small sample size (about 100).

We can compare with Banarer [8] who fixed the DCS network with an upper bound of 15 neurons to learn a single gesture and not many as in our experiment. He used simulated data of 70 samples with a random noise of up to 5° and the mean error was 4.3° compared to our result of 3.9° on real data. The measurement error of the tracker is estimated to be 4.6° standard deviation which accounts for the similar mean errors. This shows that our model scales well.

Next, we demonstrate the algorithm for obstacle avoidance. In this case 100 measurements were taken for the arm movement with different obstacle positions as shown in Fig. 8. The black lines show the 3D trajectory of the arm avoiding the obstacle which has a variable position determined by the distance B. We see how the hand backs away from the obstacle and the elbow goes down and then upward to guide the hand to its target. A is the Euclidian distance between the start and end positions of the hand. The grey lines represent a free path without

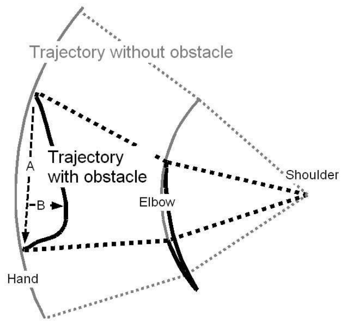

Fig. 8. Trajectory for obstacle avoidance in 3D space.

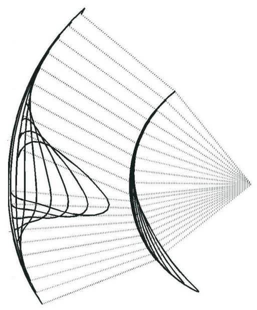

Fig. 9. Variation of arm trajectory with respect to the obstacle.

obstacles. In this case we need to only take the first eigenvector from PCA to capture the variation of trajectories due to obstacle position. This deformation mode is shown in Fig. 9 We define the relative position of the obstacle to the movement as simply $p = \frac{B}{A}$. The DCS network learns the mapping between p and the eigenvalue with only 5 neurons. The learned movement can thus be used to avoid any obstacle between the start and end positions regardless of orientation or movement scale. This demonstrates how relatively easy it is to learn new specialized movements that are adaptive to constraints.

Finally, this model was demonstrated on hand grabbing. In this case 9 markers were placed on the hand to track the index and thumb fingers using a monocular camera as in Fig. 10. The 2D positions of the markers were recorded at a rate of 8.5 frames per second from a camera looking over a table. The objects to be grabbed are placed over the table and they vary by both size and orientation. The size ranged from 4 to 12 cm and orientation ranged from 0 to 60 degrees as depicted in Fig. 11 and 12. The tracker recorded 350 grabbing samples of which 280 was used for training the DCS and 70 for testing. The DCS learned the variation of movement with 95 neurons and PCA reduced the dimension from 600 to just 23. The first two modes characterize variation of scale and orientation as shown in Fig. 10. Fig. 11 and 12 depict an example comparison between grabbing movement generated by the DCS and an actual sample. Below we used two measures that characterize well grabbing: distance between the tips of the index finger and the thumb and the direction of the index finger's tip with respect the the direction of the arm. We see that the DCS and sample profiles look very similar. In general, the model's root mean square error for the first measure was 18 pixels for a 800×600 images and $8.5°$ for the second measure.

After presenting the experiments for movement generation, we demonstrate some results for movement transfer. Fig. 13 shows a 2D human trajectory that a robot has to learn. The markers $(*)$ are the constraints which represents the start and end position that the robot must maintain. The discrete cosine transform uses 8 coefficients: 4 for the x dimension and 4 for the y dimension. The algorithm

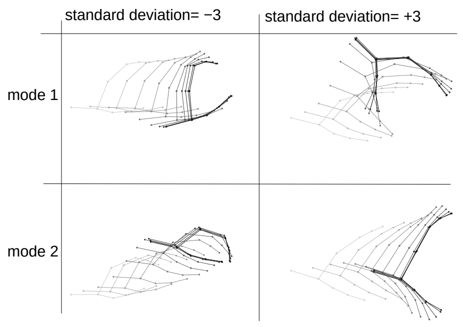

Fig. 10. The first two variation modes of grabbing.

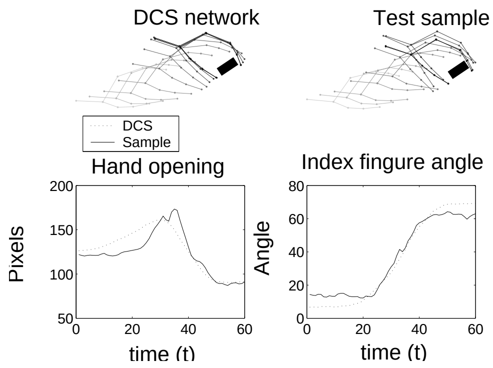

Fig. 11. Comparison between DCS and and a grabbing movement for a 4 cm object at 60° with respect to the hand.

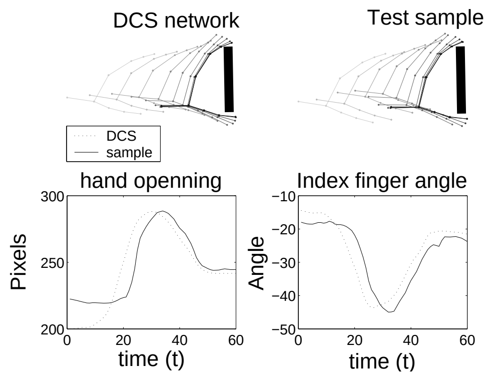

Fig. 12. Comparison between DCS and and a grabbing movement for a 12 cm object at 0°.

was able to learn the nearest trajectory after about 2000 itarations. The QDCS was set to an upper bound of 200 neurons. The algorithm selects a random action about 10% of the time and the action of maximum Q value the rest of the time. This shows that the algorithm is able to learn in a reasonable time a trajectory because the human motion and the constraints act as strong priors to guide the low dimensional frequency representation of trajectories. Adapting to more constraints is left as future work.

7 Conclusion

We proposed a learning model for generation of realistic articulated motion. The model characterizes deformation modes that vary according to constraint space. A combination of DCS network to learn the nonlinear mapping and PCA to reduce dimensionality enables us to find a representation that can adapt to con-

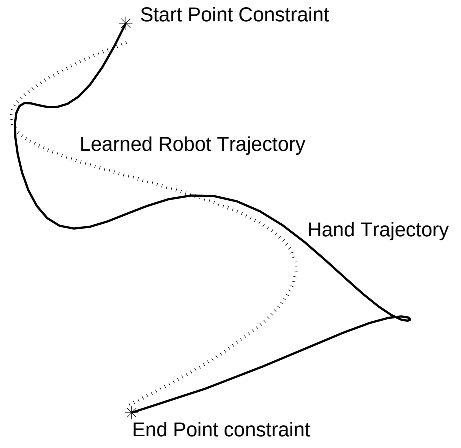

Fig. 13. Learned trajectory using start and end position constraints in 2D space

straint space with a few samples. This trajectory based method is more suited for movement generation than pose based methods which are concerned with defining priors for good fitting with image data such as tracking. The proposed method models variation of movement with respect to constraints in a more clear way than the previously proposed methods. The potential uses of our method is in developing humanoid robots that are reactive to their environment and also motion tracking algorithms that use prior knowledge of motion to make them robust. Three small applications towards that goal were experimentally validated. We also proposed a trajectory based method that transfers the human movement to a manipulator of a different embodiment using reinforcement leraning. The method uses QDCS to exploit and expore the space of trajectories that fit to constraints specified in robot space. This represents a natural extension of the first algorithm that enables adaptive movement that is retargetable to any robot manipulator system.

ACKNOWLEDGMENTS: The work presented here was supported by the the European Union, grant COSPAL (IST-2003-004176). However, this paper does not necessarily represent the opinion of the European Community, and the European Community is not responsible for any use which may be made of its contents.

References

1. Urtasun, R., Fleet, D.J., Hertzmann, A., Fua, P.: Priors for people tracking from small training sets. In: International Conference on Computer Vision (ICCV). (2005) 403–410

2. Grochow, K., Martin, S.L., Hertzmann, A., Popovic;, Z.: Style-based inverse kinematics. ACM Trans. Graph. **23**(3) (2004) 522–531
3. Schaal, S., Peters, J., Nakanishi, J., Ijspeert, A.: Learning movement primitives. In: International Symposium on Robotics Research (ISPR2003), Springer Tracts in Advanced Robotics, Ciena, Italy (2004)
4. Sidenbladh, H., Black, M.J., Fleet, D.J.: Stochastic tracking of 3d human figures using 2d image motion. In: Proceedings of the 6th European Conference on Computer Vision (ECCV '00), London, UK, Springer-Verlag (2000) 702–718
5. Sminchisescu, C., Jepson, A.: Generative modeling for continuous non-linearly embedded visual inference. In: Proceedings of the twenty-first International Conference on Machine Learning (ICML '04), New York, NY, USA, ACM Press (2004)
6. Ilg, W., Bakir, G.H., Mezger, J., Giese, M.A.: On the repersenation, learning and transfer of spatio-temporal movement characteristics. International Journal of Humanoid Robotics (2004)
7. Gleicher, M.: Retargeting motion to new characters. In: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98), New York, NY, USA, ACM Press (1998) 33–42
8. Banarer, V.: Struktureller Bias in neuronalen Netzen mittels Clifford-Algebren. Technical Report 0501, Technische Fakultät der Christian-Albrechts-Universität zu Kiel, Kiel (2005)
9. Bruske, J., Sommer, G.: Dynamic cell structure learns perfectly topology preserving map. Neural Computation **7**(4) (1995) 845–865
10. Nehaniv, C., Dautenhahn, K.: Of hummingbirds and helicopters: An algebraic framework for interdisciplinary studies of imitation and its applications. In: World Scientific Press. (1999)
11. Jacopo Aleotti, S.C.: Trajectory clustering and stochastic approximation for robot programming by demonstration. In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Edmonton, Alberta, Canada (2005)
12. J. Aleotti, S. Caselli, M.R.: Toward programming of assembly tasks by demonstration in virtual environments. In: 12th IEEE Int. Workshop on Robot and Human Interactive Communication, Millbrae, California (USA) (2003)