INSTITUT FÜR INFORMATIK

A Model for Learning and Imitating Human Movement

Stephan Al-Zubi, Gerald Sommer

Bericht Nr. 0706
Juli 2007

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

A Model for Learning and Imitating Human Movement

Stephan Al-Zubi, Gerald Sommer

Bericht Nr. 0706
Juli 2007

e-mail: sa@cs.informatik.uni-kiel.de,
gs@cs.informatik.uni-kiel.de

Abstract

We propose a model for learning the articulated motion of human arm and hand grasping. 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 of different sizes. 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. 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 grasping objects. We also introduce a method to Imitate the learned human movement by robot with different geometry using reinforcement learning. Obstacle avoidance is chosen as a case study. The reinforcement learning uses a real valued Q-learning and a DCS network to learn the trajectory of the end effector by trial and error. The learned robot trajectories are used to speed up the learning process of subsequent movements. The reward function of the reinforcement learning measures constraint satisfaction and similarity to human trajectory. The reinforcement learning uses a dimensionality reduced representation of trajectories using a discrete cosine transform. This enables the random exploration of reinforcement learning to converge quickly to a solution. Experiments on real and simulated robot environment show that the method is efficient and is able to quickly generalize robot movements.

Contents

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, 2, 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.), (E) generalize movement to other bodies with different geometries.

This report introduces two small applications for learning movement of type (C),(D) and (E). 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 . *Constraint space* describes any environmental conditions that affect the movement. 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 grasping 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 pose of the arm as shown in Figure 5. 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 grasping 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 pose to the end pose 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 report a reinforcement learning algorithm which enables us to transfer the learned human

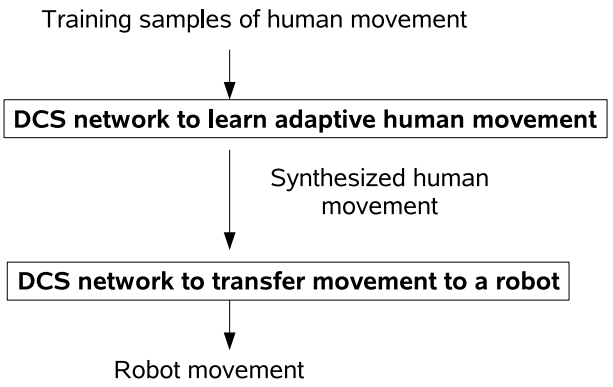

Figure 1: Architecture for learning movement and transferring it to a robot.

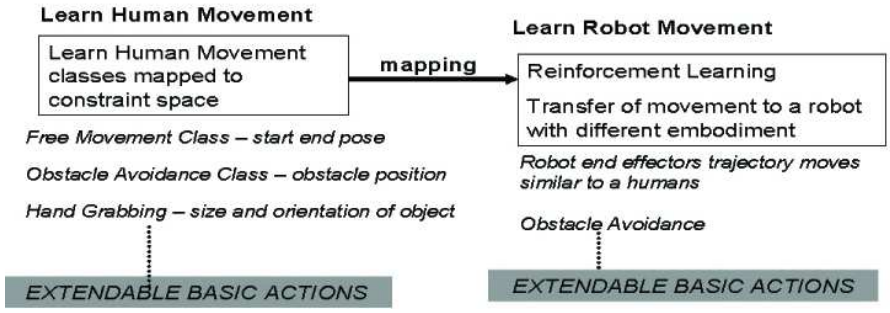

Figure 2: Learning model of movement showing the two main parts: Human and Robot movement and the movement classes under each.

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. The transfer of movement from human to robot will be show in the obstacle avoidance case.

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 Figure 1. The first network reconstructs human movement and the second transforms this movement to the robot.

This report will be basically split into the two parts mentioned in Figure 1 and illustrated in more detail in Figure 2. The first part will describe in detail the model to learn human movement applied on three cases: Free movement, Obstacle avoidance and grasping. The second part will describe the reinforcement learning applied to obstacle avoidance.

2 Learning Human Movement

To learn a movement it is generally useful to observe the following fact: Even though the human body can be described by a great many degrees of freedom and the structure of the body is very complicated, the movement manifold can be reduced to a low dimensional space. It seems that we control our bodies through a few synergies that we can effectively combine to generate any movement we like. In other words, our body poses influence each other a great deal both in time and in space. This high covariance within poses of a movement is a feature we can exploit. This leads us to use the learning model we describe in this section. We describe next some related literature and where does our work fit in it.

2.1 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.

2.2 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.

2.3 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.

2.3.1 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.
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 $\mathbf{v}_k = [\mathbf{p}_{1,k}, \mathbf{p}_{2,k} \dots \mathbf{p}_{n,k}]$ 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 Figure 3. 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 instances $\mathbf{W}_1, \mathbf{W}_2$ 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 Figure 3.

2. Scaling: We can scale the angles between path points. This assumption is made by the observation that movements are scalable. For example, when we draw a

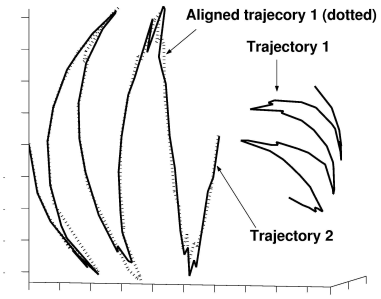

Figure 3: Alignment of two trajectories by scale and rotation. The trajectories are a sequence of direction vectors tracing curves on a unit sphere.

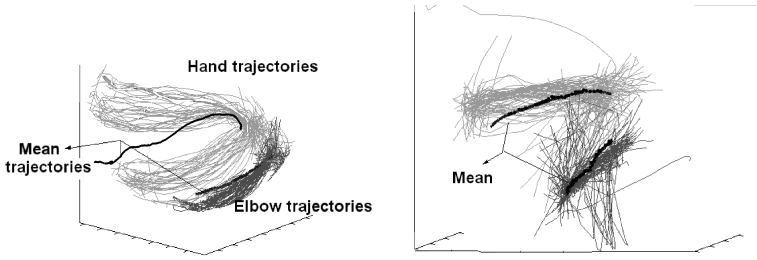

Figure 4: Example of aligning a training set of trajectories represented as direction vectors tracing curves on a unit sphere. Left before alignment and right after. We see how the hand trajectories cluster together and the mean becomes smoother.

letter smaller and then bigger, we basically move in the same way but with larger angles as shown in Figure 3. 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 Figure 3.

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 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 Figure 4.

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 $X = [\mathbf{S}_1^T \ldots \mathbf{S}_k^T \ldots \mathbf{S}_p^T]^T$. Prin-

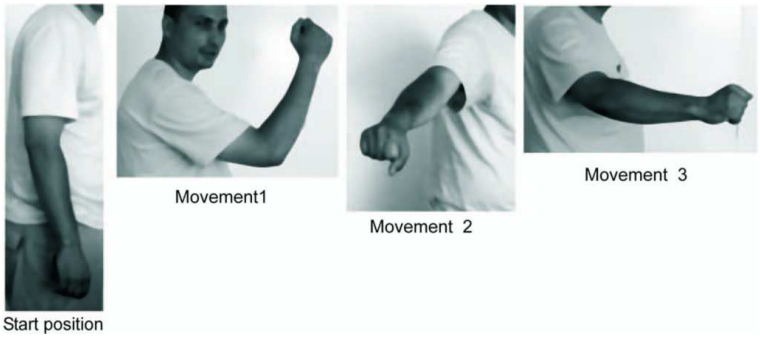

Figure 5: Three movements of the arm that all begin with the same start position (left image), the rest are end positions.

cipal component analysis is applied on X yielding latent vectors $\mathbf{\Psi} = [\psi_1, \psi_2, \ldots, \psi_n]$. Only the first q components are used where q is chosen such that the components cover a large percentage of the data $\mathbf{\Psi}_q = [\psi_1, \psi_2, \ldots, \psi_q]$. 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 Figure 6. 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.

2.4 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 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.

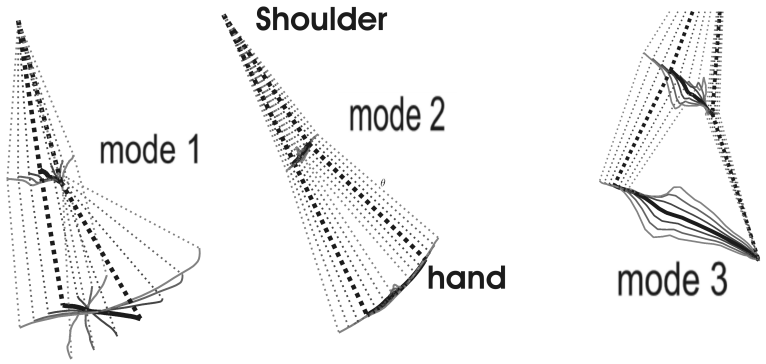

Figure 6: The first three variation modes of a kinematic chain representing the shoulder, elbow and hand constructed in 3D space. The middle thick line is the mean trajectory and the others represent $\pm 1, \pm 2, \pm 3$ standard deviations along each eigenvector.

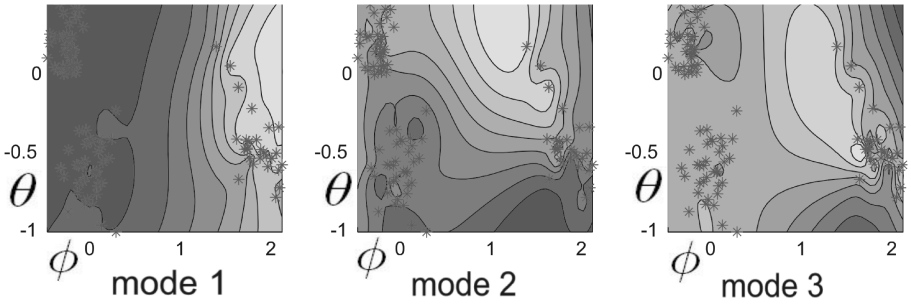

Figure 7: Distribution of eigenvalues (bright regions represent maxima) in the angular space of the end position of the hand.

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

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

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.

2.5 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 Figure 5.

The first three eigenvalues have a smooth nonlinear almost unimodal distribution with respect to the start-end space as shown in Figure 7. 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 a sample size of 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 Figure 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 obstacles. In this case we need to only take the first eigenvector from PCA to capture the variation of trajectories due to

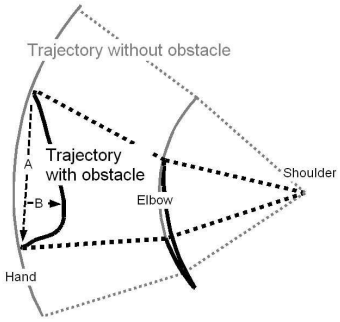

Figure 8: Trajectory for obstacle avoidance in 3D space.

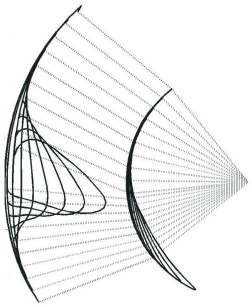

Figure 9: Variation of arm trajectory with respect to the obstacle.

obstacle position. This deformation mode is shown in Figure 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 grasping. In this case 9 markers were placed on the hand to track the index and thumb fingers using a monocular camera as in Figure 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 Figure 11 and 12. The tracker recorded 350 grasping 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 Figure 10. Figure 11 and 12 depict an example comparison between grasping movement generated by the DCS and an actual sample. Below we used two measures that characterize well grasping: 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. Training the DCS takes only a few seconds for all the movement classes presented.

After presenting the experiments for movement generation, we demonstrate some results for movement transfer. Figure 13 shows a 2D human trajectory that a robot has to learn. The end markers (shown as $(*)$) in the figure 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 was able to learn the nearest trajectory after about 2000 iterations. 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

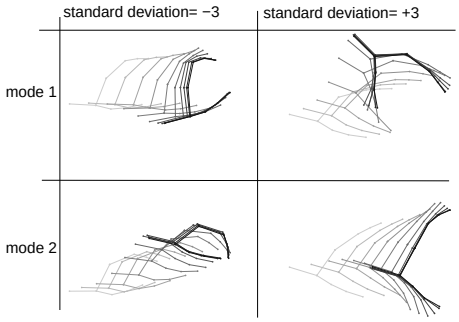

Figure 10: The first two variation modes of grasping.

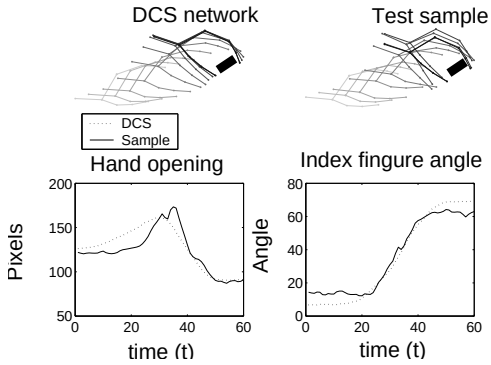

Figure 11: Comparison between DCS and and a grasping movement for a 4 cm object at $60°$ with respect to the hand.

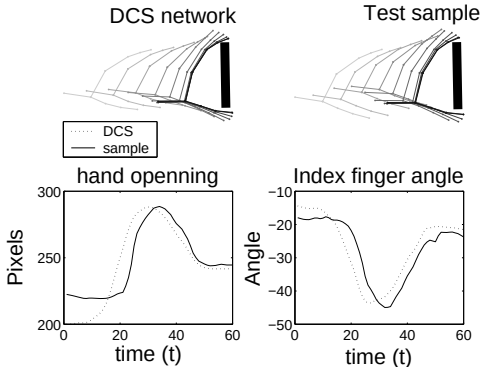

Figure 12: Comparison between DCS and and a grasping movement for a 12 cm object at $0°$.

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.

3 Transferring of Movement from human to robot

3.1 Introduction

Based of the network shown in Figure 1, we divided the problem of learning human movement into two parts: Learning the human movement adapted to constraints and then Learning the mapping between the human movement and a robot. We will propose a reinforcement learning algorithm to transfer the movement from human to robot that is flexible, efficient and generalizes over the constraint space. We will discuss next both parts briefly and how they will work together:

- Learning human movement: In this case we generalize the movement over the space of environmental constraints. These constraints were defined on a case by

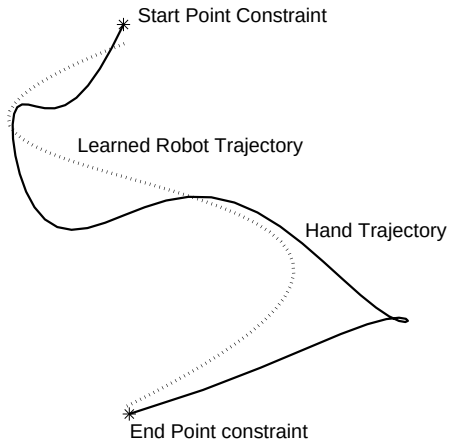

Figure 13: Learned trajectory using start and end position constraints in 2D space

case basis as was shown in free movement, obstacle avoidance and hand grasping. The output of the trained network in this stage is generalizable human body movement. The limitation with that is that is that the proposed approach is unable to transfer the learned movement to a machine with a different geometry than the human body. For example, if the human arm was modeled with 30 degrees of freedom, then how we can train a machine with 6 degrees of freedom and different segment lengths than the arm to do an equivalent action. Equivalence of actions will be explained in the next point.

- Leaning movement mapping between human and robot: In order to transfer a learned human movement, we have to train the robot to imitate it. This means the robot should be able to apply the movement successfully satisfying the constraints which the human movement did. In addition to constraint satisfaction, we can make the robot move in a way that looks geometrically similar to the human. Geometric similarity here is a problem specific term which differs from one case to another. But basically we can define it as enabling the robot to use human movement as a prior to guide its movements to converge faster to a solution. This will be elaborated in more detail later. A nice appealing effect of using geometric similarity is that movements under different embodiments look to some extent similar to each other. In other words, action equivalence of different embodiments is the set of all possible movements of the second embodiment (Robot) that satisfy the constraints that the first embodiment (Human) did. From this set we choose that movement that makes the second embodiment look (subjectively) similar to the movement of the first embodiment.

Movement similarity of different embodiments can be found in some research papers. The solutions found in literature basically fall into two categories: The first class looks only at the effects of actions on the environment as in [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 explicit function

which measures the similarity between the robot and human poses as in [6]. We propose to solves this problem by using a reinforcement learning rather than explicitly defining the mapping function. The reward function in reinforcement learning enables us to easily model both those aspects of movement similarity. The approach we propose keeps the learning algorithm of the robot general and free from the any problem specific solutions which is delegated to the fitness function. The reinforcement learning approach we will propose is general, efficient and flexible because it can be applied to many movement cases.

A reinforcement learning has also been used by Schaal [3] in his work with humanoid robots. When comparing our approach with his, we can see that our approach can deal with more dissimilar embodiments than humanoid robots. Another aspect is that his method learns the mapping of a single movement which can be varied adapted in a very limited way by changing the weights of his movement pattern generators DMPs. Our approach learns not only a single movement but generalizes this mapped movement over the constraint space.

3.2 Method

After reviewing some literature and comparing it with the proposed method, we discuss next our reinforcement learning approach to mapping movement. Imitation of movement is a basic social mechanism in humans. The child can observe the movements of adults through his eyes and tries through his observation to repeat the same movement. Through many trails and errors he begins to correct and refine his own movement by watching himself until it becomes similar to what he observed. He then uses this ability to do useful manipulation of his environment as discussed in [11]. In our learning approach we try to use the perception-action cycle in a similar way. In the case of the robots perception, observed movements of the human or movements made by the robot are tracked trajectories on an image(s). The image(s) of these movements is taken by a static camera(s) viewing the human or robot. This camera(s) is the robots eye(s) so to speak. We assume that this camera(s) is un-calibrated. The movements performed by the robot are done by activating its motors. There is no explicit mapping between a point on the image(s) of the un-calibrated cameras and points in the real world in which the robot moves in. The robot can only learn by a series of trials and errors and observing its own movement and comparing it to the humans in image space. This is why we need reinforcement learning. This approach differs from servoing which learns the map between image space points and real world coordinates in that we try to learn joint-trajectories and not just individual points. The imitation perception-action cycle of the robot works as follows: The robot observes the human movement on the image either tracked directly or generated by the first network as shown in Figure 1. The robot then tries random movements until its observed movement in the image becomes more and more similar to the observed human movement in the same image and also until all the environmental constraints are satisfied.

In order to simplify the movement transfer problem we restrict the problem to transferring movement to the end effector of the robot. The restriction is done because it

proves the concept and because the gripper which is equivalent to the human hand is the most important point in a manipulator system. There is no reason that this restriction can be removed in the future when using the methods that will be described here to more degrees of freedom in the robot space provided the conditions are suitable.

The first network in Figure 1 generates trajectories of each joint satisfying the constraints. To be consistent with this output, we designed the second network learn from generated trajectory samples also.

3.2.1 Dimensionality Reduction

In order to accomplish trajectory learning, we have to be careful with respect to dimensionality of the action space. If we have many degrees of freedom and too many trajectories to choose from then no reinforcement learning algorithm can converge in a reasonable time. What will happen is that all the random trials will fall under the curse of dimensionality. This is why we have to reduce the dimensionality of action space. This is done in two ways:

- Converting robot trajectory to a frequency representation. When we have in such a representation we can throw away the higher frequency coefficients and try to optimize only the low frequency vector. A suitable trajectory representation is using discrete cosine transform. This can reduce the robots trajectory from say 100 intermediate points to 4-5 low frequency coefficients for example. When trying new trajectories of the robot we can modify an existing coefficient vector and then converting the modified vector to a trajectory in the robot's workspace. The discrete cosine transform is expressed as

$$\mathbf{a}_k = w_k \sum_{n=1}^{N} \mathbf{p}_n cos(\frac{\pi(2n-1)(k-1)}{2N}), k = 1, \ldots, N, \quad (6)$$

$$w_k = \begin{cases} \frac{1}{\sqrt{N}}, & k=1 \\ \sqrt{\frac{2}{N}}, & 2 \leq k \leq N \end{cases} \quad (7)$$

where a trajectory is sampled at N points in 2D or 3D space $\mathbf{p}_n, n = 1 \ldots N$. 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: $(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m, 0, \ldots, 0)$ as follows

$$\mathbf{p}_n = \sum_{k=1}^{N} w_k \mathbf{a}_k cos(\frac{\pi(2n-1)(k-1)}{2N}), n = 1, \ldots, N, \quad (8)$$

$$w_k = \begin{cases} \frac{1}{\sqrt{N}}, & k=1 \\ \sqrt{\frac{2}{N}}, & 2 \leq k \leq N \end{cases} \quad (9)$$

- A robotic arm is basically a manipulator system that transports its end effector from one point to another in order to employ the gripper to grab and release the objects. This has an analogy to the hand in the human arm. By restricting the trajectories to that of the human hand and the end effector of the robot we reduce the problem of modeling many degrees of freedom to just one. This restriction made here does not mean that it is not possible to extend this method to more degrees of freedom. As long as there is a suitable low dimensional representation of the robot's movements, we can successfully apply reinforcement learning methods on it.

- In the framework of COSPAL project the robot workspace was restricted to 2.5D. This means that the robot gripper manipulates block objects on a 2D flat surface. The robot gripper moves on two planes parallel to the work surface: The first is close to the objects in order to grasp them and the second is above the first in which objects are being transported. This simplification can be made also in our case. We can restrict the trajectory of the robots end effector to a 2D plane above the work surface thus reducing the trajectory of the end effector from 3D to 2D. This restriction also enables the use of a monocular camera rather than a stereo pair.

3.2.2 Obstacle Avoidance

As a case study to illustrate the movement transfer concept we chose obstacle avoidance out of the three movement cases shown by the first network. This demonstrates nicely the transfer concepts involved. In this case, the human being avoids hitting an obstacle he observes by his hand. In order to do that he varies his movement to reach from a certain start position to some end position around the obstacle. The start and end position and the obstacle position and size constitute the constraints around which this movement is involved. The robot can learn this movement by assuming it avoids similar obstacles placed on the workspace. The camera overlooking the workspace in Figure 16 is used to track the robot trajectory using a colored marker placed on the end-effector shown in Figure 17. The end-effector trajectory that the robot tries, projects a trajectory on the camera image. This projected trajectory can then be compared to projected trajectories generated by a human hand tracked on the same camera or synthesized on the camera image from the network that was trained from the human hand. The position and size of the obstacle has to also be extracted from the camera image. All these information can be used to define a fitness function (or a reward function in reinforcement learning) that we try to optimize. The fitness function will take two factors into consideration when measuring the similarity between the human and the robot trajectory:

1. How similar are the trajectories in the image space. In this case a suitable measure between trajectory curves can be used such as the Hausdorf distance. This helps the robot to converge more quickly to satisfy the constraints using the human trajectory as a prior.

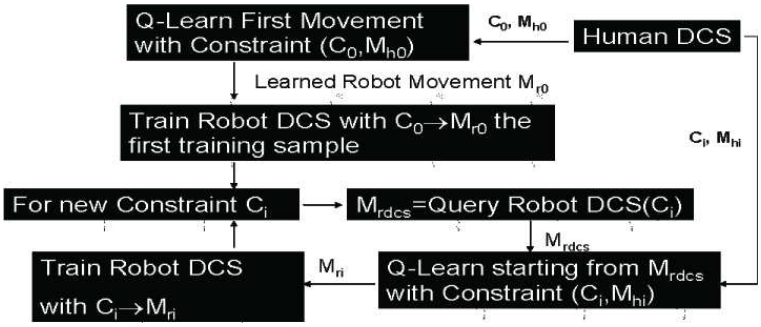

Figure 14: The online learning cycle between the robot DCS and Q-learning

2. Constraint satisfaction which in the case of obstacle avoidance means that the robot start and end points are close to the humans and that the robot trajectory did not hit the obstacle. The obstacle is roughly approximated in the camera image as circle with a radius r and center (x_0, y_0). This approximation of obstacles does not take into account the object geometry and variations due to viewpoint. But it is the simplest representation of an obstacle captured in the image as shown in figure 18.

The fitness function cab be expressed by the following formula:

$$r(\mathbf{u}, \mathbf{v}, C) = \alpha_1 f_1(\mathbf{u}, \mathbf{v}) + \alpha_2 f_2(\mathbf{v}, C) \tag{10}$$

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 weights, f_1 measures the similarity between the shapes of trajectories and f_2 measures constraint satisfaction in the robot space. An Example is shown in Figure 13. Here we see both start and end point constraints and how the robot tries to reach both and be similar to the human trajectory at the same time.

3.2.3 The learning algorithm

So far we have explained the idea behind movement transfer and how it relates to the first network. We also explained how reduce the dimension of the action space and that it is necessary for reinforcement learning to work. We explained also that matching between trajectories is done through fitness or a reward function that takes into account both the human trajectory and the constraints in image space. Next, we will explain how the learning itself takes place. The learning algorithm actually goes through several steps as shown in Figure 14:

1. In the first step the Human network generates an image of the imitated movement M_{h0} with constraints C_0 in the image space of the robot as shown in Figure 14. The robot stats from a null trajectory which it then modifies at random to generate increasingly bigger trajectories. The trajectories are then captured using the tracker in image space and compared with the human trajectory generated from the top

network in Figure 1. In this case a modified type of Q-learning is used to search the action space of the robot through the fitness or reward function until a suitable trajectory M_{r0} that comes close to satisfying the similarity to the human and the constraint space is achieved. This step is the most time consuming because it requires many trials to reach from the null trajectory to the point where the robot learned the first movement.

2. Once the first mapped robot movement M_{r0} is learned for the first obstacle constraint C_0. This is added as a training sample to an online DCS network for the robot as shown in Figure 14. Now the human movement network generates a new trajectory M_{hi} for a new obstacle C_i which is close to the first human trajectory example. Now we apply the Q-learning again to find a suitable robot trajectory M_{ri}. We do not begin with the null trajectory this time. Instead we query the DCS network of the robot for the closest initial trajectory which we can begin with. This means that since we start with a sufficiently close solution for the robot trajectory we will need less number of trials to achieve the goal of satisfying the new constraints. Once a suitable robot trajectory is found, we add the new constraint and the new robot trajectory to the online DCS network of the robot. In other words, we used the last step as a bootstrap to learn the next robot movement more quickly. This principle of continual learning enables the robot to quickly generalize from the first few movements it learned and correct its wrong guesses more and more quickly using reinforcement learning through trail and error.

3. The last step is iteratively repeated for different sizes and positions of trajectories until the algorithm converges to a mapped robot trajectory generalized to the constraint space in images.

This continuous robot cycle of guessing a good initial move, trying new variations and then correcting the online knowledge is a the learning mechanism which enables the robot network to be trained from the human movement network and generalize from it. The First human network trains by providing a sequence where samples are close to each other that enables the robot neural network to learn and generalize its movement more quickly. Even if the temporal closeness from the training samples of the first network is violated, the second network will try to provide a good initial guess of the movement by interpolating between the closest movements to the constraint presented in the image. This learning approach in which one network trains another is an example of a hierarchical learning system in which learning takes place at many levels and the knowledge acquired is used to bootstrap the learning process incrementally.

The Q-learning part of the algorithm will be explained next. A good reference about Q-learning can be found in [12]. The Q learning has been modified from discrete to a continuous version by representing the Q-table as a graph. The nodes represent real valued states and the edges represent actions that connect states to other states. The edges store the q-value for the state action pair. Actions are simply random small increments to the state vector. Figure 22 shows the Q-graph with the optimal path marked with a thick line. In order to avoid loops we constrain this graph to be a tree. The algorithm resets the current state to the start state periodically in order to update the

Initialize the Q network with the inital state state $\mathbf{\Theta}_0$.
$\mathbf{\Theta} \leftarrow \mathbf{\Theta}_0$
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:
$\alpha[r + \gamma max_{\Delta\mathbf{\Theta}'} QDCS(\mathbf{C}, \mathbf{\Theta}', \Delta\mathbf{\Theta}') - QDCS(\mathbf{C}, \mathbf{\Theta}, \Delta\mathbf{\Theta})]$
$\mathbf{\Theta} \leftarrow \mathbf{\Theta}'$
Reset $\mathbf{\Theta}$ to $\mathbf{\Theta}_0$ with a probability ϵ
Until $\mathbf{\Theta}$ converges to a good solution

Figure 15: Learning optimal trajectories using Q-learning

Q-values at the edges and also to branch from existing states to new subtrees. A simplified version of the algorithm is represented in Figure 15. In this case whenever we reach a new state, we create a node in the graph. In our experiments we select the next state by doing n random actions and choosing the best one that has the maximum reward. We also modified the algorithm to update the Q-value of the current edge of the state and simultaneously all the Q-values of the ancestor states in order to avoid too much resetting of the current state.

3.2.4 Why use learning?

After describing how the learning works, One can argue that instead of using a learning scheme, a trajectory is directly generated by the robot that approximates the human trajectory using, for example, a simple spline function that interpolates between the points in work space which the human hand went through. The following is a comparison between direct mapping and learning that defend the learning approach.

- An advantage is that learning the mapping between the image space and the movement in robot space frees us from calibrating the camera and defining an appropriate camera model that can map an image coordinate to the robot position. In other words the robot learns to associate the percepts it acquires using a camera and the actions it takes. This frees us from any explicit modeling of the environment and enables us to reuse the same learning mechanism regardless of what type of camera we have.

- Another advantage lies in the flexibility in defining the fitness function. We are free to choose any fitness function that satisfies constraint or similarity to human motion depending on the problem. Using explicit mapping gives less flexibility in this regard. We can also vary the fitness function dynamically during the learning process when some information is not available. For example, if a human trajectory is not available for some constraints, the fitness function can measure only

satisfaction constraint for trajectories that were already learned. We are also able to compare different fitness functions that lead to different solutions in terms of learning rate and quality of learning.

- Learning in terms of generalization, the robot is able to generate and explore new situations in the environment from the movements it had already learned, whereas, generalization is not possible in this sense using explicit mapping because every movement of the robot has to be transformed directly from an equivalent human movement. This means that that learning is adaptable and transferable where explicit mapping is not.

- In terms of efficiency, the explicit mapping requires little computational cost. But the reinforcement learning is also very efficient because it uses a low dimensional state space representation which represents trajectories as coefficients. This enables the random exploration to converge quickly to the desired solution. Using a learned trajectory to adapt to a new constraint also accelerates learning. These features enable it to be used for online learning.

- In terms of extendability to more degrees of freedom. An explicit mapping is able to model only 1:1 mappings from a human joint to a robot joint. The reinforcement learning method that uses an arbitrary fitness function is able to incorporate more degrees of freedom for this purpose that don't use an explicit mapping.

3.3 Experiments

After explaining in the previous sections how reinforcement learning is used to transfer movement. We show now a demonstration of these concepts on our movement case: Namely, how an industrial robot learns from observing a human how to avoid an obstacle placed on its workspace. The experimental setup will be shown and then the results on both real and simulated environments will give a good idea how quickly did learning convergence occur.

Trajectory mapping for obstacle avoidance was tested on both simulated and real environments. The main advantage of using a simulated environment is that it enables us to test how well the system learns without waiting long hours till the robot completes all its movements. In addition to that, Simulations enable us to teach quickly the robot and then use the trained network directly in the workspace. The experiments measure both the time it takes to learn the first obstacle and then how well the system learns subsequent moves for different obstacle sizes and start and end positions. We will discuss next the experimental setup.

An industrial robot RX90 was used to learn the movements. There is a camera overlooking the workspace of the robot from the side and slightly above the workspace as shown in Figure 16. A view from the camera can be seen in Figure 17. In this figure we can see that above the gripper of the robot there is a piece of paper with a colored circle in it. This circle can be easily segmented in the image by thresholding the hue and the centroid of the circle can be easily calculated from the segmented image. The centroid

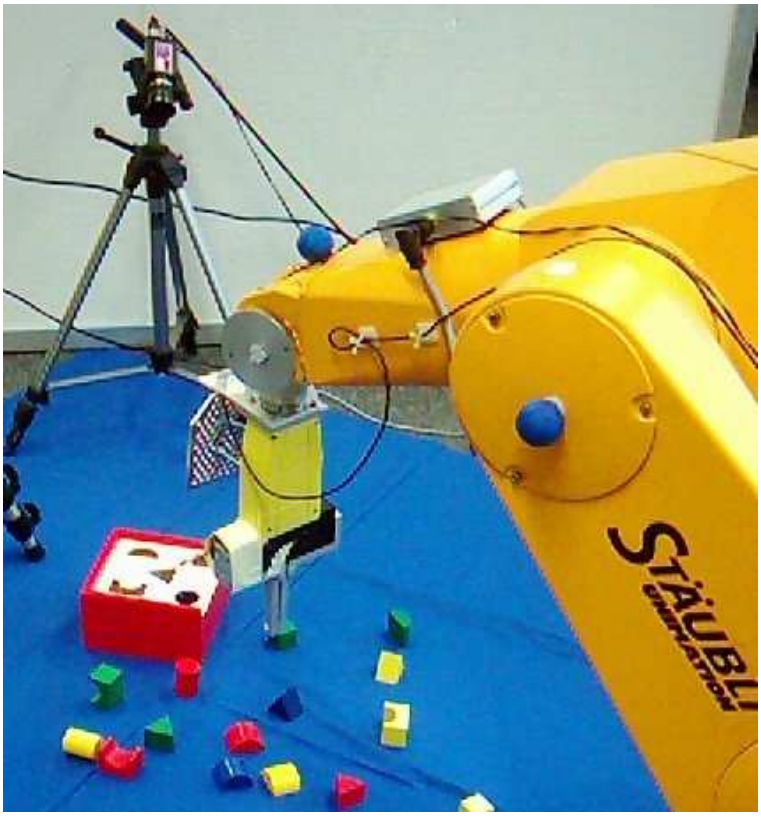

Figure 16: Setup showing the robot and the static camera

coordinates mark the position of the gripper in image space. A robots trajectory is captured by moving the gripper on a plane parallel to the work space of the robot. During this movement the position is measured in the image at regular intervals. These image coordinates make up a way point representing the projection of the trajectory in image space. For these experiments we used 20 way points to represent each trajectory. This is how the robot trajectories are recorded. The human trajectories are recorded by moving the human hand with a marker placed on it approximately in the same plane of the grippers marker. The human trajectory is tracked on the image and re-sampled to the same number of way points as the robot so that they are equidistant in time. In addition to measuring the human trajectory, we have to have an estimate of the obstacle size and position in the image. This is done by manually marking the obstacle in the image using an a circle approximation as shown in Figure 18. This enables a simple representation of the size and position of the obstacle without having to deal with geometric details of the object. This setup was also simulated by taking the image coordinates of the four corner points of the robots workspace also shown on the same figure. Between these points interpolation was used to approximately map any point in the image to the world and vice versa.

As mentioned in Figure 15, in the first step the robot has to learn the first example observed from the human shown in Figure 18. In this case the real experiments show

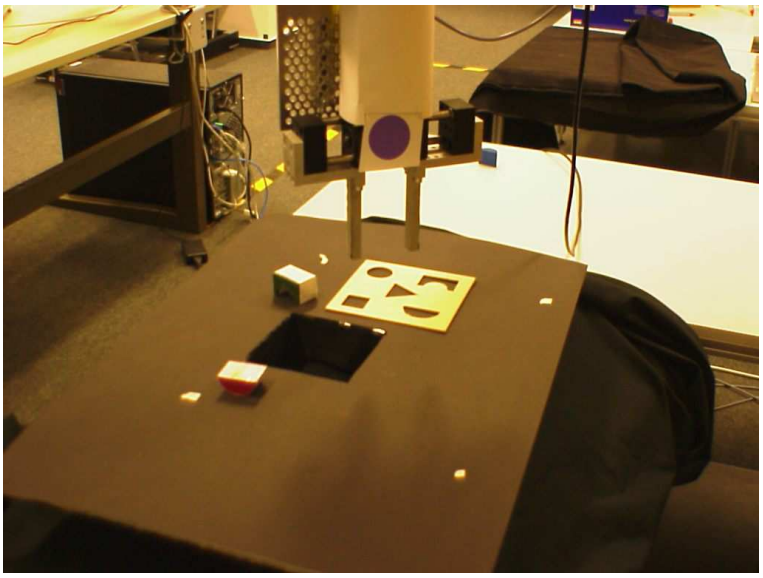

Figure 17: A view from the camera overlooking the workspace

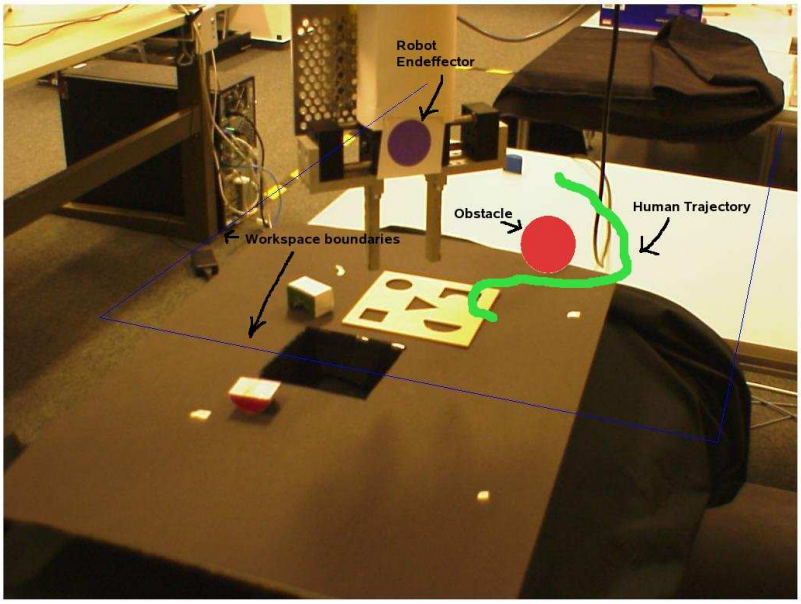

Figure 18: An obstacle avoidance trajectory and an obstacle represented as a circle in the image

that that the robot needs about 370 trajectories in the best case to learn this movement and converge to the solution shown in Figure 21. The progression through some intermediate states is shown in Figure 19. The total time taken by the robot to learn this move is 2.36 hours. This make an average of 23 seconds per trajectory. In this case the robot trajectory was represented using 10 DCS coefficients: 5 for the x-axis and 5 for the y-axis. The Q-learning algorithm generates new states by adding a small random vector δ to an existing state θ: $\theta' = \theta + \delta$ and then the new trajectory is tried out and the reward function measures the similarity to the human curve. In simulated experiments the robot may take from 300 to 1500 attempts to learn the first movement depending if it has luck and stumbles on a solution quickly through its random trails.

The q-learning trace of states is shown in Figure 22. The nodes in this graph represent DCT coefficient projected onto a plane by taking the first two DCT coefficients of a trajectory. The start state is the null movement and is represented by a square. The edges (directed) represent random actions that changed one trajectory state to the other. The diamonds represent states that branched to more than one state during the search. The final states are represented by circles and the best path leading from the start state to the solution state is highlighted with a thicker line. It takes 100 to 200 stored states to learn a single target trajectory. The Q-learning algorithm uses these stored states to make semi-random walks in the action space and also to retry some old states and branch out from them which enables it to escape local minima.

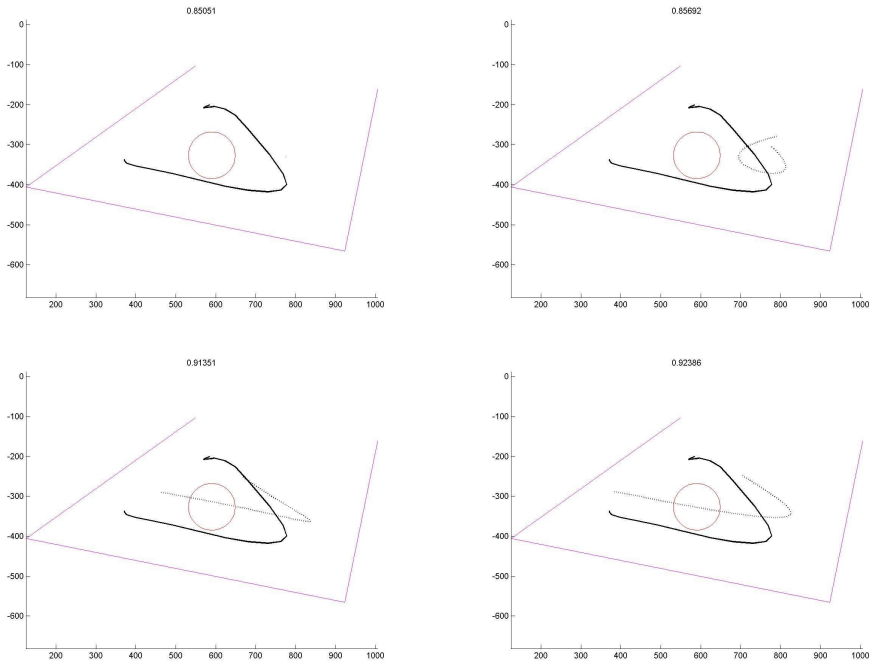

Figure 19: Part A: Progression of trials and errors to learn the first movement. The number above the trajectories is the reward function. The dotted line is the robot trajectory and the solid line is the human trajectory. See the rest in Part B.

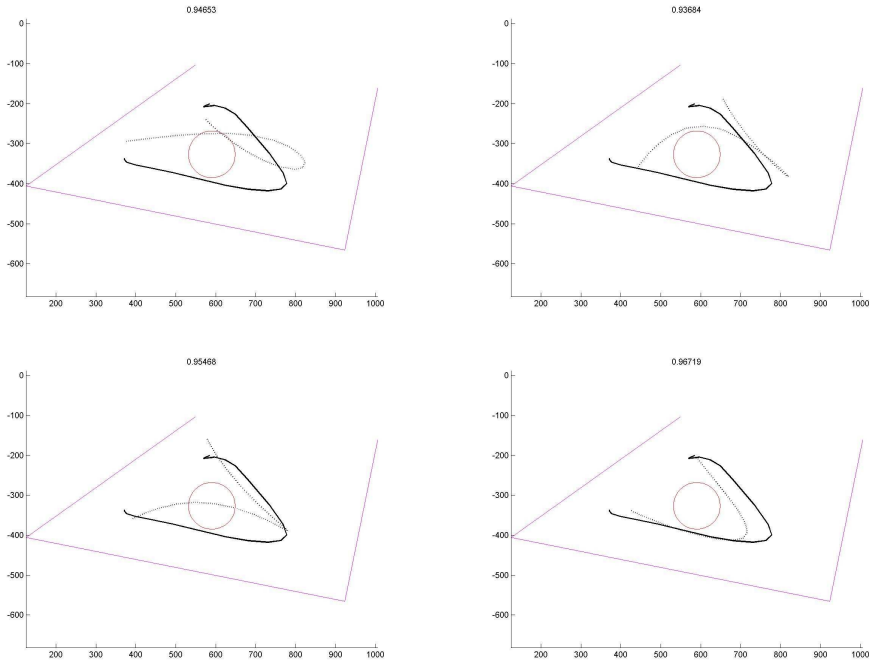

Figure 20: Part B: Progression of trials and errors to learn the first movement.

After the first step is archived, the training sample is added to the on line DCS network of the robot.

The subsequent learning steps use the learned samples to guide them a quicker convergence.Instead of starting from a null-trajectory as in the first time, the robot will try to make a good guess and begin from there. In this case closely similar trajectory samples were generated by the first network in Figure 1 for different obstacle sizes and end point positions as shown in Figure 23. The training of the robot trajectory required between $10 - 30$ attempts with an average of 21.4 to adapt to each new constraint. An example is shown in Figure 25 where the dcs guesses the closest movement (top left) that satisfies the constraints.This is a significant speed up over the $300 - 1500$ trials that was required to learn the first move. This shows how continual learning can speed up the learning process. The online DCS network of the robot required $15 - 20$ neurons to learn the obstacle variations in Figure 23. This figure shows how the learned robot network generates close and comparable trajectories to the human.

In conclusion, these experiments demonstrated how to use continual online learning and a teacher DCS network can speed up the learning of robot movement significantly by a factor of about 50. The learn process can also be simulated and the trained robot network can be used directly on the real robot.

Although this learning algorithm was tested only on the end effector trajectory, it is possible to extend this approach to more degrees of freedom. The simplest way to start is to add when the robots gripper can open and close for example when doing the grasping movement. This becomes simply a simple additional constraint to try.

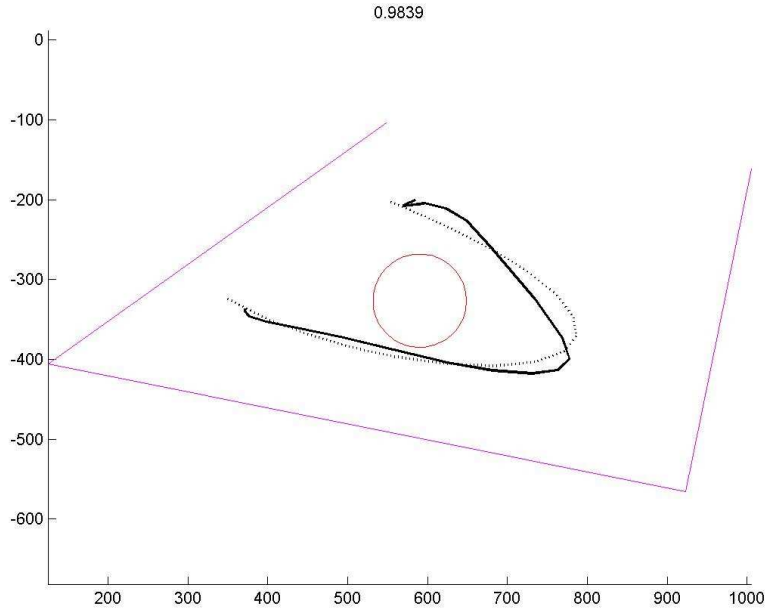

Figure 21: The First Learned movement with reward value on top.

4 Conclusion

We proposed a learning model for generation of realistic articulated human 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 constraint space with a few samples. This trajectory based method is more suited for movement generation than pose based methods which are concerned with defining good priors for good fitting with image data in applications such as tracking. The proposed method models variation of movement with respect to constraints in a more direct 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 learning. The method uses DCS networks and Q-learning to exploit and explore 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

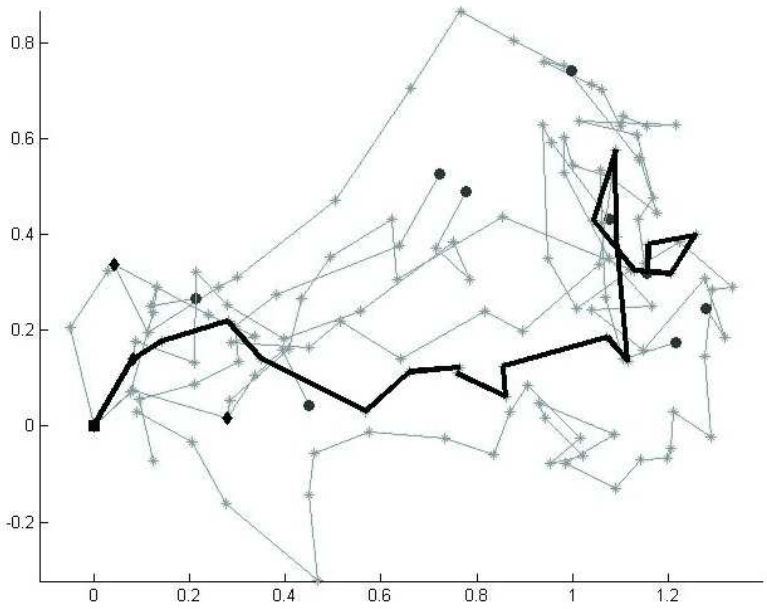

Figure 22: Projected robot-trajectory states stored during the search using Q learning. The start state is the labeled with a square and the thick path represents the walk to the solution. The number of states is 142.

is not responsible for any use which may be made off 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)

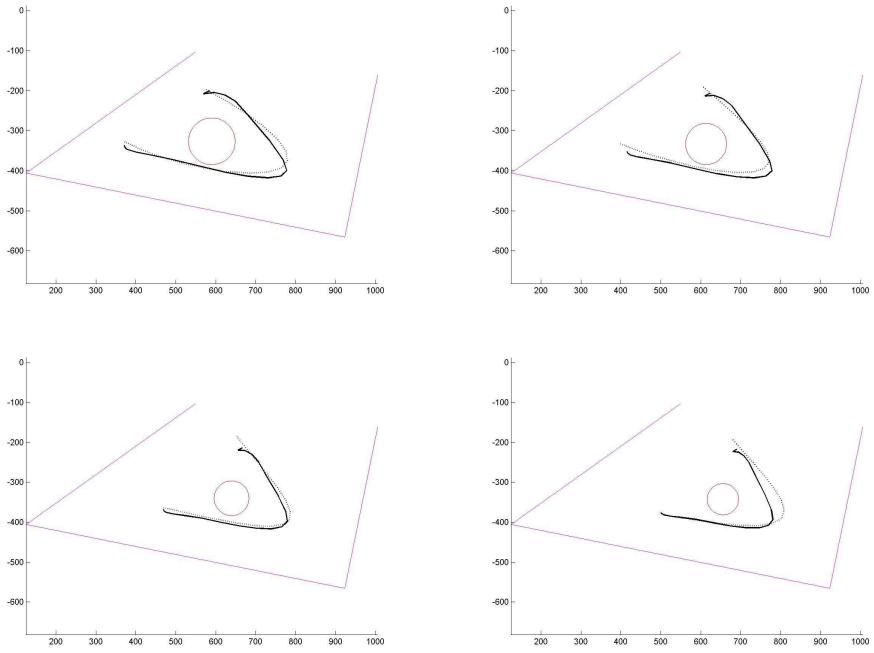

Figure 23: Part A: Comparison between the learned robot trajectory (dotted) and the human trajectory (solid) with different obstacle size and position. See the rest in Part B.

[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] Breazeal, C., Buchsbaum, D., Gray, J., Gatenby, D., Blumberg, B.: Learning from and about others: Towards using imitation to bootstrap the social understanding of others by robots. Artif. Life **11**(1-2) (2005) 31–62

[12] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)

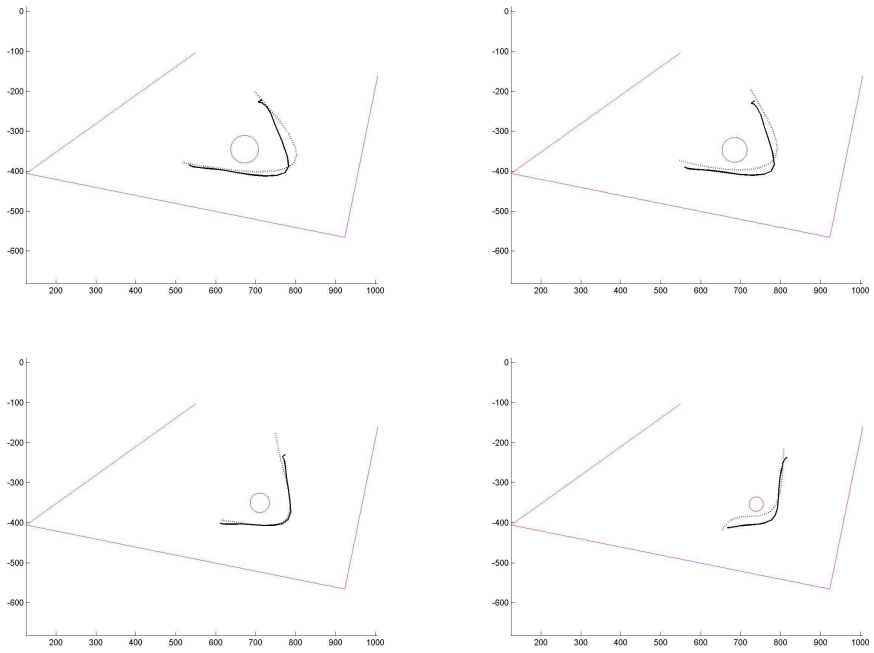

Figure 24: Part B: Comparison between the learned robot trajectory (dotted) and the human trajectory (solid) with different obstacle size and position.

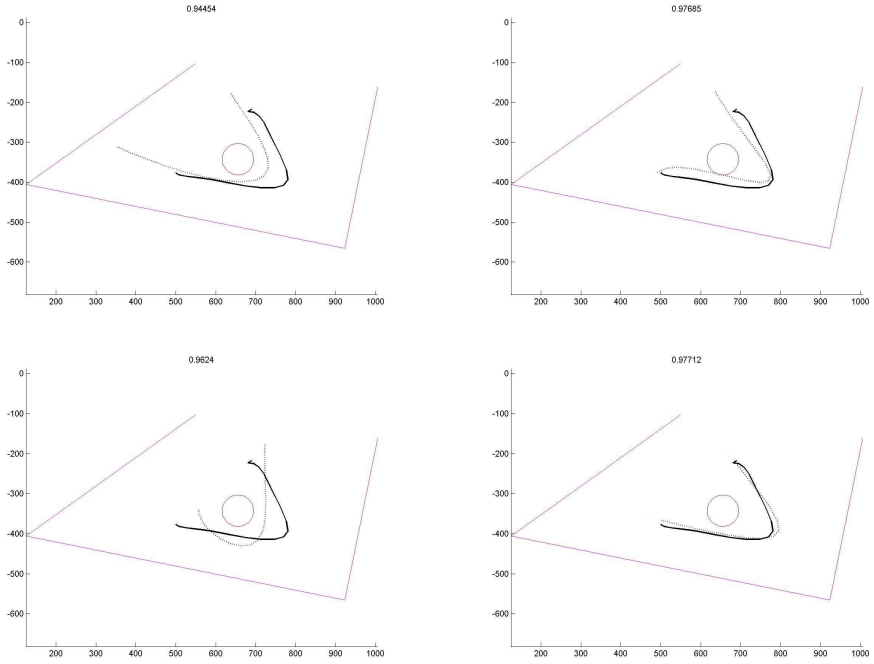

Figure 25: Learning a subsequent move starting from a known robot trajectory (dotted) and the human trajectory (solid) with different obstacle size and position.