Reaching through Learned Forward Model

GANGHUA SUN & BRIAN SCASSELLATI

Computer Science Department, Yale University
New Haven, Connecticut 06511, United States

{*ganghua.sun, brian.scassellati*}*@yale.edu*

Abstract. This paper presents a learning approach for a humanoid robot to reach for objects in its environment. Instead of assuming that the exact forward kinematics of the arm are given, we address the reaching problem by first learning the forward kinematics with a radial basis function network (RBFN) through autonomously gathered training samples. The learned forward model is subsequently used to construct Jacobian matrices to incrementally generate straight reaching trajectory exhibited by humans. We show that if the learning parameters are set appropriately, a RBFN trained on a small number of samples corrupted by perception noise can still lead to high reaching accuracy. The size of the training set can be further reduced without severe performance degradation if limited visual feedback is used to aid reaching after the end effector has been moved into the neighborhood of the desired object.

Keywords: Reaching; Forward kinematics; Inverse kinematics.

1. Introduction

Most robotic literature dealing with the reaching problem focuses almost exclusively on the aspect of inverse kinematics. A large number of solutions to inverse kinematics are based on the Resolved Motion Rate Control (RMRC) scheme proposed by Whitney and its extension proposed by Liegeois, both of which require the exact forward kinematics for the computation of local Jacobian matrices.[1,2] Such a requirement does not present a problem for high-precision robotic assembly systems, but for a research platform, or for most consumer goods, this requirement is unrealistic due to wear and tear on typical systems.

Additionally, for a biological system, such as human, the parameters of the arm are not readily available and change gradually over the lifespan of the individual. Human infants typically start goal-directed reaches around 4 to 5 months of age. Adult-level reaching proficiency cannot be achieved until they are 3 years old.[3] Suggested models of how to learn reaching without prior knowledge of the arm parameters can be grouped into two categories, both of which require a series of random arm movements to be performed beforehand to build up a training set. Models in the first category attempt to learn inverse kinematics without building a forward model.[4,5] After the learning, a direct mapping either from task space vector x to joint vector θ or from $\dot{x}$ to $\dot{\theta}$ is constructed. Such an approach is usually complicated by the fact that inverse kinematics is a one-to-many mapping.[6] Special caution must be taken to ensure the accuracy of the learned inverse model.

Models in the second category use the training set to build a forward model and then use this forward model to solve the inverse kinematics problem.[7,8] Such an approach is inspired by adaptive control theory and usually requires the identification of the underlying system before control can be performed.[9]

We have adopted this indirect learning approach to study reaching arm move-

ments in a humanoid robot. First a radial basis function network (RBFN) is used to learn forward kinematics of the arm from a training set consisting of autonomously gathered samples. After the RBFN is trained, local Jacobian matrices are generated through differentiation of the radial basis functions in the hidden layer. In this way, the existing efficient and flexible solutions to inverse kinematics based on RMRC and its extensions can be exploited to generate reaching trajectories.

Short learning time and high reaching accuracy despite perceptual noise are the two most important factors that determine the applicability of our approach on a physical robotic platform. Two contributions of this paper are a detailed description of the parameter tuning process for the RBFN training and a characterization of the influence of perceptual noise on the reaching accuracy. We show that with optimized parameters, high reaching accuracy can be achieved with a small training set which requires only a short time to construct, even if only proprioceptive feedback is available during the reaching movement. The size of the training set can be further reduced if we allow a one-time visual feedback to aid reaching after the end effector has been moved into the neighborhood of the desired object.

This paper is organized as follows. Section 2 presents the hardware and software platform of our humanoid robot and the system architecture for learning reaching. Section 3 characterizes our approach to forward kinematics learning and describes our solution to inverse kinematics. Section 4 presents the parameter tuning process for forward kinematics learning and simulation results for reaching accuracy. Section 5 presents the experimental results for an implementation of this strategy on a humanoid robot. The paper is concluded with a discussion.

2. Experiment Setting

2.1. *Hardware and software platform*

Nico is an upper-torso humanoid robot modelled after the body dimensions of a one-year-old human child. The mechanical structure has a seven-DOF head, a six-DOF arm and and a one-DOF waist. The other arm and an additional two DOFs in the waist are under construction. Four of the seven DOFs in the head are neck joints for the control of head orientation, which can be sensed by a 3-axis gyroscope mounted on the top. The vision system of Nico consists of four miniature CCD video-cameras divided into two sets, one for each eye. In each set, there is one long focal length camera for fovial vision and one short focal length camera for peripheral vision. The six-DOF arm with a total length of about 300mm from shoulder to wrist has a motion range similar to a one-year-old. Currently, a 60mm steel shaft is attached to the wrist plate with a ϕ19.05mm wooden ball at the tip as the end-effector. Each joint of Nico is driven independently by a DC motor with an integrated high-resolution optical encoder. All motors and sensors on Nico are connected via extension cables and respective control units to a computer rack of sixteen nodes running the QNX real-time operating system. Nodes are connected through a 100Mbit backbone switch and a number of direct point-to-point network

links.

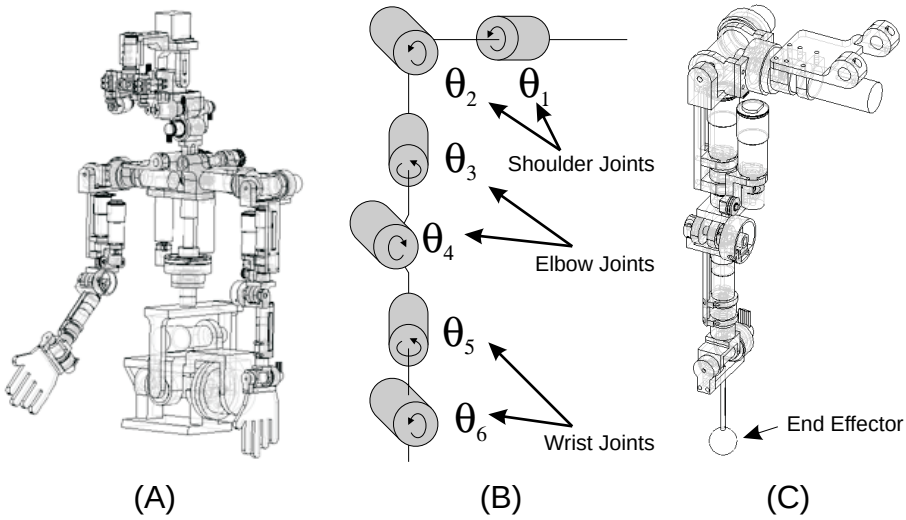

Fig. 1. (A) Mechanical design for Nico, an upper-torso humanoid designed to match the size of a one-year-old child. (B) The kinematic structure of the robot's arm with six degrees of freedom, including two for the shoulder, two for the elbow and two for the wrist. (C) A detailed view of the arm structure. A steel shaft with a wooden ball as finger tip is currently attached to the wrist plate for easy detection of the end-effector.

A set of modular software components have been implemented on the robot, ranging from low-level device drivers to selected higher-level cognitive functions. During run-time, selected modules are instantiated on the same or different nodes depending on their computation requirements. Active modules can selectively communicate with one another through a common communication interface. Whether a data exchange takes place on the same node or across the network is totally transparent to an individual module.

2.2. *System architecture for reaching*

The overall system architecture for learning to reach is shown in Fig. 2. A stereo vision subsystem provides the target position, a training module constructs the forward model and a reaching execution module generates the reaching trajectory. The modules with dashed boundaries along with their respective data flows are instantiated only during a training session. The training module and the reaching execution module will be described in detail in the next two sections, so only the stereo vision subsystem is treated here.

The stereo vision subsystem retrieves video data from the two short focal length cameras as input. The two long focal length cameras prove to be impractical for the

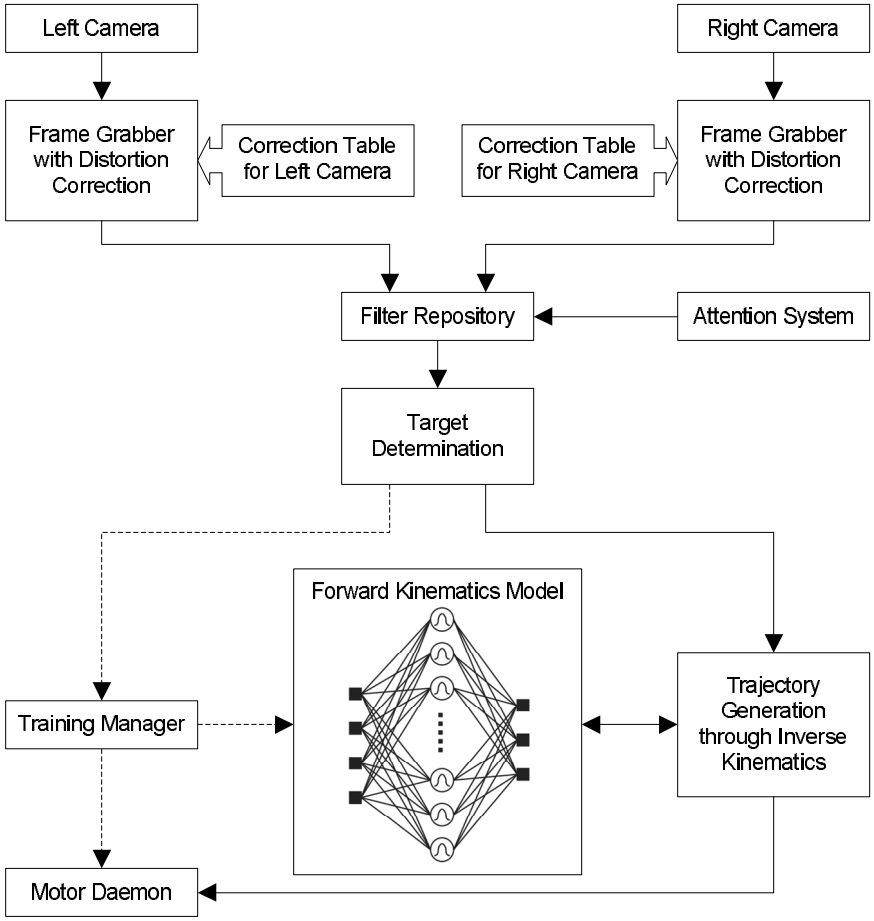

Fig. 2. System structure for learning to reach – The video frames grabbed from the eye cameras are first corrected for radial distortion. The corrected frames are either filtered by color during a training session or by a member from a filter repository selected by the attention system during a test session. Information in the filtered stereo frame pair is subsequently used by the target determination module to calculate the object position. This position is then used either by the training manager for learning or by the trajectory generator for reaching. The latter module uses the forward model constructed by the training manager to generate a reaching trajectory.

stereo vision needed for the reaching behavior, because their common vision field has only a small overlap with the reachable space of the robot arm. The high radial distortions of the two short focal length cameras are corrected by their respective frame grabbers in real-time. The radial distortion coefficients K_1 and K_2 and other camera parameters are measured for each camera through the Camera Calibration Toolbox for Matlab developed by J.-Y. Bouguet.[10] The pixel value of position (x, y) in the corrected frame is filled with the pixel value of position (x', y') in the original frame through the following equation[11] —

$$\begin{cases} x' = x(1 + K_1 r^2 + K_2 r^4) \\ y' = y(1 + K_1 r^2 + K_2 r^4) \end{cases} \quad (1)$$

If values of x' and y' are not integers, they are substituted by their respective integer parts. During the startup of the vision subsystem, a lookup table is built for each camera consisting the mappings from (x, y) to (x', y') for all possible (x, y). The pre-built lookup tables enable an efficient distortion correction in real time. We currently use a resolution of 320x240 for both cameras.

(A) (B)

Fig. 3. Original image (A) and image (B) corrected for radial distortion through Eq. (1)

During a training session the video frames are filtered by color to identify the blob representing the finger tip. During the test sessions, the grabber outputs are fed into a filter repository module that implements a variety of filters. An attention module dynamically determines the filter to be applied to the grabber outputs. A subsequent target determination module uses the processed stereo frame pair to output a spatial position. Its inputs are basically filtered frames, each of them containing at most one blob representing the finger tip. At the current stage the target determination module simply replaces each blob with its centroid and uses two corresponding centroid positions to calculate the target position to reach. Throughout all experiments described in this paper, the two eye cameras are positioned parallel to each other, so the calculation of a target position is trivial.[12]

3. Forward and Inverse Kinematics

3.1. *Forward Kinematics Learning*

Forward kinematics is defined as a mapping $\mathcal{F} : \theta \rightarrow x$, where $\theta \in R^n$ is the joint vector and $x \in R^m$ is the task space vector. It has been known in neurophysiology that the two shoulder joints and the two elbow joints move independently of the wrist joints for the most part during a reaching movement performed by human subjects.[13] This suggests that a reaching movement can be decoupled into first moving the hand into the vicinity of the desired object by actuating the shoulder and elbow joints and then aligning the hand to the object through the wrist joints. At

the current stage, we require our robot to touch a presented object without putting any restriction on the orientation of its end effector. This requirement eliminates the need for recruiting the wrist joints. Only the two shoulder joints and the two elbow joints marked in Fig. 1(B) are used. The dimensionality of the task space is reduced to three since only the spatial position is of concern. So the final form of the forward kinematics to be learned is simplified to $\mathcal{F} : \theta \rightarrow x$ with $\theta \in R^4$ and $x \in R^3$.

Learning the forward kinematics using visual feedback is essentially a procedure of approximating unknown function through training samples of the form $(\theta_i, x_i)_{i=1,2,...,n}$, where n is the total number of training samples available. The actual reach location contains noise introduced by the stereo vision system. Neural networks such as multi-layer perceptron (MLP) and radial basis function networks (RBFN) are commonly used for such function approximation tasks. We have adopted RBFN for the forward kinematics learning for a number of reasons. It has been shown that RBFNs with a set of basis functions that have a common form but different centers can approximate any continuous input-output mapping.[14] The study of empirical risk minimization shows that the complexity of the model generated by a learning algorithm from the training data determines its generalization performance.[15] Such complexity can be directly controlled for a RBFN through restricting the number of nodes in its hidden layer. We use the orthogonal least squares algorithm (OLS) introduced by Chen et al. to increase the size of the hidden layer gradually until the approximation error falls below a predetermined limit.[16] By adjusting the predetermined limit, the sensitivity of the learned function to the noise in the training data can be reduced. Unlike MLP, the linear weights between the hidden layer and the output layer of an RBFN can be determined by the linear least square method. In this way, the problem of being stuck in local minima commonly encountered in the training phase of MLP is avoided. Furthermore, it has been suggested that RBFN could be the actual learning mechanism used by biological entities for sensorimotor transformation.[17,18,19]

The training manager module shown in Fig. 2 generates one random joint vector at a time that contains for each arm joint an angular value within its limit. The joint vector is then sent to the motor daemon to initiate the actual arm movement. After the movement, if the finger tip of the arm is detectable by both video cameras, the training manager receives its spatial position from the stereo vision subsystem. This position together with its associated joint vector forms a training sample. The whole process repeats itself autonomously until enough samples are gathered. These samples are subsequently used to train a RBFN. After the training, the resulting RBFN is saved for future use. The algorithmic description of the training procedure is summarized as follows —

begin initialize $samples \leftarrow \{\}, i \leftarrow 0$
    while $i <$ *sample_number_needed*
        Randomly generate joint vector θ

```
            Send θ to the motor daemon
            Wait until the arm movement is finished
            x ← output of the stereo subsystem
            if x is empty
                continue
            else
                samples ← Union(samples, (θ, x))
                i ← i + 1
            end
        end
        Use samples to train a RBFN to approximate forward kinematics.
        Save the result.
end
```

3.2. *Solution to Inverse Kinematics*

3.2.1. *Background*

Inverse kinematics solves the problem inverse to the one forward kinematics deals with. It maps an m-dimensional task space vector into an n-dimensional joint vector. For a redundant manipulator with $n > m$, the inverse kinematics is a one-to-many mapping. In this case, the problem becomes how to select one particular solution from multiple solutions. The most popular approach to this problem is to set some criteria and find the particular solution which optimizes these criteria. The criteria to be optimized can either be global or local. Global optimization is usually too computationally expensive to be calculated on-line. Local optimization is much more flexible and computationally less expensive.

Most local optimization methods are based on the Resolved Motion Rate Control (RMRC) proposed by Whitney in 1969.[1] RMRC utilizes the following equation to solve inverse kinematics —

$$\dot{x} = J(\theta)\dot{\theta}, \tag{2}$$

where $J(\theta)$ is the Jacobian matrix for the current joint vector θ, $\dot{x}$ is the end-effector velocity and $\dot{\theta}$ is the joint velocity. For a desired position x_{target} and a start position x_{start}, θ is incrementally adjusted such that $\dot{x}$ moves the end-effector toward the x_{target}. In the ideal case, the end-effector position coincides with x_{target} at the end of the reaching movement. For a non-redundant manipulator, J is for the most part invertable so that Eq. (2) can be solved through

$$\dot{\theta} = J^{-1}\dot{x}. \tag{3}$$

For redundant manipulators, J is not invertable. To address this problem, Liegois proposed a method to solve Eq. (2) that can be expressed through

$$\dot{\theta} = J^{\#}\dot{x} + \alpha(J^{\#}J - I_n)\nabla H, \tag{4}$$

where $J^{\#} = J^T(JJ^T)^{-1}$ is the pseudo-inverse of J,[2] H is an extra optimization criterion to be minimized, and $J^{\#}J - I_n$ is the null space of J. $\alpha(J^{\#}J - I_n)\nabla H$ has no influence on $\dot{x}$ and is used only to move the joint vector to a local minimum of H. With H set to zero, Eq. (4) is simplified to a succinct form —

$$\dot{\theta} = J^{\#}\dot{x}. \tag{5}$$

The local optimization method proposed by Liegois is extremely flexible. It has numerous extensions addressing problems such as singularity avoidance, obstacle avoidance and optimization of multiple criteria.[20,21,22] The optimization criterion H can be changed during the run-time to satisfy different requirements during different phases of the reaching movement. To facilitate the subsequent parameter tuning and error analysis, we have adopted Eq. (5), Liegois' method in its simplest form, to solve the inverse kinematics problem based on the learned forward model. The $\dot{\theta}$ obtained through Eq. (5) is the one among all possible solutions to Eq. (2) with the minimum Euclidean norm.[23]

3.2.2. *Incremental generation of reaching trajectories*

Studies of physiology and neuroscience on human subjects have shown that Cartesian hand trajectories approximately follow the straight line connecting the start position and the target position.[24,25] To simulate this behavior, the end-effector is moved from its starting position x_{start} toward the desired position x_{target} through a number of via-points x_1, x_2, ..., x_{n-1} on the straight line connecting x_{start} and x_{target}. x_{start} is selected from the training samples with small Z components. The straight line between x_{start} and x_{target} is divided into segments of a fixed length. The last segment is of variable length. The endpoints of these segments are designated as the via-points. Before the execution of the reaching movement, the sequence of the joint vectors corresponding to the via-points and the x_{target} is calculated incrementally.

Eq. (5) is used to generate appropriate joint velocities over the course of the reaching movement. In our case, the joint velocities are determined by the motor controllers. Instead of doing velocity control, we use a modified form of Eq. (5) shown below to calculate the joint vectors corresponding to the via-points and the end point of the trajectory —

$$\Delta\theta \approx J^{\#}\Delta x. \tag{6}$$

Eq. (6) provides a very good approximation for small Δx, because inverse kinematics is locally linear despite its global nonlinearity. If the exact forward kinematic function along with all associated parameters is known, there is a well-known procedure for calculating J. What we have at hand is only a RBFN representing an approximation of forward kinematics designated as $\tilde{f}$ with $x \approx \tilde{f}(\theta)$. Thanks to the differentiability of the Gaussian basis function, we can easily get an approximation of J by replacing the basis functions in the hidden layer of the RBFN with their appropriate partial derivatives. For instance, in order to get an approximation of $[J_{11}, J_{21}, J_{31}]^T$ for a certain $\theta = [\theta_1, \theta_2, \theta_3, \theta_4]^T$, we simply replace the basis functions by their partial derivatives with respect to x_1 and use the output of the network as the approximation result. In this way, an approximated Jacobian matrix $\tilde{J}$ can be constructed. With J substituted by $\tilde{J}$, Eq. (6) is transformed into

$$\Delta\theta \approx \tilde{J}^{\#}\Delta x. \tag{7}$$

The algorithm for the incremental generation of a reaching trajectory is summarized below. The array $x_{i,i=1,2,...n}$ specifies the trajectory for the reaching movement. The $\tilde{x}_{i,i=1,2,...n}$ vector is the actual trajectory swept by the end-effector over the course of the movement. x_i and $\tilde{x}_i$ are not identical but should be close to each other. $\tilde{\theta}_{i,i=1,2,...,n}$ are the joint vectors corresponding to $\tilde{x}_{i,i=1,2,...n}$. They are incrementally generated according to Eq. (7) for $i = 1, 2, ..., n$.

begin

initialize $x_{start}, \theta_{start}, x_{target}, step\_length$
$dist \leftarrow \| x_{target} - x_{start} \|_2, n \leftarrow floor(dist/step\_length) + 1$
for $i \leftarrow 1$ **to** $n - 1$
$x(i) \leftarrow x_{start} + i * step\_length/dist * (x_{target} - x_{start})$
end
$x(n) \leftarrow x_{target}$
$\tilde{x}(0) \leftarrow x_{start}$
$\tilde{\theta}(0) \leftarrow \theta_{start}$
for $i \leftarrow 1$ **to** n
$\Delta x \leftarrow x(i) - \tilde{x}(i-1)$
Calculate $\tilde{J}$ and $\tilde{J}^{\#}$ for $\tilde{\theta}(i-1)$
$\tilde{\theta}(i) \leftarrow \tilde{\theta}(i-1) + \tilde{J}^{\#}\Delta x$
if i **equal** n
$\tilde{x}(i) \leftarrow f(\tilde{\theta}(i))$
else
$\tilde{x}(i) \leftarrow \tilde{f}(\tilde{\theta}(i))$
end
end

Output $\tilde{\theta}(i)_{i=1,2,\ldots,n}$ to the motor daemon.

end

4. Parameter Tuning through Simulations

4.1. *Simulation Settings*

An existing implementation of RBFN training with OLS in Matlab called *newrb* provides us with an efficient way to conduct simulations in order to tune the associated parameters of the forward kinematic learning and evaluate its performance. All simulations follow a common scheme shown in Fig. 4. Instead of being used to initiate actual arm movements, the randomly generated joint vectors are mapped into spatial positions of the end effector through standard homogenous transformations. The transformation matrices are constructed with the parameters in the design specification of the arm. The original stereo vision system is replaced by a simulated version using the parameters of the actual stereo camera system on the robot head.

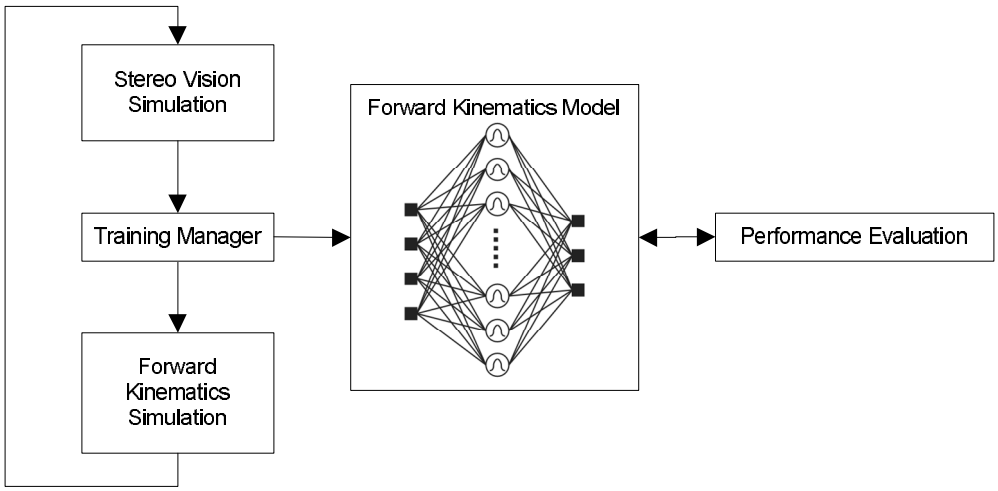

Fig. 4. Architecture of Matlab simulations for tuning the associated parameters of the forward kinematics learning and evaluating its performance. In the simulations, actual arm movements and stereo position measurements are replaced by their corresponding simulated versions. A separate module evaluates the approximation accuracy and the resulted reaching accuracy of the forward model.

Fig. 5 shows the scatter plots of a set of 4000 task space positions that fall into the overlapped vision field of the two cameras. Each of them corresponds to a randomly generated joint vector. The ranges of motion for the four joints are $\theta_1 \in [-20°, 120°], \theta_2 \in [-60°, 15°], \theta_3 \in [-60°, 45°]$ and $\theta_4 \in [0°, 120°]$. This figure shows the approximate extent of the actual reachable space. No position outside

this space can be reached, because it is either out of the overlapped vision field or has no correspondent joint vector.

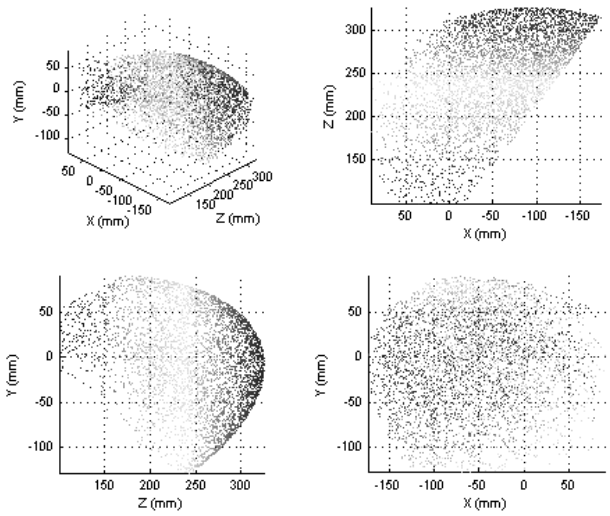

Fig. 5. The scatter plots of a set of 4000 reachable task space positions that fall into the overlapped vision field of the stereo cameras. These positions are both plotted in 3-D and projected to the X-Z, Y-Z and X-Y planes. They show the approximate extent of the actual reachable space. Part of the boundary of the reachable space is determined by the joint limits of the arm. The rest is determined by the boundary of common vision field of the cameras, which can be clearly seen from the X-Z projection that shows a straight edge on the right side.

4.2. *Error Classification*

The errors from the motor control system and the stereo vision system prevent the learning of an exact forward model. The error of the motor control system is mainly the residual positioning error of the underlying PID controllers. The error of the stereo vision system is the result of limited camera resolution. In our case, the latter one is dominant so that it is incorporated into the stereo vision simulation module to study its influence on the reaching accuracy. It is also referred to as *StError* throughout this section.

The cause of *StError* is exemplified in Fig. 6(A). As can be seen from this drawing, all positions in the shaded area falls to the same pixel on both the left and the right image plane, so the stereo vision system maps all positions in this area to one common position marked by the dot. For a position $x = f(\theta)$ in the reachable space, the associated stereo error is defined through $StError(x) = st(x) - x$, where $st(x)$ is the output of the stereo vision system for x as the input. Both x and $st(x)$ are defined in the coordinate system shown in Fig. 6(B).

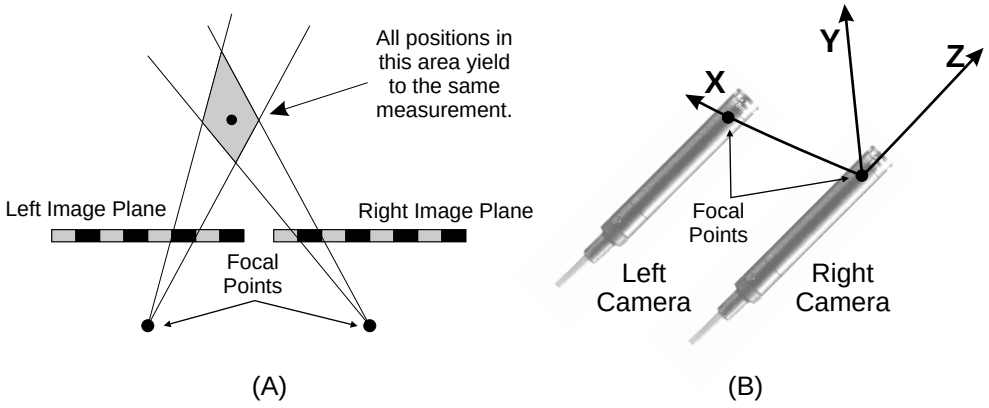

Fig. 6. (A) exemplifies the error of the stereo vision system through exaggerating the pixel size on the image planes. Only a small number of pixels are shown for each image plane. Any task space position falling into the shadowed area extended by the same pixel on each image plane yields to the same measurement which is calculated by the stereo vision system and marked with the dot in the center. (B) shows the coordinate system used to measure the positions in the task space. Its origin coincides with the focal point of the right camera. The X axis points to the focal point of the left camera. The Z axis lies on the principle axis the right camera and points outwards. The Y axis is determined by the X and Z axes through the right hand rule.

The task space vector in each training sample has a $StError$ associated with it, because after each random movement, the end effector position is measured by the stereo vision system. The RBFN trained on these samples can provide us with only an approximate forward model $\tilde{f}$. For each joint vector θ, the RBFN outputs a position vector $\tilde{f}(\theta)$. The position approximation error for θ is defined through $PosError(\theta) = f(\theta) - \tilde{f}(\theta)$. The local Jacobian matrix $\tilde{J}(\theta)$ constructed through the RBFN is also different from its true value $J(\theta)$. However, the difference $J(\theta) - \tilde{J}(\theta)$ is not a convenient measure for the Jacobian approximation error. Instead, the Jacobian approximation error for θ is defined as $JacobError(\theta) = f(\theta) - \tilde{x}(n)$, where $f(\theta)$ is assigned to x_{target} and the original algorithm is run with line 17 substituted by $\tilde{x}(i) \leftarrow f(\tilde{\theta}(i))$ to calculate $\tilde{x}(n)$. In this way, the effect of approximated Jacobian matrices on the reaching accuracy is completely isolated from the position approximation errors because $\tilde{f}$ is no longer used for the trajectory generation.

The last error category we want to define here is the actual reaching error $ReachError$. This error is the compound effect of $StError$, $PosError$ and $JacobError$. Both $PosError$ and $JacobError$ affect $ReachError$ because $\tilde{f}$ and $\tilde{J}$ are used for the calculation of the reaching trajectory. The direct effect of $StError$ on $ReachError$ is caused by the position perception error of the desired object. $ReachError(\theta)$ is defined as $f(\theta) - \tilde{x}(n)$, where in contrast to $JacobError(\theta)$, $\tilde{x}(n)$ is calculated by assigning $st(f(\theta))$ to x_{target} and then running the original trajectory generation algorithm.

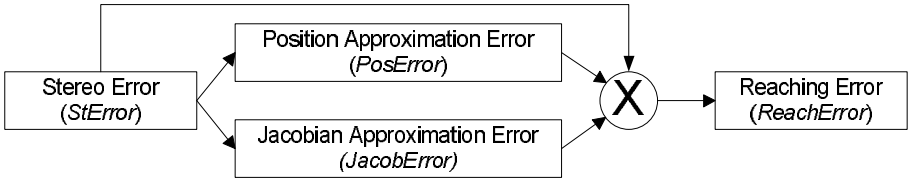

Fig. 7. Relationship between *StError*, *PosError*, *JacobError* and *ReachError*. *StError* is the result of limited camera resolution. *PosError* and *JacobError* are caused by the approximation error in the learned forward model. They are influenced by *StError* and the values of learning parameters. *ReachError* is the final position error of the end effector after the reaching movement. Its magnitude is determined by *StError*, *PosError* and *JacobError*.

4.3. *Parameter Tuning*

Each radial basis function in the hidden layer of our RBFN can be expressed as $g(x) = exp(\parallel x - c \parallel * 0.8326/spread)$. x is the input vector to this function, c is the adopted center of the function g. The parameter *spread* controls the extent of g's influence in its neighborhood. Another parameter *error_margin* is used as the stop criterion for the learning procedure. Learning is stopped if the mean square error of the network output averaged over the dimension of the output vector falls below $error\_margin^2$. Both *spread* and *error_margin* must be specified before training. An additional parameter to be determined is the size of the training set (*size*). Setting the values of *spread* and *size* appropriately makes it possible to use only a small number of training samples to achieve a high reaching accuracy. The error measures defined in Section 4.2 provide us with the criteria to determine the appropriate values for *spread*, *error_margin* and *size*. As can be seen from Fig. 7, *ReachError* is the compound effect of *StError*, *PosError* and *JacobError*. Through parameter tuning, we can only influence *PosError* and *JacobError*. So only these two error measures are used for the subsequent tuning process.

Since we only have three parameters to tune, simple exhaustive search enables us to find appropriate values for them. During simulations, values for *spread*, *error_margin* and *size* are chosen from $[20, 30, ..., 140, 150]$(degree), $[1, 2, ..., 7, 8]$(mm) and $[40, 80, 120, 160, 200, 400]$ respectively. For each particular value combination (sp, e, si), a random training set of size si is generated. A RBFN is trained on this training set with learning parameters set to (sp, e). A test set of 400 samples is used to measure the performance of the trained RBFN. Each test sample is a joint vector θ_i. The corresponding end effector position associated with each θ_i is within the common field of view of the stereo cameras. Calculation of *PosError* is straight forward for each θ_i. We use a fixed x_{start} to calculate *JacobError* for each θ_i.

The *PosError* averaged over the test set and all possible *error_margin* for each particular combination of *spread* and *size* is shown in Fig. 8(A). The *PosError* averaged over the test set and all possible *spread* for each particular combination of *error_margin* and *size* is shown in Fig. 8(B). From Fig. 8(A), it can be seen that

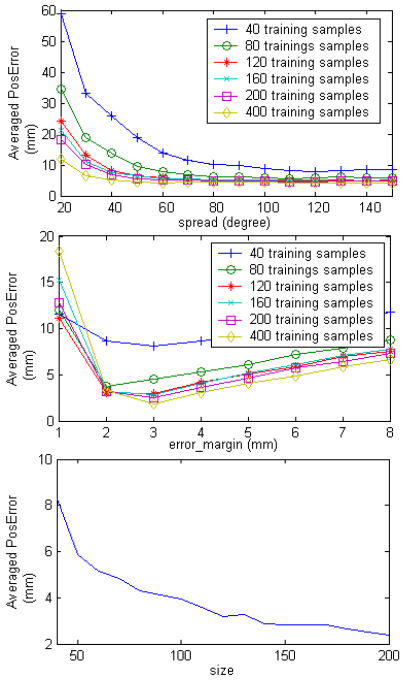

Fig. 8. Simulation results for tuning of the three parameters - *spread*, *error_margin* and *size* - for the learning procedure. Refer to Section 4.2 for detailed discussions.

the change of *spread* has a relatively large influence on the averaged *PosError* when *spread* is small. For *spread* larger than 100, the change of *spread* no long has any noticeable influence on the averaged *PosError* regardless of the size of the training set. From Fig. 8(B), it is clear that the averaged *PosError* reaches its minimum at $error_margin = 3$mm for almost all training sets of different sizes. The averaged *PosError* skyrockets when *error_margin* is reduced below 2, which indicates that the *newrb* function is unable to construct a RBFN with a very small approximation error within the given parameter space.

We use 110 and 3 as the optimal parameter values for *spread* and *error_margin* and measure the averaged *PosError* against different *size*. The result (see Fig. 8(C)) shows that the averaged *PosError* decreases dramatically at the beginning but such decrease becomes much slower for *size* larger than 120.

For the actual experiments carried out on Nico, the error in the training set is certainly not the same as the one used for the simulations. Uncertainties of camera calibration, uneven illumination, using centroids for position determination all contribute to a larger *StError*. From Fig. 8(B) we can see that if the *error_margin* is set too small or too large, the resulted network will deliver an averaged *PosError* larger than the minimum. It is very hard to produce curves like those in Fig. 8(B) through actual experiments. But the hidden layer size of the trained RBFN can

give us a clue to select the optimal *error_margin*. Fig. 9(A) and 9(B) show the relationship of averaged *PosError* and hidden layer size versus *error_margin* for $spread = 110$, $size = 400$ and 320x240 video resolution. We can see that the optimal *error_margin* for *PosError* also corresponds to the transition point of the curve of hidden layer size. Fig. 9(C) and 9(D) show the same relationship, only with video resolution reduced to 160x120. Lower video resolution makes the average *StError* larger and also shifts the optimal *error_margin* to the right. But the new optimal *error_margin* still corresponds to the transition point of the curve of hidden layer size. This heuristics is used to select the optimal value of *error_margin* for the experiments conducted on Nico.

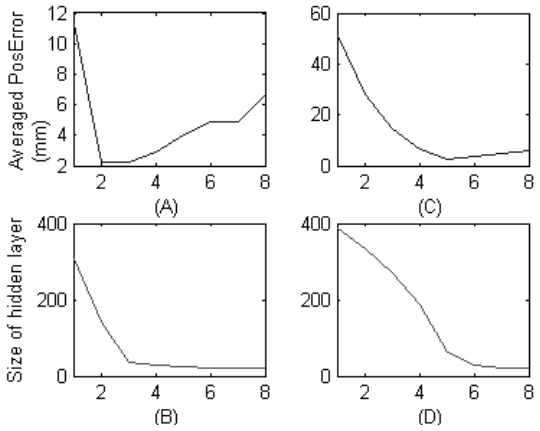

Fig. 9. (A) and (B) show averaged *PosError* and hidden layer size versus *error_margin* represented by the X-axis for *spread*=110, *size*=400 and 320x240 video resolution. (C) and (D) show the same relationships, only with the video resolution reduced to 160x120. It can be seen that the same *error_margin* value corresponds to both the transition point of the curve of the hidden layer size and the minimum point of the *PosError* curve.

It should be noted that we have only used *PosError* for parameter tuning because the curves of *JacobError* versus *spread*, *error_margin* and *size* display very similar features to those of *PosError*. The only differences are that the curves of *JacobError* are much flatter and *JacobError* is typically much smaller than *PosError*.

4.4. *Reaching Accuracy and Visual Feedback*

Fig. 10 shows the histograms of the four different error types for $spread = 110, error\_margin = 3, size = 120$. The mean and standard deviation of *ReachError* is 4.03mm and 3.94mm respectively. It can be observed that the *JacobError* caused by Jacobian approximation is much smaller than *PosError*.

Since the reaching trajectory is precalculated, no visual feedback is used during the reaching movement.

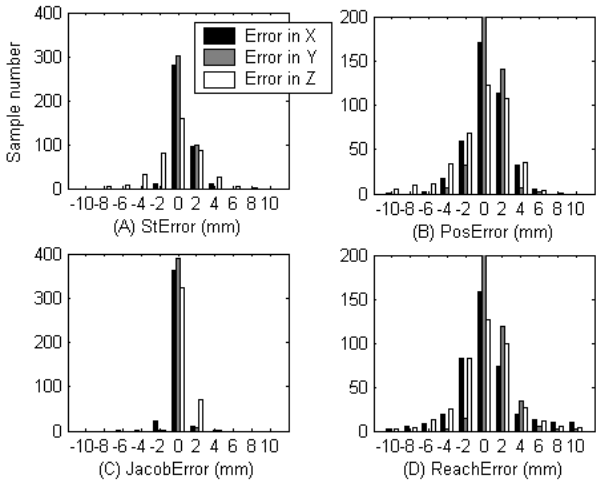

Fig. 10. (A),(B),(C) and (D) show the histograms of the four types of error calculated on a test set of 400 samples. It can be observed that the *JacobError* is much smaller than *PosError* on average. mean(*StError*)=2.20, std(*StError*)=1.59, mean(*PosError*)=3.37, std(*PosError*)=2.43, mean(*JacobError*)=0.75, std(*JacobError*)=0.84, mean(*ReachError*)=4.03, std(*ReachError*)=3.94.

Fig. 11(A) shows the *ReachError* histogram for $spread = 110, error\_margin = 3, size = 40$. It can be seen that with less training samples, the histogram exhibits significantly more outliers. According to the trajectory generation algorithm described in Section 3.2.2., the final reaching error *ReachError* is caused in a large part by the error in $\tilde{x}(n-1)$. If we allow a one-time visual feedback at the $(n-1)$-th step of the trajectory generation, which essentially delivers the perceived position of the end effector at this time step, $st(f(\tilde{\theta}(n-1)))$ instead of $\tilde{f}(\tilde{\theta}(n-1))$ is assigned to $\tilde{x}(n-1)$. Since the stereo error is smaller than the position approximation error, the one-time visual feedback should improve the reaching accuracy. Simulation result shown in Fig. 11(B) confirms this conjecture. It shows the histogram of *ReachError* for the same parameter setting if a one-time visual feedback is allowed. It is visually very close to Fig. 10(D).

5. Experiment Results on Nico

Experiments have been carried out on Nico to test the performance of the proposed approach. 120 samples are gathered before the forward kinematics learning. It takes only about 15 minutes to gather a training set of this size. *spread* is set to 110 as

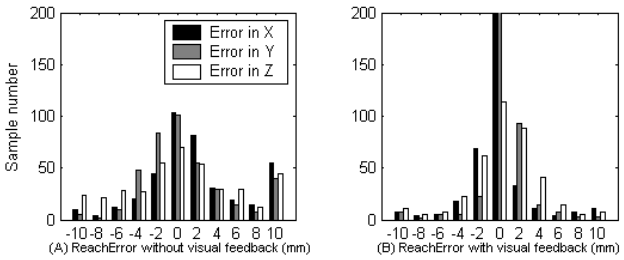

Fig. 11. The left histogram shows the distribution of *ReachError* when the reaching trajectory is generated without visual feedback. The values of learning parameters are $spread = 110$, $error\_margin = 3$ and $size = 40$. Because the training set is very small, the histogram of *ReachError* is more spread out and exhibits quite a few outliers. The right histogram shows the distribution of *ReachError* when a one-time visual feedback is allowed during the reaching movement. It can be seen that *ReachError* is drastically reduced and the distribution is visually almost identical to Fig. 10(D).

usual. Fig. 12 shows the plot of hidden layer size versus *error_margin*. Using the heuristics described in Section 4.3, we set the final value of *error_margin* to be 3.8.

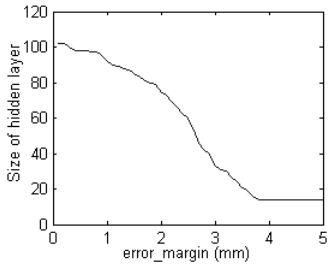

Fig. 12. The curve of hidden layer size versus *error_margin* for a training set of 120 autonomously gathered samples. It shows a salient transition point at $error\_margin = 3.8$. Using the heuristics described in Section 4.3, we use this value for training the RBFN to evaluate the reaching performance.

After the training, we use a wooden ball of the same size as the one attached to the wrist plate of the robot as the reaching object. The ball is fixed on the tip of a modified retractable TV antenna that allows flexible positioning of the object. In a test session, the ball was put into 100 different positions for Nico to reach. If a frontal contact took place at the end of the reaching movement, it was counted as a success. The concept of frontal contact is illustrated in Fig. 13. No visual feedback was used throughout all reaching movements. A sequence of images capturing the course of one successful reach is shown in Fig. 14. The final success rate of the test session is 92%. Most of the unsuccessful movements happened when the ball was placed near the boundary of the reachable space of the robot.

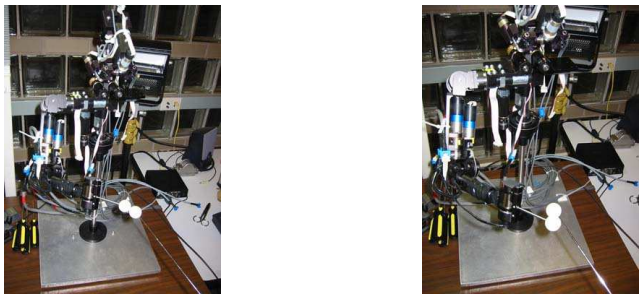

Fig. 13. (A) shows an example of frontal contact, where Nico's finger tip touches the object on the surface part facing the eye cameras. Small overshoot or undershoot within 9mm also counts as a frontal contact. (B) shows an example of Nico's finger tip moving past the object. Although at the end of this reaching movement the finger tip also touches the object, the result does not count as a frontal contact.

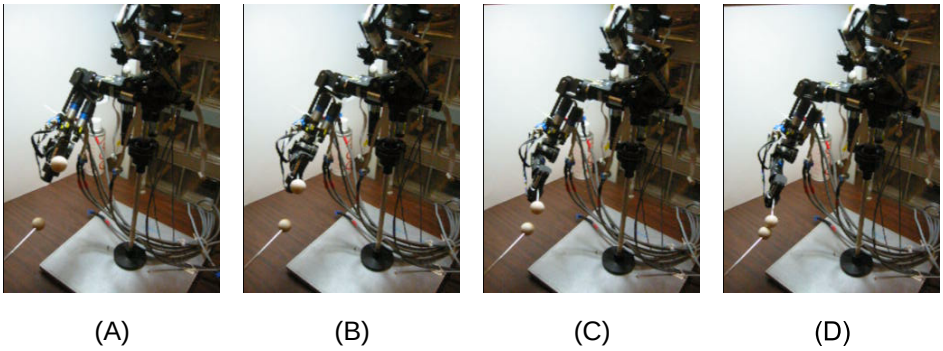

Fig. 14. (A),(B),(C) and (D) show a sequence of images capturing the course of one successful reach.

6. Discussion

Through this paper, we have proposed a procedure of learning to reach through first learning a model of the arm forward kinematics and then using the learned model to generate reaching trajectories. An error classification scheme and the detailed parameter tuning process are included in this paper. We have shown through simulations and actual experiments that a small training set for learning the forward model allows accurate reaching movements relying solely on proprioceptive feedback. The size of the training set can be cut back further if a one-time visual feedback is used to aid reaching when the end effector is already moved into the neighborhood of the desired object. The modular structure of the system devised and implemented to evaluate our approach can be easily extended to allow the robot to reach for different objects in its environment according to the focus of its

attention.

Interestingly, some psychological studies on human reaching behavior reveal two important features that are also present in our system. One study conducted by R.S. Johansson and colleagues shows that adults almost never explicitly look at their hands during reaching movements.[26] Experiments by R.K. Clifton and colleagues show that infants can reach glowing objects in the dark without visual feedback of their hand position.[27] Our study manifests that reaching without visual feedback is indeed possible even an approximate forward model is learned on a small training set. A study by von Hofsten shows that the arm movements of neonatal infants consist of multiple segments.[28] This work has been confirmed by other researchers.[29] The multiple segments might be the results of incremental trajectory generation, which is the approach we use to solve inverse kinematics.

Natural extensions to the work described in this paper include exploiting visual feedback to a greater extent to improve reaching accuracy, investigating the performance of our learning approach if the original version of Liegois's method instead of its simplified form is used for trajectory generation, and learning to use additional DOFs for reaching while avoiding the curse of dimension.

Acknowledgements

Support for this work was provided by a National Science Foundation CAREER award (#0238334). Some parts of the architecture used in this work was constructed under NSF grants #0205542 (ITR: A Framework for Rapid Development of Reliable Robotics Software) and #0209122 (ITR: Dance, a Programming Language for the Control of Humanoid Robots) and from the DARPA CALO project (BAA 02-21).

References

1. D.E. Whitney. Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine Systems*, 10(2):47-53, 1969.
2. A.Liegeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Transactions on Systems, Man and Cybernetics*, 7(12):868-871, 1977.
3. J.Konczak and J.Dichgans. The development toward stereotypic arm kinematics during reaching in the first 3 years of life. *Experimental Brain Research*, 117:346-354, 1997.
4. K.-K. Kim and Y.-S. Yoon. Practical inverse kinematics of a kinematically redundant robot using a neural network. *Advanced Robotics*, 6(4):431-440, 1992.
5. A.DSouza, S.Vijayakumar and S.Schaal. Learning inverse kinematics. *Proc. of 2001 IEEE/RSJ IROS*, Maui, Hawaii.
6. D.DeMers, K.Kreutz-Delgado. Inverse kinematics of dextrous manipulators. in *Neural Systems for Robotics*, Academic Press, New York, 1997, pp. 75-116.
7. S. Lee and R.M. Kil. Bidirectional continuous associator based on gaussian potential function network. *Proc. of 1989 IJCNN*, Washington, D.C.
8. M.I. Jordan and D. Rumelhart. Supervised learning with a distal teacher. *Cognitive Science*, 16:307-354, 1992.
9. K.Narendra and K.Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4-27, 1990.

10. J.-Y. Bouguet. Camera Calibration Toolbox for Matlab. *http://www.vision.caltech.edu /bouguetj/calib_doc/.*
11. J. Heikkilä and O. Silvén. Calibration procedure for short focal length off-the-shelf CCD cameras. *Proceedings of 13th International Conference on Pattern Recognition*, pp.166-170, 1996.
12. L.G. Shapiro and G.C. Stockman. *Computer Vision.* (Prentice-Hall, New Jersey, 2001), pp.397-398.
13. F. Lacquaniti and J.F. Soechting. Coordination of arm and wrist motion during a reaching task. *Journal of Neuroscience*, 2(2):339-408, 1982.
14. J. Park and I.W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246-257, 1991.
15. V.N. Vapnik. *The Nature of Statistical Learning Theory*, 2nd edn. (Springer Verlag, New York, 1999).
16. S. Chen, C.F.N. Cowan and P.M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302-309, March 1991.
17. T.A. Poggio. A theory of how the brain might work. *Cold Spring Harbor Symp. Quant. Biol.*, 55:899-910, 1990.
18. A. Pouget and T.J. Sejnowski. Spatial transformations in the parietal cortex using basis functions. *Journal of Cognitive Neuroscience*, 9:227-237, 1997.
19. A. Pouget and L.H. Snyder. Computational approaches to sensorimotor transformations. *Nature Neuroscience*, 3:1192-1198, November, 2000.
20. Y. Nakamura and H. Hanafusa. Inverse kinematic solutions with singularity robustness for robot manipulator control. *ASME Journal of Dynamic Systems, Measurement and Control*, 108:163-171, 1986.
21. A.A. Maciejewski and C.A. Klein. Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *The International Journal of Robotics Research*, 4(3):109-117, 1985.
22. C. Kapoor and M. Cetin. Performance based redundancy resolution with multiple criteria. *Proc. of DETC98*, Atlanta, Georgia.
23. D.N. Nenchev. Redundancy resolution through local optimization: a review. *Journal of Robotic Systems*, 6(6):769-798, 1989.
24. P. Morasso. Spatial control of arm movements. *Experimental Brain Research*, 42:223-227, 1981.
25. P. Vindras and P. Viviani. Frames of reference and control parameters in visuo-manual pointing. *Journal of Experimental Psychology: Human Perception and Performance*, 24:569-591, 1998.
26. R.S.Johansson, G.Westling, A.Bäckström and J.R. Flanagan. Eye-hand coordination in object manipulation. *The Journal of Neuroscience*, 21(17):6917-6932, 2001.
27. R.K.Clifton, D.W.Muir, D.H.Ashmead and M.G.Clarkson. Is visually guided reaching in early infancy a myth? *Child Development*, 64:1099-1110, 1993.
28. C.von Hofsten. Development of visually directed reaching: the approach phase. *Journal of Human Movement Studies*, 5:160-168, 1979.
29. A.Mathew and M.Cook. The control of reaching movements by young infants. *Child Development*, 61:1238-1257, 1990.