"Neural Network System Techniques and Applications",
International Series on Advances in Control and Dynamic Systems, ed. C.T. Leondes, pp. 353–394, Academic Press, 1998.

Radial Basis Function Network Approximation and Learning in Task-dependent Feedforward Control of Nonlinear Dynamical Systems

Dimitry Gorinevsky*
Honeywell-Measurex, North Vancouver, B.C., Canada V7J 3S4

Contents

*Current address: Honeywell Technology Center, Honeywell CA35-5272H, One Results Way, Cupertino, CA 95014, email: dimitry.gorinevsky@honeywell.com

Abstract

The paper considers intelligent control system architectures for task-level control. The problem is to compute feedforward control for a sequence of control tasks. Each task can be compactly described by a task parameter vector. The control update is performed in a discrete time: from task to task. The paper considers an innovative controller architectures based on Radial Basis Function (RBF) approximation of nonlinear mappings. The more advanced of these architectures enable on-line learning update for optimization of the system performance from task to task. This learning update can be considered as a generalization of the well-known learning (repetitive) control approach. Unlike the repetitive control, which is only applicable to a single task, the proposed algorithms work for a parametric family of such tasks. As an example, a task-level feedforward control of a flexible articulated arm is considered. A vibration-free terminal control of such arm is achieved using a task-level algorithm that learns optimal task-dependent feedforward as the arm goes through a random sequence of point-to-point motions.

1 Introduction

Motor control in humans and animals is believed to use feedforward widely instead of relying only on feedback. One hypothesis on feedforward control organization is that control programs of complex motions are learned, memorized, and then just extracted from the memory when needed. This hypothesis provided an inspiration for this paper, which presents a paradigm for the task-level feedforward control. The practical value of this paradigm is demonstrated by applying it to solve a difficult control problem.

Classical automatic control theory heavily concentrates on the issues of low-level feedback control or feedforward compensation of disturbances. This is because most controlled systems used in industrial and other applications in the past two decades and many of those used now are characterized by a low level of computational power and relatively simple logic of operation. In addition to feedback control, classical control theory paid much attention to the issues of open-loop (programmed) control, in particular, optimal control. This was initially motivated by space flight and ballistic missile control applications, where the control program for a mission can be accurately computed in advance.

Many modern advanced control systems are integrated systems with high-performance computers, multiple sensors and actuators. These advanced systems possess automated operation, adaptation and self-tuning features, build-in identification and fault diagnosis capability. The trend in control practice is towards development and deployment of intelligent control systems performing increasingly complex tasks with minimal or no operator supervision and adapting to changing operation condition. The setup, commissioning, and operation regime change for such systems should be also automated. Development of intelligent systems requires comprehensive multi-layered control and information processing architectures, that are more complicated than classical feedback loops.

The task-level control algorithms considered in this paper are assumed to be executed one level above the traditional planning and servo feedback level. To define this new control level, we assume that the overall motion planning and feedback tracking of this planned motion is a sequence of separate (but possibly related) tasks. We assume that each task and a control problem associated with this task are completely defined by a few task parameters. The algorithms considered in this paper operate in discrete time: from task to task. In this respect the proposed controllers can be considered as hybrid systems.

There are a few groups of papers in the literature more specifically related to the topic and technical approaches of this paper. Recently, a *learning control* paradigm has been considered by a number of authors, starting by Arimoto [3], mostly with regard to the manipulator path tracking. The paradigm regards a feedforward control program as a high-dimension array, which stores a time-history of the feedforward control input. Authors of [2, 3, 20, 38, 39, 41, 48, 54, 68] assume that a dynamic model of the system is to some extent known and consider an iterative method for improving the performance by updating a feedforward control program in the course of repeated motion trials. The iterations converge for a *single* given motion, but the learned control would not work for another trajectory.

Another direction of work related to the technical approach of this paper is associated with approximation-based control. In a sense, any practical linear control approach is based on a model, which is but an approximation of a real plant. Herein we consider nonlinear gray box approaches that use generic computational architectures to approximate unknown nonlinear mappings. Such approximation-based approaches are usually applied within neural network or fuzzy logic framework.

Many authors use neural networks to acquire knowledge of the controller plant dynamics that allows to compute feedforward control at each instant, given the planned motion, velocity, and acceleration (for instance, see [42, 46, 62]). Some related papers consider different techniques for computing the feedforward through approximation of the dynamics mapping of the system by using polynomial associative memories [4, 69, 70] or fuzzy control. Such an approach to learning is capable of generalization, which means that the knowledge of inverse dynamics acquired for

one trajectory allows to compute feedforward control for other motions. Yet robustness of such approach to high-order dynamics is not always acceptable and real-time implementation of the approach requires enormous resources, especially if the order of the system dynamics is high.

A distinct learning control paradigm combining features of the two above-described approaches was proposed in the author's earlier papers [22, 24, 25]. According to this paradigm, *a mapping* from the task parameters to the feedforward control program is learned (approximated). An example of the task parameters are an initial and a desired final state of the system in a transient motion control. The described approach is also related to the *task-level control* paradigm proposed in [1, 8].

This paper surveys the author's work on approximation and learning in task-level control problems [22, 24, 36, 35, 34]. Herein, we mostly concentrate on the paradigms and algorithm ideas. The convergence results and applications of the approach are referenced but not presented in any detail. The learning process, which we consider in this paper, can be regarded as an adaptive control of a linearly parametrized nonlinear system. In [27] we demonstrate an application of the algorithms considered in this paper to the adaptive control of an unstructured nonlinear system.

The proposed learning control paradigm has several fundamental advantages that make it potentially very useful for many applications. First, our approach is well suited for real-time implementation, since the entire control program is computed before the motion begins. Second, unlike sophisticated feedback controllers, the approach is robust to high-order unmodelled dynamics. Third, high sensitivity to the system parameter variations, which is inherent to open-loop control, is compensated by the on-line feedback update of the controller parameters.

Our approach is based on using a connectionist network approximation of the control program dependence on the task parameter vector. Such network has a fixed number of weights that can be updated based on the system operation results. This paper uses a Radial Basis Function (RBF) network architecture for approximating dependences such as feedforward program dependence on the task parameters. It has recently been acknowledged that in many problems the RBF networks possess superior spatial filtration and approximation accuracy properties, as compared to the Multilayered Perceptron networks [12, 13, 33, 40, 44, 50, 64]. Even more important for this study is that the RBF network output is *linear* in the network weights. This property makes the powerful tools of the linear system theory applicable to system estimation (identification) with an RBF network. So designed algorithms converge much faster than usual neural network algorithms based on gradient descent (backpropagation). Furthermore, the computational complexity of the RBF network approximation does not grow with the dimension of the output variable, which is usually large for the problems of the considered type.

Task-level algorithms considered in this paper are expected to be useful for many application. Currently, applications of these algorithms have been applied to problems of robotics, process control, automotive, and flexible spacecraft control. In this paper, we demonstrate efficiency of the proposed approach in control of flexible arm motions. We simulate very fast arm motions that take only about 1.5 periods of the lowest eigenfrequency oscillations. The system is oscillatory and very nonlinear, and, therefore, difficult to control using classical approaches even if the system dynamics are known exactly. For this system, we achieve a high control performance by using a learning control algorithm without any a priori information on the system dynamics.

The outline of the paper is as follows. Section 2 considers a general statement of the task-level control problem. One of the section goals is to show how such problems relate to the classical feedback and programmed control problems. A consize formal mathematical problem statement of the task-level control is related to discrete-time on-line optimization of a static multivariate vector-valued mapping. We formulate four different problem statements, which assume different degrees of uncertainty about the system. The problems stated in Section 2 are then considered in four subsequent sections of the paper.

Section 3 presents some background facts on the RBF approximation. The formulation of this section constitutes the basis for the development of Sections 5 and 6. The section also discusses a basic architecture of task-level controller based on the RBF network approximation. Section 4 discusses learning control algorithms applicable for a fixed parameter vector. The problem statement is similar to the standard learning control papers cited above, but the suggested algorithms differ from most of these papers. The advantage of the algorithms considered is that they can be conveniently generalized to the case of changing task parameter vector.

Sections 5 and 6 propose task-level feedforward control algorithms with learning (adaptive) capabilities. In Section 5 we assume that the system sensitivity information is available prior to the system operation, similar to system gain information in feedback control. We then derive an algorithm for on-line update of the feedforward control approximation based on the task completion errors. In Section 6, the sensitivity information is assumed to be inaccurate and is updated on-line in the adaptive control manner. Section 6 also presents an application of the algorithm to the terminal control of a two-link flexible manipulator. The manipulator performs a random sequence of point-to-point motion tasks.

2 Problem statement

This section presents a formal statement of the learning control problem. The problem is formulated in a general form, so that the statement is applicable to a large class of control systems and tasks. To make the exposition more transparent, Subsection 2.3 introduces an example problem - one of feedforward control of a flexible planar arm - and illustrates how the general formulation considered can be applied to this example. Further, in Section 6, the algorithms developed in the following sections are applied to the same example.

2.1 Control formulation

This paper considers task-level control problems and algorithms. Such algorithms perform computations and updates of control variables and internal models from task to task. They are essentially discrete-time algorithms evolving with the performed task number. The goal of this and the two following subsections is to explain how these task-level algorithms relate to more classical issues of feedback, feedforward, and programmed control. To this end, we start with a continuous-time controlled system in a state-space form and then arrive at the task-level control formulation for such system. The task-level control problems and algorithms considered further, however, are generic discrete algorithms (in the same sense as classical optimization algorithms) and can be applied to a wider class of practical problems.

Let us consider a parametric family of nonlinear time-invariant systems of the form

$$\dot{x} = f(x, u; \mu), \tag{1}$$

$$y = h(x), \tag{2}$$

where $x \in \Re^{n_x}$ is a state vector, $y \in \Re^{n_y}$ is an observation vector, $u \in \Re^{n_u}$ is a control input vector, $\mu \in \mathcal{M} \subset \Re^{n_\mu}$ is a vector of system parameters, and $\mathcal{M}$ is a given domain. We further assume that the nonlinear mappings $f(\cdot, \cdot; \cdot)$ and $h(\cdot)$ are smooth and the nonlinear system (1), (2) is known to be observable and reachable for each value of the parameter $\mu \in \mathcal{M}$. We will discuss the assumptions about the smoothness, controllability, observability, and nature of available information about the system in more detail later on as they will be needed.

The system (1), (2) describes a task being executed. This system evolves in a local time, which is the time since the beginning of the task. In the sections to follow, we will consider issues of *task-level* control, i.e., the control and estimation evolving from one task to another. It is assumed that a local time t can be used for description of system dynamics in each task.

In each task, we consider a controlled motion of the system (1), (2) on the given interval $[0, T]$ of the local time t and assume that the initial state vector belongs to a smooth manifold of the form

$$x(0) = \psi(\lambda), \tag{3}$$

where $\lambda \in \Re^{n_\lambda}$ is a vector that defines initial conditions.

The control problem for the task is to find a control input $u(\cdot)$ defined on the time interval $[0, T]$ that allows to achieve the control goal formulated as minimization of the performance index of the form

$$J_1(y(\cdot), y_d(\cdot\,; \lambda, \nu); u(\cdot)) \to \min \tag{4}$$

Here $\nu \in \Re^{n_\nu}$ is a vector that defines the task goal, for instance, in the form of the desired system state at the end time T, trajectory of motion, etc.; and $y_d(t; \lambda, \nu) \in \Re^{n_y}$ is a preplanned desired output of the controlled plant in the task. We will call ν a vector of the control goal parameters. The preplanned output y_d depends on both the initial condition vector λ and the control goal vector ν.

Let us introduce a task parameter vector p comprising the initial condition vector λ (3), the vector of the system parameters μ (1), and the vector of the control goal ν (4) for this task

$$p = \begin{bmatrix} \lambda \\ \mu \\ \nu \end{bmatrix} \in \Re^{N_p}, \quad N_p = n_\lambda + n_\mu + n_\nu \tag{5}$$

The optimal feedforward control input $u(\cdot)$ that solves the problem (1)–(4) depends on the vector p (5). We assume that the vector p belongs to a given compact domain $\mathcal{P} \subset \Re^{n_p}$.

We further assume that we can repeatedly apply the computed feedforward control $u(\cdot)$ to the system (1), (2) in different tasks, observe its output $y(\cdot)$ on the time interval $[0, T]$, and, possibly, use the obtained observations to update the control. For each task, we suppose that the local time t is reset to zero and the initial condition has

form (3). We assume that the parameter vectors λ, μ, and ν can take different values for different runs (system motions). The control objective is to minimize the performance index (4) for each value of the parameter vector $p = [\lambda^T \ \mu^T \ \nu^T]^T$ (5) in the given domain: $p \in \mathcal{P} \subset \Re^{N_p}$.

At this stage, we do not specify how the parameter vector p (5) changes from one task to another. This issue is discussed further on in Section 4. The task-level learning control algorithms, which we are going to discuss, are more general than those considered in previous papers on learning and repetitive control. Standard learning control formulations, e.g., [2], correspond to assuming the parameters λ, μ, and ν to be fixed. Repetitive control setting [39] assumes that the initial conditions for each task coincide with the final system state at the previous task, and the variation of the initial conditions is small. Furthermore, the prior learning control work mostly confines itself to the trajectory tracking problem. The lask-level learning paradigm of [1, 8] is related to the above general problem statement, but covers a somewhat different class of problems. Herein we formulate a control problem as a minimization of a general performance index, which includes most of the previous learning control formulations as special cases. In particular, as shown below, such formulations can be used for both the trajectory tracking and terminal control problems. A few recently published papers study control approaches related to some aspects of the considered formulation [6, 63].

As already mentioned in the Introduction, the learning control papers usually consider a trajectory tracking problem. This problem can be described with a performance index of the form (4)

$$J_2(y, y_d, u) = \int_0^T \|y(t) - y_d(t\,; \lambda, \nu)\|^2 dt + \rho \int_0^T \|u(t)\|^2 dt \ \rightarrow \ \min, \tag{6}$$

where $\|\cdot\|$ denotes the Euclidean norm of a vector. The first term in (6) is a penalty for the deviation of the system output from its desired value. Since the trajectory tracking problem is generally ill-posed, the second term in the performance index (6) is very important. It regularizes the solution in accordance with the technique of [67]. For $0 < \rho \ll 1$, a solution to (6) provides good quality of the trajectory tracking (see [24] for more discussion on the subject).

Similarly, the point-to-point control problem can be described with the performance index (4) of the form

$$J_{2f}(y, y_d, u) = \int_T^{T_f} \|y(t) - y_d(T; \nu)\|^2 dt + \rho \int_0^T \|u(t)\|^2 dt \ \rightarrow \ \min, \tag{7}$$

where we assume that the output y of the plant (1), (2) can be monitored on the time interval $[T, T_f]$, and $T_f > T$. The first term in (7) gives a measure of the overshoot, the second term, of the control effort. We assume that on the interval $T \leq t \leq T_f$ the desired output y_d is a constant defined by the vector ν, and $u(t) \equiv 0$, which corresponds to the system being in the desired final steady state. Unlike (6), the performance index (7) penalizes only overshoot of the system output *after* the end of the desired motion. Under appropriate conditions of controllability and observability, a solution $u(\cdot)$ of (7) approaches quadratic-optimal terminal control as $\rho \rightarrow 0$. For linear systems, this is studied in more detail in [23] and, generally, a smooth nonlinear system can be linearized in the vicinity of the optimal solution to make the linear system results applicable.

Unlike most other work on learning control, we will be interested in obtaining a *dependence* $u(\cdot; p)$ of the feedforward control (1) on the parameter vector (5), rather than in learning the feedforward control for a *single given value* of the parameter vector p.

The mapping $p \mapsto u(\cdot)$ defined by the optimal control problem (1)–(3), (4), and (5) is generally complicated and nonlinear, even for a linear system (1), (2). We suppose that the system(1)–(2) and the performance index (4) are such that this mapping is continuous. For instance, it is continuous, if the right-hand side mappings in (1)–(2) are smooth, the system is reachable, $\rho > 0$, and we are considering a quadratic performance index (6) or (7). Yet, we would like to note that study of the properties of the mapping from the coordinates into control for a general nonlinear system is a complicated problem. In particular, it is known that for certain systems with smooth right-hand sides, such as nonholonomic systems, no continuous stabilizing state feedback exists [9]. Some discussion regarding approximation of the feedforward shape dependence on the task parameter vector for a nonholonomic system can be found in [34].

2.2 Example: Control of two-link flexible arm

Let us consider a two-link articulated arm shown in Figure 1. We assume that inertial drives placed in the arm joints are connected to the links through lumped elastic elements and all motion is in a horizontal plane. Our goal is to demonstrate an application of the approach of this paper to a standard problem. Therefore, we employ commonly made assumptions about the elastic-joint manipulator dynamics [65]. In particular, we assume that the damping in

the elastic elements is negligible, and that angular motion of the drive rotors is decoupled from the arm structure motion. The latter assumption holds for the drives with high transmission ratios.

Let $q \in \Re^2$ be the vector of the arm joint angles; $\gamma \in \Re^2$, the vector of the rotation angles for the output shaft of the drive; and $\mu \in \Re^1$, a lumped mass (payload) attached to the arm tip. Under the assumptions made, the equations of motion of the system have the form

$$M(q)\ddot{q} + C(q,\dot{q}) + K(q-\gamma) = 0, \tag{8}$$

$$J\ddot{\gamma} + B\dot{\gamma} + K(\gamma - q) = \tau, \tag{9}$$

where $M(q) \in \Re^{2,2}$ is the inertia matrix of the arm; $C(q,\dot{q}) \in \Re^2$ is the vector of Coriolis and centrifugal forces; the matrix $K = \mathrm{diag}\{k_1, k_2\} \in \Re^{2,2}$ defines the elastic element stiffnesses k_1 and k_2; $J = \mathrm{diag}\{j_1, j_2\} \in \Re^{2,2}$ is the diagonal matrix of the drive rotor inertias; $B = \mathrm{diag}\{\beta_1, \beta_2\} \in \Re^{2,2}$ is the matrix of the internal drive damping; and $\tau \in \Re^2$ is the drive torque vector, which we consider as a control variable. The exact form of the nonlinear functions $M(q)$ and $C(q,\dot{q})$ for a planar arm with uniform links and a lumped payload can be found elsewhere, e.g., in [14].

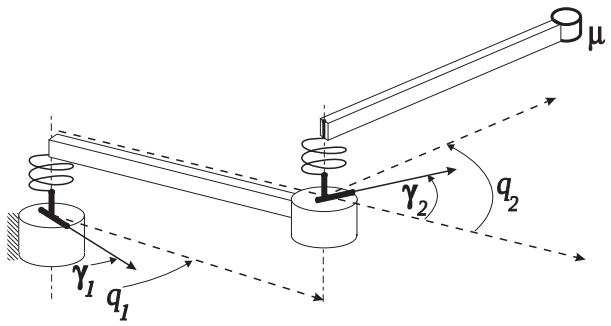

Figure 1: Two-link planar arm with flexible joints

We further assume that the control torque vector τ applied to the drives is computed in the usual way, as a sum of a proportional and a derivative (PD) drive position feedback and feedforward compensation $u(t)$

$$\tau(t) = K_*(q_d(t) - \gamma(t)) + B_*(\dot{q}_d(t) - \dot{\gamma}(t)) + u(t), \tag{10}$$

where $q_d(t)$ is the reference drive position, $K_* = \mathrm{diag}\{k_{*1}, k_{*2}\} \in \Re^{2,2}$ is the matrix of the proportional drive position feedback gains; and $B_* = \mathrm{diag}\{\beta_{*1}, \beta_{*2}\} \in \Re^{2,2}$ is the matrix of the velocity feedback gains.

We consider the feedback gains K_* and B_* as fixed parameters and the vector of the feedforward joint torques $u(t) \in \Re^2$ as an external (control) input to the system. We assume that the observations used in the learning include the drive rotation angles q and the elastic element deformations $q - \gamma$. We suppose that the drive velocities $\dot{\gamma}$ are available for the low-level servocontrol (10), but not to a higher level controller implementing the learning algorithm. Such situation is very common in robot control practice, where servocontrollers of the drives usually do not provide the upper level of the control system with the velocity information.

For the system (8)–(10), the state vector has the form $x = [q^T \ \dot{q}^T \ \gamma^T \ \dot{\gamma}^T]^T \in \Re^8$ and the observation vector, $y = [q^T \ (q-\gamma)^T]^T \in \Re^4$. The system (8)–(10) is a special case of the system (1), (2).

Let us introduce a vector $\lambda \in \Re^2$ of the initial condition parameters, so that the initial state vector (3) has the form

$$x(0) = \left[\lambda^T \ 0 \ 0 \ \lambda^T \ 0 \ 0\right]^T, \tag{11}$$

where $\lambda \in \Re^2$ defines the initial joint angles of the arm and the drive angles. The initial condition (11) means that $q(0) = \gamma(0) = \lambda$ and $\dot{q}(0) = \dot{\gamma}(0) = [0 \ 0]^T$.

Let us consider the control goal parameter vector $\nu \in \Re^2$ that is equal to the desired joint angle vector $q_d(T)$ after the motion of the arm. The preplanned output $y_d(\cdot;\lambda,\nu)$ describes the desired path of the arm motion, which is used as a reference in the PD controller (10) and the planned joint deformations. The path planning method commonly used in Robotics is to compute the reference path as a straight line in the joint angle space that can be written in the form

$$q_d(t) = \lambda(1 - s(t)) + \nu s(t), \tag{12}$$

where $s(t)$ is a smooth scalar-valued function (e.g., a third order polynomial) such that $s(0) = s'(0) = s'(T) = 0$, and $s(t) \equiv 1$ for $t \geq T$. We assume that the planned values of the joint deformations $q_d(t) - \gamma_d(t)$ and their derivatives are always zero.

Let us consider a problem of point-to-point control of the arm stated as a so-called "cheap control problem", e.g., the problem of minimizing the performance index of the form (7), where $0 < \rho \ll 1$. For $t > T$, we assume that the feedforward is zero and the PD controller uses the constant set point $y_d(T)$. As mentioned in Subsection 2.1., a solution of the control problem (7) approaches a quadratic optimal solution of the terminal control problem as $\rho \to 0$.

2.3 Discretized problem

Subsection 2.1 states the problem of approximating the optimal feedforward control u over a domain of the parameter vector p. This approximation problem involves a mapping from a domain $\mathcal{P} \subset \Re^{N_p}$ of the parameter vector p (5) into the Banach space $\mathcal{L}_2(\Re^{n_u}; 0, T)$, where the feedforward control input $u(\cdot)$ belongs. An implementable computational algorithm that solves the stated problem has to use a truncation of Banach space vectors such as $u(\cdot)$. Furthermore, a control algorithm implemented with a digital computer introduces the truncation in the form of input and output signal sampling. The problem treatment based on the truncation is acceptable, since in practice only a restricted approximation accuracy is required. After the truncation, the time-histories of the feedforward input $u(\cdot)$ to the system (1), and the system output $y(\cdot)$ (2) will be represented by finite-dimensional vectors, and the approximation problem will include mappings between finite-dimensional vector spaces.

In order to truncate the formulated Banach-space problem, let us introduce a set of the shape functions $\phi_j(\cdot) : [0, T] \mapsto \Re$, $(j = 1, \ldots, N)$. The shape functions $\phi_j(\cdot) \in \mathcal{L}_2(\Re; 0, T)$ make a basis of the linear manifold $\Pi_N \subset \mathcal{L}_2(\Re; 0, T)$. By using the Galerkin-Rietz approach to solution of the problem (1)–(3), we consider a projection of the control $u(\cdot) \in \mathcal{L}_2(\Re^{n_u}; 0, T)$ onto the linear manifold $\Re^{n_u} \bigotimes \Pi_N$, where $\bigotimes$ denotes a direct product of two vector spaces. This projection, also known as an assumed mode expansion, has the form

$$u(\cdot) = \sum_{j=1}^{N} U_j \phi_j(\cdot); \quad U = \begin{bmatrix} U_1 \\ \vdots \\ U_N \end{bmatrix} \in \Re^{N_u}, \tag{13}$$

where we have collected the weight vectors U_j into the vector U of the dimension $N_U = n_u N$. The vector U (13) is a coordinate vector on the linear space $\Re^{n_u} \bigotimes \Pi_N$. It is well known that for many choices of the shape functions - such as a trigonometric or polynomial Fourier series, B-spline approximations, or wavelet expansions - the Galerkin-Rietz method converges to the exact solution of the continuous-time problem, as $N \to \infty$.

We do not discuss a particular choice of the shape function set or the expansion order here, since this is a well-studied problem addressed elsewere. We assume known that the expansion of the form (13) gives an acceptable solution of the problem. An additional insight can be obtained from the papers [17, 23, 71, 63], among many others, where the applications of expansions of the form (13) to related continuous-time control problems are discussed in more detail.

In many applications it is advantageous to use B-splines in the expansion (13). The B-splines shape functions with the same bases (support) differ only by translations. It is possible to use B-splines of various orders in (13). In particular, zero-order B-splines will give a piece-wise constant feedforward (13), while cubic B-splines will result in a twice continuously differentiable feedforward. In the application example of Section 6, first order B-spline function are used.

We further assume that the shape function set is given and vector U defines the feedforward control on the interval $[0, T]$, i.e., for the task in question. We will call U a *control input vector* or a *control program*.

Similarly, we describe the system output with a finite-dimensional output vector Y. We introduce a sampling time sequence $\{t_j\}_{j=1}^{L}$ and *output vector* Y and a desired output vector Y_d that have the form

$$Y = \begin{bmatrix} y(t_1) \\ \vdots \\ y(t_L) \end{bmatrix} \in \Re^{N_Y}, \qquad Y_d = \begin{bmatrix} y_d(t_1) \\ \vdots \\ y_d(t_L) \end{bmatrix} \in \Re^{N_Y}, \tag{14}$$

where $y \in \Re^{n_y}$ is the system output (2), $y_d(t) = y_d(t; \lambda, \nu)$ is the desired output as in (6) and (7), and $N_Y = n_y L$. Under appropriate observability conditions imposed on the sampling time sequence and on the controlled plant, the desired output of the system (1), (2) can be evaluated by monitoring only the sampled output (14). Due to the measurement sampling in a digital computer, the measured output has the form (14) in most practical cases.

The input vector U defines the output vector Y in accordance with equations (1)–(2), (5), (13), and (14). We will write this dependence in the form

$$Y = S(U, p) \tag{15}$$

and assume that the system (1) is such that S is a smooth mapping. Let us also consider a modified form of the performance index (4)

$$J(Y, Y_d; U; p) \rightarrow \min, \tag{16}$$

where Y_d is obtained by sampling the preplanned output $y_d(t; \lambda, \nu)$ in the same ways as $y(t)$ in (14). The vector p in (15) is presented explicitly in order to show the dependence of the control problem on the task parameters λ, μ, and ν that influence the output according to (1), (2), (3), (13), (14), and (16). For a broad class of controlled systems, it is shown in [23] that, under appropriate conditions, a solution of the discretized problem as introduced in this subsection approaches the solution of the original continuous-time problem. Because of space limitations, we do not discuss this issue in more detail here and further will limit the consideration with the discretized problem (14), (15).

For the performance index (16), one can write the condition of the extremum in the form

$$\frac{\partial J}{\partial U}(U_*, p) + G^T(U_*, p)\frac{\partial J}{\partial Y}(U_*, p) = 0, \qquad G(U, p) = \frac{\partial Y}{\partial U} = \frac{\partial S}{\partial Y}(U, p), \tag{17}$$

where $G \in \Re^{N_Y, N_U}$ is an input/output sensitivity matrix of the system - a Jacobian matrix of the mapping (15). If the mapping (15) is known analytically, one can use (17) to find analytically or numerically an optimal control input U_* for each value of p. For the performance index of the form (6) or (7), we can write the modified performance index (16) in the form

$$J = \|Y - Y_d\|^2 + \rho\|U\|_R^2 \rightarrow \min, \tag{18}$$

where $\|\cdot\|$ denotes the Euclidean norm in $\Re^{N_Y}$ and $\|U\|_R^2 = U^T R U$, where $R \in \Re^{N_U, N_U}$ is a positive definite matrix defined through a Grammian of the shape function set $\{\phi_j(\cdot)\}_{j=1}^N$. Performance index (18) can approximate either (7) or (6), assuming that the sampling of the output $y(t)$ (14) is uniform on the interval $[T, T_f]$, or $[0, T]$ respectively. To make the presentation more transparent, we further assume that in (18) $R = I_{N_U}$ is a unity matrix and both norms in (18) are the regular Euclidean norms. In particular, this is valid if the shape functions (13) are orthonormal. In general case, the orthonormality of the shape functions can be achieved with a simple linear transformation.

For problem (18), the extremum condition (17) has the form

$$\rho U + G^T[S(U_p) - Y_d] = 0 \tag{19}$$

and could be solved analytically or numerically, once the form of the mapping (15) is known. The solution to (19) has the form

$$U = U_*(p), \qquad p \in \mathcal{P}, \tag{20}$$

where we assume that the mapping $U_*(p)$ is smooth, which is valid for a broad class of systems (15).

2.4 Problems of task-dependent feedforward control

In what follows, we consider a few different control architectures and problems related to (15), (16). In particular we will concentrate on the performance index (16) of the form (18).

The first problem we are going to consider is to build a controller implementing the solution (20) to the problem (15), (16). The key issue here is to design a *practically implementable* controller that is able to compute the optimal feedforward vector U_* in real time with limited computational resources. This could be achieved by computing $U_*(p)$ *approximately*, rather than exactly. Such an approach is perfectly justified from a practical viewpoint because, anyway, real-life systems always differ from their computational models, however accurate the latter are.

Problem 1 Computing Task-dependent Approximation for the Feedforward.
Design a practically implementable controller computing an accurate approximation $\hat{U}(p)$ of the optimal solution $U_*(p)$ of the problem (15), (16) for any parameter vector p (5) in the given domain $\mathcal{P}$.

The design of an approximation-based controller solving Problem 1 is discussed in Subsection 3.4. This design employs a Radial Basis Function network for approximating the mapping $U_*(p)$.

As will be discussed further in more detail, Problem 1 can be solved by first obtaining optimal control U_* for selected task parameters p and then interpolating this data. As an alternative to numerical model-based optimization, an optimal control vector U_* for a fixed given task parameter vector p can be directly learned in the course of repeated execution of the same task. Though repeated experiments are only possible in some problems, the learning problem stated below is of didactic importance to us. The RBF network learning problems considered further can be seen as generalizations of this problem.

Problem 2 Iterative Learning of Feedforward for a Given Task.
Let us assume that p in (15) is given and fixed. We assume that the mapping (15) is unknown, but we can repeatedly execute the same task defined by (1), (2), (3), (5), (13), (14). For each task repetition, an input vector U is applied and the output vector Y is observed. The problem is to design a learning control algorithm that iteratively updates the input vector U in order to optimize the performance index (18).

A learning procedure solving Problem 2 is studied in Section 4 of this paper.

For many real problems, designing a feedforward controller by interpolating solutions of Problem 2 is not practical, since this would require multiple repetitions of the same task. A more practical controller can be obtained by first using an available model of the system for off-line design and then updating the feedforward control approximation based on output errors registered as it operates. Such an approach follows usual practice of the general feedback controller design where setpoints can be calculated off-line or on an upper level of control and then tracked using an error feedback. The difficulty in updating the control approximation $\hat{U}(p)$ is that, unlike the considered Problem 2, in the course of normal operation of the system, the sequence of the task vectors p is defined by a higher control level. Thus, it is unreasonable to make any assumptions about this sequence when designing the controller. The formal statement of the discussed problem is as follows.

Problem 3 On-line Learning Update in Task-dependent Feedforward Approximation.
Assume that an imprecise approximation $\hat{U}(p)$ of the control input optimal in the sense (16) is available for each value of p. Assume further that an approximation $\hat{G}(p)$ of the system output Y sensitivity (16) to input U is available for each value of p. For a generic sequence of the parameter vectors p (5), and a corresponding sequence of the optimal control input vectors $\hat{U}(p)$ (1), (13) applied to the system (1)–(2), (15) observe the sequence of the output vectors Y (2), (14) and update an approximation $\hat{U}(p)$ of the control input to optimize it in the sense (16) for each value of p.

Problem 3 is studied in Section 5.

As mentioned above, Problem 3 is essentially about a discrete-time feedback controller design. In some cases such model-based controller may not perform well enough in practice, in particular, if an available model of the system sensitivity (gain) is not sufficiently accurate. In such cases, one may want to use an *adaptive* or self-tuning controller, which estimates the system gain based on the closed-loop operation data (possibly with self-excitation added). The problem of designing such an adaptive controller is as follows.

Problem 4 Adaptive Learning of Task-dependent Approximation for Feedforward.
Given a sequence of the parameter vectors p (5), apply a sequence of the control input vectors U (1), (13) to the system (1)–(2), (15) and by observing the sequence of the output vectors Y (2), (14) estimate for each p a local model of the input–output mapping $U \mapsto Y$ including the system sensitivity. Using this model, update an approximation $U_*(p)$ of the optimal control input (20).

Section 6 of this paper presents an iterative adaptive learning procedure that updates an approximation of the mapping $p \mapsto U_*(p)$ for a priori unknown mapping (15), as required in Problem 4.

3 Radial Basis Function approximation

This section considers a generic approximation problem, and introduces a RBF network architecture suitable for solving such problems. We then show how a controller using a RBF network approximation can be used to solve Problem 1 stated in Subsection 2.4. The material of this section us used as a base for developing algorithms in the subsequent sections.

3.1 Exact RBF interpolation

Let us consider an auxiliary problem of approximating a smooth nonlinear mapping $G(\cdot)$: $\Re^{N_p} \mapsto \Re^{N_y}$ over a compact domain

$$Y = g(p); \quad Y \in \Re^{N_y}; \ p \in \mathcal{P} \subset \Re^{N_p}, \tag{21}$$

where p is an input parameter vector, $\mathcal{P}$ is a compact domain, and Y is an output vector. We assume that a scattered (irregularly placed) set of N_t input/output pairs is available and call this set *the training data set*

$$\{Y^{(j)} = g(p^{(j)}),\ p^{(j)}\}, \quad (j = 1, \ldots, N_t) \tag{22}$$

The problem is to find an approximation $\hat{Y} = \hat{g}(p)$ of the mapping (21) that can be used for any argument $p \in \mathcal{P}$.

A computationally convenient way of representing an unknown nonlinear function is to present it as an expansion. The expansion is linear in parameters that are assumed to be unknown. Let us consider an approximation of the mapping (21) that has the form

$$\hat{Y} = \hat{g}(p) = \sum_{j=1}^{N_a} Z^{(j)} w_j(p), \tag{23}$$

where N_a is the order of the expansion, $w_j(p)$ are scalar expansion shape functions, and $Z^{(j)} \in \Re^{N_y}$ are the expansion weights. The truncated Fourier series expansion, polynomial expansion and B–spline expansion all have the form (23). In the ANN literature approximations of the form (23) are known as Functional Link Networks [11, 55] and we further call vectors $Z^{(j)}$ the network weight vectors.

Given the expansion shape functions $w_j(p)$ (23), a standard way to solve the Scattered Data Approximation problem is to choose the parameter vectors $Z^{(j)}$ by fitting the expansion (23) to the data (22) with a least error. In the special case $N_a = N_t$, when the number of the expansion weight vectors (23) coincides with the number of the training set data pairs (22), one can generally fit the training data exactly.

In this paper we consider an expansion of the form (23) with functions $w_j(\cdot)$ that depend on the radii $r_j = \|Q^{(j)} - p\|$, where $Q^{(j)} \in \Re^{N_p}$ are given vectors. Such expansion is known under the name of Radial Basis Function (RBF) approximation. RBF approximation has been used in computer graphics and experimental data processing applications (e.g., geophysical data) for two decades and has been demonstrated to provide for a high quality of approximation. One can find further details and references in [16, 18, 19, 45, 60, 61] The cited papers employ the method recently referred to as an *exact RBF interpolation*. This method uses the radial functions centered at each of the data points (22). In this method, the radial function centers are $Q^{(j)} = p^{(j)}$, and the expansion functions in (23) have the form

$$w_j(p) = h(p - p^{(j)}), \tag{24}$$

where $h(\cdot)$ is a radial function, i.e., $h(p - p^{(j)})$ depends on the radius $\|p - p^{(j)}\|$. Some most commonly used radial basis functions are

$$\begin{aligned} h(p) &= \exp(-\|p\|^2/d^2); \\ h(p) &= (1 + \|p\|^2/d^2)^{1/2}; \\ h(p) &= (1 + \|p\|^2/d^2)^{-1/2}, \end{aligned} \tag{25}$$

where $\|\cdot\|$ denotes the Euclidean vector norm. The first radial function in (25) is Gaussian, and the last two are called Hardy Multiquadrics and Reverse Multiquadrics respectively [45]. Usually, the radial function width parameter d in (25) is chosen to be about an average distance between the neighboring node centers [7, 16, 18, 61].

Let us introduce the data matrix $\boldsymbol{Y}$ and the parameter matrix θ built of vectors (22) and (23)

$$\begin{aligned} \boldsymbol{Y} &= [Y^{(1)}, \ldots, Y^{(N_t)}] \in \Re^{N_y, N_t}; \\ \theta &= [Z^{(1)}, \ldots, Z^{(N_a)}] \in \Re^{N_y, N_a} \end{aligned} \tag{26}$$

In the exact RBF interpolation we have $N_a = N_t$. By substituting (23) and (24) into (22), and using (26), we can write the condition of the exact fit for the training data $\hat{Y}(p^{(j)}) = Y^{(j)}$ in the matrix form

$$\boldsymbol{Y} = \theta H, \quad H = \{h(p^{(i)} - p^{(j)})\}_{i,j=1}^{N_t} \in \Re^{N_t, N_t}, \tag{27}$$

The symmetrical matrix H in (27) is called *interpolation matrix*. This matrix has been proved to be invertible for the commonly used radial functions, if the vectors $p^{(j)}$ are distinct [49]. With (23), (26), and (27) we obtain the interpolation of the mapping (21) of the form

$$\hat{Y} = \hat{f}(p) = \boldsymbol{Y} H^{-1} \bar{h}(p), \;\; \bar{h}(p) = \mathrm{col}(\{h(p - p^{(j)})\}_{j=1}^{N_t}) \in \Re^{N_t} \tag{28}$$

It has recently been acknowledged that RBF interpolation minimizes a certain regularization performance index that describes the interpolated surface roughness [16, 58, 60]. Different forms of radial functions (25) correspond to minimization of different regularization indexes.

Note that the approximation (28) is *linear* in the data vectors $Y^{(j)}$ (26). Thus, the computational complexity of the method remains moderate even for a large dimension N_y of the vector Y.

The exact RBF interpolation (28) is *global* in the sense that it has the same form for any $p \in \mathcal{P}$. One needs to complete the most computationally expensive part of (28) - inversion of matrix H (27) - only once for any number of points, where the approximation (28) of the function (21) is to be computed. Yet, this advantage cannot be used if the training set (22) grows with time.

3.2 RBF network approximation

Recently, some authors have treated the RBF approximation in the connectionist network setting [5, 40, 47, 50, 56, 57, 58]. They consider the radial function centers $Q^{(j)}$ (we will call $Q^{(j)}$ the network node centers) that do not coincide with the training set points, so that the expansion functions in (23) have the form

$$w_j(p) = h(p - Q^{(j)}), \quad (j = 1, \ldots, N_a) \tag{29}$$

Suppose that the node centers $Q^{(j)}$, $(j = 1, \ldots N_a)$ are given and fixed vectors. Let us fit the training set data (22) using the network (23). Employing the same notation (26) as in (27), we can represent the fitting problem in the regression form

$$\boldsymbol{Y} = \theta\boldsymbol{\Phi} + \mathcal{E}, \quad \boldsymbol{\Phi} = \{h(p^{(k)} - Q^{(j)})\}_{j,k=1}^{N_a,N_t} \in \Re^{N_a,N_t}, \tag{30}$$

where $\mathcal{E} = [e^{(1)}, \ldots, e^{(N_t)}] \in \Re^{N_y,N_t}$ is a residual error matrix. Since we cannot be sure that $\boldsymbol{\Phi}$ is a well conditioned or even a full rank matrix, we will look for a regularized least square solution to (30) that minimizes

$$\|\mathcal{E}\|_F^2 + \alpha\|\theta\|_F^2 \;\rightarrow\; \min, \quad 0 < \alpha \ll 1, \tag{31}$$

where $\|\cdot\|_F$ denotes a matrix norm equal to the square root of the sum of the squared matrix entries (the Frobenius norm). In (32), α is a scalar regularization parameter, introduced to obtain solution of possibly ill-conditioned problem following the regularization technique of [67]. The parameter α is small and does not influence the solution if the problem is well conditioned. Solving (30) and (31) for θ gives

$$\theta = \boldsymbol{Y}\boldsymbol{\Phi}^T(\alpha I_{N_a} + \boldsymbol{\Phi}\boldsymbol{\Phi}^T)^{-1}, \tag{32}$$

where I_{N_a} is the $N_a \times N_a$ identity matrix.

If $N_t \gg N_a$, and the training set inputs $p^{(j)}$ are uniformly distributed in the input domain, matrix $\boldsymbol{\Phi}$ should have a rank N_a, because the basis functions (29) are linearly independent. The condition that $\boldsymbol{\Phi}$ has the full rank is called the Persistency of Excitation condition. As the exact RBF interpolation is known to yield very accurate results, one can expect that an RBF network with fixed centers can provide good approximation accuracy. More discussion of the properties of the RBF network with nodes placed on a uniform grid can be found in [64]. The idea discussed in [64] and theoretically studied in more detail in [61] is that an RBF interpolation on a uniform grid performs a *spatial* filtering of the approximated function. Thus, the RBF approximation error is small, if the function has small high-frequency contents.

3.3 Recursive identification of the RBF model

Equation (32) describes the computation of the network parameter matrix θ with the method that is called *batch learning* in the ANN literature. This method assumes that the whole training data set (22) is available at once. In many practical situations, however, the training data pairs arrive one by one, and a recursive weight updating procedure is desirable. The recursive weight update enables the user not to keep track of all upcoming data, but rather modify the parameter matrix θ as the new data arrive. This feature is especially important for RBF network-based nonlinear adaptive control applications, such as these considered further in this paper.

We can apply a well-known recursive least square estimation methods to update an already available estimate of the matrix θ (26). Let us introduce a regressor vector for the expansion (23), (29)

$$\Phi(p) = [h(p - Q^{(1)}) \;\ldots\; h(p - Q^{(N_a)})]^T \tag{33}$$

and denote $\Phi^{(k)} = \Phi(p^{(k)})$. Note that the vectors $\Phi^{(k)}$ are the columns of the regression matrix $\boldsymbol{\Phi}$ (30). The RBF approximation (28), (29) can be presented in the form

$$\hat{Y} = \hat{g}(p, \theta) = \theta\Phi(p); \qquad \theta = [Z^{(1)} \;\ldots\; Z^{(N_a)}]^T, \tag{34}$$

where $\theta \in \Re^{N_Y,N_a}$ is an RBF network parameter matrix.

A recursive estimation technique commonly used in signal processing and adaptive control is projection estimation. The projection estimation is a special case of the Least Mean Square algorithm, which is known as the Widrow-Hoff updating rule in the signal processing literature and as a delta rule in the ANN literature. To derive the projection update, instead of minimizing a mean error index (31), let us minimize a one-step error increment index similar to (31). Let $\hat{\theta}^{(k)}$ be an estimate of θ available at step k and $e^{(k)}$ be the k–th step approximation error

$$\begin{aligned} \|e^{(k)}\|^2 + \alpha\|\hat{\theta}^{(k+1)} - \hat{\theta}^{(k)}\|_F^2 \;\rightarrow\; \min, \\ e^{(k)} = Y^{(k)} - \hat{\theta}^{(k)}\Phi^{(k)}, \end{aligned} \tag{35}$$

The solution of (35) for $\hat{\theta}^{(k)}$ has the form similar to (32)

$$\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)} + a^{(k)} e^{(k)} {\Phi^{(k)}}^T / (\alpha + \|\Phi^{(k)}\|^2), \tag{36}$$

where $\|\Phi^{(k)}\|^2 = {\Phi^{(k)}}^T \Phi^{(k)}$ and the deadzone parameter $a^{(k)}$ is 1. In the presence of the approximation error, $a^{(k)}$ should be chosen zero inside a deadzone to in the usual way to ensure robust convergence of the algorithm despite the approximation error and, possibly, measurement noise, e.g., see [21] for more detail.

Let us discuss the algorithm convergence issue. We assume that the RBF network approximation (34) of the mapping (21) can be made accurate enough by appropriate choice of the network weight matrix θ, i.e., for some $\theta = \theta_*$

$$\|g(p) - \hat{g}(p, \theta_*)\| \leq \delta_Y, \tag{37}$$

where δ_Y is a "sufficiently" small approximation error. The convergence of the recursive estimation algorithm (36) in the presence of the approximation error and, possibly, measurement noise can be guaranteed according to the standard results of adaptive control and parameter estimation theory [21, Sect. 3.6, pp. 88–91]. To ensure the convergence, the deadzone parameter sequence $a^{(k)}$ should be chosen as

$$a^{(k)} = \begin{cases} 0, & \text{if } \|e^{(k)}\| \leq \delta_Y \\ 1, & \text{otherwise} \end{cases} \tag{38}$$

Projection estimation of RBF network weights in adaptive control of a nonlinear system is considered, for instance, in [51]. The papers [12, 11] consider an application of the Orthogonal Least Square modification of the RLS algorithm to RBF network approximation. The papers [12, 44] consider modifications of the RLS identification of an RBF network for the cases of dynamical node creation, update and clustering of the node centers. In this paper, we consider the numbers of the nodes and the node centers as fixed parameters.

The well-known condition for the estimation algorithm (36) to converge to the correct parameter matrix θ_* is *persistency of excitation* condition [21]. The persistency of excitation requirement is that for $N_p \geq 1$, $\delta > 0$ exists such that for any n

$$\underline{\sigma} \left(\sum_{k=n+1}^{n+N_p} \Phi^{(k)} {\Phi^{(k)}}^T \right) > \delta, \tag{39}$$

where $\underline{\sigma}(A)$ denotes a minimal singular value of the matrix A. According to (33), the persistency of excitation depends on the sequence of training inputs $p^{(k)}$. In [28], it is proved that in the RBF network identification, persistency of excitation is provided if the inputs $p^{(k)}$ are in certain neighborhoods of the network node centers $Q^{(k)}$.

It is a well established fact that for the projection update the prediction errors $e^{(k)}$ always converge into the $2\delta_Y$ deadzone [21]. The convergence result is valid for any regressor vector sequence, i.e., for an arbitrary task parameter vector sequence $\{p^{(k)}\}$.

3.4 RBF approximation of task-dependent feedforward

Let us return to Problem 1 as introduced in Section 2.4. It is possible to design a task dependent approximation $\hat{U}(p)$ of the optimal feedforward vector on the task parameter vector p (5) by using the RBF network approximation technique discussed in the beginning of this section.

Let us assume that the optimal feedforward vectors $U_{*,k} = U_*(Q^{(k)})$ are known for certain (discrete) values $Q^{(k)}$ of the task parameter vectors p. Given N_a pairs of vectors $Q^{(k)} \in \mathcal{P}$ and $U_{*,k}$ as $\{ Q^{(k)},\ U_{*,k} = U_*(Q^{(k)}) \}_{k=1}^{N_a}$, we can approximate the mapping $U_*(p)$ over the given domain $\mathcal{P}$ of the task parameter vector p by using an exact RBF interpolation as discussed in Subsection 3.1. This approximation can be also represented by an RBF network of the form

$$\hat{U}(p) = K\Phi(p), \tag{40}$$

where $\Phi(p) \in \Re^{N_a}$ is the RBF regressor vector (33) and $K \in \Re^{N_U, N_a}$ is the RBF network weight matrix.

It follows from (23),(24) and (28), (40) that the matrix K can be found from exact RBF interpolation conditions as

$$K = [U_{*,1} \ldots U_{*,k}] H^{-1}, \tag{41}$$

where $H \in \Re^{N_a, N_a}$ is the RBF interpolation matrix of the form (27)

$$H = \left\{ h(Q^{(i)} - Q^{(j)}) \right\}_{i,j=1}^{N_a} \tag{42}$$

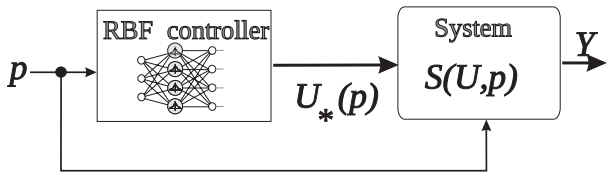

Figure 2: Schematics of feedforward controller using RBF approximation of dependence on task parameter vector

The RBF network task-level controller defined by (40) is schematically shown in Figure 2. A sequence of tasks, each defined by a task parameter vector p is generated externally with respect to the controller. The vector p is supplied to the input of the system mapping $Y = S(U, p)$ (15) and to the input of the controller, which computes the feedforward U by using the RBF network approximation. The task parameter vector p changes in discrete time (from task to task). This change cannot be predicted by the controller and can be considered as an external disturbance acting on the task completion process. The designed RBF controller provides a feedforward compensation for this (measurable) disturbance.

Design of the proposed RBF network controller is straightforward. We assume that the dependence (20) of the optimal feeforward vector on the task parameters is a smooth mapping. Such mapping can be approximated by an RBF network to arbitrary accuracy, provided the grid of the RBF nodes $Q^{(k)}$ is dense enough. Thus, for $K = K_*$

$$\|K_*\Phi(p) - U_*(p)\| \leq \delta_U, \tag{43}$$

where, as mentioned above, δ_U can be made as small as needed by choosing a denser grid of the RBF node centers. The RBF network approximation weights K_* can be computed according to (41) or by another method.

The optimal feedforward vectors $U_{*,k}$ for the RBF interpolation (41) can be obtained in different ways. The vectors $U_{*,k}$ can be computed in the course of numerical optimization by using a detailed numerical model of the system. An advantage of an RBF network approximation in this case is that it can be used for a fast computation in real time, while the computationally expensive numerical optimization is done off-line. An application of this approach in control of car parking is considered in [34], in control of free-flying space robot in [35], and in 3-D slewing maneuvers of a flexible spacecraft system in [37].

Alternatively, the optimal feedforward vectors can be learned in the course of iterative repetitive experiments with the system, where the task parameter vector p repeatedly takes the respective value $Q^{(k)}$. This approach, discussed in the next section, does not require detailed knowledge of the system dynamics. An experimental application of such approach in control of fast motions for a direct-drive robot is studied in [36].

4 Learning feedforward for a given task

This section is devoted to the problem of learning a single optimal shape vector U_*. We assume that the task parameter vector p is fixed. Therefore, we will not write explicitly dependencies on p unless this is needed to avoid ambiguity. The section has a didactic purpose and exposes some background ideas of learning control that are further elaborated in more comprehensive approaches of Sections 5 and 6.

4.1 Learning control as on-line optimization

Let us assume that the parameter vector p is fixed. In order to achieve the control goal, a feedforward shape vector U_* has to be found that minimizes the performance index (18). Let us first assume that an estimate $\hat{G}$ for the Jacobian matrix $G = \frac{\partial S}{\partial U}$ of the mapping (15) is known for the optimal input U_*. Let $U^{(n)}$ and $Y^{(n)} = S(U^{(n)})$ be input and output vectors obtained at iteration n. By using the Levenberg–Marquardt algorithm [15], the next minimizing input guess can be computed as

$$U^{(n+1)} = U^{(n)} - \left(I_{N_u}(\rho + \mu_n) + \hat{G}^T\hat{G}\right)^{-1}\left(\rho U^{(n)} + \hat{G}^T(Y^{(n)} - Y_d)\right), \tag{44}$$

where $\mu_n \geq 0$ is a step length parameter, I_{N_U} is a $N_U \times N_U$ unity matrix and $\hat{G} = G(U^{(n)})$.

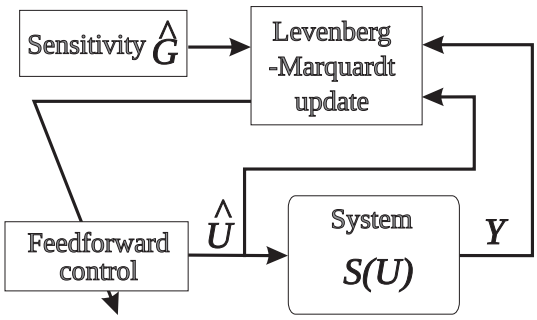

Figure 3: Schematics of learning update in a feedforward controller

The motivation for the update (44) is as follows [15]. Let us consider a local affine model of the mapping (15) of the form

$$\hat{Y} = Y^{(n)} + \hat{G}(U - U^{(n)}); \tag{45}$$

Let us also demand that the minimization step length does not exceed a given value $d_n > 0$

$$U^{(n+1)} = U^{(n)} + s^{(n)}; \quad \|s^{(n)}\| \le d_n, \tag{46}$$

By solving (18) and (45) with respect to $U^{(n+1)}$, and using the Lagrange multiplier method to satisfy the constraint (46), we arrive at (44). The Lagrange multiplier μ_n is nonnegative and can be computed once the Jacobian estimate $\hat{G}$ and the allowed step length d_n are given. With increase of μ_n in (44), all eigenvalues of the inverted matrix in (44) increase, hence, the step length $\|s^{(n)}\|$ decreases. Therefore, the dependence of μ_n on d_n is decreasing. In practice, instead of computing μ_n from d_n, usually μ_n itself is made a parameter of choice. More detail on the method can be found in [15].

If μ is small, the method approximates the Newton-Gauss method; if μ is large, it approximates the downhill (gradient) method. Each iteration in (44), presumes a repeated execution of the same given task; each time using a different feedforward input, $U^{(k)}$, and measuring the corresponding sampled output vector $Y^{(k)}$. Thus, (44) is a learning control iteration. In the commercially available software implementing the Levenberg-Marquardt method, the step limiting parameter μ in (44) is chosen at each step. This, however, requires several evaluations of the minimized function. In the learning control problems, each evaluation is a completion of the task for the controlled system and, thus, has a very large cost. Therefore, we are using the update (44) with a constant preselected parameter μ.

The schematics of the discussed learning update is shown in Figure 3. At step k, the feedforward vector $\hat{U} = U^{(k)}$ stored in the memory is applied to the system and the output $Y^{(k)} = S(U^{(k)})$ is obtained. This output us used in the Levenberg-Marquardt update (44) to compute an update for $\hat{U}$. The updated value of $\hat{U}$ is applied at the next iteration and so on. As shown in Figure 3, the update uses an estimate $\hat{G}$ for the sensitivity matrix $G(U_*)$.

The update rule (44) presumes that an estimate of the gradient(input/output sensitivity) matrix G is known and is sufficiently accurate. If unknown, this matrix can be estimated with a finite difference method. The next subsection presents a simple result showing that the update (44) has certain robustness to error in the estimate $\hat{G}$. Subsection 4.3 discusses a finite difference update for this estimate. Such an update would add an extra feedback loop for the gain $\hat{G}$ in Figure 3 and make the learning algorithm adaptive.

4.2 Robust convergence of the learning control algorithm

Analysis of the Levenberg-Marquardt algorithm convergence for a nonlinear problem can be found in [15]. This analysis assumes that the Jacobian G is known exactly at each step. Herein we consider the update (44) as a part of a discrete-time closed-loop control system. Following the established approach to analysis of such systems, let us study robust stability of the linearized loop. We assume that the system (15) is affine in U in the vicinity of the optimum. The affine model has the form

$$Y = GU + Z \tag{47}$$

In the linear-quadratic setting (18), (47), the Levenberg-Marquardt algorithm (44) converges for any positive values of the parameters ρ and μ, and it is robust with respect to the error in estimating the matrix G. A sufficient condition for the convergence is given by the following Theorem.

Theorem 1 *Let us consider the update (44) of the system (47) input. The algorithm asymptotically converges for any initial condition $U^{(1)}$, if some $k_0 \geq 1$ exists such that for any $k \geq k_0$ the maximal singular value of the gradient estimation error satisfies the following inequality*

$$\bar{\sigma}(\hat{G} - G) < \frac{2\rho}{\sqrt{\rho + \mu}} \tag{48}$$

Theorem 1 shows that the convergence robustness improves for larger values of the regularization parameter ρ and is absent if no regularization is performed. At the same time, increasing ρ increases the steady-state tracking error $\|Y - Y_d\|$ for the convergence achieved.

Proofs of Theorem 1 can be found in the papers [24, 36], which also present an analysis of static error for the learned feedforward depending on the error in the estimation of matrix G. The papers [24, 36] demonstrate experimental results in application of the learning control update of the form (44) to trajectory control of robotic arms.

4.3 Finite difference update of the gradient

Let us proceed with a situation, where the Jacobian G is not known and we can only evaluate the function mapping (15) pointwise, by executing the respective task with the given input U and observing the output Y. In such case, a possible approach is to introduce an affine model (45) of the mapping (15) and update estimates of parameters of this model from the available input/output measurements.

The most common practically used method for estimating the Jacobian G is the *Broyden secant update*. Let $\hat{G}^{(n)}$ be an estimate of the Jacobian at the step n. Denote by $s^{(n)}$ variation of input, and by $w^{(n)}$ corresponding variation of the output at the previous minimization step. For a small step length $\|s^{(n)}\|$, the updated estimate should provide fit to the observed data, i.e.,

$$Gs^{(n)} = w^{(n)}, \quad s^{(n)} = U^{(n)} - U^{(n-1)}, \quad w^{(n)} = Y^{(n)} - Y^{(n-1)} \tag{49}$$

The Broyden update rule can be considered as an application to (49) of the projection estimation algorithm [21], which is very popular in adaptive control and signal processing applications. The Broyden update is used in conjunction with the input update (44) and has the form

$$\hat{G}^{(n+1)} = \hat{G}^{(n)} + (w^{(n)} - \hat{G}^{(n)} s^{(n)}) s^{(n)T} / (c^2 + \|s^{(n)}\|^2), \tag{50}$$

where $c > 0$ is a scalar parameter used to avoid division by zero.

For a nonlinear mapping, a local convergence of the Levenberg-Marquardt algorithm with the Broyden secant update can be proved using the *bounded deterioration* technique as considered in [15]. The idea of such a proof is that in the vicinity of the optimum and for a sufficiently small initial error of approximating the gradient G, the algorithm will converge before the gradient approximation error will have time to grow due to the system nonlinearity.

Let us now consider another method for estimating the Jacobian G, which can be more appropriate if the measurements are corrupted with a noise. In the on-line learning algorithms discussed in the subsequent sections, approximation errors can be considered as such a noise. Let us write the affine model (47) for the mapping (15) in the form of a linear regression

$$\hat{Y} = \hat{G}U + \hat{Z} = \hat{\theta} W; \quad \hat{\theta} = [\hat{Z}/c \;\; \hat{G}], \;\; W = \begin{bmatrix} c \\ U \end{bmatrix}, \tag{51}$$

where c is a positive scaling constant, $\hat{\theta} \in \Re^{N_Y, N_U+1}$ is a regression parameter matrix, and W is a regressor vector. The Broyden gradient update (50) is a two-step estimation algorithm for the regression (51) that first sets $\hat{Z} = \hat{G}U^{(n-1)} - Y^{(n-1)}$ in (51) and then updates an estimate for G with the projection method. A more natural, one-step projection estimation algorithm for the model (51) has the form

$$\hat{\theta}^{(n+1)} = \hat{\theta}^{(n)} + a^{(n)} (Y^{(n)} - \hat{\theta}^{(n)} W^{(n)}) W^{(n)T} / \|W^{(n)}\|^2, \tag{52}$$

where $\hat{\theta}^{(n)} = [Z^{(n)}/c \;\; G^{(n)}]$, $W^{(k)} = [c \;\; U^{(k)T}]^T$, and $a^{(k)} \in \{0, 1\}$ is a scalar deadzone parameter. Unlike the secant update (49), (50), which uses function values obtained on *two* consecutive steps, the update (52) uses only

one function value. This makes it possible to generalize the update (52) for the optimization with changing task parameters, as shown in the next sections.

Let us write a step of the Levenberg-Marquardt algorithm for the affine model in the form (51). By solving (18) and (51) with respect to $U^{(n+1)}$, and using the Lagrange multiplier method to satisfy constraint (46), we obtain

$$U^{(n+1)} = U^{(n)} - \left(I(\rho + \mu_n) + {G^{(n)}}^T G^{(n)} \right)^{-1} \left(\rho U^{(n)} + {G^{(n)}}^T (\hat{Y}^{(n)} - Y_d) \right), \tag{53}$$

where $\hat{Y}^{(n)} = G^{(n)} U^{(n)} + Z^{(n)}$. We have $\hat{Y}^{(n)} = Y^{(n)}$ for the projection update (52), as long as $a^{(n)} = 1$. Therefore, (53) coincides with the Levenberg-Marquardt step (44). A more detailed theoretical study is presented in [31]. The experiments in applying learning control algorithms with an adaptive update of G as considered in this section to trajectory control of a direct-drive manipulator are described in [36].

5 On-line learning update in task-dependent feedforward

This section considers Problem 3 stated in Subsection 2.4. We assume that a model of the system is available for the design of the task-dependent feedforward controller. This model is, however, imprecise and the approximation $\hat{U}(p)$ of the optimal control input (1), (13), (16) is not satisfactory accurate.

This section presents an algorithm that allows to update (learn) the approximation $\hat{U}(p)$ in the course of normal system operation, i.e., assuming that the sequence $p^{(k)}$ of the task parameter vector is arbitrary. Upon completion of each task, characterized by the task vector $p^{(k)}$, the output vector $Y^{(k)}$ is used to compute an update of the controller approximation $\hat{U}(p)$ available at this step. In what follows, we propose and study such an update.

5.1 Approximating system sensitivity

Similarly to the learning algorithm considered in Section 4, the algorithm of this section updates the guess of the optimal feedforward control based on the available estimate of the system input-output sensitivity matrix G in (17). Unlike Section 4, in this section we have to consider the dependence of this sensitivity matrix on the changing vector p. Let us introduce the matrix-valued function defining the system sensitivity at the optimal feedforward input

$$G_*(p) = \left. \frac{\partial S(U,p)}{\partial U} \right|_{U=U_*(p)}, \tag{54}$$

Though the mapping (54) is not known exactly, it can be approximated based on the available system model, in the same way as the approximation $\hat{U}(p)$ for the optimal control is built in Section 3.

Let us assume that for each of the RBF approximation nodes $p = Q^{(k)}$ used for building approximation of the optimal feedforward in Section 3.4, the sensitivity matrix $G_{*,k} = G_*(Q^{(k)})$ is computed along with the optimal input vector $U_{*,k} = U_*(Q^{(k)})$. The sensitivity matrix would usually be computed as a by-product of a numerical optimization procedure (such as Levenberg-Marquardt) applied to the available system model to find $U_*(Q^{(k)})$. In the process of the numerical optimization, a matrix $G_*(Q^{(k)})$ can be obtained, for instance, by a finite difference method.

Similarly to the approximation (40) for the mapping $U_*(p)$, let us use an RBF network approximation for the mapping $G_*(p)$. Unlike a *vector-valued* mapping $U_*(p)$, the mapping $G_*(p)$ is *matrix-valued*. To facilitate work with such mappings, let us introduce the vectorization operator $\mathrm{vec}(\cdot)$. For a matrix $A \in \Re^{m,n}$, the vector $\mathrm{vec}(A) \in \Re^{mn}$ is composed of all the entries of A, column by column. An RBF network approximation $\hat{G}(p)$ for $G_*(p)$ can be presented in the form, similar to (40)

$$\mathrm{vec}(\hat{G}(p)) = \sum_{j=1}^{N_a} \mathrm{vec}(G_j) h(p - Q^{(j)}) = \Gamma \Phi(p), \qquad p \in \mathcal{P}, \tag{55}$$

where $G_j \in \Re^{N_Y, N_U}$ are the RBF network weights, and $\Phi(p)$ is the RBF regression vector (33). Note that (55) can be also represented in the form

$$\hat{G}(p) = \sum_{j=1}^{N_a} G_j h(p - Q^{(j)}) \tag{56}$$

The weights of the RBF network (55) can be computed so as to implement exact RBF interpolation of the matrices $G_*^{(k)} = G_*(Q^{(k)})$. In this case, the network node centers in (55) coincide with the data points $p = Q^{(k)}$ and the weight matrix Γ can be computed similarly to (41) as

$$\Gamma = \left[\mathrm{vec}(G_1^{(1)}) \quad \ldots \quad \mathrm{vec}(G_{(N_a)})\right] H^{-1}, \tag{57}$$

where H is the RBF interpolation matrix of the form (42).

Under general conditions, the function $G_*(p)$ has derivatives, which are uniformly bounded on $\mathcal{P}$, provided that the derivatives of $U_*(p)$ are bounded. Hence the error of the RBF network approximation (55) can be made small for high order N_a of the expansions (55).

Computing and storing the approximation $\hat{G}(p)$ of the sensitivity matrix $G_*(p)$ is similar to the usual practice of storing the system gain information along with setpoints as a part of a feedback controller design. It is particularly related to gain scheduling methods, where gain and setpoint tables are stored in the controller.

5.2 Local Levenberg-Marquardt update

In this subsection we assume that the RBF approximation $\hat{U}(p)$ (40) is not accurate enough and design an update for the feedforward, using the input-output data for the system (15). Let us assume that in the task k an input vector $U^{(k)}$ (13) was applied to the system, and an output vector $Y^{(k)}$ (14) was obtained, while the task parameter vector was $p^{(k)}$. Note that the applied input will be $U^{(k)} = \hat{U}(p^{(k)})$, where $\hat{U}(p)$ is the approximation (40) of the optimal feedforward mapping as available at this step.

Following the usual practice of iterative optimization, such as discussed in the derivation of the Levenberg-Marquardt algorithm in Subsection 4.1, let us consider an affine model of the mapping (15). A local model valid for the current task, i.e., for $p = p^{(k)}$, can be obtained by using the input/output data $U^{(k)}$, $Y^{(k)}$ and the sensitivity estimate $\hat{G}(p)$ (55). This local affine model has the form similar to (45)

$$\hat{Y}(p^{(k)}) = Y^{(k)} + \hat{G}(p^{(k)})(U - U^{(k)}) \tag{58}$$

By substituting the affine model into the performance index (18) and finding the minimum, we arrive at the optimality condition

$$\hat{G}(p^{(k)})(\hat{Y}(p^{(k)}) - Y_d) + \rho U = 0 \tag{59}$$

where $\hat{Y}(p^{(k)})$ is defined by (58).

By solving (58), (59) and limiting the update step as in (46), we obtain the Levenberg-Marquardt update similar to (44). This update gives us $U = U^{(k|k+1)}$, the *a posteriori* optimal control input for task k (task parameter vector $p = p^{(k)}$),

$$\begin{aligned} U^{(k|k+1)} &= U^{(k)} + \Delta U^{(k)}, \\ \Delta U^{(k)} &= -\left(\hat{G}^T(p^{(k)})\hat{G}(p^{(k)}) + (\rho+\mu)I\right)^{-1}\left(\hat{G}^T(p^{(k)})(Y^{(k)} - Y_d) + \rho U^{(k)}\right), \end{aligned} \tag{60}$$

where $\Delta U^{(k)}$ is the update step for the feedforward U. Note that the update (60) is calculated assuming that the task parameter vector is fixed, $p = p^{(k)}$. In fact, our goal is to calculate an update for the RBF approximation controller (40) for all p. This can be done based on the update (60) as explained in the next subsection.

5.3 Update of RBF approximation in the feedforward controller

As mentioned above, we assume that the feedforward vector U applied at the step k is computed using the task-dependent RBF approximation controller (40) shown in Figure 2. In accordance with (40), this control has the form

$$U^{(k)} = K^{(k)}\Phi(p^{(k)}) \tag{61}$$

where $K^{(k)}$ is the weight matrix of the feedforward RBF controller available at step k. Based on the *a posteriori* optimal feedforward solution (60), the weight matrix K should be modified so that the controller would yield this new optimal solution for $p = p^{(k)}$. The latter condition can be written as

$$U^{(k|k+1)} = K^{(k+1)}\Phi(p^{(k)}) \tag{62}$$

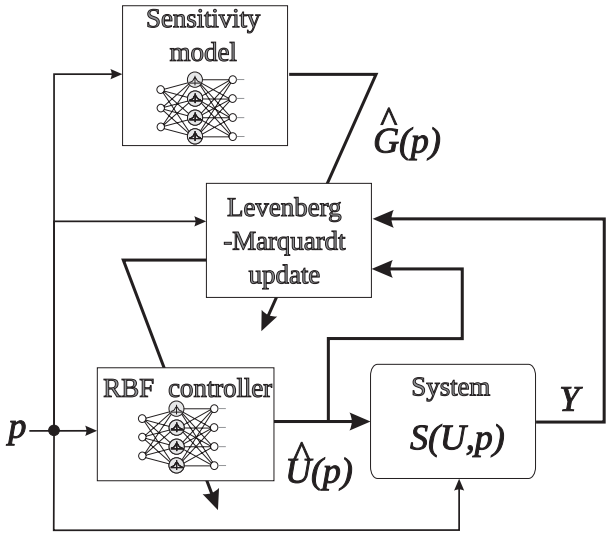

Figure 4: Schematics of learning update in a task-dependent RBF approximation feedforward controller

where $K^{(k+1)}$ is the updated RBF network weight matrix.

By using (60) and (62), we obtain a projection update for the controller (61) in the same way as (36) is obtained from (35). Modified to include a deadzone compensation for the approximation error, this update has the form

$$K^{(k+1)} = K^{(k)} - a^{(k)} \Delta U^{(k)} \Phi^T(p^{(k)}) / \|\Phi(p^{(k)})\|^2 \tag{63}$$

where $a^{(k)} = \{0, 1\}$ is a deadzone parameter and $\Delta U^{(k)}$ is defined by (60). The update (60), (63) is analogous to the projection update (36). As one can easily check by substitution, for $a^{(k)} = 1$ (62) holds exactly. For $a^{(k)} = 0$, no update is performed.

Figure 4 illustrates the proposed design of the learning controller. The task parameter vectors p are generated externally to the diagram of Figure 4 and are supplied to the controller (40) computing an RBF approximation for $U_*(p)$. The RBF approximation in (40) is updated in accordance with (60), (63) depending on the system output $U^{(k)}$. The update (60) uses the approximation for the Jacobian $\hat{G}(p)$ in (60), which is computed by the RBF network (55). The weights of the network (55) are computed off-line, e.g., according to (57).

The deadzone parameter $a^{(k)}$ in (63) should be chosen similarly to (38) in order to ensure the algorithm convergence. The deadzone must compensate for the approximation error (43) and also initial error of the approximation (61). It is possible to demonstrate deadzone convergence of the proposed algorithm and estimate the necessary deadzone by using a modification of standard convergence results in [21]. However, presenting such proof is beyond the scope of this paper. The algorithms of this section have been applied by the author in control of slewing maneuvers of a flexible spacecraft system with nonlinear rotational dynamics.

6 Adaptive learning of task-dependent feedforward

This section proposes a solution to Problem 4 stated in Subsection 2.4. The problem is to learn (update) an approximation $\hat{U}(p)$ (18) of the optimal feedforward input for an arbitrary sequence of the parameter vectors p (5) by using only input/output data. Unlike previous section, an accurate approximation for the Jacobian $G_*(p)$ dependence on the task parameters is no longer assumed to be available in advance. The available approximation of the Jacobian is assumed to contain an error and is refined on-line as a part of the learning controller we are going to develop.

In order to solve the stated problem, we introduce an affine RBF network model of the system mapping and estimate this model on-line. The algorithm discussed in this section resembles the algorithm for on-line parametric nonlinear least square optimization proposed and studied in [29, 31]. This algorithm can be viewed as a discrete-time adaptive algorithm for nonlinear system control.

6.1 Affine RBF Network Model of the System Mapping

The on-line parametric optimization algorithm of [29] we are about to derive can be considered as an extension of the Levenberg-Marquardt algorithm. Similarly to a standard derivation of the Levenberg-Marquardt algorithm given earlier, our derivation here will be based on an affine model (45) of the mapping (15). Such affine model can be written in the form

$$\hat{Y} = \hat{G}(p)U + \hat{Z}(p), \tag{64}$$

where $\hat{G}(p)$ and $\hat{Z}(p)$ are (smooth) matrix and vector functions of p. For a fixed task-parameter vector p, the model (64) is affine in U; at the same time, the model depends on p in a nonlinear way.

Let us introduce functions:

$$Y_*(p) = S(U_*(p), p), \qquad Z_*(p) = Y_*(p) - G_*(p)U_*(p) \tag{65}$$

where $G_*(p)$ is given by (54). Similarly to (56), let us use RBF networks for approximating the functions $Z_*(p)$ and $G_*(p)$. Let us assume that these mappings can be represented in the form

$$Z_*(p) = \sum_{j=1}^{N_a} Z_{*j} h(p - Q^{(j)}) \; + \; \delta Z(p), \quad \|\delta Z(p)\| \le \delta_Z, \;\; p \in \mathcal{P}, \tag{66}$$

$$G_*(p) = \sum_{j=1}^{N_a} G_{*j} h(p - Q^{(j)}) \; + \; \delta G(p), \quad \|\delta G(p)\| \le \delta_G, \;\; p \in \mathcal{P}, \tag{67}$$

where $Z_{*j} \in \Re^{N_Y}$, and $G_{*j} \in \Re^{N_Y, N_U}$ are the expansion weights. The residual errors δ_Z and δ_G can be made small for high order N_a of the expansions (66), (67), i.e., by selecting a sufficiently high density of the RBF network nodes.

We are now in position to explain the basic algorithm of [29, 31], which we are going to use for adaptive update of the controller. As in Section 5, when deriving the algorithm, we neglect the approximation errors δ_U, δ_Z, and δ_G. These errors are taken into account in the algorithm convergence analysis of [29, 31]. In the absence of the approximation error in (42), (for $\delta_U = 0$), (40) can be represented in the linear regression form

$$U_*(p) = K_* \Phi(p), \tag{68}$$

$$\tag{69}$$

Similarly to (68), we can present (66) and (67) in the form of regressions linear in the weights Z_{*j} and G_{*j}. With these regression representations (66) and (67) in mind, the model of the form (51) can be represented as the following regression:

$$\hat{Y} = \Theta \bar{\Phi}(p, U), \quad \Theta \in \Re^{N_Y, N_a(N_U+1)}, \quad \bar{\Phi}(p, U) \in \Re^{N_a(N_U+1)}, \tag{70}$$

$$\bar{\Phi}(p, U) = \Phi(p) \otimes W, \quad W = [c \; U^T]^T, \tag{71}$$

where $\otimes$ denotes the Kronecker (direct) product of matrices, and $c > 0$ is a scalar scaling parameter, $\Phi(p)$ is the regressor vector (33), $\bar{\Phi}(p, U)$ is an extended regressor vector, and Θ is the RBF network weight matrix. For a fixed parameter p, the model (70) has the form (64). By substituting $\Theta = [Z_1/c \;\; G_1 \;\; \ldots \;\; Z_{N_a}/c \;\; G_{N_a}]$ into (70), one obtains an affine model of the form (64), where

$$\hat{Z}(p) = \sum_{j=1}^{N_a} Z_j h(p - Q^{(j)}), \quad \hat{G}(p) = \sum_{j=1}^{N_a} G_j h(p - Q^{(j)}) \tag{72}$$

We assume that for $\Theta = \Theta_*$ and in the absence of the approximation errors, i.e., for $\delta y = \delta z = \delta u = 0$, the affine model(70) gives exactly the linearization of the mapping (15) in the optimum (66), (67). In other words,

$$\Theta_* = [Z_{*1}/c \;\; G_{*1} \;\; \ldots \;\; Z_{*N_a}/c \;\; G_{*N_a}] \tag{73}$$

Note that the approximation (72) for the function $G_*(p)$ (54) has the same form as the approximation (57) considered in Section 5. Unlike Section 5, where the approximation (57) was estimated using RBF interpolation of the pre-computed data on the Jacobian matrix, in this section we consider an algorithm that estimates (72) by only using values of the mapping (15), i.e., input and output vectors for each task.

6.2 Adaptive Update Algorithm

The algorithm we are going to present is an extension of the Section 5 algorithm that updates a guess of the optimal weight matrix K_* in (68). Our goal is to build an approximation of the form (68) to the optimal input mapping $U_*(p)$. We assume that a sequence of the information vectors $\{p^{(k)}\}_{k=1}^{\infty}$ is given. Let $K^{(k)}$ be the value of the input parameter matrix in (68) at the step k. Then, in accordance with (40) and similar to (61), the input vector for task k is $U^{(k)} = K^{(k)}\Phi(p^{(k)})$. Let us denote by

$$U^{(k+1|k)} = K^{(k)}\Phi(p^{(k+1)}) \tag{74}$$

the output of the controller (40), which would be obtained at step $k+1$ if the matrix $K^{(k)}$ is not updated. As in Section 5, we denote the output vector for task k by $Y^{(k)} = S(U^{(k)}, p^{(k)})$.

Let us demand that, similarly to the standard Levenberg-Marquardt method, the step of the control update should be bounded. Instead bounding step for the updated variable k, we will consider the change in U that this update brings, and bound this change. We will use condition similar to (46) but somewhat differing from it

$$U^{(k+1)} = U^{(k+1|k)} + s^{(k)}, \quad \|s^{(k)}\| \le d_k, \tag{75}$$

If in (46) we do not consider dependence on p, in (75) we do. Therefore, both $U^{(k+1|k)}$ (74) and $U^{(k+1)}$ given by (61) are computed for the same task parameter vector $p = p^{(k+1)}$) before and after the update of the RBF weight matrix $k^{(k)}$, respectively. The output of the off-line RBF model (70) at the step $k+1$ is

$$\begin{aligned} \hat{Y}^{(k+1)} &= \Theta\bar{\Phi}(p^{(k+1)}, U^{(k+1)}) \\ &= \Theta\bar{\Phi}(p^{(k+1)}, U^{(k+1|k)}) + \Theta(\Phi(p^{(k+1)}) \otimes W_s^{(k)}) \\ &\equiv \hat{Y}^{(k+1|k)} + \hat{G}(p^{(k+1)})s^{(k)}, \end{aligned} \tag{76}$$

where $W_s^{(k)} = [0 \ {s^{(k)}}^T]^T$ and $\hat{Y}^{(k+1|k)} = \Theta\bar{\Phi}(p^{(k+1)}, U^{(k+1|k)})$. By substituting $\hat{Y}^{(k+1)}$ in (76) as Y and $U^{(k+1)}$ in (74) as U into (18), and optimizing the minimization step $s^{(k)}$ subject to the constraint $\|s^{(k)}\| \le d_k$, we obtain

$$s^{(k)} = -(I_{N_U}\mu_k + \hat{D}^{(k+1)})^{-1}\left(\hat{G}^T(p^{(k+1)})(\hat{Y}^{(k+1|k)} - Y_d) + U^{(k+1|k)}\right) \tag{77}$$

$$\hat{D}^{(k+1)} = I + \hat{G}^T(p^{(k+1)})\hat{G}(p^{(k+1)}) \tag{78}$$

where μ_k is a Lagrange multiplier that is introduced to comply with the step boundedness condition $\|s_k\| \le d_k$. As for the classical Levenberg-Marquardt method explained earlier, the dependence of μ_k on d_k is monotone nonincreasing and instead of empirically choosing d_k first and computing the Lagrange multiplier μ_k based on d_k, it is advisable to make μ_k itself a parameter of choice. Recall that we update the input U indirectly, by updating the weights of the RBF network approximation for $U_*(p)$. According to (68), (74), and (75) we can write

$$s^{(k)} = (K^{(k+1)} - K^{(k)})\Phi(p^{(k+1)}) \tag{79}$$

By finding a least square solution of (79) for the RBF weight matrix update $K^{(k+1)} - K^{(k)}$ and substituting (77) for $s^{(k)}$, we obtain a step of the proposed basic parametric NLS optimization method:

$$K^{(k+1)} = K^{(k)} - (I_{N_U}\mu_k + \hat{D}^{(k+1)})^{-1}\left(\hat{G}^T(p^{(k+1)})(\hat{Y}^{(k+1|k)} - Y_d) + U^{(k+1|k)}\right)\frac{\Phi^T(p^{(k+1)})}{\|\Phi(p^{(k+1)})\|^2} \tag{80}$$

where $\hat{G}(p)$ is defined by (72); $\hat{D}^{(k)}$, by (78); $\hat{U}^{(k+1|k)}$, by (74) and (33); and $\hat{Y}^{(k+1|k)}$ is defined in accordance with (70), (74), (76) as $\hat{Y}^{(k+1|k)} = \Theta\bar{\Phi}(p^{(k)}, U^{(k+1|k)})$.

As discussed above, the affine model (64) can be written in the regression form (70). Therefore, at each step of the proposed update algorithm, we can use the projection update for an estimate of the parameter matrix Θ in (70). This update has the form

$$\hat{\Theta}^{(k+1)} = \hat{\Theta}^{(k)} + a^{(k)}(Y^{(k)} - \hat{\Theta}^{(k)}\bar{\Phi}^{(k)})\bar{\Phi}^{(k)T}/\|\bar{\Phi}^{(k)}\|^2, \tag{81}$$

where $\hat{\Theta}^{(k)}$ is the regression parameter matrix at step k and $\bar{\Phi}^{(k)} = \bar{\Phi}(p^{(k)}, U^{(k)})$ is an extended regressor vector at step k. The update (81) can be considered as a generalization of the Broyden update (50). In (81), $a^{(k)}$ is a scalar deadzone parameter that is introduced in the usual way to compensate for the influence of the mismodeling error. The deadzone parameter $a^{(k)}$ is zero if the prediction error $Y^{(k)} - \hat{\Theta}^{(k)}\bar{\Phi}^{(k)}$ is within the mismodeling bounds defined by the approximation errors δ_Y, δ_Z in (66), (67); $a^{(k)}$ is unity otherwise.

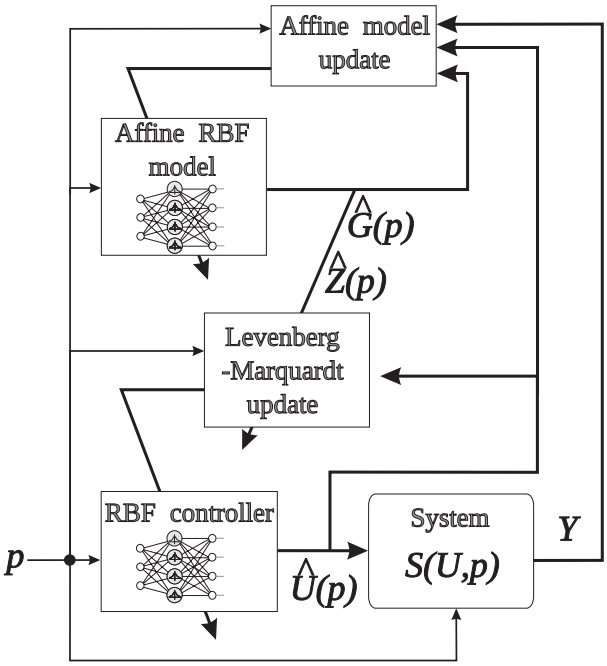

Figure 5: Schematics of adaptive learning update in a task-dependent 'RBF approximation feedforward controller

Figure 5 illustrates the designed controller. Vector p is defined externally to the control diagram of Figure 5 and acts as a disturbance for control (18). The Levenberg-Marquardt update (80) modifies the weight matrix K in the RBF network controller (18). The Levenberg-Marquardt update (80) uses estimates of the output $\hat{Y}^{(k+1|k)}$ (75) and of the Jacobian (sensitivity) matrix $\hat{G}(p)$ obtained from the affine model (70), (71). The external loop in Figure 5 updates an estimate for this affine model according to (81).

The discussion on the choice of the deadzone parameter $a^{(k)}$ in (81), as well as a proof for the algorithm convergence, are considered in [29, 31]. To ensure convergence of the estimation algorithm, a small self-excitation signal can be added to the computed control $U^{(k)}$ before it is applied to the system. This self-excitation is needed to make the regressor vector sequence in (81) persistently exciting as discussed in [28, 29].

6.3 Discussion

Equations (33), (71), (80), and (81) constitute the basic algorithm for on-line parametric NLS optimization first proposed in [29]. An analysis of the algorithm convergence is presented in [29, 31]. For a generic nonlinear mapping (15) it is only possible to prove *local* convergence of the algorithm. The locality here means that the initial approximation to the nonlinear feedforward control mapping (68), (33), should be sufficiently close to the optimum. The domain of the algorithm convergence theoretically ensured in [29, 31] depends on the degree of the system nonlinearity (second derivative bound).

The local convergence results of [29, 31] demonstrate that the algorithm of this section is consistent. It can be shown that under certain persistence of excitation conditions (such as discussed in [28]) the parameter matrix Θ of the RBF affine model (70) converges into a deadzone neighborhood of the matrix (73). At the same time, the approximation (61) of the feedforward converges into a neighborhood of the "best" approximation (43). The practical usefulness of the algorithm depends on its performance in a particular application. The next subsection considers one application of the proposed approach.

6.4 Application example: Learning control of flexible arm

This section applies the developed learning algorithm to control of fast motions in a flexible-joint arm. No a priori knowledge of the system dynamics is assumed to be available. Let us consider a planar flexible-joint arm example as introduced in Subsection 2.2, and the problem of point-to-point control for such arm. We assume that the joint torques of the arm are computed according to (10), where the desired trajectory is planned as in(12).

The control problem is to compute feedforward $u(\cdot)$ so that the arm comes to the final position $q_d(T) = \nu$ at time $T = 1.5$ without oscillations. This is a difficult problem, since the oscillation period for the lowest eigenfrequency of the system is close to unity.

We divide the motion interval $[0, T]$ into seven subintervals $[\tau_j, \tau_{j+1}]$, $(j = 0, \ldots, 7)$, $\tau_0 = 0, \tau_j = T$ and consider the feedforward input (13), $U \in \Re^{12}$, that is piecewise linear on these subintervals and zero at times 0 and T. In other words, the shape functions $\phi_j(\cdot)$ in (13) are the first-order (triangular) B-splines. We monitor the arm motion on the interval $[T, T_f]$, $T_f = T + 0.5$ at $L = 14$ uniformly spaced output sampling instants $t_1 = T, \ \ldots, \ t_{14} = T_f$. The measurement vector y comprises drive angles and joint deformations, $y = [q^T \ (q - \gamma)^T]^T \in \Re^4$. By sampling the vector y at instants t_j, we obtain the output vector $Y \in \Re^{56}$.

The task mapping (15) in this problem depends on the task parameter vector p (5) that includes the initial and desired final configurations of the arm. Since the system is cyclic, the control depends only on the *variation* of the first joint angle. Thus, we can write the task parameter vector p in the form

$$p = [q_{2d}(0) \ \ q_{2d}(T) \ \ (q_{1d}(T) - q_{1d}(0))]^T \tag{82}$$

We assume that the task parameter vector (82) remains bounded inside the following domain

$$\mathcal{P} = \{p \in \mathcal{P} : \ \frac{\pi}{4} \le q_{2d}(0), q_{2d}(T) \le \frac{3\pi}{4}; \quad -\frac{\pi}{2} \le q_{1d}(T) - q_{1d}(0) \le \frac{\pi}{2}\} \tag{83}$$

To implement the adaptive task-level learning algorithm of Section 6, we use a network of the Gaussian Radial Basis Functions with the node centers $Q^{(j)}$ placed on a uniform mesh $5 \times 3 \times 3$ in the task parameter space. When *implementing* the control algorithm, we assume the dynamical model of the system to be completely unknown and set the initial estimate of the parameter matrix Θ in (81) to be zero.

The adaptive update algorithm of Section 6 was implemented as a Matlab program on an 1.2 MFlops computer, and the arm motion simulation was coded in C. The task-level control algorithm does not exploit any initial knowledge of the controlled system dynamics available for simulation and uses just the input/output data for the control tasks. Given the input and output dimensions $U \in \Re^{12}$ and $Y \in \Re^{56}$, and the number of the RBF network nodes $N_a = 45$, the sizes of the matrices in (71) are $\bar{\Phi}(p, U) \in \Re^{585}$ and $\Theta \in \Re^{56,585}$. These sizes cause no computational problems, as the updates (80) and (81) only include matrix multiplications, and the matrix inverted in (80) has the (15) size.

For our Matlab implementation of the algorithm, the control update (80) took 0.16 sec, and the affine model update (81), 0.23 sec. These computational delays could be acceptable even for the feedforward control of a real-life system, since the updates need to be done only once for each motion. The computation of control in accordance with (40) takes less than 25 ms, which suggests that the proposed algorithm is feasible for on-line control, especially if the updates (80) and (81) are scheduled outside time-critical feedback loops.

When *simulating* the planar arm motion, we assume that the arm links are uniform rods of unit mass and length. We take the moments of inertia of the drive rotors as $J = \mathrm{diag}\{2, \ 2\}$, the damping in drives as $B = \mathrm{diag}\{0, \ 0\}$, and the angular stiffnesses of the lumped elastic elements in the joints as $K = \mathrm{diag}\{200, \ 200\}$. We further assume that the angular position gain of the PD feedback controller (10) is $K_* = \mathrm{diag}\{100, \ 100\}$, and the angular velocity gain is $B_* = \mathrm{diag}\{40, \ 40\}$. Note that for the above parameters of the system, the period of oscillations with the lowest eigenfrequency is about 1, if the elbow angle is $3\pi/4$. The motion time $T = 1.5$ is close to this period, which makes the control problem very difficult. We have found that adding a small measurement noise to the simulated system output does not change the algorithm performance in any visible way. The reason is that for a random parameter vector sequence $p^{(k)}$, the error approximating the system mappings with the RBF networks in the algorithm already acts in the same way as an output noise.

In a numerical experiment, a sequence of the task parameter vectors p is generated so that the initial arm configuration coincides with the final arm configuration at the end of the previous task. Figure 6 shows the progress of the error $\|Y - Y_d\|$ with the optimization iteration number. One can see that the control error converges to a small acceptable value over the entire parameter vector domain. The error, which is achieved at the end of the optimization process, is about 20 times less than the initial error, which is obtained without feedforward. The oscillations of the motion error in Figure 6 are related to the variation in the arm motion amplitude as new task parameter vectors p are randomly generated in the course of the learning.

Figure 7 illustrates the feedforward control computed as a result of the RBF network approximation after the algorithm convergence for the motion with the initial joint angles $\lambda = [0^o \ \ 60^o]^T$, and the final angles $\nu = [70^o \ \ 105^o]^T$.

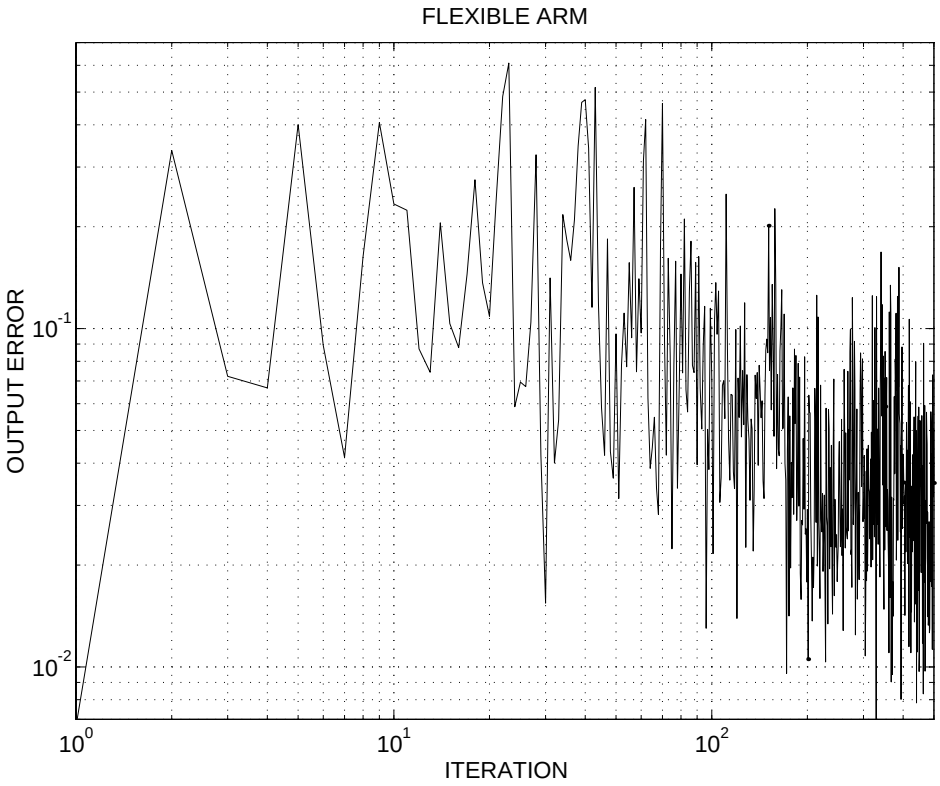

Figure 6: The progress of the terminal error $\|Y\|$ with the optimization iteration number. The arm moves through a randomly generated goal positions sequence.

Figure 8 shows the joint deformations for the same motion. Thanks to the computed feedforward, the deformation is small after the time $T = 1.5$, which means the arm arrives to the final position without visible oscillations. The acceptable motion accuracy is achieved despite the high motion speed, low feedback gains, moderate network size, and large covered domain of the task parameters (83). For comparison, Figure 9 shows the deformations for the same motion in the absence of the feedforward.

Figures 7 and 8 illustrate an example of the learned motion pattern produced by the network after the end of the training procedure. Figure 7 shows the time-dependence of the feedforward control computed by the network. Figure 8 illustrates the time histories of the joint deformations. Thanks to the computed feedforward, the deformation is small after the time $T = 1.5$. The acceptable motion accuracy is achieved despite the high motion speed, low feedback gains, moderate sizes of the RBF networks, and large covered domain of the task parameters (83). For comparison, Figure 9 shows results for the same motion as in Figure 8 obtained in the absence of the feedforward. In Figure 9, the oscillations continue long after the desired terminal time.

7 Conclusions

We have presented a new paradigm and RBF network architectures for task-level feedforward control of nonlinear systems. These algorithms belong to the realm of intelligent control and work at a higher hierarchical level compared to classical feedback or programmed control algorithms. We assume that the system operation can be considered as a sequence of clearly defined tasks and compute feedforward control for each task. The learning features of the proposed algorithms are aimed at optimizing the feedforward from one task to another based on the performance for a completed task. Dependence of the feedforward control on the task parameters is approximated using an RBF network.

The surveyed applications of the algorithms demonstrate their usefulness. This is illustrated by this paper example of point to point control of a flexible articulated arm. Computational resources required for practical implementation of the algorithms are moderate, especially since the algorithms need to run only once for each task. The application of the proposed paradigm can help to solve difficult practical control problems, for which other known methods are

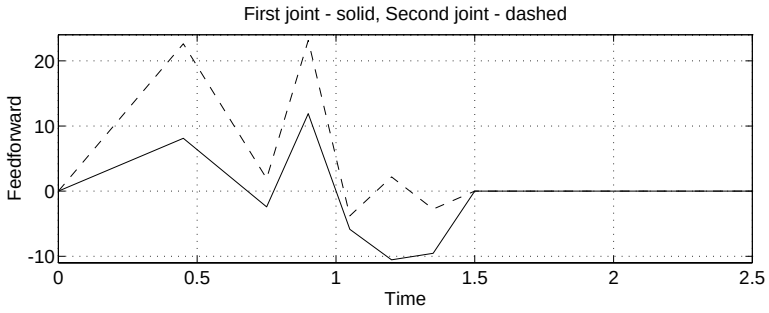

Figure 7: Feedforward for a test motion after the algorithm convergence
Shoulder joint - solid, elbow joint - dashed

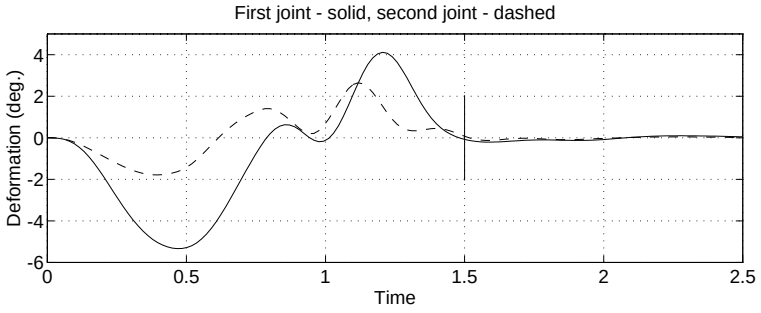

Figure 8: Joint deformations for the test motion with the approximated feedforward
Shoulder joint - solid, elbow joint - dashed

not adequate. The algorithms can also be extended to allow adaptive feedback control of nonlinear systems.

The main limitation of common approximation-based approaches to nonlinear control, such as neural networks, is the necessity of completing many learning trials in order to train the network. Generally, a number of examples required for the network identification (training) grows exponentially with the input variable dimension. In the proposed paradigm, such input variable is the task parameter vector p. Thus, the proposed task-level control technique offers the best advantage compared to the state-space learning approaches if there are few task parameters and the state dimension is high. This advantage is achieved because the proposed algorithms do not attempt to approximate full nonlinear dynamics of the controlled system and limit themselves to optimizing performance just for a parametric family of control tasks.

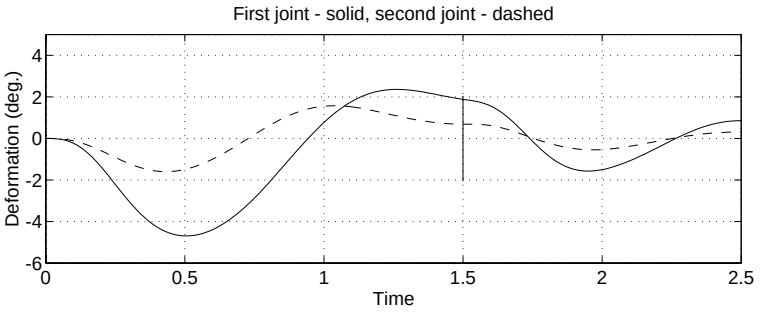

Figure 9: Joint deformations for the test motion with zero feedforward
Shoulder joint - solid, elbow joint - dashed

References

[1] Aboaf, C.G., Atkeson, C.G., and Reikensmeyer, D.J. "Task-level robot learning," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp.1311–1312, Philadelphia, PA, April 1988.

[2] Arimoto, S. "Learning control theory for robotic motion," *Int. J. of Adaptive Contr. and Signal Processing*, vol. 4, pp.453–564, 1990.

[3] Arimoto, S., Kawamura, S., and Miyazaki, F. "Bettering operation of robots by learning," *J. of Robotic Syst.*, vol. 1, pp.123–140, 1984.

[4] Atkeson, C.G. "Using locally weighted regression for robot learning," *Proc. 1991 IEEE Int. Conf. on Robotics and Automation*, pp.958–963, Sacramento, California, April 1991.

[5] Bishop, C. "Improving the generalization properties of radial basis function neural networks," *Neural Computation*, vol.3, pp.579–588, 1991.

[6] Bock, O., D'Eleuterio, G.M.T., Lipitkas J., Grodski, J.J. "Parametric motion control of robotic arms - a biologically based approach using neural networks," *Telematics and Informatics*, vol.10, no.3, pp.179-185, 1993.

[7] M. Botros and C. G. Atkeson, "Generalization Properties of Radial Basis Functions," in *Advances in Neural Information Processing Systems 3*, R. P. Lipmann, J. E. Moody, and D. S. Touretzky, (Eds.). Morgan Kaufmann, vol. 3, pp. 707–713.

[8] Branicky, M.S. "Task-level learning: Experiments and extensions," *Proc. 1991 IEEE Int. Conf. on Robotics and Automation*, pp.266–271, Sacramento, CA, April 1991.

[9] Brockett, R.W. "Control theory and singular riemanian geometry," in P.Hilton and G.Young, eds, *New Directions in Applied Mathematics*, Springer-Verlag, 1981

[10] Broomhead, D.S., and Lowe, D. "Multivariable functional interpolation and adaptive networks," *Complex Systems*, no.2, pp.321–355, 1988.

[11] Chen, S., and Billings, S.A. "Neural networks for non-linear dynamic system modelling and identifications," *Int. J.Control*, vol.56, no.2, pp.319–346, 1992.

[12] Chen, S., Billings S.A., and Grant P.M. "Recursive hybrid algorithm for non-linear systems identification using radial basis function networks," *Int. J.Control*, vol.55, no.5, pp.1051–1070, 1992.

[13] Chen, S., Cowan, C.F.N., and Grant, P.M. "Ortogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. on Neural Networks*, vol.2, no.2, pp.837–863, 1991.

[14] Craig, J. *Introduction to Robotics*, 2nd edition. Addison-Wesley, New York, 1989.

[15] Dennis, J.E., Schnabel, R.B. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice Hall, Englewood Cliffs, 1983.

[16] Dyn, N. "Interpolation of scattered data by radial functions," Schumaker, L.L. Chui, C.K., and Utreras F.I., ed., *Topics in Multivariable Approximation*. Academic Press, Boston, 1987. pp. 47-61.

[17] Fernandes, C., Gurvits, L., and Li, Z.X. "Foundations of nonholonomic motion planning," Technical Report, Robotics Research Laboratory TR 577 RR 253, Courant Insititute of Mathematical Sciences, N.-Y., 1991.

[18] Franke, R. "Scattered data interpolation: Test of some methods," *Math. of Computation*, vol. 38, no.157, pp.181–200, 1982.

[19] Franke, R. "Recent advances in the approximation of surfaces from scattered data," Schumaker, L.L. Chui, C.K. and Utreras F.I., ed., *Topics in Multivariable Approximation*. Academic Press, Boston, 1987. pp. 79-98.

[20] Geng Z. et al. "Learning control system design based on 2-D theory - an application to parallel link manipulator," *Proc. 1990 IEEE Int. Conf. on Robotics and Automation*, pp.1510–1515, Cincinnaty, Ohio, April 1990.

[21] Goodwin, G.C., and Sin, K.S. *Adaptive Filtering, Prediction and Control*, Prentice-Hall, Englewood Cliffs, NJ, 1984.

[22] Gorinevsky, D.M. "Learning and approximation in database for feedforward control of flexible-joint manipulator," *ICAR '91: 5th Int. Conf. on Advanced Robotics*, pp.688–692, Pisa, June 1991.

[23] Gorinevsky, D.M. "On the approximate inversion of linear system and quadratic-optimal control," *J. Comput. Syst. Sci. Internat.*, vol.30 , no.6, pp.6-23, 1992.

[24] Gorinevsky D.M. "Experiments in direct learning of feedforward control for manipulator path tracking," *Robotersysteme*, vol.8, pp.139–147, 1992.

[25] Gorinevsky, D.M. "Modeling of direct motor program learning in fast human arm motions," *Biological Cybernetics*, vol. 69, no. 3, 1993, pp. 219-228

[26] Gorinevsky D.M. "Adaptive learning control using Radial Basis Function network approximation over task parameter domain" *Proc. 1993 IEEE Int. Symp. on Intelligent Control*, Chicago, IL, August 1993

[27] Gorinevsky, D.M. "Sampled-data indirect adaptive control of bioreactor using affine Radial Basis Function network approximation," *Tr. ASME. J. Dynam. Syst. Meas. and Control*, vol. 188, March 1996

[28] D.M. Gorinevsky, "On the persistency of excitation in Radial Basis Function network identification of nonlinear systems," *IEEE Tr. on Neural Networks*, Vol. 6, No. 5, pp. 1237–1244, 1995.

[29] D.M. Gorinevsky, "An algorithm for on-line parametric nonlinear least square optimization," *33rd IEEE CDC*, Lake Buena Vista, FL, December 1994.

[30] Gorinevsky, D.M. "Learning task-dependent input shaping control using Radial Basis Function network," *IEEE World Congress on Computational Intelligence*, Orlando, FL, June 1994

[31] Gorinevsky, D.M., "An approach to parametric nonlinear least square optimization and application to task-level learning ontrol," *IEEE Tr. on Automatic Control*, (submitted)

[32] Gorinevsky, D., Feldkamp, L. "RBF network feedforward compensation of load disturbance in idle speed control," *IEEE Control System Magazine*, December 1996

[33] Gorinevsky, D.M., and Connolly, T.H. "Comparison of some neural network and scattered data approximations: The inverse manipulator kinematics example," *Neural Computation*, vol. 6, no. 3, 1994, pp. 519-540

[34] Gorinevsky, D.M., Kapitanovsky, A., and Goldenberg, A.A., "Neural network architecture for trajectory generation and control of automated car parking," *IEEE Tr. on Control Systems Technology*, Vol. 4, No. 1, pp. 50–56.

[35] Gorinevsky, D.M., Kapitanovsky, A., and Goldenberg, A.A. "Radial Basis Function network architecture for nonholonomic motion planning and control of free-flying manipulators," *IEEE Tr. on Robotics and Automation*, vol. 12, no. 3, pp. 491–496.

[36] Gorinevsky, D., Torfs, D, and Goldenberg, A.A. "Learning approximation of feedforward control dependence," *IEEE Tr. on Robotics and Automation* (to appear)

[37] D. Gorinevsky and G. Vukovich, "Control of flexible spacecraft using nonlinear approximation of input shape dependence on reorientation maneuver parameters," *13th World Congress of IFAC*, San Francisco, CA, June 1996.

[38] Gugliemo, K., and Sadeh, N. "Experimental evaluation of a new robot learning controller," *Proc. 1991 IEEE Int. Conf. on Robotics and Automation*, pp.734–739, Sacramento, CA, April 1991.

[39] Hara, S., Yamamoto, Y., Omata, T., and Nakano, M. "Repetetive control systems: A new type of servo systems for periodic exogeneous signals," *IEEE Trans. on Automat. Cont.*, vol. 33, no. 7, pp.659–668, 1988.

[40] Hartman, E. and Keeler, D. "Predicting the future: Advantages of semilocal units," *Neural Computation*, no.3, pp.566–578, 1991.

[41] Horowitz, R., Messner, W., and Moore, J.B. "Exponential convergence of a learning controller for robot manipulator," *IEEE Trans. on Automat. Cont.*, vol. 36, no. 7, pp.890–894, 1991.

[42] Hunt, K.J., Sbarbaro, D., Zbikowski, R., and Gawthorp, P.J. "Neural networks for control systems - a survey," *Automatica*, vol.28, no.6, pp.1083–1112, 1992.

[43] Ishihara, T., Abe, K., Takeda, H. "A discrete-time design of robust iterative learning algorithm," *IEEE Trans. on Systems Man and Cybernetics*, vol. 22, no. 1, pp. 74–84, 1992.

[44] Kadirkamanathan V., Niranjan M., and Fallside, F. "Sequential adaptation of radial basis function neural networks and its application to time-series prediction," Moody, J.E., Lipmann, R.P., and Touretzky D.S., ed., *Advances in Neural Information Processing Systems 3*, vol.3, pp.721–727. Morgan Kaufmann, San Mateo, CA, 1991.

[45] Kansa, E.J. "Multiquadrics - a scattered data approximation scheme with applications to computational fluid dynamics - I," *Computers Math. Applic.*, vol.19, no.8, pp.127–145, 1990.

[46] Kawato, M. "Adaptation and learning in control of voluntary movement by the central nervous system (tutorial)," *Advanced Robotics*, vol. 3, no. 3, pp.229–249, 1989.

[47] J. A. Leonard, M. A. Kramer, and L. H. Ungar, "Using radial basis functions to approximate a function and its error bounds," *IEEE Trans. Neural Networks*, vol. 3, no. 4, pp. 624–627, 1992.

[48] Messner, W. et al. "A new adaptive learning rule," *IEEE Trans. on Automat. Cont.*, vol. 36, no. 2, pp.188–197, 1991.

[49] Micchelli, C.A. "Interpolation of scattered data: Distance matrices and conditionally positive definite functions," *Const. Approx.*, vol.2, pp.11–22, 1986.

[50] Moody, J., and Darken, C.J. "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol.2, pp.281–284, 1989.

[51] Mukhopadhyay, S., and Narendra, K.S. . "Disturbance rejection in nonlinear systems using neural networks," *IEEE Tr. on Neural Networks*, vol.4, no.1, pp.63–72, 1993.

[52] Narendra, K.S. "The maturing of adaptive control," *Foundations of Adaptive Control*, Kokotovich, P.V., ed., pp.3–36, Lecture Notes in Control and Information Science, vol.160, Springer-Verlag, Berlin, 1993.

[53] Narendra, K.S., and Parthasarathy, K. "Identification and control of dynamical systems using neural networks," *IEEE Tr. on Neural Networks*, vol.1, no.1, pp.4–26, 1990.

[54] Oh, S.R., Bien, Z., and Suh, I.H. "An iterative learning control method with application for the robot manipulator," *IEEE J. on Robotics and Automation*, vol. 4, no. 5, pp.508–514, 1988.

[55] Pao, Y.-H. *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley, Reading, MA, 1989.

[56] Park, J., and Sandberg, I.W. "Universal approximation using radial-basis-function networks," *Neural Computation*, vol.3, pp.246–57, 1991.

[57] Platt, J. "A resource-allocating network for function interpolation," *Neural Computation*, vol.3, no.3, pp.213–225, 1991.

[58] Poggio, T., and Girosi, F. "Networks for approximation and learning," *Proceedings of the IEEE*, vol.78, no.9, pp.1481–1497, 1990.

[59] Polycarpou, M.M., and Ioannou, P.A., 1992, "Modelling, identification and stable adaptive control of continuous-time nonlinear dynamical systems using neural networks," *American Control Conf.*, pp.36–40, Chicago, IL.

[60] Powell, M.J.D. "Radial basis functions for multivariable interpolation: A review," Mason, J.C., and Cox, M.G., ed., *Algorithms for Approximation*, pp. 143-168, Clarendon Press, Oxford, 1987.

[61] Powell, M.J.D. "The theory of Radial Basis Function approximation in 1990," Light, W., ed., *Advances in Numerical Analysis.* vol. 2, pp. 102-205, Clarendon Press, Oxford, 1992.

[62] Sanchez, V.D., and Hirzinger, G. "State-of-the-art robotic learning control based on artificial neural networks. An overview," Khatib, O. et al., ed., *The Robotic Review 2.* MIT Press, 1991.

[63] Sadeh, N. "A perceptron network for functional identification," *IEEE Trans. on Neural Networks*, vol.4, no.6, pp.982–988, 1993.

[64] Sanner, R.M., and Slotine, J.-J.E. "Gaussian networks for direct adaptive control," *IEEE Trans. on Neural Networks*, vol.3, no.6, pp.837–863, 1992.

[65] Spong, M. "Modeling and control of elastic joint robots," *Trans. ASME. J. Dynam. Syst. Meas. and Control*, vol.109, no.4, pp.310–319, 1987.

[66] Tao, K.M., Kosut, R.L., and Aral, G. "Learning feedforward control," *American Contr. Conf*, pp.2575–2579, Baltimore, MD, 1994.

[67] Tikhonov, A.N., and Arsenin, V.Ya. *Methods for Solution of Ill-Posed Problems.* Nauka, Moscow, 2nd edition, 1979. (in Russian).

[68] Togai, M., and Yamano, O. "Learning control and its optimality," *Proc. 1986 IEEE Conf. on Robotics and Automation*, pp.248–243, San Francisco, CA, 1986.

[69] Tolle, H., et al. "Learning control with interpolating memories – general ideas, design lay-out, theoretical approaches and practical applications," *Int. J. Cont.*, vol. 56, no. 2, pp.291–317, 1992.

[70] Tolle, H., Militzer, J., and Erzüe, E. "Zur Leistungsfähigkeit lokal verallgemeinender assoziativer Speicher und ihren Einsatzmöglichkeiten in lernenden Regelungen," *Messen Steuern Regeln*, vol. 32, no. 3, pp.98–105, 1991.

[71] Vlassenbroeck, J., and Rene, Van D. "A Chebyshev technique for solving nonlinear optimal control roblems," *IEEE Trans. on Automat. Cont.*, vol. 33, no. 4, pp.333–340, 1988.