A dissertation submitted for the degree of

Doctor of Philosophy

of the

Pierre and Marie Curie University

in

Mechanics and Robotics

presented by

Camille SALAÜN

LEARNING MODELS TO CONTROL REDUNDANCY IN ROBOTICS

30^{th} *August 2010*

Committee in charge

M. P. Bidaud	Professor at the Pierre & Marie Curie University	Examiner
M. M. Butz	Assistant professor at the University of Würzburg, Accreditation to Supervise Research	Referee
M. J.-Y. Fourquet	Professor at the National Engineering School of Tarbes	Referee
M. V. Padois	Assistant professor at the Pierre & Marie Curie University	Co-Adviser
M. O. Sigaud	Professor at the Pierre & Marie Curie University	Adviser
M. P. Souères	Research Director at the National Centre for Scientific Research	Examiner

Abstract

The context of this work is the emergence of service Robotics, where robots will need adaptive capabilities to interact with people in unforseen circumstances. More precisely, this thesis is concerned with the design of adaptive model-based control methods for redundant systems in the case where kinematics and/or dynamics are not precisely known in advance or change along time. We present an adaptive control approach combining model learning methods with the Resolved Motion Rate Control approach. The main justification of that framework lies in the necessity of combining several tasks, ranked by priority, controlling explicitly the redundancy of the robot. We learn the forward kinematic model of a system and use standard algebraic methods to extract pseudo-inverses and projectors from it. This combination endows the robot with the ability to realize hierarchically organised tasks in parallel, using tasks null space projectors built upon the learnt models. We illustrate the proposed method on several simulated systems, highlighting the fact that the internal mobility can be controlled with learnt models ensuring that the hierarchy is verified. We have also implemented our method on the iCub humanoid robot, performing simple tasks such as dialling on a numeric keyboard or drawing a circle. The second part of our framework is concerned with the learning of inverse dynamics with state-of-the-art methods. Learning this model allows to stress the versatility of our approach to external perturbations and thus to unknown environment.

The experiments performed in the thesis illustrate the capability of the framework to deal with several hierarchically organised tasks as well as to adapt to external perturbations. They also reveal the limitations of the framework and of the learning algorithm we used in terms of scalability and speed of adaptation.

Finally, we derive some lines of research that should be investigated to go further, such as using vision sensors to learn models from exteroceptive data or dealing with systems endowed with artificial muscles to benefit from the compliance of such actuators.S

This thesis tries to develop a framework in order to include adaptive capabilities in the control loop of future service robots in interaction with people in an unstructured and evolving environment.

Contents

List of Symbols

Scalar are lower-case letters. Matrix are upper-case letters. Vector are noted in bold lower-case letters. A † in exponent of a letter represent a task to accomplish. A $\star$ in exponent of a letter represent an intermediate task used as control input. A T in exponent of a matrix represent its transpose. A ˆ above a letter symbolize an estimation.

α	Learning rate. Tune the speed of the convergence of a learning algorithm
β	Linear weight or linear model or coefficient of regression
$\delta \boldsymbol{y}$	error between desired value and learnt value
δt	A time step
$\dot{\boldsymbol{\nu}}^{\dagger}$	Desired operational acceleration often used as feed-forward
$\dot{\boldsymbol{\nu}}^{\star}$	Input control operational acceleration
ϕ	Radial basis function
Ψ_i	A coordinate frame named i
$\boldsymbol{\epsilon}$	Unmodelled dynamical effects
$\boldsymbol{\nu}^{\dagger}$	Desired operational velocity
$\boldsymbol{\nu}_{ab}^{c}$	Linear velocity of the body B_b relatively to the body B_a expressed in the coordinate frame Ψ_c
$\boldsymbol{\omega}_{ab}^{c}$	Angular velocity of the body B_b relatively to the body B_a expressed in the coordinate frame Ψ_c
$\boldsymbol{\tau}$	Torque vector
$\boldsymbol{\tau}^{ext}$	Torque resulting from external forces
$\boldsymbol{\xi}$	Operational space parameters
$\boldsymbol{\xi}^{\dagger}$	Desired task space parameters
$\boldsymbol{b}$	Vector representing the sum of all non-linear effects
$\boldsymbol{c}$	Vector of centre of a receptive field or a Gaussian or a classifier

$\boldsymbol{g}$ Gravity vector

$\boldsymbol{n}$ Vector of non-linear Coriolis and centrifugal effects

$\boldsymbol{p}$ A point

$\boldsymbol{q}$ Joint space parameters also named configuration parameters or generalised coordinates

$\boldsymbol{T}_{ab}^{c}$ Twist of the body B_b relatively to the body B_a expressed in the coordinate frame Ψ_c

$\boldsymbol{x}$ Input vector of a learning algorithm

A Inertia matrix

$Ad(H)$ Adjoint of the homogeneous matrix H

B_i A body named i

E_i^j Joint between two rigid bodies B_i and B_j

H_{ab} Homogeneous matrix representing the relative configuration of a rigid body B_b relatively to a body B_a

$J_{i,j}$ Jacobian matrix of body B_j expressed in coordinate frame Ψ_i

K_d Gain proportional to a velocity error

K_i Gain proportional to the integral over the time of all position errors

K_p Gain proportional to a position error

m Number of degrees of freedom of the end-effector

n Number of degrees of freedom of a rigid poly-articulated system

n_b Number of bodies in a poly-articulated system

n_c Number of constrained movements

R A rotation matrix

t Current time

$T_{ab}^{c}(t)$ Twist, under its matrix form, of the body B_b relatively to the body B_a expressed in the coordinate frame Ψ_c

v Excitation level

n_n Number of neurons, number of receptive fields or number of classifiers

Introduction

Real-world Robotics applications are evolving from the industrial domain, where the application is well-defined within a structured environment, to the service domain where it is much harder to model the overall mission of a robot. The service Robotics domain implicates robots in interaction with people, at home, at work or in medical institutions for instance. In contrast with the former industrial Robotics domain, a service robot cannot be designed for a single specific environment. Its interactions with people cannot be predicted and its missions can be quite diverse and unforeseen. Thus service Robotics induces complexity and uncertainty both in terms of the mission of the robot and in terms of the nature of the environment where robots are supposed to move. In parallel to this growing complexity and partly induced by it, the complexity of the robots themselves increases too, both in terms of the number of sensors they are now equipped with and in terms of number of degrees of freedom (e.g., mobile manipulators such as the humanoid robot iCub [Metta et al., 2008] or the wheeled assistant PR2[1]). Before examining the impact of this growing complexity, we must stress that most robotic applications can be modelled at two levels, the mission level and the task level, which are presented hereafter.

The mission level The mission of a robot can generally be decomposed into a set of tasks that need to be scheduled appropriately so that the mission succeeds. The mission is independent of the detailed execution of each task. For instance, the *Mars Exploration Rover Mission* which consists in investigating an area on the surface of Mars with wheeled robots [Crisp et al., 2003] may require to achieve a series of tasks such as *land on Mars*, *navigate*, *take picture*, *collect minerals*, *analyse minerals*, etc.

At the mission level, most deterministic task scheduling methods can address very complex mission in the case where everything can be envisioned in advance [Lee et al., 1997]. But these methods have to be reconsidered in the service domain so as to incorporate uncertainty in their control mechanisms [Lussier et al., 2007] and benefit from a richer external and contextual information from the sensors of the robot. This task scheduling issue in the presence of uncertainty is the matter of an important research effort that lies outside the scope of this thesis since we do not address this mission level.

The task level At the task level, the robot must reach a set of elementary goals by performing adequate movements, using appropriate sensors and actuators into a control loop. A task can be regarded as several goals that can be achieved by the robot, totally

[1] http://www.willowgarage.com/pages/robots/pr2-overview

or partially, simultaneously or not, using different effectors. Thus, the task level can be seen either as a motion generation problem or as a control problem, depending on the application and on the focus of the research. As a consequence, some robot architecture distinguish a task level and a control level. In this thesis, we will consider the task level and the control level as one unique level, but we will distinguish the motion generation view and the control view of task achievement. In the context of service Robotics, when the task achievement is seen as a pure motion-planning problem, motion planning methods have to be reconsidered too. For instance, configuration space planning methods [Latombe, 1991; Laumond, 1998] consist in building a path from an initial state to a goal state in the so-called configuration space in such a way that it is guaranteed that the robot can execute the trajectory without meeting any obstacle. Important progress have been made recently with the intensive use of randomness in the path or graph building process with families of methods called Rapidly exploring Random Trees (RRT) [LaValle, 2006] and Probabilistic Road-Maps (PRM) [Kavraki and Latombe, 1998]. But configuration space planning requires that the environment itself is perfectly known before planning can take place, and the methods are very sensitive to unknown environmental features. Thus, these methods can hardly be used in the broad context of service Robotics outlined above.

When task achievement is rather seen as a control problem, different representations and methods are used. It is first necessary to formalize the tasks, associated constraints and evaluation criteria with languages that give birth to natural control methods. Such languages are usually defined by calling upon specific spaces: the task space, the joint space, the sensory space, the contact space, etc. The spaces of utmost importance, for most task achievement problems, are the task space and the joint space. Indeed, actuators of a robot generally act on joints, but the tasks or constraints associated to a mission can seldom be described in the joint space in a natural way. The task space, also called operational space, provides an alternative and more natural space, for such a definition.

Then, once such spaces are defined, several control approaches can be distinguished:

- One may call upon motion planning methods in the task space and then upon task space to joint space transformation to generate adequate joint controls. This approach suffers from the same limitation as configuration space motion-planning in terms of reactivity to uncertainties or unexpected events. Nevertheless, this problem has been locally resolved with elastic strips/band planning [Quinlan and Khatib, 1993; Brock and Khatib, 2002].
- One may call upon optimal control methods that define task-related criteria in the task space and look for the joint controls that optimize those criteria. Here again, such methods hardly take uncertainties or unmodelled effects into account. Nevertheless, new methods such as Stochastic Optimal Feedback Control can incorporate some uncertainties modelled as Gaussian noise [Todorov, 2004; Li, 2006]. Above all, these methods are currently limited in terms of number of degrees of freedom they can address (e.g. the ILQG algorithm can be used on a 2 degrees of

freedom arm with 10 actuators).

- In contrast with such methods that need to envision the complete task execution in advance and thus require knowledge that is often not available in service Robotics contexts, reactive methods do not suffer from such a necessity. An instance of such methods is the *Resolved Motion Rate Control* (RMRC) approach [Whitney, 1969]. Another purely reactive approach is the Sequential Quadratic Programming (SQP) algorithm which is an optimization tool used to solve a problem with linear constraints minimizing some cost function. A new trend consists in combining the reactive properties of the previous methods and the capability to optimize criteria and constraints fulfilment over an horizon of optimal control methods through a cascade of reactive optimization controllers. Such methods are instances of model predictive control methods. Research on such hybrid methods is extremely active within the context of humanoid Robotics at the moment [Kajita et al., 2003; Wieber, 2006; Kanoun et al., 2009].

Combining tasks The opportunity to achieve several tasks simultaneously has increased with the emergence of more and more complex robots and mastering this capability is mandatory in many service Robotics contexts. For instance, for hammering a nail, the robot will need to hold the nail at the required position and to move the hammer to hit the nail in the appropriate direction. A mathematical framework that facilitates the definition and the unperturbed execution of a hierarchy of tasks in the case of redundant robots was proposed by Liégeois [1977]. A robot is said redundant relatively to a task if it is endowed with more degrees of freedom than necessary to achieve this task. Some supernumerary degrees of freedom are available and can be used to achieve supplementary tasks without perturbing the main task.

Identification versus learning

The motion controllers developed for service robots have to be either highly robust to uncertainties in the model of the robot and its environment or adaptive, i.e., able to build their own model on-line. All control methods cited above are model-based as they need a model of the geometry, the kinematics and sometimes the dynamics of the system they are controlling. A real knowledge in Mechanics and an hard work in modelling is necessary to design these models, particularly in the case of complex poly-articulated systems such as humanoid robots. If they are only extracted from physics equations, those models are called *deduced models*. They rely on a theoretical modelling of a system such as the CAD of the robot, and cannot represent precisely the material system. In fact, due to physical differences such as friction, weights, misalignments, lengths or backlashes, during the design, manufacture or the assembly of multiple robots considered as identical, it is difficult to control them with a unique deduced model.

As engineers want to control real robots with models, they use deduced models melded with parametric identification methods [Ljung, 1986]. Those models are called *induced*

models as they are created from a great amount of data coming from the real system, in a batch mode or incrementally. Parametric identification is usually performed off-line but induced models can also be estimated along the life-time of the robot in a process called Adaptive Control [Sastry and Bodson, 1989; Landau et al., 1998; Siciliano et al., 2008]. Instead of having an unchanging model, adaptive control adapts parameters of a model to overcome difficulties in control that can appear if the system is modified. This method allows to adapt to unmodelled perturbations or system modifications which occur during an experiment such as an increasing friction due to a misalignment in some joint.

With parametric identification methods or adaptive control, omitting in the model of the simulation some physical phenomena acting on the system may introduce a bias. Unfortunately, when the structure of the model is fixed, if this structure does not match the physical structure of the mechanical system, adjusting the parameters will not be sufficient to correct the resulting model errors. Unstructured adaptive models, also called non-parametric models, can deal with such problems. They may be obtained using machine learning methods and included into classical robotic controllers. It can be done using model-based controllers while using machine learning methods capable of building models on-line.

Machine learning methods learn a model which does not depend on a predefined structure but approximates a function based only on data received from the sensors of the real system without making assumptions about their physical relationship. The information used in machine learning is the size of the data which determines the number of input/output dimensions of the model and the nature of the data which induces the dependencies within the model. Learning algorithms provide a way to avoid introducing a priori knowledge in the model. For example, if we give as input: joint position and as output: operational position, the model learnt is a forward kinematic model. That model does not explicitly depend on the dimensions of the robot while the lengths are learnt in the model. Machine learning algorithms can also adapt to environmental changing constraints or to modifications of the robot itself.

Applications of machine learning in Robotics

Machine learning methods outlined above are mandatory to address the future service Robotics context. For instance, consider a general purpose robot equipped with a hand and whose goal is to achieve a task that requires a tool, such as screw-driving, hammering a nail or a similar task. When holding a tool, the end point effector is usually a functional tip of the tool itself (e.g. the head of a hammer or the teeth of a fork).

If the tool is grasped by robotic hands rather than precisely set by an expert roboticist, the grasping configuration may be unprecisely known. In that case, the control law will have to be adaptive with respect to the geometry relating the end-point effector to the configuration of the robot + tool system. One way to make the control law adaptive is to learn this geometry on-line.

Furthermore, the robot may meet and grab in its environment some tools whose

dynamical properties such as the global mass, inertia parameters, etc., are relevant for their use but not known in advance. For instance, the force necessary for a hammering gesture is highly dependent on the mass of the head of the hammer.

Learning both the geometry and the dynamics can also bring important advantages in the emerging domain of compliant devices. Indeed, the geometry of a compliant arm will change temporarily if it bends at the contact of some surface in its environment. There might also be some complex relationship between the geometry and the dynamics in the case of a very flexible body.

Finally, the emerging domain of modular Robotics can also benefit from learning such mechanical models. Modular Robotics is a field where robots are composed of multiple simple modules connected altogether. A modular robot can reconfigure itself depending on its task. The reconfiguration induces a change in the morphology of the robot that can perturb its control but also a change of the dynamics induced by the addition of various segments that alter the inertia seen by actuators of the corresponding mechanical chain. In the case of self-assembling modular robots, the connection of modules may also be somewhat imprecise, giving rise to uncertainties in the geometry and dynamics of the global system.

Goal of the thesis

The robotics context described above are research challenges for the future rather than topics for current robotics industrial development. Thus, in this thesis, we do not pretend to provide a ready-to-use solution for any of such challenges. Rather, our goal consists in laying the foundations of a framework for addressing some aspects of such a challenge. More precisely, we are trying to design adaptive model-based control methods for redundant systems in the case where kinematics and/or dynamics are not precisely known in advance or change in time. We propose a framework that addresses the control of redundant robots in the operational space thanks to models learnt with non-parametric representations. The main justification of that framework lies in the necessity of combining several tasks, ranked by priority, controlling explicitly the redundancy of the robot. We have chosen to solve control problems of redundant robots at the task level using the RMRC framework.

To design such a framework, we have to ensure two features.

- As the velocity kinematic level is sufficient to define and solve internal mobility, our framework considers separately the kinematics and the dynamics. A side effect of this explicit separation is that a sub-part of our framework can address velocity controlled redundant systems, such as most of the humanoid robots, which do not give access to control at the joint torque level.
- Both mechanical models of the system have to be learnt from experience by the robot in any state so that those global models are always determined. Such a property is particularly necessary when the task can be executed in an infinite number of ways.

This framework is not targeting any specific real-world application but rather trying to design a general method. As a result, we do not hesitate to simplify the context of our experiments when necessary. Of course, if these assumptions are reasonable given the *methodological* nature of our goal, the spawn limitations will need to be overcome to address real applications. For instance, in all chapter, we learn mechanical models of systems whereas we know the analytical ones. Those models are useful as we can simulate the system, perform random bootstrap or get any operational position without considering practical considerations. It is clear that, in the case of a real robot, the random initialization will be less easy and operational positions will have to be measured from exteroceptive sensors. This is the matter of an important discussion in the conclusive chapter of this thesis.

The thesis is organised as follows.

- In Chapter 1, we describe different levels of forward and inverse mappings which can be used to relate the joint space to the task space and vice versa. We also present different contexts in which several tasks can be combined depending on their compatibility. Then, we give some background information on hierarchical control, a model-based control approach which provides a mathematical framework allowing to control a robot in task space and to combine several tasks.
- In order to endow this framework with adaptive capabilities, one must develop learning methods and associated representations fitting the needs of the corresponding control techniques. In Chapter 2, we give an overview of the learning methods that can be applied to learn these forward and inverse mappings. Learning models of the robot is achieved using specific representations such as Neural Networks [Van der Smagt, 1994] or Radial Basis Function Networks [Sun and Scassellati, 2004], Gaussian Processes [Shon et al., 2005] or [Nguyen-Tuong et al., 2009], Gaussian Mixture Models [Calinon et al., 2007], Locally Weighted Regression methods [D'Souza et al., 2001; Natale et al., 2007; Peters and Schaal, 2008] or Learning Classifier Systems [Butz and Wilson, 2000; Wilson, 2002] but the control methods used in the corresponding works do not always take advantage of the state-of-the-art techniques developed in recent Robotics research. We conclude from our overview that no currently available combination of model learning method and model-based control framework meets the requirement expressed above.
- In Chapter 3, we present our approach, explaining how we are able to learn the mechanics of a poly-articulated system at the kinematic and dynamic levels with two algorithms: LWPR and XCSF. First, we learn forward velocity kinematics which is inverted to control the system in the task space, so that we can control redundancy. Second, we describe how to learn the inverse dynamics which is used to transform a desired joint velocity into torques that must be applied to the system. We present the process used to learn the models as they differ depending on the learning algorithm used.
- In Chapter 4, we introduce a series of simulations where we learn kinematic models of simple systems. We evaluate our approach in the context of controlling the

redundancy of simulated robots in different constrained cases.

- In Chapter 5, we present experiments performed on iCub, a 56 degrees of freedom humanoid robot. Those experiments highlight the necessity of controlling the redundancy of such robots and the limitations of our framework when applied to a real velocity-controlled robot. We stress the necessity to use exteroceptive sensor to learn models and close the control loops if we want to move from the proof-of-concept context to real applications.
- In Chapter 6, we present further experimental work in the case where we learn both the kinematics and the dynamics of a system to control it with torques. Experiments are performed again with a simple simulated system. We underline the necessity of learning both models to control explicitly the redundancy. We show that learning the dynamics results also in the capability to adapt to external perturbations such as a force applied to the end-effector.
- In Chapter 7, we draw the lessons from our experimental evaluations by highlighting the contributions and limitations of our work. In particular, we stress the additional research that should be performed to start addressing real robotics applications in the service domain. We conclude by suggesting some perspectives that result from our research effort.

1 Controlling redundant systems in operational space

Contents

This chapter presents a mathematical framework in which we can design mechanical models to describe a large variety of robots. It first gives some background information about the Mechanics of rigid bodies and introduces joint space to task space mappings of poly-articulated systems at the kinematic, velocity kinematic and dynamic level. We recall the general expression of minimum norm solutions in the redundant case and give an overview of redundancy resolution schemes in control at the velocity level. We also explain how control in the operational space may be integrated in a control loop with an inverse dynamics model.

1.1 Mechanics of rigid bodies

The rigid body description used in this thesis is defined in this section. It is inspired by the thesis of Duindam [2006] that is based on the works of Selig [2005] and Stramigioli and Bruyninckx [2001]. It explains how to position a body in space which is necessary to position the bodies of a poly-articulated system or to define a task.

1.1.1 Configuration of a rigid body

The configuration of a rigid body B_i can be described by a coordinate frame[1] Ψ_i attached to this body. The pose of a coordinate frame is composed of a position, the position of its origin, and an orientation usually defined by an orthonormal basis. A coordinate frame is defined relatively to another coordinate frame, for example the reference frame chosen arbitrarily.

A point $\boldsymbol{p}$, defined relatively to a frame, can be positioned freely with three degrees of freedom known as its coordinates

$$\boldsymbol{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}. \tag{1.1}$$

We call degrees of freedom the independent displacements which specify the position and the orientation of a body. The application which defines a position of a point in space relatively to a frame belongs to the translation group called $T(3)$. The application which defines an orientation of a frame belongs to the 3 dimensional rotation group called $SO(3)$. The combination of those two groups is called $SE(3)$, the Special Euclidean group in three dimensions. Thus, a body owns a maximum of six degrees of freedom. The pose of a coordinate frame defined relatively to a reference coordinate frame belongs to $SE(3)$ and can be described by a homogeneous matrix.

A homogeneous matrix H is a matrix of the form

$$H = \begin{bmatrix} R & \boldsymbol{p} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \tag{1.2}$$

with R a rotation matrix, i.e., $R^{-1} = R^T \in \mathbb{R}^{3\times3}$ where $.^T$ denotes the transpose of the matrix and $det(R) = 1$. Besides $\boldsymbol{p} \in \mathbb{R}^{3\times1}$.

In contrast with other representations such as Roll, Pitch, Yaw angles or Euler angles, homogeneous matrices are globally continuous and can be easily multiplied using basic matrix products. Another globally continuous representation consists in using quaternions which can also be easily combined.

1.1.2 Velocity of a rigid body

If $H_{ab}(t)$ is the homogeneous matrix representing the relative configuration of a rigid body B_b relatively to a body B_a, we can describe the velocity of the rigid body B_b relatively to the rigid body B_a in the coordinate frame Ψ_c by a matrix twist $T^c_{ab}(t)$

[1] if it is not specified otherwise, all coordinate frames are right-handed

which can be computed from $H_{ab}(t)$ as

$$T_{ab}^{c} = \begin{bmatrix} \tilde{\Omega}_{ab}^{c} & \boldsymbol{\nu}_{ab}^{c} \\ 0 & 0 \end{bmatrix} = H_{ca}\dot{H}_{ab}H_{bc} = \begin{bmatrix} R_{ca}\left(\dot{R}_{ab}R_{ba}\right)R_{ac} & R_{ca}\dot{\boldsymbol{p}}_{ab} - R_{ca}\dot{R}_{ab}\boldsymbol{p}_{bc} \\ 0 & 0 \end{bmatrix} \tag{1.3}$$

where $\tilde{\Omega}$ denotes the vectorial pre-product matrix associated with vector $\boldsymbol{\omega}$ and t the current time. $\tilde{\Omega}$ is defined by

$$\tilde{\Omega} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \qquad \forall\ \boldsymbol{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \in \mathbb{R}^3. \tag{1.4}$$

It can be written in a compact form

$$\boldsymbol{T}_{ab}^{c} = \begin{bmatrix} \boldsymbol{\omega}_{ab}^{c} \\ \boldsymbol{\nu}_{ab}^{c} \end{bmatrix} \in \mathbb{R}^6 \tag{1.5}$$

where $\boldsymbol{\omega}_{ab}^{c}$denotes the angular velocity of the body B_b relatively to the body B_a expressed in the coordinate frame Ψ_c and $\boldsymbol{\nu}_{ab}^{c}$ denotes its linear velocity.

It is also necessary to define the adjoint operator, used to express a twist in another reference frame.

$$\boldsymbol{T}_{ab}^{d} = Ad(H_{dc})\boldsymbol{T}_{ab}^{c} \tag{1.6}$$

where $Ad(H) = \begin{bmatrix} R & 0 \\ \tilde{\boldsymbol{p}}R & R \end{bmatrix}$ with $H = \begin{bmatrix} R & \boldsymbol{p} \\ 0 & 1 \end{bmatrix}$.

1.1.3 Configuration of a rigid poly-articulated system

Poly-articulated systems, also known as rigid mechanisms, are defined by a finite number of rigid bodies interconnected by ideal joints. An ideal joint E_i^j is a kinematic relationship between two rigid bodies B_i and B_j. This relationship can be described in ways such that it either allows or constraints the possible movements between the two bodies it links. It is named ideal if there is no backlash and no friction forces in the joint. Those hypotheses are realistic from a kinematic point of view.

Degrees of freedom Identically to the free body definition, degrees of freedom are the independent possible displacements of a rigid poly-articulated system. A rigid poly-articulated system of n_b bodies has a maximum of $6n_b$ degrees of freedom which can be described using $6n_b$ parameters called primitive parameters and denoted $p_1, p_2, \ldots, p_{6n_b}$. Those $6n_b$ possible movements can be constrained by joints, reducing the number of degrees of freedom of a rigid poly-articulated system to n, with $0 \leq n \leq 6n_b$. In

the holonomic[2], open-chain[3], minimum representation case, $n = 6n_b - n_c$ where $n_c = \sum_{k=1}^{n_b}(n_{c,k})$ with $n_{c,k}$, the number of constrained movements at joint E_{k-1}^{k}. In that case, n is also the minimum number of independent parameters chosen to describe the robot configuration.

Joint space The n independent parameters chosen to describe the configuration of a robot along with the n_c joint equations are sufficient to describe its configuration, i.e., the pose of each body composing it. These parameters can be gathered to form the vector of configuration parameters $\boldsymbol{q}$ also named generalized coordinates. The configuration of a robot is thus defined on an n-dimensional space called *joint space* or *configuration space*.

Operational space While configuration parameters are often chosen in such a manner that they can be intuitively related to the motion of the actuators of the system, it is very difficult to use them to describe the tasks of a robot. In order to facilitate the description of these tasks, an alternative space can be used which is often called *task space* or *operational space*.

The unitary task considered consists in positioning a coordinate frame in space relatively to a reference coordinate frame, i.e., defining the pose of a body. A task can be less constrained, restraining only a few parameters of a pose such as positioning only the origin of a coordinate frame in space. We have chosen to describe a task with a maximum of six parameters gathered to form the desired task space parameters $\boldsymbol{\xi}^{\dagger}$ of size $m^d \leq 6$. Six parameters are sufficient to span $SE(3)$ which is a six dimensional space.

The operational space, or task space, is usually defined from the frame attached to the end-effector(s) of the robot but can be defined from any frame of the robot. The end-effector is the device designed to interact with the environment. While operational space and task space denote the same space, we use the term *operational space* to describe the space associated to the end-effector and the term *task space* to describe the task. More generally, the operational space can be represented by any set of parameters of interest which can be described as a function of $\boldsymbol{q}$. We consider one task space for each end-effector of the robot. The m degrees of freedom of an end-effector are defined as the number of operational space parameters, named $\boldsymbol{\xi}$, that can vary independently to achieve a task. They are dependent of the configuration of the system. We have chosen to describe the pose of an end-effector with six parameters to ensure the capability of the end-effector to perform its associated task.

Forward kinematics At the geometric level, the forward kinematics[4] $\mathcal{FK}$ is a non-linear function such as

$$\boldsymbol{\xi} = \mathcal{FK}(\boldsymbol{q}). \tag{1.7}$$

[2]Holonomic joint equations are integrable and can be expressed as $f(p_1, p_2, \ldots, p_{6n_b}, t) = 0$.
[3]An open-chain mechanism is such that the graph representation of its bodies and joints is acyclic.
[4]in french *Modèle Géométrique Direct* as it is independent of the movement (= *kine*)

It relates the joint space to the operational space as illustrated in Figure 1.1.

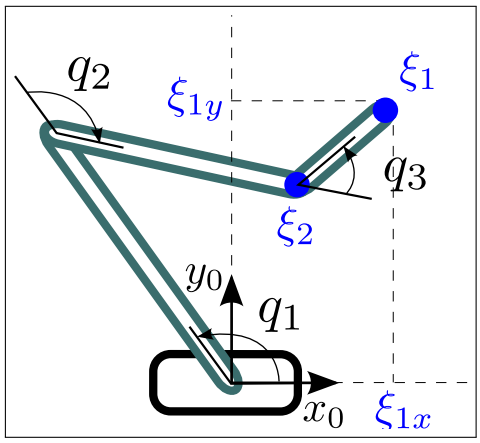


Figure 1.1: Description of a three degrees of freedom planar robot. q_i are the relative angles between two consecutive bodies. ξ_i are the operational space parameters.

In some configurations, called singular configurations, poly-articulated systems can lose degrees of freedom of the end-effector. This may occur for example when the axes of two revolute joints become collinear (see Figure 1.2). Except when explicitly mentioned, robots are considered to be in regular configurations in the remaining of this thesis.

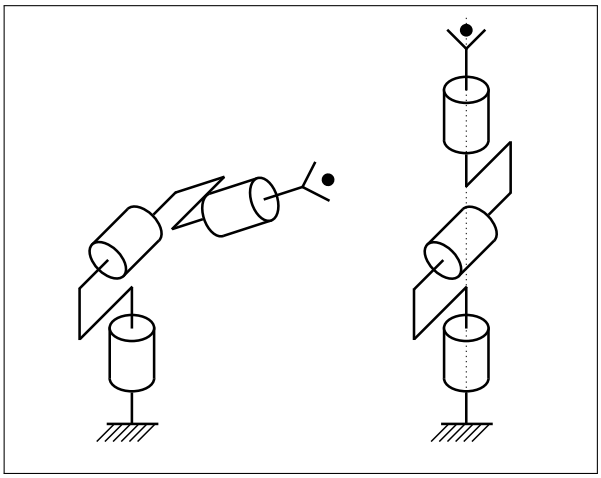


Figure 1.2: Description of a three degrees of freedom robot in a random configuration (left) and in a singular configuration (right).

Task space to joint space mapping Whereas a task is, by definition, easily described in the operational space, it can rarely be described in the joint space in a natural way. By contrast, control is generally expressed in the joint space. Thus, it is useful to have

access to the operational space to joint space mapping. This mapping is the inverse of the forward kinematics $\mathcal{FK}$. We can split this mapping into three cases which differ regarding the size of m and n.

Fully constrained case In the case where m is equal to n, the robot has exactly the number of degrees of freedom to achieve the task. In that case, there is a finite number of inverses to the forward kinematics relation $\mathcal{FK}$ (see Figure 1.3).

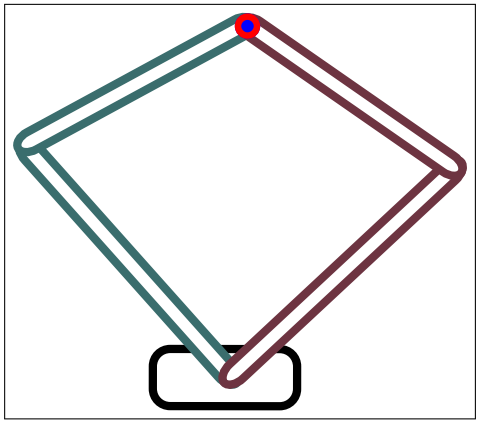


Figure 1.3: Description of a planar two degrees of freedom robot. There are exactly two possible configurations for a desired end-effector position.

Over-constrained case If $m > n$, the mechanical device is said under-actuated relatively to the task. It has less degrees of freedom than those necessary to achieve the task. It can only partially fulfil the task with an approximate solution (see Figure 1.4).

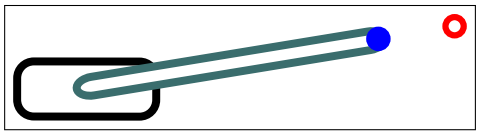


Figure 1.4: Description of a planar one degree of freedom robot. There is a configuration approximating the solution for the problem of positioning the end-effector, disk in blue to the desired end-effector position, ring in red.

Under-constrained case Considering minimum representations for the joint and task spaces, if $m < n$, the system is redundant relatively to the task. It means it has more actuated degrees of freedom than those necessary to perform that task (see Figure 1.5). The system has $n - m$ degrees of redundancy. In that case, there is an infinite number of possible inverses leading to an exact solution to the inversion of the forward kinematics

relation $\mathcal{FK}$. The problem is said ill-posed and thus, it is not possible to find a closed-form solution to this non-linear problem. It is consequently impossible to span the set of possible solutions at the geometric level.

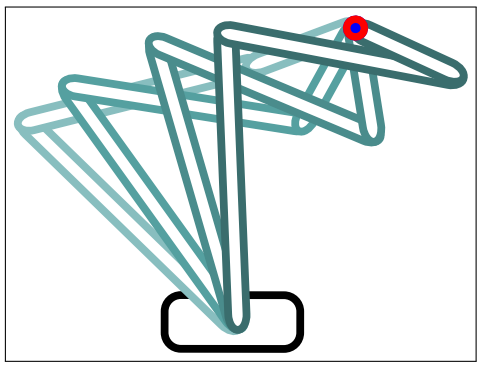


Figure 1.5: Stroboscopic view of a planar three degrees of freedom robot. There is an infinite number of configurations which solve the problem of positioning the end-effector (disk in blue) to the desired end-effector position (ring in red). The robot maintains an end-effector position and can still move.

1.1.4 Velocity of a rigid poly-articulated system

The generalized velocity, or joint space velocity, is described by the derivative of joint space parameters $\boldsymbol{q}$ relatively to time noted $\dot{\boldsymbol{q}}$. The operational velocity of an end-effector is described by a twist $\boldsymbol{T}_{ab}^{c} = \begin{bmatrix} \boldsymbol{\omega}_{ab}^{c} \\ \boldsymbol{\nu}_{ab}^{c} \end{bmatrix}$. The linear velocity $\boldsymbol{\nu}_{ab}^{c}$ of the end-effector is the derivative of the position of the end-effector parameters relatively to time. Concerning the orientation, as mentioned in Selig [2005], there is no minimal representation at the geometric level which continuously spans the $SO(3)$ space. It is thus impossible to compute the derivative of the forward kinematics on the whole space with a minimal operational representation. The homogeneous matrix representation is a non-minimal representation that allows to compute the forward velocity kinematics[5]

$$\boldsymbol{T}_{0i}^{0} = J_{0,i}(\boldsymbol{q})\dot{\boldsymbol{q}} = \left[\sigma_i^1 Ad\left(H_{00}\right) J_{0,1}\left(q^1\right) \cdots \sigma_i^n Ad\left(H_{0n-1}\right) J_{n-1,n}\left(q^n\right)\right] \begin{bmatrix} \dot{q}^1 \\ \vdots \\ \dot{q}^n \end{bmatrix} \tag{1.8}$$

where $J_{0,i}$ is called the Jacobian matrix of body B_i expressed in coordinate frame Ψ_0, and where σ_0^i equals 1 if joint E_i^j is in the path from body B_i to body B_0, for $j \in \{1, \cdots, n\}$ and σ_0^i equals 0 otherwise. The joint E_i^j links the body B_i to the body B_j.

[5] in french *Modèle Cinématique Direct*

It is useful to express the Jacobian matrix $J_{0,i}$ in another coordinate frame Ψ_l, fixed in the world or not

$$J_{l,i} = \begin{bmatrix} R_{l,0} & 0 \\ 0 & R_{l,0} \end{bmatrix} J_{0,i}. \tag{1.9}$$

It is of interest to consider only the position of the end-effector and not its orientation. The derivative of the forward kinematics relatively to time is written

$$\frac{\partial \boldsymbol{\xi}}{\partial t} = \frac{\partial \mathcal{FK}(\boldsymbol{q})}{\partial t} \tag{1.10}$$

or also

$$\frac{d\boldsymbol{\xi}}{dt} = \frac{\partial \mathcal{FK}(\boldsymbol{q})}{\partial \boldsymbol{q}} \frac{d\boldsymbol{q}}{dt}. \tag{1.11}$$

It can be written

$$\boldsymbol{\nu}_{0,i}^{0} = J_{0,i}(\boldsymbol{q})\, \dot{\boldsymbol{q}} \tag{1.12}$$

where $\boldsymbol{\nu}_{0,i}^{0}$, the operational velocity of the point $\boldsymbol{\xi}$, is the linear velocity of the twist $\boldsymbol{T}_{0i}^{0}$. The Jacobian matrix $J_{0,i}(\boldsymbol{q}) = \partial \mathcal{FK}(\boldsymbol{q}) / \partial \boldsymbol{q}$ is a $m \times n$ matrix, called the Jacobian matrix of body B_i expressed in coordinate frame Ψ_0.

Image and kernel A Jacobian matrix J is a linear application from joint space of size n to operational space of size m. We define the image of J as

$$\mathrm{im}(J) = \{\boldsymbol{\nu} \in \mathbb{R}^m, \exists\, \dot{\boldsymbol{q}} \in \mathbb{R}^n : \boldsymbol{\nu} = J\dot{\boldsymbol{q}}\} \tag{1.13}$$

where $\mathrm{im}(J)$, the image of J, is the set of all velocities of the end-effector generated by the set of all joint velocities. The dimension of the image of J is also named the rank of J. The rank is useful to evaluate if a robot is in a singular position or not. If J is full row rank, the robot is not in singularity.

Similarly, we define the kernel of J as

$$\ker(J) = \{\dot{\boldsymbol{q}} \in \mathbb{R}^n : J\dot{\boldsymbol{q}} = 0\} \tag{1.14}$$

where $\ker(J)$ is the set of all joint velocities which do not generate velocities on the end-effector.

A Jacobian matrix J can be related to a task. The kernel of J is not empty if a robot is locally redundant relatively to that task.

1.1.5 Dynamics of a rigid poly-articulated system

Developing mechanical equations of a rigid poly-articulated system written in Newton-Euler or Lagrange formalisms leads to:

$$\boldsymbol{\tau} = A(\boldsymbol{q})\, \ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\epsilon}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{\tau}^{ext} \tag{1.15}$$

where $A(\boldsymbol{q})$, $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, $\boldsymbol{g}(\boldsymbol{q})$, $\boldsymbol{\epsilon}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and $\boldsymbol{\tau}^{ext}$ are respectively the $n \times n$ inertia matrix of the system, the vector of the non-linear Coriolis and centrifugal effects, the vector of gravity effects, the vector of unmodelled effects and the torques resulting from external forces applied to the system. $\boldsymbol{\tau}$ is the vector representing joint torques.

The inertia matrix $A(\boldsymbol{q})$ is square, semi-definite, positive, and thus is easily invertible.

The unmodelled dynamical effects $\boldsymbol{\epsilon}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ come from viscous or static frictions or from backlash induced by misalignments during the assembly of the robot. Many friction models have been developed [Canudas de Wit et al., 1995] which depend on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. We have chosen to merge all those parameters into the unmodelled effects.

The dimension of the torque vector $\boldsymbol{\tau}$ equals the number of degrees of freedom n.

Equation (1.15) may be written

$$\boldsymbol{\tau} = A(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{\tau}^{ext} \tag{1.16}$$

where $\boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\epsilon}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the sum of all non-linear effects.

For a better understanding, Equation (1.15) is noted

$$\boldsymbol{\tau} = \mathcal{ID}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{\tau}^{ext}\right). \tag{1.17}$$

For a fully-actuated system which does not contain redundant actuators, Equation (1.17) represents a joint space to joint space mapping at the dynamics level with only one solution.

For more details about inverse dynamics, interested readers can refer to Khalil and Dombre [2002].

1.2 Model-based control

The robot being controlled at the joint level, the operational space control approach requires the knowledge of the mapping between the joint space and the task space. More specifically, it is the inverse mapping which is often of interest: given a task, what are the actions required in the joint space to achieve it?

The inverse dynamics model relates forces and torques applied on a system to its joint positions, velocities and accelerations. It is necessary to include dynamics in a control law. It is of interest to control the dynamics to compensate for undesired effects. For example, it is useful to compensate for gravity effects that modify the efficiency of the controller, inducing a deviation of the controlled end-effector towards the gravity direction. It is also useful to compensate for friction efforts in each joint to control them identically. Controlling dynamics also allows to compensate for inertia and precisely control the robot, avoiding overshoot. One way to include dynamics in a control law is to separate kinematics and dynamics and to use the inverse of the models at both levels to control the system.

A hierarchical control loop, illustrated in Figure 1.6, is divided into three steps. A first planning step consists in specifying a task $\boldsymbol{\xi}^\dagger$ and transforming it into an operational acceleration. A second kinematic step transforms the operational desired acceleration into a desired joint acceleration. A third step uses the inverse dynamics (cf. Section 1.1.5) to compute torques.

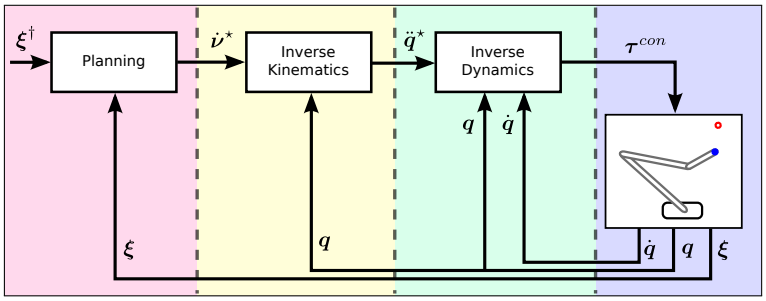


Figure 1.6: Classical control loop using planning, inverse velocity kinematics and inverse dynamics.

1.2.1 Task controller

The task controller module specifies a trajectory which should be followed by an end-effector in order to reach its associated task $\boldsymbol{\xi}^\dagger$. A trajectory is defined by a composition of a pose along time, a twist and an acceleration of a frame, linear and angular, or one of these components. An example of such trajectory can be a circle where the end-effector is maintained vertical. Another trajectory can just be defined by a straight line up to the goal without controlling the orientation.

To follow such a trajectory, it is necessary to discretize it into small steps. Those small steps can be considered as subtasks that are closer to the current position and that must be reached in sequence.

A lower level controller transforms a goal to reach into an operational acceleration using a corrector based on the error between the goal and the current pose of the controlled end-effector. This goal can be either a final desired position $\boldsymbol{\xi}^\dagger$ or a point evolving along a discrete desired trajectory.

As described in Park and Ravani [1997], it is possible to transform that error in pose into an equivalent twist that must be applied, as input control on the robot, to reach the goal. It is also possible to compute an input control operational acceleration using an error based on quaternions or Euler angles.

Low level controllers If we consider a goal $\boldsymbol{\xi}^\dagger$ as a final desired position to reach, and not an orientation, the input control operational acceleration $\dot{\boldsymbol{\nu}}^\star$ can be computed using

the task space parameters error

$$\dot{\boldsymbol{\nu}}^{\star} = K_p \left(\boldsymbol{\xi}^{\dagger} - \boldsymbol{\xi} \right), \tag{1.18}$$

where K_p is a symmetric positive definite matrix and $\boldsymbol{\xi}^{\dagger}$ denotes the desired position of the task space parameters. This corrector is called proportional. It can be seen as a virtual spring of stiffness K_p acting between the goal and the end-effector.

Another controller is the Proportional Derivative ($\mathcal{PD}$) controller which computes an acceleration based on two terms: one is proportional to the error in position, the other is proportional to the error in velocity

$$\dot{\boldsymbol{\nu}}^{\star} = K_p \left(\boldsymbol{\xi}^{\dagger} - \boldsymbol{\xi} \right) + K_d \left(\boldsymbol{\nu}^{\dagger} - \boldsymbol{\nu} \right), \tag{1.19}$$

where K_d is a symmetric positive definite matrix and $\boldsymbol{\nu}^{\dagger}$ denotes the velocity that should be applied to fulfil the task. That controller can be seen as a spring between the goal and the end-effector coupled with a damper with a K_d damp coefficient.

The Proportional Integral Derivative ($\mathcal{PID}$) controller computes an acceleration based on the sum of three terms: the first is proportional to the error in position, the second is proportional to the error in velocity, the third is proportional to the integral of the error in position along time.

$$\dot{\boldsymbol{\nu}}^{\star} = K_p \left(\boldsymbol{\xi}^{\dagger} - \boldsymbol{\xi} \right) + K_d \left(\boldsymbol{\nu}^{\dagger} - \boldsymbol{\nu} \right) + K_i \int_0^t \left(\boldsymbol{\xi}^{\dagger} - \boldsymbol{\xi} \right) dt, \tag{1.20}$$

where K_i is a symmetric positive definite matrix and t denotes the current time. This integral corrector compensates for static errors.

A more sophisticated input control computation can include a feed-forward term $\dot{\boldsymbol{\nu}}^{\dagger}$

$$\dot{\boldsymbol{\nu}}^{\star} = \dot{\boldsymbol{\nu}}^{\dagger} + K_p \left(\boldsymbol{\xi}^{\dagger} - \boldsymbol{\xi} \right) + K_d \left(\boldsymbol{\nu}^{\dagger} - \boldsymbol{\nu} \right) + K_i \int_0^t \left(\boldsymbol{\xi}^{\dagger} - \boldsymbol{\xi} \right) dt. \tag{1.21}$$

This term anticipates the desired acceleration instead of being subjected to constantly trying to cancel a position error.

1.2.2 Inverse kinematics

To control the task space velocity, we need to define the task space to joint space relationship

$$\boldsymbol{q} = \mathcal{IK}\left(\boldsymbol{\xi} \right). \tag{1.22}$$

We have seen that a robot is redundant if it owns more degrees of freedom than those necessary to perform a task. That redundancy cannot be controlled at the geometric level. However, it is possible to span the set of all possible solutions at the velocity

level as the relationship is linear with respect to joint space parameters. This section describe how inverse velocity kinematic model, also named named Jacobian matrix, can be inverted using linear algebra techniques.

Inverting velocity kinematics

In the fully constrained case, m is equal to n, the robot has exactly the number of actuated joints necessary to perform the task and the Jacobian matrix is square. If J is full row rank, the robot is not in singularity and the model is easily invertible

$$\dot{\boldsymbol{q}} = {J_i}^{-1}(\boldsymbol{q})\,\boldsymbol{\nu}_{0,i}. \tag{1.23}$$

For the sake of clarity, we delete the reference coordinate frame which can be arbitrarily chosen according to necessities. It can for example be related to the ground or to the visual referential.

The case where $m > n$ describes an under-actuated robot which is out of the scope of this thesis.

In the under-constrained case, $m < n$, the Jacobian matrix is not square and is not classically invertible. The system has more degrees of freedom than the end-effector, there is an infinite number of inverse mappings and the robot is said redundant with respect to the task. In the non-singular case, i.e., $\operatorname{rank}(J(\boldsymbol{q})) = m$, there is an infinite number of generalised inverses of $J(\boldsymbol{q})$ written $J^{\sharp}(\boldsymbol{q})$ (see Ben Israel and Greville [2003] for details)

$$\dot{\boldsymbol{q}} = J_i^{\sharp}(\boldsymbol{q})\,\boldsymbol{\nu}_{0,i}. \tag{1.24}$$

Among these inverses, weighted pseudo-inverses [Doty et al., 1993; Park et al., 2001] provide minimum norm solutions. If $\operatorname{rank}(J(\boldsymbol{q})) = m$, they can be written as

$${J_i}^{W+}(\boldsymbol{q}) = W^{-1}{J_i}^{T}(\boldsymbol{q})\left[J_i(\boldsymbol{q})\,W^{-1}{J_i}^{T}(\boldsymbol{q})\right]^{-1}, \tag{1.25}$$

where W is a symmetric and positive definite matrix of dimension $n \times n$. The Moore-Penrose inverse or pseudo-inverse ${J_i}^{+}(\boldsymbol{q})$ corresponds to the case where $W = I_n$.

Singular Value Decomposition

It is possible to decompose J using the Singular Value Decomposition (SVD) method [Golub and Van Loan, 1996]. The SVD of any Jacobian matrix J is given by

$$J = UDV^T \tag{1.26}$$

where U and V are orthogonal matrices of dimensions $m \times m$ and $n \times n$ respectively. D is an $m \times n$ diagonal matrix with a diagonal composed of the m singular values of J in decreasing order. The singular values are the square roots of the eigenvalues of the

matrix $J^T J$ in the case of redundant robots. The columns of V are the eigenvectors of $J^T J$ which correspond to the eigenvalues of D. They are called right singular vectors.

Using the SVD formalism, Equation (1.12) can be rewritten

$$\boldsymbol{\nu} = UDV^T \dot{\boldsymbol{q}} \tag{1.27}$$

Efficient computation of ${J_i}^+(\boldsymbol{q})$ can be performed using the SVD of $J(\boldsymbol{q})$. Given this decomposition, the pseudo-inverse of J can be computed as follows

$$J^+ = VD^+U^T, \tag{1.28}$$

where the computation of D^+ is straightforward given its diagonal nature. A weighted extension of the SVD can be used in the case where $W \neq I_n$. Details about this extension can be found in Ben Israel and Greville [2003].

It is also possible to decompose J using the NIPALS algorithm described in Appendix A.3.

Control

Whitney [1969] calls *Resolved Motion Rate Control* (RMRC) the method which consists in using the inverse velocity kinematics to control the end-effector of a mechanical device. Given a desired task velocity $\boldsymbol{\nu}^\star_{0,i}$, the inverse mapping of Equation (1.12) which minimises the Euclidean norm[6] of the solution is given by

$$\dot{\boldsymbol{q}}^\star = {J_i}^+(\boldsymbol{q})\,\boldsymbol{\nu}^\star_{0,i}. \tag{1.29}$$

Equation (1.29) gives the desired joint velocities necessary to perform a task described by operational velocities $\boldsymbol{\nu}^\star_{0,i}$.

1.2.3 Redundancy resolution schemes

When the robot has more degrees of freedom than those necessary to perform the task, the supplementary degrees of freedom can be used to control the *redundancy* which define the *set of all the possible movements which do not interfere with the task*. The control of that redundancy can be useful to fulfil supplementary goals such as controlling another end-effector, limiting torques, optimising an energy criterion or the distribution of the efforts, avoiding geometrical singularities, avoiding obstacles, getting far from limits [Nakamura, 1990]. To understand how to control that redundancy, we first define the concept of orthogonal projector.

[6] $\sqrt{\dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}}}$, also noted $||\dot{\boldsymbol{q}}||$.

Projector

In the SVD decomposition, the first m columns of V form an orthonormal basis for the subspace of $\dot{\boldsymbol{q}}$ generating an end-effector velocity. The $m+1$ to n columns of V form an orthonormal basis for the null space of $J(\boldsymbol{q})$, i.e., the subspace of $\dot{\boldsymbol{q}}$ which does not generate any end-effector velocity. It is possible to define a projector $P_J(\boldsymbol{q})$ as

$$P_J(\boldsymbol{q}) = [\boldsymbol{v}_{m+1} \dots \boldsymbol{v}_n] [\boldsymbol{v}_{m+1} \dots \boldsymbol{v}_n]^T, \tag{1.30}$$

where $\boldsymbol{v}_i$ is the i^{th} column of V. This projector $P_J(\boldsymbol{q})$ allows to span the null space of the linear application defined by J. Another commonly used expression for $P_J(\boldsymbol{q})$ is

$$P_J(\boldsymbol{q}) = \left(I_n - J^{\sharp}(\boldsymbol{q}) J(\boldsymbol{q})\right). \tag{1.31}$$

This method is computationally more expensive than the one defined in Equation (1.30) as it is necessary to compute the whole inverse of the Jacobian matrix using for example the SVD method. Moreover, as this method depends of the rank of J, it becomes problematic if the system loses a degree of redundancy, e.g., in the singularity case. In such cases, it is easy to consider this rank reduction by deleting corresponding vectors in Equation (1.30).

Controlling redundancy

As Equation (1.24) is not the unique solution to the inverse mapping problem, it is of interest to compute other solutions giving rise to internal motions that do not induce any perturbation to the task. This particular subset of solutions corresponds to the null space of $J(\boldsymbol{q})$ and the general form of the minimum norm solutions to Equation (1.12) can be written

$$\dot{\boldsymbol{q}}^{\star} = {J_i}^{+}(\boldsymbol{q}) \boldsymbol{\nu}_{0,i}^{\star} + P_{J_i}(\boldsymbol{q}) \dot{\boldsymbol{q}}_0^{\star}, \tag{1.32}$$

where $P_{J_i}(\boldsymbol{q})$ is a projector on the null space of $J_i(\boldsymbol{q})$ and $\dot{\boldsymbol{q}}_0^{\star}$ is any vector of dimension n. Equation (1.32) is the minimum norm solution that minimises $||\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_0||$.

Local Optimization

Different possible redundancy resolution schemes can be used in control, depending on the compatibility of the tasks or constraints which have to be solved. It is possible to optimize locally the configuration of the robot using the opposite of the gradient of a cost function $Q(\boldsymbol{q})$ with respect to $\boldsymbol{q}$. The resulting solution leads to the local optimization of the cost function as long as this secondary constraint does not induce any perturbation on the first task. The general form of this solution is written

$$\dot{\boldsymbol{q}} = J_i^{+}(\boldsymbol{q}) \boldsymbol{\nu}_{0,i}^{\star} + \alpha P_{J_i} \nabla Q(\boldsymbol{q}), \tag{1.33}$$

where α is a scalar used to tune the steepness of the gradient descent [Snyman, 2005]. It can be viewed as a learning rate. When $\alpha > 0$, the equation maximizes this optimization term whereas when $\alpha < 0$, it minimizes it.

This method is often used in the incompatible case, i.e., when it is known in advance that the task will not be perfectly achieved but when only a global trend has to be followed such as minimising the kinetic energy of the system, avoiding joint limits or setting the robot to a referent configuration.

Realising multiple tasks

The redundancy resolution scheme presented above allows to perform simultaneously different tasks with different end-effectors. It is possible to achieve all tasks only if the robot has enough degrees of freedom to accomplish them. Let us consider two tasks of respective dimensions m_a and m_b with associated Jacobian matrices J_a and J_b such that $rank\left(J_a\left(\boldsymbol{q}\right)\right) = m_a$ and $m_a \leq n$ and $rank\left(J_b\left(\boldsymbol{q}\right)\right) = m_b$ and $m_b \leq n$. These two tasks are said compatible if $J_{ext} = \left[J_a^T \; J_b^T\right]^T$ is full row rank. This condition is equivalent to saying that the m_{ext} parameters of the augmented task space are in minimum number and that $rank\left(J_{ext}\right) \leq n$. Given this definition, one has to consider the under-constrained (compatible, infinity of solutions), fully-constrained (compatible, one solution) and over-constrained (incompatible, no exact solution) cases. In these three cases, one can solve the inverse velocity kinematic problem using the solution proposed initially by Maciejewski and Klein [1985]

$$\dot{\boldsymbol{q}}^\star = J_a^+ \boldsymbol{\nu}_{0,a}^\star + \left(J_b P_{J_a}\right)^+ \left(\boldsymbol{\nu}_{0,b}^\star - J_b J_a^+ \boldsymbol{\nu}_{0,a}^\star\right). \tag{1.34}$$

In the compatible case, the tasks a and b will be achieved perfectly. In the incompatible case, task a will also be perfectly achieved whereas the error in the achievement of task b will be minimised [Nakamura, 1990]. This solution can present singularities when tasks are highly incompatible, i.e., when m_{ext} is much greater than n, but this can be compensated for using a proper damped-least square regularization method [Chiaverini, 1997]. This task projection scheme can be extended to several tasks [Mansard and Chaumette, 2007].

Another method, originally proposed in Baillieul [1985], consists in writing an extended Jacobian matrix $J_{ext} = \begin{bmatrix} J_a \\ J_b \end{bmatrix}$ in order to reach the fully constrained case ($m_{ext} = n$ and $rank(J_{ext}) = m_{ext}$) and thus to simplify the inversion problem to a square, regular matrix inversion.

$$\dot{\boldsymbol{q}} = J_{ext}^{-1} \begin{bmatrix} \boldsymbol{\nu}_{0,a}^\star \\ \boldsymbol{\nu}_{0,b}^\star \end{bmatrix}. \tag{1.35}$$

If the tasks are compatible and the matrix is full rank, this is achieved automatically. If the tasks are incompatible, some projections on the kernel of the first Jacobian matrix

are necessary to ensure both a square Jacobian matrix and priorities between tasks

$$\dot{\boldsymbol{q}} = \begin{bmatrix} J_a \\ P_{J_a} J_b \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\nu}^\star_{0,a} \\ \boldsymbol{\nu}^\star_{0,b} \end{bmatrix}. \tag{1.36}$$

1.2.4 Inverse dynamics in control

The third step of a hierarchical control loop is the control of the dynamics. In this section, we describe two different control loop used in robotics: the decentralized control and the dynamic decoupling.

Decentralized control

Controlling the dynamics means calculating the torques necessary to accelerate the system in a desired way. This calculation, which depends on the current state of the system, can be performed by a $\mathcal{PID}$ controller. With such a controller, this method, named decentralized control, can easily be applied to any system but requires to tune different parameters [Voda and Landau, 1992, 1995] which are, most of the time, fixed on the whole space to ensure stability. As those parameters do not depend on the configuration, they are suboptimal.

Figure 1.7 illustrates a classical control loop using a $\mathcal{PID}$ controller as low level controller on each joint.

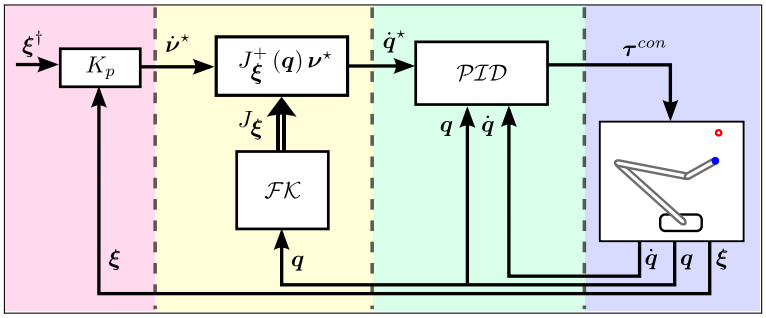


Figure 1.7: Classical control loop using inverse velocity kinematics and $\mathcal{PID}$ controller as inverse dynamics.

Decentralized control is used on many robots as low level controllers are usually already developed for each motor. Those controllers are often considered to belong to the robot and, in most cases, cannot be tuned. Such controllers are locally stable independently on each articulation but cannot ensure a global stability for the whole robot. For complex robots such as humanoid robots, the global stability can be enhanced by tuning each controller on each joint independently and adequately to the tasks they are assigned to. The controllers of the joints of a hand can be for example tuned smoother than the ones controlling the ankle.

Dynamics decoupling

Dynamics decoupling, a more sophisticated dynamics controller, leads to invert the dynamics of the robot itself and includes it in the control loop. It can be understood as imposing a desired acceleration through the inverse dynamics model to obtain torques to apply at the joint level in order to reach the desired acceleration. Equation (1.17) represents the inverse dynamics of a poly-articulated system. It is of interest to compute this mapping since, in any given state $(\boldsymbol{q}, \dot{\boldsymbol{q}})$, it gives the torques $\boldsymbol{\tau}^{con}$ to be applied by the actuators of the robot to obtain the desired acceleration $\ddot{\boldsymbol{q}}^{\star}$ at the next step

$$\boldsymbol{\tau}^{con} = \mathcal{ID}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}^{\star}, \boldsymbol{\tau}^{ext}\right). \tag{1.37}$$

For a fully-actuated system, the dimension of the actuation torque vector $\boldsymbol{\tau}^{con}$ equals the number of degrees of freedom n.

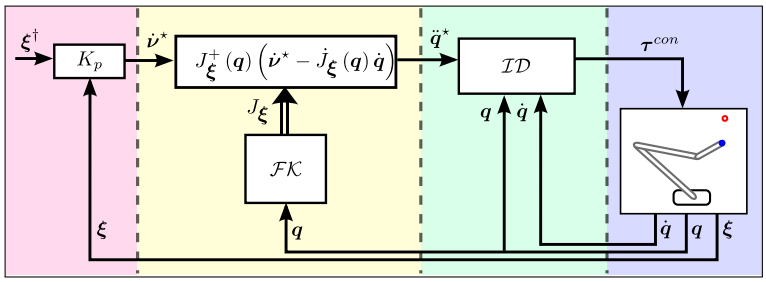


Figure 1.8: Classical control loop using both inverse velocity kinematics and inverse dynamics.

The inverse kinematics in acceleration comes from the derivation of Equation 1.29 relatively to time

$$\ddot{\boldsymbol{q}}^{\star} = J^{+}(\boldsymbol{q})\left(\dot{\boldsymbol{\nu}}^{\star} - \dot{J}(\boldsymbol{q})\dot{\boldsymbol{q}}\right). \tag{1.38}$$

The dynamic simulation can be summed up by Algorithm 1.

Algorithm 1 Simulation loop (acceleration)

$\dot{\boldsymbol{\nu}}^{\star} = K_p\left(\boldsymbol{\xi}^{\star} - \boldsymbol{\xi}\right)$	*desired operational velocity*
$\ddot{\boldsymbol{q}}^{\star} = J^{+}(\boldsymbol{q})\left(\dot{\boldsymbol{\nu}}^{\star} - \dot{J}(\boldsymbol{q})\dot{\boldsymbol{q}}\right)$	*desired joint velocity*
$\boldsymbol{\tau}^{con} = \mathcal{ID}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}^{\star}, \boldsymbol{\tau}^{ext}\right)$	*desired torques*
$\ddot{\boldsymbol{q}} \leftarrow \mathcal{D}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}^{con}, \boldsymbol{\tau}^{ext}\right)$	*simulator integration*

1.2.5 Operational space control

An alternative formulation of inverse dynamics was first proposed by Khatib [1983] [7] which expresses the dynamics of the system in the task space

$$\boldsymbol{f} = \Lambda(\boldsymbol{q})\,\dot{\boldsymbol{\nu}} + \boldsymbol{\mu}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{p}(\boldsymbol{q}) + \bar{\boldsymbol{\epsilon}}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{f}^{ext} \tag{1.39}$$

where $\Lambda = (JA^{-1}J^t)^{-1}$, $\boldsymbol{\mu} = \Lambda(JA^{-1}\boldsymbol{c} - \dot{J}\dot{\boldsymbol{q}})$ and $\boldsymbol{p} = \Lambda JA^{-1}\boldsymbol{q}$. The terms $\Lambda(\boldsymbol{q})$, $\boldsymbol{\mu}(\boldsymbol{q},\dot{\boldsymbol{q}})$, $\boldsymbol{p}(\boldsymbol{q})$ and $\bar{\boldsymbol{\epsilon}}(\boldsymbol{q},\dot{\boldsymbol{q}})$ are the projection in the task space of the inertia matrix, the Coriolis and centrifugal forces, the gravity forces and the unmodeled effects forces. $\boldsymbol{f}^{ext}$ is the vector of external forces applied to the system and expressed at the end-effector level.

Equations (1.39) can be written in a more general manner as

$$\boldsymbol{f} = \mathcal{IOD}\left(\boldsymbol{q},\dot{\boldsymbol{q}},\dot{\boldsymbol{\nu}},\boldsymbol{f}^{ext}\right). \tag{1.40}$$

Equation (1.39) relates the task space acceleration vector, of dimension smaller than or equal to 6, to the joint torques vector, of dimension equal to the number of actuators needed to accomplish the task.

It is possible to use the Jacobian matrix, defined in Equation (1.8), to compute the torques acting on each articulation resulting from the application of a force to a poly-articulated system.

$$\boldsymbol{\tau}^{con} = J(\boldsymbol{q})^T\,\boldsymbol{f} \tag{1.41}$$

Equations (1.39) and (1.41) can be summed in one

$$\boldsymbol{\tau}^{con} = J(\boldsymbol{q})^T\left(\Lambda(\boldsymbol{q})\,\dot{\boldsymbol{\nu}}^{\star} + \boldsymbol{\mu}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{p}(\boldsymbol{q}) + \bar{\boldsymbol{\epsilon}}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{f}^{ext}\right). \tag{1.42}$$

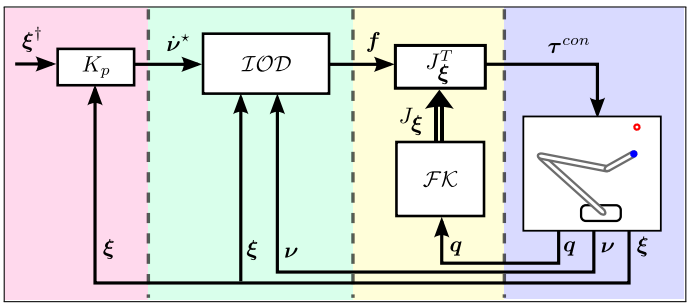


Figure 1.9: Control loop including inverse operational dynamics.

In contrast with the control scheme illustrated in Figure 1.8, the one illustrated in

[7] Readers can refer to Sentis and Khatib [2005] for a detailed presentation and Nakanishi et al. [2008] for a recent survey.

Figure 1.9 correlates the velocity kinematic level and the dynamics level. The control loop is closed at the dynamics level in the operational space.

With this method, it is also possible to control redundancy with the inverse kinematic methods presented in Section 1.2.3.

$$\boldsymbol{\tau}^{con} = J\left(\boldsymbol{q}\right)^{T} \boldsymbol{f} + \left(I - J\left(\boldsymbol{q}\right)^{T} \left(J\left(\boldsymbol{q}\right)^{T}\right)^{\sharp}\right) \boldsymbol{\tau}_{0} \tag{1.43}$$

The difference resides in the expression of the dynamics. While the inverse dynamics expressed in the joint space is unique for the robot, the inverse dynamics expressed in the task space differs for each end-effector. It is thus necessary to compute an inverse dynamics model for each task if we want to control the redundancy of the robot. Moreover, for each task, we have to determine its associated Jacobian matrix to compute the control torques from the force expressed in task space.

1.3 Conclusion

In this chapter, we have presented the formalism used in the reminder of this thesis to describe poly-articulated systems. We explained how to combine kinematics and dynamics in a control loop, insisting on the difficulties raised by inverting kinematics to control redundant systems. We have described how a system can be controlled in task space by applying adequate torques. Different planning and inverse kinematics schemes have been presented and compared, building the foundations for the approach proposed in Chapter 3 of this thesis. Particularly, we have detailed how to control redundancy in a realistic dynamic simulation loop.

2 Machine learning in Robotics

Contents

We have seen in Chapter 1 how models can be used to control redundant robotic systems. This chapter presents the state of the art methods used to learn models necessary to control robotic systems. In the first section, we briefly explain basic concepts of supervised learning methods used to approximate non-linear functions. We start from artificial neural networks approximating non-linear functions before presenting two more recent statistical learning algorithms used in this thesis. Then, in the second part of this chapter, we mention how supervised learning methods are used to learn mechanical models in the Robotics literature. We focus on the nature of the models which are learnt and on the problems induced by choosing such or such formalism. We conclude this section by showing that none of these works solve the constraints exposed in the introduction which consist in controlling explicitly the redundancy of a poly-articulated system with learnt models over all the reachable space of the robot.

2.1 Supervised learning methods

Supervised learning, a class of machine learning methods, is used when a supervisor provides an input/output pair that can be used directly by some entity to learn a mapping between the corresponding training data. In supervised learning, the classification problem consists in predicting the class of the presented input whereas the regression problem consists in approximating linear and non-linear functions by predicting continuous values.

In this section, we present supervised learning methods such as multi-layer perceptrons and radial basis function networks which help understanding locally weighted regression methods such as LWPR [Vijayakumar and Schaal, 2000]. We also present XCSF, a method that shares some similarities with LWPR though it comes from a very different lineage, LWPR and XCSF being used in the second part of this thesis.

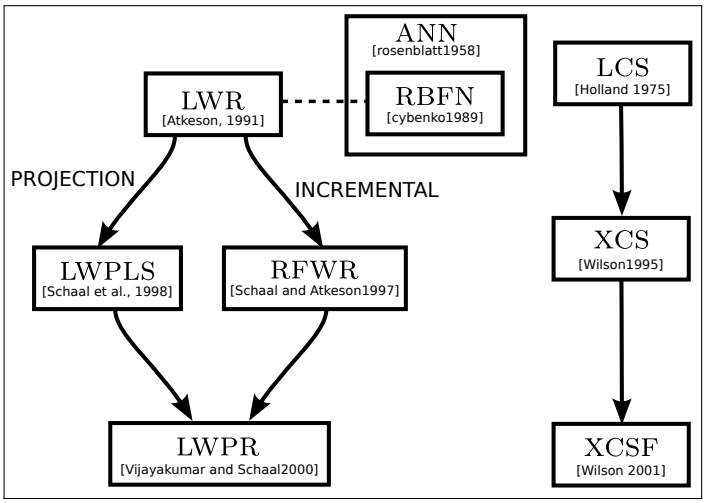


Figure 2.1: Evolution of supervised learning algorithms of interest in this thesis.

2.1.1 Artificial neural networks

Artificial neural networks (ANN) are a wide class of general function approximators. They are declined under different forms.

A commonly used form of feed-forward ANN is the Multi-Layer Perceptron (MLP) [Rosenblatt, 1958] (see Figure 2.2) where the excitation level $v_i(\boldsymbol{x})$ of each neuron i is calculated as

$$v_i(\boldsymbol{x}) = \sum_{i=1}^{n_n} \beta_i x_i$$

where n_n is the number of neurons, x_i is an input of the neuron i and β_i is the associated linear weight. The activation function $y_i = f(v_i)$ uses the excitation level to determine if the neuron fires or not. y_i is the output of the neuron i. Classical activation functions are $y_i = \tanh(v_i)$ or $y_i = (1 + e^{-v_i})^{-1}$.

The weights may be updated through a back-propagation error method based on the error δy_i between the desired value and the real output of each neuron i.

$$\delta\beta_{ij} = -\alpha\delta y_i \frac{\partial f(v_j)}{\partial v_j}. \tag{2.1}$$

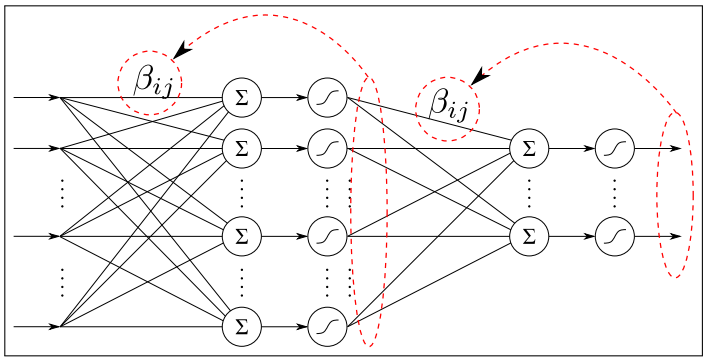


Figure 2.2: The ANN algorithm may be split into two steps: a learning and a predicting step. During the prediction step (black, plain lines), the network computes an excitation level for each neuron depending on the input and applies an activation function on this excitation level to determine if the neuron is active or not. During the learning period (red, dashed lines), it computes an error of the global network output to update all weighted connections.

where β_{ij} is the linear weight between the input x_i and the neuron i, α is the learning rate.

2.1.2 Radial basis function networks

Radial Basis Function Networks (RBFN), illustrated in Figure 2.3, are another type of ANN where weighted sum and activation functions are temporally inverted compared to multi-layer perceptrons [Cybenko, 1989].

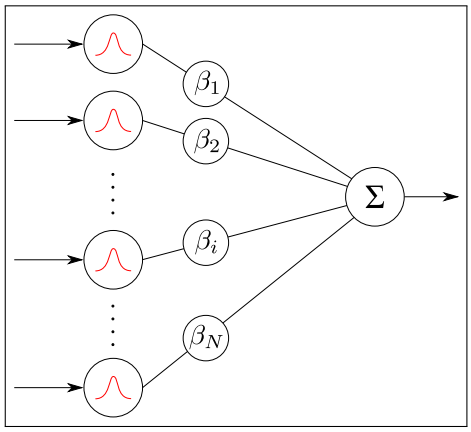
A Radial Basis Function (RBF) ϕ is a function whose output value y, also called activation function, depends only on the distance r between its center c and the input value x. The input value x is copied in a vector $\boldsymbol{x}$ as many times as necessary to compute the error with the corresponding vector of centres $\boldsymbol{c}$:

$$\boldsymbol{r} = \boldsymbol{x} - \boldsymbol{c}.$$

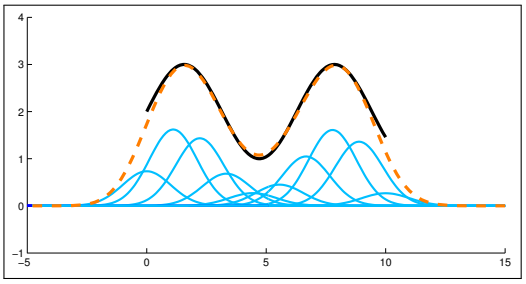
A standard RBF is a Gaussian function defined as $\phi(r_i) = e^{-D_i r_i^2}$ where D_i, the smoothing parameter, is a positive distance metric which determine the size and the shapes of the Gaussian i. It is the inverse of the squared variance of the Gaussian. The output of all functions are weighted and summed together to get the global network output

$$\hat{y} = \sum_{i=1}^{n_n} \beta_i \phi(r_i) \tag{2.2}$$

where n_n is the number of neurons and the β_i are the linear weights.



(a) Schematic view of RBFN



(b) Approximation (in doted orange) of a sinus function (top in black) with an RBFN (bottom in cyan).

Figure 2.3: The RBFN prediction algorithm may be split into three steps. First, apply an activation function on the input to determine if the neuron is active or not. Second, compute an output, also called excitation level, for each neuron depending on its weight and activation. Third, sum the output of each neuron.

The *learning* process consists in computing an error of the global network output to update all weighted connections. Weights are updated incrementally with different methods such as gradient descent depending on the error between the output of the network and the desired value as

$$\delta\beta_i = -\alpha\delta y_i \frac{\partial\phi(r_i)}{\partial v_i} \tag{2.3}$$

where α is the learning rate and $\delta y_i = \hat{y}_i - y_i$.

ANN are considered as black boxes because tuning some parameters such as D_i, c_i or α is usually empirical and the relation between neurons has no understandable meaning. RBFNs could be treated as grey boxes as they have the advantage of generating differentiable continuous approximations [Sun and Scassellati, 2004].

2.1.3 Locally weighted regression

Locally Weighted Regression (LWR) methods [Atkeson, 1991] combine the Gaussian validity of RBFNs with regression[1] (see Figure 2.4). This method requires to store relevant data to compute a regression around an input vector $\boldsymbol{x}$ to approximate a scalar output y. The relevant data are chosen among previous points $\left[\boldsymbol{x}_1 \cdots \boldsymbol{x}_{n_p}\right]$ where n_p is the number of encountered points.

[1] More details on regression are presented in Appendix A

Around each point $\boldsymbol{x}$, the region of validity, also called *receptive field*, is computed with a Gaussian function $\phi_k = e^{-\frac{1}{2}d_k}$ where $d_k = \sigma(\boldsymbol{x}_k - \boldsymbol{x})^2$ is the Mahalanobis distance [Mahalanobis, 1936]. This distance depends on σ, the inverse of the variance of the Gaussian functions and $k = 1 \cdots n_p$.

LWR globally approximates non-linear functions with local linear models. The computation of the linear models β is performed by a weighted pseudo-inverse using training data

$$\beta = X\boldsymbol{\phi}^{+}\boldsymbol{y} \tag{2.4}$$

where $X = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_p]$, $\boldsymbol{y} = [y_1 \cdots y_p]$ and $\boldsymbol{\phi} = [\phi_1 \cdots \phi_p]$. See appendix B.5 for an explanation on weighted pseudo-inverse. As presented in Appendix A, β can also be computed recursively.

It is thus possible to predict an output $\hat{y}$ with the model β

$$\hat{y} = \beta\boldsymbol{x}. \tag{2.5}$$

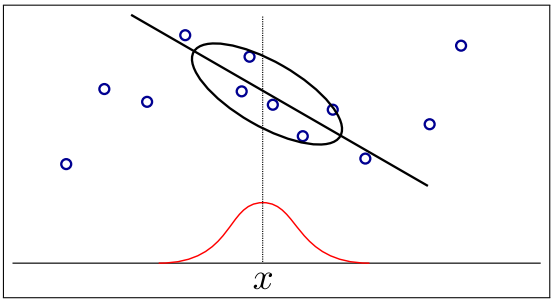


Figure 2.4: In LWR, x is the input of a function we want to approximate. A Gaussian, centred in x, weights stored point with respect to their proximity to x. A regression is then computed using the weighted points [Cohn et al., 1996].

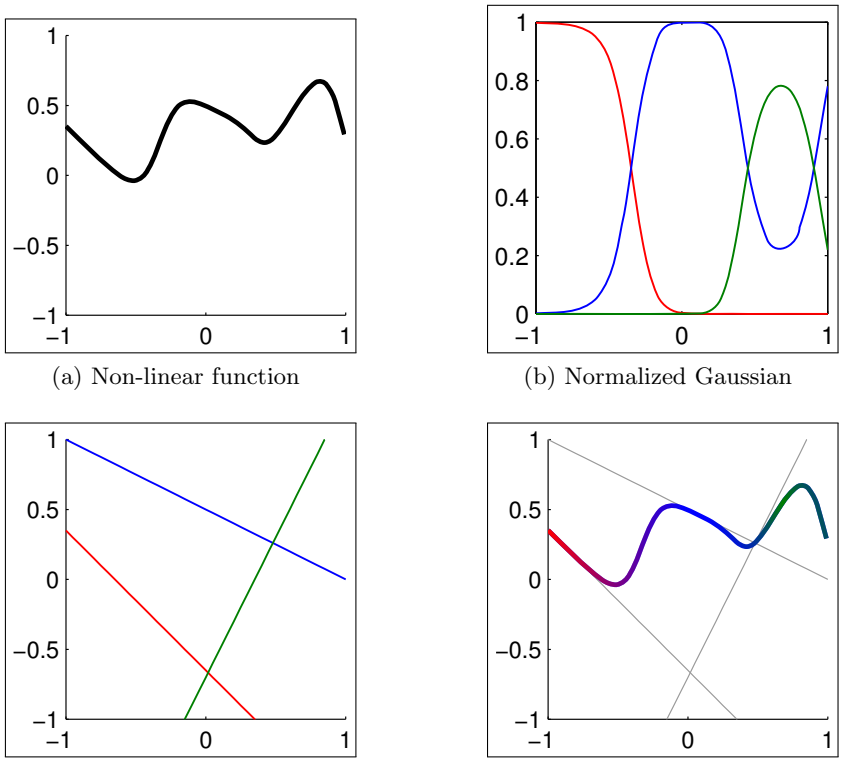
Compared to ANN or RBFN, LWR is a memory-based method as it retains training data and uses them to create its models when needed. One limitation of LWR is that, when we try to approximate a function in a large space, the number of retained points increases with experience, spending more and more memory. Recent methods, described below, tend to avoid this limitation by only keeping relevant points and creating local models around them. Those methods also use projection mechanisms to neglect irrelevant dimensions and incremental statistical algorithms to minimize computational cost.

2.1.4 Locally weighted projection regression

As described in Schaal et al. [2002], lots of models have followed LWR, such as LWPLS [Schaal et al., 1998] which reduces the input dimensionality or RFWR [Schaal and Atkeson,

1997] which transforms the algorithm into an incremental regression method, avoiding to store data.

Locally Weighted Projection Regression (LWPR) [Vijayakumar and Schaal, 2000], illustrated Figure 2.5, is an algorithm which performs simultaneously incremental regression like RFWR and input projection like LWPLS. It is a function approximator providing accurate approximations in large spaces while giving performance metric.



(a) Non-linear function

(b) Normalized Gaussian

(c) Linear model of each Receptive field

(d) Approximated function

Figure 2.5: The LWPR algorithm approximates non-linear functions such as the one shown in (a). The algorithm delimits the input space with Gaussians, also called receptive fields. Gaussians are normalized (b) respectively to their capacity to approximate the non-linear function. The sum of the active linear models, shown in (c), weighted by their respective relative Gaussian performs an approximation of the non-linear function (d).

Like previous learning algorithms, LWPR uses Gaussian functions $\phi_i(\boldsymbol{x})$ as receptive fields

$$\phi_i(\boldsymbol{x}) = e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{c}_i)^T D_i(\boldsymbol{x}-\boldsymbol{c}_i)}. \tag{2.6}$$

where the positive distance metric D_i of the Gaussian delimits a region which is updated during learning to match the training data.

Each receptive field has a corresponding linear model β_i which is used to predict a local scalar output y_i relative to an input vector $\boldsymbol{x}$ as in Equation 2.5

$$\hat{y}_i(\boldsymbol{x}) = \beta_i \boldsymbol{x}. \tag{2.7}$$

The approximation performed by LWPR is the sum of the linear models weighted by their respective active surrounding relative Gaussian weights

$$\hat{y}(\boldsymbol{x}) = \frac{1}{\Phi(\boldsymbol{x})} \sum_{i=1}^{n_n} \phi_i(\boldsymbol{x}) \hat{y}_i(\boldsymbol{x}) \tag{2.8}$$

where $\Phi(\boldsymbol{x}) = \sum_{i=1}^{n_n} \phi_i(\boldsymbol{x})$ and n_n is the number of receptive fields. Later in this thesis, we note

$$\hat{y} = \text{LWPR}_{predict}([\mathcal{M}_{\text{LWPR}}], [\boldsymbol{x}])$$

the instantiation of Equation 2.8 which consists in predicting the output $\hat{y}$, providing the input $\boldsymbol{x}$ and the model $\mathcal{M}_{\text{LWPR}}$ already learnt during the learning period symbolized as

$$\mathcal{M}_{\text{LWPR}} = \text{LWPR}_{learn}([\boldsymbol{x}], [y]).$$

A receptive field is considered active if it is trustworthy, i.e., if it has a model based on sufficient data. A standard threshold to consider a receptive field trustworthy consists in checking if it has received as many example as the double of the dimension of the input $\boldsymbol{x}$. It is also possible to consider as an active receptive field, one whose weight ϕ_i is over a cut-off threshold.

The distance metric D allows creation or pruning of receptive fields in order to improve the coverage of the input space. It is computed as $D = M^T M$ where M is a upper-triangular matrix to ensure D to be positive. M is adapted recursively and individually to regulate the size of the receptive fields with the update law

$$M = M - \alpha \frac{\partial C}{\partial M} \tag{2.9}$$

where α is a learning rate and C is a minimized cost function. This cost function, given by Equation 2.10, is rather complex compared to the weighted mean square error criterion of the LWR algorithm.

$$C = \frac{1}{\Phi} \sum_{i=1}^{p} \sum_{k=1}^{r} \frac{\phi_i \| y_{k,i} - \hat{y}_{k,i} \|^2}{\left(1 - \phi_i \boldsymbol{x}_i^T P \boldsymbol{x}_i\right)} + \gamma \sum_{i,j=1}^{n_n} D_{ij}^2 \tag{2.10}$$

where $P = \left(X \Phi X^T\right)^{-1}$, $X = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_p]$ and γ is the regularization penalty term. The cost function is computed for a first part, by a leave one out cross-validation method. The second part is a penalty term that prevents LWPR to collapse. Larger values of γ

enforce smaller distance metrics, corresponding to wider receptive fields which in turn implies smoother functions.

The real implementation of LWPR uses statistics to update the models and is much more complicated. Equation 2.10 is not used in practice since the incremental computation of the cost leads to a different formulation. In fact, LWPR inherits from the capacity of LWPLS to reduce the input dimensionality using the Partial Least Square algorithm (PLS) [Wold, 1975; Elden, 2004; Tenenhaus, 1998]. The dimensionality is reduced with the projection made on the input vector to model high dimensional functions only on their most relevant directions. We refer the reader to Schaal et al. [2002] or Vijayakumar and Schaal [2000] for a presentation of the incremental version of the algorithm.

As a consequence of using the PLS regression, we note here that it is possible to learn one single model with a vectorial output instead of a scalar one. All output can be projected onto all input at each PLS projection step (see Appendix A.3). The chosen direction for the projection is chosen as the mean projection among all projections of each output. If output are decorrelated, the directions of all projections are not the same and this approach is suboptimal. The LWPR algorithm is thus more precise in the case where one model represents one output at a time.

Gradient of the approximated function

In many contexts such as the one presented in Chapter 3, it is useful to determine the gradient of the function we are approximating. As the model learnt by LWPR is only composed of linear models weighted and summed altogether, it is possible to differentiate it with respect to the input. As the model is learnt incrementally, the gradient is also provided through an incremental algorithm [Klanke et al., 2008].

Considering the global approximation provided by LWPR

$$\hat{y} = \frac{\sum_{i=1}^{n_n} \phi_i(x)\, y_i(x)}{\Phi(x)} \tag{2.11}$$

where $\Phi(x) = \sum_{i=1}^{n_n} \phi_i(x)$, it is possible to differentiate it with respect to the input

$$\frac{\partial \hat{y}}{\partial \boldsymbol{x}} = \frac{\partial \left(\frac{\sum_{i=1}^{n_n} \phi_i(x)\, y_i(x)}{\Phi(x)} \right)}{\partial \boldsymbol{x}}. \tag{2.12}$$

and thus

$$\frac{\partial \hat{y}}{\partial \boldsymbol{x}} = \frac{1}{\Phi} \sum_{i=1}^{n_n} \left(\frac{\partial \phi_i}{\partial \boldsymbol{x}} y_i + \phi_i \frac{\partial y_i}{\partial \boldsymbol{x}} \right) - \frac{1}{\Phi^2} \sum_{i=1}^{n_n} \phi_i y_i \sum_{k=1}^{n_n} \frac{\partial \phi_k}{\partial \boldsymbol{x}} \tag{2.13}$$

where $\frac{\partial y_i}{\partial \boldsymbol{x}}$ is the local gradient of each receptive field.

We note $([\hat{y}], [[\frac{\hat{\partial y}}{\partial \boldsymbol{x}}]]) = \text{LWPR}_{predictJ}([\mathcal{M}_{\text{LWPR}}], [\boldsymbol{x}])$ the way LWPR predicts the gradient of the output relatively to input.

2.1.5 Extended classifier system for function approximation

In this section, we present the eXtended Classifier System for Function approximation (XCSF) algorithm, another function approximator that shares some similarities with LWPR but comes from Learning Classifier System (LCS), a family of adaptive rule-based[2] systems first proposed by Holland [1975] that combines reinforcement learning mechanisms [Sutton and Barto, 1998] and genetic algorithms [Goldberg, 1989].

The immediate ancestor of XCSF is the eXtended Classifier System (XCS) [Wilson, 1995], an efficient accuracy-based LCS designed to solve discrete classification problems and sequential decision problems. XCS finds accurate evolving solutions for a wide range of problems with high probability in polynomial time [Butz et al., 2005]. For detailed information, the interested reader is referred to the algorithmic description of XCS [Butz and Wilson, 2000].

XCSF [Wilson, 2001, 2002] is an evolution of XCS towards function approximation. Each classifier is composed of a condition part and a prediction part. It specifies in its condition part its applicability i.e., the context in which the rule can be applied, and in its prediction part its prediction using for example some learnt linear models. In our case of interest, the prediction is linear with respect to the input. The classifiers in XCSF form a population P that partitions the input space into a set of overlapping prediction models. The resulting smoothed surface forms the function approximation surface.

We first introduce the global structure of the algorithm. Then we explain how XCSF relies on update rules and evolutionary mechanisms to update parameters. Then we explain the compaction rule.

Structure of the approximator

XCSF is a generic framework that can use different kinds of prediction models (linear, quadratic, etc.) and can pave the input space with different families of receptive fields (Gaussians, hyper-rectangles, etc.). In the context of this thesis, we will only consider the case of linear prediction models and Gaussian receptive fields. In this case, the structure of the function approximation built by XCSF is similar to the one built by LWPR. Thus, in that case, XCSF is a local regression method in the same sense as LWPR. However, a specificity of XCSF is that it distinguishes a *prediction input space* and a *condition space*. The *prediction input space*, named $\mathcal{X}$, corresponds to the input used to compute the linear models with regression. The *condition space*, named $\mathcal{Z}$, is paved

[2] In LCSs, a rule is called a classifier.

with Gaussian *classifiers*, corresponding to the receptive fields of LWPR, which define a domain $\phi_i(\boldsymbol{z})$ in the following way

$$\phi_i(\boldsymbol{z}) = e^{-\frac{1}{2}(\boldsymbol{z}-\boldsymbol{c}_i)^T D_i(\boldsymbol{z}-\boldsymbol{c}_i)} \tag{2.14}$$

where c_i and D_i are respectively the center of the Gaussian i and a positive distance metric, inverse of the squared variance of the Gaussian i. $\boldsymbol{z} \in \mathcal{Z}$, the input of the condition space, defines the space where the Gaussian have an influence on the global prediction. As described later, splitting the condition space and the prediction input space is a feature which plays an important role in the context of learning mechanical models.

Each classifier has a corresponding linear model β_i which is used to predict a local output vector $\boldsymbol{y}_i$ relative to an input vector $\boldsymbol{x}$ as in Equation 2.5

$$\hat{\boldsymbol{y}}_i(\boldsymbol{x}) = \beta_i \boldsymbol{x} + \beta_0. \tag{2.15}$$

While Equation 2.15 is more generic with the β_0 offset, it complicates the model if the learnt mapping is linear and not affine. In the linear case, the learnt offset is null and its presence unnecessarily complicates the learning process. Older versions of LWPR used an offset β_0 as well but this offset can be removed by enhancing the input $\boldsymbol{x}$ with 1 as $\boldsymbol{x}_{aug} = \begin{bmatrix}\boldsymbol{x}^T & 1\end{bmatrix}^T$. Equation 2.15 would be simpler using the augmented input trick.

XCSF uses only a subset of the classifiers to generate an approximation. Indeed, at each learning iteration, XCSF generates a match set M that contains all reliable classifiers in the population P which condition space $\mathcal{Z}$ match the input data $\boldsymbol{z}$, i.e., for which $\phi_i(\boldsymbol{z})$ is above a threshold ϕ_0[3].

In XCSF the learning output $\hat{\boldsymbol{y}}$ is given for a $(\boldsymbol{x}, \boldsymbol{z})$ pair as the sum of the linear models of each classifier i weighted by their respective fitness F_i

$$\hat{\boldsymbol{y}}\left(\boldsymbol{x}, \boldsymbol{z}\right) = \frac{1}{F\left(\boldsymbol{z}\right)} \sum_{i=1}^{n_M} F_i\left(\boldsymbol{z}\right) \hat{\boldsymbol{y}}_i\left(\boldsymbol{x}\right) \tag{2.16}$$

where $F\left(\boldsymbol{z}\right) = \sum_{i=1}^{n_M} F_i\left(\boldsymbol{z}\right)$ and n_M is the number of classifiers in the match set M.

In the remainder of this thesis, we note

$$\hat{\boldsymbol{y}} = \text{XCSF}_{predict}\left(\mathcal{M}_{\text{XCSF}}, [\boldsymbol{z}], [\boldsymbol{x}]\right) \tag{2.17}$$

the instantiation of Equation 2.16 and we note

$$\mathcal{M}_{\text{XCSF}} = \text{XCSF}_{learn}([\boldsymbol{z}], [\boldsymbol{x}], [\boldsymbol{y}]) \tag{2.18}$$

[3]This threshold, named θ_m in [Butz and Herbort, 2008]

the model learnt by XCSF given a set of condition states $[\boldsymbol{z}]$ and a pair of input/output prediction states $[\boldsymbol{x}], [\boldsymbol{y}]$.

To update the β_i, c_i and D_i parameters, XCSF relies on two interacting learning components presented below: the update rules and the evolution mechanism. This interaction is important as a good estimation of the classifier utility makes the evolution more reliable and effective while avoiding misleading signals.

A global vision of the algorithm is illustrated in Figure 2.6. The execution order of each step is symbolized by the numbers explained in Algorithm 2.1.5 where θ_{GA} is a threshold that controls the number of time steps between each iteration.

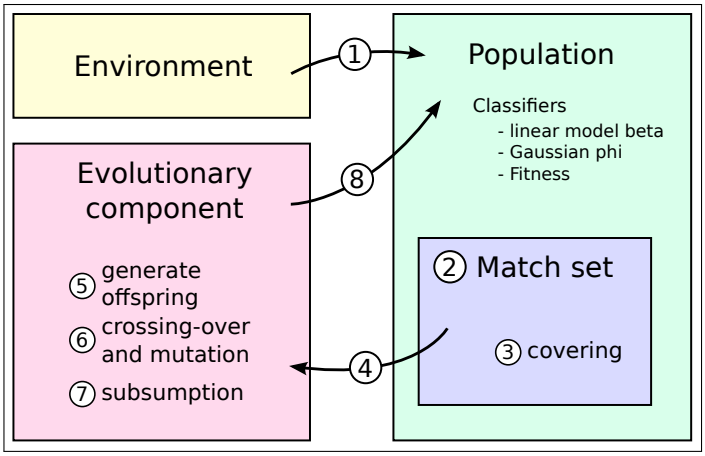


Figure 2.6: A schematic view of the XCS and XCSF algorithms.

Update rules

At each time step, each classifier i in the match set M makes an error $\delta \boldsymbol{y}_i = \boldsymbol{y} - \hat{\boldsymbol{y}}_i$ used to update its linear model, its prediction and its accuracy.

The linear model is updated using the RLS algorithm (see Equation A.9 in Appendix A)

$$\delta \beta_i = \delta \boldsymbol{y}_i \boldsymbol{x}_i^T \alpha$$

where α is a learning rate.

The prediction error approximates the mean absolute deviation of its prediction using the delta rule

$$\delta \boldsymbol{\epsilon}_i = \eta \left(|\boldsymbol{y}_i| - \boldsymbol{\epsilon}_i \right)$$

where η is a learning rate and $\boldsymbol{\epsilon}_i$ the mean absolute error of the classifier i at the previous time step.

Algorithm 2 Pseudo-code of XCSF (see Figure 2.6)

```
[x], [y], [z] ← environment                          (1) get problem instance
if φ_i(z) > φ_0 then
    M ← P, [z]                                       (2) generate match set out of P using z
    if M is empty then
        M ← [z]                                      (3) cover with a new classifier using z
    end if
end if
M ← M, [z]                                           update matching classifiers
if time − LastAverageTime(M) < θ_GA then
    Parents ← M, Fitness                             (4) select two parents
    Offspring ← Parents                              (5) generate the offspring
    Offspring ← Offspring, Parents                   (6) crossover and mutation
    Offspring ← Offspring, Parents, M                (7) testing if subsumption
    P ← Offspring                                    (8) insertion in P. deletion if max is reached.
end if
```

The classifier accuracy is determined by the scaled inverse of the mean absolute error. Classifiers with a mean absolute error lower than a threshold ϵ_0 are considered trustworthy.

The fitness value is derived from the relative classifier accuracy and thus reflects the quality of a classifier in comparison to all others in the match set M.

Evolutionary component

In XCSF, a classifier is created when there is no classifier in the match set corresponding to an input. This covering mechanism creates a matching condition which corresponds to that input with a random bounded condition space. Frequently, classifiers are not generated by the covering mechanism but by an evolutionary component based on a niching steady-state Genetic Algorithm (GA) Wilson [2002]. It encourages the recombination of efficient classifiers by crossing-over and mutations. A GA is performed if the average time since the last GA occurrence exceeds a threshold θ_{GA}. The generalization in XCSF is achieved by the reproduction of classifiers, performed within the match set M, and the deletion of inaccurate classifiers performed in the whole population P [Wilson, 1995; Butz et al., 2004]. Moreover, general classifiers are reproduced more often as they match the input data more often than the others.

Recombination The GA selects, in a match set M, two classifiers regarding their current fitness values, i.e., their relative predictive accuracy facing the other classifiers. The GA uses a set-size-relative tournament selection as described in [Butz et al., 2003]

to select two parents used to generate two offspring. To generate the new children attributes, XCSF utilizes uniform crossover with a fixed probability μ, typically 0.5. New centres c_i^{new} are moved within an interval defined as $\|c_i - c_i^{new}\| \leq \sqrt{-2D_i \ln \phi_0}$ where ϕ_0 is the same threshold used to generate the match set M. The shape of the ellipsoids, defined by σ_i, the square-root of D_i, are also randomly increased or decreased within a limited interval.

The third part of the recombination phase is the subsumption process. When created, each classifier owns a counter n_{count} fixed at one. This counter is the number of classifier it has subsumed; if a classifier contains completely another classifier, the contained one is destroyed while the containing one increments its counter and is now named a macro-classifier. During all phases of the algorithm, a macro-classifier can be viewed as n_{count} micro-classifiers.

Deletion Once, in the whole population P, the maximum number of classifiers n_P^{max} is reached, classifiers may be deleted before inserting the offspring. Supernumerary classifiers are deleted from P with a probability proportional to an estimate of the size of the match sets that they occur in. If a classifier is sufficiently experienced and its fitness F is significantly lower than the average fitness of classifiers in P, its deletion probability is further increased [Kovacs, 1999].

In sum, the evolutionary mechanism is designed to evolve partitions in which linear approximations are maximally accurate. Indeed, XCSF strives to evolve complete, maximally accurate, and maximally general function approximations represented in its population of classifiers. This makes XCSF an iterative clustering mechanism that clusters the condition space for the generation of maximally accurate predictions in the input space. The mechanism also ensures complete coverage of the encountered problem space since overrepresented subspaces undergo more deletions, while reproduction is occurrence-based [Butz and Goldberg, 2003].

Compaction

At the end of a learning process, the final population is composed of highly overlapping classifiers. To reduce the size of the population, Butz and Herbort [2008] propose to use a Closest Classifier Matching (CCM) algorithm to have a fixed size matching set M.

This compaction algorithm consists in spanning, by order of accuracy, the centres of all classifiers in the whole population P. For each centre, a match set is created to select all corresponding overlapping classifiers. Then the most accurate classifier subsumes all classifiers in the match set. This compaction algorithm is simplified by the use of CCM which forces to represent uncovered input regions.

A more complete description of XCSF can be found in Butz et al. [2004] or in Butz and Herbort [2008].

2.2 Learning Mechanics

Since supervised learning methods can approximate any function, they can be used to approximate a mechanical model. As this process is only dependent upon data of the mechanical system itself, it is often named self-supervised learning. In that context, learning algorithms are used to approximate models to control robotic systems. This section describes how kinematics and dynamics can be learnt as well as the learning methods employed.

Our survey of the literature is focused on methods used in order to learn the model of mechanical systems. There is another important set of works that call upon learning methods to perform imitation learning. Some of these work use tools such as Gaussian Mixture models or Gaussian Process Regression. As these works will not be treated here, interested readers can refer to Calinon [2009] for an excellent survey. Note however that some of these methods are starting to get used for learning mechanical model as well. For instance, Nguyen-Tuong et al. [2008] compare Locally Weighted Projection Regression, Gaussian Processes Regression and Support Vector Regression performances in learning the inverse task space dynamics of a seven degrees of freedom Sarcos arm.

2.2.1 Learning inverse kinematics

We have seen in Section 1.2.2 that controlling systems in the operational space requires the knowledge of inverse kinematics. It seems natural to learn directly that inverse mapping to control a system. The *end-effector/joint* positions pairs, given by the laws of Mechanics, can be used as *input/output* of a supervised learning method to approximate inverse kinematics (see Figure 2.7).

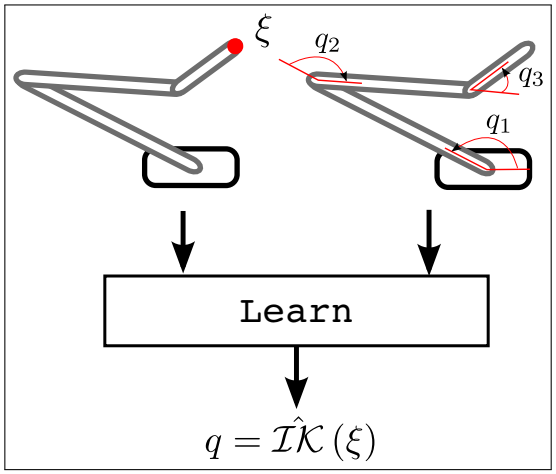


Figure 2.7: Learning inverse kinematics with a learning algorithm. The model is learnt providing as input an end-effector position and a configuration.

In the case of non-redundant robots, this approach does not raise any specific issue. Indeed, without considering singularities, when the dimension of the controlled end-effector position is equal to the dimension of the joint position, the robot is fully constrained and its inverse kinematics is unique. The mapping is thus a function that can be learnt. Based on self-organising maps [Kohonen, 2001], Ritter et al. [1989] and Martinetz et al. [1990b] learn a mapping between the three dimensional end-effector position measured by two cameras and joint positions on a three degrees of freedom robot. The mapping is made by a three dimensional self-organising map. Walter and Schulten [1993] learn inverse kinematics on a 6 degrees of freedom industrial robot Puma 562. As its wrist is rigid, the robot is non-redundant with respect to the task which consists in positioning the end-effector in the Cartesian space.

By contrast, this approach raises problems in the redundant case. While forward kinematics of under-constrained redundant robots is injective, inverse kinematics is not. As explained in Section 1.2.2, there exists an infinity of possible inverse kinematics corresponding to the possible internal movements. The problem of identifying one particular inverse kinematics among infinity of such mappings is ill-posed.

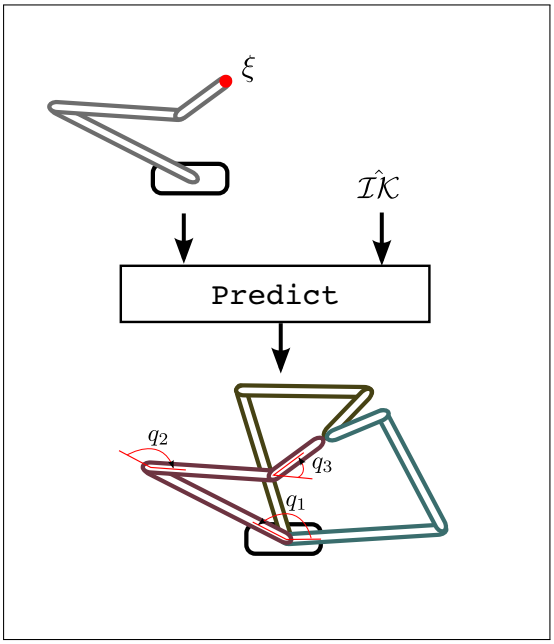


Figure 2.8: Reproducing inverse kinematics in approximating an output given the learnt model and an input. Supervised learning methods provide only one configuration among an infinity. Three different configurations are represented here.

Figure 2.8 implies that for predicting inverse kinematics, it is necessary to provide an operational position and a learnt model. Different types of solutions are proposed in the literature to learn the inverse kinematics of redundant systems. All the methods presented below transform the ill-posed problem into a well-posed problem by adding

constraints. Those added constraints can consist in augmenting the task to have a full row rank Jacobian matrix or in solving an optimisation problem to minimize for example an energy function. With the additional constraints, the inverse kinematic model becomes unique and can then be learnt without loss of information. Using that resolution scheme, those methods can control redundant robots by solving an equivalent problem but without the possibility to control explicitly the redundancy.

Neural networks have been used to learn the inverse kinematics, solving an unconstrained optimisation problem formulated as an energy function. This energy function is minimised during the gradient descent and weights are consequently updated. It is thus possible to obtain a desired joint velocity corresponding to a minimum energy function as

$$\dot{q}^* = \arg\min_{\dot{q}} \left(E\left(\dot{q}\right) \right)$$

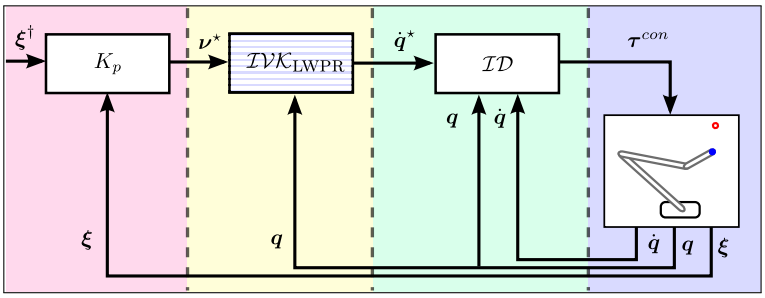
where E is the energy function based on different errors.

Pourboghrat [1989] minimises an energy which leads to learn the Moore-Penrose inverse (as seen in Equation 1.25). Barhen et al. [1989] learn inverse kinematics and resolve redundancy by minimizing different Lyapunov functions with goal attractors. Ahmad and Guez [1990] optimise the manipulability criterion: $H = \sqrt{\|JJ^T\|}$ in each configuration. Lee and Kil [1990] learn the Jacobian matrix of a redundant robot, minimising an energy function composed of two types of constraints: one is minimising the velocity of the first joint and another is locating the joints in the middle of joint-limits. Brüwer and Cruse [1990] learn, with a multi-layer perceptron, two kinematic models on a three degrees of freedom planar arm. The first model positions the end-effector in a plane and the second perceives this position by means of a retina-like input layer. The networks are trained using data from a human subject and thus the redundancy is solved during the learning period. Based on self-organising maps [Kohonen, 2001], Martinetz et al. [1990a] learn a mapping between the end-effector position measured by two cameras and joint positions on a three degrees of freedom robot. The mapping is build on a three dimensional self-organising map. They resolve redundancy by minimising the variation of the joint angles during the learning process. D'Souza et al. [2001] learn an inverse kinematic model with LWPR (see Figure 2.9). The model is an inverse of the extended Jacobian matrix learnt while performing the task with the input $\left(\boldsymbol{q}, \dot{\boldsymbol{\xi}}\right)$ and the output $(\dot{\boldsymbol{q}})$

$$\mathcal{IVK}_{\text{LWPR}} = \text{LWPR}_{learn}\left(\left[\boldsymbol{q}, \dot{\boldsymbol{\xi}}\right], [\dot{\boldsymbol{q}}]\right).$$

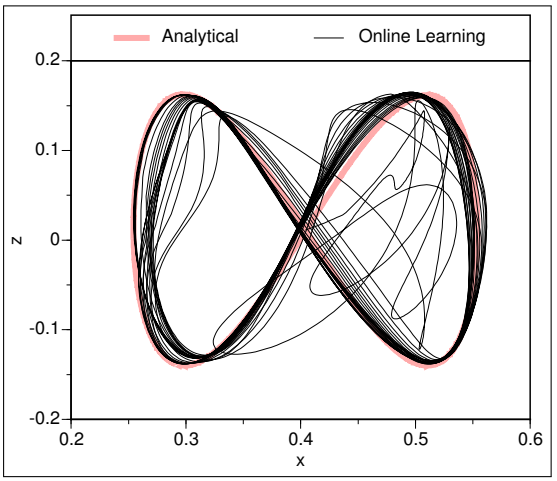
The model is learnt along an operational trajectory, solving redundancy by minimising a cost function dependent of the posture. Doing so, no inversion is involved and singularity problems are avoided.

The results presented in Figure 2.9 correspond to the control of the ATR robot where the goal is to control the hand in the Cartesian space. The robot owns 26 actuated joints among which only 4 are relevant for the learning algorithm. Thus, the dimension of the input is: $dim(\xi, q) = 29$ and the dimension of the output is: $dim(\dot{q}) = 26$ even though



(a) Control scheme

(b) ATR robot [Kawato, 1999]



(c) Followed trajectory

Figure 2.9: Learning inverse kinematics of the ATR robot with LWPR [D'Souza et al., 2001].

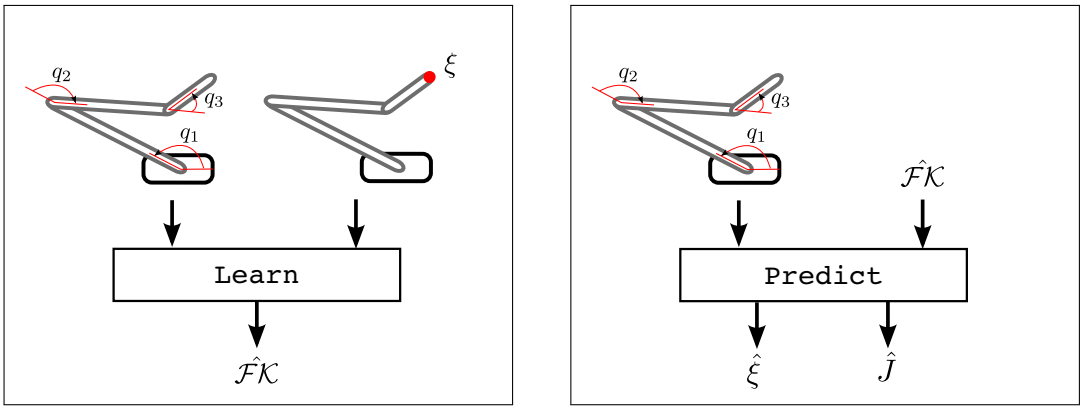
22 of these dimensions are irrelevant and probably not used.

As redundant problems have been solved by adding constraints, it is only possible to control the robot considering those constraints. If the constraints change (the size of the object, the position of an obstacle, etc.), those methods can adapt only by learning a new model which corresponds to the new constraints.

2.2.2 Learning forward kinematics

In the case of redundant robots, in contrast with inverse kinematics, forward kinematic and forward velocity kinematic models are unique. A model is unique if the dimension of its output is lower than the dimension of the principal components of its input. In that case, the model is a function that can be approximated without loss of information.

Learning forward kinematics consists in providing as input the joint positions and as output the position of the end-effector we want to control (see Figure 2.10a).



(a) Learning forward kinematics with a learning algorithm.

(b) Reproducing forward kinematics in approximating an output providing the learnt model and an input.

Figure 2.10: Learning forward kinematics providing joint positions and end-effector positions along time. With some methods, it is possible to approximate the derivative of the learnt model.

After a learning period, the learning algorithm can estimate the position of the end-effector by providing it with any joint configuration (see Figure 2.10b). It is important to notice that, as described in Section 2.1.4, learnt models can be derived. In fact, some supervised learning methods such as RBFNs or LWPR are using a weighted sum of derivable functions to perform a mapping. The global derived model is consequently a combination of the derived functions. In the case of the forward kinematics, the global first order derivative relatively to input $\boldsymbol{q}$ is the Jacobian matrix also named forward velocity kinematics.

Few research works try to learn forward models of serial redundant robots as they can easily be computed analytically when the geometry of the robot is know accurately. Nevertheless, learning kinematics of those robots does not raise particular conceptual problems because this application is injective and thus unique.

Learning kinematics is advantageous when the model changes over time or when the geometry is not known in advance, as highlighted in the introduction.

A few authors have used multi-layer perceptrons networks or self-organising maps to learn the forward kinematics model of various simple systems [Nguyen et al., 1990; Wang and Zilouchian, 1997]. The evaluation is based on the accuracy of the model itself rather than on its control capabilities. Boudreau et al. [1998], Sadjadian and Taghirad [2004] and Sang and Han [1999] learn the forward models of parallel robots which involves highly coupled non-linear equations that are difficult to model analytically.

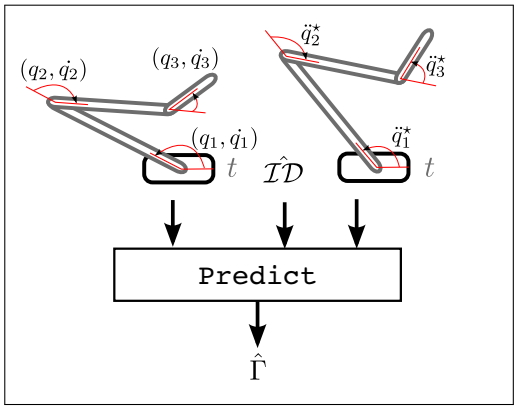
Very close to our approach, Sun and Scassellati [2004, 2005] learn forward kinematics with a RBFN. They derive each basis functions to obtain a Jacobian matrix, using RMRC as a basis for their incremental trajectory generation algorithm (ITGA). However, they

note that their algorithm can be adapted to any extension of RMRC. They use both cameras of a humanoid robot to obtain the operational position used in their controller.

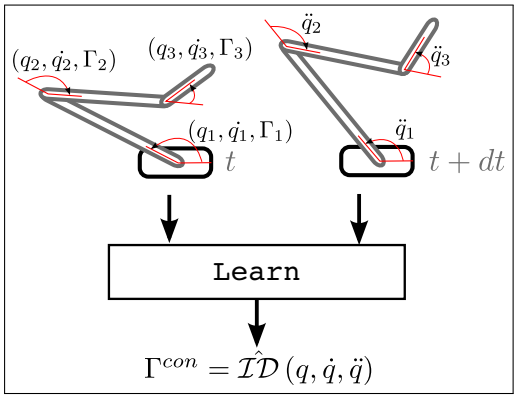
Also close to our approach, Butz et al. [2007]; Butz and Herbort [2008]; Butz et al. [2009]; Stalph et al. [2009] learn the forward kinematic model of a four degrees of freedom human-like arm using XCSF in order to model motor adaptation in arm reaching experiments. They inverse the model with classical analytical methods in order to control an end-effector in three dimension while controlling the redundancy. They have chosen three different secondary tasks in the joint space which consist in minimizing angular velocities, avoiding joint limits or specifying a global posture. The method employed is well described in the next chapters as it is the one we use when we learn mechanical models with XCSF.

2.2.3 Learning inverse dynamics

We have seen in Section 1.2.4 that it is possible to include inverse dynamics in a control loop. The inverse dynamics allows to impose a desired acceleration, in any given state $(\boldsymbol{q}, \dot{\boldsymbol{q}})$, with torques, applied at the joint level. For a fully-actuated system, the dimension of the actuation torque vector $\boldsymbol{\tau}^{con}$ equals the number of degrees of freedom n. The mapping is *unique* if there are no redundant actuators, i.e., if there is only one actuator on each joint. In that case, it is possible to learn the inverse mapping as described in Figure 2.11a.



(a) Using inverse dynamics to obtain a torque given a current state, the learnt model and a desired joint space acceleration.



(b) Learning inverse dynamics with a learning algorithm.

Figure 2.11: Learning and reproducing inverse dynamics with a learning algorithm.

As in learning kinematics schemes, the physics is still the supervisor of the supervised learning algorithm. To learn inverse dynamics as described in Figure 2.11b, it is necessary to have access to joint positions and velocities for a given task space, the control torques applied, and the resulting acceleration at the next time step. Thus the input is

$(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t+dt))$ and output is $\boldsymbol{\tau}^{con}(t)$. The application is a function from $\mathbb{R}^{3n}$ to $\mathbb{R}^n$ where n is the dimension of the joint space. If the complexity of the poly-articulated system increases, n increases too and it becomes more and more difficult to learn this mapping.

When we learn dynamics in such a way, it is important to notice that the learning algorithm needs to access to the acceleration of the next time step $\ddot{\boldsymbol{q}}(t+dt)$. Thus, the learning iteration has to be performed at the next time step. Nevertheless, while kinematics depends exclusively of the configuration of the system, it is conceptually different to learn dynamics as it depends on the duration of the time step. This may raise problems in different situations such as changing the time step from a simulation to another, using a simulator with a non-fixed time step integration scheme or being subject to delays when shifting from a simulation to a real robot.

Learning inverse dynamics on whole spaces

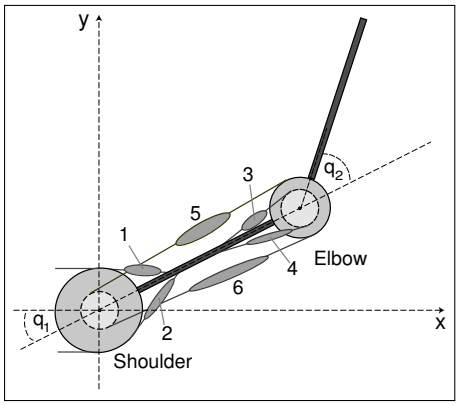
It is possible to learn inverse dynamics in any configuration, for every possible joint velocity and joint acceleration. Narendra and Parthasarathy [1990] learn inverse dynamics of artificial systems in state space with neural networks to show that it is possible to use learnt models in control. Jordan and Rumelhart [1992] provide a good review on how to learn forward and inverse dynamics and how to use them in control. Lin and Goldenberg [2001] learn an inverse dynamics model of a 4 degrees of freedom manipulator in three dimensions with an RBFN using Lyapunov functions.

More recently, Mitrovic et al. [2008] learn inverse dynamics of a two degrees of freedom arm in the whole planar space with LWPR. The input is $(\boldsymbol{q}(t),\dot{\boldsymbol{q}}(t),\boldsymbol{u}(t))$ and the output is $(\dot{\boldsymbol{q}}(t),\ddot{\boldsymbol{q}}(t+dt))$. The actuation controls are named $\boldsymbol{u}$ instead of $\boldsymbol{\tau}$ as the actuators are two artificial muscles (see Figure 2.12a) for each joint with two more artificial muscles to couple the joints. Thus the control is not a torque but a force. The dimension of the input is $dim(\boldsymbol{q}(t),\dot{\boldsymbol{q}}(t),\boldsymbol{u}(t)) = 10$ and the output dimension is $dim(\dot{\boldsymbol{q}}(t),\ddot{\boldsymbol{q}}(t+dt)) = 4$.

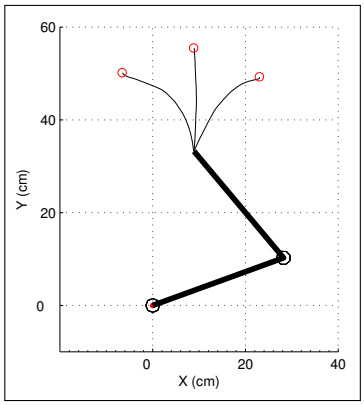
The direct dynamics learnt is redundant with respect to its actuation. They use an optimal feedback control loop (see Figure 2.13) based on the ILQG algorithm [Li, 2006] to find the optimal forces and to control the system. After a learning period containing $1.2\ 10^6$ training data points, the model is composed of 852 receptive fields and succeeds in reaching three different targets as shown in Figure 2.12b.

Learning inverse dynamics along a trajectory

It becomes more and more difficult to learn inverse dynamics as the dimension of the joint space increases. Given the larger size of the joint space, one may think of only approximating sub-spaces of interest, for instance along a given joint space trajectory or in a domain corresponding to the minimisation of a cost function.



(a) Human arm model with 6 muscles



(b) Same arm model with selected targets (circles) and ILQG generated trajectories as benchmark data

Figure 2.12: Learning inverse dynamics on whole space using iterative Linear Quadratic Gaussian controller [Mitrovic et al., 2008]. Figure 2.12a represent the human-like arm actuated with models of artificial muscles. Figure 2.12b illustrate initial followed trajectories.

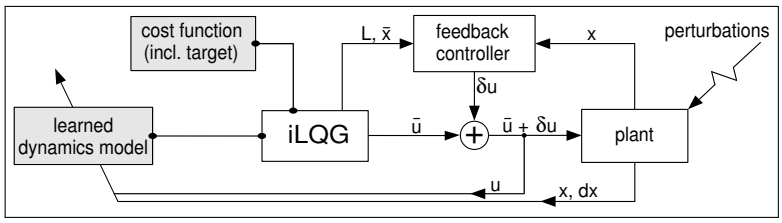


Figure 2.13: Learning inverse dynamics on whole space using iterative Linear Quadratic Gaussian controller [Mitrovic et al., 2008].

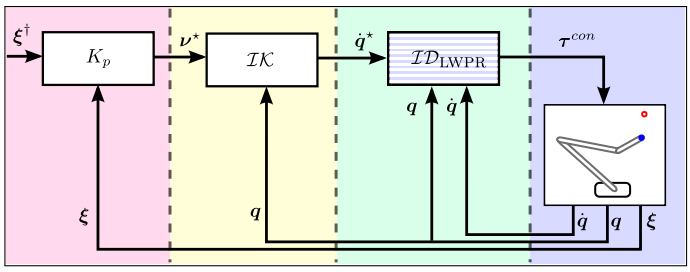
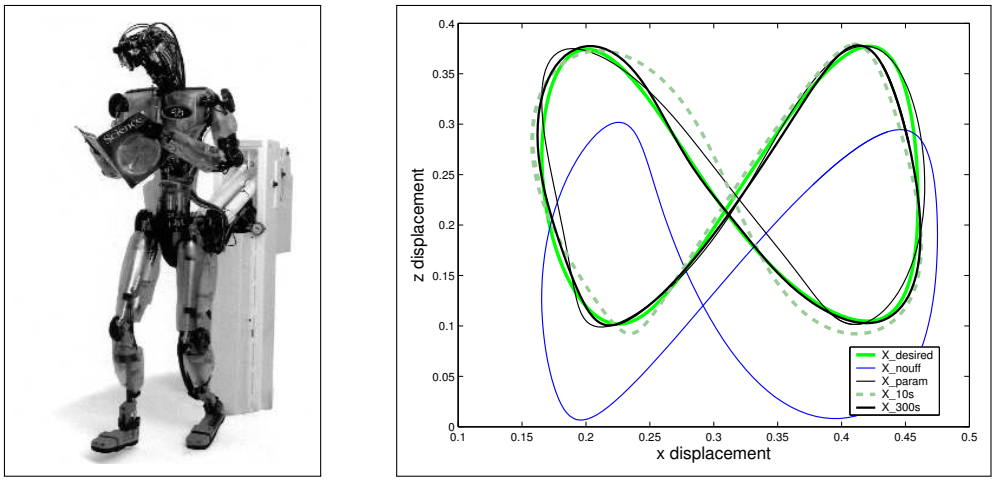


Figure 2.14: Model-based control in task space using learnt inverse dynamics.

Figure 2.14 describes how Vijayakumar and Schaal [1997], Vijayakumar and Schaal [2000], Schaal et al. [2002] and Vijayakumar et al. [2005] learn an inverse dynamics with

LWPR along an operational trajectory. The trajectory is followed with a $\mathcal{PD}$ controller, described in Figure 1.7 page 24, and an additional noise to learn, around the trajectory, an envelope of possible configurations. A limitation of this approach is that, if we learn a model only along the trajectory and use it in a control loop and if the model is not precise enough, the trajectory will not be followed exactly. The robot may then reach a configuration unknown to the model which thus may return an inadequate torque. They use their algorithm on an anthropomorphic Sarcos robotic arm composed of 7 degrees of freedom. The input of the learning algorithm is $(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t+dt))$ of dimension 21 and the output is $\boldsymbol{\tau}(t)$ of dimension 7. They use 5 10^4 training data points and 260 receptive fields to obtain a model.

Vijayakumar et al. [2005] also use the same approach to learn dynamics on the ATR humanoid robot [Kawato, 1999]. The input of the learning algorithm is $(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t+dt))$ of dimension 90 and the output is $(\dot{\boldsymbol{\tau}}(t))$ of dimension 30. They use 7.5 10^6 training data points and 2200 receptive fields to obtain a model. To our knowledge, this is the largest space in which a function has been learnt with LWPR, but the function is learnt along a trajectory in a much smaller subspace than the input space.



(a) ATR humanoid robot. (b) Followed trajectory.

Figure 2.15: Learning inverse dynamics along a trajectory with LWPR. Controlled robot and results.

As the learning of 30 parallel LWPR models would be computationally expensive, they choose to learn one single model with 30 dimensional output projected, thanks to the PLS algorithm, onto the input data. As we have seen, this approach is suboptimal but computationally less expensive.

Learning inverse dynamics in task space

Learning an inverse dynamic model in the task space is a smaller learning problem than learning it in the joint space. Indeed, inverse dynamics in task space is a mapping from $\mathbb{R}^{2n} + \mathbb{R}^m$ to $\mathbb{R}^m$ whereas inverse dynamics in joint space is a mapping from $\mathbb{R}^{3n}$ to $\mathbb{R}^n$ where n is the dimension of the joint space and m is the dimension of the task space. Poly-articulated systems have a task space dimension which is usually smaller[4] or equal to the dimension of the joint space, thus, it can be of interest to learn inverse dynamics in the task space. Nevertheless, learning inverse dynamics in the task space implies learning one dynamical model for each controlled end-effector whereas a unique inverse dynamic model in the joint space is sufficient for every task.

In both cases, it is necessary to compute the kinematics even to compute the joint velocities from operational velocities or, in the case of inverse dynamics in the task space, to compute the control torques from operational forces.

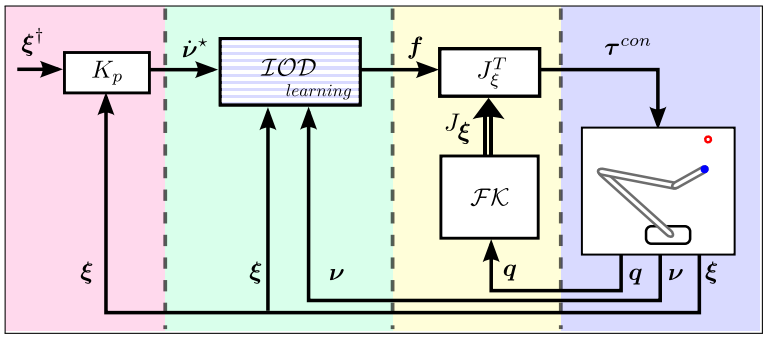
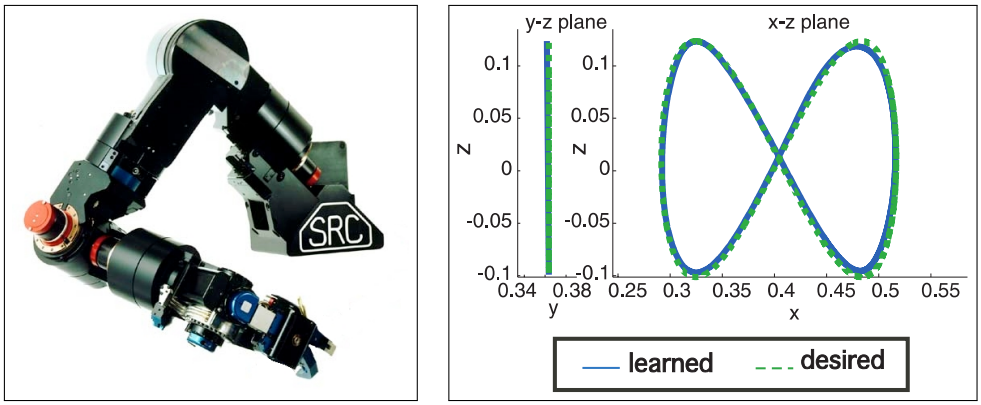


Figure 2.16: Peters and Schaal [2008] learn inverse dynamics in the operational space.

As described in Figure 2.16, Peters and Schaal [2008] use LWPR to learn inverse dynamics in task space (see Section 1.2.5). They use their control law, based on the learnt dynamic model, on the Anthropomorphic Sarcos arm (see Figure 2.17a) and on the Mitsubishi PA-10 robot (see Figure 2.18a). For both systems, the dimension of the input equals $dim(\boldsymbol{q} + \dot{\boldsymbol{q}} + \boldsymbol{\nu}) = 17$ and the dimension of the output equals $dim(\boldsymbol{\tau}) = 7$.

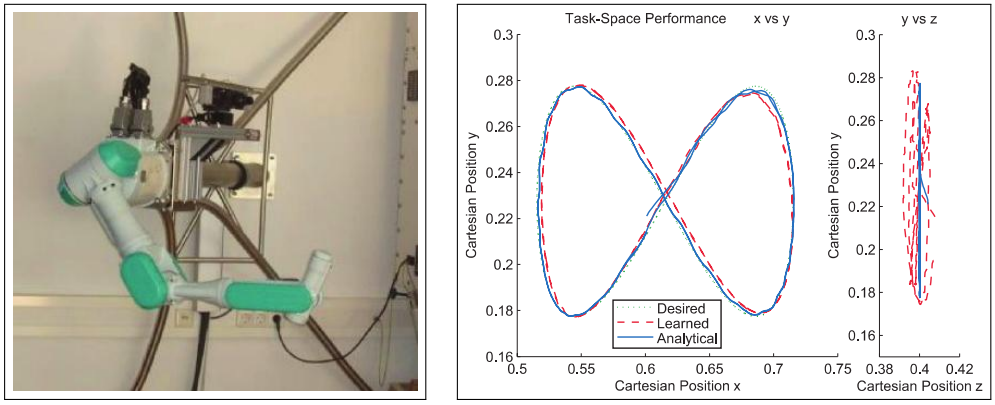
Those dimensions have to be compared to the size of two separate models which can be learnt independently: one inverse kinematic model which links a 3 dimensional space to a 7 dimensional space and an inverse dynamic model which links a 21 dimensional space to a 7 dimensional space. If we only compare the dimensions of the learnt inverse dynamic model, we can see that the dimension is lower in the operational space without considering that, in the other case, the inverse kinematics has to be learnt.

[4]The maximum dimension of the task space is 6 and the dimension of the joint torques space equals approximately 30 for a standard humanoid robot.



(a) Anthropomorphic Sarcos arm. (b) Target trajectory.

Figure 2.17: Learning inverse dynamics in the task space with LWPR. Anthropomorphic 7 degrees of freedom Sarcos arm and the well followed target trajectory.



(a) Mitsubishi PA-10 robot. (b) Target trajectory.

Figure 2.18: Learning inverse dynamics in task space with LWPR. Mitsubishi PA-10 and the well followed target trajectory.

2.3 Conclusion

In this chapter, we have presented a family of supervised learning methods that are used to learn some models of mechanical systems, then we have presented how these methods are used in practice to control the corresponding robots. One result of this survey of the literature is that some state-of-the-art learning methods seem efficient enough to learn some models of relatively complex mechanical systems. However, it turns out that most works in the domain are focused on directly learning the inverse kinematics [DeMers and Kreutz-Delgado, 1997; D'Souza et al., 2001; Ruiz de Angulo and Torras, 2002] of the robot, which raises specific difficulties in the case of redundant systems.

Though some of these methods can control redundant robots, none of them can explicitly control redundancy because they solve an equivalent non-redundant problem during the learning period and not during control. Very few systems learn their forward kinematics or velocity kinematics -Sun and Scassellati [2005] and Butz and Herbort [2008] being an exception- though it seems necessary to call upon the forward velocity kinematic model to solve redundancy in a way that makes the robot capable of combining several tasks properly.

It is important to note that for most of the supervised learning algorithms, the dimension of the output is not a limiting factor space as they learn each output separately. The correlation are only made between all input and one output. Thus the complexity of the created model depends mostly on the dimension of the input.

More importantly, the very few authors who propose a method giving rise to the possibility to control redundant robots do so in a context where the dynamics is given, or learnt only along specific trajectories [Vijayakumar et al., 2005]. Thus, to our knowledge, there is no model today capable of combining several tasks while learning the dynamics of the system on the whole joint space.

Finally, we must note that the papers cited in this chapter are all concerned with learning the model of a mechanical system without distinguishing if the modelled entity is the robot itself or a more complex system composed of the robot and interacting elements in its environment. There is now a significant shift of the literature towards more complex questions such as learning an explicit model of the interaction between a robot and an unknown grabbed object [Petkos et al., 2006; Petkos and Vijayakumar, 2007] or reconstructing, from the data, the constraints that hold in a task [Howard and Vijayakumar, 2007] in order to control a redundant robot.

3 Constraints to control redundancy

Contents

In previous chapters, we have presented methods for controlling redundant robots as well as supervised learning methods used to approximate non-linear functions. We have described various control architectures using learnt models to control robotic systems, stressing in particular the problems that arise in the case of redundant robots. We have highlighted that none of these architectures is able to control redundancy explicitly with learnt models.

In this chapter, we introduce a method which can combine learning techniques and control in operational space in such a way that we can hierarchically deal with several tasks and constraints when the robot is redundant with respect to its tasks. Our approach is largely based on frameworks presented in the previous chapters and we frequently refer to them. In particular, our approach consists in learning a forward kinematic model using the RMRC formalism (see Section 1.2) in our control scheme. We show how we can inverse the mapping at the velocity level and use the projectors which are necessary to combine several tasks.

The chapter is organized as follows. In Section 3.1, we detail how we learn forward kinematics and inverse it to control redundancy. We explain in detail our methodology to learn the kinematic models, focusing on the difficulties that must be addressed to combine several tasks hierarchically. Then, we detail how to learn kinematics with both LWPR and XCSF algorithms. In Section 3.2, we explain how to integrate a learnt inverse dynamics in our control scheme, again with LWPR and XCSF. In Section 3.3, we present the methods employed to evaluate the quality of the prediction.

One must note that there is no redundancy in the dynamics model thus the model is unique and we can learn its inverse directly without loss of information, in contrast with

the velocity kinematic level. It is also important to stress that the learning of the inverse dynamics in operational space is out of the scope of the work presented here, interested readers can refer to [Peters and Schaal, 2008] and [Nguyen-Tuong et al., 2008].

3.1 Learning forward kinematic models to control redundancy

The main specificity of our control and learning architecture results from the fact that we want to combine several tasks with our system in controlling its redundancy. This design goal generates the following constraints.

- While it is possible to learn directly an inverse velocity kinematic model such as $\dot{\boldsymbol{q}} = \mathcal{IVK}(\boldsymbol{q}, \boldsymbol{\nu})$, it is necessary to learn the forward velocity kinematics and inverse it afterwards in order to control the redundancy. To control that redundancy, the inversion can be conveniently done at the velocity level whereas at the geometric level, the relation is not linear relatively to joint velocities (see Chapter 1) and cannot be easily inverted. Indeed, to invert a model easily, we need to represent it explicitly under a linear form. Learning the forward velocity kinematics such as $\boldsymbol{\nu} = \mathcal{FVK}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is conceptually very different from learning the Jacobian matrix $\boldsymbol{J}(\boldsymbol{q})$ such that $\boldsymbol{\nu} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$. While the second is invertible analytically and used to control the robot, the first one may only be used to estimate an operational velocity given the state of the robot.
- If we want to control the redundancy of a system, our claim is that it is necessary to get joint velocity kinematics in the whole joint space. Furthermore, as the system we control spans an infinity of possible configurations using its redundancy, we have to learn an inverse dynamics model spanning a significant set of those configurations. Even if learning on a large input space is computationally expensive, it seems impossible to control redundancy with models learnt only along joint trajectories. This leads to bigger models (or less precise ones) but it is the only way to control the internal mobility everywhere.

In our approach, we restrain our operational space parameters to linear velocities without considering angular velocities[1] to decrease the number of dimensions of the learning problem.

The constraint to get the forward velocity kinematics under a matrix form results in two different learning approaches depending on the algorithm we use. Indeed, as we detail in the following subsections, the way to deal with this problem is different depending on whether we use LWPR or XCSF.

[1] It seems that there is no conceptual problem in learning a Jacobian matrix to control the orientation. This can be done by using as input the joint position $\boldsymbol{q}$ and as output the orientation part of the operational space parameters $\boldsymbol{\xi}$ represented with quaternions.

3.1.1 Learning forward velocity kinematics with LWPR

A solution that satisfies the constraints expressed above consists in learning a forward kinematic model with LWPR and using the capability of the algorithm to provide the first order derivative of the output relatively to the input under a matrix form (see Section 2.1.4). To use LWPR, we need to learn an initial model based on random data from the analytical model. We compute, for each random datum, the operational position depending on the analytical kinematics as $\boldsymbol{\xi} = \mathcal{FK}(\boldsymbol{q})$ in order to feed the learning algorithm with the corresponding $(\boldsymbol{q}, \boldsymbol{\xi})$ pairs. When the input is the joint position $\boldsymbol{q}$ and the output is the operational space parameters $\boldsymbol{\xi}$, the derivative of the output relatively to the input corresponds to the Jacobian matrix linking linear velocities

$$\mathcal{FK}_{\text{LWPR}} = \text{LWPR}_{learn}([\boldsymbol{q}], [\boldsymbol{\xi}])$$

where $\mathcal{FK}_{\text{LWPR}}$ is the forward kinematics learnt with LWPR. It is important to stress here that there is one LWPR model for each dimension of $\boldsymbol{\xi}$ which are all independent.

Those data are chosen randomly between joint limits in order to span a large variety of input values $\boldsymbol{q}_{limit} = [0, 2\pi]$. As we reduce the learning space to an interval, it may induce continuity problems for the learnt models which should be continuous. For instance, if the joint position oscillate around $2\pi rad$, the used part of the model will switch from 0 to 2π.

Thus, it is possible to use the learnt model to compute the prediction

$$\left[\hat{\boldsymbol{\xi}}, \hat{J}\right] = \text{LWPR}_{predict}\left([\mathcal{FK}_{\text{LWPR}}], [\boldsymbol{q}]\right)$$

3.1.2 Learning forward velocity kinematics with XCSF

As explained in Section 2.1.5, XCSF distinguishes between a prediction space and a condition space. To illustrate the possibility it opens, we recall the kinematic Equation 1.12 $\boldsymbol{\nu} = J(\boldsymbol{q})\,\dot{\boldsymbol{q}}$ where the Jacobian matrix J relates $\boldsymbol{\nu}$ to $\dot{\boldsymbol{q}}$ and is only dependent on the joint parameters $\boldsymbol{q}$.

We can get such a representation using Equation 2.18 applied to forward velocity kinematics

$$\mathcal{FVK}_{\text{XCSF}} = \text{XCSF}_{learn}([\boldsymbol{q}], [\dot{\boldsymbol{q}}], [\boldsymbol{\nu}]) \tag{3.1}$$

where $\boldsymbol{q}$ defines the condition space, i.e., the space specifying the explicit dependency of the model, the space where the model is learnt, whereas the $[\dot{\boldsymbol{q}}], [\boldsymbol{\nu}]$ pair defines the prediction space used to compute the errors and create the model. The model $\mathcal{FVK}_{\text{XCSF}}$ represents the model of XCSF which includes all classifiers with their respective parameters. It has to be distinguished from the estimated Jacobian $\hat{J}(\boldsymbol{q})$.

Each classifier in the XCSF population P contains a local matrix. But the domain of these rules overlap. Thus, extracting the Jacobian matrix of a particular $\boldsymbol{q}$ from the

model involves some weighted combination

$$\hat{J}(\boldsymbol{q}) = \sum_{i=1}^{n_M} \phi_i(\boldsymbol{q})\,\beta_i. \tag{3.2}$$

where n_M is the number of classifier in the match set. As the match set is computed at each time step, the selection made among all classifiers may vary from a time step to another. Thus, the estimated Jacobian matrix may not be continuous.

Equation 3.2 will be noted

$$\hat{J}(\boldsymbol{q}) = \text{XCSF}_{predictJ}\left([\mathcal{FVK}_{\text{XCSF}}], [\boldsymbol{q}]\right) \tag{3.3}$$

when the model is only used to predict a Jacobian matrix.

3.1.3 Combining tasks

We want to combine several tasks hierarchically so that the lesser priority tasks do not perturb the higher priority tasks. As described in Figure 3.1 and in Figure 3.2, we use both model learning schemes within the RMRC formalism detailed in Section 1.2. If we want to control the dynamics of the system, the architecture presented in Section 1.2 needs to be enhanced. We have stated that, in order to control dynamics, it is necessary to have access to the desired joint accelerations to compute the associated torques. However, an alternative control loop, illustrated in both figures, consists in approximating the desired acceleration by the velocity as : $\ddot{\boldsymbol{q}}^\star = (\dot{\boldsymbol{q}}^\star - \dot{\boldsymbol{q}})/dt$. This approach has the advantage of avoiding to compute the derivative of the Jacobian matrix.

In order to do so, we must first learn the models necessary to achieve these tasks separately with LWPR as

$$\left[\hat{\boldsymbol{\xi}}_1, \hat{J}_1\right] = \text{LWPR}_{predict}([\mathcal{FK}_{1\text{LWPR}}], [\boldsymbol{q}])$$

and

$$\left[\hat{\boldsymbol{\xi}}_2, \hat{J}_2\right] = \text{LWPR}_{predict}([\mathcal{FK}_{2\text{LWPR}}], [\boldsymbol{q}])$$

and with XCSF as

$$\hat{J}_1(\boldsymbol{q}) = \text{XCSF}_{predictJ}\left([\mathcal{FVK}_{1\text{XCSF}}], [\boldsymbol{q}]\right) \tag{3.4}$$

and

$$\hat{J}_2(\boldsymbol{q}) = \text{XCSF}_{predictJ}\left([\mathcal{FVK}_{2\text{XCSF}}], [\boldsymbol{q}]\right). \tag{3.5}$$

Each desired operational velocity $\boldsymbol{\nu}_i^\star$ is computed using the task space parameters error

$$\boldsymbol{\nu}_i^\star = K_{pi}\left(\boldsymbol{\xi}_i^\dagger - \boldsymbol{\xi}_i\right) \tag{3.6}$$

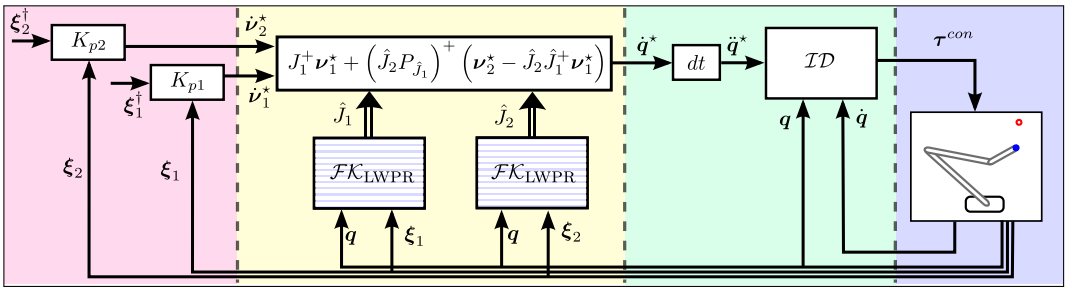


Figure 3.1: Our control architecture using LWPR where we derive learnt forward kinematic models and invert them to control two end-effectors with different tasks. The learnt parts are hashed coloured boxes.

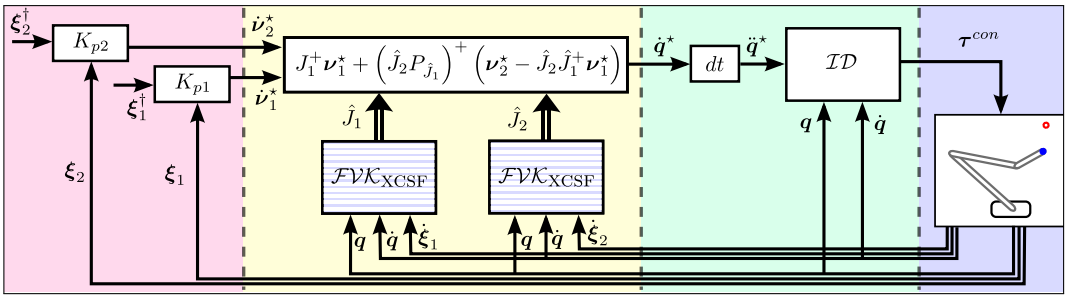


Figure 3.2: Our control architecture using XCSF to learn two forward velocity kinematic models and invert them to control two end-effectors with different tasks. The learnt parts are hashed coloured boxes.

where K_{pi} is a symmetric positive definite matrix and $\boldsymbol{\xi}_i^\dagger$ denotes the final desired position to reach.

We have shown in Section 1.2.3 that, in the case of complex missions where the tasks and constraints constantly evolve, one cannot ensure compatibility between tasks at all times. Thus, the solution chosen to invert the kinematics while combining multiple tasks in a hierarchical manner is the one of [Maciejewski and Klein, 1985] (see Equation 1.34)

$$\dot{\boldsymbol{q}}^\star = \hat{J}_1^+\boldsymbol{\nu}_1^\star + \left(\hat{J}_2 P_{\hat{J}_1}\right)^+ \left(\boldsymbol{\nu}_2^\star - \hat{J}_2\hat{J}_1^+\boldsymbol{\nu}_1^\star\right) \tag{3.7}$$

where $\hat{J}_1$ and $\hat{J}_2$ are the Jacobian matrices estimated by a learning algorithm. $P_{\hat{J}_1}$ and pseudo-inverses of $\hat{J}_1$ and $\hat{J}_2 P_{J_1}$ are computed using their SVD as presented in Section 1.2.2.

In the projection method case, the simulation can be summed up by Algorithm 3.

This projection method is more versatile than the extended Jacobian matrix method

Algorithm 3 Simulation loop with projection method

$\boldsymbol{\nu}_1^\star(t) = K_p\left(\boldsymbol{\xi}_1^\dagger(t) - \boldsymbol{\xi}_1(t)\right)$ *desired operational velocity*

$\boldsymbol{\nu}_2^\star(t) = K_p\left(\boldsymbol{\xi}_2^\dagger(t) - \boldsymbol{\xi}_2(t)\right)$ *desired operational velocity*

$\dot{\boldsymbol{q}}^\star(t) = \hat{J}_1^+\boldsymbol{\nu}_1^\star(t) + \left(\hat{J}_2 P_{\hat{J}_1}\right)^+\left(\boldsymbol{\nu}_2^\star(t) - \hat{J}_2\hat{J}_1^+\boldsymbol{\nu}_1^\star(t)\right)$ *desired joint velocity*

$\ddot{\boldsymbol{q}}^\star(t) = \left(\dot{\boldsymbol{q}}^\star(t) - \dot{\boldsymbol{q}}(t)\right)/dt$ *desired joint acceleration*

$\boldsymbol{\tau}^{con} = \mathcal{ID}\left(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}^\star(t)\right)$ *desired torques*

$\dot{\boldsymbol{q}}(t+dt) \leftarrow \mathcal{D}\left(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \boldsymbol{\tau}^{con}\right)$ *simulator integration*

as tasks do not have to always be compatible. Nevertheless, to compare the approaches, we also implement the extended Jacobian matrix method that can be written

$$\dot{\boldsymbol{q}}^\star = \begin{bmatrix} \hat{J}_1 \\ \hat{J}_2 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\nu}_1^\star \\ \boldsymbol{\nu}_2^\star \end{bmatrix}. \tag{3.8}$$

When using this approach, we do not learn the inverse mapping as in D'Souza et al. [2001] but rather the forward mapping which we inverse.

The simulation using the extended Jacobian matrix method can be summed up by Algorithm 4.

Algorithm 4 Simulation loop with the extended Jacobian matrix method

$\boldsymbol{\nu}_1^\star(t) = K_p\left(\boldsymbol{\xi}_1^\dagger(t) - \boldsymbol{\xi}_1(t)\right)$ *desired operational velocity*

$\boldsymbol{\nu}_2^\star(t) = K_p\left(\boldsymbol{\xi}_2^\dagger(t) - \boldsymbol{\xi}_2(t)\right)$ *desired operational velocity*

$\dot{\boldsymbol{q}}^\star(t) = \begin{bmatrix} \hat{J}_1 \\ \hat{J}_2 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\nu}_1^\star(t) \\ \boldsymbol{\nu}_2^\star(t) \end{bmatrix}$ *desired joint velocity*

$\ddot{\boldsymbol{q}}^\star(t) = \left(\dot{\boldsymbol{q}}^\star(t) - \dot{\boldsymbol{q}}(t)\right)/dt$ *desired joint acceleration*

$\boldsymbol{\tau}^{con} = \mathcal{ID}\left(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}^\star(t)\right)$ *desired torques*

$\dot{\boldsymbol{q}}(t+dt) \leftarrow \mathcal{D}\left(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \boldsymbol{\tau}^{con}\right)$ *simulator integration*

In the case of velocity controlled robots, $\dot{\boldsymbol{q}}$ obtained from Equation 3.7 or from Equation 3.8 can be used as the control input of the robot.

An additional motivation for learning explicitly the Jacobian matrix rather than learning the inverse velocity kinematic mapping is the dimension of the input. The explicit Jacobian matrix can be learnt using the derivative of the forward kinematics. From one side, to learn the forward velocity kinematic model, the input is the vector of joint position $\boldsymbol{q}$ and the output is the vector of operational position $\boldsymbol{\xi}$. From the other side, to learn the inverse velocity kinematic mapping, we need to provide $\boldsymbol{q}$ and $\boldsymbol{\nu}$ as input and $\dot{\boldsymbol{q}}$ as output. The dimension of the input is thus bigger when we learn the forward velocity kinematic mapping instead of the Jacobian matrix.

In this approach, we propose to learn the forward velocity kinematics giving as input

$\boldsymbol{q}_a$ which is the remainder of the division of $\boldsymbol{q}$ by 2π.

What we have described so far is enough to control a robot that would be perfectly servoed in position and velocity. We describe in Chapter 4 and Chapter 5 the results that we obtained within diverse contexts where simulated system or robots are controlled in position and velocity.

3.2 Learning inverse dynamics

We may learn inverse dynamics along a trajectory, however the control of redundancy in a non-static manner in the case of tasks combination or constraints switching may lead to the exploration of unexpected parts of the state space where the model of the system is not learnt. So, for the same reason as for the kinematic case, controlling the redundancy implies to learn the inverse dynamics over the whole state space and for all possible values of joint space acceleration.

It is also possible to learn inverse dynamics expressed in operational space (see Section 2.2.3). In task space, the dimension of each inverse dynamic model is lower than in joint space but it is necessary to have one dynamical model for each controlled end-effector. Thus, as learning dynamics in task space increases the complexity of the global architecture, we have chosen to learn it in the joint space.

The dynamic simulation, similar to the one presented in the previous section, can be summed up by Algorithm 5.

Algorithm 5 Simulation loop learning dynamics

$\boldsymbol{\nu}_1^\star(t) = K_{p1}\left(\boldsymbol{\xi}_1^\dagger(t) - \boldsymbol{\xi}_1(t)\right)$	*desired operational velocity*
$\boldsymbol{\nu}_2^\star(t) = K_{p2}\left(\boldsymbol{\xi}_2^\dagger(t) - \boldsymbol{\xi}_2(t)\right)$	*desired operational velocity*
$\dot{\boldsymbol{q}}^\star(t) = \hat{J}_1^+\boldsymbol{\nu}_1^\star(t) + \left(\hat{J}_2 P_{\hat{J}_1}\right)^+\left(\boldsymbol{\nu}_2^\star(t) - \hat{J}_2\hat{J}_1^+\boldsymbol{\nu}_1^\star(t)\right)$	*desired joint velocity*
$\ddot{\boldsymbol{q}}^\star(t) = \left(\dot{\boldsymbol{q}}^\star(t) - \dot{\boldsymbol{q}}(t)\right)/dt$	*desired joint acceleration*
$\boldsymbol{\tau}^{con} = \hat{\mathcal{ID}}\left(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}^\star(t)\right)$	*desired torques*
$\dot{\boldsymbol{q}}(t+dt) \leftarrow \mathcal{D}\left(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \boldsymbol{\tau}^{con}\right)$	*simulator integration*

It is important to note here that, unlike some other approaches in the literature, the dynamics model being first learnt on the whole spaces (joint positions, velocities and accelerations), if a new task has to be executed, the kinematic model corresponding to this new task must be learnt whereas the inverse dynamics model is already known. The learnt model is global and can be adapted for every task.

3.2.1 Learning inverse dynamics with LWPR

LWPR allows us to approximate the inverse dynamics as a relation/mapping between the input $[\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t+dt)]$ and the output $[\boldsymbol{\tau}^{con}(t)]$.

$$\mathcal{ID}_{\text{LWPR}} = \text{LWPR}_{learn}\left(\left[\boldsymbol{q}\left(t\right), \dot{\boldsymbol{q}}\left(t\right), \ddot{\boldsymbol{q}}\left(t+dt\right)\right], \left[\boldsymbol{\tau}^{con}\left(t\right)\right]\right)$$

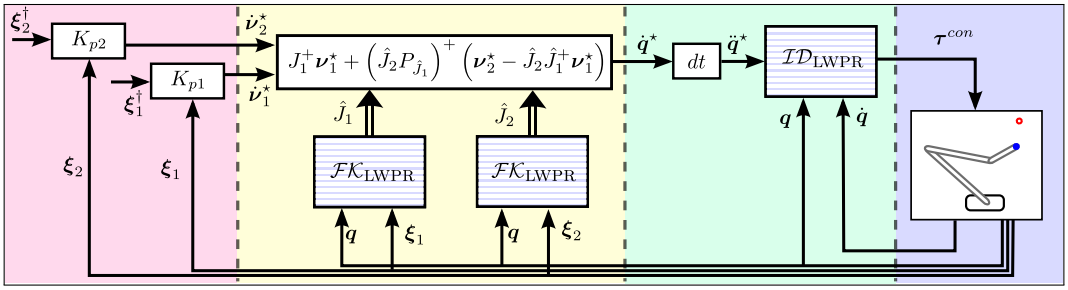


Figure 3.3: Our control architecture, deriving learnt forward kinematics, inverting it and using learnt inverse dynamics to compute control torques. The learnt parts are hashed coloured boxes.

It is then possible to control our system with the learnt inverse dynamics replacing our previous controller with

$$\boldsymbol{\tau}^{con}\left(t\right) = \text{LWPR}_{predict}\left(\mathcal{ID}_{\text{LWPR}}, \left[\boldsymbol{q}\left(t\right), \dot{\boldsymbol{q}}\left(t\right), \ddot{\boldsymbol{q}}^\star\left(t+dt\right)\right]\right)$$

where $\ddot{\boldsymbol{q}}^\star\left(t+dt\right) = \dfrac{\dot{\boldsymbol{q}}^\star\left(t+dt\right) - \dot{\boldsymbol{q}}\left(t\right)}{dt}$ is the desired acceleration (see Figure 3.3).

3.2.2 Learning inverse dynamics with XCSF

In order to highlight the way XCSF can learn the inverse dynamics in contrast with LWPR, one has to come back to the Equation 1.16 of the dynamics of a poly-articulated without perturbations forces

$$\boldsymbol{\tau}^{con} = A\left(\boldsymbol{q}\right)\ddot{\boldsymbol{q}} + \boldsymbol{b}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}\right). \tag{3.9}$$

This equation can be viewed as a linear relation between $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\tau}^{con}$ with dependencies on $\boldsymbol{q}\left(t\right)$ and $\dot{\boldsymbol{q}}\left(t\right)$

$$\boldsymbol{\tau}^{con} = \begin{bmatrix} A\left(\boldsymbol{q}\right) & \boldsymbol{b}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}\right) \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ 1 \end{bmatrix}. \tag{3.10}$$

In this context, $\boldsymbol{b}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}\right)$ can be seen as an offset.

Thus, similarly to the velocity kinematic case described in the previous section, one can learn an inverse dynamic model by providing at each time step a state $\left(\boldsymbol{q}\left(t\right), \dot{\boldsymbol{q}}\left(t\right)\right)$ as condition parameters and a $\left(\left[\ddot{\boldsymbol{q}}\left(t+dt\right)\right], \left[\boldsymbol{\tau}^{con}\left(t\right)\right]\right)$ pair as input of the prediction space.

$$\mathcal{ID}_{\text{XCSF}} = \text{XCSF}_{learn}\left(\left[\boldsymbol{q}\left(t\right), \dot{\boldsymbol{q}}\left(t\right)\right], \left[\ddot{\boldsymbol{q}}\left(t+dt\right)\right], \left[\boldsymbol{\tau}^{con}\left(t\right)\right]\right) \tag{3.11}$$

As XCSF distinguishes between a prediction space and a condition space, this approach is much less expensive in terms of dimensionality than the one used in LWPR. Indeed, one needs to span $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ instead of $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$ to learn the model. As a result, the space is 1/3 smaller.

As illustrated in Figure 3.4, the control torque is computed as

$$\boldsymbol{\tau}^{con} = \text{XCSF}_{predict}\left(\mathcal{ID}_{\text{XCSF}}, [\boldsymbol{q}, \dot{\boldsymbol{q}}], [\ddot{\boldsymbol{q}}]\right). \tag{3.12}$$

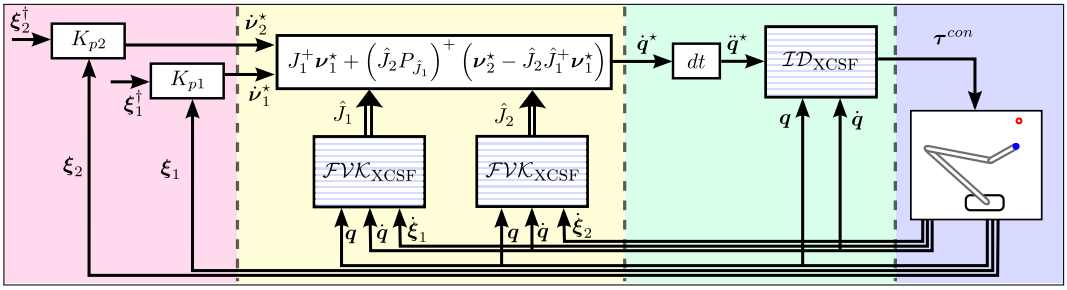


Figure 3.4: Our control architecture, deriving learnt forward kinematics, inverting it and using learnt inverse dynamics to compute control torques. The learnt parts are hashed coloured boxes.

3.3 Evaluation of the prediction quality

To evaluate the quality of the prediction of the learning algorithms, we can use different types of prediction errors.

The *Mean Square Error* (MSE), also named Residual Sum of Squares divided by the number of data, is a standard evaluation of the quality of the prediction of a learning algorithm on a fixed number of data. It measures the average of the square of the error between a real value and its estimate

$$\text{MSE} = \frac{1}{n_p} \sum_{i}^{n_p} (y_i - \hat{y}_i)^2$$

where n_p is the number of points used to compute this error, y_i is the i^{th} value of the data obtained by the analytical model of the robot and $\hat{y}_i$ is the i^{th} predicted value of the learnt model.

The *Root Mean Square Error* (RMSE) is the square root of the MSE. This prediction quality is appreciated as it performs an estimation which has the same dimension and

the same unit as y_i

$$\text{RMSE} = \sqrt{\frac{1}{n_p}\sum_{i}^{n_p}(y_i - \hat{y}_i)^2}.$$

The *Mean Absolute Error* (MAE), an estimator which has also the same dimension and unit as y_i, is computed as

$$\text{MAE} = \frac{1}{n_p}\sum_{i=1}^{n_p}\|y_i - \hat{y}_i\|.$$

Every evaluation of the prediction is computed on a one dimensional input. The error of each dimension may be normalized to be compared with each other and dealing with the case where each dimension has different ranges; for example to compare two joint torques with strongly different strengths. The normalization is used to be performed by either the variance or the difference between the maximum and the minimum of the presented data.

In this thesis, we use the NMSE to compare our results with those of the literature.

Thus we use the *Normalized Mean Square Error*(NMSE) computed as

$$\text{NMSE} = \frac{1}{\sigma^2}\frac{1}{n_p}\sum_{i}^{n_p}(y_i - \hat{y}_i)^2$$

where σ^2 is the sample variance of y: $\sigma^2 = \frac{1}{n_p}\sum_{k=1}^{n_p}(y_k - \overline{y}_k)^2$.

To evaluate the prediction, we also use the *Normalised Mean Absolute Error* (NMAE) computed as

$$\text{NMAE} = \frac{1}{n_p\left(\max(y_i) - \min(y_i)\right)}\sum_{i=1}^{n_p}\|\hat{y}_i - y_i\|$$

where $\max(y_i)$ and $\min(y_i)$ are respectively the maximum and the minimum of the data values and n_p is the number of points used to compute the error. This error seems more appropriate as it is the only one which computes, at each time step, an error of the same dimension and the same scale as the estimated data.

3.4 Discussion

Using our approach, we can notice that errors of kinematic and dynamic models are not cumulative as they are expressed in different spaces. With both learning algorithms, the system is controlled with a loop closed kinematically in the operational space. Consequently, an approximation error in the kinematics results necessarily in an end-point position error as the controlled point is perceived at another position. However smaller errors made by the approximation of the inverse dynamic model can be rejected. In

fact, the inverse dynamic model can be view as a $\mathcal{PID}$ controller whose gains depends on the configuration. A small error in a gain may lead to a worse decoupling but not to a positioning error.

In the redundant case, learning forward kinematic models does not lead to a loss of information about the system. In our method, one can choose to add any secondary task independently from the first one and any weighting matrix W_q can be used[2] when performing the inverse of the Jacobian matrix. That is not the case when inverse models are learnt directly since this corresponds to a specific choice of inverse.

The only way to get a Jacobian matrix with LWPR is to learn the forward kinematic and to derive each local model. The error used to compute the model is thus computed with operational positions whereas we want to predict operational velocities. Conceptually, this method is thus necessarily worse than the learning scheme used with XCSF. In fact, with XCSF, the learnt model directly reflects the $\dot{\boldsymbol{\xi}} = J(\boldsymbol{q})\dot{\boldsymbol{q}}$ equation as it generates model in joint space, making the relation between operational and joint velocities. The learning error is thus directly computed with operational velocities. Conceptually, using XCSF as learning algorithm in our control scheme seems more adapted and more natural to learn a Jacobian matrix.

[2]The use of a proper weighting matrix different from the Identity can be crucial in the dynamic case, see [Khatib, 1987].

4 Simulations on combining multiple tasks

Contents

In this chapter, we focus on learning and controlling the velocity kinematics, considering that the inverse dynamics is known. We present simple simulations of a three degrees of freedom planar arm designed to compare different tasks, over-constrained or not, using learnt models.

We compare our approach to the one of D'Souza et al. [2001] where the inversion problem that arises in the redundant case is solved, in a less generic manner, during learning and not during control (see Section 3.1). The corresponding results have been published in [Salaün et al., 2009a, 2010]. Preliminary results were published in [Salaün et al., 2009b]. This comparison is drawn using models learn both with LWPR. We also compare the performance and applicability of both LWPR and XCSF.

In this chapter, we consider, when we learn a model from random data, that it is possible to determine, at each moment, the operational position corresponding to any configuration. This raises no difficulty in simulation where we can successively compute the analytical forward kinematics of different random joint positions, whereas this becomes difficult using a real robot.

4.1 A rigid body dynamics simulator with contacts: ARBORIS

To simulate our robots, we use ARBORIS, a dynamic simulator which was first implemented in matlab [Barthélemy and Micaelli, 2008] and which is now recoded in python[1]. Based on the algorithms described in [Murray et al., 1994], [Liu and Wang, 2005] and [Park, 2005], the resulting simulator can be regarded as an implementation of the work of Duindam [2006]. To simulate dynamics of rigid bodies, it uses the Boltzmann-Hamel equation

$$A(t)\ddot{\boldsymbol{q}}\,(t+dt) + \boldsymbol{b}(t)\dot{\boldsymbol{q}}(t+dt) = 0. \tag{4.1}$$

where A and $\boldsymbol{b}$ are respectively the mass matrix and the non-linear effects matrix which includes the viscosity. This equation, similar to Equation 1.6, updates the dynamics without involving contacts, actuation or gravity.

ARBORIS calls upon a method which includes the actuation forces applied to a robot and a resolution of additional constraints such as contacts and kinematic loops based on the Gauss-Seidel algorithm. Thus, it uses time stepping and a semi-implicit Euler integration scheme to update the joint positions [Liu and Wang, 2005].

The simulator integrates different types of robot such as the HUMAN36 robot copied from the simulator HUMANS [Wieber, 2006], the humanoid robot ICUB which model is based on the CAD of the ROBOTCUB website[2], few classic architectures such as PUMA or SCARA and some simpler robots (2 or 3 degrees of freedom planar arms) designed to test algorithms.

We have chosen to set the integration time step of the simulator to 10 milliseconds in all simulations.

We have chosen to evaluate the compared approaches on a 3 degrees of freedom planar system, shown in Figure 4.1. Sticks lengths are 0.50m, 0.40m and 0.20m with respective masses of 1kg, 0.8kg and 0.2kg.

4.2 Initializing an inverse kinematic model with LWPR

To learn our models, we use the LWPR algorithm implementation from the Institute of Perception, Action and Behaviour by the Statistical Machine Learning and Motor Control Group[3] of the University of Edinburgh.

Unlike XCSF, LWPR has a more complicated initialization and tuning. Thus we learn an initial model based on random data from the analytical model while tuning different parameters. Each parameter has an influence on the prediction quality. In this section, we detail the influence of the parameters used in LWPR using the errors estimation of the quality of the prediction described in Section 3.3.

[1] http://github.com/sbarthelemy/arboris-python
[2] http://www.robotcub.org
[3] http://www.ipab.inf.ed.ac.uk/slmc/software/lwpr/

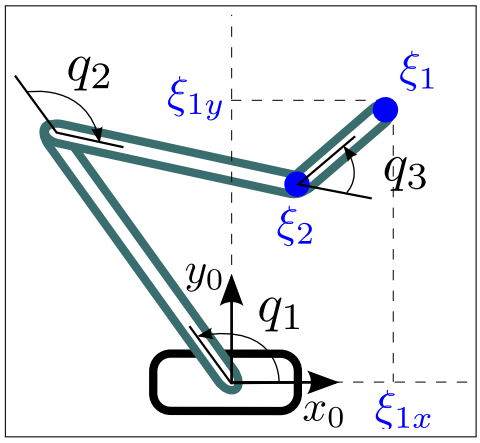


Figure 4.1: Schematic view of our simulated system.

4.2.1 Tuning parameters of LWPR

LWPR is difficult to tune. As it comes with some parameters that need to be initialised, we perform this tuning as proposed by Klanke et al. [2008]. Each parameter is successively tuned experimentally from a set of random data obtained from the system. The data are joint positions chosen randomly and their corresponding operational positions.

We have seen in Equation 2.6 that the D parameter is proportional to the inverse of the variance of each receptive field. Thus, D is responsible for the size of the Gaussians. Small D induce wide receptive fields. As those models evolve statistically in function of the input and output, the D parameter has to be initialized. The $init_D$ coefficient corresponds to the initial size of all receptive fields. This coefficient significantly affects the convergence time of LWPR.

Another important parameter for our simulations is w_{gen}. It is a threshold responsible for the creation of a new local model. Higher is its value, more it facilitate the creation of new receptive fields. It is thus responsible for the global complexity of the model. Equation 2.6 computes the weight of each Receptive field for a given input and, if no weight is high enough, i.e., if $\phi_i < w_{gen}$ for all $i \in [1, n_n]$, then a new model is created with the previously tuned $init_D$ parameter and with the current input as center.

Finally, from our first experiments on fixed models, updating D is not so important once the initialisation is well done but we still keep this option as it improves the precision of the model.

The regularization penalty coefficient γ is critical to the evolution of the size of receptive fields (see Equation 2.10). As it penalizes the sum of all D squared in the minimized cost function, a small penalty term increases precision but decreases the smoothness of the model.

The α parameter is the distance metric learning rate for gradient descent during the

computation of the D parameter (see Equation 2.9 page 35). The $init_\alpha$ which correspond to the initialization of the parameters α. A too large learning rate may induce instability in convergence while a too small one induces a slow convergence. Thus we have activated meta learning to alleviate the tuning of α while having a good precision.

To summarize, all parameters are grouped in Table 4.1.

Parameters	$norm_{in}$	$init_D$	$update_D$	$init_\alpha$	w_{gen}	γ
Values	2π	20	1	10000	0.5	$1e^{-6}$

Table 4.1: Parameters used to learn an inverse dynamic model with LWPR

4.2.2 Prediction quality using LWPR

As we control our robot with the Jacobian matrix, we compute the error on operational velocities and not on operational positions. The operational velocities are computed from the analytical Jacobian matrix multiplied by joint velocities, arbitrarily fixed to 1.00 $rad.s^{-1}$. The approximated operational velocities are computed from the estimated Jacobian matrix multiplied by the same arbitrarily fixed joint velocities.

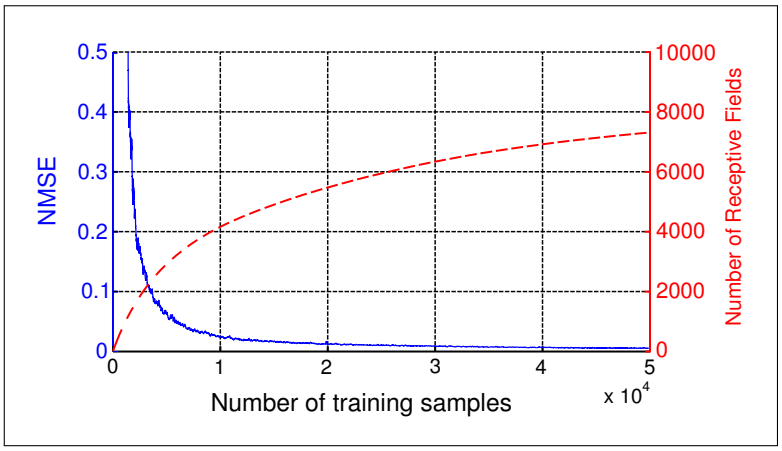


Figure 4.2: Evolution of the NMSE of the task space velocity prediction (blue, plain line, scale left) and of the number of receptive fields for one output (red, dashed line, scale right) using LWPR (average over 40 trials). The 50000 joint configurations are chosen randomly.

As can be seen on Figure 4.2, the NMSE of the predicted velocity decreases during the random bootstrap. A bootstrap phase with 5000 samples is, in this case, sufficient for LWPR to cover roughly the joint space, having an output, even bad, in each configurations, and to predict a Jacobian matrix.

At the end of each experiment, the number of receptive fields varies between 2000 and 8000 for each output depending on the duration of the bootstrap stage.

The relatively large number of receptive fields generated in these experiments is due to two factors. The first one is the precision of the required prediction which is related to our choice of parameters for the LWPR algorithm. The second factor is more specific to the redundancy of the robot that we are exploiting. This redundancy induces possible internal motions from secondary tasks which cannot be predicted a priori. Thus, a good coverage of the joint space is necessary in addition to learning along specific trajectory.

4.3 Combining tasks with LWPR as learning algorithm

In this section, we detail three different constrained cases and associated tasks in order to study the robustness of our control scheme.

While, during the bootstrap, the presented results were averaged over 40 trials, the results presented below are just representative trials.

4.3.1 Under-constrained case

The first studied task is a simple reaching task using our control scheme (see Figure 3.3). From an initial end-effector position $\boldsymbol{\xi}_1^i = [0.10\ 1.00]^T\ m$, the end-effector of the robot has to reach a target $\boldsymbol{\xi}_1^\dagger = [0.20\ 0.50]^T\ m$ with a specified precision of 0.01 meter. There is no reference trajectory but just a goal to reach. Once the task is achieved, the end-effector is sent back to its initial position using the same task controller $K_p = 5s^{-1}$. This point to point movement is repeated until the end of the simulation.

For this simple reaching task, the task space dimension is 2, thus the Jacobian matrix of the end-effector is redundant relatively to the task and there are an infinity of ways to reach the goal. In this experiment, the system is controlled as Equation 3.7 without any secondary task; the internal mobility is not controlled.

In this experiment, in order to highlight model adaptation during the control phase, we only make bootstrap using 2000 samples. Figure 4.3(a) represents the first two seconds of simulation. The model of the robot is still quite approximative and the RMRC method is too dependent of the model to compensate for its inaccuracies. Figures 4.3(b) (between 2s and 4s) and 4.3(c) (between 6s and 8s) show the evolution of the trajectories. It can be noticed that the learnt model is still adapting during the control phase. Also, the precision requirements (errors smaller than 0.01m) in terms of target reaching are met. After 20s (see Figure 4.3(d)), the precision is improved and the trajectory of the robot trajectory is almost linear as one would expect when using a resolved motion rate controller.

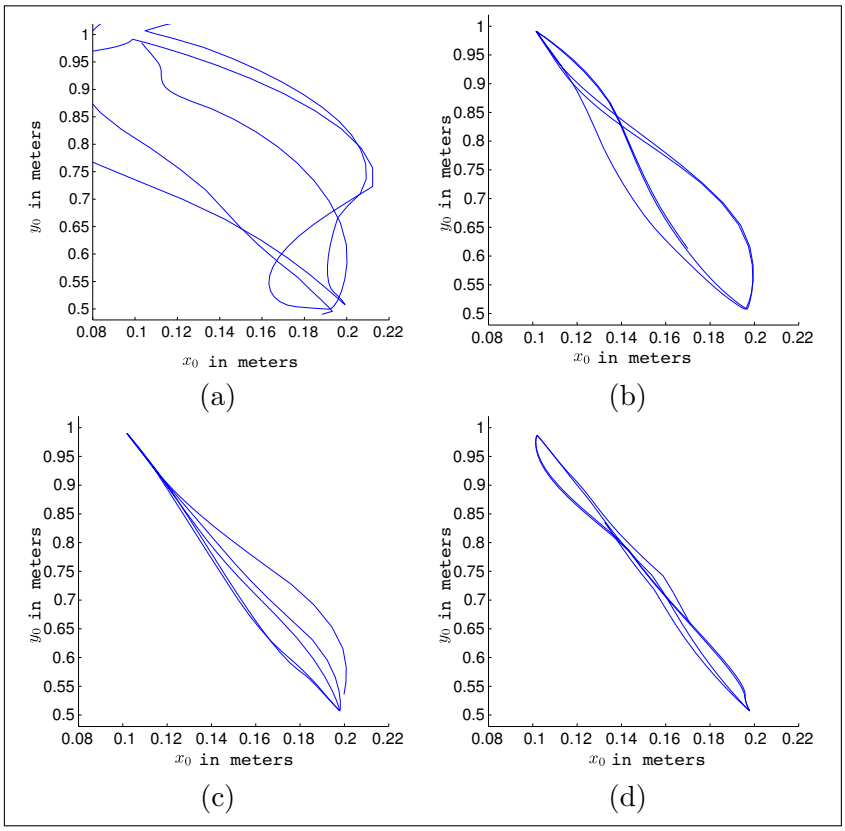


Figure 4.3: Evolution of operational trajectories while learning a Jacobian matrix realising an under-constrained task. Each figure represents two seconds of simulation.(a): 0s to 2s.(b): 2s to 4s.(c): 6s to 8s.(d): 20s to 22s.

4.3.2 Fully constrained case

The second experiment consists in reaching $\boldsymbol{\xi}_1^\dagger = [0.20\ 0.50]^T\ m$ with an end-effector parametrized by $\boldsymbol{\xi}_1$ and keeping the end-effector in this position while realising a second task. This second task consists in alternatively reaching, with another end-effector parametrized by $\boldsymbol{\xi}_2$, the goals 0.10 m and 0.30 m along the x_0 axis. The first task is a two dimensional task whereas the second one is one dimensional. As those goals can be reached simultaneously, the system is said fully constrained with respect to the tasks.

For these two tasks, we have solved the redundancy using two different algorithms: the extended Jacobian matrix [Baillieul, 1985] and the projection method [Nakamura, 1990]. In the case of the projection method, the second task is projected in the null space of the first one according to Equation 3.7. In the case of the extended Jacobian matrix method, J_{ext} is chosen as Equation 3.8 in order to simplify the inversion problem to a square, regular matrix inversion

$$\dot{\boldsymbol{q}} = J_{ext}^{-1} \begin{bmatrix} \boldsymbol{\nu}_1^\star \\ \boldsymbol{\nu}_2^\star \end{bmatrix}. \tag{4.2}$$

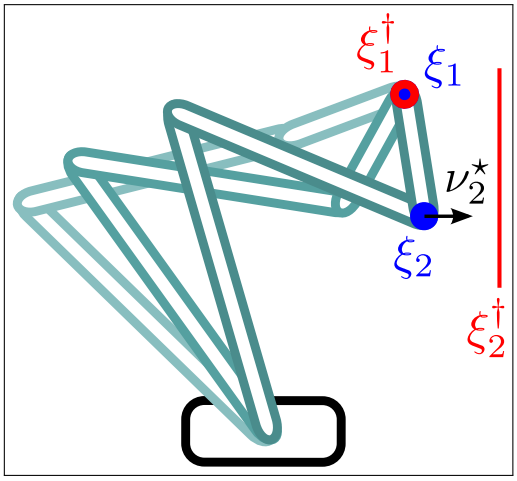


Figure 4.4: Stroboscopic view of the robot realising two prioritised tasks in the fully constrained case. The first task consists in maintaining the end-effector position ξ_1 on a two degrees of freedom goal $\xi_1^\dagger$ (a point in 2D). The second task consists in positioning a second end-effector defined by ξ_2 on a one degree of freedom goal $\xi_2^\dagger$ (a line in 2D).

In the fully constrained case, the precision requirements ($0.01m$) are also met for both tasks which respectively constrain the position of the end-effector $\boldsymbol{\xi}_1$ and the position along the $\boldsymbol{x}_0$ axis (see Figure 4.4) of the wrist of the robot $\boldsymbol{\xi}_2$.

This is illustrated in Figure 4.5 for the first task. It is shown that there is no major difference in the precision obtained when controlling redundancy using an extended Jacobian matrix or using the projector approach. From 0 to 1s, the reference task point is not reached yet, which explains the large error (the initial error, not shown in the figure, is 0.65m). After 1s, the required precision is obtained and errors are due to the cyclic change of reference point for the second task. These errors decrease with time thanks to the on-line improvement of the learnt model.

These errors are due to the fact that a learnt model is used as well as to the fact that learning errors are propagated when computing the inverse velocity kinematics mapping from the forward one. Using the extended Jacobian matrix approach or ours, if the Jacobian matrices are not accurately predicted, errors disturb the higher priority task. Similarly, when specifically using the projection method, an error in the prediction of the first task Jacobian matrix induces an error in the computation of the associated projector leading to disturbances induced by the second task on the first one.

Despite these error propagation effects, the achieved performances are satisfactory.

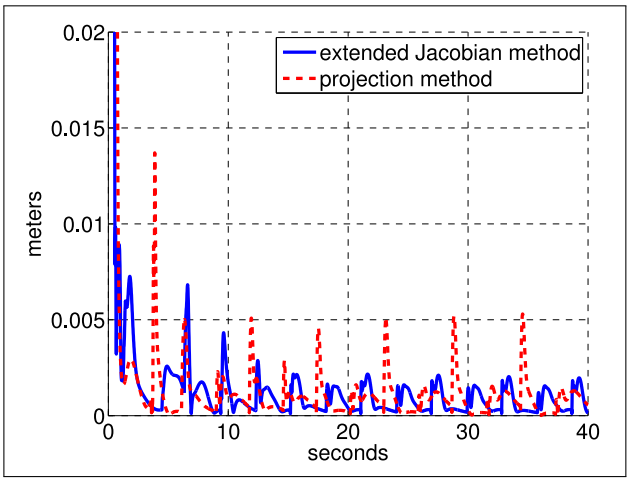


Figure 4.5: Norm of the end-effector error induced by a second compatible task for two different controllers.

4.3.3 Over-constrained case

The last experiment is similar to the previous one. The first task is identical whereas the second one is a two dimensional task for the end-effector parametrized by $\boldsymbol{\xi}_2$ which has to reach $\boldsymbol{\xi}_2^\dagger = [0.45\ 0.25]^T\ m$. This second task is not compatible with the first one. The system is over-constrained.

Regarding this experiment, the projection method is the only one that is tested since the extended Jacobian matrix method would require the same projection in order to obtain a square Jacobian matrix J_{ext} as

$$\dot{\boldsymbol{q}} = \begin{bmatrix} J_{\xi_1} \\ P_{J_{\xi_1}} J_{\xi_2} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\nu}_1^\star \\ \boldsymbol{\nu}_2^\star \end{bmatrix}. \tag{4.3}$$

The results obtained from the last experiment illustrate the effectiveness of the projection method. The controller maintains the distance between the end-effector point $\boldsymbol{\xi}_1$ and the desired reference point $\boldsymbol{\xi}_1^\dagger$ under 0.01m. At the same time, the second task is partially achieved as expected from the chosen redundancy resolution scheme. It is achieved with the minimum possible error and without inducing any disturbance on the first task: the task hierarchy is respected.

These results are illustrated in Figure 4.6 where the final configuration of the system is shown as well as intermediate configurations, illustrating the convergence of the second task toward the best possible result.

Figure 4.7 gives a view of the positioning errors for both tasks. Similarly to the last experiment, error propagation effects are present but once again the end-effector error in position is very low. We can see that between the second and the third second of

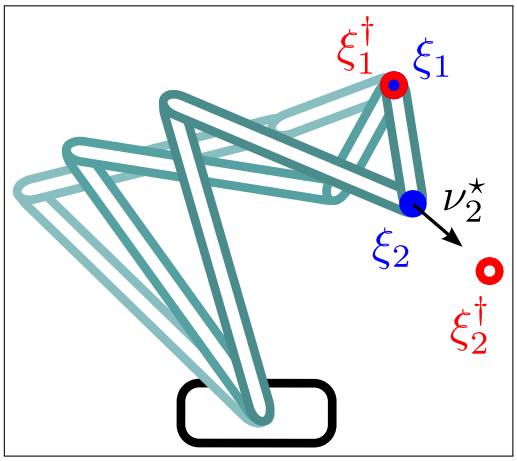


Figure 4.6: Stroboscopic view of the robot realising two prioritised tasks in the incompatible case. The task hierarchy is respected as the end-effector $\boldsymbol{\xi}_1$ stays fixed on $\boldsymbol{\xi}_1^\dagger$ while the distance between $\boldsymbol{\xi}_2$ and $\boldsymbol{\xi}_2^\dagger$ is minimum.

simulation, the error of the second task is lower than the steady state error. It can happen only because the end-effector have not reach yet its final position.

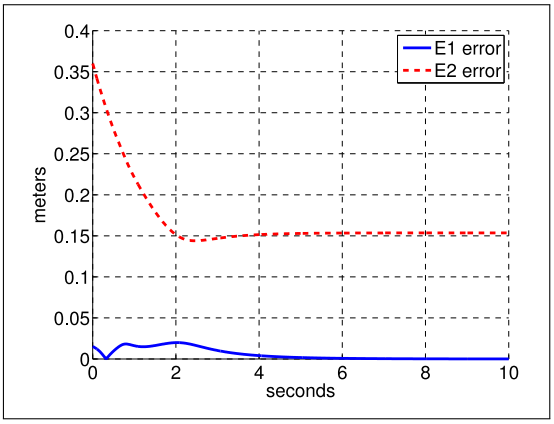


Figure 4.7: Reaching errors for the first (blue, plain line) and second (red, dashed line) tasks in the incompatible case after learning.

4.4 Learning velocity kinematics with XCSF

We use the XCSF implementation from the COgnitive BOdy Spaces: Learning And Behavior (COBOSLAB) of the University of Würzburg[4].

In comparison to LWPR, XCSF is easier to tune. The parameters which have to be tuned are by order of importance:

- the threshold for the error, set to 0.1, under which the system stops adding new classifiers but tries instead to generalize,
- the maximum number of micro-classifiers which has been set to 350,
- the maximum number of iterations set to 150000,
- the number of iterations where we condensate the model to have a better generalization, set to 100000,
- the number of runs, set to 20 to ensure a global stability of the model.

All other parameters are set to their default value, as specified in XCSF configuration file. In contrast with LWPR, XCSF does not require a random bootstrap stage for convergence. Thus we directly study one of the reaching task presented in the previous section to provide a preliminary performance comparison.

As in Section 4.3.1, the studied task is a simple reaching task. From an initial configuration chosen randomly, the end-effector of the robot has to reach a random target $\boldsymbol{\xi}_1^\dagger$ with a specified precision of 0.01 meter. The task is chosen to be in the reachable operational space. The task is not defined as a target trajectory but just as a goal to reach. Once the task is achieved, a new random task is specified if the reachable space with the same required precision. K_p equals $5s^{-1}$. This point to point movement is repeated until the end of the simulation.

For this simple reaching task, the task space dimension is 2, thus the Jacobian matrix is redundant and there is an infinity of ways to reach the goal.

Figure 4.8 illustrates the evolution of the MAE, as a function of the number of iterations. The curve is an average over 20 runs. The variance between those runs is represented on the figure. It highlights the model adaptation from scratch during the control phase as there is no bootstrap stage before controlling the system. We note that the error decreases with a large variance. We can see in the same figure the evolution of the number of micro- and macro-classifiers. Once the maximum number of micro-classifiers is reached, the number of macro-classifiers drastically decreases during the compaction stage. Both desired performance and stability are reached after compaction (see Section 2.1.5).

In contrast with the experiments presented in Section 4.3.1, when a task is reached, the new task is chosen randomly in order to drive the system to explore the whole space. This contributes to explaining that the variance is still high just before compaction since the system may be driven to areas where the model is not learnt yet.

[4] http://medal.cs.umsl.edu/files/XCSF_Ellipsoids_Java.zip

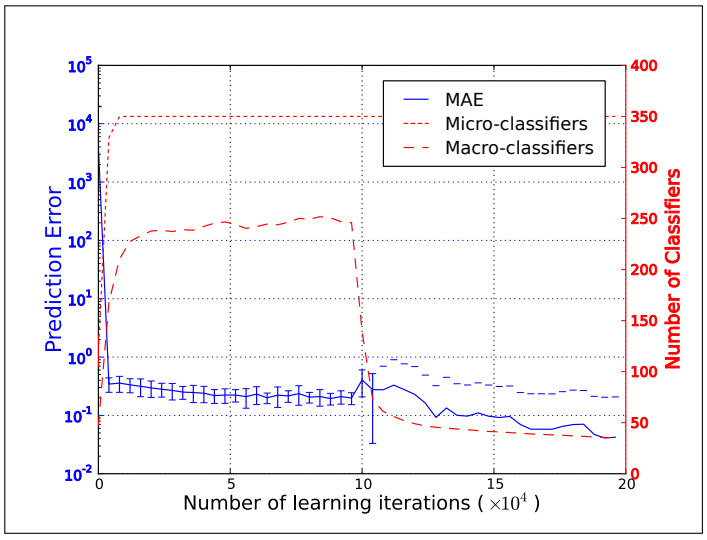


Figure 4.8: MAE (blue, plain line) for the prediction along the x axis XCSF. Number of Classifiers (red, dashed lines).

Nevertheless, one can see that after compaction, the prediction error gets very low. Figure 4.9 illustrates the operational space error between the end-effector and the task to reach using a learnt model after compaction. The system go straight to all new goals with different operational gain values.

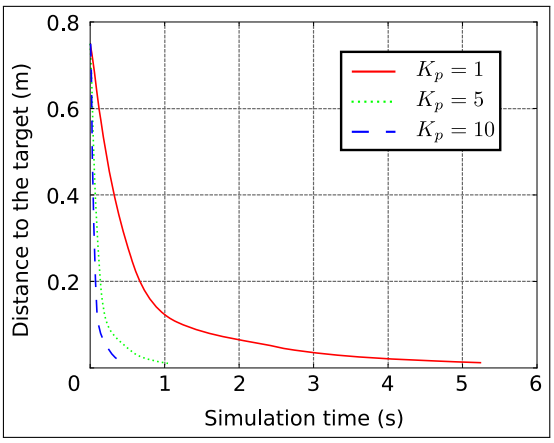


Figure 4.9: Evolution of operational trajectories while learning a Jacobian matrix with XCSF realising an under-constrained task. Each line represents a different proportional operational controller.

In order to compare those results to those presented in Section 4.3.1, we have reproduced the point to point experiments (see Figure 4.10). In the contrast with LWPR-based

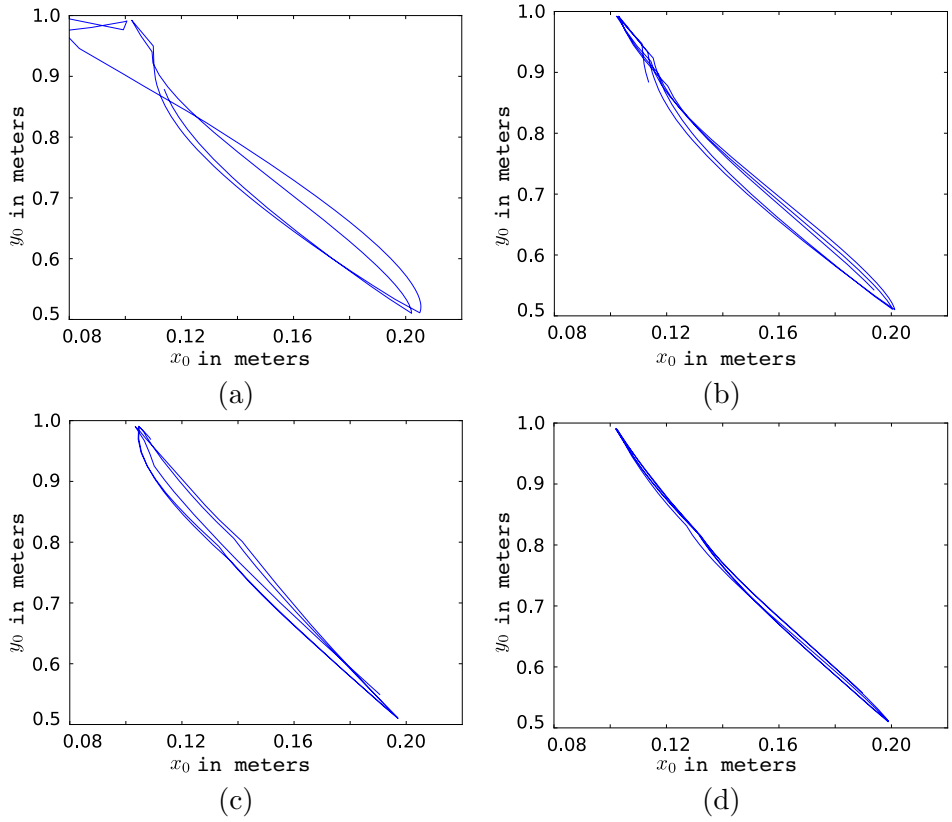


Figure 4.10: Evolution of operational trajectories while learning the forward velocity kinematics realising an under-constrained task with XCSF. Each figure represents 4 seconds of simulation.(a): 0s to 4s.(b): 4s to 8s.(c): 12s to 16s.(d): 36s to 40s.

learning scheme, there is no bootstrap here and the adaptation is made on-line. After 40 seconds, the trajectories gets straight.

4.5 Discussion

The results of the simulations for the under-constrained case show that after some random bootstrap with LWPR, or directly with XCSF, our method is able to improve the model of the system while controlling it so that a reaching task is achieved with a prescribed precision. The end-effector trajectory converges to what would be expected in the case of resolved motion rate control. A similar result has already been obtained by D'Souza et al. [2001], using the extended Jacobian matrix approach and learning directly the inverse velocity kinematics mapping on a trajectory.

One may argue that inverting a learnt mapping will lead to the amplification of learning errors. This is true. However, the results presented in this chapter demonstrate that

this approach actually provides good enough results when combined with a closed-loop controller.

The fully constrained case conveys more original results. To our knowledge, the only prior work which learns the forward kinematic model, derives it and uses it in a control loop is the work of Sun and Scassellati [2005]. Our approach is original in the sense that we use this learnt model to compute a projector allowing to combine several learnt tasks and thus to control redundancy. We show that this method induces error compared to an analytical model but the performance of the controller remains satisfactory. Still in that context and in parallel to our work, Butz and Herbort [2008] have solved an under-constrained case by adding a constraint at each time step and thus by solving a fully constrained case.

The simulation of the over-constrained case reinforces the results of the fully constrained case by showing that several learnt tasks can be combined in a hierarchical manner in the case where those tasks are not compatible. This has been a state-of-the-art result for a while in model-based control in Robotics [Nakamura, 1991]. However, to the best of our knowledge, this is the first time that such results are achieved in the case of learnt models [Salaün et al., 2009a]. The retained redundancy resolution scheme in that last case is the projection method which leads to the optimal solution for both tasks. In fact, in the over-constrained case, the only effective redundancy resolution scheme is the projection method. The extended Jacobian matrix approach can be applied in that case but in a way that requires projections similar to the one used in our approach.

The extended Jacobian matrix approach is not satisfactory in the case where models have to be learnt. Combining two tasks in a single one in order to simplify inversion leads to unnecessary constraints on the learning problem to be solved whereas it is simpler to learn elementary tasks separately. Furthermore, tasks combination is easier when the tasks are learnt separately. In the extended Jacobian matrix method, the learnt inverse velocity kinematic model depends on the supplementary task. The whole model has to be learnt again if this task changes whereas learning separately different Jacobian matrices leave them independent and changing one of them does not impact the others. Thus, the projection method is shown to be more versatile than the extended Jacobian matrix approach.

Most of the experimental work achieved in the context of this thesis was performed using LWPR as learning tool. XCSF was discovered lately, so the experiments performed with this second tool are preliminary and the insights we draw from them are only indicative. Furthermore, a direct comparison of the performance of LWPR and XCSF is difficult for several reasons.

- The performance of both algorithms is sensitive to the parameter set-up, thus the performance comparison is sensitive to the relative effort put in tuning the algorithm.
- To work properly, LWPR requires a random bootstrap stage with data spanning the input/output space, whereas XCSF, without such a stage, can quickly provide

a model used in a controller. Thus, the performance comparison is also sensitive to the effort put in the pre-tuning of LWPR.

- The errors of both algorithms are computed in different spaces. The kinematics learnt with LWPR is computed from a Cartesian position error whereas the velocity kinematics learnt with XCSF depends on an operational velocity error.

Nevertheless, some general comparative insights can be extracted from the preliminary experiments we performed.

- First, the fact that XCSF does not require a random bootstrap is an important feature of this algorithm with respect to LWPR, because performing this initialization onto the whole input/output spaces with a real robot is difficult to achieve, due to joint limits, auto-collision, initialization duration, admissible torques, etc.
- Second, the capability of XCSF to distinguish a condition space and a prediction space leads to a very elegant representation of the problem consisting in directly learning the Jacobian matrix of a robot which depends on joint position while providing as input/output a joint and operational velocity.
- Third, our preliminary simulations with XCSF tend to show that the algorithm performs very well and requires less tuning effort than LWPR to reach the same satisfactory level of precision.
- However, one must note that the performance of XCSF is much less steady than the performance of LWPR. This can be explained easily by the fact that the pre-tuning stage with LWPR guarantees a good coverage of the input/output spaces thus less sensitivity to the variation of the function to be approximated, whereas XCSF must update the model each time it reaches a subspace that was not sampled before. However, we have seen after reaching random goals and after compaction that the control with the model learnt by XCSF is more versatile and that the learnt model may be exploited with different operational gains.
- Finally, LWPR is much appreciated for its capability to learn local models in high dimensional spaces along trajectories. So far, whereas we did not measure the sensitivity of XCSF computational efficiency in the context of spaces with many dimensions. The scalability of the approach to systems with a higher number of degrees of freedom is still a matter for discussion. When using LWPR, the main bottleneck is the possibility to tune the parameters and to initialise the model with an initializing with bootstrapping stage that must be longer as the dimensionality of the system grows.

5 Experiments on the iCub humanoid robot

Contents

In this chapter, we show results of the application of the approach presented in Chapter 3 using lwpr to the iCub humanoid robot. We show how this approach, applied on a 3 degrees of freedom mechanical system in 2 dimensions, can be extended to a more complex system with 4 degrees of freedom in 3 dimensions. We first present results obtained on a simulator of the robot, then on the robot itself. This study reveals the limitations of the framework when applied on a real robot.

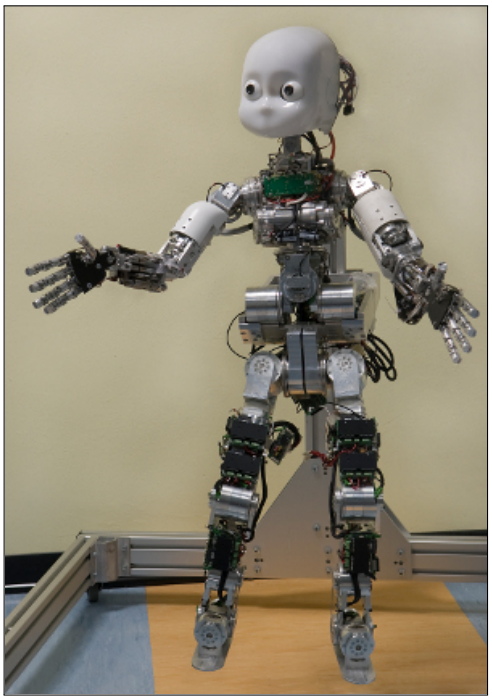
Learning a model in simulation raises no difficulty as we can successively compute the analytical forward kinematics of different random joint positions. As it becomes difficult to do it using a real robot, our bootstrap will always be performed in the iCub simulator.

In the following section, we briefly describe the iCub humanoid robot. Then we explain how we tune lwpr to finally present three experiments chosen to study the transfer from simulation to the real robot.

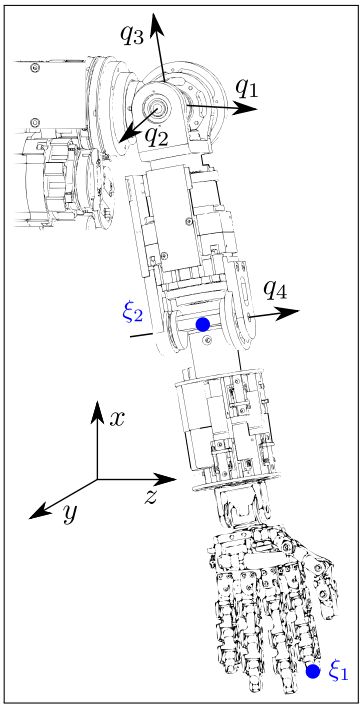
5.1 Description of iCub

iCub is a 53 degrees of freedom robot mostly actuated with DC brushless motors. It was designed in the context of the RobotCub project[1] [Metta et al., 2008] and is present in several research laboratories in Europe. The iCub software architecture is based on Yarp [Fitzpatrick et al., 2008]. Yarp provides a powerful abstraction layer allowing to test most of the codes in simulation before actually executing them on the real robot. This is made possible with the use of the iCub rigid body dynamics simulator [Tikhanoff

[1] www.robotcub.org



(a) Front view of iCub.



(b) iCub left arm front view.

Figure 5.1: The iCub humanoid robot and its left arm. The shoulder and elbow controlled axis are represented as well as the reference frame.

et al., 2008] based on ODE[2]. This simulator is also using Yarp and its software interface is thus similar to the one of the robot.

In the following experiments, we use the left arm of iCub in different contexts. The shoulder is a 3 degrees of freedom joint with some coupling between degrees of freedom. The decoupling is done with a low level controller. The elbow is a one degree of freedom joint. Each motor is deported and the transmission is made using cables and pulleys.

5.2 Initializing an inverse kinematic model with LWPR on iCub

We use lwpr to learn, on the whole joint space, forward kinematic models of two operational end-effectors which are the hand and the elbow. As described in the previous section, we provide $\boldsymbol{q}$ as input and $\boldsymbol{\xi}_1$ or $\boldsymbol{\xi}_2$ as output depending on whether we want to learn the hand to joint mapping or the elbow to joint mapping respectively.

If we want to learn forward kinematics on a real robot, the method described in Section 4.2 is not directly applicable. While it is easy to compute the model of the robot

[2] www.ode.org

in simulation, it is harder to do it in the real platform. In fact, in simulation, every configuration is reachable at any moment. Any operational position is easily computed from any random configuration. On a real robot, it is impossible to reach instantaneously a random position. If we want however to reach a random position, we can do it with a low level controller such as $\mathcal{PD}$ joint controller while checking, at each time step, that the system is not in self-collision. For those reasons, we cannot use directly random data to learn on the real system. Besides, the output operational positions have to be estimated from exteroceptive sensors such as vision. In our experiments on a real robot, joints positions are measured whereas operational positions are for now, and for the sake of simplicity, computed from a model computed with the Kinematic Dynamics Library (KDL) from the Orocos Project[3] [Smits et al., 2008]. In all experiments, we compare the KDL model to the model learnt with LWPR. Unfortunately, the LWPR model is generated with data coming from the KDL model, thus it cannot be more precise than that model.

The models used on iCub are thus first learnt in simulation and then refined on the real robot with joint positions measured from its sensors and operational positions reconstructed from the KDL model.

In order to use LWPR with sufficient precision, some parameters need to be initialised differently from the method proposed in Chapter 4. The first parameter is the input normalisation $norm_{in}$ which controls the span of the data in all dimensions. As the input are the joint positions of the robot, the $norm_{in}$ parameter is set to the range of motion of each joint (measured in degrees): $norm_{in} = [105.5, 160.8, 117, 111.5]$. $init_D$ is tuned experimentally from comparing the performance over a set of bootstrap phases to find the best value corresponding to the minimal prediction error which corresponds to $init_D = 150$. While evaluating different $init_D$ values, the $update_D$ parameter, which drives receptive fields adaptation, must be null to avoid interferences. The last parameter to be tuned is w_{gen} that we initialise to 0.2.

We consider being precise enough in the prediction with 5.10^4 training samples for the hand kinematic model and 3.10^4 training samples for the elbow kinematic model.

5.3 Combining two incompatible tasks on iCub

In Chapter 4, we presented some simulation experiments where either compatible or incompatible tasks were combined on a 3 degrees of freedom planar robot. Here, to examine how our framework scales up, we perform an experiment with two incompatible tasks with the arm of iCub both with the simulator and with the real robot. The first goal $\boldsymbol{\xi}_1^\dagger$, associated to the fingertip, has the highest priority and should be fully fulfilled. Some errors may appears due to uncertainties in the learnt model. The second goal $\boldsymbol{\xi}_2^\dagger$, associated to the elbow, is a secondary task performed with minimum error and without introducing any perturbation on the realisation of the first task. We set $\boldsymbol{\xi}_1^\dagger$ in absolute coordinates to $[0.00, -0.05, 0.03]$ meters on each axis x, y and z respectively and $\boldsymbol{\xi}_2^\dagger$ to

[3] http://www.orocos.org

$[0.00, -0.05, 0.03]$ meters. The tasks are incompatible as we want to control 6 parameters with only 4 degrees of freedom. However, in some cases, they can be compatible if the distance between $\boldsymbol{\xi}_1^\dagger$ and $\boldsymbol{\xi}_2^\dagger$ is greater than the distance between the corresponding points on the mechanical chain of the robot. When defining $\boldsymbol{\xi}_1^\dagger$ and $\boldsymbol{\xi}_2^\dagger$, we ensure that it is not the case.

$\boldsymbol{\nu}^\star$ is computed using a proportional controller such as $\boldsymbol{\xi}^\star = K_p\left(\boldsymbol{\xi}^\dagger - \boldsymbol{\xi}\right)$. We choose K_p to be diagonal and we set K_{p11} and K_{p22} respectively to $3.0s^{-1}$ and $0.2s^{-1}$.

Given the selected LWPR parameters, we approximate one forward kinematic model for each task computed with the same initialization parameters.

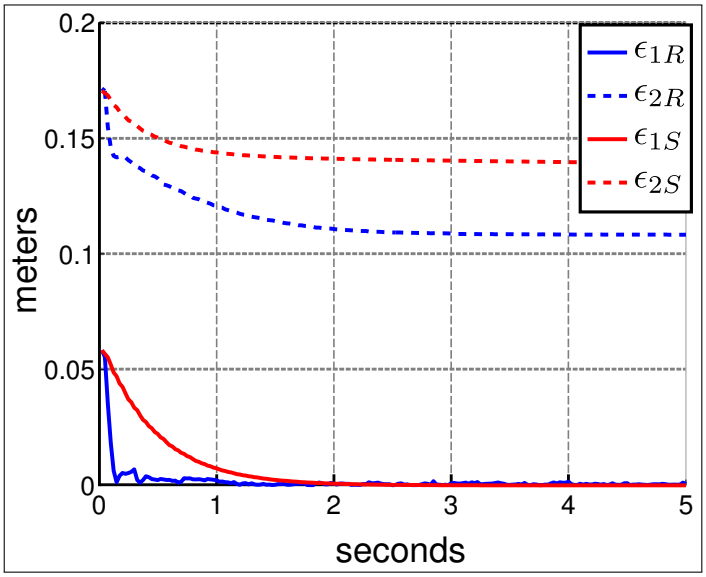


Figure 5.2: Evolution of the operational error relative to task 1 and task 2 over learning in simulation (a) and on the iCub robot (b). $\boldsymbol{\epsilon}_{1S}$ is the error in position between the end-effector $\boldsymbol{\xi}_1$ and the task $\boldsymbol{\xi}_1^\dagger$ in simulation. $\boldsymbol{\epsilon}_{2S}$ is the error relative to the second task in simulation.$\boldsymbol{\epsilon}_{1R}$ and $\boldsymbol{\epsilon}_{2R}$ are the errors relative to both tasks on the real robot.

Figure 5.2 shows the evolution of the operational error for both tasks, $\boldsymbol{\xi}_1^\dagger$ and $\boldsymbol{\xi}_2^\dagger$, using the simulated robot and the real robot, indexed with an S and an R respectively. The first task in the hierarchy converges sooner than the second one. In simulation, the achievement of the second task induces a transient error in the first task. This comes from the fact that, the learnt model of the tasks being imperfect, the projector of the second task into the null-space of the first task is slightly wrong, inducing some perturbation from the second task on the first task. As a result of the perturbation of the first task, the second task can be temporarily achieved with less error than it could if the first task was always perfectly performed.

The evolution of the operational error for both tasks using the real robot are approximately equivalent in the simulated case. In order to obtain such a similarity, the

operational controller gains have to be lowered for the real robot. This difference in operational gains explains the much smaller perturbations induced by the second task on the first task in the case of the real robot experiments: high gains amplify the coupling more than low gains. This difference in gains is due to the fact that, in the simulator, the joint level controllers are based on torque PID controllers whereas on iCub they are PID controllers based on velocity, leading to different low level controllers dynamics. The structure and/or gains of the low level controllers of the simulator should be improved and/or tuned in order to improve the similarity of the simulation with respect to the reality and minimize additional tuning when transferring to the real robot.

As a general lesson from this experiment, the transfer from the simulation to the real robot is far from immediate and requires some tuning that is not addressed by our learning framework.

5.3.1 The numpad experiment

To illustrate a reaching task corresponding to a potential real world application, we have implemented a sequential task which consists in pressing different keys on a representation of a numpad drawn on a white board. The positions of the keys are known, the organisation of the corresponding numbers can be seen in Figure 5.3. A key is considered as pressed when the end-effector is less than 0.01 meters away from the center along the x and z axes and less than 0.001 meters away along the y axis. There is no feedback sent to the robot to validate if a key is pressed or not: the operational positions are computed with KDL. The sequence that we have chosen to implement is $1 - 5 - 9 - 3$.

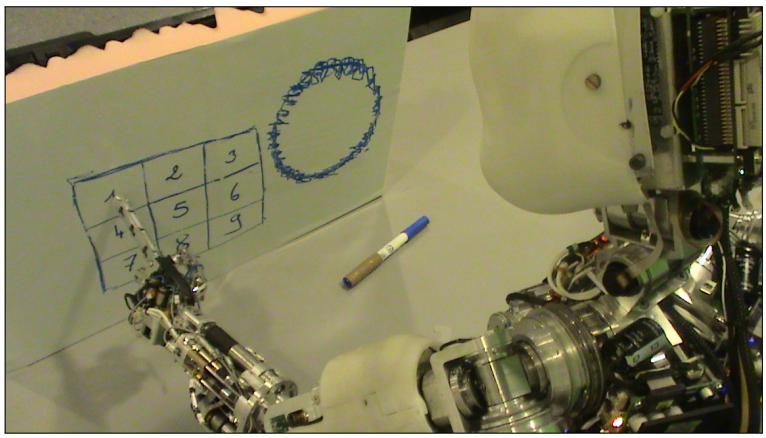


Figure 5.3: The numpad experiment.

Figure 5.4 shows the operational trajectories obtained when controlling the robot with the model learnt with LWPR (after convergence, i.e., several trials).

As one can see, on the real platform, the sequence is well performed with the model learnt in simulation but the obtained trajectories are still far from the straight lines that

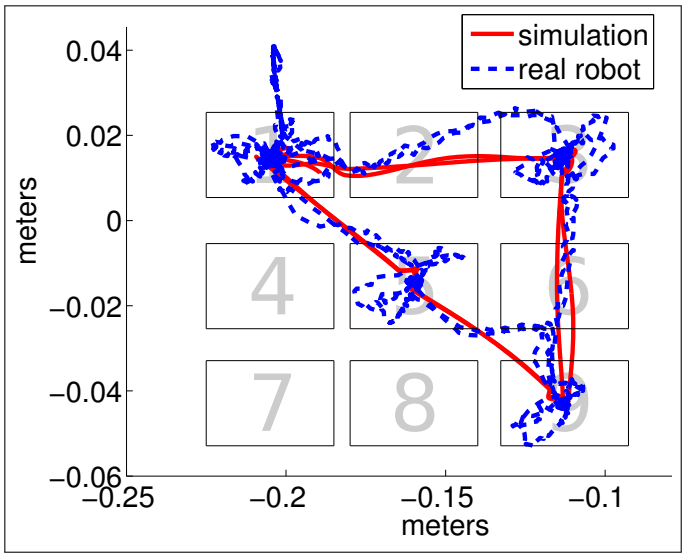


Figure 5.4: Trajectory of the end-effector in the numpad experiment with the model learnt with LWPR in simulation and with iCub.

one would expect from an optimal control as we get in simulation. The global movement is however the same.

In the literature, Vijayakumar et al. [2005] have shown that providing a learning algorithm with real data increases the noise and facilitates the excitation and thus the prediction of the learning process. In our case, learning with joint positions coming from the real robot have not increased the precision of the learnt model. This is probably due to the fact that our approach is dependent of the precision of the KDL model because LWPR learns a model from data originally from that model. The lesson from that experiment is that, even though LWPR succeeds in learning the KDL model accurately, this velocity kinematic model is not sufficient to accurately control the robot. That insufficiency is independent of the learning process that seems to provide an accurate prediction and can be imputed to the fact that the robot is controlled in velocity, with local joint $\mathcal{PID}$ controller and without any dynamic model.

5.3.2 Drawing a circle with iCub

In this experiment, the robot holds a pen in its hand and draw a circle on a white board that is in front of it. Technically, this consists in tracking a point $\boldsymbol{\xi}^{\dagger}(t)$ moving along a circle in the plane of the board.

The quality of the tracking depends on the proportional operational gain K_p which can be considered as the stiffness of a spring between the end-effector and the tracked point. The quality of the tracking also depends on the joint level velocity controllers as

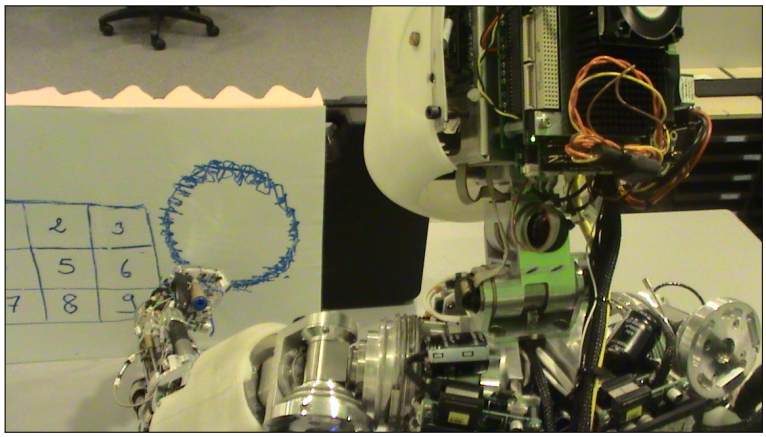


Figure 5.5: The circle drawing experiment.

well as on the velocity of the tracked point. We use the default tuning of the iCub joint level controllers, we choose K_p as $3.0s^{-1}$ and we choose a velocity of $2.0rad.s^{-1}$ in the xz plane to track the moving reference point.

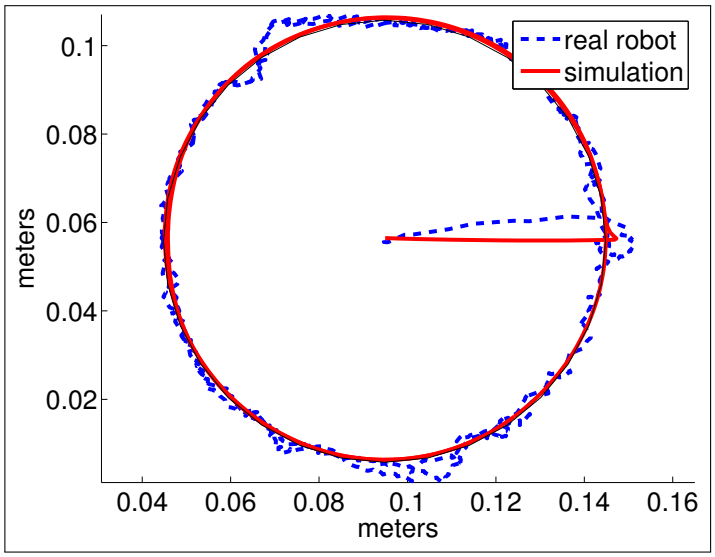


Figure 5.6: Trajectory of the end-effector in the circle drawing experiment in simulation (red plain line) and on the iCub robot (blue dashed line) in the (x_0, z_0) plane.

Figure 5.6 shows the trajectory of the tip of the pen in the (x_0, z_0) plane, i.e., at the surface of the board. Both results in simulation and with the real iCub robot are shown. As one can see, in simulation we get a nearly perfect circle whereas the circle obtained with the robot is noisy.

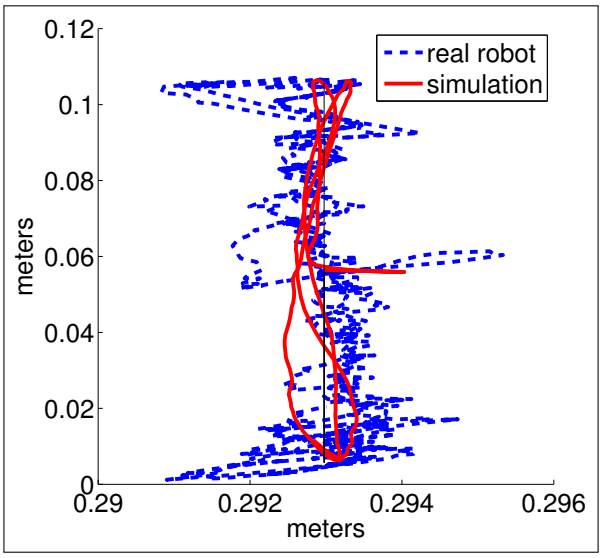


Figure 5.7: Trajectory of the end-effector in the circle drawing experiment in simulation (a) and on the iCub robot (b) in the (x_0, z_0) plane.

Figure 5.7 shows the same trajectories in the sagittal plane, i.e., the plane orthogonal to the board around the z axis. While in simulation the depth is still perfectly controlled, in the case of the robot the contact between the pen and the board is also noisy. However, in Figure 5.7, the scale of the error is in centimetres in the (x_0, z_0) plane whereas it is in millimetres in the (x_0, z_0) plane. Thus the error is quite reasonable.

Our analysis of this experiment is as follows. The fact that the circle is perfectly reproduced in simulation indicates that the learnt velocity kinematic model is accurate. A further inquiry in the results obtained on the iCub robot based both on the KDL model and the learnt model reveals that there is some discrepancy between the desired velocity that is transmitted to the actuators and the observed velocity actually measured on the robot.

Regarding the control of depth, the real robots experiments induce a contact of the effector with the board which effects are only accommodated by the on-line learning process. Adding a contact or a force sensor would greatly improve the tracking performance.

5.4 Discussion

The first experiments on iCub after experiments with its simulator have revealed that the transfer from simulation to the robot is not immediate. There are some differences in the structure and the tuning of the joint level controllers between the simulator and the real robot.

We can conclude that LWPR is able to learn a model that is close to the KDL model provided with the ICUB robot. The accuracy of the learnt model after enough learning experiments gets similar to the accuracy of the KDL model. However, using a velocity kinematic model learnt from that KDL model does not result in accurate control of the robot.

To go beyond this limitation, it would be necessary to use sensors such as vision to determine the actual kinematics and velocity kinematics of the robot from data measured during experiments. Closing the control loop with such external sensors and learning the corresponding models relating the sensory space to the actuation layer is an important matter for future work. Also, closing the control loop at the task level using exteroceptive sensor information results in the possibility to compensate for model uncertainties. It should improve both the quality of the learnt model and the resulting control. We highlight those points further in Chapter 7 of this thesis.

Finally, despite the promises of the model learning approach, the accuracy of the low level controllers remains a key requirement for accurate control, as well as an access to motor current intensity and, potentially, to torque based control. To stress that point further, as presented in Section 5.3.2, given the nature of the errors that we observe in the experiments, a good model of the velocity kinematics used for model-based velocity control is not enough to accurately control a robot unless the low-level actuation layer is extremely stiff. In the case of fixed based robots with a limited number of degrees of freedom, this can be achieved using very powerful actuators coupled with an inverse dynamic model or even stiff joint level $\mathcal{PID}$ controllers. This combination endows the robot with the capability to reject most of the perturbations due to gravity or dynamics coupling effects. This is not the case of most humanoid robots and particularly of the developmental robot ICUB. The weight of the overall structure has to be limited for obvious reasons and the available space for actuators is also limited. This induces the use of less powerful actuators and low level $\mathcal{PID}$ controllers cannot be made stiff enough to properly reject perturbations due to gravity and dynamics coupling effects and explains the strong limitations observed in the control performances obtained with ICUB.

Accurate decentralized control requires stiff controllers, high gains and usually considers interaction with the environment as perturbations that must be rejected. In the context of service robotics, it is necessary to interact with the environment and particularly with human users (lifting objects, etc.). In that context, the control must be sensitive to perturbation rather than attempting to reject them. The system must be compliant, relying on low gains, and thus must be based on the dynamics of the robot. For the moment, our implementation in ICUB only focuses on learning kinematics and thus cannot modify the already implemented low-level controllers. In the next chapter, we present preliminary experiments towards learning the dynamics of a mechanical system as we consider this domain as of crucial strategic relevance for the future of service robotics.

6 Dynamic simulations

Contents

We have seen in previous chapters the limitations that arise when one does not control the dynamics of a robot. In the case of service Robotics, a lot of tasks will imply some interactions with the environment where the resulting contact forces should not be seen as perturbations and rejected. Thus, the goal of this chapter is to explain how our control framework, based on learnt models, can be extended to the case of torque control where the model of the dynamics is learnt too. Moreover, we study the case where forward kinematics and inverse dynamics are learnt and combined to control a robot. The experiment presented here are just a proof-of-concept performed in simulation. Learning these two models separately is achieved in parallel within two distinct processes. In practice, to learn our models with LWPR, we make a bootstrap stage which consists in learning imprecise forward velocity kinematic and inverse dynamic models with random data.

In the first section, we describe how to learn an inverse dynamic model on the whole joint position, velocity and acceleration spaces of a robot. The method is illustrated on a simulated three degrees of freedom planar arm. The second section describes how perturbations, such as an applied distal force, can be compensated for with our control scheme. We also detail how, when learning a dynamic model of a robot subjected to a perturbation, this may induce some undesired effects named after-effects.

6.1 Learning inverse dynamics

In this section, we first describe how to learn the inverse dynamics of a three degrees of freedom planar arm. We describe how to tune parameters and how our random bootstrap stage with the LWPR algorithm is made. We also describe how to learn dynamics with XCSF, highlighting the fact that the learnt model can be quickly used in a control loop.

The three degrees of freedom planar robot used is the one used in Chapter 4 and described in Figure 4.1, page 69.

6.1.1 Learning inverse dynamics with LWPR

We use LWPR with parameters initialized in the same way as described in Section 4.2.1. As the input are the joint positions, velocities and accelerations, the $norm_{in}$ parameter is set to the range of motion of each joint, its velocity range and its acceleration range. First simulations with analytical models have allowed us to determine approximately the range of each input. It is not necessary but recommended to facilitate the learning algorithm. In the case of real robots, the normalization can be made using the technical characteristics of the system.

Joint positions are still limited between 0 and 2π rad. Joint velocities are signed as they rotate positively or negatively around an axis. They are normalized by 10 which is approximatively the biggest attained value. Joint accelerations are normalized by 200. Even if it is an approximation, we have normalized identically each joint to simplify our learning procedure. Having a specific normalization for each joint could yield better results. However, we recall that this normalization is just here to provide normalized input to the learning algorithm, i.e., input in the same interval $[0, 1]$. It is not critical if the input appears to be upper than the normalization factor. For instance, if a joint velocity appears to be 11 rad/sec, it will be divided by 10 and the conditioning of the input will be worse as the resulting value 1.1 will not be in the $[0, 1.1]$ interval. However, the normalization frontier is not strict and the learning algorithm will still be able to approximate it.

To summarise, all parameters are given in Table 6.1.

Parameters	$norm_{in}$	$init_D$	$update_D$	$init_\alpha$	w_{gen}	γ
Values	$[2\pi, 10, 200]$	0.1	1	10000	0.5	$1e^{-6}$

Table 6.1: Parameters used to learn an inverse dynamic model with LWPR

We change the $init_D$ parameter because learning dynamics is different from learning kinematics. Lowering this parameter increases the initial size of the receptive fields. It results in a rougher and simpler model. We changed this parameter because, if it is too high, the number of receptive fields increases quickly over several thousands and learning gets too slow.

We first initialize a model providing the learning algorithm with random data chosen between the estimated ranges: $\boldsymbol{q}_{limit} = [0, 2\pi]$, $\dot{\boldsymbol{q}}_{limit} = [-10, 10]$, $\ddot{\boldsymbol{q}}_{limit} = [-200, 200]$. We provide as second output the torques computed from the analytical dynamics as $\boldsymbol{\tau}^{con} = \mathcal{ID}\left(\boldsymbol{q}\left(t\right), \dot{\boldsymbol{q}}\left(t\right), \ddot{\boldsymbol{q}}\left(t + dt\right)\right)$. In real world applications, this torque would have to be obtained either from torque sensors at the joints level (usually strain gauges) or from electrical current measurements on the actuators.

The learning process is thus implemented as

$$\mathcal{ID}_{\text{LWPR}} = \text{LWPR}_{learn}\left(\left[\boldsymbol{q}\left(t\right), \dot{\boldsymbol{q}}\left(t\right), \ddot{\boldsymbol{q}}\left(t+dt\right)\right], \left[\boldsymbol{\tau}^{con}\left(t\right)\right]\right). \tag{6.1}$$

Initializing an inverse dynamics model with LWPR

The results of the bootstrapping stage are shown in Figure 6.1, where we illustrate the evolution of the inverse dynamics model approximation while learning with random data chosen among all possible random states.

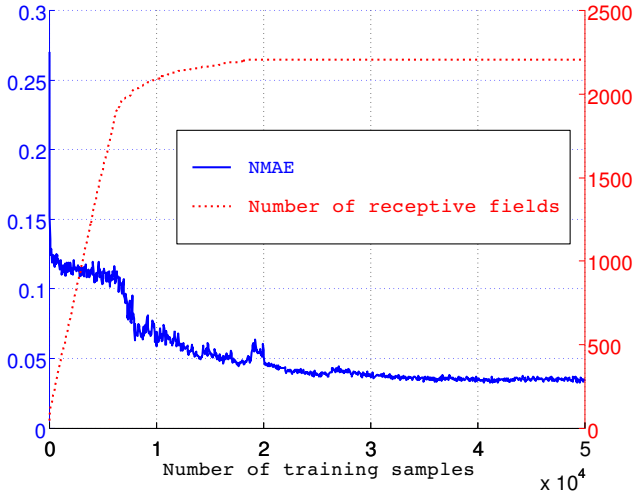


Figure 6.1: Evolution of the NMAE of the predicted torque for the first axis and the number of receptive fields in function of the sampled data.

Every 50 samples, to evaluate the effectiveness of the prediction, we use the Normalized Mean Absolute Error (NMAE) computed as

$$\text{NMAE} = \frac{1}{n_p \|\hat{\tau}^{con}_{max}\|} \sum_{i=1}^{n_p} \|\hat{\tau}^{con} - \tau^{con}\|$$

where $\hat{\tau}^{con}_{max}$ is the maximum predicted torque and $n_p = 1000$ is the number of points used to compute the error.

Given the selected parameters, the approximated model is stabilized both in terms of prediction error and in term of number of receptive fields after 3.10^4 training samples.

One can see that the normalized error is stabilized approximatively to 0.4. This is explained by our parameter set-up that prevents the number of receptive fields to grow above 2300. Our choice is to obtain a smaller approximated model at the expense of precision. This choice should facilitate the scaling of our framework to higher dimension

systems since we could, if necessary, increase the number of receptive fields while keeping the same precision.

Bootstrapping the model along a desired trajectory

Preliminary experiments (not described here) show that the initialization described above is enough to bootstrap a successful learning of the inverse dynamics of a two degrees of freedom planar robot, but it is not enough to bootstrap the process for a three degrees of freedom robot. As a consequence, to get the results described below, we further train the model performing the required trajectory using an analytical model of the dynamics.

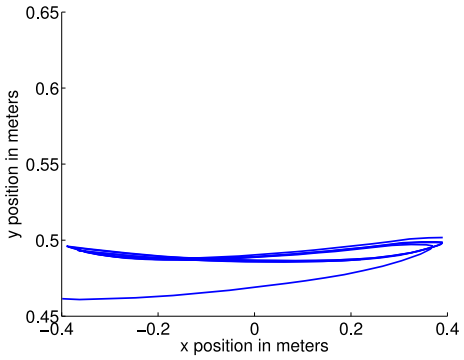


Figure 6.2: Controlling the robot along a straight line to refine the model.

As described in Figure 6.2, the robot performs a quasi-straight line, the small error being due to the fact that the tasks are defined as goals to reach and not as trajectories to follow. At the beginning of each reaching task movement, the error between the current position and the goal is equal to the distance between the previous goal and the current one. The error is multiplied at each time step by the Jacobian matrix of the current state to obtain joint velocities. Since the Jacobian matrix is the first order approximation of the system around a configuration, it may induce a deviation. As proof, taking a smaller time step decreases the deviation. As a consequence, increasing the proportional operational coefficient K_p increases the operational velocity error and thus increases the deviation too.

During this refining period, the $init_D$ parameter has been set to 5. Thus, the learnt model has been refined along the trajectory with smaller receptive fields. After few forward and backward movements, it becomes possible to control the system as accurately as with the analytical model.

Learning inverse dynamics realising two compatible tasks

To highlight how several tasks can be learnt and combined within our framework, we simulate our system with two hierarchical goals $\boldsymbol{\xi}_1^\dagger$ and $\boldsymbol{\xi}_2^\dagger$. $\boldsymbol{\xi}_1^\dagger$ is fixed to $0.00m$ on the x_0 axis and to $0.50m$ on the y_0 axis. $\boldsymbol{\xi}_2^\dagger$ is alternatively set to $-0.15m$ and $0.15m$ along the x_0 axis. The switch between goals occurs when the task is achieved with a position error arbitrarily chosen to be less than or equal to $0.001m$. K_{p1} and K_{p2} are chosen to be respectively $40s^{-1}$ and $0.5s^{-1}$.

To address this combination of tasks, we use the control scheme described in Chapter 3 where kinematic and dynamic models are both learnt with LWPR. We first learn the kinematics using the parametrization and the process described in Chapter 4. The forward velocity kinematic models of both tasks are learnt on the whole joint space: $\forall i, 0 \leq q_i \leq 2\pi$. After validating our kinematic model in simulation, realising both hierarchical tasks, we then learn the inverse dynamics. To do this, we use the same process described in the previous section, learning first a global rough model refined along the desired trajectory. Learning is still active during evaluation, resulting in further refinements of the model around the trajectory. The simulation is described in Algorithm 5 page 61.

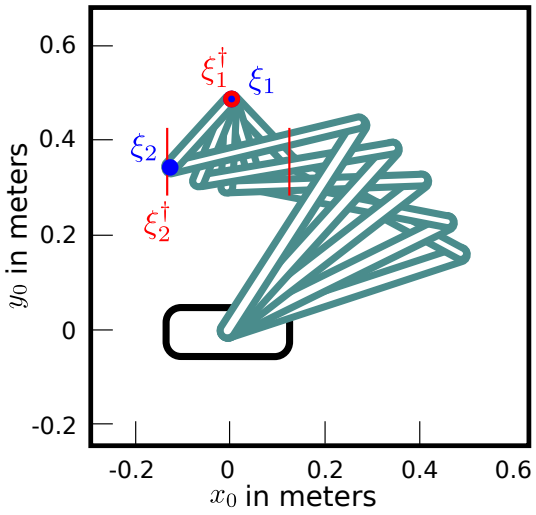


Figure 6.3: Stroboscopic view of the robot which end-effector $\boldsymbol{\xi}_1$ is fixed and whose second task is to go from $x_0 = 0.15m$ to $x_0 = -0.15m$ with the second effector $\boldsymbol{\xi}_2$.

Figure 6.3 shows the evolution of the configuration of the robot realising tasks $\boldsymbol{\xi}_1^\dagger$ and $\boldsymbol{\xi}_2^\dagger$ which are compatible and both fully achieved.

Figure 6.4 shows the distance between the end-effector and its target with three conditions: with analytical kinematics and dynamics, with an analytical kinematics and a learnt inverse dynamics, and in the case where both models are learnt. One can see that

there is no significant difference between the results obtained when only learning the inverse dynamics model or when learning both kinematics and dynamics models. The errors, inferior to $0.015m$, are due to the limited precision of the dynamics model and not due to the kinematic one. Each peak of maximum error corresponds to the times where the goal is changing.

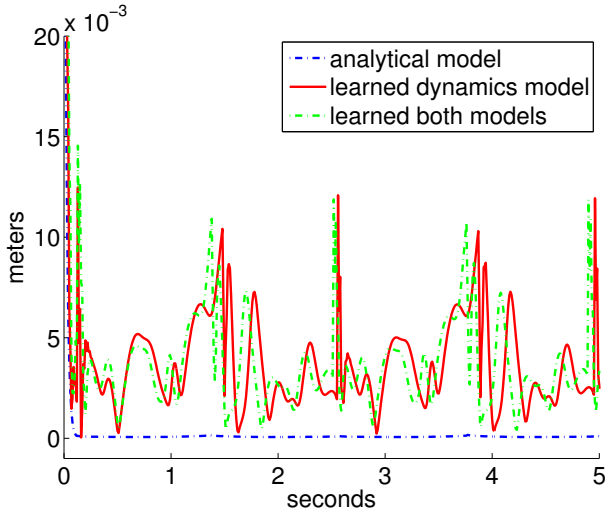


Figure 6.4: Error on the position of the end-effector of the robot realising two compatible tasks.

6.1.2 Learning inverse dynamics with XCSF

We use XCSF to learn the inverse dynamics of the same simulated three degrees of freedom planar system with the method described in Section 3.2.2. In this section, we do not learn any kinematic model.

We provide as condition parameters a state $[\boldsymbol{q}, \dot{\boldsymbol{q}}]$ normalized by its maximum values chosen identically to the input normalization of LWPR. We provide as prediction output the controlled torques $\boldsymbol{\tau}^{con}$ computed with the analytical dynamic model and as prediction input the resulting acceleration at the next time step

$$\mathcal{ID}_{\text{XCSF}} = \text{XCSF}_{learn}\left(\left[\boldsymbol{q}\left(t\right), \dot{\boldsymbol{q}}\left(t\right)\right], \left[\ddot{\boldsymbol{q}}\left(t+dt\right)\right], \left[\boldsymbol{\tau}^{con}\left(t\right)\right]\right). \tag{6.2}$$

The parameters have been tuned, for a majority, as for the kinematic experiments presented in Chapter 4. The only modifications are :

- the maximum number of micro-classifiers is set to 1000,
- the minimum size of the classifiers at creation is set to 0.250 in each dimension instead of 0.005 by default,

- there is no compaction,
- the *closest classifier matching* is always activated.

All the other parameters are set to their default value.

We have tried to perform a compaction stage but it only resulted in the deletion of useful models without generalization. The model was just forgetting previous spanned regions, thus compaction was disruptive.

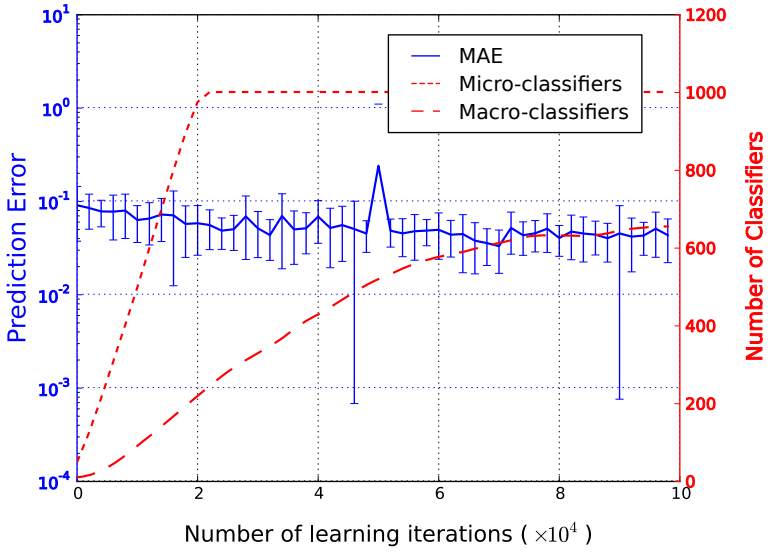


Figure 6.5: Evolution of the MAE of the predicted torque and the number of classifiers in function of the sampled data using an inverse dynamic model learnt with XCSF. The error is computed on the first joint; errors of the other axis are similar.

Figure 6.5 illustrates the MAE of the predicted torques. This error is computed while performing random reaching as described in Section 4.4. The number of micro-classifiers increases until it stabilizes at its maximum value. Since there is no compaction, the number of macro-classifiers also increases until it stabilizes.

Oppositely to LWPR, we used XCSF with only one population of classifiers to predict every output. The number of classifiers represented in Figure 6.5 is thus the same for each axis. The learning process is thus based on the average error of each dimension.

Learning inverse dynamics with XCSF performing an under-constrained task

This section detail how an inverse dynamic model learnt with XCSF can be used in the control law of Figure 3.4 without learning the kinematics. It illustrates the evolution of the trajectories while performing this point to point movement. The targets are still positioned at $[0.20\ 0.50]^T\ m$ and $[0.10\ 1.00]^T\ m$ and are reached with a precision of 0.01 meter. K_p equals $5s^{-1}$.

Torques are computed with the learnt model as

$$\boldsymbol{\tau}^{con} = \text{XCSF}_{predict}\left(\mathcal{ID}_{\text{XCSF}}, [\boldsymbol{q}, \dot{\boldsymbol{q}}], [\ddot{\boldsymbol{q}}]\right). \tag{6.3}$$

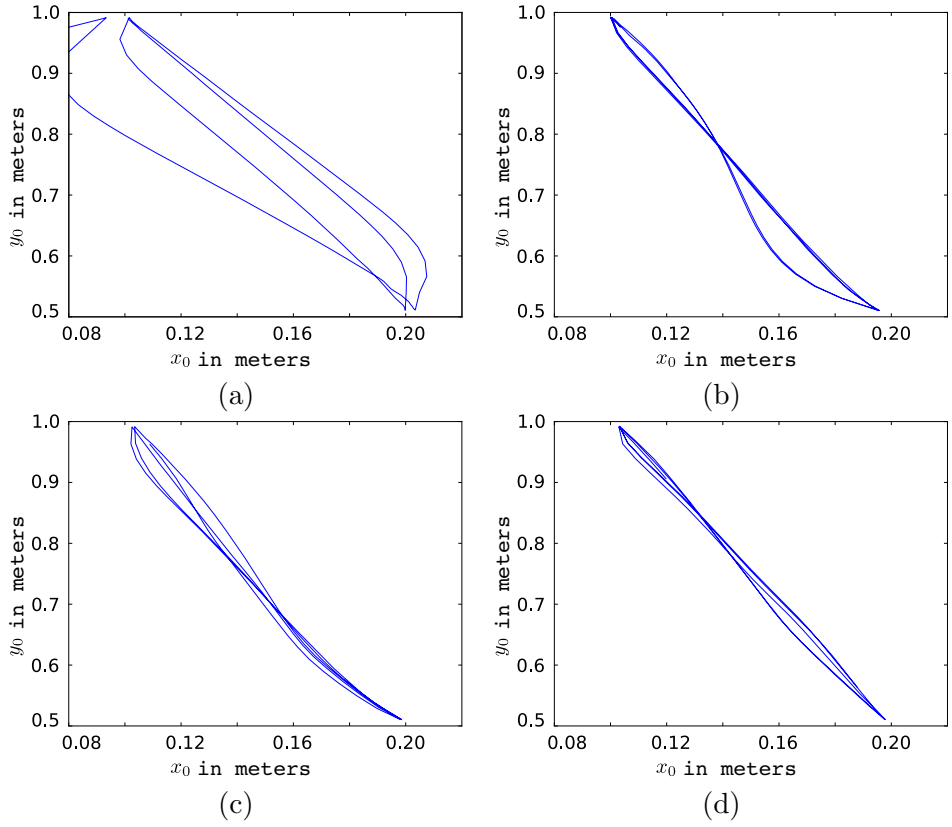


Figure 6.6: Evolution of operational trajectories while learning the inverse dynamics realising an under-constrained task. Each figure represents 4 seconds of simulation.(a): 0s to 4s.(b): 24s to 28s.(c): 52s to 56s.(d): 76s to 80s.

The performed trajectory evolves until reaching a quasi-straight line.

6.2 Dealing with perturbations

The goal of this section is to demonstrate the capability of our control scheme to a learnt inverse dynamic model as described in the previous section. We first show how the system can adapt to unknown perturbation with LWPR. Then, we describe how learnt dynamic models are subject to after-effects when using a model that has been learnt in the perturbed case when the perturbation is removed. The simulated robot used is a two degrees of freedom robot composed of the first two segments of the three

degrees of freedom robot previously used.

6.2.1 Adapting to perturbations with LWPR

In order to show that a system controlled by our framework can adapt to external perturbations, we define a unique task $\xi_1^\dagger$ which consists in reaching alternatively two goals along the x_0 axis while applying a constant $30N$ vertical force $\boldsymbol{f}$ at the distal point of the robot that strongly affects the dynamics of the system. In order to simulate the application of such a force in ARBORIS, a disturbance torque $\boldsymbol{\tau}^{perturb} = J^T \boldsymbol{f}$ is applied at the joint level. The goal changes alternatively from $x = -0.40m$ to $x = 0.40m$. The switch between goals occurs when the task is achieved with an arbitrary precision error equal to $0.05m$.

There are two ways to compensate for the perturbation $\boldsymbol{f}$. The first one consists in learning directly the perturbation from a force sensor positioned where the force is applied, position which is unlikely to be known a priori. Even if research in the design of synthetic skins for robots [Someya et al., 2004; Cannata et al., 2008; Lamy et al., 2009] has recently made great progress in measuring force in any part of the robot, it is still unprecise and difficult to use in practice. With common robots, we cannot access easily this perturbation force. We only access the state of the system and the controls applied to it. This means that we have access, during a time step, to the torques from a controller while the perturbation is not yet perceived. We can have a model of the perturbation only through the state of the system after the torques $\boldsymbol{\tau}^{con}$ have been applied.

The inverse dynamics learning is thus learnt as before

$$\mathcal{ID}^{perturb}_{\text{LWPR}} = \text{LWPR}_{learn}\left(\left[\boldsymbol{q}\left(t\right), \dot{\boldsymbol{q}}\left(t\right), \ddot{\boldsymbol{q}}\left(t+dt\right)\right], \left[\boldsymbol{\tau}^{con}\left(t\right)\right]\right).$$

The difference between this experiment and the previous one without perturbations, is that the acceleration computed at the next time step now depends on the applied force. The learnt inverse dynamics is thus

$$\boldsymbol{\tau}^{con} = A\left(\boldsymbol{q}\left(t\right)\right)\ddot{\boldsymbol{q}}\left(t+dt\right) + \boldsymbol{b}\left(\boldsymbol{q}\left(t\right), \dot{\boldsymbol{q}}\left(t\right)\right) + \boldsymbol{g}\left(\boldsymbol{q}\left(t\right)\right) + \boldsymbol{\epsilon}\left(\boldsymbol{q}\left(t\right), \dot{\boldsymbol{q}}\left(t\right)\right) - \boldsymbol{\tau}^{perturb} \quad (6.4)$$

The $\boldsymbol{\tau}^{con}$ torques computed with the learnt inverse dynamics now depends on the constant unknown applied force

$$\boldsymbol{\tau}^{con} = \text{LWPR}_{predict}\left(\mathcal{ID}^{perturb}_{\text{LWPR}}, \left[\boldsymbol{q}\left(t\right), \dot{\boldsymbol{q}}\left(t\right), \ddot{\boldsymbol{q}}^{\star}\right]\right)$$

To obtain a better adaptation to the perturbation, we have change the parameter w_{prune} to 0.9. This parameter allows to delete obsolete receptive fields and makes the learning process more sensitive to newer input.

The control approach is summarised by Algorithm 6.

Algorithm 6 Simulation loop with unknown force applied.

$\boldsymbol{\nu}_1^\star(t) = K_{p1}\left(\boldsymbol{\xi}_1^\dagger(t) - \boldsymbol{\xi}_1(t)\right)$ *desired operational velocity*

$\dot{\boldsymbol{q}}^\star(t) = \hat{J}_1^+ \boldsymbol{\nu}_1^\star(t)$ *desired joint velocity*

$\ddot{\boldsymbol{q}}^\star(t) = \left(\dot{\boldsymbol{q}}^\star(t) - \dot{\boldsymbol{q}}(t)\right)/dt$ *desired joint acceleration*

$\boldsymbol{\tau}^{con} = \hat{\mathcal{ID}}\left(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}^\star(t)\right)$ *desired torques*

$\left(\boldsymbol{q}(t+dt), \dot{\boldsymbol{q}}(t+dt), \boldsymbol{\xi}(t+dt)\right) \leftarrow \mathcal{D}\left(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \boldsymbol{\tau}^{con} + \boldsymbol{\tau}^{perturb}\right)$ *integration*

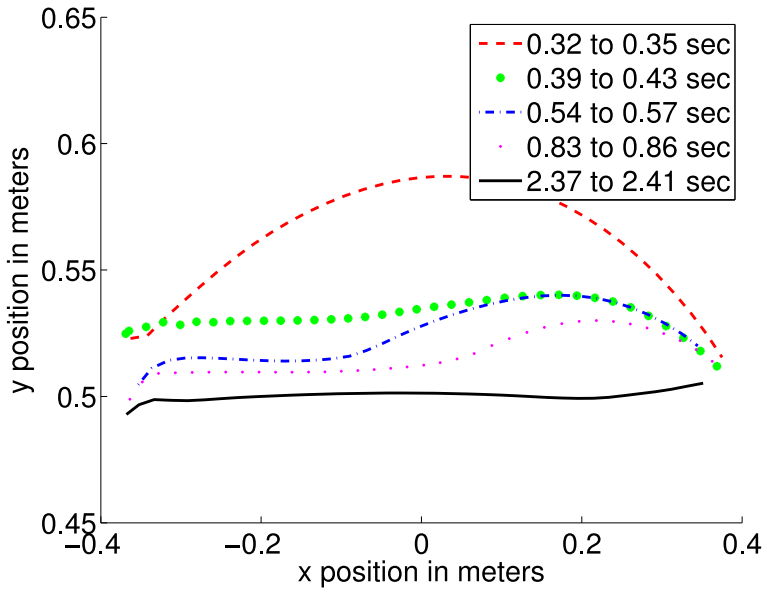


Figure 6.7: Evolution of the operational position while applying an unpredicted force, starting from $x = -0.40m$ and going to $x = 0.40m$ (the way back is not shown). An inverse dynamics learnt with LWPR allow to adapt the trajectory according to the applied force in order to perform a straight line. The applied force is $f = 30$ Newton and the task space proportional controller $K_p = 2s^{-1}$. The control law is the same as the one described in Figure 3.4.

Figure 6.7 represents the vertical deviation of the end-effector induced by the application of the $30N$ vertical force. The deviation is visible since the system is reaching a goal and not tracking a trajectory. When the learning algorithm is off, the deviation is not compensated for. However, when learning is active, the dynamics model is incrementally updated and, after a short time, the learnt model is adapted to the new encountered force and compensates for it. The five different curves in Figure 6.7 illustrate this adaptation along time and this experience demonstrates that any unmodelled force (such as external perturbations) can be compensated for just by adapting the dynamics model.

However, adaptation induces after-effects when the perturbation is removed, as described below.

6.2.2 After-effects

As for every inverse dynamic model including the compensation for a force, if that perturbation is removed, the model-based controller will continue to compensate for that force thus the trajectory will be deviated. This effect, present in human control, is called after-effect and has been illustrated in [Shadmehr and Mussa-Ivaldi, 1994].

As our control scheme is based on an adaptive inverse dynamic model, the deviation is present until the model re-adapts to the new unperturbed dynamics.

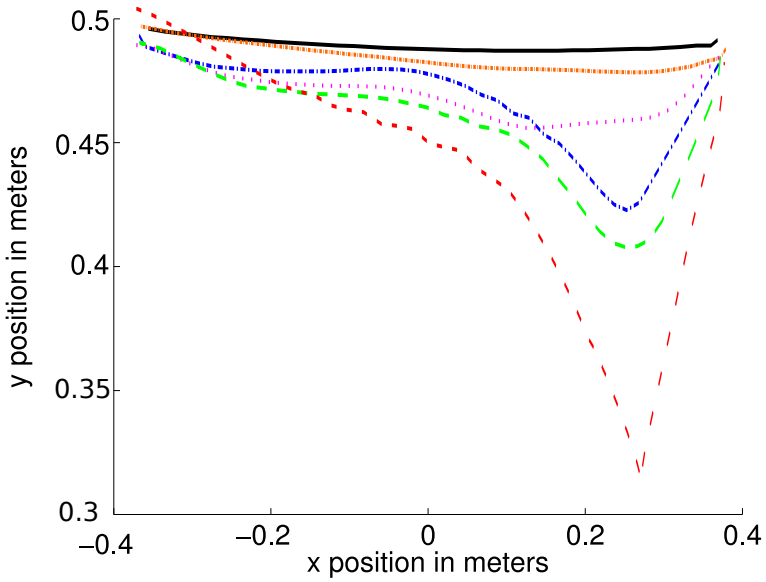


Figure 6.8: Evolution of the after-effect using a learnt model based on a dynamic perturb by a force. From the left goal to the right one.

Figure 6.8 illustrates the evolution of the trajectory using the inverse dynamics learnt during the perturbation stage. We can see that the trajectory is very perturbed at the beginning of the experiment. Time after time, the model re-adapts to the new unperturbed dynamics. The end-effector takes an average of $0.4s$ to go from a point to another and it takes approximately 20 seconds to get back to a straight trajectory.

First results on XCSF, not presented here, let show a possibility of adaptation quite similar to LWPR. However, XCSF requires more time to perform an adaptation in comparison to LWPR. The fact that LWPR owns a forgetting parameter clearly plays an important role.

6.3 Discussion

The experiments presented in this chapter are designed to show that, in order to make profit of the redundancy of a mechanical system in the context of adaptation to perturbations or when realizing a combination of tasks, it is necessary to separately learn a direct velocity kinematic model and a dynamics model, either direct or inverse. As discussed in Section 2.2.3, the alternative approach that consists in directly learning the task space velocity to torques mapping is appealing because the learnt model is smaller, but with such an approach, one model for each task is necessary. Thus, even if learning separately an inverse dynamics and a kinematic model is more difficult than learning only an inverse mapping relating operational velocities to torques, we have shown in this chapter that state-of-the-art statistical learning methods such as LWPR or XCSF are efficient enough to solve this more difficult problem on moderately complex systems.

Moreover, the proposed work also demonstrates the advantages of separately learning models at the kinematic and dynamic levels instead of directly learning an inverse mapping from task space velocities to joint torques.

- The first advantage of learning separated models lies in the fact that adaptation to unpredicted force perturbation does not require a new learning process of both models but only requires the adaptation of the dynamics level.
- Secondly, when a new task has to be learnt (and, as a consequence, a new forward velocity kinematic model), the dynamics model does not need to be adapted as it has already been learnt on the whole space.
- Finally, we have seen that using LWPR to learn a model in task space is smaller than learning it in joint space as the maximum dimension of the task space is 6 whereas n, the joint space input dimension, is usually higher (see Section 2.2.3). However, using XCSF, both model have the same condition input space; the difference resides only in the size of the prediction space that seems to be less important.

One may note that, even if we learn the kinematic model before the dynamics, learning *simultaneously* both models should be possible as they are defined on different spaces, the errors in the velocity kinematic model should not interfere with the errors in the inverse dynamic model and vice-versa. The only reason why it may fail would be due to some interactions between the explorations of both spaces, since errors in one model may drive the system in inadequate regions for learning the second type of model. However, active exploration and testing this possibility of simultaneous learning is a matter of future work that is beyond the scope of the work performed here.

While it seems likely that the approach consisting in learning a unique inverse mapping from task space velocities to joint torques can scale up reasonably well, one may wonder if this is the case with the approach we presented, when facing Robotics systems with numerous degrees of freedom. This question is a strong incentive for future research, either towards more efficient model learning tools as investigated in Nguyen-Tuong et al. [2008] or towards learning and combining several smaller weakly coupled sub-models.

At last, one may compare the complexity of both algorithm. While LWPR uses 2300 receptive fields to learn an inverse dynamic model for each of the three dimensions, and thus 6900 receptive fields, XCSF only uses 1000 classifiers to learn simultaneously the three dimensions of the same model. Without considering the slower adaptation of the XCSF algorithm, it seems more appropriate to learn mechanical models.

7 Conclusion and perspectives

Contents

In this chapter, we first detail our contributions and the limitations of our approach. Then we present some perspectives by introducing the future work that could be developed on our work.

7.1 Contributions

Service robots evolve in interaction with people. Those interactions are rarely well modelled and the encountered situations are often diverse and unforeseen. Moreover, those robots are more and more complex either by their sophisticated sensors or by their numerous degrees of freedom. This thesis tried to develop a framework in order to include adaptive capabilities in the control loop of such robots. As a solution, we have chosen to learn mechanical models of robots to control them.

More precisely, we have used two state-of-the-art function approximation techniques, LWPR and XCSF, to learn the forward kinematics, the forward velocity kinematics and the inverse dynamics of multiple robotic systems. Even if the experiments performed here were corresponding to a simple proof-of-concept, we had applications to unstructured and evolving environments in mind.

7.1.1 Conceptual contributions

At the conceptual level, we have shown how a model learning processes could be combined with state-of-the-art operational space control techniques to control a robot. In particular, we demonstrated that we can benefit from the hierarchical combination capabilities of the operational space control framework to achieve several learnt tasks in parallel even when those tasks are not fully compatible. This was made possible by learning the unique forward mapping for each task and then inverse it instead of directly learning an inverse mapping. More precisely, our contributions are the following.

- We explained how one can obtain the forward velocity kinematics under a matrix form using two different learning schemes. The first one, using LWPR, is based on learning the forward kinematics and using the derivative of the learnt function to get the forward velocity kinematics. The second scheme, using XCSF, learns directly the forward velocity kinematics as a set of overlapping linear models.
- We have highlighted the advantages of our framework with respect to alternatives such as the extended Jacobian matrix approach when one wants to combine several tasks hierarchically. We have shown that, if we want to combine unforeseen tasks, it is necessary to learn the models in the whole reachable space of the robot rather than along some local trajectories.
- We have stressed the interest of separating the kinematic level from the dynamic level. Despite a higher computational learning cost in some cases, this framework seems to be the most adapted to control explicitly the redundancy as we do not need to learn an inverse dynamic model for each end-effector.

7.1.2 Experimental contributions

At the experimental level, we have validated our approach with different systems so as to study different properties:

- a torque controlled 3 degrees of freedom planar arm with a known dynamics, that helps illustrating the phenomena resulting from the combination of tasks, when they are compatible or not,
- a complex velocity-controlled humanoid robot named ICUB, that allowed to evaluate the empirical applicability of our method to state-of-the-art, complex, and mechanically not-so-accurate robots,
- a torque controlled version of a 3 degrees of freedom planar arm where an inverse dynamic model is learnt with LWPR or XCSF,
- a torque controlled version of a 2 degrees of freedom planar arm that helps illustrating the dynamic phenomena resulting from the occurrence of a dynamic perturbation.

In chapter 6, we have illustrated the fact that it was possible to learn a model in a non-stationary environment, adapting an inverse dynamic model to perturbations.

To summarize, we have reached the goal expressed in the introduction that consists in proposing a general framework to endow robots with adaptive capabilities when we want to combine several tasks in diverse areas of the reachable space while adapting to unforeseen perturbations.

Finally, we have gained a lot of experimental insights about the way to use learning algorithms within our control loop, about the way to tune these algorithms and about the current limitations of their capabilities. Indeed, our experiments have revealed difficulties using these learning algorithms in practice as well as some limitations of our framework.

7.2 Limitations

In this section, we describe the work that should be done to conclude more firmly on our work.

7.2.1 Further study of XCSF versus LWPR

We already stated in Chapter 4 that our use of XCSF as learning tool is preliminary, though it seems to be very efficient for learning mechanical models of robots. A lot of experiments would have been required to better evaluate XCSF capabilities with respect to the ones of LWPR.

- We have not compared the performance of XCSF with a random bootstrap equivalent to the one used with LWPR. Though the capability of XCSF to learn without random bootstrap is a nice feature for robotics application and opens the way to experiments with more complex systems, such a comparison would have allowed us to draw firmer conclusions about the relative stability of both algorithms, specially using real robots such as the iCub humanoid robot.
- As stated in Section 6.2.2, we have performed the adaptation to perturbation and after-effect experiments with LWPR. Preliminary trials with the XCSF algorithm studying those tasks seem to reveal that, XCSF being slower, its adaptation capabilities are weaker than the ones of LWPR. One obvious and immediate perspective for the work presented in this thesis consists in pushing further the comparison between XCSF and LWPR, reproducing the experiments performed so far with LWPR, using XCSF instead.
- Most importantly, we have not compared seriously the approach consisting in learning the kinematic model and then deriving it to get a matrix from Jacobian and learning the velocity kinematic model directly. Even though the latter seems more natural, a problem may appear when learning such model on a real robot from external sensors. Indeed, when we learn the kinematics, the learning error depends on operational positions that are easier to get from sensors such as vision. Using velocity kinematic data, the learnt model depends on operational velocities which can only be reconstructed from the operational positions using the derivative of

a position sensor signal. Thus, it induces a dependency to the time step into the learning process and increases the complexity.

7.2.2 Scalability issues

With respect to the long standing challenges expressed in the introduction of this thesis, we are still far from proposing the versatile learning algorithm that would learn from scratch the whole body model of a complex humanoid robot to achieve any combination of tasks in interaction with an unmodelled environment. But the limitations result more from computational limitations of the algorithms than from fundamental flaws in the framework.

First, LWPR is limited by the necessity of performing a bootstrap stage using random data. This is quite simple to do when working in simulation. However, preliminary experiments on a real humanoid robot (see Section 5) have convinced us that random bootstrapping is a complex task both in terms of safe implementation and of intrinsic complexity when dealing with systems such as humanoid robots. Thus, a balance has to be found between a wide exploration of the state space and a realistic implementation on real robots. Developing active exploration capabilities (e.g. Robbel [2005]) may be a way to provide this balance. Indeed, active exploration is probably a more general way to reduce the learning complexity and consequently to facilitate the scaling of our approach to higher dimensions.

We have seen that machine learning models are, for the moment, not as precise as analytical models. They usually use probabilistic algorithms to allow incremental learning which may induce errors in the model. However, the adaptability of machine learning to unforeseen environments or interactive robots make them promising tools for future works. Indeed, learnt models are conceptually closer to real robots than fixed analytical models as they can adapt to modification of the modelled system.

Nevertheless, as for analytical or learnt models, the source of the data is still problematic. While we can easily imagine learning algorithms with computationally great capabilities such as learning the model of a complete human like robotic system, the precision of the sensors is still a limitation that seems conceptually difficult to overcome.

Finally, given the preliminary nature of our empirical studies with XCSF, we cannot conclude precisely on how many degrees of freedom such a method can deal with when learning the kinematics and the inverse dynamic model of a robot. However, preliminary investigations tend to indicate that going beyond 10 degrees of freedom, within a reasonable amount of time, is still out of reach with the current methods.

To go further, one can first think about combining efficient mechanism from LWPR and XCSF to design an even more efficient algorithm in terms of input/output dimensions and/or in terms of precision. For example, trying to split each output with its own population may also split the complexity of the learnt model. In the same line, one may think of using the PLS within XCSF to keep only the convenient information. In the discussion of Chapter 6, we also mentioned the work of [Nguyen-Tuong et al., 2008] in

that direction. Second, a different approach to even more complex system mentioned in Chapter 4 consists in learning local models of sub-part of systems and finding a way to represent how these smaller models are coupled together. This approach is clearly an important matter of future work.

7.2.3 Vision for measuring operational positions

As noted in Chapter 5, one of the main limitations of our results so far, when trying to learn a model on a real robot, is that we do not have access to operational space parameters by any other way than using a given joint space to operational space mapping, which is exactly what we are trying to learn.

An obvious way to overcome this limitation consists in using external sensors such as cameras to estimate the operational space parameters directly [Natale et al., 2007]. In the case of iCub, the robot has been designed as a developmental robot with natural human-robot interactions in mind. Consequently, it possesses two cameras representing its eyes. Like in human vision, it is possible to reconstruct a Cartesian position of an end-effector using triangulation methods.

However, this approach raises several issues

- The estimated position of the end-effector can be melded with information coming from exteroceptive sensors such as vision in order to be more precise. Data from exteroceptive or proprioceptive models depend on sensors. Joint sensors of the robot provide accurate estimates of the joint position, velocity or accelerations. By contrast, operational space positions would be estimated from noisy sensors with delays induced by image processing.
- Taking into account the estimated nature of the operational state information provided by sensors implies that we call upon specific state estimation tools such as Kalman filters.

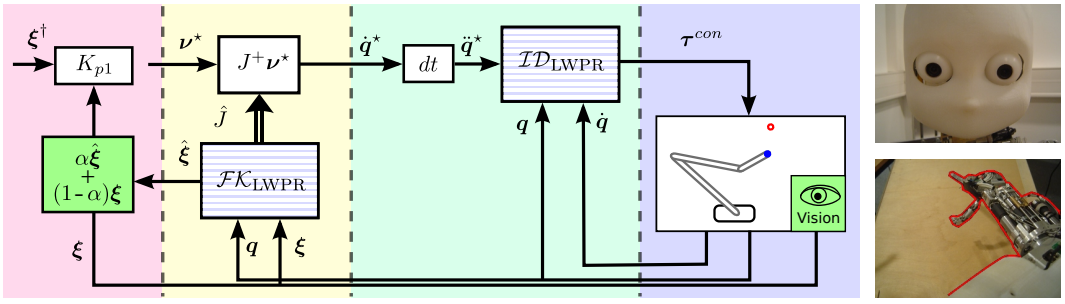


Figure 7.1: Locate end-effectors using binocular vision, geometric detection and multi-sensors state estimation.

Figure 7.1 illustrates a possible short-term perspective control loop using the cameras of iCub to learn a vision-based kinematic model.

7.3 Perspectives

The previous section was mostly dedicated to work that should have been performed to draw conclusion from our work. In this section, we rather present new research lines that should result from the contributions of this thesis.

7.3.1 Trajectory optimization

We have focused on redundancy in kinematics but not in trajectory generation and planning. As we explained in Chapter 1, the RMRC framework is a reactive method that drives the system toward its goal in a linear way. However, as described in Figure 7.2, there is an infinity of ways to go from a position $\boldsymbol{\xi}$ to a goal $\boldsymbol{\xi}^\dagger$, even if a via point $\boldsymbol{\xi}^\star$ is defined, and different trajectories may correspond to a different cost.

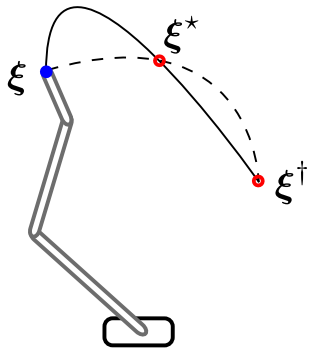


Figure 7.2: Representation of two possible ways to go from an operational position $\boldsymbol{\xi}$ to a goal $\boldsymbol{\xi}^\dagger$ passing through the via point $\boldsymbol{\xi}^\star$. Figure inspired by [Kawato, 1993].

As stated in the introduction, defining a strategy to reach a goal is a topic in itself. When one wishes to plan a path in a cluttered space, this is usually accomplished with configuration space planning methods [Latombe, 1991; Laumond, 1998]. It consists in building a path in such a way that it is guaranteed that the robot can perform the trajectory without meeting any obstacle. But configuration space planning requires that the environment itself is known perfectly before planning can take place, and the methods are very sensitive to uncertainties in the knowledge of the environment. Thus, these methods can hardly be used in the broad context of service Robotics. Learning a model of the environment to choose an optimal solution, an optimal goal and/or an optimal strategy seems to be a promising approach.

A lot of optimal control and or optimization methods can be used here. One of the most promising approch would consists in using the Sequential Quadratic Programming methods using learnt models to generate an optimal trajectory. Besides, of particular interest in the context of our research is the possibility to learn optimal trajectories using a reinforcement learning approach. A general review of such methods is in [Peters

et al., 2009] and a new promising recent algorithm, PoWER, is developed in [Kober and Peters, 2008].

7.3.2 Active learning and artificial curiosity

Learning a model of the whole environment and finding an optimal trajectory in the corresponding space is extremely costly. If one wants to get a not-too-bad behaviour quickly, one needs to drive the exploration of the reachable space towards interesting areas. Artificial curiosity [Oudeyer and Kaplan, 2004] is a recent research domain focusing in creating goals and action strategies maximizing the learning progress of robots through neither too predictable nor too unpredictable situations. The corresponding intrinsic motivation mechanism is a source of self-development for the robot that only tries to learn what is "learnable" while avoiding already known or too difficult situations (see Figure 7.3).

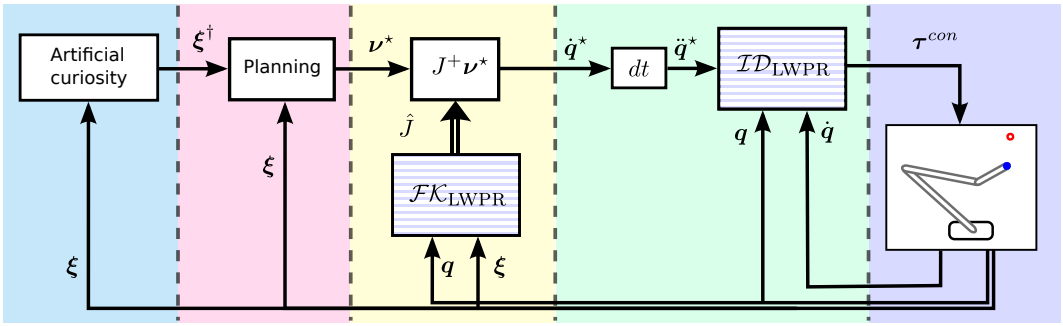


Figure 7.3: Artificial curiosity and planning methods as future works to our control schemes.

More generally, active learning [Robbel, 2005] takes into account the fact that the robot does not have to learn a model everywhere but only where it needs to. The performance of a learning control system can be improved with an efficient exploration strategy which consists in exploring an area of high uncertainty. Active learning is superior in the sense that it focuses the learning effort in the most useful space. Such methods may be useful using algorithms such as LWPR, where a random bootstrap is necessary to cover the entire space in a homogeneous manner while the robot will only use a subspace of its reachable domain.

7.3.3 Inverse dynamics, actuator models and muscles

In the introduction of this thesis, we explained that, in the service Robotics context, robots would have to deal with many contacts with their environment, both when interacting with human users and when using objects or tools. However, in most experiments presented in this thesis, we learn models while the robot is moving freely in an empty

space. The only case of interaction that we modelled is the case of a perturbation with a constant force, which does not summarise all the kinds of interactions a robot may meet in a service Robotics context.

Artificial muscles is a promising approach to endow a robot with the compliance properties that will be necessary to deal with more general interaction contexts. In particular, artificial muscles seem to be an easy way to control the impedance of mechanical systems as they are physically less rigid than motors. The co-contraction of two antagonist muscles can be tuned dynamically, giving rise to changes in the compliance of the system required by the context. However, the control of such device is difficult as the mechanics of such systems is extremely non-linear and a model is thus difficult to obtain. One can think about using machine learning methods presented in this thesis to control artificial muscles such as pneumatic actuators presented in Figure 7.4. The function that results from phenomena such as friction and undesired couplings is highly dependent on external parameters such as temperature and humidity of the environment that cannot be controlled and that are difficult to model. One should think about learning the dynamics of a single muscle using its physical input/output. For example, it seems possible to learn a mapping between the current of an electro-valve actioning a muscle and the resulting generated force. It may then be possible to use this model, or an inverse of it, to control this non-linear system.

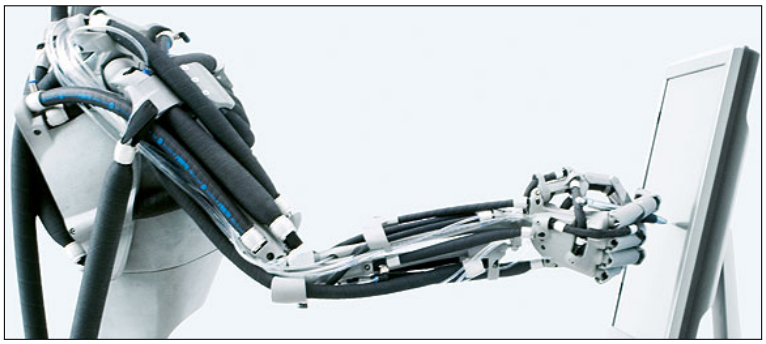


Figure 7.4: Festo arm with pneumatic actuation.

A Linear regression

Linear regression is used to model a linear relationship between a vector input $\boldsymbol{x}$ named regressor and a scalar output y named regressand. The objective is thus to find a model named A, which link those variables.

In this appendix, the number of input data is considered to be sufficient and linearly independent to have an invertible matrix $X_{n_p}X_{n_p}^T$.

A.1 Least squares

Linear regression can easily be computed using the least squares algorithm. This algorithm consists in computing a linear model A_{n_p} from a huge amount of data stored along time such as

$$Y_{n_p} = A_{n_p}X_{n_p} \tag{A.1}$$

where n_p is the number of encountered points, $X_{n_p} = [\boldsymbol{x}_1\ \boldsymbol{x}_2\ ...\ \boldsymbol{x}_{n_p}]$ the stored vectorial input of dimension n and $Y_{n_p} = [\boldsymbol{y}_1\ \boldsymbol{y}_2\ ...\ \boldsymbol{y}_{n_p}]$ the stored vectorial output of dimension m. Thus, A_{n_p} is a matrix of dimension $[m \times n]$

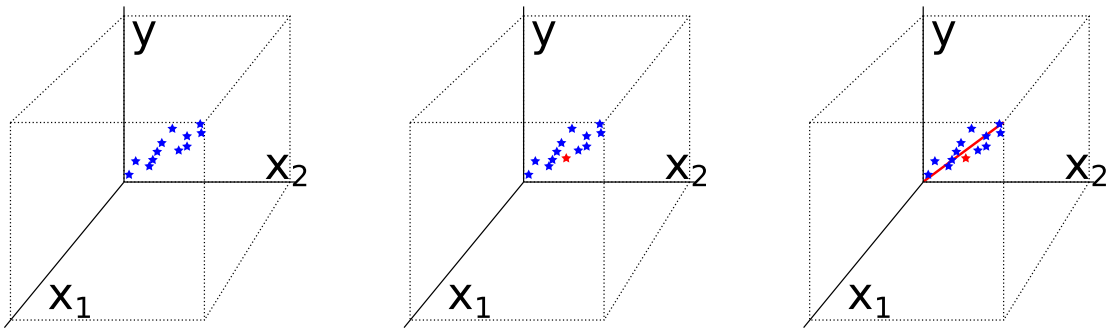


Figure A.1: Least squares method

Multiplying Equation A.1 from both side by $X_{n_p}^T(X_{n_p}X_{n_p}^T)^{-1}$, we obtain

$$Y_{n_p}X_{n_p}^T(X_{n_p}X_{n_p}^T)^{-1} = A_{n_p}X_{n_p}X_{n_p}^T(X_{n_p}X_{n_p}^T)^{-1} \tag{A.2}$$

which can be rewritten

$$A_{n_p} = Y_{n_p}X_{n_p}^+ \tag{A.3}$$

where $X_{n_p}^{+} = X_{n_p}^T (X_{n_p} X_{n_p}^T)^{-1}$ is called the pseudo-inverse of X_{n_p} [1]

This model A_{n_p} can be used to predict any $\boldsymbol{y}_{n_p}$ corresponding to the input $\boldsymbol{x}_{n_p}$ such that

$$\hat{\boldsymbol{y}}_{n_p} = A_{n_p} \boldsymbol{x}_{n_p}. \tag{A.4}$$

The matrix A_{n_p} is the model minimising the sum of squared errors between the estimated output $\hat{\boldsymbol{y}}$ and the measured output $\boldsymbol{y}$.

Two main problems appear using this method. The first one is the fact that we need to store all points to compute the model. The second problem is the computational cost. Inverting the matrix $(X_{n_p} X_{n_p}^T)$ of Equation A.2 is computationally expensive and the computational cost increases with the number of stored data.

A.2 Recursive least squares

The least squares can be transformed into an incremental version as follows

$$A_{n_p+1} = f(A_{n_p}, \boldsymbol{x}_{n_p}, \boldsymbol{y}_{n_p}) \tag{A.5}$$

Thus, A_{n_p} can be viewed as the memory of all n_p previous presented data.

Multiplied on both side of the equality by $X_{n_p}^T$, Equation A.1 becomes

$$E_{n_p} = A_{n_p} F_{n_p}$$

where $E_{n_p} = Y_{n_p} X_{n_p}^T$ and $F_{n_p} = X_{n_p} X_{n_p}^T$.

By definition,

$$E_{n_p} = Y_{n_p} X_{n_p}^T = Y_{n_p-1} X_{n_p-1}^T + \boldsymbol{y}_{n_p} \boldsymbol{x}_{n_p}^T$$

thus

$$E_{n_p} = E_{n_p-1} + \boldsymbol{y}_{n_p} \boldsymbol{x}_{n_p}^T. \tag{A.6}$$

Replacing E_{n_p-1} by $A_{n_p-1} F_{n_p-1}$ in Equation A.6 we obtain

$$E_{n_p} = A_{n_p-1} F_{n_p-1} + \boldsymbol{y}_{n_p} \boldsymbol{x}_{n_p}^T. \tag{A.7}$$

By adding and subtracting the term $A_{n_p-1} \boldsymbol{x}_{n_p} \boldsymbol{x}_{n_p}^T$ from Equation A.7, we obtain

$$E_{n_p} = A_{n_p-1} F_{n_p-1} + \boldsymbol{y}_{n_p} \boldsymbol{x}_{n_p}^T + A_{n_p-1} \boldsymbol{x}_{n_p} \boldsymbol{x}_{n_p}^T - A_{n_p-1} \boldsymbol{x}_{n_p} \boldsymbol{x}_{n_p}^T$$

and thus

$$E_{n_p} = A_{n_p-1} \left(F_{n_p-1} + \boldsymbol{x}_{n_p} \boldsymbol{x}_{n_p}^T \right) + \boldsymbol{y}_{n_p} \boldsymbol{x}_{n_p}^T - A_{n_p-1} \boldsymbol{x}_{n_p} \boldsymbol{x}_{n_p}^T$$

[1] It can be noticed that the pseudo-inverse can be computed with the SVD algorithm, one can thus avoid the standard inversion problem.

which can be rewritten

$$E_{n_p} = A_{n_p-1}F_{n_p} + \left(\boldsymbol{y}_{n_p} - A_{n_p-1}\boldsymbol{x}_{n_p}\right)\boldsymbol{x}_{n_p}^T.$$

Consequently we can write

$$E_{n_p}F_{n_p}^{-1} = A_{n_p-1} + \left(\boldsymbol{y}_{n_p} - A_{n_p-1}\boldsymbol{x}_{n_p}\right)\boldsymbol{x}_{n_p}^T F_{n_p}^{-1}$$

and thus the expected recursive equation

$$A_{n_p} = A_{n_p-1} + \left(\boldsymbol{y}_{n_p} - A_{n_p-1}\boldsymbol{x}_{n_p}\right)\boldsymbol{x}_{n_p}^T F_{n_p}^{-1}. \tag{A.8}$$

Equation A.8 is the recursive equation used to update the linear model A at each time-step.

The computation of $F_{n_p}^{-1}$ is expensive and, while it is possible to compute it recursively, people use to replace $F_{n_p}^{-1}$ in Equation A.8 by a learning rate α to obtain

$$A_{n_p} = A_{n_p-1} + \left(\boldsymbol{y}_{n_p} - A_{n_p-1}\boldsymbol{x}_{n_p}\right)\boldsymbol{x}_{n_p}^T\alpha. \tag{A.9}$$

The higher is α, the more the current modelling error affects the update of the linear model A. In this formulation, one can forget previous data that can be considered wrong. *A contrario*, one can also evolve the model slowly by decreasing α.

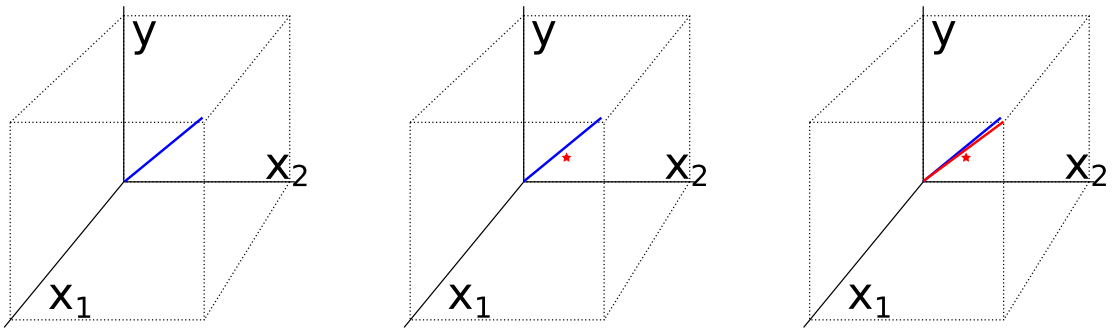


Figure A.2: Recursive least square

While Equation A.8 is mathematically identical to the least squares algorithm presented in Equation A.3, numerical errors can appear by providing the same point many times when using Equation A.9.

The resulting degeneracy phenomena are relevant on adaptive control because, when the trajectory of a learning system converges, it always samples the same data.

A.3 Partial least squares

The assumption made by using PLS is that the considered system is dependent upon data which dimension are lower than the dimension of the input data. Partial least squares is the same algorithm as recursive least squares but with a projection performed on the input to keep only convenient data named latent vectors.

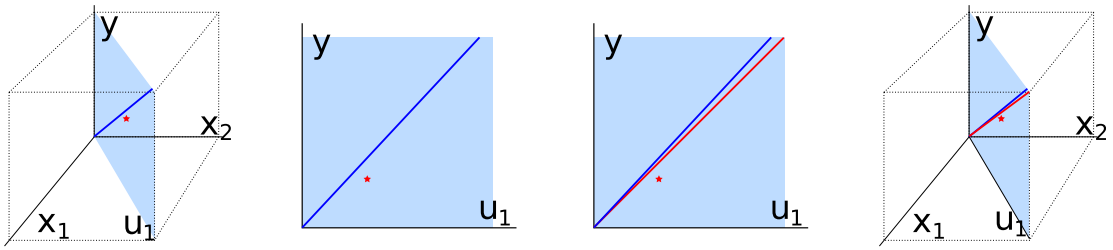


Figure A.3: Partial least square

The goal of PLS is to estimate an output $\boldsymbol{y}$ while having an inner relation between $\boldsymbol{x}$ and $\boldsymbol{y}$.

$$X = TV^T$$

$$Y = UW^T$$

The simplest relation between the two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ consists in relating both score matrices with a linear regression coefficient A_i, projecting u_i onto t_i

$$u_i \leftarrow A_i t_i^T \tag{A.10}$$

where $A_i = (u_i^T t_i)/(t_i^T t_i)$

The inner relation between the two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ can be improved by exchanging both score matrices in an incremental loop until convergence equivalent to the NIPALS algorithm.

Concerning the input X, the error of the PLS algorithm is thus computed as

$$error_X = X - \sum_{i=1}^{rank(X)} T_i V_i^T.$$

Concerning the output, the error is computed as

$$error_Y = Y - \sum_{i=1}^{rank(X)} U_i W_i^T.$$

Algorithm 7 The PLS algorithm

$E_1 \leftarrow X$
$F_1 \leftarrow Y$
for $i = 1$ to r **do**
  $t \leftarrow rand(size(X, 1), 1)$ *first arbitrary random score vector t*
  $u \leftarrow rand(size(Y, 1), 1)$ *first arbitrary random score vector u*
  $v \leftarrow (u^T E_i)/(u^T u)$ *project E_i onto u to create a first v*
  $w \leftarrow (t^T F_i)/(t^T t)$ *project F_i onto t to create a first w*
  $error_v = 10$ *first arbitrary error*
  $error_w = 10$ *second arbitrary error*
  while $error_v > error_{max}$ & $error_w > error_{max}$ **do**
    $v_{old} \leftarrow v$ *save the loading vector v*
    $w_{old} \leftarrow w$ *save the loading vector w*
    $v \leftarrow (u^T E_i)/(u^T u)$ *project E_i onto u, exchanging scores*
    $w \leftarrow (t^T F_i)/(t^T t)$ *project F_i onto t, exchanging scores*
    $v \leftarrow v/\|v\|$ *normalize the loading vector v*
    $w \leftarrow w/\|w\|$ *normalize the loading vector w*
    $t \leftarrow (E_i v)/(v^T v)$ *project the matrix X onto v*
    $u \leftarrow (F_i w)/(w^T w)$ *project the matrix Y onto w*
    $error_v \leftarrow \|v_{old} - v\|$ *error between the previous and current scores*
    $error_w \leftarrow \|w_{old} - w\|$ *error between the previous and current scores*
  end while
  $E_{i+1} = E_i - tp^T$ *remove principal component from previous residual*
  $F_{i+1} = F_i - uq^T$ *remove principal component from previous residual*
  $T_i \leftarrow t$ *store the score vector t*
  $V_i \leftarrow v$ *store the loading vector v*
  $U_i \leftarrow u$ *store the score vector u*
  $W_i \leftarrow w$ *store the loading vector w*
end for

It is possible to replace u_i using EquationA.10 to obtain

$$error_Y = Y - \sum_{i=1}^{rank(X)} A_s t_s^T W_s^T$$

Since the rank of Y is not decreased by one at each iteration, the outer loop can process until the last eigenvalue of X. We can notice here that iterating over all eigenvalue is not necessary as the last eigenvalue are not convenient to estimate Y. Some processes estimate the improvement of adding a projection component before applying it really or not.

B Linear systems and pseudo-inverses

Given a matrix $A \in \mathbb{R}^{m \times n}$ of rank r and two vectors $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$, a problem consists in finding a vector $\boldsymbol{x}$ which satisfy the equation

$$\boldsymbol{y} = A\boldsymbol{x}. \tag{B.1}$$

The input vector $\boldsymbol{x}$ is also named regressor or sample vector and the output y is named regressand. We can notice here that any problem which can be written $\boldsymbol{y} = A\boldsymbol{x} + b$, where b is an offset, can be expressed as Equation B.1 by augmenting $\boldsymbol{x}$ with a one as

$$\boldsymbol{y} = [A\ b] \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}. \tag{B.2}$$

B.1 Null Space

Also named kernel, the null space of the $m \times n$ matrix A is the subset $\mathcal{N}(A)$ such as

$$\mathcal{N}(A) = \{x \in \mathbb{R}^n | A\boldsymbol{x} = 0\}.$$

B.2 Generalized pseudo-inverse

In the case $m = n$, the matrix A is square. If A is full row rank, i.e., $r = m = n$, a solution of that problem exists and is given using the inverse of the matrix A as

$$\boldsymbol{x} = A^{-1}\boldsymbol{y}. \tag{B.3}$$

The case where $m > n$ describes a non-square matrix with no exact solution which does not interest us.

In the case $m < n$, the matrix A is not square and is not classically invertible. There is an infinite number of inverses for A which are solution to Equation B.1. In the non-singular case, i.e., $\text{rank}\,(A) = m$, there is an infinite number of generalised inverses of A written $A^\sharp$ (see Ben Israel and Greville [2003] for details)

$$\boldsymbol{x} = A^\sharp \boldsymbol{y} \tag{B.4}$$

Among these inverses, weighted pseudo-inverses [Doty et al., 1993; Park et al., 2001] provide minimum norm solutions. If rank $(A) = m$, they can be written as

$$A^{W+} = W^{-1}A^T[AW^{-1}A^T]^{-1}, \tag{B.5}$$

where W is a symmetric and positive definite matrix of dimension $n \times n$. The Moore-Penrose inverse or pseudo-inverse A^+ corresponds to the case where $W = I_n$. Then

$$\boldsymbol{x} = A^+\boldsymbol{y} \tag{B.6}$$

is a least squares solution to $\boldsymbol{y} = A\boldsymbol{x}$.

B.3 Eigen-decomposition

The bilinear decomposition (see [Abdi and Williams, 2010]) of a matrix A is written as a product of two matrices, an orthogonal loading matrix C and an orthonormal matrix of factor scores matrix T

$$A = TV^T.$$

The score matrix T represents distance from the origin of the projection of the sample vector $\boldsymbol{x}$ along the principal component direction relative to the eigenvalue i.

V is composed of the eigenvectors of A^TA arranged in columns. This matrix is also named projection matrix as the eigenvectors form a basis of the kernel of the linear application A.

The NIPALS algorithm can be used to generate that bilinear decomposition. This algorithm does not compute all the principal components at once. It first computes the principal component from the first score and loading vectors $\boldsymbol{t}_1$ and $\boldsymbol{v}_1$. Then $\boldsymbol{t}_1\boldsymbol{v}_1^T$ is subtracted from A to compute the residual E_1. The residual is then used to compute the second score and loading vectors $\boldsymbol{t}_2$ and $\boldsymbol{v}_2$. This process is iterated until the rank of A equals 0 or once the precision of the product $[\boldsymbol{t}_1 \cdots \boldsymbol{t}_i]\,[\boldsymbol{v}_1 \cdots \boldsymbol{v}_i]^T$ is sufficient to predict A.

B.4 Singular Value Decomposition

It is possible to decompose the matrix A using the Singular Value Decomposition (SVD) method [Golub and Van Loan, 1996], a generalization of the eigen-decomposition. The SVD decomposes any rectangular matrix A into three simple matrices as

$$A = UDV^T \tag{B.7}$$

where U and V are orthogonal matrices of dimensions $m \times m$ and $n \times n$ respectively. D is an $m \times n$ diagonal matrix with a diagonal composed of the m singular values of

Algorithm 8 The NIPALS algorithm

$E_1 \leftarrow A$
for $i = 1$ to r **do**
  $\boldsymbol{t} \leftarrow rand(m, 1)$ — *first arbitrary random score vector t*
  $\boldsymbol{v} \leftarrow (\boldsymbol{t}^T E_i)/(\boldsymbol{t}^T \boldsymbol{t})$ — *project E_i onto t to create a first v*
  $error_v = 10$ — *first arbitrary error*
  while $error_v > error_{max}$ **do**
    $\boldsymbol{v}_{old} \leftarrow \boldsymbol{v}$ — *save the loading vector v*
    $\boldsymbol{v} \leftarrow (\boldsymbol{t}^T E_i)/(\boldsymbol{t}^T \boldsymbol{t})$ — *project E_i onto t*
    $\boldsymbol{v} \leftarrow \boldsymbol{v}/\|\boldsymbol{v}\|$ — *normalize the loading vector v*
    $\boldsymbol{t} \leftarrow (E_i \boldsymbol{v})/(\boldsymbol{v}^T \boldsymbol{v})$ — *project the matrix E_i onto v*
    $error_v \leftarrow \|\boldsymbol{v}_{old} - \boldsymbol{v}\|$ — *error between the previous and current loading vectors*
  end while
  $E_{i+1} = E_i - \boldsymbol{t}\boldsymbol{v}^T$ — *remove principal component from previous residual*
  $T_i \leftarrow \boldsymbol{t}$ — *store the score vector t*
  $V_i \leftarrow \boldsymbol{v}$ — *store the loading vector v*
end for
$error_{\text{NIPALS}} = A - TV^T$

A in decreasing order. The singular values are the square roots of the eigenvalues of the matrix $A^T A$ in the case of redundant robots. The columns of V are the normalized eigenvectors of $A^T A$. Also called right singular vectors, they define, like in the eigen-decomposition, a basis for the null-space of the linear application A. Each eigenvector is ordered in column in the same order as the eigenvalues in D.

U: the (normalized) eigenvectors of the matrixAA^T (i.e., $P^T P = I$). The columns of P are called the left singular vectors of A.

Efficient computation of its Moore-Penrose pseudo-inverse A^+ can be done using the SVD

$$A^+ = VD^+U^T, \tag{B.8}$$

where the computation of D^+ is straightforward given its diagonal nature.

B.5 Weighted pseudo-inverse

A M_x, M_y weighted pseudoinverse of A can be written:

$$A_W^+ = M_x^{-1} G^T \left(G M_x^{-1} G^T\right)^{-1} \left(F^T M_y F\right)^{-1} F^T M_y, \tag{B.9}$$

where $A = FG$ is a full rank factorization of A, *i.e.* F and G are of dimension $m \times r$ and $r \times n$ respectively. M_x^{-1} and M_y are symmetric positive definite[1] matrices of dimension

[1]The Matlab function associated to this work can deal with the case where M_x^{-1} and M_y are semi-positive definite. The associated equations still have to be written in this document.

$n \times n$ and $m \times m$ respectively.

$A^\sharp$ provides the minimum M_x-weighted L_2 norm solution which minimises the M_y-weighted L_2 norm of the error $(\boldsymbol{y} - A\boldsymbol{x})$. Linear regression is used to model a linear relationship between a vector input $\boldsymbol{x}$ named regressor and a scalar output y named regressand. The W-weighted L_2 norm of any vector $\boldsymbol{x}$ is given by:

$$\|\boldsymbol{x}\|_W = \sqrt[2]{\boldsymbol{x}^T W \boldsymbol{x}}. \tag{B.10}$$

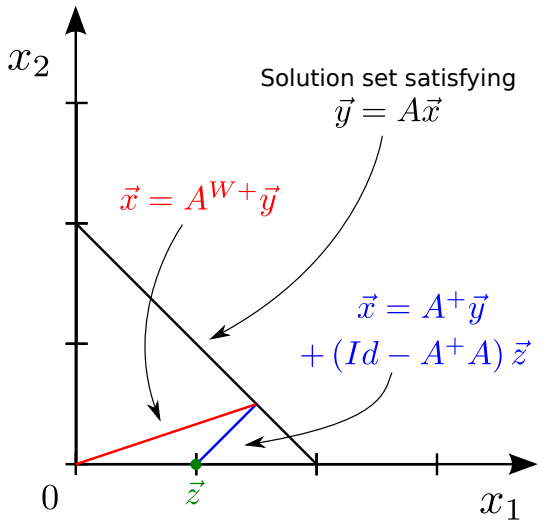


Figure B.1: Using weighted pseudo-inverse and projector method to find the same solution $\boldsymbol{x}$.

The regressor $\boldsymbol{x}$, solution to Equation B.1, which minimize the Euclidean norm $\sqrt{\dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}}}$ is defined as

$$\boldsymbol{x} = A^{+}\boldsymbol{y}. \tag{B.11}$$

The regressor $\boldsymbol{x}$, solution to Equation B.1, which minimize the weighted Euclidean norm $\sqrt[2]{\boldsymbol{x}^T W \boldsymbol{x}}$ is defined as

$$\boldsymbol{x} = A^{W+}\boldsymbol{y}, \tag{B.12}$$

Figure B.1 illustrate that this solution may also be reached by the projection method

$$\boldsymbol{x} = A^{+}\boldsymbol{y} + \left(Id - A^{+}A\right)\boldsymbol{z}, \tag{B.13}$$

Bibliography

Abdi, H. and Williams, L. (2010). Principal component analysis. *WIREs Comp Stat*.

Ahmad, Z. and Guez, A. (1990). On the solution to the inverse kinematic problem. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 1692–1697.

Atkeson, C. (1991). Using locally weighted regression for robot learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 958–963.

Baillieul, J. (1985). Kinematic programming alternatives for redundant manipulators. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 2, pages 722–728.

Barhen, J., Gulati, S., and Zak, M. (1989). Neutral learning of constrained nonlinear transformations. *Computer*, 22(6):67–76.

Barthélemy, S. and Micaelli, A. (2008). Arboris for Matlab.

Ben Israel, A. and Greville, T. (2003). *Generalized inverses: Theory and applications*. Springer, second edition. ISBN 0-387-00293-6.

Boudreau, R., Darenfed, S., and Gosselin, C. (1998). On the computation of the direct kinematics of parallel manipulators using polynomial networks. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 28(2):213–220.

Brock, O. and Khatib, O. (2002). Elastic strips: A framework for motion generation in human environments. *The International Journal of Robotics Research*, 21(12):1031.

Brüwer, M. and Cruse, H. (1990). A network model for the control of the movement of a redundant manipulator. *Biological Cybernetics*, 62(6):549–555.

Butz, M., Herbort, O., and Hoffmann, J. (2007). Exploiting redundancy for flexible behavior: Unsupervised learning in a modular sensorimotor control architecture. *Psychological Review*, 114(4):1015–1046.

Butz, M., Pedersen, G., and Stalph, P. (2009). Learning sensorimotor control structures with XCSF: redundancy exploitation and dynamic control. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1171–1178. ACM.

Butz, M. V. and Goldberg, D. (2003). Bounding the population size in XCS to ensure reproductive opportunities. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724, pages 1844–1856. Springer-Verlag.

Butz, M. V., Goldberg, D., and Lanzi, P. (2005). Computational Complexity of the XCS Classifier System. *Foundations of Learning Classifier Systems*, 51:91–125.

Butz, M. V., Goldberg, D., and Tharakunnel, K. (2003). Analysis and improvement of fitness exploitation in XCS: bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*, 11(3):239–277.

Butz, M. V. and Herbort, . (2008). Context-dependent predictions and cognitive arm control with XCSF. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1357–1364. ACM New York, NY, USA.

Butz, M. V., Kovacs, T., Lanzi, P. L., and Wilson, S. W. (2004). Toward a theory of generalization and learning in xcs. *IEEE Transactions on Evolutionary Computation*, 8(1):28–46.

Butz, M. V. and Wilson, S. (2000). An Algorithmic Description of XCS. In *Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems(IWLCS 2000)*, pages 253–272, Paris, France. Springer-Verlag.

Calinon, S. (2009). *Robot programming by demonstration*. EPFL/CRC Press.

Calinon, S., Guenter, F., and Billard, A. (2007). On Learning, Representing and Generalizing a Task in a Humanoid Robot. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(2):286–298.

Cannata, G., Maggiali, M., Metta, G., and Sandini, G. (2008). An embedded artificial skin for humanoid robots. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008*, pages 434–438.

Canudas de Wit, C., Olsson, H., and Lischinsky, P. (1995). A new model for control of systems with friction. *IEEE Transactions on automatic control*, 40(3):419–425.

Chiaverini, S. (1997). Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Transactions on Robotics and Automation*, 13(3):398–410.

Cohn, D., Ghahramani, Z., and Jordan, M. (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4(1):129–145.

Crisp, J., Adler, M., Matijevic, J., Squyres, S., Arvidson, R., and Kass, D. (2003). Mars exploration rover mission. *Journal of Geophysical Research-Planets*, 108(E12):8061.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

DeMers, D. and Kreutz-Delgado, K. (1997). Inverse kinematics of dextrous manipulators. *Neural Systems for Robotics*, pages 75–116.

Doty, K., Melchiorri, C., and Bonivento, C. (1993). A theory of generalized inverses applied to Robotics. *The International Journal of Robotics Research*, 12(1):1–19.

D'Souza, A., Vijayakumar, S., and Schaal, S. (2001). Learning inverse kinematics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 298–303.

Duindam, V. (2006). *Port-Based Modeling and Control for Efficient Bipedal Walking Robots*. PhD thesis, University of Twente.

Elden, L. (2004). Partial least-squares vs. Lanczos bidiagonalization-I: analysis of a projection method for multiple regression. *Computational Statistics and Data Analysis*, 46(1):11–31.

Fitzpatrick, P., Metta, G., and Natale, L. (2008). Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1):29–45.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Golub, G. H. and Van Loan, C. F. (1996). *Matrix computations*. The John Hopkins University Press, third edition. ISBN 0-8018-5414-8.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, Ann Arbor, MI.

Howard, M. and Vijayakumar, S. (2007). Reconstructing null-space policies subject to dynamic task constraints in redundant manipulators. *WS robotics and mathematics*.

Jordan, M. I. and Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16(3):307–354.

Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., and Hirukawa, H. (2003). Biped walking pattern generation by using preview control of zero-moment point. In *IEEE International Conference on Robotics and Automation, 2003. Proceedings. ICRA*, volume 2, pages 1620–1626.

Kanoun, O., Lamiraux, F., Wieber, P., Kanehiro, F., Yoshida, E., and Laumond, J. (2009). Prioritizing linear equality and inequality systems: application to local motion planning for redundant robots. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 724–729. Institute of Electrical and Electronics Engineers Inc., The.

Kavraki, L. and Latombe, J. (1998). Probabilistic roadmaps for robot path planning. *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, 53.

Kawato, M. (1993). Optimization and learning in neural networks for formation and control of coordinated movement. In *Attention and performance XIV (silver jubilee volume)*, page 849. MIT Press.

Kawato, M. (1999). Internal models for motor control and trajectory planning. *Current opinion in neurobiology*, 9(6):718–727.

Khalil, W. and Dombre, E. (2002). *Modeling, Identification & Control of Robots*. Butterworth-Heinemann.

Khatib, O. (1983). Dynamic control of manipulators in operational space. In *Sixth CISM-IFToMM Congress on Theory of Machines and Mechanisms*, pages 1128–1131.

Khatib, O. (1987). A unified approach to motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53.

Klanke, S., Vijayakumar, S., and Schaal, S. (2008). A library for locally weighted projection regression. *The Journal of Machine Learning Research*, 9:623–626.

Kober, J. and Peters, J. (2008). Policy search for motor primitives in robotics. *Advances in Neural Information Processing Systems (NIPS)*, pages 1–8.

Kohonen, T. (2001). *Self-Organizing Maps*. Springer, Berlin.

Kovacs, T. (1999). Deletion Schemes for Classifier Systems. In *GECCO-99: proceedings of the Genetic and Evolutionary Computation Conference: a joint meeting of the eighth International Conference on Genetic Algorithms (ICGA-99) and the fourth annual Genetic Programming Conference (GP-99)*, pages 329–336, Orlando, Florida.

Lamy, X., Colledani, F., Geffard, F., Measson, Y., and Morel, G. (2009). Robotic skin structure and performances for industrial robot comanipulation. In *IEEE Advanced Intelligent Mechatronics (AIM'09)*, pages 427–432, Singapore.

Landau, I., Lozano, R., M'saad, M., Modestino, J., Fettweis, A., Massey, J., Thoma, M., Sontag, E., and Dickinson, B. (1998). *Adaptive control*. Springer-Verlag New York, Inc. Secaucus, NJ, USA.

Latombe, J. (1991). *Robot Motion Planning*. Kluwer Academic.

Laumond, J. (1998). *Robot motion planning and control*. Springer.

LaValle, S. (2006). *Planning algorithms*. Cambridge Univ Pr.

Lee, C., Lei, L., and Pinedo, M. (1997). Current trends in deterministic scheduling. *Annals of Operations Research*, 70:1–41.

Lee, S. and Kil, R. (1990). Robot kinematic control based on bidirectional mapping neural network. In *Proceedings of International Joint Conference on Neural Networks (IJCNN)*, volume 3, pages 327–335.

Li, W. (2006). *Optimal Control for Biological Movement*. PhD thesis, University of California, San Diego.

Liégeois, A. (1977). Automatic supervisory control of the configuration and behavior of multibody mechanisms. *Systems, Man and Cybernetics, IEEE Transactions on*, 7(12):868–871.

Lin, S. and Goldenberg, A. (2001). Neural-network control of mobile manipulators. *IEEE Transactions on Neural Networks*, 12(5):1121–1133.

Liu, T. and Wang, M. Y. (2005). Computation of three-dimensional rigid-body dynamics with multiple unilateral contacts using time-stepping and gauss-seidel methods. *IEEE Transactions on Automation Science and Engineering*, 2(1):19–31.

Ljung, L. (1986). *System identification: theory for the user*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

Lussier, B., Gallien, M., Guiochet, J., Ingrand, F., Killijian, M., and Powell, D. (2007). Planning with Diversified Models for Fault-Tolerant Robots. In *Proceedings of The International Conference on Automated Planning and Scheduling (ICAPS07)*.

Maciejewski, A. and Klein, C. (1985). Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *The International Journal of Robotics Research*, 4(3):109–117.

Mahalanobis, P. (1936). On the generalized distance in statistics. In *Proceedings of the National Institute of Science, Calcutta*, volume 12, page 49.

Mansard, N. and Chaumette, F. (2007). Task sequencing for sensor-based control. *IEEE Transactions on Robotics*, 23(1):60–72.

Martinetz, T., Bitter, H., and Schulten, K. (1990a). Learning of Visuomotor-Coordination of a robot arm with redundant degrees of freedom. In *Proceedings of the Third International Symposium on Robotics and Manufacturing–Research, Education, and Applications*, page 521. Amer Society of Mechanical.

Martinetz, T., Ritter, H., and Schulten, K. (1990b). Three-dimensional neural net for learning visuomotor coordination of a robot arm. *IEEE Transactions on Neural Networks*, 1(1):131–136.

Metta, G., Sandini, G., Vernon, D., Natale, L., and Nori, F. (2008). The iCub humanoid robot: an open platform for research in embodied cognition. In *Permis: performance metrics for intelligent systems workshop*, Washington DC, USA.

Mitrovic, D., Klanke, S., and Vijayakumar, S. (2008). Adaptive Optimal Control for Redundantly Actuated Arms. In *From Animals to Animats 10: 10th International Conference on Simulation of Adaptive Behavior, Sab 2008, Osaka, Japan, July 7-12, 2008, Proceedings*, page 93. Springer.

Murray, R. M., Sastry, S. S., and Li, Z. (1994). *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA.

Nakamura, Y. (1990). *Advanced robotics: redundancy and optimization*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Nakamura, Y. (1991). *Advanced Robotics: redundancy and optimization*. Addison Wesley. ISBN 0-201-15198-7.

Nakanishi, J., Cory, R., Mistry, M., Peters, J., and Schaal, S. (2008). Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757.

Narendra, K. and Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27.

Natale, L., Nori, F., Metta, G., and Sandini, G. (2007). Learning precise 3d reaching in a humanoid robot. In *Proceedings of the IEEE International Conference of Development and Learning (ICDL)*, pages 1–6, London, UK.

Nguyen, L., Patel, R., and Khorasani, K. (1990). Neural network architectures for the forward kinematics problem in robotics. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 3, pages 393–399.

Nguyen-Tuong, D., Peters, J., Seeger, M., and Schoelkopf, B. (2008). Learning inverse dynamics: a comparison. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*.

Nguyen-Tuong, D., Seeger, M., and Peters, J. (2009). Real-time local gp model learning. In *From motor to interaction learning in robots*.

Oudeyer, P. and Kaplan, F. (2004). Intelligent adaptive curiosity: a source of self-development. In *Proceedings of the 4th International Workshop on Epigenetic Robotics*, volume 117, pages 127–130. Citeseer.

Park, F. C. and Ravani, B. (1997). Smooth invariant interpolation of rotations. *ACM Trans. Graph.*, 16(3):277–295.

Park, J. (2005). Principle of Dynamical Balance for Multibody Systems. *Multibody System Dynamics*, 14(3-4):269–299.

Park, K. C., Chang, P.-H., and Lee, S. (2001). Analysis and control of redundant manipulator dynamics based on an extended operational space. *Robotica*, 19(6):649–662.

Peters, J., Mülling, K., Kober, J., Nguyen-Tuong, D., and Krömer, O. (2009). Towards motor skill learning for robotics. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, pages 1–14.

Peters, J. and Schaal, S. (2008). Learning to control in operational space. *The International Journal of Robotics Research*, 27(2):197–212.

Petkos, G., Toussaint, M., and Vijayakumar, S. (2006). Learning multiple models of non-linear dynamics for control under varying contexts. *Artificial Neural Networks–ICANN 2006*, pages 898–907.

Petkos, G. and Vijayakumar, S. (2007). Load estimation and control using learned dynamics models. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1527–1532. Citeseer.

Pourboghrat, F. (1989). Neural networks for learning inverse-kinematics of redundant manipulators. In *Proceedings of the 32nd Midwest Symposium on Circuits and Systems*, volume 2, pages 760–762.

Quinlan, S. and Khatib, O. (1993). Elastic bands: Connecting path planning and control. In *IEEE International Conference on Robotics and Automation*, pages 802–802. Citeseer.

Ritter, H. J., Martinetz, T. M., and Schulten, K. J. (1989). Topology-conserving maps for learning visuo-motor-coordination. *Neural networks*, 2(3):159–168.

Robbel, P. (2005). *Active Learning in Motor Control*. Master's Thesis, University of Edinburgh, UK.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408.

Ruiz de Angulo, V. and Torras, C. (2002). Learning inverse kinematics via cross-point function decomposition. *Artificial Neural Networks-ICANN 2002*, pages 134–134.

Sadjadian, H. and Taghirad, H. D. (2004). Numerical methods for computing the forward kinematics of a redundant parallel manipulator. In *Proceedings of the IEEE Conference on Mechatronics and Robotics*, Aachen, Germany.

Salaün, C., Padois, V., and Sigaud, O. (2009a). Control of redundant robots using learned models: an operational space control approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 878–885.

Salaün, C., Padois, V., and Sigaud, O. (2009b). A two-level model of anticipation-based motor learning for whole body motion. In Pezzulo, G., Butz, M., Sigaud, O., and Baldassarre, G., editors, *Anticipatory Behavior in Adaptive Learning Systems, From Psychological Theories to Artificial Cognitive Systems*, volume 5499 of *Lecture Notes in Computer Science / Lecture Notes in Artificial Intelligence*, pages 229–246. Springer.

Salaün, C., Padois, V., and Sigaud, O. (2010). Learning forward models for the operational space control of redundant robots. In Sigaud, O. and Peters, J., editors, *From Motor Learning to Interaction Learning in Robots*, chapter 8, pages 170–194. Springer.

Sang, L. H. and Han, M.-C. (1999). The estimation for forward kinematic solution of stewart platform using the neural network. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 501–506.

Sastry, S. and Bodson, M. (1989). *Adaptive control: stability, convergence, and robustness*. Prentice Hall.

Schaal, S. and Atkeson, C. G. (1997). Receptive field weighted regression. *ART Human Inf. Process. Lab., Kyoto, Japan, Tech. Rep. TR-H-209*.

Schaal, S., Atkeson, C. G., and Vijayakumar, S. (2002). Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, 17(1):49–60.

Schaal, S., Vijayakumar, S., and Atkeson, C. (1998). Local dimensionality reduction. *Advances in neural information processing systems*, pages 633–639.

Selig, J. M. (2005). *Geometric fundamentals of robotics*. Springer Verlag.

Sentis, L. and Khatib, O. (2005). Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *The International Journal of Humanoid Robotics*, 2(4):505–518.

Shadmehr, R. and Mussa-Ivaldi, F. (1994). Adaptive representation of dynamics during learning of a motor task. *Journal of Neuroscience*, 14(5):3208.

Shon, A., Grochow, K., and Rao, R. (2005). Robotic imitation from human motion capture using gaussian processes. In *Proceedings of the IEEE-RAS/RSJ International Conference on Humanoid Robots (Humanoids)*.

Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2008). *Robotics: modelling, planning and control*. Springer Verlag.

Smits, R., De Laet, T., Claes, K., Soetens, P., De Schutter, J., and Bruyninckx, H. (2008). Orocos: A software framework for complex sensordriven robot tasks. *IEEE Robotics and Automation Magazine*.

Snyman, J. A. (2005). *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*. Springer.

Someya, T., Sekitani, T., Iba, S., Kato, Y., Kawaguchi, H., and Sakurai, T. (2004). A large-area, flexible pressure sensor matrix with organic field-effect transistors for artificial skin applications. *Proceedings of the National Academy of Sciences of the United States of America*, 101(27):9966.

Stalph, P., Butz, M., and Pedersen, G. (2009). Controlling a four degree of freedom arm in 3D using the XCSF learning classifier system. *KI 2009: Advances in Artificial Intelligence*, pages 193–200.

Stramigioli, S. and Bruyninckx, H. (2001). Geometry and Screw Theory for Robotics. In *ICRA 2001*.

Sun, G. and Scassellati, B. (2004). Reaching through learned forward model. In *4th IEEE/RAS International Conference on Humanoid Robots*, volume 1, pages 93–112, Los Angeles, USA.

Sun, G. and Scassellati, B. (2005). A fast and efficient model for learning to reach. *International Journal of Humanoid Robotics*, 2(4):391–414.

Sutton, R. and Barto, A. (1998). *Reinforcement learning*. MIT Press.

Tenenhaus, M. (1998). *La régression PLS: théorie et pratique*. Editions Technip.

Tikhanoff, V., Fitzpatrick, P., Nori, F., Natale, L., Metta, G., and Cangelosi, A. (2008). The icub humanoid robot simulator. *International Conference on Intelligent RObots and Systems IROS, Nice, France*.

Todorov, E. (2004). Optimality Principles in Sensorimotor Control. *Nat. Neuroscience*, 7(9):907–915.

Van der Smagt, P. P. (1994). Minimisation methods for training feedforward neural networks. *Neural Networks*, 7(1):1–11.

Vijayakumar, S., D'Souza, A., and Schaal, S. (2005). LWPR: A scalable method for incremental online learning in high dimensions. Technical report, Edinburgh University Press.

Vijayakumar, S. and Schaal, S. (1997). Local dimensionality reduction for locally weighted learning. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 220–225.

Vijayakumar, S. and Schaal, S. (2000). Locally weighted projection regression: An o (n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford, CA*, pages 1079–1086.

Voda, A. A. and Landau, I. D. (1992). An analytical method for the auto-calibration of pid controllers. *Proceedings of the 31st IEEE Conference on Decision and Control*, 4:3237 – 3242.

Voda, A. A. and Landau, I. D. (1995). A method for the auto-calibration of pid controllers. *Automatica*, 31(1):41–53.

Walter, J. and Schulten, K. (1993). Implementation of self-organizing neural networks for visuo-motor control of an industrial robot. *IEEE Transactions on Neural Networks*, 4(1):86–96.

Wang, D. and Zilouchian, A. (1997). Solutions of kinematics of robot manipulators using a kohonen self-organizing neural network. In *Proceedings of the IEEE International Symposium on Intelligent Control*, pages 251–255.

Whitney, D. E. (1969). Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine Systems*, 10(2):47–53.

Wieber, P. B. (2006). Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 137–142, University of Genova, Genova, Italy.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175.

Wilson, S. W. (2001). Function approximation with a classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 974–981, San Francisco, California, USA. Morgan Kaufmann.

Wilson, S. W. (2002). Classifiers that Approximate Functions. *Natural Computing*, 1(2-3):211–234.

Wold, H. (1975). Soft modelling by latent variables: the non-linear iterative partial least squares (NIPALS) approach. *Perspectives in Probability and Statistics (papers in honour of MS Bartlett on the occasion of his 65th birthday)*, pages 117–142.