Autonomous sensori-motor learning in a humanoid robot

Lorenzo Jamone

DIST

University of Genoa, Italy
Italian Institute of Technology

Supervisors:
Prof. Lorenzo Natale
Prof. Giorgio Metta

A thesis submitted for the degree of

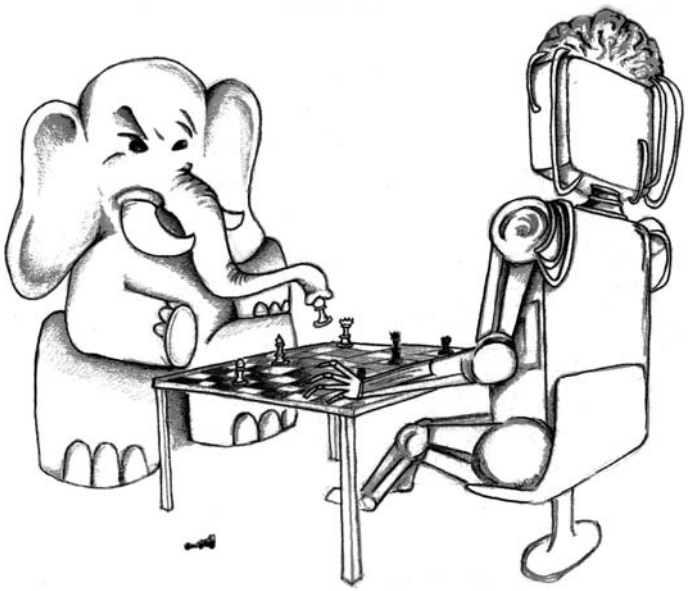
PhilosophiæDoctor (PhD)

February 2010

Abstract

Humanoid robots that are thought to operate alongside humans, helping them at home or at workplace by sharing the same daily objects and physical constraints, must exhibit adaptive and flexible behavior. We believe that the most efficient and elegant way to deal with such a situation (and perhaps the only one) is to develop a form of *embodied cognition*, enabling the robot to learn autonomously how to behave by acting on the world (*action*) and understanding the consequences of its own actions from its sensory measurements (*perception*): to find the relation between *action* and *perception* is the process called *sensori-motor learning*. Furthermore, the study of learning and development in artificial systems can also help neuroscience and developmental psychology to gain insight about human cognition, by testing existing theories and by providing possible ideas for further investigations.
The main goal of this research is to build a bio-inspired framework which enables a humanoid robot to autonomously learn how to reach for a visually identified object. In doing so, we also hope to provide some general results about autonomous online learning in complex systems. The work is implemented on a real 22-DOF humanoid torso, equipped with moving eyes and head, a 7-DOF arm and an anthropomorphic hand. The robot is bootstrapped with a limited set of basic motor skills allowing to start interaction with the environment. Based on them, the robot will attain more advanced motor behaviors. The biological plausibility of the proposed approach is supported by evidence in neuroscience and developmental psychology. Gaze direction (i.e. head configuration while fixating a target object) is taken as a reference frame for the reaching action. Consequently, the first developmental step consists in the acquisition of neck orientation control,

which is the main prerequisite for gazing. This controller is learned online and autonomously by the robot, exploting sensory measurements and incremental learning algorithms. Concerning reaching, the proposed controller integrates a ballistic open-loop component and a closed-loop correction. The open-loop controller uses a arm-head forward kinematic map, while the closed-loop controller exploits the arm visual Jacobian; both functions are learned autonomously by the robot. Learning is performed purely online, without any separation between the training and the execution phase. Moreover, preliminary studies have been carried on to understand how force and tactile information can be integrated with vision and proprioception in order to better control the interaction with the external environment.



To my family...

Acknowledgements

I must apologize for all the people I'm forgetting at the moment of writing these acknowledgements. Anyway, if you feel to have been of any help, you probably did, and I thank you.
After this mandatory clarification, the first thank is to the people that substantially contributed to the present work. Matte, the 'mechanic' (how many tendons have you repaired?), Sere, Arjan (for LS-SVM analysis), Captain Randaz. Then, of course, the old guys. Lorenzo Natale, Giorgio Metta, Francesco Nori. They have been all fundamental for this work completion, in different ways and moments; in particular Lorenzo, who helped me a lot in the final revision of this thesis. Finally, Giulio Sandini, the boss. His passion for research has always inspired me, and actually made this work possible.
Then, all IIT colleagues, for 5 pm tea breaks, coffes and lunches together. In particular, I thank Fra Campa for the nice, interesting (and sometimes almost crazy!) talks. Moreover, all friends that made me smile at least once in the last three years deserve an acknowledgement, since being relaxed and happy helps more than anything. Then, I will never tire of saying that having Erica by my side has been extremely important to let the work stress gently disappear due to a smile, hug or kiss. Thanks my darling.
Concluding, I want to thank my family for... ...everything! Indeed, this thesis is dedicated to the Three Musketeers: mum, dad and syster. Am I the fourth one?

Contents

List of Figures

List of Tables

1

Introduction

"Computers are incredibly fast, accurate, and stupid. Human beings are incredibly slow, inaccurate, and brilliant. Together they are powerful beyond imagination." Albert Einstein.

The dream about the realization of intelligent artificial systems comes from the hope that bridging the gap between digital computers and human brains is actually possible. Indeed, even if humans are physically weak, mentally slow and dominated by emotions, they largely outperform current robots in most everyday tasks, mainly exploiting their learning and adaptation capabilities. Therefore, to take inspiration from biology in the realization of robot structures and controllers appears mandatory. Clearly, the first step to achieve this is to improve our knowledge about human intelligence, or, in other words, to understand what creates this huge gap between artificial and biological thinking.

We want humanoid robots to help humans in different aspects of daily living: assisting them at home, cooperating with them at work and even substituting them in the execution of the most tedious or dangerous jobs. Adaptability and flexibility become crucial requirements for such robots, that must operate in the unstructured dynamic environment in which we live and act. They should be able to cope efficiently with unexpected changes in the perceived environment as well as modifications of their own physical structure. We believe that the best way to achieve such performances is to develop a form of *embodied cognition*, enabling the robot to learn the relationship between its *actions* and the consequent *perceptions* through its own experience on the physical world (i.e. *interaction*).

Within this framework, the robot can build representations of its own structure and of the external world that are always up-to-date, based on the informations it gathers autonomously through interaction, exploiting multi-sensori integration and on-line learning algorithms. Moreover, by implementing learning and development strategies in artificial systems we can also test theories from neuroscience and developmental psychology about human cognition, also providing possible ideas for further investigations. In this first chapter we introduce the scientific framework in which this research has been carried out.

1.1 Humanoid robots

The word 'robot', introduced in 1921 by the Czech writer Karel Capek in his play *Rossum's Universal Robots*, comes from the word robota meaning literally serf labor, and, figuratively, 'drudgery' or 'hard work' in Czech, Slovak and Polish. The idea of having automaic machines doing work in place of humans was in fact not new, and hoped for since antiquity (*"If every tool, when ordered, or even of its own accord, could do the work that befits it... then there would be no need either of apprentices for the master workers or of slaves for the lords."* Aristotle, 322 B.C.). Indeed, the first robots appearing on the market, after about 30 years, were manipulators designed and realized for industrial purpose, in order to replace humans in the execution of repetitive tasks, in a completely known and structured environment, like an assembly line. In 1954 George Devol applied for patent No. 2,988,237 for the first robotic manipulator ever, named *Unimate*, which was issued in 1961. Unimate joined the assembly line at a General Motors plant to work with heated die-casting machines. The manipulator took die castings from machines and performed welding on auto bodies; tasks that are unpleasant for people.

Conversely, although the first project of a humanoid automatic machine has been realized by Leonardo Da Vinci in 1478 (probably inspired by the ideas of the Arab polymath Al-Jazari, which in 1206 wrote the *Book of Knowledge of Ingenious Mechanical Devices*, describing fifty mechanical devices along with instructions on how to construct them), humanoid robots showed up only during the last decades of the twentieth century. During the last 35 years, after the first one came to light

(1973: WABIAN-1, at Waseda University), humanoid robots have been improving substantially in terms of mechanics and electronics. In particular, apart from the exterior appearance, big advances have been achieved in terms of sensory equipment, number of degrees of freedom and overall robustness and reliability. Anyway, other important features have been partially left aside, and some fundamental requirements for humanoids are still missing.

Two main aspects can be highlighted which characterize humanoid robots, differentiating them from other kinds of robots (mobile robots, industrial robots, etc...). First, they have to resemble human beings in terms of physical appearence: they can miss some body parts, but the parts they possess must be human-like, at least from a functional point of view (e.g. number of joints, approximate size, range of motion). Second, they must mimic human autonomy both in behaving and in learning how to behave. In other words, humanoid robots must appear as humans and behave as humans. This is because we want them to operate in the same situations and conditions as humans, to use the same tools, to interact with and to understand the same world in which we daily live and act. In achieving this, we do not want to provide robots with a rigid set of rules driving their action, but we want them to be able to find their own rules, to learn how to behave in our world pushed by their own motivations, finding smart ways to cope with the challenges posed by the dynamic and unstructured environment in which they operate.

Human-like intelligence and autonomy is the main lacking element in current humanoids. Given that the solutions adopted by human beings to survive could help roboticists to find smart algorithms for robot learning, some foundamental questions arise.

In order to behave as we do, do humanoids have to learn in the same way we learn? How much the biomechanical differences could affect the learning processes? How the phylogenetical contribution to human intelligence can be replaced in robots? All these questions haven't received satisfactory answers yet, due to the limited knowledge we have about human intelligence and learning and to the youth of humanoid robotics as a science.

1.2 Artificial intelligence

"...every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it" John McCarthy, 1955

"...machines will be capable, within twenty years, of doing any work a man can do" Herbert Alexander Simon, 1965

Like that of automatic machines, the history of artificial intelligence began in antiquity. There have always been myths, stories and rumors of artificial beings endowed with intelligence or consciousness by master craftsmen. Independently, philosophers (beginning as early as Aristotle) have attempted to describe the process of human thinking as the mechanical manipulation of symbols. This work culminated in the invention of the programmable digital computer in the 1940s, a machine based on the abstract essence of mathematical reasoning. This device and the ideas behind it inspired a handful of scientists to begin seriously discussing the possibility of building an electronic brain.
Dartmouth conference in 1956 marks the birth of Artificial Intelligence as a science. Promoted by John Mc Carthy and Marvin Minsky, the conference was a one-month brainstorming session aimed at making advances in some fundamental topics which are still open problems today: how to make machines use language, form abstractions and concepts, solve the kind of problems tipically reserved for humans, and improve themselves. Cognitivism uses the metaphor of the mind as computer: information comes in, is being processed, and leads to certain outcomes. This approach sees an intelligent system as a rational reasoning device and is often called Classical AI (or Symbolic, Knowledge-based AI). Classical AI uses symbols to represent knowledge so that a machine can work with them to derive some additional knowledge.
After Darthmouth conference researchers expressed an intense optimism in private and in print, predicting that a fully intelligent machine would be built in less than 20 years. On May 11, 1997, *Deep Blue* won a six-game chess match (two wins to one with three draws) against the world champion Garry Kasparov. *Deep*

Blue was a chess-playing computer developed by IBM within a project started in 1989. Classical AI would consider *Deep Blue* as a machine showing intelligent behavior, considering intelligence the ability to manipulate symbols in order to find correct answers. Anyway, even if this approach has produced successful results in particular domains, it has also run into at least two fundamental problems. Firstly, symbolic systems lacked the ability to detect and use context information; even within their domain of operation they often failed to provide the correct answer in ambiguous situations where humans would easily be successful (consider for instance the problem of pattern recognition or language understanding). On the other hand these techniques seemed unable to generalize knowledge and use it across different domains; as a result it was impossible to scale up from limited, tractable domains to more complicated situations. The second fundamental problem is thus how to apply symbolic AI to real-life problems, usually encountered for instance in the field of computer vision and robotics (consider for example navigation and surveillance). In this case the most difficult problem is perhaps how to synthesize a symbolic representation of the real world (especially when the latter it is not 100 % deterministic), and relate it with the information received from the sensory system. Moreover, real-life problems, when reduced to formal logic, easily become computationally intractable.

1.3 Embodied cognition

"...it is unfair to claim that an elephant has no intelligence worth studying just because it does not play chess." Rodney Brooks, 1990

As opposed to Classical AI, Behavioral Based AI considers the existence of a body and its interaction with the environment a necessary condition for the emergence of intelligence and cognition. Digital computers are not seen anymore as a 'model' for human brain, but just as a powerful 'tool' to investigate the intelligence issue.

Embodiment theory was brought into Artificial Intelligence most notably by Rodney Brooks in the 1980s. Brooks showed that robots could be more effective if they 'thought' (planned or processed) as little as possible. The robot's intelligence is geared towards only handling the minimal amount of information necessary to

make its behavior be appropriate and/or as desired by its creator. Brooks (and others) claim that to achieve strong AI all autonomous agents need to be both embodied and situated [Brooks, 1990]. Action and perception play a critical role for cognitive development: indeed, intelligence is not handcrafted, but it develops with the agent's experience on the physical world as the ability of understanding the consequences of its own action on the external environment and on itself. Furthermore, intelligence is no longer thought to be localized just in the brain. On the contrary, it is present in the whole body of the learning agent. As a consequence, a machine without a body cannot develop human-like intelligence.

1.4 Thesis outline

This first chapter has introduced the scientific framework in which our research has been carried out. We firmly believe that taking inspiration from biological systems in the design of robot body and brain, taking in particular consideration the role of actions in cognitive development, could trace a viable route toward the realization of artificial intelligent systems. Therefore, in chapter 2 we provide some details about learning in biological and artificial systems, trying to highlight the main suggestions that robotics can take from biology:

- learning from interaction (*embodiment*);
- balance between a-priori knowledge and acquired knowledge;
- on-line learning.

Then, in chapter 3, the humanoid robot we used as experimental platform is presented and described in details, both concerning its hardware and software architecture.
The following three chapters present the experimental results, that demonstrate the feasibility of our approach to learning in humanoid robots. In particular, chapter 4 discusses the on-line learning of the tendon-driven neck controller, chapter 5 describes the autonomous attainment of reaching behavior, and chapter 6 proposes possible ways to interact with the external environment exploiting force and tactile sensing.

Finally, in chapter 7 we draw our conclusions, illustrating possible ways to enhance and extend the present work. Technical details about some mathematical and software tools used in this thesis are reported in the Appendix.

2

Learning in biology and robotics

An organism cannot develop without some built-in ability. If all abilities are built in, however, then the organism does not develop either. There is an optimal level for how much phylogeny should provide and how much should be acquired during the life time. Most of our early abilities have some kind of built-in base. It shows up in the morphology of the body, the design of the sensori-motor system, and in the basic abilities to perceive and conceive the world. One of the greatest challenges of development is to find out what those core abilities are and how they interact with development in building basic skills. Anyway, although all our basic behaviors are deeply rooted in phylogeny, they would be of little use if they did not develop. Core abilities are not fixed and rigid mechanisms but are there to facilitate development and the flexible adaptation to many different environments. Development is the result of a process with two foci, one in the central nervous system and one in the subjects dynamic interactions with the environment. The brain undoubtedly has its own dynamics that makes neurons proliferate, migrate and differentiate in certain ways and at certain times. However, the emerging action capabilities are also crucially shaped by the subjects interactions with the environment. Without such interaction there would be no functional brain. Perception, cognition and motivation develop at the interface between neural processes and actions. They are a function of both these things and arise from the dynamic interaction between the brain, the body and the outside world.

2.1 Sensori-motor learning

Piaget put forward that human beings go through several distinct stages of cognitive development [Piaget, 1954]. Each stage involves the acquisition of new skills and rest upon the successful completion of the preceding one.
The first stage is the sensorimotor (0-2 years). Until about four months of age, the infant can not differentiate itself from the environment. Gradually the child learns to distinguish people from objects and understands that both have an existence independent of their immediate perception. This stage draws its name, sensorimotor, from the fact that the child learns mainly by touching , manipulating and physically exploring the environment. By the end of this stage the child understands that its environment has distinctive and stable properties.
At the core of human ability to deal with different tasks and contexts during everyday life is sensorimotor learning. Indeed, some simple species exist that actually do not show any motor learning: the need for motor learning arises in species in which the organism's environment, body or task change. Specifically when such changes are unpredictable, a static model cannot account for them, and so flexibility in the control process is crucial. Similarly, as body size and proportions change with development, significant changes in the controller are required.
The study of motor control is fundamentally concerned with the relationship between sensory signals and motor commands. The mapping between them is bidirectional. To specify the direction under consideration a definition is generally adopted in which forward indicates the causal direction from motor commands into their sensory consequences and inverse indicates the opposite direction (i.e. transforming a desired sensory consequence into the motor commands that would achieve it).
The transformations from motor commands to their sensory consequences and vice versa are determined by the physics of the environment, musculoskeletal system, neural conduction and processing, and the sensory receptors. However, these physical transformations may also be represented internally within the central nervous system and the expression internal model is used to distinguish the actual transformation and its consequent representation in the nervous system.

Evidence shows that internal models exploited by the human brain are updated when changes in the perceived environment or in the body morphology occur. Furthermore, it has been shown experimentally that distinct models exist for kinematic and dynamic mappings, which are learned and updated separately [Krakauer et al., 1999].

2.2 Biological development

Motor development happens in a predictable sequence of events for most children, but each child varies in age when each skill is mastered. For example, although most children begin to walk independently around twelve to fourteen months, some children are walking as early as nine months. Further, children differ in terms of the length of time it takes to develop certain motor skills, such as the baby who sits up, virtually skips crawling, and begins walking. Most children develop from head to toe, or cephalocaudal. Initially, the head is disproportionately larger than the other parts of the infant's body. The cephalocaudal theory states that muscular control develops from the head downward: first the neck, then the upper body and the arms, then the lower trunk and the legs. Motor development from birth to six months of age includes initial head and neck control, then hand movements and eye-hand coordination, followed by preliminary upper body control. The subsequent six months of life include important stages in learning to control the trunk, arms, and legs for skills such as sitting, crawling, standing, and walking.
Hereinafter we further discuss the development of the motor skills which are more relevant in the context of our work.

2.2.1 Neck control and gazing

Babies are born with very little ability to control their head and neck muscles. This crucial skill, which is the foundation for all later movements such as gazing at objects, reaching for them, sitting up and walking, is developed in increments during the first six months of life.
The ability to raise the head in the prone position is seen as one of the first

major motor achievements of the newborn. Given the dimension of the head in relation to the body at birth, it is no mean skill to raise the head, as it has a large proportion of body weight and length at birth. Indeed, although 90% of neonates at two weeks of age can lift their head from a horizontal surface, it is not until they are three months of age that they can fully extend their neck while lying in a prone position [Frankenburg et al., 1992]. Following this achievement, the infant soon develops the ability to raise the head and chest, eventuqally pushing up on extended arms. At three months of age, the infant can usually maintain the head erect and upright while being held in a sitting or standing position. A more difficult task is raising the head while supine, a feature that is generally achieved by five months of age. As the head contains two sensory systems essential for balance control, namely the vestibular system and the visual system, head control is especially important in the development of postural control [Bertenthal, 2001]. One issue that has received considerable attention is the ability of young infants to track target with the eyes and the role that head movements play in this. Although neonates can track object at birth, they tend to use eye movements initially and turn their head only after their eye movements have reached the perifery [Bloch and Carchon, 1992]. By one month of age, head movements play a significant role in object tracking [Bertenthal and von Hofsten, 1998].

2.2.2 Reaching

At birth, a human infant can neither reach nor grasp. The onset of functional reaching depends on the acquisition of several abilities: differentiated control of the arm and hand, improved postural control, precise perception of depth through binocular disparity, perception of motion, control of smooth eye tracking, development of muscles strong enough to control reaching movements, and a motivation to reach. Anyway, babies as early as one week of age will attempt small arm movements directed toward objects, and are capable of orienting towards and tracking a moving object by rotating both head and eyes, although their heads may wobble considerably (Trevarthen, 1980). These early arm movements occur unpredictably, but they are not the result of random activity or pure reflex actions (i.e. they are goal directed). While the arm movements of newborns are

characterized by a rather fluid inter-joint pattern, reach and grasp motions of two- and three-month old infants reveal either short swiping motions or relatively long lasting jerky movements. These movements appear to be pre-programmed, "ballistic" motions, because trajectory correction is absent [Bower and Broughton, 1970]. That is, in early ontogenesis the role of visual information seems to be restricted to triggering the movement, rather than to visually guiding the hand toward the target. Along the same line is a result by Clifton and coworkers, who found that babies between 6 and 25 weeks of age did not rely on vision of the hand when attempting to reach for an object [Clifton et al., 1993c]. Babies contacted glowing objects in darkness, when vision of the hand was restricted, at the same rate as during normal daylight conditions, when they were able to see their hands. The first successful goal-directed reaches of human infants appear around the age of 4 to 5 months [Konczak et al., 1995; Thelen et al., 1993; Von Hofsten, 1991]. When young infants attempt their first reaches, their movements are jerky and look ataxic. In contrast to the stereotypic kinematic patterns seen in adults, infant hand paths do not follow a straight line, nor do the corresponding velocity profiles reveal a bell-shaped form [Hofsten, 1979; Konczak et al., 1995; Mathew and Cook, 1986]. Within the first 4 to 8 weeks after the onset of goal-directed reaching, kinematic improvements are dramatic (see figure 2.1). About 3 months after the onset of reaching, infants reach consistently for objects in their surround and rarely miss their target. By the same time, infants reveal improvements in their manipulative skills (i.e. precision grip). Kinematically, their hand paths become straighter, but more important, they now show signs of external force exploitation. For example, they learn that gravity and motion-dependent forces alone can extend their forearms. Consequently, they do not have to initiate elbow extension through muscular activation, but let gravity do the work [Konczak et al., 1979].

2.2.3 Hints from biology

One major difference between biological and traditional AI systems is that a biological system does not come "tabula rasa". In a wide variety of different species,

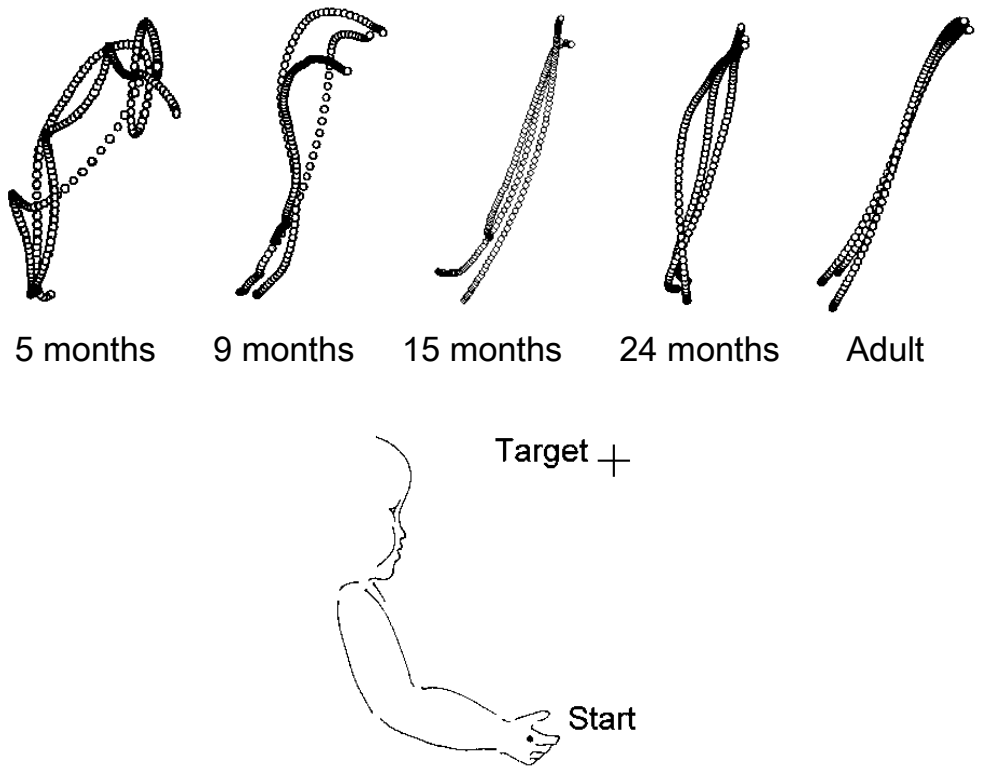


Figure 2.1: Evolution of reaching trajectories in infants. The picture shows the progression towards a stable kinematic pattern and the straightening of the trajectory. Trajectories are projected onto a vertical plane: the starting point is on the bottom left corner of the image, and the stationary target is toward the upper right corner (adapted from [Konczak and Dichgans, 1997]).

one can observe stereotyped inborn movement sequences that are clearly unlearned. Ethologists have argued for a long time that many behaviors, especially those of lower animals, cannot be explained on the basis of sensori-motor learning alone [Eibl-Eibesfeld, 1970; Gould, 1982]. Newborn human infants already possess a repertoire of coordinated movements. For example, they can perform a series of complex multi-joint bilateral movements (e.g. kicking, grasping, etc) and have available a set of so-called primitive patterns that are triggered by a sensory stimulus. Yet these motor primitives may also serve a second function. They help to build up a relationship between vision and proprioception. For example, during pre-reaching the presence of the Asymmetric Tonic Neck Reflex (ATNR, see figure 2.2) plays a crucial role in allowing babies to see their hand and in increasing visual fixation of the hands [Bushnell, 1981; White et al., 1964]. Besides this aspect, the whole structural organization of reaching movements in

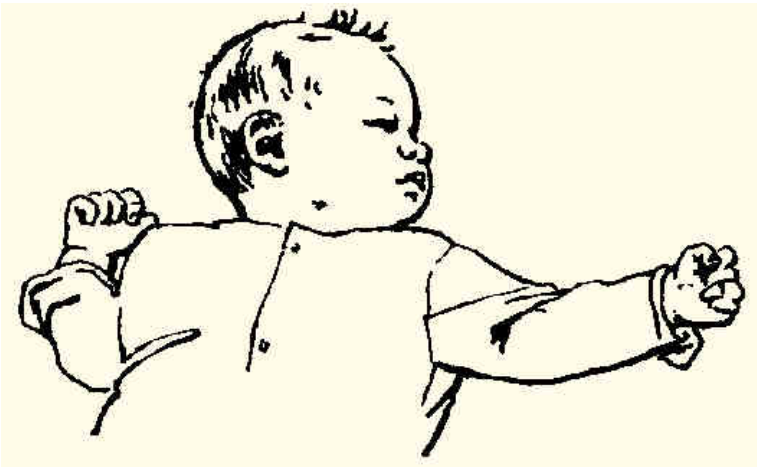


Figure 2.2: The Asymmetric Tonic Neck Reflex. The stimulus for the ATNR is the turning motion of the head. This head turn triggers a complex bilateral synergy. The infant's arm is extended to the side the infant is looking, effectively bringing the hand into the field of view. The contralateral arm is flexed as part of crossed extensor reflex spanning both homologous limbs. Thus, this multi-muscle synergy, coupling arm and head movements, provides an effective mean of linking visual and proprioceptive maps. Note the typical "fencer" position with one arm flexed and the other arm extended. The ATNR can be elicited up to the 4th postnatal month.

humans provides interesting suggestion for the realization of robotic reaching, which have been taking into account in our system (see section 5). Indeed, the required fixation of the target object before the action starts and the combination of open-loop (ballistic) and closed-loop (corrective) control, have turned out to be practical solutions for several reasons which will be better explained later.
Another interesting feature of biological systems is that they are organized in a modular structure, characterized by different interconnected subsystems, rather than being a unique complex system. This has inspired the design of the software architecture which is at the basis of robot behaviors. Furthermore, the development of one subsystem can rely on a previously developed one: for example, successful reaching relies on the acquisition of neck control, and the effect of gravity on arm motion is learned during reaching attempts.

2.3 Learning in humanoids

The complexity of the kinematic and dynamic structure of humanoid robots make conventional analytical approaches to control increasingly unsuitable for such systems. Learning techniques offer an interesting way to aid controller design if insufficient analytical knowledge is available, and seem even mandatory when humanoid systems are supposed to become completely autonomous and gain knowledge through their own experience.
While recent research in neural networks and statistical learning has focused mostly on learning from finite data sets without stringent constraints on computational efficiency, learning for humanoid robots requires a different setting, characterized by the need for real-time learning performance from a potentially infinite stream of incoming data.
Among the characteristics of the motor learning problems in humanoid robots are high dimensional input spaces with potentially redundant and irrelevant dimensions, nonstationary input and output distributions, essentially infinite training data sets with no representative validation sets, and the need for continuous incremental learning. Then, most learning tasks fall into the domain of regression problems, as in learning internal models.

Interestingly, recent developments in statistical learning provide intriguing solutions to this class of learning problems, considering the characteristics above. Bayesian inference [Bishop, 1995] is often computationally too expensive for real-time application as it requires representation of the complete joint probability densities of the data. The framework of structural risk minimization [Vapnik, 1995], the most advanced in form of Support Vector Machines, excels in classification and finite batch learning problems, but has yet to show compelling performance in regression and incremental learning. Conversely, techniques from nonparametric regression, in particular the methods of locally weighted learning [Schaal and Atkenson, 1998], have recently advanced to meet all the requirements of real-time incremental learning, also in high-dimensional spaces [Schaal et al., 2000].

2.3.1 Exploration and exploitation

Any machine-learning problem is characterized by the presence of three elements: the system under investigation, a set of data providing information about it (typically, input and output samples) and the learning algorithm. The latter is thought to work on the data in order to describe the system behavior.
A major difference of robotic learning from this classical paradigm is that the robot is not given any data describing the system. On the contrary, it should gather this information autonomously from the environment, exploiting its own action and perception capabilities. In other words, the robot must actively explore the learning space. To do so, it also need a motivation (i.e. a goal) and a sufficient degree of autonomy which allows him to perform the interaction with the external environment. Furthermore, if we consider a growing artificial system, its ability to interact with the external world is clearly subject to its level of development; on the other hand, development is the result of the interaction with the environment, which allows the robot to collect sensory data and use them for incremental learning (presented this way, can seem like chasing one's tail!). To solve the problem, a smart balance between *exploration* and *exploitation* is needed.
Past works [Demers and Kreutz-delgado, 1992; Natale et al., 2007; Rougeaux and

Kuniyoshi, 1998] tackled the problem by considering two distinct phases: an exploration phase in which the robot collects data about the system, typically by using some simple handcrafted controller driving its motion, followed by a later exploitation phase in which the robot performs its actions using a neural network which has been trained with the previously gathered data. This approach runs in at least a couple of problems. First, at some point the robot (i.e. learning system) should decide to stop learning and activate the controller that uses the network. Besides the problem of individuating this special time (how can the robot say that it has learned enough?), such a learned model would be no longer useful if the system changes. Moreover, even if the robot could decide that it has learned enough, this process would probably take too much time, especially if we want the learning space to be explored exhaustively (i.e. the curse of dimensionality [Bellman, 1956]). It is clear, in particular for biological systems, that learning must take place in a reasonable amount of time. Then, two different controllers are needed to drive the robot actions in the two phases, as the goal changes (i.e. during the exploration phase the goal is just to gather data for learning). This is neither intuitive nor practical. Indeed, the robot should explore the learning space driven by the same motivations that characterize its later actions (i.e. *goal-directed exploration*). This also helps to reduce the size of such a space, by individuating the areas which are more important to the task fulfillment, at least at the current developmental stage. Furthermore, if we have a look at what happens in humans, converging evidences show that even the primitive neonatal behaviors are goal-directed actions rather than reflexes [Van der Meer, 1997; Von Hofsten, 1982]. In our work we tried to merge exploration and exploitation in a unique behavior.

In order to let the robot start exploring the learning space, when not even one training sample has been collected (i.e. the robot does not know anything about the system yet), an initialization of the learning structures is required. This raises the question of a-priori knowledge.

On one side, some a-priori knowledge can be a robust basis on which a learning strategy can be built, reducing the learning space and providing a viable trace for the initial exploration steps. On the other hand, putting too much knowledge in the system could somehow limit its performances, introducing hard constraints

that cannot be loosened; such constraints can be particularly dangerous of they come from an incomplete or partially wrong model (which could be the case for complex system whose modeling is difficult). To avoid this possible drawback, we must be sure that learning will suppress this initial structures, which should not affect negatively the system behavior after a reasonable level of development has been reached.
Making a biological analogy, we can see a-priori knowledge as the "innate" knowledge that the individuals inherit from their own species phylogeny, while sensory-based learning can be the artificial equivalent of biological ontogeny. Thus, a-priori knowledge could be any non-learned component of the system (e.g., the structure of a controller), opposite to what is learned from experience (e.g. the estimated function used by a controller). Moreover, biological systems do not come as a monolithic structure, rather they have a modular structure where different parts develop one on top of the other. In this sense, a-priori knowledge can be represented as a module that is already working initially and provides training signals for a developing one.

2.3.2 Offline VS Online

Typically, machine-learning algorithms can be defined as either *offline* (or *batch*) or *online*. This refers to the way by which training data are evaluated. Batch algorithms require the whole training set to learn a model; the model can be used only after learning. On the contrary, online algorithms update incrementally: each new training sample generates a change in the learned model. Both techniques can be useful to model *black box* systems (characterized by the absence of any knowledge about their behavior), or in general to estimate complex systems, which are tipically hard to be described with analitical models. In particular, offline algorithms can work well in situations in which all the data is available from the beginning, and we do not expect the system to change in the future. For this reason, they do not allow flexible and adaptive control. Conversely, autonomous robots must cope with unexpected changes both in the environment and, possibly, in their own structure.
Furthrmore, the training set may not have a dimension known beforehand: data

gathering should be continuous since changes in the system are reflected in the collected data, being them the only source of information for the robot.
In most of the present work we thus use online learning algorithms. In section 6.1 we provide a detailed comparison between some offline learning techniques, applied to a specific regression problem, showing also the pros and cons against model based estimation; however, the results obtained are not exclusive for offline learning, and preliminary data concerning online estimation are reported as well.

2.4 Reaching in humanoids

Current realizations of reaching and grasping behaviors on humanoid robots mostly rely on a-priori knowledge about the system (at least, a kinematic model of the robot) and about the environment (reconstructed 3D model of the scene, a-priori knowledge of the object properties). These settings allow to plan reaching trajectories and grasping actions efficiently, even if we consider a whole humanoid, which can employ more degrees of freedom to fulfill the reaching task (e.g. bending its knees or waist) and must also maintain its balance [Berenson et al., 2007; Park et al., 2007; Yoshida et al., 2008]. To get rid of these requirements, the employment of a rich set of sensors, learning algorithms and smart control strategies is desirable. Visual information about the environment can be exploited using visual servoing techniques, when both the end-effector and the target that has to be reached are in the field of view [Hutchinson et al., 1996]. A kinematic model of the system, as well as other non-linear sensorimotor mappings, can be learned off-line by applying non-linear regression tools. Anyway, on-line learning and adaptation become crucial when we deal with dynamically changing environments. Furthermore, even the kinematic and dynamic properties of the robot can vary over time. The description of the entities relevant for reaching and grasping task and the design of the controller can be simplified in order to better cope with the proposed scenario. Some authors have already proposed to encode target 3D position in space using an *internal representation* [Metta et al., 1999; Rougeaux and Kuniyoshi, 1998]. Assuming that the robot is fixating the target, the head joints configuration can be used to describe uniquely the target position (instead of using, for instance, the euclidean distance with respect to an external

fixed reference frame). Furthermore, the use of a gaze-centered frame of reference for reaching is supported by converging evidence in neurophysiology [McIntyre et al., 1989; Soechting and Flanders, 1989], even in the case of whole-body reaching [Flanders et al., 1999]. After having fixated the target, the appropriate arm configuration which brings the end-effector (i.e. robot hand) in the fixation point must be recovered: the motion toward this configuration is generally known as *ballistic reaching* (no visual feedback is required). As previously discussed, this could be done by exploiting a kinematic model of the robot. If such a model is not available, or it is too difficult to be computed precisely, it can be learned through experience. In previous works this learning has been performed either off-line using previously collected sensory measurements [Natale et al., 2007] or on-line during a training phase, separated from the subsequent execution phase [Rougeaux and Kuniyoshi, 1998]. Other solutions allow the on-line updating of this learned map during action execution [Marjanovic et al., 1996; Metta et al., 1999], while [Gaskett and Cheng, 2003] propose also a way to further correct position errors after the *ballistic reaching* using visual feedback, as was originally suggested in [Blackburn and Nguyen, 1994]. The biological plausibility of this reaching strategy (open loop + closed loop) is supported by results in developmental psychology [Clifton et al., 1993a,b; Konczak, 2004; Von Hofsten, 1982] which show that children first develop the open-loop (ballistic) controller, and then, after six/eight months, start correcting for hand position errors using visual closed loop control. These considerations can be extended to the situation in which more DOFs are employed, and preshaping and grasping skills are included in the system. The work presented in [Laschi et al., 2006] can be of inspiration for the realization of a scheme which joins together reaching, preshaping and grasping in a unique action.

Humans are very good at generalizing concepts and skills. After they have learned how to reach for visually identified objects, they can rapidly generalize to the task of reaching with a tool. Following recent neuroscience results [Maravita and Iriki, 2004], it seems that specific neural networks in the human brain hold an updated map of body shape and posture. When reaching for an object with a tool, our motor capability is extended due to changes in these neural structures: the inclusion of a tool in the 'body schema' can be seen as if our own end-effector (e.g.

the hand) was moved to the tip of the tool. A humanoid robot which is able to learn and continuously update its body map can deal with such a situation.

3

A developing robot

The work presented in this manuscript has been carried out using the humanoid robot James [Jamone et al., 2006], depicted in figure 3.1. In the following sections we give an overview of the main characteristics of the robot, in terms of its hardware and software (*bodyware* and *mindware*).

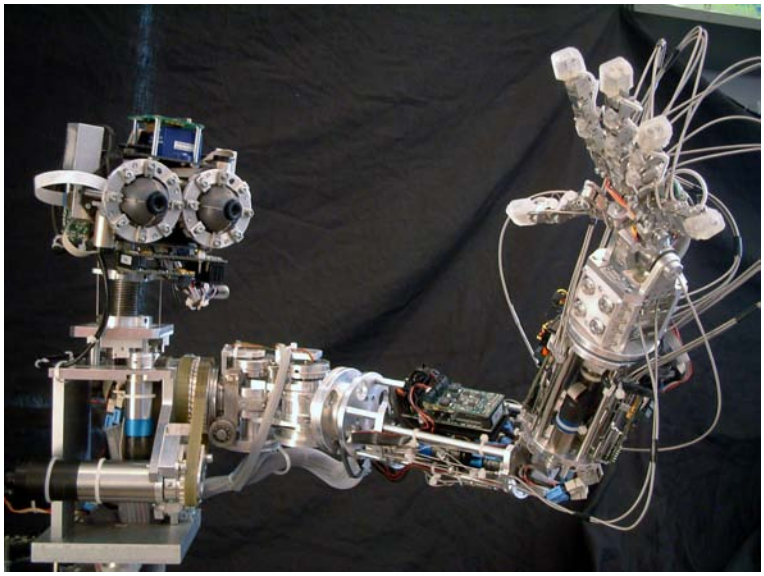


Figure 3.1: The humanoid robot James.

3.1 Design principles

As already stated, in the framework of embodied cognition we consider action and perception as a fundamental and unique tool for robot learning and development; the latter are considered to be the result of the continuous interaction with the environment and not the outcome of abstract reasoning. To achieve them, *anthropomorphism*, *compliance* and *sensorization* are of crucial importance. These three basic aspects have been taken into account while designing the humanoid robot James.

Clearly, the realization of an artificial system capable of more complex movements involves a series of technological improvements, especially if we are interested in replicating both kinematic and dynamic aspects. Recently, there has been a growing interest in developing robots whose geometric and actuation structures resemble those of a human beings. Probably, the most extreme steps in this direction are represented by Cronos [Holland and Knight, 2006], the robot recently developed by O. Holland et al., and Kotaro [Mizuuchi et al., 2006], developed at Tokyo university.

The control of these innovative architectures is a complex task, especially if compared with classical designs [Guenter et al., 2005; Kim et al., 2004] which have been usually based on rotational joints in serial configuration. Remarkably, the control complexity increases with the architecture complexity and suggests new interesting control problems. The more complex the system the more difficult to build an accurate analytical model to describe and control it; under these conditions learning and multisensori integration become a very appealing solution.

Realizing such complex architecture on a real humanoid robot is clearly an ambitious goal. Nevertheless, we believe in the importance of developing and implementing innovative and human-like actuation schemes within the field of humanoid robotics. Indeed, the advancements in robot design may be seen as the artificial counterpart of the physical adaptation all species undergo across evolution.

Actions require the existence of a physical body, which must act in the same environment as humans, sharing the same physical constraints and dealing with the same daily objects. Most of our "world" has been shaped to suit humans; for

this reason robots with an anthropomorphic structure are more likely to be able to operate successfully.
This consideration should affect both the robot joints arrangement and the choice of actuators. Concerning the actuators, electric motors are widely used in robotics for several reasons (easy control, high efficiency, long life, little mainteinance, little acoustic noise, etc...); anyway, the high reduction gears employed to increase their torque output make the system very stiff. Such stiffness is a big disadvantage in practical operations, especially if we are considering an evolving system standing at the initial developmental stages, when the controllers, still immature, produce substantial errors. Indeed, learning from experience requires the robot to actively explore its working environment. Compliance in the actuation helps this exploration to be safe both for the external environment and for the robot itself: we will show how this feature has been achieved in our robotic platform by including elastic components (e.g. steel tendons, plastic belts, springs) in the torque transmission.
Moreover, the necessity of understanding the consequences of actions requires rich perceptual capabilities, i.e. the ability of retrieving information about the world (exteroception) and about the body (proprioception). As in biological systems, intelligence should not be placed just in the brain, but also in the body itself. As an example, the intrinsic soft compliance of the skin allows humans to successfully grasp an object without the need to place the fingers exactly at the required position around it; this consideration has been taken into account while designing James tactile sensory system.

3.2 General architecture

James is a 22-DOF torso with moving eyes and neck, an arm and a highly anthropomorphic hand. The head is equipped with two eyes, which can pan and tilt independently (4 DOFs), and is mounted on a 3-DOF neck, which allows the movement of the head as needed in the 3D rotational space. The arm has 7 DOFs: three of them are located in the shoulder, one in the elbow and three in the wrist. The hand has five fingers. Each of them has three joints (flexion/extension of the distal, middle and proximal phalanxes). Two additional

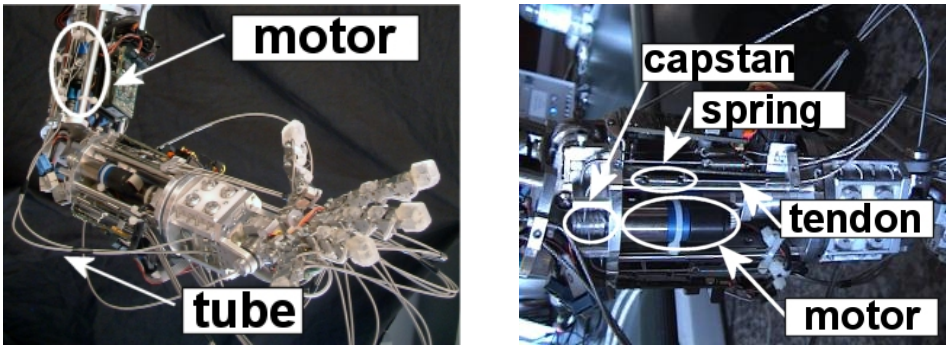


Figure 3.2: On the left, a picture of the tendon actuation; wires are routed inside flexible tubes. On the right, a picture of the tendon-spring actuation.

degrees of freedom are represented by the thumb opposition and by the coordinated abduction/adduction of four fingers (index, middle, ring, and little finger). Therefore, the hand has a total of 17 joints.

The overall size of James is that of a ten-year-old kid, with the appropriate proportions for a total weight of about 8 kg: 2 kg the head, 4 kg the torso and 2 kg arm and hand together (the robot parts are made of aluminum and Ergal). The robot is equipped with 23 rotary motors (Faulhaber), most of which directly drive a single DOF. Exceptions are in the neck, where three motors are employed to roll and pitch the head, and in the shoulder, where three motors are coupled to actuate three consecutive rotations. Transmission of the torques generated by the motors is obtained through plastic toothed belts and stainless-steel tendons; this solution is particularly useful in designing the hand since it allows locating most of the hand actuators in the wrist and forearm rather than in the hand itself, where strict size and weight constraints are present. Furthermore, tendon actuation gives a noteworthy compliance to the system[1]. Indeed, the elasticity of the transmission prevents damages to occur, especially in presence of undesired and unexpected conditions, which are usual in the "real" world: small positional errors when the hand is very close to the target, obstacles in the path of the desired movements, strokes inflicted by external agents. Extra intrinsic compliance has been added by means of springs in series with the tendons to further help the

[1]Stainless-steel tendons are extremely elastic especially when the applied tension exceeds a certain threshold.

coupling/decoupling of the fingers (see Figure 3.2).
Low-level motor control is distributed on 8 programmable DSP-cards, embedded in the robot arm, head and torso. Communication between these cards and an external cluster of PCs is achieved by the use of a CANBUS connection. Robot movement is controlled by the user with positional/velocity commands which are processed by the DSPs which generate trajectories appropriately, using standard PID control.

3.2.1 The head

Each eye is actuated independently by two motors, with a tendon-based transmission which closely resembles human eye muscles arrangement, achieving pan and tilt rotations (figure 3.3). The movements of the eyes are coupled in the low-level controller in order to obtain direct control of common pan, θ_{cp}, common tilt, θ_{ct}, and vergence, θ_v. Tilt divergence, θ_{tv}, is controlled during the initial robot calibration and set to zero.

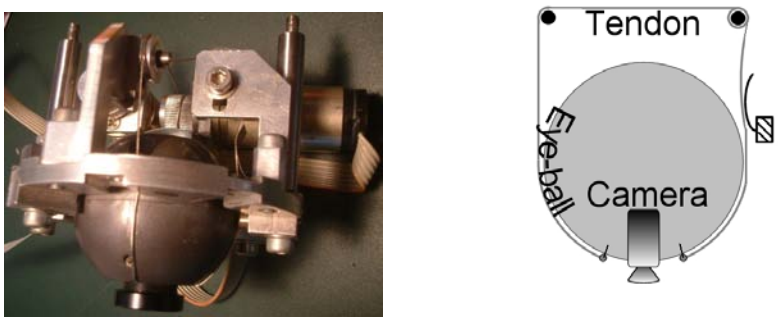


Figure 3.3: The left picture shows one of the eyes and the actuation system. The two tendons are actuated by two motors. The first motor moves the vertical tendon (tilt motion). The second motor moves the horizontal tendon (pan motion). The right figure sketches the actuation scheme, concerning just the pan rotation.

The neck bone is constituted by a steel spring, which holds the head giving it the possibility of bending forward (pitch) and laterally (roll). The actuation of these two degrees of freedom is obtained with a peculiar structure, recalling the design of a tendon-driven parallel manipulator; recent studies on this kind of actuation systems can be found in [Hay and Snyman, 2005; Verhoeven and Hiller, 2000].

It has been shown in [Tsai, 1999] that a tendon driven system with open-ended tendons requires more tendons than DOFs to be fully controllable. In particular, James neck is actuated by three motors ($\mathbf{q} \in \mathbb{R}^3$) in order to perform pitch and roll movements ($\mathbf{x} \in \mathbb{R}^2$). Mechanical constraints prevents the arbitrary choice of $\mathbf{q}$ for a desired $\mathbf{x}_d$. In particular, due to the elasticity of the structure, it exists a set of motor positions $\bar{\mathbf{q}} \subseteq \mathbb{R}^3$ for a given desired orientation $\mathbf{x}_d$. Among the possible $\mathbf{q} \in \bar{\mathbf{q}}$ there is an ideal motor configuration $\mathbf{q}^\star$ which generates an optimal value of stress of the three tendons, while achieving the main positioning task. Values of stress that are too small can send the tendons out of the capstans, while too large values can misalign the spring spirals or break the tendons.
This novel mechanical structure gives an high degree of anthropomorphism to the system. Along the same line, Albers et al. [Albers et al., 2003] have focused their attention on an innovative robotic neck, named *Vertebral Neck*, highly inspired by the human neck structure. Noticeably, the control of these innovative architectures is a complex task, especially if compared to classical designs [Guenter et al., 2005; Kim et al., 2004] which have been usually based on rotational joints in serial configuration. Within this context Terzopoulos and Lee [S.H.Lee and D.Terzopoulos, 2006] have proposed an interesting solution to control an extremely detailed and precise biomechanical model of the human head-neck system. Though limited to a simulation environment, their work is to our knowledge one of the few considering the problem of controlling an highly realistic (muscle-actuated) model of the neck.
Specifically, James neck is surrounded by three steel tendons, separated 120 deg apart. The tendons length determines the configuration of the spring and therefore, the pitch and roll orientation of the head. The length of the tendons is adjusted by means of three motors, positioned at the base of the neck (see Figure 3.5). This special geometry allows the neck to show ranges of motion comparable to the human ones; as claimed by Clarkson [Clarkson, 2000], average ranges of motion for pitch and roll rotations in adults are around ± 45 deg, while James motion has been bounded by software in the range of ± 40 deg (safe limit, lower than hardware limit). Furthermore, this peculiar structure, even if far from being a close reproduction of the human musculoskeletal system, presents some analogies with the arrangement of some human neck muscles (see Figure 3.4). Humans

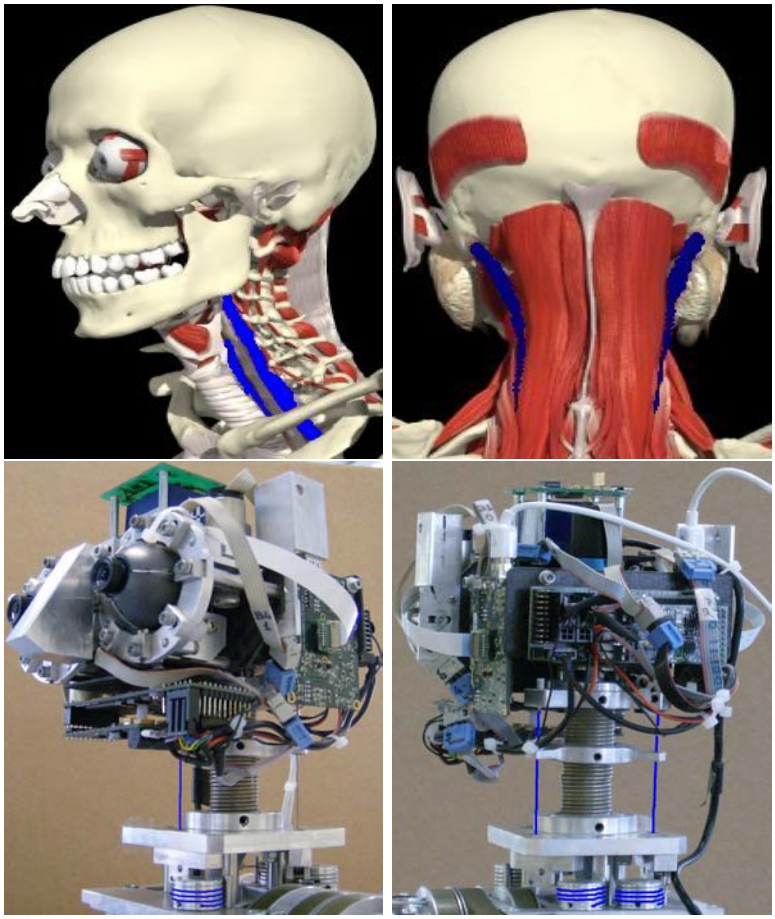


Figure 3.4: From the left. Human neck muscles we took inspiration from in the system design, highlighted in blue: *Longus Colli* in the first image, and *Longissimus Capitis* in the second image (images taken from *Primal 3D Anatomy* software [Kendall et al., 2005]). The third and fourth images show how the tendons arrangement in our robot mimics the human anatomy.

are provided with more than 20 types of muscles in the neck, with several units for each type, settled into different layers: their work is both to control the orientation of the head respect to the neck base and to give stability to the cervical spine in supporting the weight of the head [Kendall et al., 2005]. In our system, the spring tone is sufficient to support the head, and the tendons are employed just to move the head along its roll and pitch axes.
If we consider the deeper muscular layer around the neck, we can find a couple of long anterior muscles (*Longus Colli*, see top left image in Figure 3.4) that are responsible for head flexion, and a couple of long posterior muscles (*Longissimus Capitis*, see top right image in Figure 3.4) that are responsible for head extension. Both muscles are also involved in the lateral flexion of the neck, even if this movement is mostly actuated by other muscles (*Scalene Muscles*). These muscles completely surrounds the cervical spine, having their origins, roughly speaking, at the level of the scapula, and their insertions at the level of the atlas; the anterior tendon in our system (bottom left image in Figure 3.4) can work as the *Longus Colli*, while the two tendons in the back (bottom right image in Figure 3.4) can emulate the *Longissimus Capitis*.
Details about the coordinated control of these three motors in order to achieve desired pitch and roll rotations are discussed in section 4. Then, on top of the spring a fourth independent motor is mounted, directly actuating a third degree of freedom, the head yaw (θ_y, the rotation around an axis parallel to the pan axes of the two eyes).

3.2.2 The arm

The design of the tendon driven shoulder was developed with the main intent of allowing a wide range of movements. The current design, consists of three successive rotations corresponding to pitch, yaw and roll respectively (see Figure 3.7 for details). The three motors that actuate these rotations are located in the torso. This design is evidently non-standard. Standard manipulators (e.g. the Unimate Puma) have the shoulder motors in a serial configuration, with a single motor directly actuating a single degree of freedom. In the Puma arm, a pure pitch/yaw/roll rotation can be obtained by simply moving one motor and

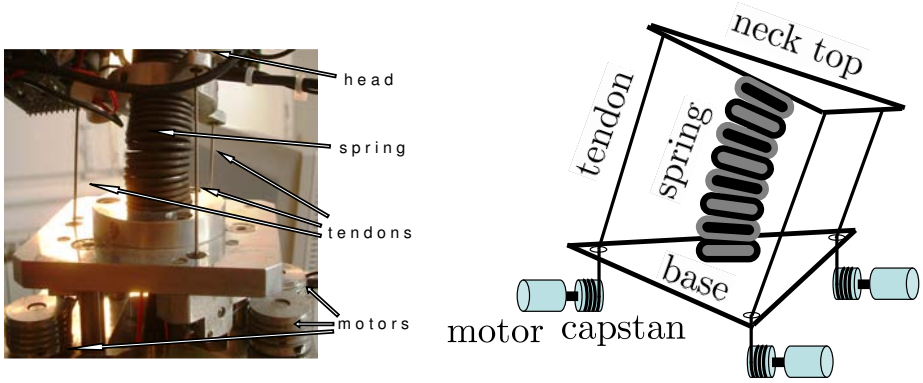


Figure 3.5: Neck actuation system and sketch of the neck kinematics. Each motor pulls a tendon which passes trough a hole in the neck base. In this way the effective tendon length can be reduced to bend the spring in different directions.

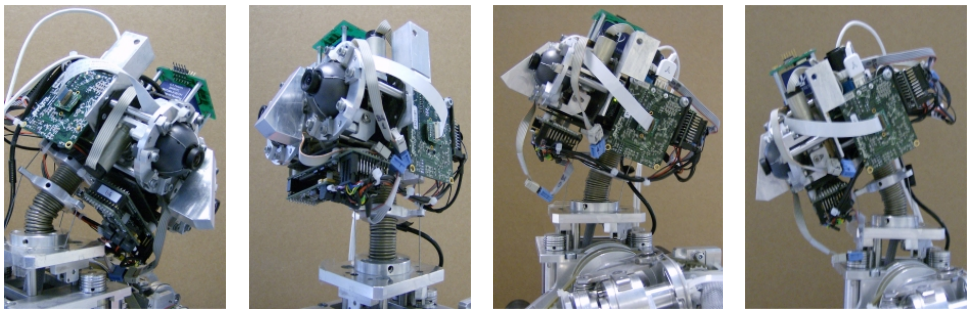


Figure 3.6: James head. Different neck configurations seen from different views. Roll movements: left pictures. Pitch movements: right pictures.

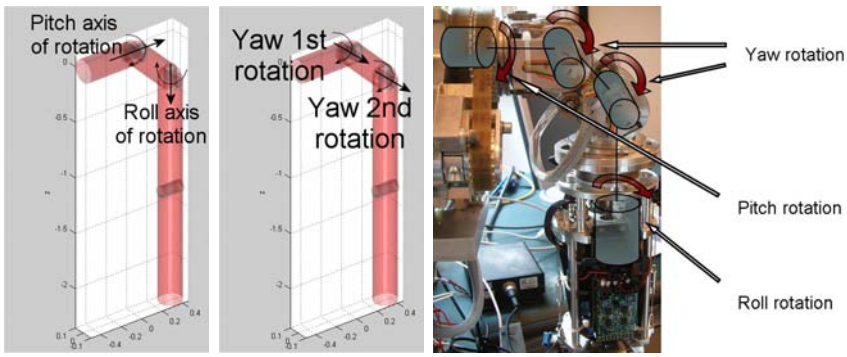


Figure 3.7: The picture shows the three degrees of freedom of the shoulder. Notice in particular how the yaw rotation is obtained by a double rotation around two parallel axes.

keeping the others fixed. On James instead, shoulder rotations are the result of the coordinated movement of the three motors. The motor positions (θ_1, θ_2, θ_3) are related to the pitch, yaw and roll rotations (θ_{sp}, θ_{sy}, θ_{sr}) by a lower triangular matrix:

$$\begin{bmatrix} \theta_{sp} \\ \theta_{sy} \\ \theta_{sr} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}.$$

Actuation is achieved by exploiting the design of tendons and pulleys. Special care has been taken in designing the abduction (yaw) movement. The abduction rotation is divided along two mechanically coupled joints. The two joints correspond to a sequence of two rotations around two parallel axes. A mechanical tight coupling forces the two angles of rotation to be equal. This 'ad hoc' solution allows the arm to gain an impressive range of movement (pitch$\simeq 360^o$, yaw$\geq 180^o$, roll$\simeq 180^o$) and to perform special tasks (e.g. position the hand behind the head) at the expense of a multiplication of the required torque by a similar amount.
The elbow flexion/extension, θ_e, is directly actuated by a single motor, located in the upper arm. Again, torque transmission is achieved by stainless steel tendons. The wrist is provided with 3 DOFs, namely pitch, roll and yaw (θ_{wp}, θ_{wr}, θ_{wy}), which correspond respectively to wrist flexion/estension, adduction/abduction and rotation. Each DOF is actuated by a dedicated motor: two motors placed in the upper arm actuate pitch and roll DOFs through tendon based transmission, while a third motor mounted inside the wrist directly actuates the yaw DOF.

3.2.3 The hand

The main constraint in designing the hand is represented by the number of motors which can be embedded in the arm, assuming there is very little space in the hand itself. The current solution uses 8 motors to actuate the hand. Note that the motors are insufficient to independently drive every joint singularly. After extensive studies, we ended up with a solution based on coupling some of the joints with springs. This solution does not prevent movement of the coupled joints when one of them is blocked (because of an obstacle) and therefore results in an intrinsic compliance. Details of the actuation for each finger are given in the following.

- Thumb. The flexion/extension of the proximal and middle phalanxes is actuated by a single motor. However, the movement is not tightly coupled. The actuation system allows movement of one phalanx if the other is blocked. Two more motors are used to directly actuate the distal phalanx and the opposition movement.

- Index finger. The flexion/extension of the proximal phalanx is actuated by a single motor. One more motor is used to move the two more distal phalanxes.

- Middle, ring and little fingers. A unique motor is used to move the distal phalanxes of the three fingers (flexion/extension). Once again, blocking one of the fingers does not prevent the others from moving. One additional motor actuates the proximal phalanxes and, similarly, the movement transfer to the others when one of the fingers is blocked.

Finally, an additional motor (for a total of 8 motors) is used to actuate the abduction/adduction of four fingers (index, middle, ring and little fingers). In this case, fingers are tightly coupled and blocking one of the fingers blocks the others.

3.3 Sensors

The importance of having a lot of sensors in the context of this research has been already stressed in the previous sections. Indeed, James is equipped with a rich set of sensors, either commercial products or devices specifically realized for this platform, which enables the robot to exploit visual, proprioceptive, kinesthetic, vestibular, tactile and force sensing.
Vision is provided by two digital CCD cameras (PointGrey Dragonfly remote head), located in the eyeballs, with the controlling electronics mounted inside the head and connected via FireWire to the external PCs; images are acquired at 15 fps framerate.
The proprioceptive and kinesthetic senses are achieved through position sensors.

Besides the magnetic incremental encoders connected to all motors, absolute-position sensors have been mounted on the shoulder to ease calibration, in the fingers (in every phalanx, for a total of 15) and in the two motors that drive the abduction of fingers and of the thumb. In particular, the hand position sensors have been designed and built expressly for this robot, combining magnets and Hall effect sensors. In order to understand the sensor's structure and functioning, consider one of the flexion joints of the fingers, between a pair of links (phalanxes). One metal ring holds a set of magnets and moves with the more distal link (of the two we consider here). The Hall-effect sensor is mounted on the first link. Magnets are laid as to generate a specific magnetic field sensed by the Hall-effect sensor, monotonically increasing when the finger moves from a fully-bended position to a fully-extended one: from the sensor output, the information about flexion angle is derived.
A 3-axis orientation tracker (Intersense iCube2) has been mounted on top of the head, to emulate the vestibular system. The tracker, which gives an absolute measure, is used both during calibration, to calibrate the motors actuating the three tendons of the neck, and during online control. The tracker is basically a gyroscope, measuring angular accelerations, velocities and position respect to rotations around the three cartesian directions X-Y-Z; for this reason, we will refer to it also as inertial sensor.
Tactile information is extracted from sensors which have been specifically designed and developed for James, and have been realized using a two-part silicone elastomer (Sylgard 186), a Miniature Ratiometric Linear Hall Effect Sensor (Honeywell, mod. SS495A) and a little cylindrical magnet (grade N35; dim. 2x1.5 mm). The sensor structure is shown in Figure 3.8: any external pressure on the silicone surface causes a movement of the magnet and therefore a change in the magnetic field sensed by the Hall-effect sensor, which measures indirectly the normal component of the force which has generated the pressure. An *air gap* between the magnet and the Hall-effect sensor increases the sensor sensitivity while keeping the structure robust enough. Two kinds of sensors have been realized, with a different geometric structure depending on the mounting location: *phalangeal-sensors* for some of the phalanxes and *fingertip-sensors* for the distal ends of each finger (Figure 3.9). The former present one sensing element,

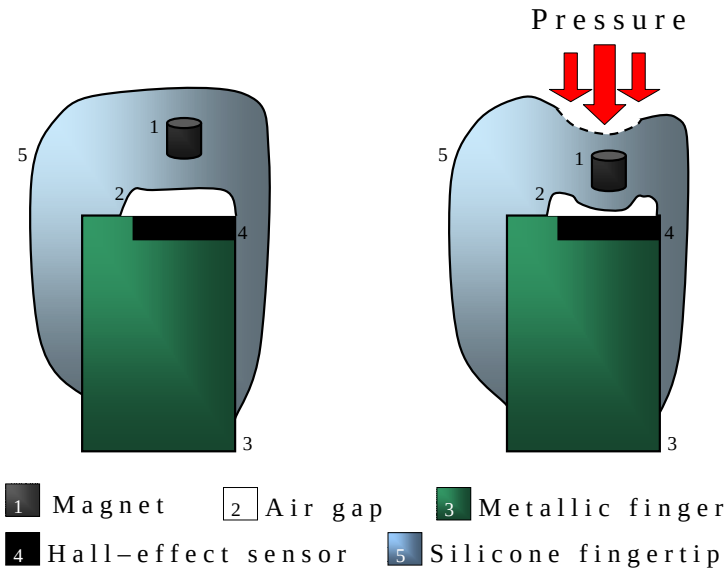


Figure 3.8: Description of the sensing method.

which can measure only a limited range of forces, even if with a high sensitivity. In practice, the response of the phalangeal sensors can be regarded as an on/off response. The fingertip-sensors, instead, have two sensing elements capable of sensing a larger range of stimuli. The *fingertip-sensor* characteristic curve

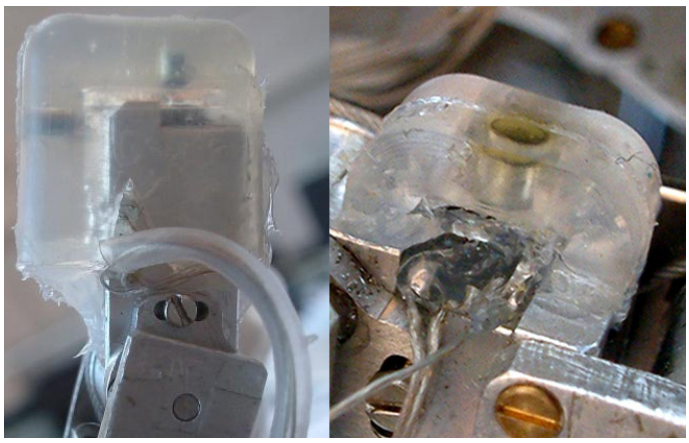


Figure 3.9: On the left, the *fingertip-sensor*, with its two sensing elements. On the right, the *phalangeal-sensor*, with a unique sensing element.

is reported in Figure 3.10, where the thicker line is the mean value over series of several measurements and the vertical bars are the standard deviation. The fact that the standard deviation is reasonably low is due to the robustness of the mechanical structure, which is robust to pressure applied at different positions and from different directions, even if with substantially different intensities. The

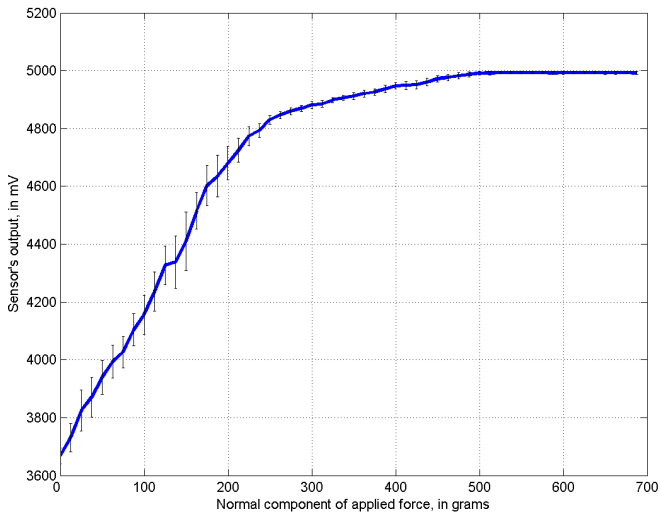


Figure 3.10: *Fingertip-sensor*'s characteristic curve.

non-linear response of the sensor is a feature because of its similarity with the log-shaped response curve observed in humans, whose sensitivity decreases with the stimulus intensity [Wolfe et al., 2005]. Furthermore, the minimum force intensity detected by the sensor (less than 10 grams for *fingertip-sensors* and about 1-2 grams for *phalangeal-sensors*) is very low, an aspect which has been pursued as the main feature of the device during the design and development of its mechanical structure. Finally, the use of the soft silicone shows an intrinsic compliance and increases the friction cone of the grasping forces. Silicone adapts its shape to that of touched object without being subject to plastic deformations, easing the interaction with unknown environments.

12 tactile sensors have been mounted on James hand: 5 *fingertip-sensors*, one for each finger, and 7 *phalangeal-sensors*, two on the thumb, ring finger and middle finger, and one on the index finger, as shown in Figure 3.11. Since each *fingertip-*

sensor presents two sensing elements, the overall number of tactile elements on the hand is 17. All the 32 Hall-effect sensors of the hand (employed in tactile and

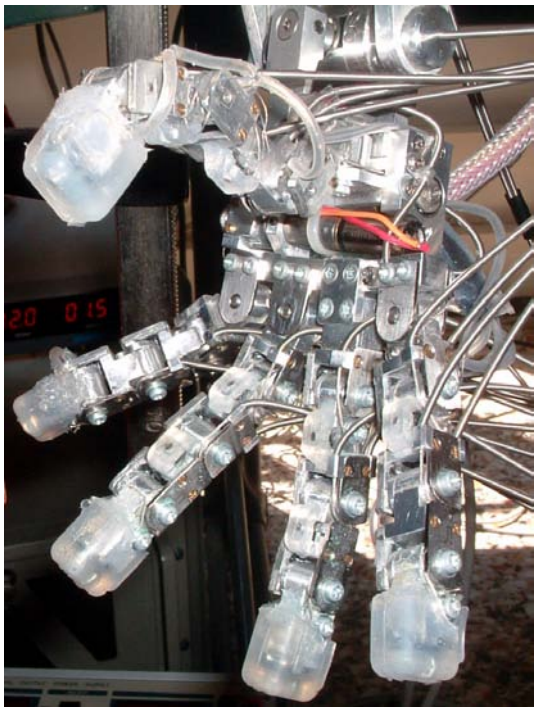


Figure 3.11: James's antropomorphic hand equipped with the 12 tactile sensors.

proprioceptive sensing) are connected to an acquisition board, mounted on the hand back and interfaced to the CAN-BUS as for the DSP-based control boards. The acquisition card is based on a PIC18F448 microcontroller, an ADC and multiplexer, and a 40-cable connector, which holds the 32 signals and provides the 5 Volt DC power supply to the sensors. To wire such a considerable number of devices in such a little space, with a reasonable resiliency (required because of the constant interaction of the hand with the external environment), we have chosen a very thin stainless-steel cable, coated in Teflon, with a 0.23 mm external diameter. Moreover, in order to further increase system robustness, cables are grouped into silicone catheters along their route between different sensors and toward the acquisition card. Finally, special connectors have been realized to interface the Hall-effect sensors and the wires endings: they are soldered on the wires and plugged in the sensors. These connectors have a double function: to

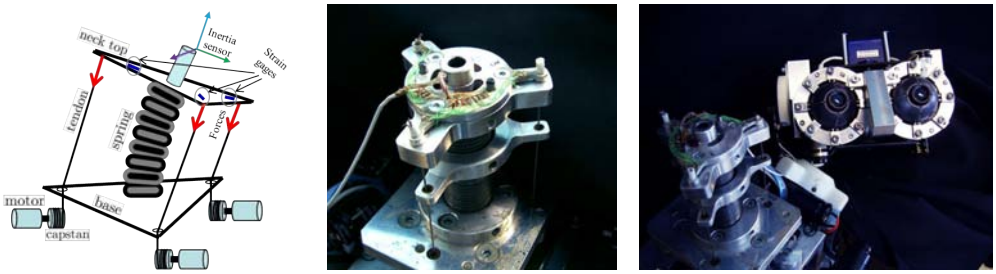


Figure 3.12: On the left, sketch of the neck actuation: three force sensors measure the tendon forces using strain gages (SG). Then, two views of the neck force sensor: from the top (center) and near the disassembled head (right).

ease mainteinance (if a sensor is broken can be unplugged, changed and plugged in again), and to work as 'weak points' during robot actions (if a cable is pulled too much due to a bump, the connector just unplug, without any risk of damage for the sensor or the wire).

The robot can also exploit force sensing through two different sensors: a force sensor we specially design for the neck, and a commercial 6-axis force/torque sensor which has been embedded in the robot arm.

The neck force sensor [Fumagalli et al., 2009] measures the forces that the actuation system transmits to the neck through the tendons (see figure 3.12). This sensor is positioned on the upper 'vertebra' of the neck. For each tendon, force is measured indirectly by measuring the deformation of a cantilever beam structure; this quantity is proportional to the tendon tension, which is the measure we generally want to control ($F \in \mathbb{R}^3$). Semiconductor strain gages (SSGs) are employed in a Wheatstone bridge configuration. A mechanical stop limit is exploited to avoid the risk of damaging the sensing structure. A custom made board provides for the force sensor data acquisition. The board is based on a 16 bit DSP from Microchip (dsPIC30F4013) which samples up to a maximum of 6 analog channels for strain gauges sensors in a bridge configuration. In our specific case, the sampling rate is 1 kHz. At the top of the upper arm, just below the shoulder, a single 6-axis F/T sensor (ATI mini45 [ATI, 1982]) is placed (see figure 3.13). The sensor is based on silicone strain gages and is able to measure the three components of the force applied to itself (as well as the three torque components). The particular placement of the sensor enables the robot to detect

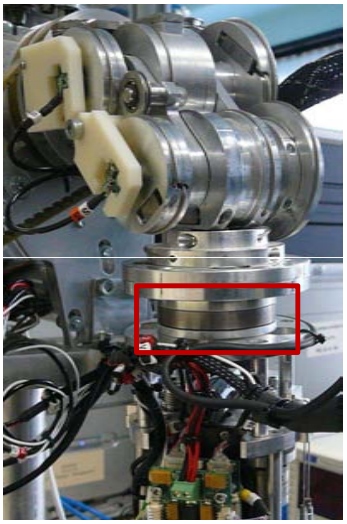


Figure 3.13: Detail of James arm. The ATI Mini45 F/T Sensor (inside the red square) is placed just below the shoulder.

both internal forces due to arm motion and external forces due to contacts between the arm/hand and the environment. Furthermore, this solution has been chosen also because most of the space in the upper and forearm is occupied by the motors actuating the wrist, elbow and fingers, and by the DSP boards used to control them. In section 6.1 we will discuss the retrieval of the external forces and the consequent need to estimate the internal ones.

3.4 Software framework

A cluster of standard PCs (Intel Core2 Duo @2.00GHz) and a Blade system (Primergy RX200 server with 6 additional blades, Intel Xeon @2.00GHz) are interconnected through a 1GB ethernet and constitute the core of the brain of James. These machines are dedicated to the high-level software, which is more computationally demanding (e.g. coordinated control, visual processing, learning), while the low-level motor control is implemented on the DSPs embedded in the robot body. All this software has been written using YARP [Metta et al.,

2006]. YARP (Yet Another Robot Platform) is an open-source software framework that supports distributed computation under different operative systems (Windows, Linux) with the main goal of achieving efficient robot control. YARP facilitates code reuse and modularity by decoupling the programs from the specific hardware (using *Device Drivers*) and operative system (relying on the *OS wrapper* given by ACE [Schmidt, 2003; Schmidt and Huston, 2002]) and by providing an intuitive and powerful way to handle inter-process communication (using *Ports* objects, which follows the Observer pattern [Gamma et al., 1995]). Furthermore, YARP provides mathematical (vectors and matrices operations) and image processing (basic *Image* class supporting IPL and OpenCV) libraries.
This software platform helped us to construct James brain as a collection of interconnected independent modules, running on different machines and exchanging data and control signals. The first reason to do so is a practical one: one single CPU, although powerful, can never be enough to cope with more and more demanding applications. Then, smaller subsystem are easier to be maintained and updated, and their employment makes the overall system cleaner. Moreover, when general enough, a single module can be connected to multiple modules, and reused in different contexts.
YARP has been used also to develop a dynamic simulator (namely, *jamesSimulator*) for the robot, useful to test some of the behaviors we then implemented on the real robot. To realize it, we mainly reuse the open source code developed by Tikhanoff et al. in [Tikhanoff et al., 2008], a simulator for the iCub robot. Indeed, iCub shares a lot of functional and technical similarities with James.
The main modifications to the iCub simulator concerned the joints exact number and arrangement, kinematic and dynamic parameters and some joint coupling. Then, models of hand tactile sensors and arm F/T sensor has been included in the system.
We carried out simulation mainly concerning learning of the reaching controller and estimation of the arm internal forces. Nevertheless, the experimental results and the discussion we will provide about these two problems (respectively in sections 5 and 6.1) are limited to real data.

3.5 Software architecture

We provide here an overview of the robot software architecture, explaining which parts of the system are hard-coded, what is learned through experience and how development can take place. In general, we refer to development meaning the sequence of learning steps taken by the robot to achieve more and more sophisticated motion and sensing abilities. Conversely, we call learning the acquisition of a particular skill, i.e. the establishment of a sensori-motor mapping. James acquires gazing and reaching skills autonomously, actively exploring its motor space and learning online from the sensory measurements it gathers. More precisely, the structures of the different controllers and some intialization values are given (i.e. a-priori knowledge), while the non-linear functions on which the controllers rely are estimated from sensory data (i.e. acquired knowledge).

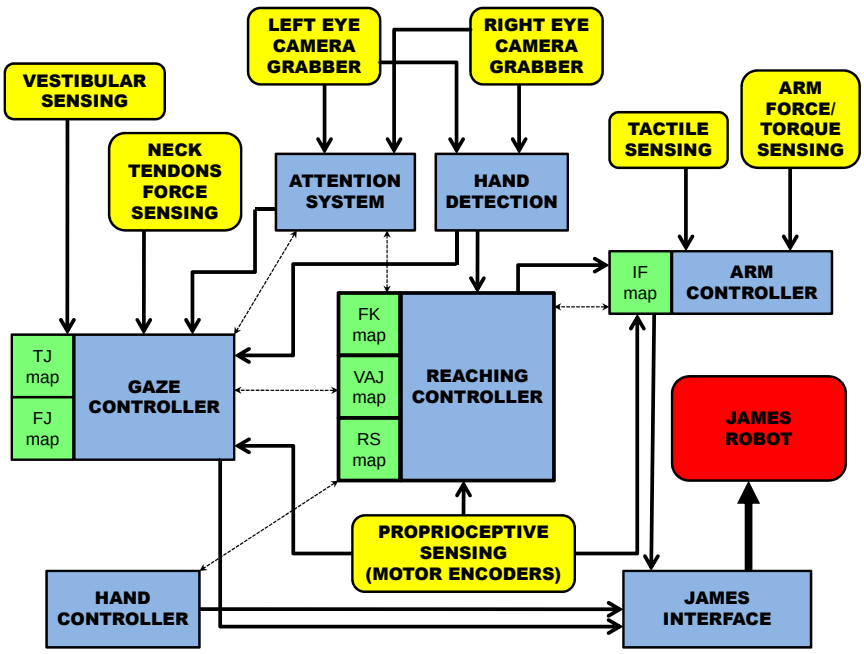


Figure 3.14: James software architecture. Yellow boxes represent sensory systems, blue boxes are software modules (in green, the learned mappings used by such modules) and the red box represents the physical robot.

Figure 3.14 depicts the robot software architecture, highlighting its sensory equipment (yellow boxes) and the software modules which control its behavior (blue boxes). The green boxes represent the learning structures exploited by each controller, which are trained online during control. Black thick connections (arrows) indicate the flow of data between the modules, while the thin dashed ones indicate mutual synchronization. The red box represents the physical robot (or, more precisely, the control boards embedded in its body).
James is bootstrapped with a repertoire of very basic motion and sensing capabilities which constitutes the basis for learning and development. The robot can understand the nature of the perceived sensory signals (e.g. understand that a signal from a particular encoder indicate the position of the corresponding motor), and can control position and velocity of its motors. Technically, this is implemented on the control boards embedded in the robot structure, which are interconnected through a CAN-BUS connection. The PC which is directly interfaced with the CAN-BUS runs a program (namely, *jamesInterface*) to communicate with these motor control boards.
In particular, concerning arm motion, there is another level of interface (namely, *armController*); this module receives position and velocity commands for the arm, in joints space, and sends them to the *jamesInterface*. Inside the module, position and velocity commands are merged together in a unique framework, based on potential force field [Khatib, 1986].
When a position command is received, the desired arm configuration becomes an attractive pole of the force field. As a consequence, velocity commands are generated following equation 3.1 and sent to the *jamesInterface* in order to move the arm towards the desired joints configuration.

$$\dot{\mathbf{q}}_{arm}(t) = -K(\mathbf{q}^t_{arm} - \mathbf{q}_{arm}(t)) \tag{3.1}$$

As the velocity depends on the distance from target $\mathbf{q}^t_{arm}$ to current $\mathbf{q}_{arm}$ arm configuration, to avoid jerky motion at the movement onset and achieve *bell-shaped* velocity profiles, a smoothing function is applied to the controller in the initial part of the movement.
When a velocity command is received, joints velocities are directly sent to the

jamesInterface; if a previous position command has not been completed yet, the corresponding attractive pole is removed.
During the motion all arm sensors are checked cyclically. Motor encoders are used to assess movement completion. Other sensori stimuli can trigger different behaviors. In particular, if external F/T are detected or the hand tactile sensors are stimulated, the arm motion stops, and different obstacle avoidance strategies can be adopted. To detect the external F/T from sensor measurements, an estimation of the internal F/T is required (*IF map*). Details are discussed in chapter 6.
Concerning the *reachingController*, a rudimental form of eye-hand coordination, inspired by the Asymetric Tonic Neck Reflex (ATNR) shown by newborns [Van der Meer et al., 1995], is present in the system at birth: a simple look-up-table takes as input the head configuration $\mathbf{q}_{head}$ and gives as output one arm configuration among four (up-left, bottom-left, up-right, bottom-right), in order to bring the hand roughly in the field of view. The employment of this look-up-table mimics the role of primitive reflexes in humans [Konczak, 2005]. In fact, during the initial stages of development the robot mainly moves using this structure, which is then gradually suppressed as learning of the arm-head kinematic (*FK map*) occurs. During actions, a visuo-arm Jacobian mapping (*VAJ map*) and a reachable space map (*RS map*) are also updated and exploited. More details are given in chapter 5.
The successful completion of the reaching action would trigger an action of the hand (i.e. grasping), governed by the *handController*. Anyway, at the current state, no hand movement has been implemented on the system. Possible solutions for the realization of a grasping controller are suggested in section 7.1.1, as future work.
Simple visual processing (see appendix 8.3) is employed to determine the position of a green ball (i.e. a marker for the robot hand) in the visual field, in camera coordinates on both eyes ($< u_L, v_L >, < u_R, v_R >$). This is done by the *handDetection* module.
Head movements are driven by the *gazeController*, described in section 4.4. The controller actuates the eyes and neck to gaze toward interesting regions in the visual space, which can be provided by the *attentionSystem* module (see appendix

8.4) or by the *handDetection* module, depending if the target of the gazing action is a generic object or the robot hand. Neck orientation is achieved by exploiting two learned maps (a task Jacobian map, *TJ map*, and a force Jacobian map, *FJ map*). Chapter 4 illustrates how these maps are learned and used for control.

4

Neck control

James neck is a redundant system in which the number of actuators is higher than the mechanical degrees of freedom (see section 3.2.1). In particular, this section deals with the coordinated control of the three motors placed at the base of the neck in order to actuate pitch and roll rotations ($\mathbf{x} = [\theta_p \theta_r]^T \in \mathbb{R}^2$, 2 DOFs). Motor positions can be directly controlled through the encoder feedback, but we are interested in regulating the neck orientation exploiting the feedback of the absolute orientation sensor.

Previous works [Fumagalli et al., 2009; Nori et al., 2007] tested different solutions to this problem, all exploiting more or less accurate analytical models of the system. In these approaches errors arise from discrepancies between the model and the real system: of course, the more complex the system the more probable the errors. In particular, non-linear and time-varying parameters (e.g. due to elasticity or deformable parts) are difficult to describe. On the other hand, the absence of a model reduces the possibility to control the system. In these cases learning offers an elegant and efficient solution to the problem, especially if we can exploit a considerable amount of sensory measurements.

Therefore, we propose a strategy based on autonomous online learning to control the neck. The controller regulates the length of the tendons to fulfill a primary task (controlling the neck orientation), and at the same time it tries to keep the tension of the tendons within safe limits, to avoid damaging of the spring which supports the head and of the tendons themselves. To achieve the primary task, a kinematic model is needed to map from the desired velocities in the task (absolute

orientation) space to the desired velocities in the motor space (i.e. we will refer to this model as the *task Jacobian*). To control the tension of the tendons, a tension sensor is used, and a model is needed to compute the necessary motor commands which modify the tendon tensions (i.e. we will refer to this model as the *force Jacobian*). Both models are approximated by specific neural networks, which are trained online and used for control. Most of the content of this chapter has been published in [Jamone et al., 2010].

4.1 Control law

Typically, learning is performed off-line. In this case training and execution are separated phases. On the contrary, on-line learning approaches need these two phases to be somehow merged. Anyway, in both cases, an efficient strategy should be implemented to gather the necessary sensory data to train the system (*exploratory movements*, i.e. movements whose main aim is to explore the state space). As we have already pointed out in section 2.3.1, in the on-line learning scenario we would like to integrate training and execution phases in a unique and continuous behavior. To achieve this, these initial exploratory movements should be driven by the same goal that will guide the "normal" behavior after learning, employing the same rule to generate the movements.
In our implementation, we designed a control law which is used during the whole robot life, from the very beginning (*exploration*) to the end (*exploitation*). The performances of such a controller improve with time, as learning occurs.
We want to achieve the main task of canceling the difference between the head position $\mathbf{x}$ and the desired value $\mathbf{x}_d$. Furthermore, we want $\mathbf{x}$ trajectories to be linear in the operational space. As a secondary task we would like to regulate the tendon tension $F \in \mathbb{R}^3$ by minimizing $F - F_d$ with $F_d = \frac{F_{max}+F_{min}}{2}$, calling F_{max} and F_{min} the maximum and minimum acceptable tendon force values. Practically, these two objectives can be obtained by solving the following optimization problem:

$$\min_{\mathbf{q}} \frac{1}{2} \left\| F(\mathbf{q}) - \mathbf{F_d} \right\|^2 \;\; s.t. \; \mathbf{x}(\mathbf{q}) - \mathbf{x_d} = \mathbf{0} \tag{4.1}$$

Following the approach of [Samson et al., 1991], we can tackle (4.1) setting the desired motor velocities $\dot{\mathbf{q}} \in \mathbb{R}^3$ as follows:

$$\dot{\mathbf{q}} = C_1 + C_2 \tag{4.2}$$

where

$$C_1 = J^{\dagger}(\mathbf{q})(\mathbf{x}_d - \mathbf{x}) \tag{4.3}$$

$$C_2 = (I - J^{\dagger}(\mathbf{q})J(\mathbf{q}))KJ_F(\mathbf{q})^T(F_d - F) \tag{4.4}$$

where $J(\mathbf{q}) \in \mathbb{R}^{2\times 3}$ is the jacobian matrix which maps from motor velocities to task velocities, $J^{\dagger}(\mathbf{q}) \in \mathbb{R}^{3\times 2}$ is its Moore-Penrose generalized inverse [Penrose, 1955] and $J_F(\mathbf{q}) \in \mathbb{R}^{3\times 3}$ is the jacobian matrix which maps from motor velocities to tendons tension velocities (i.e. tendons tension variations). These matrices are the core of the learning controller; they are initialized with random values and then updated during the head movements, relying on sensory measurements coming from tendon tension sensor, absolute position sensor and motor encoders. The way in which these matrices are initialized and then updated is the subject of subsection 4.2. Then, $K \in \mathbb{R}^{3\times 3}$ is a positive definite diagonal gain matrix, $I \in \mathbb{R}^{3\times 3}$ is the identity matrix, $\mathbf{x_d} \in \mathbb{R}^{\mathbf{2}}$ is the vector of desired positions and $F_d \in \mathbb{R}^3$ is the vector of desired tendons tension.
The control law (4.2) is the so called resolved motion rate control technique (see [Samson et al., 1991] for details and proof of convergence) applied to our problem. The operator $(I - J^{\dagger}(\mathbf{q})J(\mathbf{q}))$ projects the term which minimize the tension error in the null space of the primary task (i.e. the neck orientation). If the measured tendons tensions are outside the admissible range $[F_{min}\ F_{max}]$, the (4.2) controller is switched off, and a 'safety controller' is activated:

$$\dot{\mathbf{q}} = -G \cdot F_{err} = C_{safe} \tag{4.5}$$

$$F_{err} = \begin{cases} (F - F_{max}) \text{ if } F > F_{max} \\ (F - F_{min}) \text{ if } F < F_{min} \end{cases} \tag{4.6}$$

where $G \in \mathbb{R}^{3\times 3}$ is a positive definite diagonal gain matrix. As soon as the measured tendon tensions have come back within the limits, the main controller

(4.2) is switched on again.
This 'safety controller' has the unique task of regulating the tendons tensions to the desired values, but of course it interferes with the main neck orientation task. We will show in section 4.3 how the activation of this controller is frequent in the beginning of learning and more and more absent in the later stages.

4.2 Learning strategy

As previously stated, movements are generated by the (4.2) controller, in combination with the safe controller (4.5) when measured tendons tensions exceed the limits. Target orientations $\mathbf{x}_d$ are provided to the robot every 20 seconds, choosen randomly within the (safe) physical limits $[-35° \; 35°]$, with uniform distribution. During the motion data are gathered from absolute position sensor, $\mathbf{x} \in \mathbb{R}^2$, force sensor (tendons tensions), $F \in \mathbb{R}^3$, and motor encoders, $\mathbf{q} \in \mathbb{R}^3$. What is needed for learning are little variations of these quantities (displacements): $\Delta\mathbf{x}$, ΔF and $\Delta\mathbf{q}$. The time window on which these variations are computed is 50 ms, while the controller rate is 5 ms (200 Hz). Here follows the description of how $J(\mathbf{q})$ is learned from these sensory measurements; the same strategy applies to the learning of $J_F(\mathbf{q})$, and could be generalized to any other non-linear matrix.
From a set of couples $(\Delta\mathbf{x}, \Delta\mathbf{q})$ we can estimate a local Jacobian matrix $\hat{J}$: $\Delta\mathbf{x} = \hat{J}\Delta\mathbf{q}$ with Least Squares (LS) Regression. This $\hat{J}$ is an approximation of $J(\mathbf{q}) : \dot{\mathbf{x}} = J(\mathbf{q})\dot{\mathbf{q}}$ in a local region of the $\mathbf{q}$ space, whose accuracy depends basically on the size and distribution of the aforementioned set. In our implementation this estimation is performed on-line, with an incremental LS algorithm adapted from [Hosoda and Asada, 1994]. Then, since we need to build a global Jacobian matrix $J(\mathbf{q})$ starting from local models $\hat{J}$, we employed a Receptive Field Weighted regression neural network (RFWR [Schaal and Atkenson, 1998], an on-line machine-learning tool) to map from $\mathbf{q}$ to the corresponding local Jacobian matrix $\hat{J}$. Every time a new couple $(\Delta\mathbf{x}, \Delta\mathbf{q})$ is computed (every 50 ms) the RFWR net is queried with the current motor configuration $\mathbf{q}$ to obtain the correspondent local model $\hat{J}$. The obtained constant matrix is updated with the couple $(\Delta\mathbf{x}, \Delta\mathbf{q})$ using incremental LS. The RFWR net is then trained with the current motor configuration $\mathbf{q}$ as input and the updated constant Jacobian matrix

$\hat{J}$ as output.
The algorithm steps are summarized below:

1. **collect new sample** $(\Delta\mathbf{x}, \Delta\mathbf{q})_i$ at time i
2. **retrieve** $\hat{J}_{\mathbf{q}_i} = RFWR(\mathbf{q}_i)$
3. **update** $\hat{J}_{\mathbf{q}_i}$ with $(\Delta\mathbf{x}, \Delta\mathbf{q})_i \rightarrow \hat{J}^{up}_{\mathbf{q}_i}$
4. **train** $RFWR(\mathbf{q}_i, \hat{J}^{up}_{\mathbf{q}_i})$

Since the incremental LS algorithm requires a non-null matrix to update, the RFWR net is initialized with an arbitrary $\hat{J}$, which becomes the output for every configuration $\mathbf{q}$ received as input (in the beginning, when the net is still empty). The same holds for $\hat{J}_F$.

4.3 Results

In this section the performances of the learning controller are evaluated and discussed. The discriminants are the steady-state orientation error (pitch and roll position errors) and the slope of its convergence to zero, the linearity of the trajectories in the operational (pitch/roll) space[1], the amount of tension measured on the tendons.
In all the tests reported hereinafter, the same sequence of 13 target positions $\mathbf{x_d}$ is provided to the robot, one different position every 20 seconds: these positions have been chosen arbitrarily to cover homogeneously the task (roll/pitch) space. As previously explained, the movements during which training data are gathered and on-line learning is performed are generated providing random $\mathbf{x}_d$ to the robot.
The first set of graphs (figures 4.1, 4.2 and 4.3) shows the best performances we achieved on the system, after about 2 hours and 20 minutes of training (170000 samples of the form $(\Delta\mathbf{q}, \Delta\mathbf{x}, \Delta F)$ gathered).
The steady-state RMSE (Root Mean Squared Error) computed over the sequence of movements is 0.020° for the roll rotation and 0.018° for the pitch rotation. Figure 4.2 also shows the exponential convergence of the position error to zero.

[1]Trajectories would be perfectly linear if the Jacobian $J(\mathbf{q})$ were exactly known.

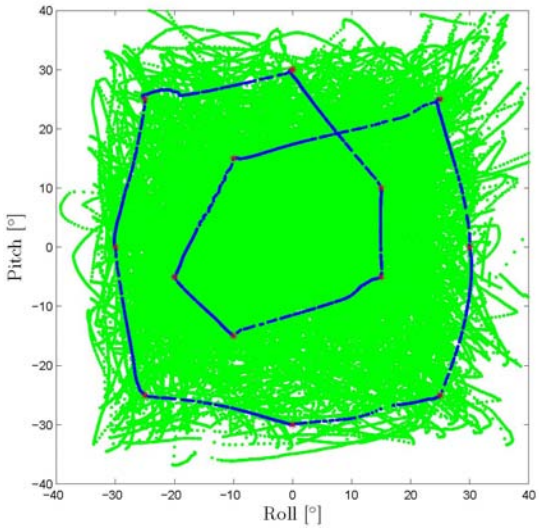


Figure 4.1: Task space trajectories (blue line) connecting the desired via points (in red). The green dots are the points spanned during learning. This performance was achieved after 2 hours and 20 minutes of training. The linearity of the trajectories proves the convergence of the estimated Jacobian.

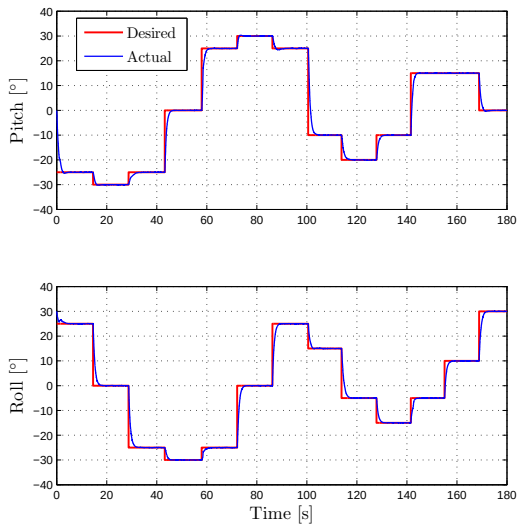


Figure 4.2: Roll and pitch step response. Desired and actual positions relative to the trajectories of figure 4.1 are shown.

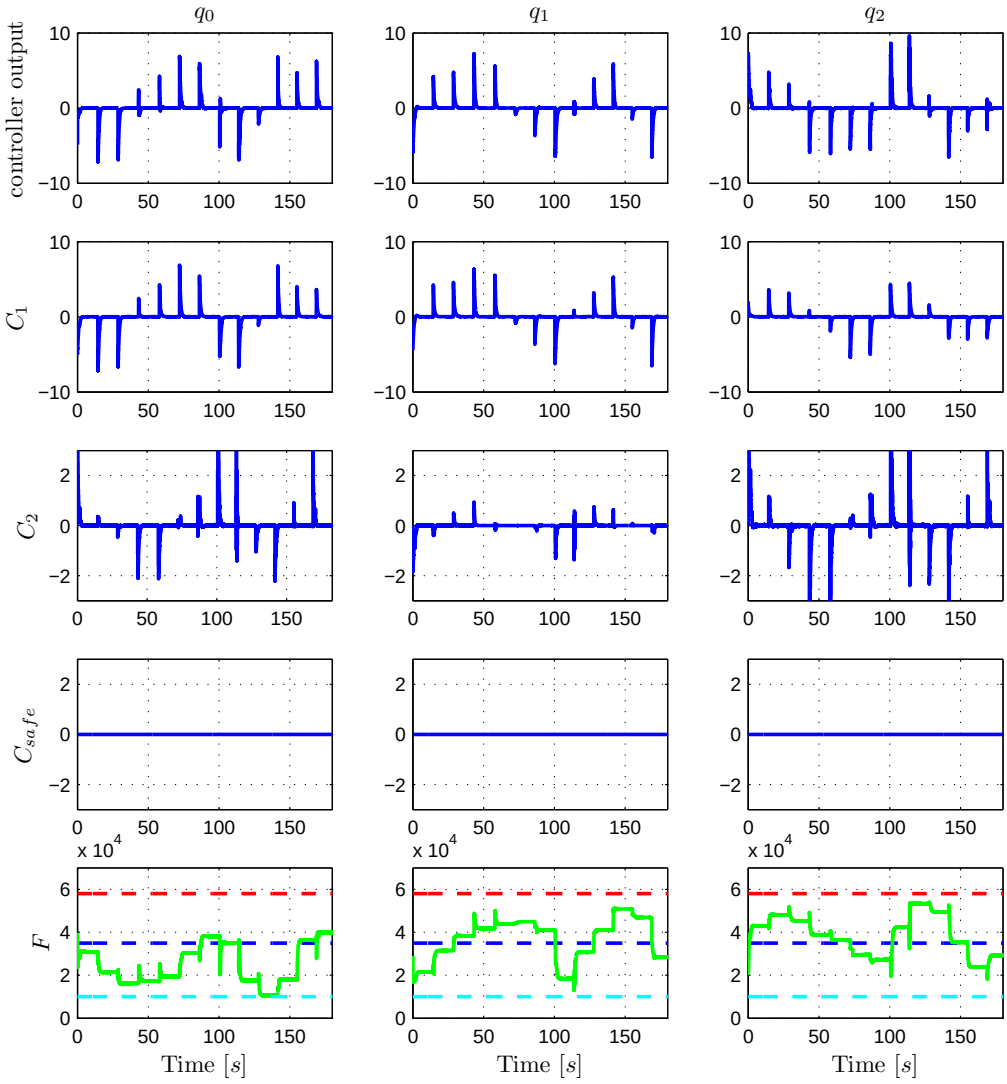


Figure 4.3: Activations of the different controllers and tension measurements, for every q. On the first row the overall controller output, on the second row the C_1 component. The third row shows the contribution of the controller C_2. When $\mathbf{x} - \mathbf{x_d} = \mathbf{0}$, The contribution of C_2 converges to zero, which proves that the first order conditions of the secondary task are satisfied. On the fourth row the 'safety controller' activation is shown, while on the fifth row tension measurements are reported. The effect of the secondary task controller C_2, guarantees that tendon tension are within the limits $F_{min} < F < F_{max}$.

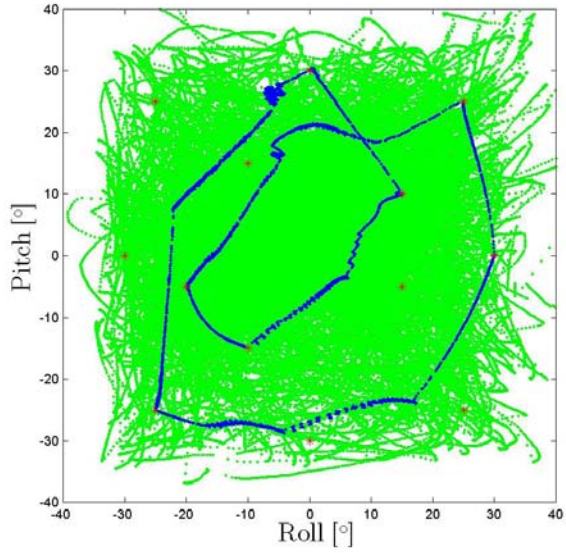


Figure 4.4: Task space trajectories without tendon tensions control. With the only controller C_1 tendon tension arise over the force limits, thus activating the controller C_{safe}. In this way, the convergence to the target positions $\mathbf{x_d}$ is not guaranteed.

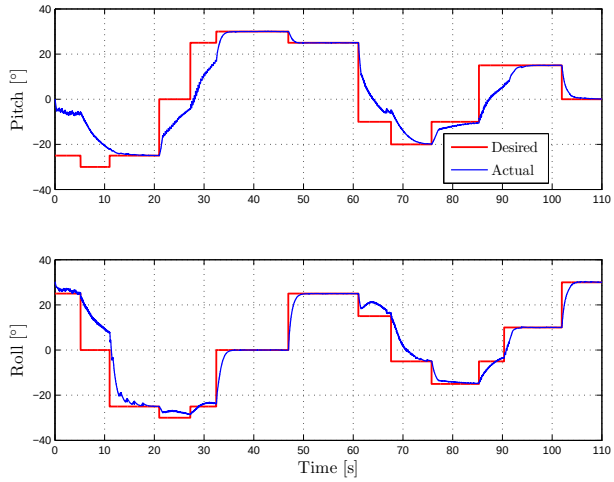


Figure 4.5: Desired and actual roll and pitch positions without the activation of tendon tensions error projection in the null space of $J((\mathbf{q}))$ (the controller C_2 is not active). The convergence to the target positions is not guaranteed, because of the activation of the controller C_{safe}.

Furthermore, the tendons tensions are kept within the limits by the controller C_2, which acts in the null-space of the primary task.
To underline the importance of the controller C_2 (4.4), we tested the system controlling just the primary task, setting $C_2 = 0$. Figures 4.4, 4.5 and 4.6 describe the behavior of the system using such a controller. We can notice from figure 4.6 that without the contribution of C_2 the 'safety controller' C_{safe} is often active; the activation of this controller affects the performances of the main controller, sometimes preventing the system from cancelling the position error. The result is that with such a controller the system is not able to consistently cancel the position error, nor to regulate tendons tensions.
Figures 4.7, 4.8 and 4.9 show the improvements of the controller during learning. In the left figures the system has been trained with just 20000 samples (about 15 minutes), and it is clear that the performances are far from being acceptable. We will not quantify the errors in this case, since they are too big. In the middle figures, on the contrary, the system is already able to reach the desired orientations $\mathbf{x}_d$, but the error convergence is not always perfect and the task space trajectories are not so linear. The steady-state RMSE computed over the sequence of movement is 0.064° for the roll rotation and 0.020° for the pitch rotation. Furthermore, the tendons tensions are not regulated properly: the 'safety controller' is activated twice, and there are some high-frequency oscillations. In the right figures the results obtained with the best controller (the one already discussed in the beginning of the section) are reported.
Finally, something can be added concerning the mere task of estimating $J(\mathbf{q})$ and $J_F(\mathbf{q})$. Figure 4.10 shows the trend of the coefficients of $J(\mathbf{q})$, where $\mathbf{q} = 0$. Remarkably, these coefficients, initialized arbitrarily as stated in subsection 4.2, seem to converge to specific values ($J(0) = [0.2; -0.1; -0.1; 0.0; 0.2; -0.2]$).
Furthermore, these values turn out to be reasonable if we analyze the structure of the neck and the considered motor configuration ($\mathbf{q} = 0$, which means also $\mathbf{x} = 0$). When the head is straight (i.e. $\mathbf{x} = 0$), to bend the neck forward the system should shorten the front tendon of a certain length (let's say L_1) and lengthen the two tendons in the back of half that length ($L_1 \cdot \sin(120°)$, being the three tendons separated 120° apart). This is consistent with the first row of $J(0)$, $[0.2; -0.1; -0.1]$. On the other hand, to bend the neck laterally the front

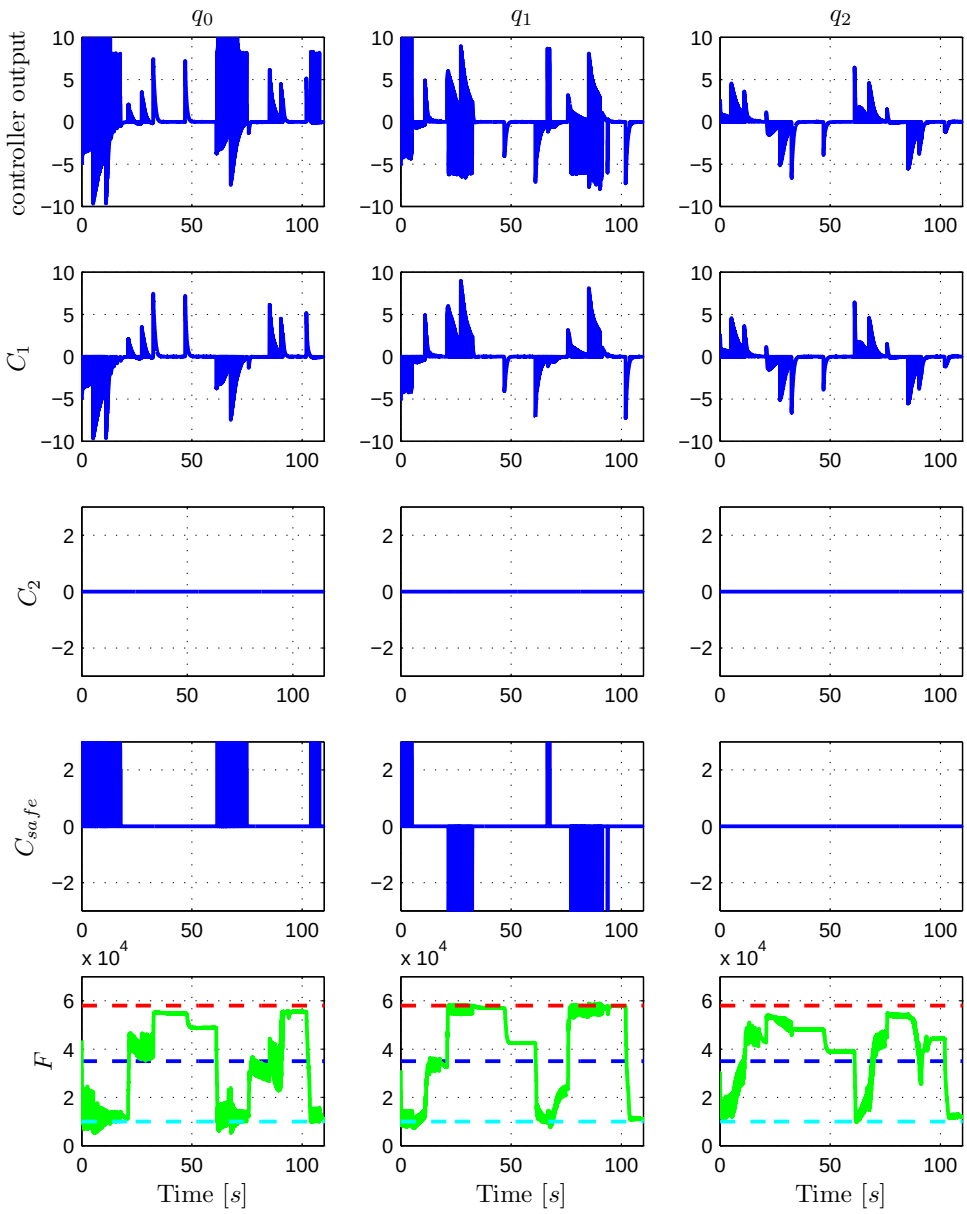


Figure 4.6: Activations of the different controllers and tension measurements, for every q, without tendon tensions control. Controller C_{safe} is often active, because the controller of the secondary task is not present.

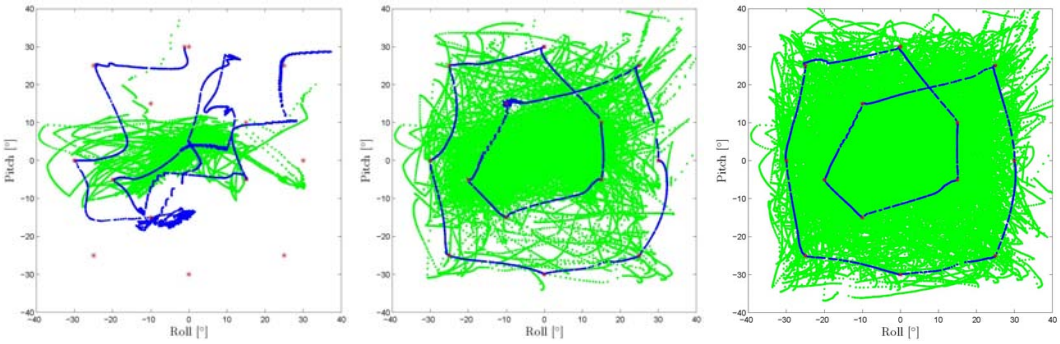


Figure 4.7: Task space trajectories at three progressive learning stages (20000, 100000, 170000 training samples, from left to right). Left figure show that most of the points are not reached because the Jacobian estimation is not accurate enough. Central figure show that the estimation of the Jacobians is converging. Right figure show that the Jacobian evaluation has converged to a good estimation of the real Jacobian, which results in linear trajectories in the Cartesian space (see figure 4.1).

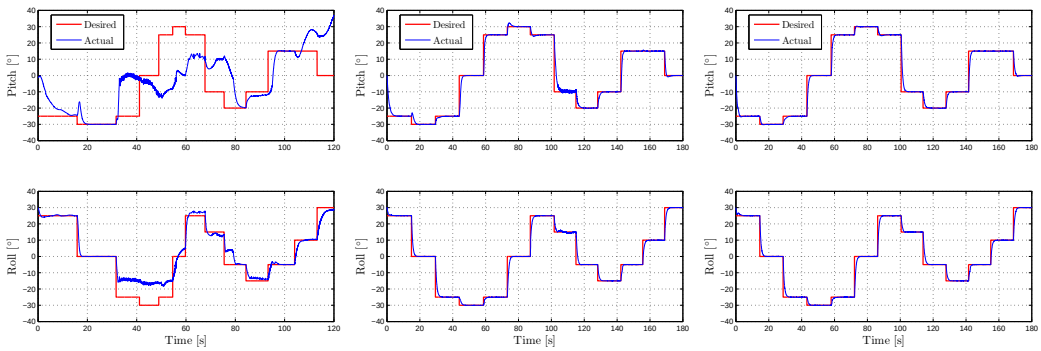


Figure 4.8: Roll and pitch desired and actual positions at three progressive learning stages (20000, 100000, 170000 training samples, from top to bottom). Top figures show that most of the target positions are not reached because the Jacobian estimation is not accurate enough. Central figures show that the estimation of the Jacobians is converging. Steady state errors are limited. Bottom figures show that the Jacobian evaluation has converged to a good estimation of the real Jacobian. Here target positions $\mathbf{x_d}$ are reached.

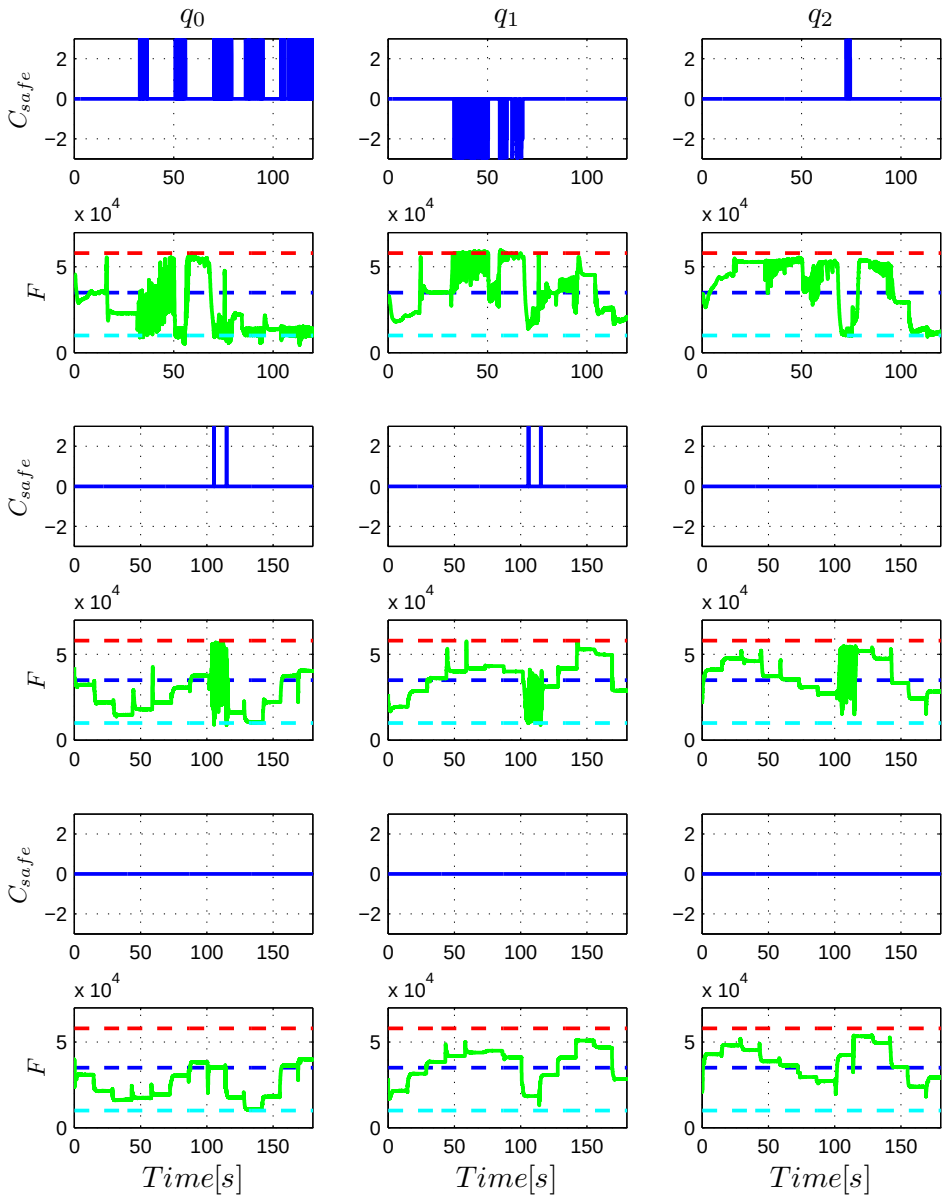


Figure 4.9: Activations of the 'safety controller' and measured tendons tensions at three progressive learning stages (20000, 100000, 170000 training samples, from top to bottom). The first two rows show that when learned Jacobian is not accurate, forces arise over the limits, thus activating the 'safety controller' C_{safe}. The more learning improves the estimation, the more the 'safety controller' becomes unnecessary.

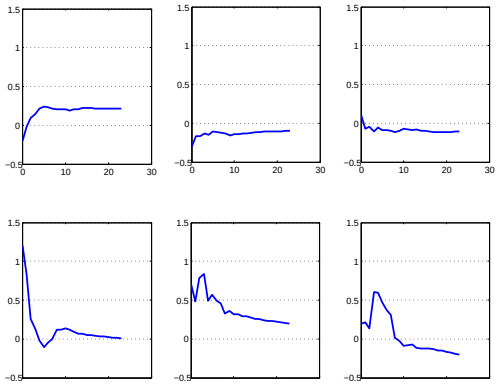


Figure 4.10: Trend of the coefficients of $J(\mathbf{q})$ in $\mathbf{q} = 0$ during learning. They converge to particular values that turned out to be quite reasonable.

tendon do not play any role, while the other two tendons must be displaced of the same quantity, but in opposite direction; this is consistent with the second row of $J(0)$, $[0.0; 0.2; -0.2]$. For a more precise description about this geometric model refer to [Fumagalli et al., 2009].

Regarding $J_F(\mathbf{q})$, some additional experiments have been performed in order to test the quality of its estimation.

First the system has been driven to a desired position $\mathbf{x}_d$ using the controller C_1 (4.3) alone; the absence of the C_2 contribution causes the raising of the tendon tensions and consequentely of the force error $F - F_d$. Then the controller C_2 was activated, with the effect of a sudden reduction of the norm of the force error, as shown in figure 4.12.

Since $C_2 = 0$ at steady-state, the system is in a configuration which satisfies the first order condition of the primary problem (4.1). Figures 4.11 and 4.12 show that when C_2 is active the error $F(\mathbf{q}) - F_d$ decreases, while maintaining the task $\mathbf{x} - \mathbf{x}_d = 0$. This means that J_F^T is a good approximation of the real Jacobian $\frac{\partial F^T}{\partial \mathbf{q}}$, or, more precisely, that $J_F^T(F(\mathbf{q}) - F_d)$ is a descent direction for the secondary function $\|F(\mathbf{q}) - F_d\|$.

In an additional experiment, the system is driven toward an arbitrary $\mathbf{x}_d$ using the controller (4.2). When at steady-state, the desired tendon force F_d is changed periodically. The experiment is shown in figures 4.13 and 4.14. As expected, steps of the desired tendon force result in steps of the actual forces in the same

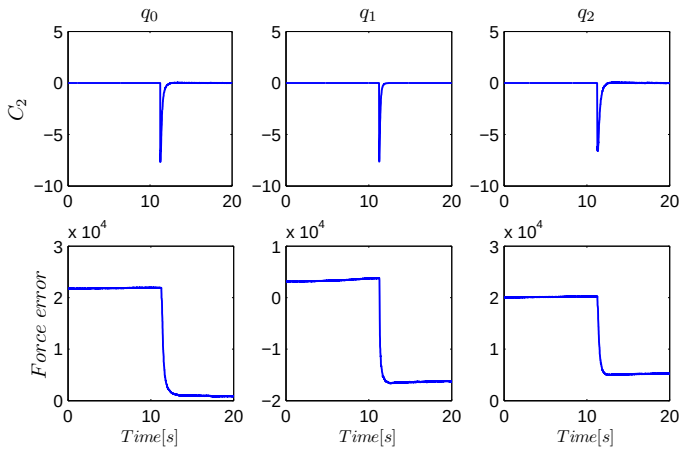


Figure 4.11: Top figures: contribution of controller C_2 for every joint. Bottom figures: force errors $F - F_d$ on every tendon.

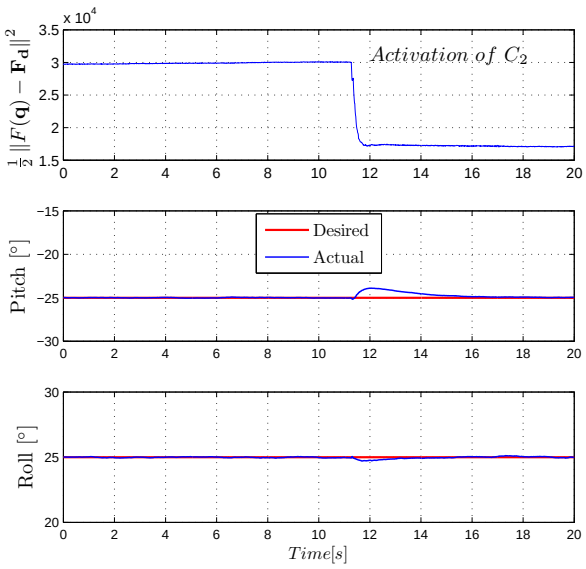


Figure 4.12: Top figure: norm of the force errors. Bottom figures: pitch and roll desired and actual positions. The activation of the controller C_2 reduces the norm of the force errors, without affecting the main positioning task, at steady-state.

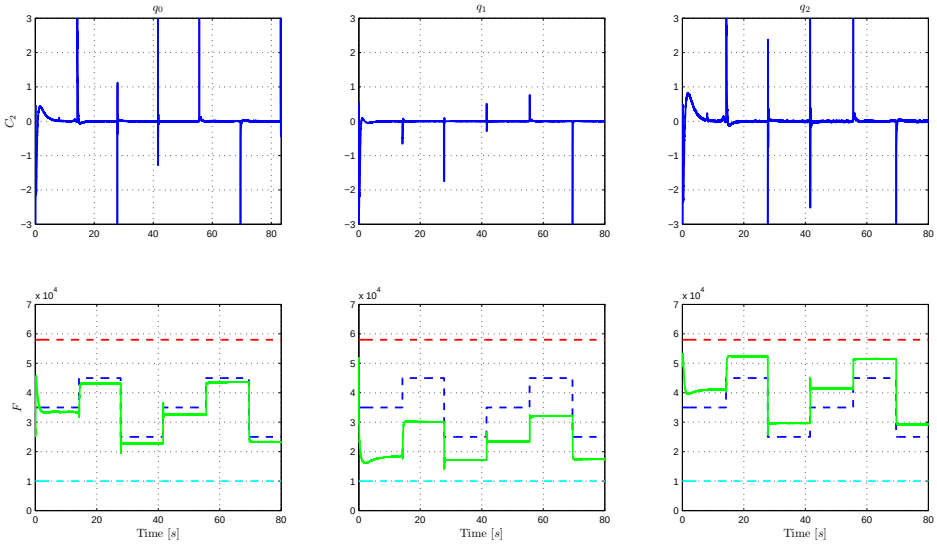


Figure 4.13: Top figures: contribution of controller C_2 for every joint. Bottom figures: actual (green solid line) and desired (blue dashed line) forces on every tendon. Step variations of the desired tendon forces are given to evaluate the performances of the controller C_2.

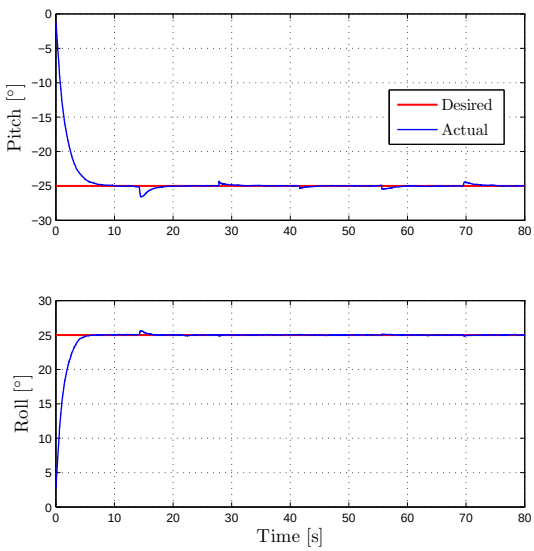


Figure 4.14: Desired and actual pitch and roll positions. The step variations of desired forces does not influence the pitch and roll positions at steady-state.

direction. Of course, since the desired force is the same for all the tendons, and the primary task is the position control of the head (see figure 4.14), steady state errors are present in the response of the system. This experiment demonstrates that the controller C_2 is able to regulate tendon tensions following F_d, without interfering with the main orientation task.

4.4 Gaze controller

The ability to orientate the neck as needed is fundamental for the robot in order to gaze toward 3D points in space. In humans, gazing movements are performed mainly to position the high-resolution foveae at the perceived interesting region in the external world, in order to improve visual perception. In our robot, the reason is primarily to create a consistent reference for learning and execution of reaching actions; this point will be better explained in section 5. Anyway, even if our cameras have fixed resolution, the central part of the image shows more uniform brightness and contrast, and is absolutely not affected by lenses distortive effects (which conversely can be seen, even if poorly, at the image borders).
In this work, the problem of gazing has been simplified substantially respect to its biological counterpart. Anyway, the goal of the gazing action (to bring the target in the central part of both eyes visual field) has been preserved.
We choose a subset of the neck/eyes DOFs to resolve the system intrinsic redundancy; in particular, we decided to actuate just 3 DOFs, namely neck yaw and pitch rotations and eyes vergence. Therefore, the head configuration considered in gaze control (and consequently also in reaching control) is defined as $\mathbf{q}_{head} = [\theta_v \ \theta_y \ \theta_p]^T$.
The target position in visual field $\mathbf{x} \in \mathbb{R}^3$ is computed from its $< u, v >$ position on both eyes cameras following equation 4.7:

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} u_L - u_R \\ \frac{u_L+u_R}{2} \\ \frac{v_L+v_R}{2} \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} u_L \\ u_R \\ v_L \\ v_R \end{bmatrix} \tag{4.7}$$

Note that this simple linear transformation do not introduce any change of coordinates and is not based on any model of the head nor of the cameras. Anyway,

it provides a couple of advantages. First, it allows to reduce the dimensionality of the hand position from 4 to 3 dimensions, without losing any information. In fact, the vertical components of target position on left and right images, v_L and v_R, should always be equal, since neck roll rotation is set to zero. Anyway, small errors in the eyes calibration (i.e. tilt divergence θ_{tv} not perfectly set to zero[1]) could lead to a small difference between v_L and v_R: therefore, we consider their average value instead of taking just one of the two components. Moreover, such an arrangement of the variables closely relates $\mathbf{x}$ components to $\mathbf{q}_{head}$ components. Indeed, x_0 can be controlled mainly actuating θ_v, and the same holds for the other two components.
A simple proportional control law is employed to bring $\mathbf{x}$ toward zero; neck and eyes velocities are generated as follows:

$$\dot{\mathbf{q}}_{head} = -K\mathbf{x} \tag{4.8}$$

where $K \in \mathbb{R}^{3\times 3}$ is a positive constant diagonal gain matrix. Clearly, $\mathbf{x} = 0$ means that u_L, u_R and $\frac{v_L+v_R}{2}$ are zero, and so that the target is in the center of both images (i.e. fixated).
This controller is used by the robot in two different situations: to fixate interesting objects, whose $< u, v >$ location is provided by the attention system described in the appendix (8.4), and to fixate its own hand (i.e. the green ball used as a marker), whose $< u, v >$ position is extracted through simple visual processing as described in the appendix (8.3).

[1]The absolute value of tilt divergence could slightly differ from the value measured with motors encoders, due to motor backlash and/or tendons tension decrease.

5

Reaching

The goal of the reaching action is to bring the robot end-effector (generally, the robot hand) into a specific target 3D position in space. In general, controlling the position of the end-effector can be useful for different tasks (e.g. pointing); nevertheless, in most cases robots (as well as humans) exploit reaching to interact with the external environment, i.e. they reach for an object. In this case, the retrieval of the object 3D position from vision requires a kinematic model of the eyes-head system and the calibration of eyes cameras; then, a kinematic model of the arm is needed to plan the appropriate motion.

In section 2.4 we discussed how the encoding of object position can be redefined in order to simplify the reaching control, using an internal representation instead of an external one. Indeed, in our approach we assume that the robot is fixating the target object when the reaching action starts. As a result, the object position can be defined as the head joints configuration. If we consider all joints of the neck/eyes system (7 DOFs), an infinite number of configurations exists which result in the fixation of a given point in space. Anyway, as explained in section 4.4, we reduced the gazing action to the control of 3 DOFs, namely neck pitch and yaw rotations and eyes vergence, while the other joints positions are set to zero: in this way the mapping from head configuration to object position becomes unique.

In this formulation, reaching can be therefore defined as the movement which brings the hand in the robot fixation point. As already mentioned in section 3.5, we simplified vision by attaching a marker (a green ball) to the robot hand. In

the following we will refer either to the hand or to the marker meaning the same physical entity.
The reaching action is accomplished in two phases: a first ballistic motion and a later corrective movement based on visual feedback. This solution is inspired by the human organization of reaching movements, and shows a couple of interesting practical aspects for robots.
The ballistic phase, which is generally responsible for longer part of the movement, both in terms of time duration and hand displacement, has two main advantages respects to the later one, due to the fact that it does not require visual feedback: it can be executed even if the hand is outside the visual field at the movement onset and it does not suffer from velocity limits imposed by feedback delays or visual sensors noise. The ballistic controller requires feedback just from the motor encoders (faster and almost noise free) in order to bring the arm to the desired target joints configuration.
The appropriate target arm configuration can be obtained by inverting a forward kinematic model of the arm-head system. The required function maps a specific arm configuration into the head configuration which allows the fixation of the hand. In our approach this function is approximated by a neural network, which is trained online by the robot during subsequents reaching trials: we will refer to it as the *arm-head forward kinematic map*. During the initial learning stages, when this map is still empty, ballistic movements are guided by a limited set of pre-coded arm motor synergies, triggered by head motion. Their use is inspired by the role of primitive reflexes (like, in particular, the ATNR) in human infants [Konczak, 2005]. As learning occurs these synergies are gradually suppressed and the system relies only on the maps learned with experience.
Of course, the accuracy of the ballistic controller depend on the quality of the approximation of the kinematic model, which improves with time, as the learning network is trained with new samples. Unfortunately, it is known that learning from examples always leave a residual estimation error, and the ballistic controller would produce no error in the final hand position only if the model estimation were absolutely perfect. Therefore, the use of visual feedback is necessary to reduce to zero the hand final position error. This happens in the second phase of the reaching action, the closed-loop corrective movement, which occurs when the

hand is close to the target object; on the other hand, at this point the velocity limits imposed by visual feedback become less critical (since when the hand approaches the target object its velocity decreases anyway). A non-linear Jacobian matrix which maps arm velocities in joints space to the corresponding hand velocities in visual space is needed by the closed-loop controller to cancel a visually detected error (the distance from the hand to the fixation point). In our system this non-linear matrix is approximated by a neural network which is updated incrementally by the robot during action: we will refer to it as the *visuo-arm Jacobian map*.

In the following more details about the control architecture and the learning process are given. We show experimental results which demonstrate that the system, starting from a very limited amount of knowledge, is able to learn in an autonomous and incremental fashion how to perform reaching actions efficiently. It is worth noting that state exploration, which is necessary in any learning system, is realized during the execution of reaching trials (*goal-directed exploration*) and does not require any particular training stage separated by the actual reaching behavior. Moreover, most of the reaching actions end successfully also at the very beginning of learning, mainly exploiting the pre-coded motor synergies (in the ballistic phase) and the visual feedback (in the corrective phase), even if the arm trajectories look rough and ungracious.

Finally, preliminary studies have been carried out concerning the possibility of creating a representation of the robot reachable space through incremental learning. Such a representation could be useful for the robot to evaluate its possibility to reach for a visually detected object before starting the actual movement, potentially giving also some hints about how a successful reaching action could be performed (e.g. bending the waist or walking toward the object).

Indeed, neurophysiological evidence show that human perception of what is reachable or not relies on motor information [Coello et al., 2008]. This suggest that a *reachable space map* can be based on an internal representation, and can be learned from motor experience. In fact, if the robot is not able to reach for a fixated target the main reason could be that the target lies outside the end-effector workspace. This is especially true as the robot becomes reliable in fulfilling reaching tasks. Positive and negative results of reaching actions can be a basis on which

an incremental map can be learned. Once trained enough, such a map would allow the robot to determine if an object is reachable or not before starting the reaching movement, by simply fixating it.
To the sake of clarity, we recall the definition of arm and head configuration, as used hereinafter:

- $\mathbf{q}_{arm} = [\theta_{sy}\ \theta_{sp}\ \theta_{sr}\ \theta_e]^T \in \mathbb{R}^4$
- $\mathbf{q}_{head} = [\theta_v\ \theta_y\ \theta_p]^T \in \mathbb{R}^3$

where θ_{sy},θ_{sp},θ_{sr} are the shoulder yaw, pitch and roll rotations (adduction/abduction, elevation/depression and rotation of the arm), θ_e is the elbow flexion/extension, θ_v is the eyes vergence, and θ_y,θ_p are the neck yaw and pitch rotations (rotation and elevation/depression of the head).

5.1 Reaching control

The reaching controller is based on the combination of two separate controllers: an open-loop controller which brings the hand in the proximity of the target (ballistic reaching) and a closed-loop controller which is activated when the hand is close to the target (Jacobian-based correction).

5.1.1 Open-loop

The control scheme for the open-loop controller is depicted in figure 5.1.
The desired arm configuration which brings the hand in the eyes fixation point is obtained by balancing two contributions with the coefficient α (which will be discussed later). This arm configuration is then sent to the *armController* (see section 3.5) which computes the appropriate motor velocities that generate the actual joints movement.
The first contribution, $\tilde{\mathbf{q}}^S_{arm} \in \mathbb{R}^4$, is the output of the *Motor Synergies* block, plus an addittive white Gaussian noise. The *Motor Synergies* block chooses one among four pre-defined arm configurations according to the current head configuration, $\mathbf{q}_{head} \in \mathbb{R}^3$, in order to bring the hand roughly in the field of view (this is implemented as a look-up table). Adding noise improves the exploration

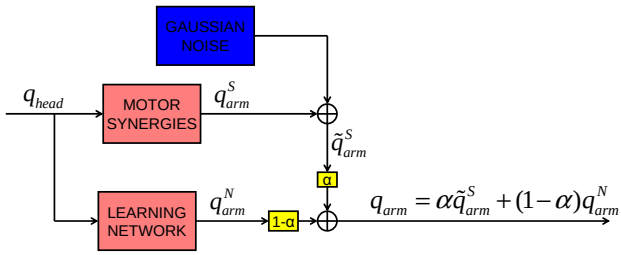


Figure 5.1: Control scheme concerning the open-loop (ballistic) component of the reaching movement.

of state space, expecially during the first reaching trials, when motion is mainly guided by these pre-coded motor synergies; this simulates the defective command generation (related to muscle control) observed in human infants [Kinney et al., 1988], which is considered to be an important aspect for motor learning.
The second contribution is given by the learning network: a minimization algorithm (IpOpt, Interior Point Optimizer, [Wachter and Biegler, 2006]) is used in combination with the arm-head forward kinematic map to compute the appropriate arm configuration which brings the hand in the robot fixation point. Of course the inverse mapping (from head configuration to arm configuration) is not unique: among all possible solutions we choose the one which minimizes the arm joints displacements from the current arm configuaration, $\mathbf{q}^C_{arm} \in \mathbb{R}^4$, to the target one. This simple but reasonable choice prevents the controller from generating "useless" motion in joint space. More formally, the target arm configuration, $\mathbf{q}^N_{arm} \in \mathbb{R}^4$, is obtained as follows:

$$\mathbf{q}^N_{arm} = \arg \min_{\mathbf{q}_{arm} \in [\mathbf{q}^L_{arm}, \mathbf{q}^U_{arm}]} \left\| \mathbf{q}_{arm} - \mathbf{q}^C_{arm} \right\|^2 \tag{5.1}$$

$$s.t. \;\; \mathbf{0} \leq (\mathbf{q}_{head} - fwdKin(\mathbf{q}_{arm})) \leq \epsilon \tag{5.2}$$

where $\mathbf{q}^L_{arm}, \mathbf{q}^U_{arm} \in \mathbb{R}^4$ are the lower and upper bounds on $\mathbf{q}_{arm}$ (joints limits), $fwdKin(\mathbf{q}_{arm})$ is the output of the arm-head forward kinematic map, queried with $\mathbf{q}_{arm}$ as input, and ϵ is an arbitrary low error threshold (we set $\epsilon = 0.0001$). The coefficient α is a function of the number of points that have been learned by the arm-head forward kinematic map: this means that in the initial stages

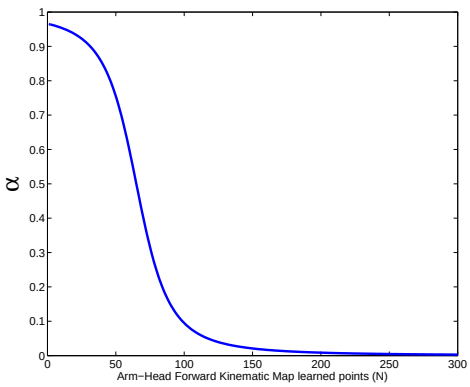


Figure 5.2: Trend of the coefficient α in comparison with the growth of the arm-head forward kinematic map (namely, N, the number of points that have been learned).

of development the robot mainly moves using the motor synergies, gradually switching to the use of the neural network with the growth of the learned data set.

Equation 5.3 relates the increase of N to the decrease of α; the function, depicted in figure 5.2, has the form of a sigmoid function, parametrized with the numeric coefficients which better fit in our situation (depending on the convergence of the arm-head forward kinematic map estimation error).

$$\alpha(N) = 0.5 - \frac{0.5(\frac{N}{8} - 8)}{\sqrt{8 + (\frac{N}{8} - 8)^2}} \tag{5.3}$$

5.1.2 Closed-loop

During the corrective movement, motor velocities $\dot{\mathbf{q}}_{arm} \in \mathbb{R}^4$ are computed by the closed-loop controller using equation 5.4:

$$\dot{\mathbf{q}}_{arm}(t) = -K\hat{J}_v^{\dagger}(\mathbf{q}_{arm})\mathbf{x}(t) \tag{5.4}$$

where $K \in \mathbb{R}^{4\times4}$ is a diagonal constant positive gain matrix, $\hat{J}_v(\mathbf{q}_{arm}) \in \mathbb{R}^{3\times4}$ is the output of the visuo-arm Jacobian map, $\hat{J}_v^{\dagger}(\mathbf{q}_{arm}) \in \mathbb{R}^{4\times3}$ is its Moore-Penrose generalized inverse [Penrose, 1955] and $\mathbf{x}(t) \in \mathbb{R}^3$ is the hand position in the visual field. These velocities are updated every 5 ms (control rate, 200 Hz)

and sent to the robot controller. $\hat{J}_v(\mathbf{q}_{arm})$ is updated at a lower rate (10Hz) by querying the visuo-arm Jacobian map with the current $\mathbf{q}_{arm}$[1].
The hand position in the visual field $\mathbf{x}$ is computed from the $< u, v >$ position of the visual marker on both cameras following equation 5.5, as described previously in section 4.4:

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} u_L - u_R \\ \frac{u_L+u_R}{2} \\ \frac{v_L+v_R}{2} \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} u_L \\ u_R \\ v_L \\ v_R \end{bmatrix} \tag{5.5}$$

5.2 Learning

Figure 5.3 describes a sequence of actions characterizing the robot behavior. The robot sensory feedback (green decision blocks) drives the choice of the action that has to be performed at a specific time (blue action blocks). It is important to stress the fact that exploration, learning and execution collapse in the very same process, which is fully autonomous and continuous. Practically speaking, the robot is just switched on and operates, without any external intervention from the experimenter (except for showing some objects periodically).
Hereinafter the robot behavior is discussed in more details.

. An attention system (whose details are reported in the appendix, section 8.4) provides the robot with an interesting target (3D point) within the visual field and drives the robot fixation toward that point. The target could be typically a colored and/or moving object.

. Once the robot has fixated the target, it tries to reach for it with the open-loop controller.

. As soon as the hand falls inside the central[2] part of the robot field of view, the Jacobian-based correction is activated. If the hand does not fall inside

[1]Note that these control parameters have proven to be appropriate in our system considering the average velocities at which the robot moves (about 0.2 m/s, end-effector velocity)

[2]Images coming from eye cameras have 320x240 pixels resolution. The image center has been arbitrarily defined as an area of 120x120 pixels centered at the image center, for both eyes.

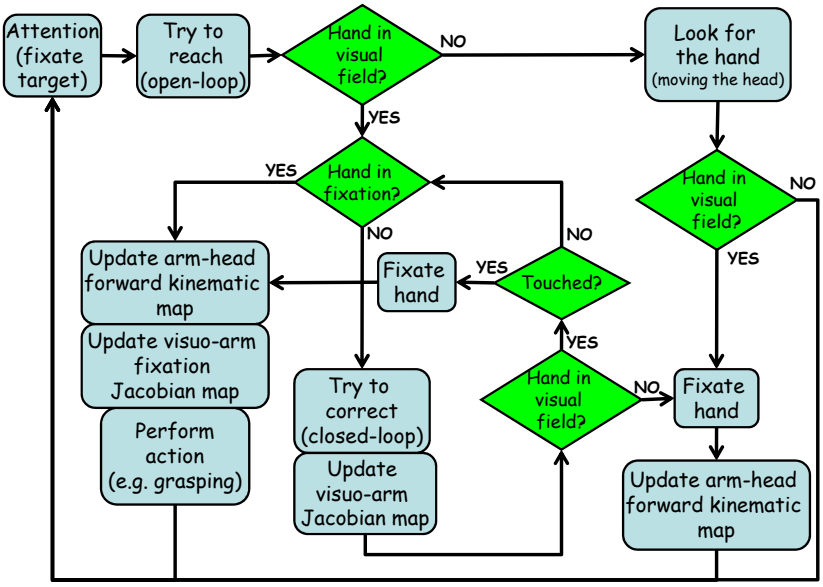


Figure 5.3: James reaching behavior. This continuous and autonomous process constitutes either exploration, learning and execution of the reaching action.

this region (which is typically the case during the first reaching trials), the Jacobian-based correction is activated anyway after the completion of the ballistic movement (i.e. after the target arm configuration in joint space has been reached).

. If the hand does not even fall inside the visual field (or it falls just at the image border), the robot starts moving the head in order to fixate the hand; if any visual cue is present (e.g. part of the ball is visible near the image border) the head follows it, otherwise a stereotyped motion strategy (i.e. random left-right movements of the neck) is employed, which allows the robot to explore its visual workspace.

  - If search for the hand fails (the hand is outside the visual workspace or it is occluded) the control comes back to the attention system, which looks for another interesting target.
  - Otherwise, if the hand is fixated, the arm-head forward kinematic map can be updated with a new couple $< \mathbf{q}_{arm}, \mathbf{q}_{head} >$; then the control comes back to the attention system.

. During the closed-loop correction the robot continuously checks the visual error.

  - If the error increases too much (the hand is falling outside the visual field) or if it has not been canceled after a certain amount of time (the quality of the Jacobian estimation is too poor or the target point lies outside the robot reachable space), the robot stops moving the arm and tries to fixate its hand moving the head.
  - Otherwise it goes on correcting until the visual error is canceled or the hand touches the target object. In the latter case, the robot moves the fixation point to the hand (learning a new couple $< \mathbf{q}_{arm}, \mathbf{q}_{head} >$), and than back to the object. During the correction the visuo-arm coarse Jacobian map is constantly updated with couples $< \mathbf{q}_{arm}, \hat{J}_v >$. If the correction ends successfully also the visuo-arm fixation Jacobian map is updated with a couple $< \mathbf{q}_{arm}, \hat{J}_v >$.

. If a contact with the object is established the interaction can start (e.g. grasping), otherwise explorative movements of the wrist can be performed to get in touch with the object.

. As soon as the robot is no longer interested in the current object, or if it does not manage to touch it, the attention system takes the control again, looking for another target.

5.2.1 Arm-head forward kinematic map

The arm-head forward kinematic map is implemented as a Receptive Field Weighted Regression neural network (RFWR, [Schaal and Atkenson, 1998]) and is trained online. Everytime the robot is fixating the hand, the map is trained with the arm configuration, $\mathbf{q}_{arm} \in \mathbb{R}^4$, as input and the head configuration, $\mathbf{q}_{head} \in \mathbb{R}^3$, as output. Fixation of the hand could occur in different situations:

. after an unsuccessful ballistic reaching: if the hand falls outside the visual field the robot tries looking for it moving the head;

. after an unsuccessful corrective movement: if the robot understand that the hand is being moving outside the visual field (visual error is increasing too much) it stops the corrective movement and fixates the hand;

. after a successful corrective movement: if the closed-loop controller succeeds the hand comes in the fixation point.

. after a touch is detected. If the hand is inside the visual field, the head gaze toward the hand;

. by chance: if the hand casually passes from the fixation point during a movement.

5.2.2 Visuo-arm Jacobian map

The visuo-arm Jacobian map is implemented as a RFWR neural network which receives the current arm configuration, $\mathbf{q}_{arm} \in \mathbb{R}^4$, as input and gives a local Jacobian matrix ($\hat{J}_v \in \mathbb{R}^{3\times4}$, 12 constant coefficients) as output. A local Jacobian

matrix $\hat{J}_v^{\widetilde{\mathbf{q}}_{arm}}$ can be estimated from sensory measurements (arm joints displacements, $\Delta\mathbf{q}_{arm}$, and hand visual displacements, $\Delta\mathbf{x}$) collected while moving in the proximity of $\widetilde{\mathbf{q}}_{arm}$. Then, the RFWR network can be trained with couples $< \widetilde{\mathbf{q}}_{arm}, \hat{J}_v^{\widetilde{\mathbf{q}}_{arm}} >$, for different $\widetilde{\mathbf{q}}_{arm}$.
Here we are considering J_v as a function of the arm configuration only; theoretically, J_v is a function of both the arm configuration and the head configuration. Anyway, if we strictly consider movements of the hand near the fixation point, we can reasonably approximate $J_v(\mathbf{q}_{arm}, \mathbf{q}_{head})$ with $J_v(\mathbf{q}_{arm})$, being $\mathbf{q}_{head}$ a function of $\mathbf{q}_{arm}$ ($\mathbf{q}_{head} = fwdKin(\mathbf{q}_{arm})$). Therefore, the RFWR network must be trained with couples $< \widetilde{\mathbf{q}}_{arm}, \hat{J}_v^{\widetilde{\mathbf{q}}_{arm}} >$ obtained when the corrective movement ends successfully (i.e. the hand is in the fixation point). In the results section (5.3.2) we will refer to this map as the visuo-arm fixation Jacobian map. Unfortunately, this map can not be used for closed-loop control from the very beginning, since it requires the hand to be in the fixation point (i.e. a successful closed-loop correction) to be updated. A different strategy is needed to drive the robot motion during the first reaching trials. Therefore, we trained another RFWR neural network, using the same strategy we employed in the learning of neck Jacobians (see section 4.2). The only difference is that here the rate at which data is gathered and the map is updated (10Hz) is lower than the control rate (200Hz). This causes adaptation to be slower in the initial stages of learning, but assures better control performances as soon as the Jacobian estimation error becomes reasonably small. Again, data consist in couples of values of arm joints displacements, $\Delta\mathbf{q}_{arm}$, and hand visual displacements, $\Delta\mathbf{x}$. Differently from the previously described visuo-arm fixation Jacobian map, here data gathering and training of the RFWR network are performed whenever the hand moves inside the visual field (also during the ballistic movement), even if not in the fixation point. Of course, approximating $J_v(\mathbf{q}_{arm}, \mathbf{q}_{head})$ with $J_v(\mathbf{q}_{arm})$ when the hand is far from the fixation point leads to larger estimation errors. Nevertheless, such a map can be used from the very beginning to control the closed-loop phase of the movement. In the results section (5.3.2) we will refer to this map as the visuo-arm coarse Jacobian map.
Clearly, at the very beginning the map is empty. Anyway a non-null matrix should be provided to the controller to generate a motion. Since we do not have

any a-priori information about the true Jacobian of the system, any arbitrary constant matrix can be chosen; the motion generated by using such a matrix will not necessary bring the hand closer to the target but will serve as explorative movement. Indeed, a feature of the employed learning algorithm is that we can initialize the map even with a "wrong" matrix, without preventing the converging of the estimation error. This aspect will be discussed later in the results (section 5.3.2, figure 5.13).

5.2.3 Reachable space map

Recent studies have tested numerical or machine-learning tools in order to build a representation of the robot reachable space [Yisheng and Yokoi, 2006; Zacharias et al., 2007]. In our system the reachable space map has been implemented using a RFWR neural network which is trained online during the sequence of reaching trials. The map is trained with the head configuration, $\mathbf{q}_{arm} \in \mathbb{R}^3$, as input and a value $S \in \{0, 1\}$ which indicates the failure/success of the reaching action as output. Every time the arm-head forward kinematic map is trained with a new sample $< \mathbf{q}_{arm}, \mathbf{q}_{head} >$, the reachable space map is trained with $< \mathbf{q}_{head}, S = 1 >$, because a feasible arm configuration which brings the hand in the fixation point defined by $\mathbf{q}_{head}$ exists. Conversely, if a reaching task has not been accomplished the map is trained with $< \mathbf{q}_{head}, S = 0 >$. Of course, some of the first reaching trials could fail even if the target lies inside the workspace, because both the arm-head forwad kinematic map and the visuo-arm Jacobian map are yet not developed. This generates wrong training data for the reachable space map. Nevertheless, this can be interpreted as noise in the input data, decreasing as the whole reaching system develops. This noise should not affect learning considerably, since the learning network is characterized by a forgetting factor that decreases as more samples are learned: the network forgets more in the beginning and less and less as learning occurs. Indeed, the last samples are more important than the old ones in the estimation of the output.
When queried with a head configuration (i.e. a fixation point) as input, the map gives as output the probability that the fixated point is reachable. Some preliminary results obtained are shown in the next section.

#	1	2	3	4	5	6	7	8	9	10
vergence	-2.5	-2.0	-1.5	-1.0	0.0	0.5	1.0	1.5	2.0	2.5
yaw	65	25	60	45	35	20	50	40	75	70
pitch	-20	15	-5	20	-10	25	10	0	-25	5

Table 5.1: The ten head configurations (corresponding to ten different 3D target points in space) used in the test experiment for the assessment of ballistic reaching performances.

5.3 Experimental results

Several measurements have been done to analyze the performances of the system during subsequent learning stages. The different parts of the system have been tested both separately and combined in order to assess their contribution to the overall behavior.

5.3.1 Ballistic reaching

In order to precisely assess the accuracy of ballistic reaching, we tested the robot while reaching for ten different positions within its workspace. These positions have been chosen to cover the robot workspace nearly uniformly. The attention system has been here replaced by a "fake" attention system which was just driving the robot fixation toward precise 3D points in space (i.e. moving the head toward ten different pre-selected configurations). After the robot has fixated the 3D point, ballistic reaching is performed (without any feedback correction) until the arm reaches its target configuration; at this point the visual error $\|\mathbf{x}\|$ is computed.

The sequence of ten movements has been repeated several times, using the arm-head forward kinematic map at different evolution stages (i.e. increasing the number of learned points). In table 5.1 the ten different head configurations (correspondent to ten different 3D target points in space) used in the test are reported. Figure 5.4 plots the RMSE (Root Mean Square Error) on the ten target points (the vertical bars indicate the standard deviation). As expected, the RMSE decreases as more samples are learned by the robot.

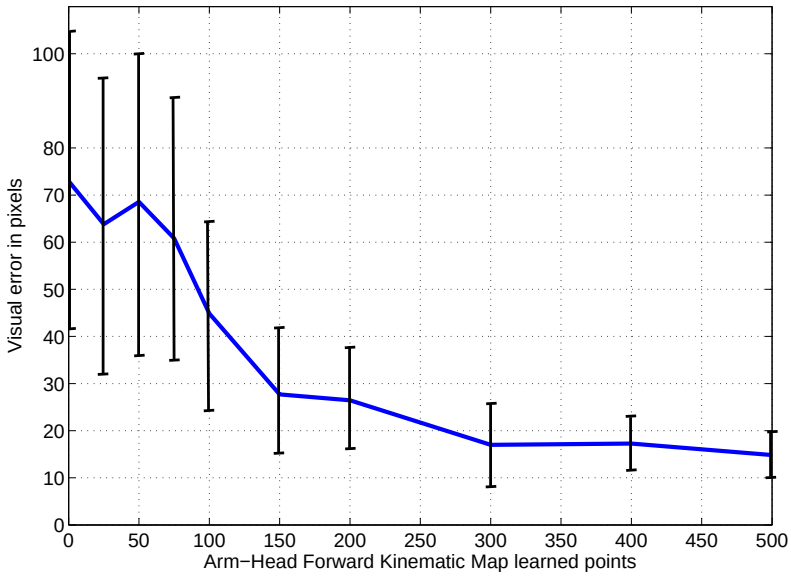


Figure 5.4: Position error of the end-effector after ballistic reaching. On x-axis the learned points of the Arm-Head Forward Kinematic Map. Ballistic motions toward ten target locations within the robot workspace have been performed at different (discrete) learning stages. RMSE of the position errors at each learning stage is reported on y-axis (mean and standard deviation of the ten different target locations).

The final RMSE, evaluated when the arm-head forward kinematic map has been trained with 500 samples, is approximatively 14 pixels, with 5 pixels of standard deviation. Considering the average absolute position of the hand respect to the eye cameras, this corresponds to an error of about 1.5 ± 0.5 cm.

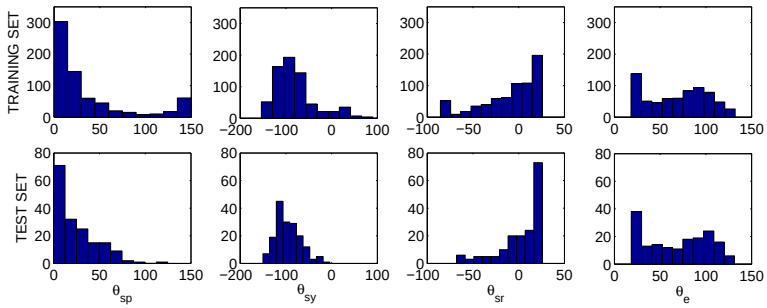


Figure 5.5: Distribution of training set and test set (input space, i.e. arm configuration). Data is the same as figure 5.4.

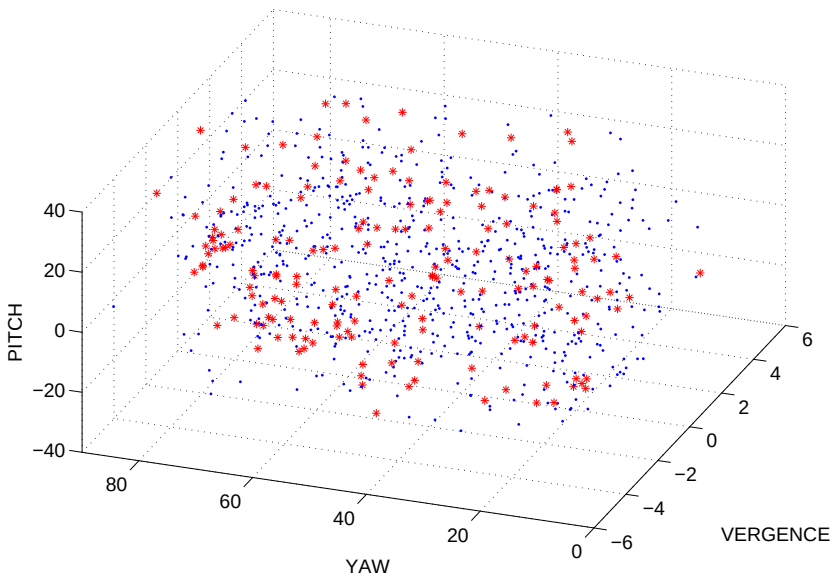


Figure 5.6: Data points of training set (blue dots) and test set (red crosses) (output space, i.e. head configuration). Data is the same as figure 5.4.

The ballistic controller relies on the knowledge of the arm-head forward kinematic map, which is learned incrementally through experience. This map is then

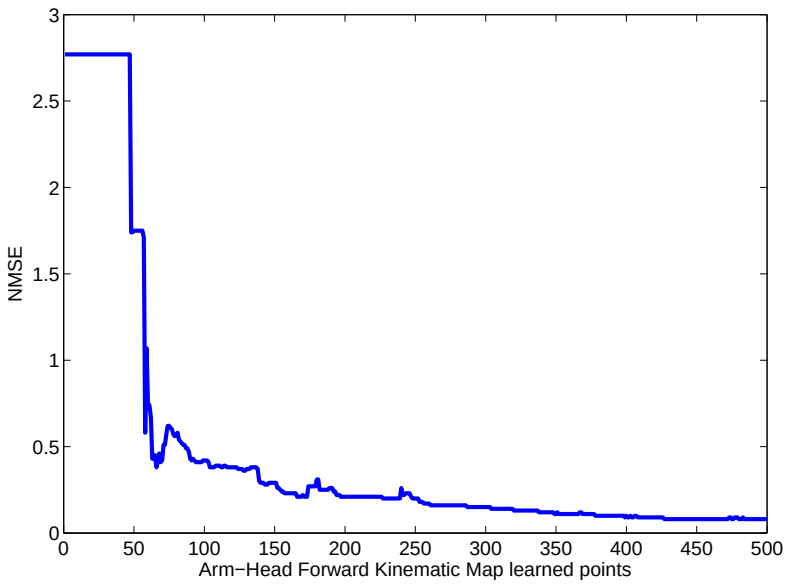


Figure 5.7: NMSE of the Head-Arm Forwad Kinematic Map output (mean of the three output components). On the x-axis the learned points of the Arm-Head Forward Kinematic Map. Training data is the same as figure 5.4.

inverted using a standard optimization algorithm and combined with the predefined motor synergies, as described in section 5.1.1. Therefore, we expect the error of the ballistic controller to follow the same trend of the arm-head forward kinematic map estimation error. To verify this, we computed the estimation error of the arm-head forward kinematic map on a test set of 200 samples (different than the 500 used for training), with increasing number of training samples. Figure 5.5 provides the distribution of training and test set, considering the input space (arm configuration). Figure 5.6 shows the output points of the training set (blue dots) and test set (red crosses), in the head configuration space. Figure 5.7 depicts the NMSE (Normalized Mean Square Error, defined as the MSE divided by the output variance) of the arm-head forward kinematic map as the number of training samples increases. The residual estimation error justifies the final position error shown by the ballistic controller in figure 5.4. In particular, the RMSE computed on the single output components (in figure 5.8) reveals that a significant error is present on the first component (eyes vergence) with respect to the other two components (neck yaw and pitch). Indeed, the residual error of vergence estimation is almost 1 deg. Such an error, considering the average hand position respect to eyes cameras, correspond to a position error of the hand of about 1 cm. This suggests that the error in the vergence estimation is the main cause of the error in the ballistic controller.

Additional analysis have been performed in order to compare the learning performances of RFWR with other state of the art machine learning tools, both online and offline. In particular, we computed the NMSE with increasing size of the training set using LWPR (Locally Weighted Projected Regression, an online technique [Vijayakumar and Schaal, 2000], whose details are reported in the appendix 8.1) and LS-SVM (an offline algorithm, whose details have been discussed in section 6.1.2). Figure 5.9 plots the reduction of NMSE as the number of training samples increases, using LWPR. The comparison between RFWR, LWPR and LS-SVM is reported in table 5.2.

We also investigated if a further increase of the size of the training set would have improved the estimation capabilities of the network on this particular problem. To the purpose, the arm-head forward kinematic map has been trained with 1300 samples, and tested with the same test set of 200 samples used in the previous

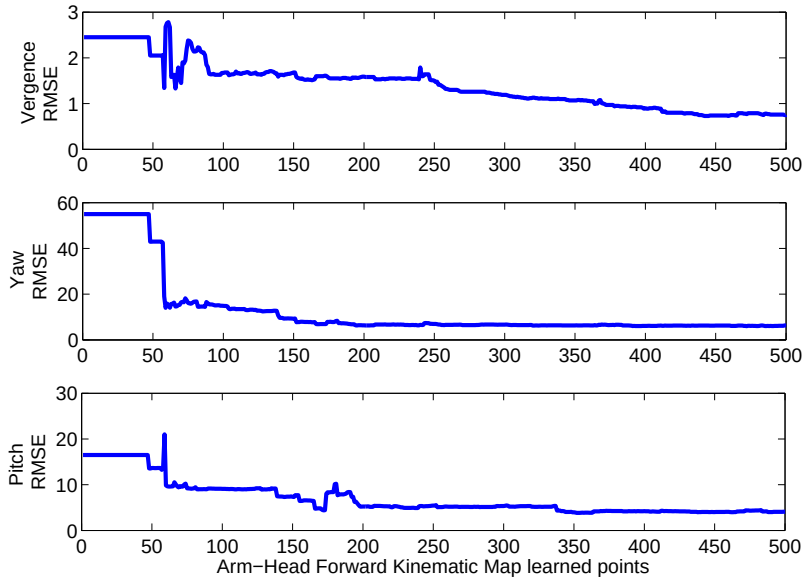


Figure 5.8: RMSE of the Head-Arm Forwad Kinematic Map output (from top: vergence, yaw and pitch). On the x-axis the learned points of the Arm-Head Forward Kinematic Map. Training data is the same as figure 5.4.

	RFWR		LWPR		LS-SVM	
Samples	250	500	250	500	250	500
vergence	0.40	0.10	0.40	0.18	0.276	0.100
yaw	0.10	0.08	0.12	0.09	0.022	0.021
pitch	0.10	0.06	0.09	0.05	0.007	0.006

Table 5.2: Comparison between RFWR, LWPR and LS-SVM in the estimation of the arm-head forward kinematic map. The NSME has been computed both with a training set of 250 samples and 500 samples, using the same test set of 200 samples. LS-SVM outperforms the other two algorithms, which behave in a similar way.

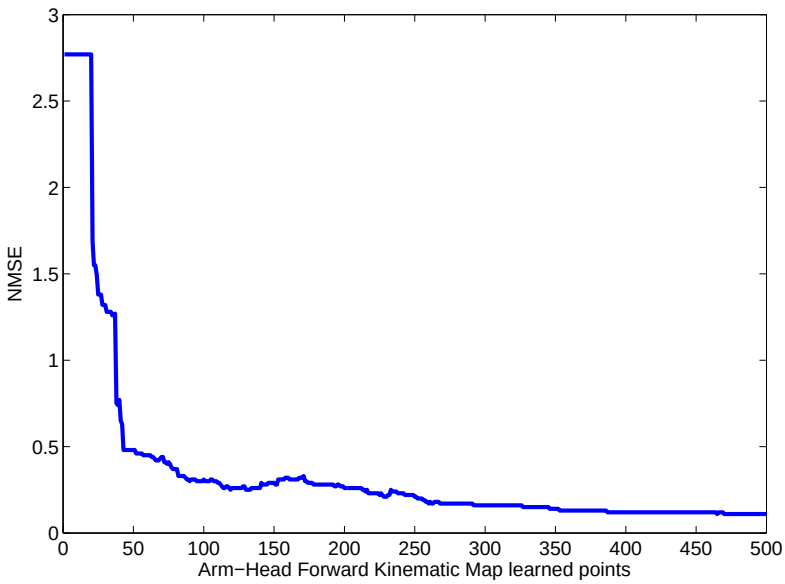


Figure 5.9: NMSE of the Head-Arm Forwad Kinematic Map output (mean of the three output components) trained with LWPR. On the x-axis the learned points of the Arm-Head Forward Kinematic Map. Training data is the same as figure 5.4.

tests. Note that the first 500 samples of the training set are the same of the previous tests.
Figure 5.10 plots the NMSE as the training set size increases. Learning has been performed using LWPR; the same analysis has been done using RWFR, ending up with similar results that are not shown here. Concerning LS-SVM, the NMSE (mean on the three output dimensions) decreases from 0.042 (500 training samples) to 0.034 (1300 training samples).
It appears quite clearly that including more data in the training set do not improve significantly the estimation in this particular situation.

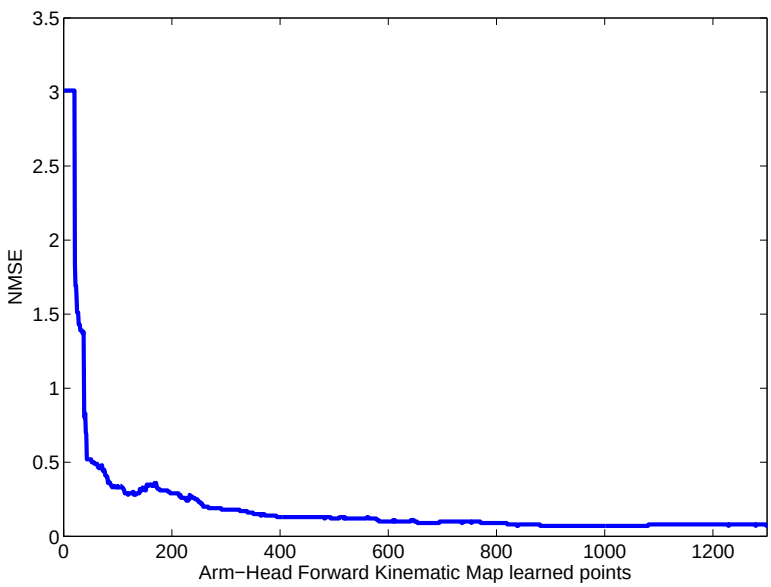


Figure 5.10: NMSE of the Head-Arm Forwad Kinematic Map output (mean of the three output components) trained with LWPR. On the x-axis the learned points of the Arm-Head Forward Kinematic Map. More training points are provided for regression (the first 500 points are the same as figure 5.4).

5.3.2 Closed-loop correction

The closed-loop controller exploits an approximation of the system visual Jacobian in order to cancel the residual visual error left by the ballistic controller.

Tests have been performed in which the approximation was achieved either with the visuo-arm coarse Jacobian map or with the visuo-arm fixation Jacobian map (both introduced in section 5.2.2). As expected, the best results were obtained with the visuo-arm fixation Jacobian map, which produces a more correct estimation of the real Jacobian. The test consisted in moving the hand toward eight symmetrical targets around the robot fixation point (forward and backward motion), while keeping the gaze still. Figure 5.11 reports the results obtained at different learning stages (i.e. map trained with increasing number of samples). The trajectory on every image is an average between the trajectories on left and right eyes (this way of depicting visual trajectories will be kept in most of the remainder of the section[1]). In the last plot on the right, obtained with the map trained with 400 samples, the hand trajectories in visual space are straight and regular, indicating a good approximation of the real Jacobian. Concerning this movement, figure 5.12 shows the trajectories of arm joints positions (on the left) and hand position errors (on the right). In general, also these trajectories are smooth and regular, with all variables converging to their desired values; the only exception is the movement of the elbow (θ_e), which looks a bit ungracious in a couple of istants, generating four undesired spikes in the vergence error (which should be always zero) and corresponding visual space trajectories not perfectly linear, precisely at time 50, 60, 130 and 140.

It must be noticed anyway that the system behavior at birth (first plot on the left in figure 5.11, map trained with 25 samples) is not acceptable: in fact, the robot is unable to reach for the targets. In fact, in the beginning the map is empty, and some initial structure which guides exploration is needed.

This problem can be solved by exploiting the visuo-arm coarse Jacobian map. In fact, while the visuo-arm fixation Jacobian map is updated only if the closed-loop correction ends successfully, the visuo-arm coarse Jacobian map is updated every time the hand moves inside the visual field, both during the open-loop and the closed-loop phase of the reaching movement (see section 4.2). Therefore an acceptable estimation (even if not perfect) can be obtained very rapidly. Such

[1] In most graphs we do not show the third visual dimension, $u_L - u_R$, since it is the one on which less motion is visible. Anyway, the trend of its convergence to zero is similar to the other two components, as it can be seen in figures 5.12 and 5.17, where it is shown.

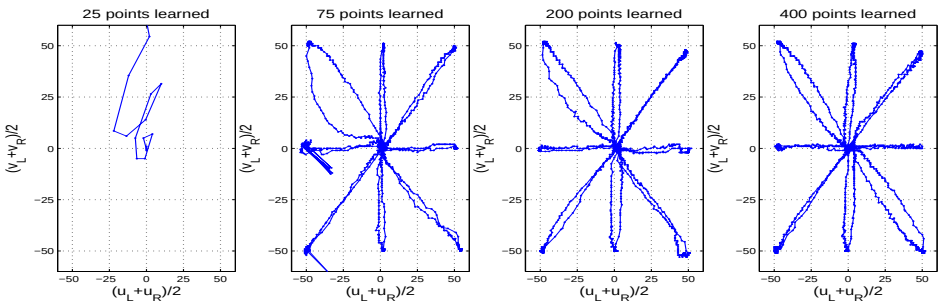


Figure 5.11: Evolution of the closed-loop visual trajectories using the visuo-arm fixation Jacobian map. As learning progresses trajectories become more and more linear. On the first figure on the left (25 learned points) the robot is unable to meet the desired points.

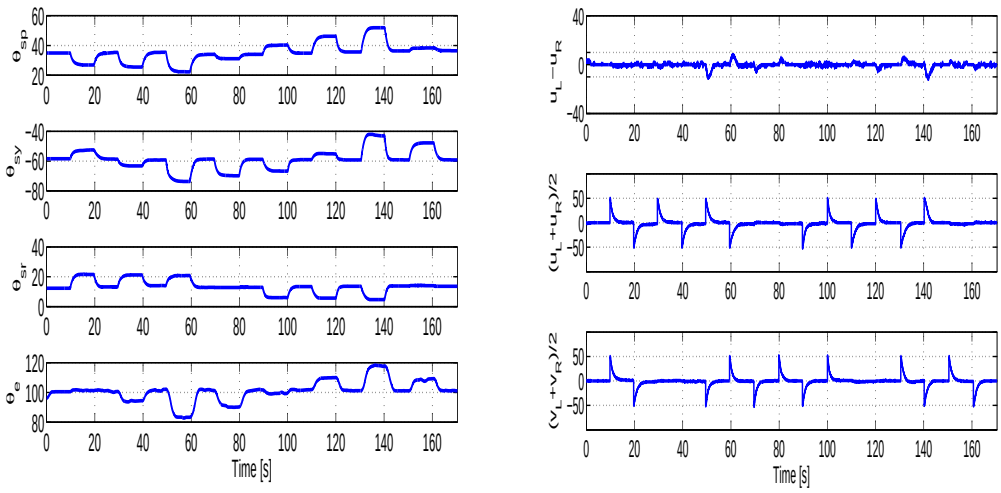


Figure 5.12: Arm joints position trajectories (left image) and hand position error trajectories (right image) using the visuo-arm fixation Jacobian map. The movements are the same depicted in the last graph on the right in figure .

an estimation can guide exploration and can be improved with time. The online learning algorithm is initialized with an arbitrary matrix, which constitutes the output of the neural network for every $\mathbf{q}_{arm}$ received as input (in the beginning, when the net is still empty). In our implementation the visuo-arm coarse Jacobian map has been initialized with a matrix, $\hat{J}_v^{INIT} \in \mathbb{R}^{3\times 4}$, with all coefficients equal to one ($\hat{J}_v^{INIT}(i,j) = 1\ \forall i,j$). In figure 5.13 we show a closed-loop movement using such a matrix (left image) and the improvements provided by the online learning (right image). The green cross indicates the starting point, while the red cross is the goal. While the initialization matrix drives the hand in a direction opposite to the target, the upadated matrix eventually brings the hand in the fixation point, with a straight trajectory. Note that learning is very fast: indeed, the whole movement duration (on right image) is about 4 seconds (average hand velocity $0.1\frac{m}{s}$).

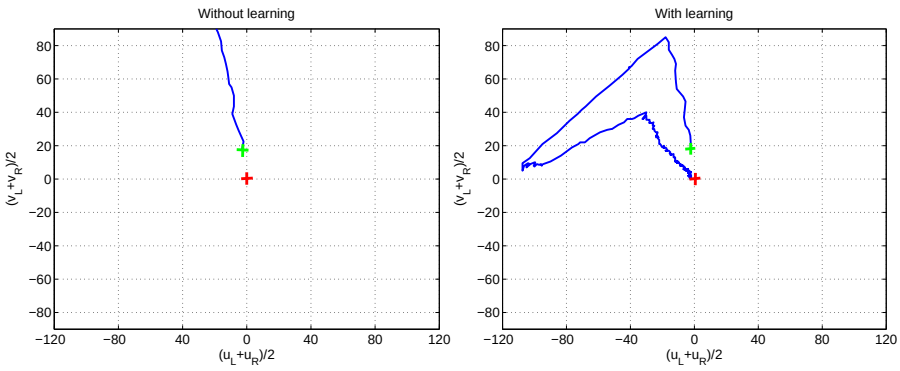


Figure 5.13: Online adaptation of the visuo-arm coarse Jacobian map. On the left, the closed-loop visual trajectory using the initialization matrix with no update. On the right, the map is updated online. The green cross in the starting point, the red one is the ending point (fixation point). While the initialization matrix drives the hand in a direction opposite to the target, the upadated matrix eventually brings the hand in the fixation point, with a straight trajectory.

The sequence of images in figure 5.14 shows the improvements of the closed-loop corrections (using the visuo-arm coarse Jacobian map) during the sequence of reaching attempts performed by the robot, trial after trial[1]. The plots give also

[1]Not all the trials are shown; we selected some of the more representative among the first 400

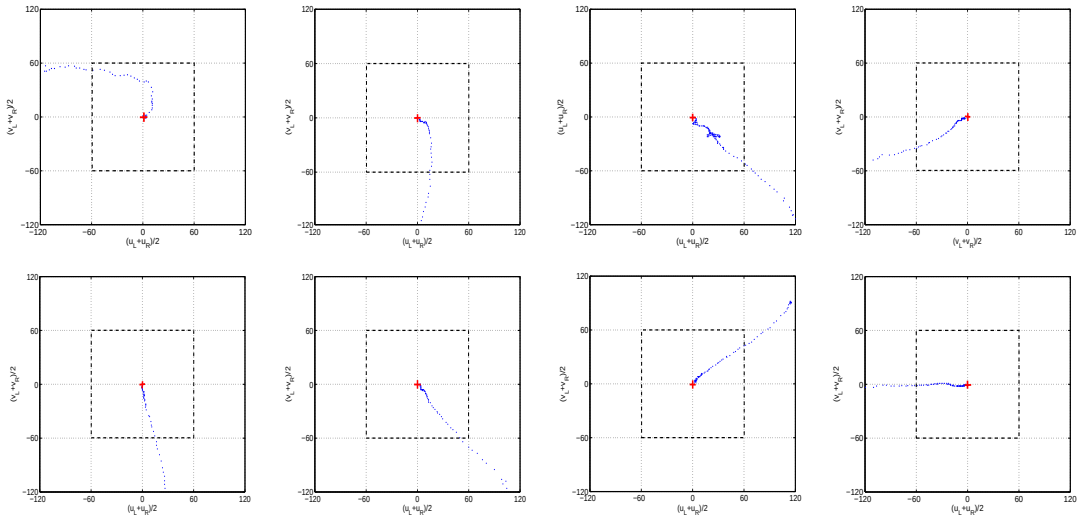


Figure 5.14: Visual trajectories of reaching movements composed by open-loop and closed-loop phase (using the visuo-arm coarse Jacobian map). The closed-loop phase starts as soon as the hand enters in the central part of the visual field (the highlighted central square).

an indication about the hand velocity during motion. Trajectory points are sampled at a fixed rate (20 Hz), so the distance between each couple of points is proportional to the actual hand velocity. It can be noticed that hand velocity decreases as the error is reduced, and that hand trajectories become more and more rectilinear and regular as learning of the visuo-arm coarse Jacobian map occurs.

Anyway, the accuracy of the controller is limited by the fact that we are using a coarse approximation of the real Jacobian (as explained in section 5.2.2). This can be better observed in the execution of the same test movement described at the beginning of this section (see figure 5.11). Figure 5.15 compares the controller behavior at birth (when the visuo-arm coarse Jacobian map is still empty) and after some learning (achieved during 400 reaching attempts toward randomly distributed objects). It is evident that, even if learning improves the system performances, hand trajectories in the right image (after learning) are still irregular and not rectilinear (as they would be if the Jacobian estimation were perfect).

In light of the reported results, the best solution for closed-loop control it is prob-

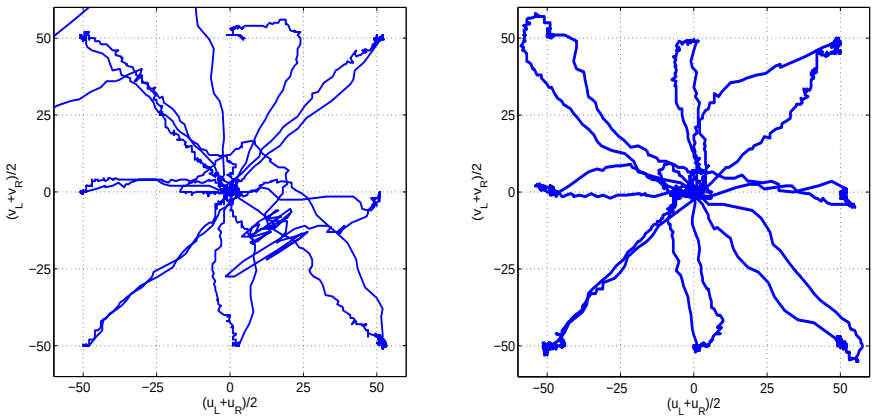


Figure 5.15: Closed-loop visual trajectories during a test movement, using the visuo-arm coarse Jacobian map. On the left, system behavior at birth, on the right, after 400 reaching trials has been performed.

ably to use both maps (the visuo-arm coarse Jacobian map and the visuo-arm fixation Jacobian map) depending on the current learning stage. The system can gradually switch from the use of the visuo-arm coarse Jacobian map (useful during the initial movements, mainly explorative) to the exploitation of the visuo-arm fixation Jacobian map, more accurate and reliable as soon as enough training samples have been gathered. A solution similar to the one adopted for the open-loop controller (where two contribution are balanced by the coefficient α) can be used also for the closed-loop controller. Anyway, such a solution has not been implemented in practice.

5.3.3 Whole reaching action

In this section we show some complete reaching trajectories, composed by the initial ballistic phase and the later Jacobian-based corrective motion, in order to further discuss some characteristics of the reaching movement. Arm joints positions have been recorder during the robot *goal-driven exploration*, at 20 Hz sampling rate. The visualization of the reaching trajectories in the 3D cartesian space is achieved through a kinematic model of James arm, realized in Matlab using the standard Denavit-Hartenberg convention [Denavit and Hartenberg, 1955].

Of course, this model provides a coarse approximation of the real arm kinematics, since it does not consider the elasticity introduced by steel tendons and plastic belts that are used for torque transmission; therefore, the displayed trajectories do not precisely match the real 3D cartesian trajectories. Anyway, the following plots give some useful indications to describe the system behavior in general.
Figure 5.16 shows a 3D reaching trajectory projected on the three bidimendional planes. The green star is the starting point, the red triangle the ending point. It can be noticed the switch from ballistic to Jacobian-based control, which happens in the proximity of the target. It is evident that the ballistic motion produces a curve trajectory in cartesian space, while the Jacobian-based correction is almost rectilinear.

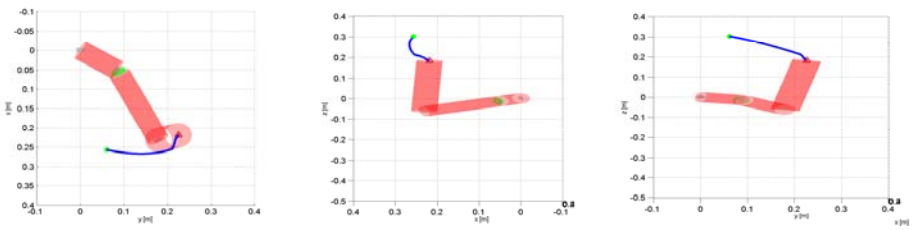


Figure 5.16: Complete reaching 3D trajectory in cartesian space. The 3D motion is projected on the three bidimensional planes.

Anyway, almost-linear paths are also observed in the visual trajectories, as shown in figure 5.17. This data directly comes from eyes cameras measurements and so they carry a more relevant information (i.e. the 3D plots in cartesian space may be affected by inaccuracies in the kinematic model).
In figure 5.18 we can appreciate how the hand velocity changes during the reaching movement. Again, trajectory points are sampled at a fixed rate, so the distance between each couple of points is proportional to the actual hand velocity. Also here, the green star is the starting point, the red triangle the ending point. A black dot marks the switch from open-loop control to closed-loop control, while a pink cross indicates the end-effector target position for the open-loop (ballistic) controller. The hand velocity is higher during the ballistic motion, with a peak in the middle of the movement[1], and smoothly decreases as the target becomes

[1]In section 6.3 more details about joints velocities during ballistic movements are given.

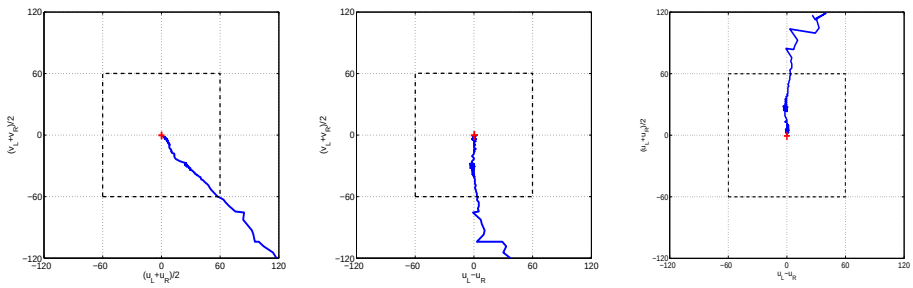


Figure 5.17: Complete reaching 3D trajectory in visual space. The 3D motion is projected on the three bidimensional planes. The projected trajectories become straight as soon as the Jacobian based correction starts (inside the highlighted central square).

closer, during the closed-loop control. A last figure (5.19) depicts the hand tra-

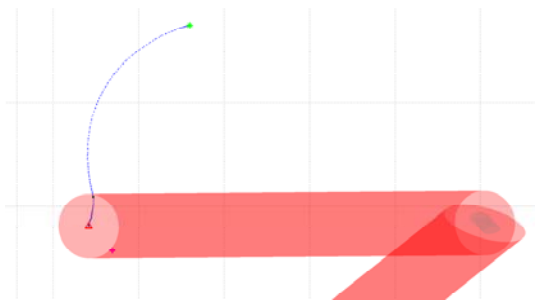


Figure 5.18: Complete reaching trajectory. The green star is the starting point, the red triangle the ending point. The black dot marks the switch from open-loop control to closed-loop control. The pink cross indicates the end-effector target position for the open-loop (ballistic) controller.

jectories in the 3D cartesian space during a sequence of reaching actions. Again, green stars are the starting points of each reaching movement (the bigger star marks the beginning of the first movement of the sequence), red triangles are the ending points (the bigger one indicates the end of the last movement) and blue crosses are the targets of the ballistic motions.

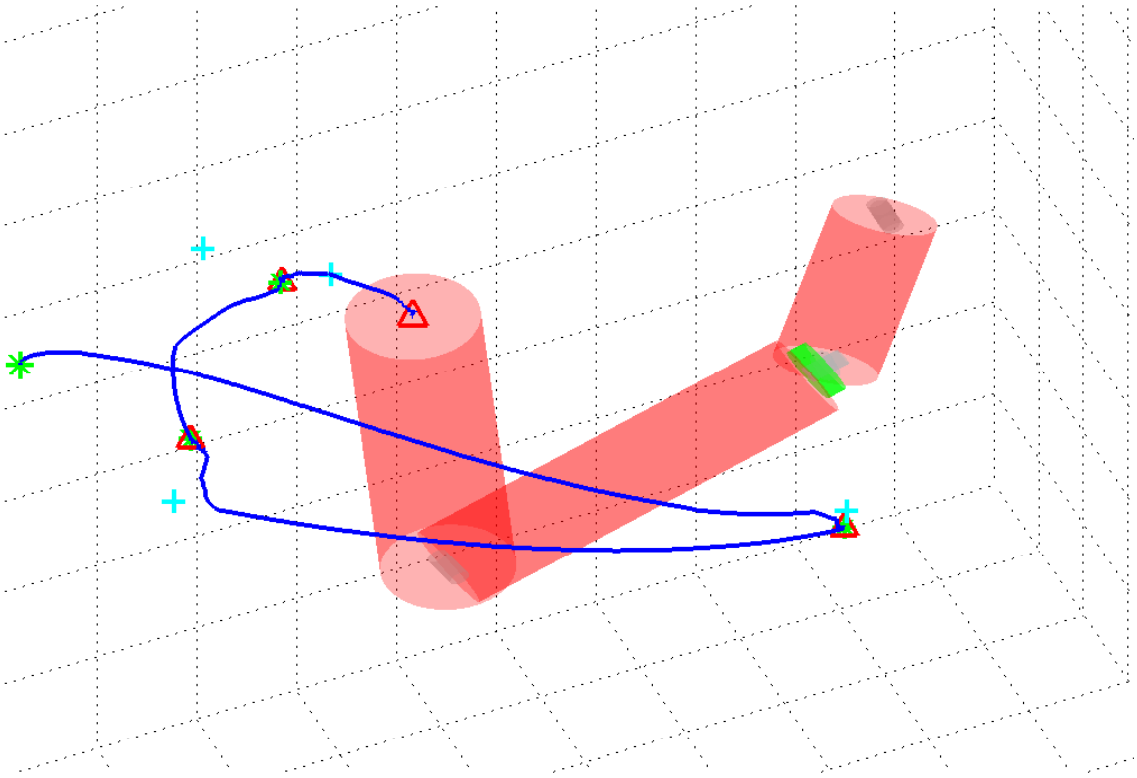


Figure 5.19: Sequence of reaching trajectories. The green stars are the starting points, the red triangles the ending points. The blue crosses indicate the end-effector target position for the open-loop (ballistic) controller.

5.3.4 Reachable space

Figures 5.20 and 5.21 present the data gathered by the robot during the excution of reaching actions, used as training data for learning a representation of the reachable space employing a RFWR network. In particular, the first figure shows the data input distribution, while the second one displays the data in the three bidimensional projections of the 3D space of head configuration, marking reachable points with red dots and points that have not been reached with blue stars. In general, it can be seen that the robot cannot reach for most targets fixated with a low value of vergence (indeed, the larger the vergence the closer the fixated object) and performs better reaching when the head yaw is around its central values (corresponding to the left part of the workspace). In particular, when the neck yaw is close to 90 deg (i.e. the robot is looking straight in front of itself) the robot can hardly reach for the fixated object due to arm joints limits (in particular, θ_{sy}, shoulder abduction, approaches its lower limit), especially for head pitch values lower than zero (i.e. robot looking up). Figure 5.22 depicts

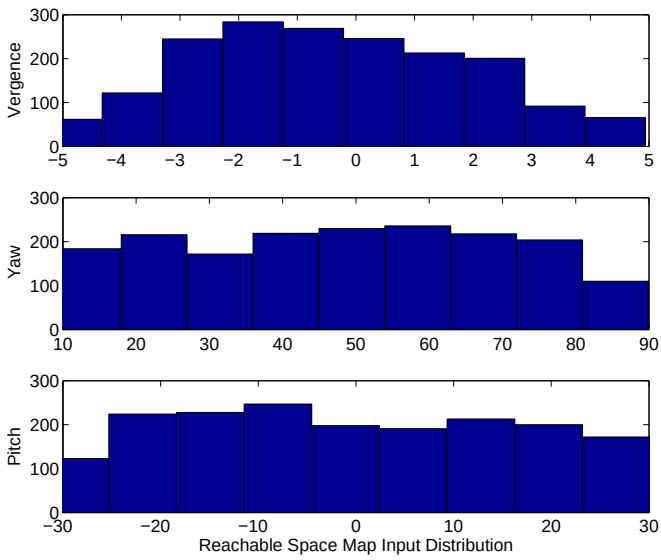


Figure 5.20: Distribution of the reachable space map training samples, concerning the input space (head configuration).

a 3D visualization of the learned reachable space after 1400 samples have been

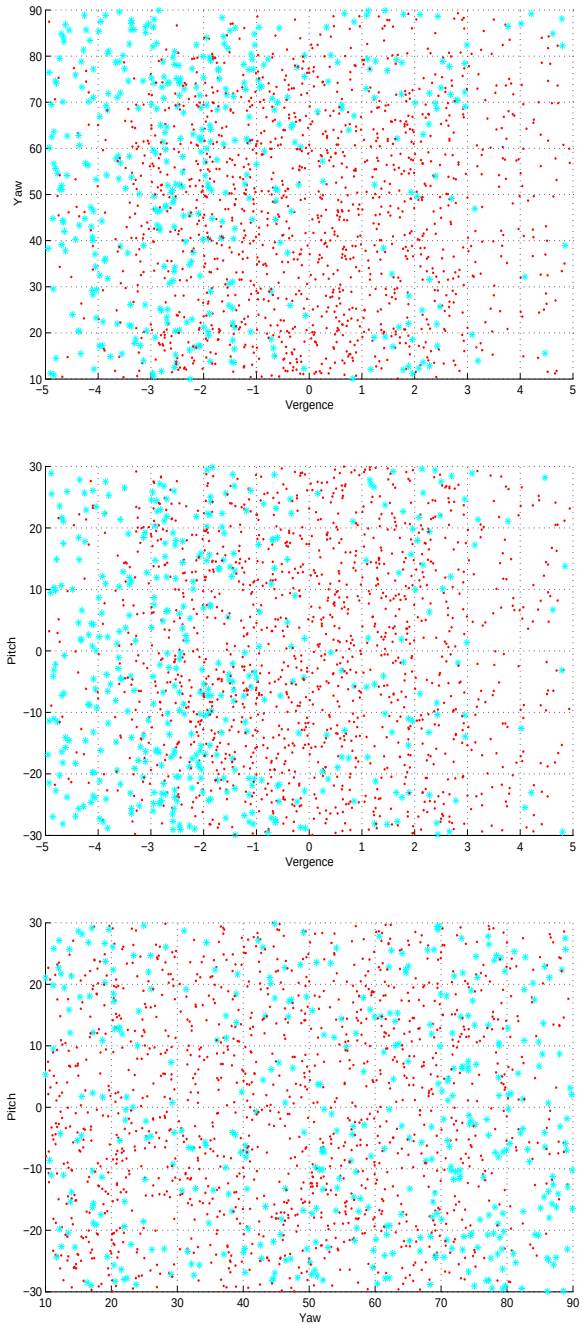


Figure 5.21: Reachable space map training data. Red dots are reachable points, blue stars are "unreachable" points (including points that have not been reached but are reachable in theory). Points are plotted on three bidimensional projection of the 3D input space (head configuration).

gathered and used for training, while figure 5.23 shows projections of the same map on the vergence/yaw plane, considering different values of pitch orientation, precisely $\theta_p = -30; -15; 0; 15; 30$. Dark red means high values of the map output (high probability of the point to be reachable). Generally, it can be noticed that the map output increases with increasing values of vergence (i.e. fixated point closer to the robot eyes). Morover, the output is complexively higher in the central part of yaw axis, far from robot physical limits; in particular, for yaw values close to 90 deg the map output is typically low, especially if vergence is negative. If we analyze the robot physical structure (indeed, similar to the human one), it is clear that considering a certain distance of the target from the eyes (encoded by vergence) such a target it is more probably reachable if the neck is bent forward (positive pitch) than backward (negative pitch), being the shoulder positioned below the head. Noticeably, increasing the pitch value the map output points become generally higher. This is also evident in figure 5.24, where head yaw is 90 deg. In the following, the learning performances over time (i.e. with increas-

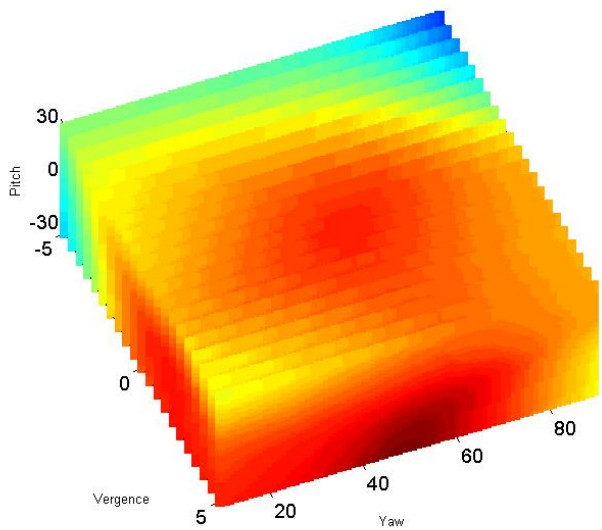


Figure 5.22: Learned map of the reachable space, after training with 1400 samples. 3D visualization in head configuration space. Dark red means high probability to be reachable, blue means low probability.

ing number of training samples) are evaluated. In particular, every time a new

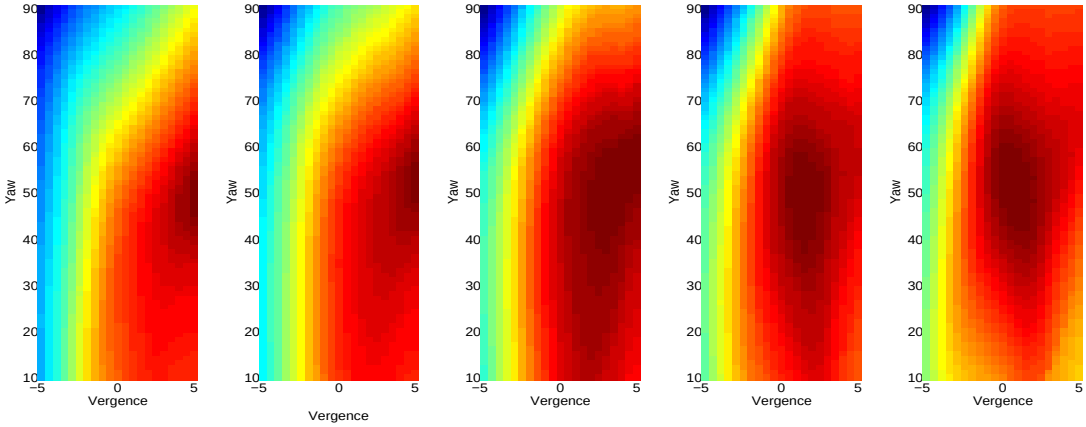


Figure 5.23: Learned map of the reachable space, after training with 1400 samples. Projections on the vergence/yaw plane, considering different values of pitch orientation, -30,-15,0,15,30. Dark red means high probability to be reachable, blue means low probability.

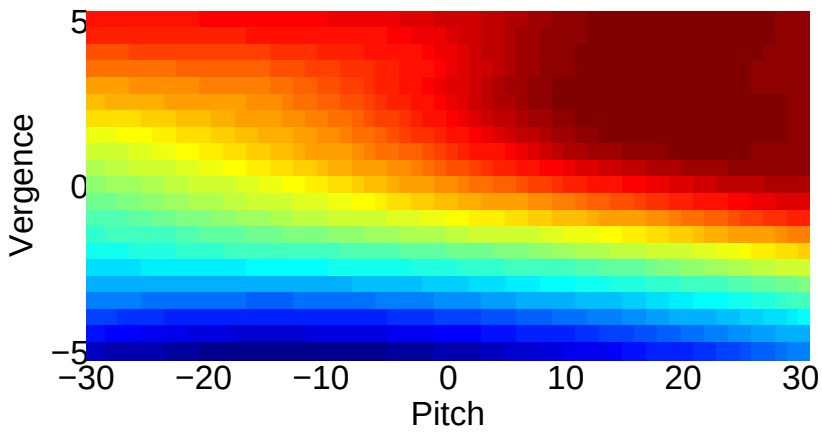


Figure 5.24: Learned map of the reachable space, after training with 1400 samples. Projections on the vergence/pitch plane, with $yaw = 90$. Dark red means high probability to be reachable, blue means low probability.

sample is gathered and used for training, the RFWR network is queried with the same test point (i.e. head configuration) in order to describe the evolution of the network response.

In figure 5.25 we used the test point $\mathbf{q}_{head} = [-1.5\ 30.68\ \ -3.81]$, which corresponds to a reachable point. Anyway, the reaching trial 269 was directed toward this point, and the robot was not able to accomplish it (probably the reaching controller at that time was not good enough); this caused a decrease of the net output. Nevertheless, the subsequent reaching trials toward surrounding regions of the space were successful, and the output value for the test point increases again toward the correct value.

A similar test is shown in figure 5.26, where three test points corresponding to "unreachable" locations in space have been used, respectively $\mathbf{q}_{head} = [-4.5\ 85\ 0]$, $\mathbf{q}_{head} = [-4.5\ 85\ \ -5]$, $\mathbf{q}_{head} = [-4.5\ 85\ 5]$. Note that the three points are relatively close one to each other, and therefore the trends of their estimation are similar.

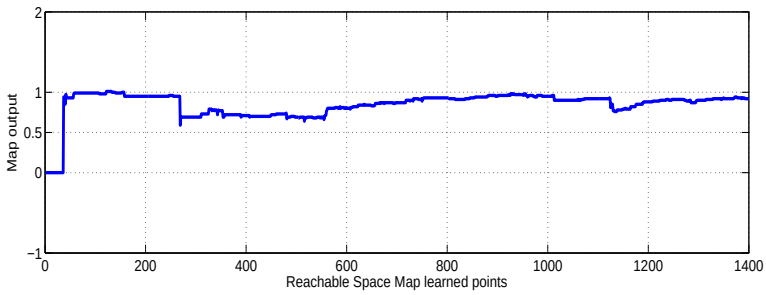


Figure 5.25: Evaluation of a reachable point using the reachable space map, trained with increasing number of samples.

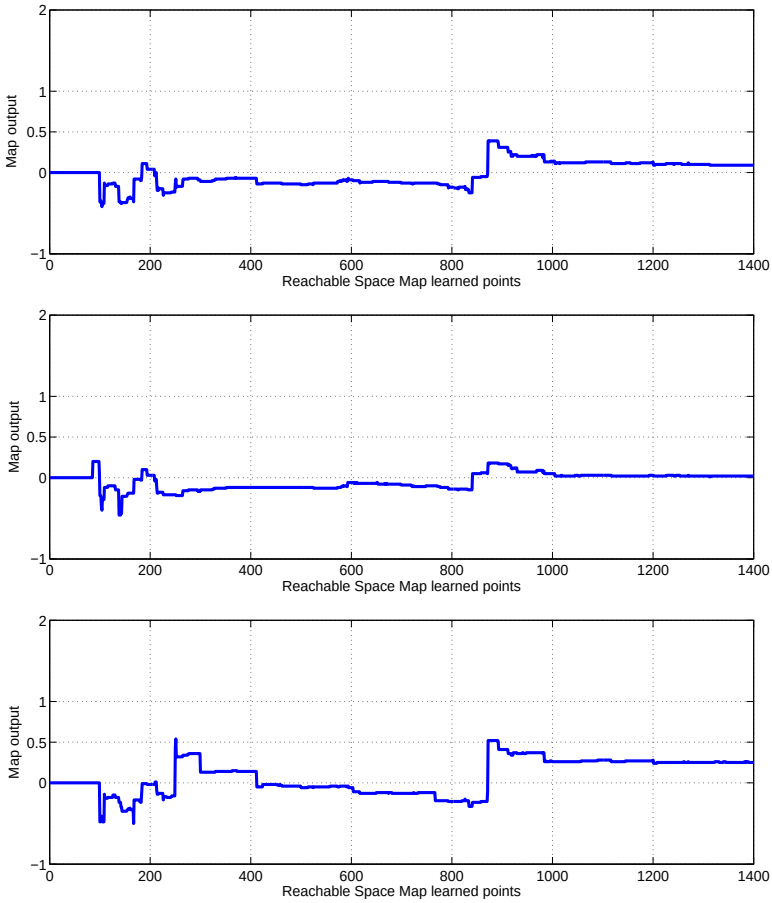


Figure 5.26: Evaluation of three "unreachable" points using the reachable space map, trained with increasing number of samples.

6

Interaction with the environment

The acquisition of gazing and reaching skills enables the robot to start interaction with the surrounding environment; the importance of this interaction for the cognitive development of the system has been sufficiently stressed in the previous sections. While trying to reach for an object (and learning how to do it properly), the robot might come in contact with the external environment or even with its own body.

We have already discussed in section 3.1 how compliance is important in order to avoid damaging the robot and the surrounding environment (physical objects and/or interacting agents) in such a situation. If passive compliance can be realized by introducing elastic components in the mechanical system, then special sensors are usually applied to detect contact situations and achieve active compliant behaviors. One way to achieve this is to exploit force sensors [De Luca et al., 2006; Shinya and Kazuhiro, 2003].

Classical approaches to manipulation exploit a force/torque (FT) sensor placed on the end-effector, where most of the interactions occur. External forces acting on the other parts of the arm, however, cannot be measured with this configuration. Furthermore, it may not be feasible to put the sensor on the end-effector, due to its excessive size or weight.

An alternative solution is to place the sensor at the base of the manipulator or along the kinematic chain (e.g. after the shoulder) [Liu et al., 1998; Lu et al., 2005]. This solution allows the robot to detect interaction with the environment not only on the end-effector (e.g. voluntarily touching or grasping an object), but

on the whole arm (e.g. hitting unexpected obstacles, being stopped by a human agent during motion). In this case, however, the FT sensor measures both external and internal forces, the latter being the ones depending on gravitational, Coriolis and inertial forces. In order to accurately detect the external contribution of the forces, the manipulator dynamics must be compensated for, i.e. the internal forces must be known, modeled or estimated.
In this chapter we discuss the estimation of internal forces and torques, which is fundamental in order to retrieve a measure of the external ones. We present some preliminary experimental results in which we use this model to control the interaction with the external environment, exploiting the integration of force, tactile, visual and proprioceptive inputs.

6.1 Internal forces estimation

Multiple approaches can be used for the estimation of these internal forces. Firstly, the functional estimation can be done using an analytical model describing the physics of the system, or at least its most significant properties. Model-based estimation strongly relies on the availability of a (mathematical) model of the robot [Sciavicco and Siciliano, 1996], and is recommended only if the kinematic and dynamic parameters are known or identifiable with high accuracy. To this purpose, rigid multi-body dynamic modeling is generally used and some or all the parameters are identified [Swevers et al., 1997] in order to improve the model accuracy. Within this context, the overall model accuracy is primarily limited by the (potentially nonlinear) effects which the model does not explicitly take into account (e.g. gearbox backlash).
Alternatively, supervised machine learning approaches can be used to approximate the internal dynamic model from a set of training examples. This approach may be preferred when explicitly modeling all possible nonlinear effects is cumbersome [Ting et al., 2006]. The main drawback of supervised learning methods is the need for collecting a rich and significant training set. Furthermore, it may be necessary to perform the training phase offline, due to the high computational requirements of these learning methods. In contrast, the model-based approach only needs to identify a small set of significant parameters; this identification

technique requires much fewer data and computational resources and therefore can typically be performed efficiently online.
In this section, we investigate an analytical model and two supervised machine learning methods (Least Squares Support Vector Machines and Feedforward Neural Networks) for the estimation of internal forces in the James robotic arm, which is equipped with a six-axis FT sensor inside the kinematic chain. It seems reasonable to assume, however, that the results can be generalized to similar problems in robotics (i.e. problems related to the estimation of the dynamical parameters of a kinematic chain).
Firstly, we focus our attention on the amount of training data necessary to obtain accurate predictions. The qualitative measure for the prediction is the average Normalized Mean Square Error (NMSE) for the forces and torques in three dimensions. In our framework, the minimum amount of external forces that the robot can detect is proportional to the magnitude of this estimation error; this value is critical in order to have safe interaction with the environment. We expect machine learning methods to benefit from larger data sets; on the other hand, model based techniques should be more insensitive to the size of the training set.
Secondly, we pose our interest in understanding how the type of supplied data influences the estimation error. In particular, we verify empirically the usefulness of velocity and acceleration measurements when estimating (relatively complex) dynamical models. This issue is particularly important in the field of humanoid robotics, where smooth motions are usually preferred. As a consequence, velocities and accelerations need to satisfy some smoothness requirements, which prevent data from being completely random[1]. Therefore, successfully exploiting velocity and acceleration data is difficult, especially without specific sensors dedicated to their measurement. At last, we analyze the effect of sampling distribution on the generalization performance. This analysis can give information about the way training data should be gathered or subsampled to practical dimensions from a larger training set.

[1]Typical identification techniques make strong assumptions on the supplied data set. Machine learning techniques assume sufficiently distributed samples that cover the variability of the underlying function. Model based approaches assume persistently exciting conditions (see [Krzysztof, 1998] for a definition).

All these investigations are carried on considering an offline learning scenario. In the last part of this section we discuss how the internal forces can be estimated exploiting online learning, and we provide experimental data in which the trend of NMSE during learning with RFWR and LWPR is shown.
Most of the content of this section has been published in [Fumagalli et al., 2010].

6.1.1 Problem formulation

As explained before, the FT sensor measures both internal and external forces, the latter being the ones to be determined for interaction control purpose (e.g. obstacle detection and avoidance). The whole arm surface is taken into account for possible contact points (so the contact may happen in any point on the arm, not only on the end-effector). In the following we will discuss the retrieval of the external forces and the consequent need to estimate the internal ones.
Let us consider an open kinematic chain with n degrees of freedom. Let $\vec{q} \in \mathbb{R}^n$ be the generalized coordinates describing the pose of the kinematic chain. The FT sensor measurement will be denoted $\vec{x} = [\vec{f}^\top, \vec{\tau}^\top]^\top \in \mathbb{R}^6$. As previously said, this quantity contains both external and internal forces $\vec{f} \in \mathbb{R}^3$ and torques $\vec{\tau} \in \mathbb{R}^3$. Specifically we have:

$$\vec{x} = \vec{x}_I + \vec{x}_E \ , \tag{6.1}$$

where $\vec{x}_I$ and $\vec{x}_E$ refer to the internal and external forces/torques, respectively. More precisely, neglecting the effect of the elasticity of the transmissions and defining $\vec{f}_E, \vec{\tau}_E$ as the external forces and torques applied at the contact point, equation (6.1) can be expanded as follows (see [Sciavicco and Siciliano, 1996] for details on the derivations):

$$\begin{bmatrix} \vec{f} \\ \vec{\tau} \end{bmatrix} = \underbrace{M(\vec{q})\ddot{\vec{q}} + C(\vec{q}, \dot{\vec{q}})\dot{\vec{q}} + G(\vec{q})}_{\vec{x}_I} + \underbrace{T(\vec{q}, \vec{d}) \begin{bmatrix} \vec{f}_E \\ \vec{\tau}_E \end{bmatrix}}_{\vec{x}_E} \ , \tag{6.2}$$

where M, C and G are the inertial, Coriolis and gravity matrices of the dynamic system equations, and T is a roto-translation matrix describing the transformation of the external forces from the contact point reference frame to the sensor reference frame, with $\vec{d}$ being the distance vector from the contact point to the

sensor.
Whenever the robot interacts with an external object, a non-null external force component $\vec{x}_E$ arises: in order to detect a collision or a contact, $\vec{x}_E$ must be identified from the sensor measurements $\vec{x}$. Practically, the identification can be performed by subtracting the internal forces ($\vec{x}_I$) from the measured ones ($\vec{x}$):

$$\vec{x}_E = \vec{x} - \vec{x}_I \ , \tag{6.3}$$

which yields an indirect measurement of the external forces and torques[1]. Then, the vector $\vec{x}_I$ must be computed from the model, or derived from experimental data. When the robot moves freely in its workspace, the sensor only perceives the internal components of forces and torques (i.e. $\vec{x}_I = \vec{x}$). These components only depend on position, velocity and acceleration of the joints. The problem of retrieving $\vec{x}_E$ is therefore reduced to the estimation of the internal forces and torques, i.e. the mapping from $\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}}$ to $\vec{f}, \vec{\tau}$:

$$\vec{x}_I = f(\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}}). \tag{6.4}$$

6.1.2 Proposed approaches

Two distinct ways to identify $f(\cdot)$ in equation (6.4) are: (1) deriving it analytically, (2) approximating it using a set of examples (i.e. machine learning). In the latter case, the learning algorithm is agnostic to the underlying dynamics model that is used to produce the examples. One advantage of this approach is that nonlinear effects do not need to be explicitly modeled, as these are learned implicitly by the algorithm.
It is worth discussing some issues related to the noise effecting the measurement equation (6.4) which is at the base of all the proposed identification methods. Among the different contributions, the F/T sensor itself is a primary source of noise (see the specifications in [ATI, 1982]) but it is not the only one. As a matter

[1]Notice that $\vec{x}_E$ does not correspond to the real external forces $\vec{f}_E$ and torques $\vec{\tau}_E$, but to their projection on the sensor, i.e. $\vec{x}_E = T(\vec{q}, \vec{d})[\vec{f}_E^\top, \vec{\tau}_E^\top]^\top$. To retrieve the real external forces and torques, we must also know the distance $\vec{d}$ from the contact point to the sensor. In the future, we plan to mount a full body sensing skin [Cannata et al., 2008], which will provide the necessary feedback to detect the contact location.

of fact, also the velocity and acceleration measurements are subject to numerical inaccuracies, as these are computed using first and second order numerical differentiation of the position measurements. These derivatives are estimated based on the difference between the samples at time t and $t - W$, i.e.

$$\dot{\vec{q}}_t = \frac{\vec{q}_t - \vec{q}_{t-W}}{W\Delta T} \ , \ \ddot{\vec{q}}_t = \frac{\dot{\vec{q}}_t - \dot{\vec{q}}_{t-W}}{W\Delta T} \quad \text{for } t = W, \ldots, \infty \ . \tag{6.5}$$

This computation is performed on the DSP boards embedded in the robot arm at 1 kHz rate, i.e. $\Delta T = 1$ ms. The window length W has been set to 35, i.e. $W\Delta T = 35$ms[1]. Other sources of noise include communication delays effecting the synchronization of sensory measurements and reliability of the position measurements in presence of elasticity in the actuation design (elastic tendons and rubber transmission belts). The complex interaction of these various sources of noise makes it very difficult to characterize the overall system noise and therefore a complete analysis will be left outside the scope of the current work.
In the following, we will detail the model-based approach and two machine learning algorithms, namely Least Squares Support Vector Machines and Neural Networks.

Model-Based Approach

Let us consider a robotic manipulator with n degrees of freedom and links, and a force/torque sensor located in the middle of the kinematic chain, immediately after one of the joints. As already pointed out, the sensor will measure both internal ($\vec{x}_I$) and external ($\vec{x}_E$) force/torque component acting on the following links. Anyway, hereinafter, we assume that the FT sensor measures only the internal forces, i.e. $\vec{x}_E \equiv \vec{0}$. Therefore, (6.2) can be written as:

$$\begin{bmatrix} \vec{f} \\ \vec{\tau} \end{bmatrix} = M(\vec{q})\ddot{\vec{q}} + C(\vec{q}, \dot{\vec{q}})\dot{\vec{q}} + G(\vec{q}) \ . \tag{6.6}$$

Starting from this formulation, we derive a model based approach for estimating the parameters in (6.6). In order to tune this model, a set of parameters that best fits the force/torque acquisition, given a certain data set of joint positions

[1]This window length has been chosen specifically to low-pass filter the position measurements and the computed velocities, whilst maintaining sufficient accuracy.

$\vec{q}$, velocities $\dot{\vec{q}}$ and accelerations $\ddot{\vec{q}}$, needs to be found. Equation (6.6) is written as the linear product of a matrix $D(\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}})$ and a vector $\vec{\eta}$ (see Krzysztof [1998] for details). The matrix $D(\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}})$ depends solely on the joint positions, velocities and accelerations, whereas $\vec{\eta}$ contains the dynamical parameters that we would like to estimate. Practically:

$$\vec{x} = \begin{bmatrix} \vec{f} \\ \vec{\tau} \end{bmatrix} = D(\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}})\vec{\eta} \quad , \tag{6.7}$$

where $\vec{\eta}$ has a complex structure that can be formalized as follows:

$$\vec{\eta} = [\psi, \ \Psi]^\top \quad .$$

The row vectors ψ and Ψ depend only on the system dynamical parameters and have the following structure:

$$\psi = \left[\begin{array}{ccc} m_1\varphi & \cdots & m_n\varphi \end{array} \right] \in \mathbb{R}^{n^\psi} \tag{6.8}$$

$$\Psi = \left[\begin{array}{ccc} \vec{r}_1 & \cdots & \vec{r}_n \end{array} \right] \in \mathbb{R}^{n^\Psi} \quad , \tag{6.9}$$

where

$$\varphi = \left[\begin{array}{cc} \mathbf{l}_1^\top \cdots \mathbf{l}_n^\top, & \mathbf{c}_1^\top \cdots \mathbf{c}_n^\top \end{array} \right] \in \mathbb{R}^{n^\varphi} \tag{6.10}$$

$$\vec{r}_i = \left[\begin{array}{cc} \vec{s}_{i,1} \cdots \vec{s}_{i,n}, & I_i^1 \cdots I_i^6 \end{array} \right] \tag{6.11}$$

$$\vec{s}_{i,j} = \left[\begin{array}{ccc} m_i\varphi_j^2 \ m_i\varphi_j\varphi_{j+1} & \cdots & m_i\varphi_j\varphi_{n_\varphi} \end{array} \right] \quad . \tag{6.12}$$

For each link $i = 1, \ldots, n$, $\vec{l}_i \in \mathbb{R}^3$ is the vector representing the lengths of the link in the x, y and z directions with respect to the previous joint's reference frame, $\vec{c}_i \in \mathbb{R}^3$ is the vector of the center of mass of each link, with respect to the same reference frame[1]. Further, $m_i \in \mathbb{R}$ and $I_i^k \in \mathbb{R}$ are the mass and inertial parameters of each link for $k = 1, \ldots, 6$.
Interestingly, equation (6.7) can be further simplified to the form $\vec{x} = \hat{D}(\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}})\vec{\phi}$, where $\vec{\phi}$ is the minimum set of identifiable parameters, i.e. a linear combination

[1]Each kinematic chain link has an associated reference frame, defined by the Denavit-Hartenberg convention [Denavit and Hartenberg, 1955] . All the dynamic and kinematic quantities of each link (e.g. center of mass, inertia, lengths) refer to the associated reference frame.

of the elements of vector $\vec{\eta}$ (see [Krzysztof, 1998] for details).
The vector $\vec{\phi}$ of the system dynamical parameters can be often retrieved from an accurate model of the robot (e.g. CAD drawings), but this procedure is typically neither feasible nor accurate. Different ways for identifying the dynamic parameters can be found in the literature, but their discussion is out of the scope of this work (the interested reader should refer to [Krzysztof, 1998; Liu et al., 1998]). Here we focus on a technique based on a weighted linear least squares solution.
Let us define a weighting diagonal matrix ω containing the variances of each component of force (f_x, f_y, f_z) and torque (τ_x, τ_y, τ_z):

$$\omega = \begin{bmatrix} \frac{1}{\sigma^2_{f_x}} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma^2_{f_y}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma^2_{\tau_z}} \end{bmatrix} . \tag{6.13}$$

At the ith time instant, we measure the system position ($\vec{q}_i$), velocity ($\dot{\vec{q}}_i$) and acceleration ($\ddot{\vec{q}}_i$) and the associated force sensor output $\vec{x}_i$. After ℓ time samples, a possible estimation for the vector $\vec{\phi}$ of dynamical parameters is given by the vector $\vec{\phi}^\circ$ which minimizes the (weighted) norm of the error vectors $(\vec{x}_i - \hat{D}_i\vec{\phi})$, i.e.:

$$\vec{\phi}^\circ = \arg\min_{\vec{\phi}} \sum_{i=1}^{\ell} (\vec{x}_i - \hat{D}_i\vec{\phi})^\top \omega (\vec{x}_i - \hat{D}_i\vec{\phi}) \ . \tag{6.14}$$

where we defined $\hat{D}_i = \hat{D}(\vec{q}_i, \dot{\vec{q}}_i, \ddot{\vec{q}}_i)$. The explicit solution is given by:

$$\vec{\phi}^\circ = \Delta^\dagger_\Omega \vec{Y} = \left[\Delta^\top \Omega \Delta\right]^{-1} \Delta^\top \Omega \vec{Y} \ , \tag{6.15}$$

where $\Omega = diag(\omega)$ and

$$\Delta = \begin{bmatrix} \hat{D}_1 \\ \hat{D}_2 \\ \vdots \\ \hat{D}_\ell \end{bmatrix} \quad \vec{Y} = \begin{bmatrix} \vec{y}_1 \\ \vec{y}_2 \\ \vdots \\ \vec{y}_\ell \end{bmatrix} . \tag{6.16}$$

Given the discussion above, learning the optimal parameters value $\vec{\phi}^\circ$ consists in a matrix inversion. With simple algebraic simplifications, it can be proved that

the dimension of the matrix to be inverted does not depend on the number of acquired data but only on the dimension of the vector $\vec{\phi}$. Similarly, once the model has been trained, the model prediction (the prediction of $\vec{x}$ given $\vec{q}$, $\dot{\vec{q}}$ and $\ddot{\vec{q}}$) consists in evaluating $\hat{D}(\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}})$ and the product $\hat{D}(\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}})\vec{\phi}^{\circ}$. Therefore, the computational complexity of the prediction depends mainly on the evaluation of the matrix $\hat{D}$ (in our example represented by ~ 1700 multiplications, ~ 700 sums and 8 sine/cosine evaluations).

Least Squares Support Vector Machines for Regression

Least Squares Support Vector Machines (LS-SVMs) belong to the class of kernel methods, which use a positive definite kernel function to estimate a linear approximator in a (usually) high-dimensional feature space [Suykens et al., 2002]. Its formulation shares similarities with the Support Vector Machine for Regression (SVR) [Smola and Schölkopf, 2004]. Let us define the data set $\mathcal{S} = \{\vec{x}_i, y_i\}_{i=1}^{\ell}$, where inputs $\vec{x}_i \in \mathbb{R}^n$ and corresponding outputs $y_i \in \mathbb{R}$ for $i = 1, \ldots, \ell$. LS-SVM estimates a linear decision function of the form $f(\vec{x}) = \langle \vec{w}, \phi(\vec{x}) \rangle + b$, where b is a bias term and $\phi(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}^f$ maps samples from the input space into a (usually) high-dimensional feature space. The weight vector $\vec{w}$ and bias b are chosen such that both the squared norm of $\vec{w}$ and the sum of the squared errors $\epsilon_i = y_i - f(\vec{x}_i)$ are minimized. This is described by the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\vec{w}\|^2 + \frac{1}{2}C\sum_{i=1}^{\ell}\epsilon_i^2 \\ \text{subject to} \quad & y_i = \langle \vec{x}_i, \vec{w} \rangle + b + \epsilon_i \qquad \text{for } 1 \leq i \leq \ell \ , \end{aligned} \tag{6.17}$$

where C is a regularization constant. Standard application of the Lagrange method yields the dual optimization problem [Suykens et al., 2002]:

$$\text{maximize} \quad \frac{1}{2}\|\vec{w}\|^2 + \frac{1}{2}C\sum_{i=1}^{\ell}\epsilon_i^2 - \sum_{i=1}^{\ell}\alpha_i\left(\langle \vec{x}_i, \vec{w} \rangle + b + \epsilon_i - y_i\right) \ . \tag{6.18}$$

Here $\alpha_i \in \mathbb{R}$ are the Lagrange multipliers associated with each sample. Using this dual formulation, the decision function can be rewritten as $f(\vec{x}) = \sum_{i=1}^{\ell}\alpha_i \langle \phi(\vec{x}_i), \phi(\vec{x}) \rangle + b$. One particular advantage of this expansion is that the solution is described in terms of inner products with respect to the training samples $\vec{x}_i$. Hence, a kernel function $k(\vec{x}_i, \vec{x}_j) = \langle \phi(\vec{x}_i), \phi(\vec{x}_j) \rangle$ can be used

to implicitly map the data into the feature space. Given a kernel matrix $K = \{k(\vec{x}_i, \vec{x}_j)\}_{i,j=1}^{\ell}$, the solution to the optimization problem in (6.18) is given by a system of linear equations:

$$\begin{bmatrix} \vec{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} K + C^{-1}I & \vec{1} \\ \vec{1}^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} \vec{y} \\ 0 \end{bmatrix} \quad . \tag{6.19}$$

Note that solving the LS-SVM optimization problem reduces to a $(\ell+1) \times (\ell+1)$ matrix inversion, which in return can be solved efficiently using Cholesky decomposition [Cawley, 2006]. Another advantage of LS-SVM over other kernel methods (e.g. SVR), is that the Leave-One-Out (LOO) error can be computed exactly using a single training run on the complete data set [Cawley, 2006]. It is important to note that the final generalization performance of the LS-SVM is strongly dependent on the selection of both C and the kernel function. For our experiments, we consider the commonly used Radial Base Function (RBF) kernel $k(\vec{x}_i, \vec{x}_j) = \exp(-\gamma \|\vec{x}_i - \vec{x}_j\|^2)$, where parameter γ tunes the radius of the Gaussian. A grid search on the range $C \in [2^0, 2^1, \ldots, 2^{16}]$ and $\gamma \in [2^{-11}, 2^{-10}, \ldots, 2^1]$ is used to select "optimal" hyperparameters, where the generalization performance of each configuration is estimated using the LOO error on the training set. Furthermore, we considered the cases that $\vec{x} = \{\vec{q}, [\vec{q}, \dot{\vec{q}}], [\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}}], [\vec{q}, \ddot{\vec{q}}]\}$. As the output y is limited to scalar values, a distinct machine has to be trained for each output dimension, such that $y = \{f_x, f_y, f_z, \tau_x, \tau_y, \tau_z\}$.

Though LS-SVM has several advantageous properties with respect to SVM, one apparent disadvantage is that it does not produce sparse models. Input samples $\vec{x}_i$ can only be removed from the kernel expansion when $\alpha_i = 0$, which in return is only the case if $\epsilon_i = 0$. As a result, on practical problems all input samples will be included in the model. This reflects negatively on the prediction time. For m output dimensions and assuming the RBF kernel function, the prediction of an n dimensional input vector requires $m\,(\ell(n+1)+1)$ sums, $m\ell(n+2)$ products, and $m\ell$ exponentials; where ℓ is the size of the data set the m distinct machines has been trained on.

The scope of this analysis is limited to batch learning on small to medium sized data sets. Nguyen-Tuong et al. [Nguyen-Tuong et al., 2009] have shown that – on a similar learning problem – Gaussian Processes Regression (GPR) commonly

outperforms other methods in this particular context. It is worth noting that the performance of LS-SVM can be expected to be similar to GPR, as both methods share some similarities in the approximation function[1].

Neural Networks

Lastly, a multiple input - multiple output one-hidden-layer (OHL) feed-forward neural network (NN) is chosen as the second machine learning method, for its generalization and approximation capabilities [Hornik et al., 1989], and de-noising property when dealing with experimental data. More specifically, we constrain the approximation function to take on a fixed, parameterized structure, that is a ν "neurons", n inputs, m outputs neural network, $\hat{\vec{\mu}}(\cdot, \vec{w})$, with sigmoidal activation functions σ in both hidden and output layer[2]:

$$\hat{\vec{\mu}}(\tilde{\vec{x}}, \vec{w}) = col\left(\tilde{\vec{\sigma}}_j \left[\sum_{h=1}^{\nu} c_{hj}\sigma(\tilde{\vec{x}}, \vec{\kappa}_h) + b_j\right], j = 1, \ldots, m\right) \tag{6.20}$$

where $\hat{\vec{\mu}}(\cdot, \vec{w}) : \mathbb{R}^n \times \mathbb{R}^W \mapsto \mathbb{R}^m$, $c_{hj}, b_j \in \mathbb{R}, \vec{\kappa}_h \in \mathbb{R}^{n+1}, j = 1, \ldots, m$, being ν the number of *neurons* constituting the network.

The vector $\vec{w} \in \mathbb{R}^W, W = (n+1)\nu + (\nu+1)2m$ collects all the parameters to be optimized. The notations $\tilde{\vec{\sigma}}$ and $\tilde{\vec{x}}$ account for the output and input normalization.[3]

Furthermore, we trained four different type of networks, with $\vec{x} = \{\vec{q}, [\vec{q}, \dot{\vec{q}}], [\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}}], [\vec{q}, \ddot{\vec{q}}]\}$ and $n = \{4, 8, 12\}$ respectively, for different number of neurons $\nu = 5, 20, 50, 100, 150$ and different training sets. The number of outputs was always fixed to 6 (forces and torques).

The training algorithm is based on the well known Levenberg-Marquardt (LM) algorithm [Levenberg, 1944; Marquardt, 1963]. The criterion for training the network (that is to find the optimal parameters $\vec{w}^\circ$) is to minimize the mean square

[1]The main differences between both methods are that LS-SVM includes a bias term and requires less assumptions on the distribution of the data.

[2]We chose a sigmoidal output layer (instead of a classical linear output layer) since it naturally generates bounded values within a specific range, which are consistent with the output ranges after data normalization. This choice allows to remove signal constraints and not to take care of the possibility that the network generates inconsistent values.

[3]The input variables are normalized from their original range to $[-1, 1]$, while the network outputs are scaled from $[-1, 1]$ (the output range of a sigmoidal tanh-based neural network) to the forces and torques real ranges.

error between the estimated and the measured data:

$$\text{minimize} \quad \Phi(\vec{w}) = \frac{1}{2}\sum_{i=1}^{\ell} \vec{\epsilon}_i^{\top}(\vec{w})\vec{\epsilon}_i(\vec{w}) \ , \tag{6.21}$$

where $\vec{\epsilon}_i(\vec{w}) = \vec{y}_i - \hat{\vec{\mu}}(\tilde{\vec{x}}_i, \vec{w})$ is the error between the measured and the predicted data. Once all the partial derivatives of the error function Φ are back-propagated, the weights update equation is applied:

$$\vec{w}_{k+1} = \vec{w}_k - [J^{\top}(\vec{w}_k)J(\vec{w}_k) + \mu I]^{-1} J^{\top}(\vec{w}_k)\vec{\epsilon}(\vec{w}_k) \ , \tag{6.22}$$

where $\vec{\epsilon}(\vec{w}_k) = [\vec{\epsilon}_0(\vec{w}_k), \ldots, \vec{\epsilon}_{N-1}(\vec{w}_k)]$, and $J(\vec{w}_k) \in \mathbb{R}^{N\times W}$ is the Jacobian matrix of the errors with respect to the parameters of the NN:

$$J(\vec{w}) = \begin{bmatrix} \frac{\partial\vec{\epsilon}_0}{\partial w_0} & \frac{\partial\vec{\epsilon}_0}{\partial w_1} & \cdots & \frac{\partial\vec{\epsilon}_0}{\partial w_{W-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial\vec{\epsilon}_{N-1}}{\partial w_0} & \frac{\partial\vec{\epsilon}_{N-1}}{\partial w_1} & \cdots & \frac{\partial\vec{\epsilon}_{N-1}}{\partial w_{W-1}} \end{bmatrix} . \tag{6.23}$$

The parameter μ, adjusted iteratively, balances the LM between a steepest descent and a Gauss-Newton algorithm. To improve the training performance we used the Nguyen-Widrow (NW) method [Nguyen and Widrow, 1990] to initialize the network (see also [Hagan and Menhaj, 1994]).

The proposed neural network training is designed for batch learning, and generically the estimate improves with the growth of both training set and number of parameters[1]. Since the training is performed offline, in the prediction phase the computation is quite fast, consisting only of a single forward pass of the network. More precisely, given the number of neurons ν for a OHL neural network with n inputs, m outputs, sigmoidal activation functions (hyperbolic tangent $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$) in the hidden and output layer, the necessary operations are $\nu(n+m)$ products, $\nu(n+m+1)+m+2(\nu+m)$ sums, $2(\nu+m)$ exponentials and $\nu+m$ divisions, and the flops count is linear with ν. As an

[1]The number of parameters usually depend on three factors: the complexity of the function to be approximated (i.e. a very smooth function requires fewer neurons than a highly varying one, as more basis function are necessary to approximate the irregular changes of the latter), the dimension of the training set and the quality of the training set (i.e. to which extent the training set is representative for the variable space).

	$q[^\circ]$				$\dot{q}[^\circ/s]$				$\ddot{q}[^\circ/s^2]$			
joint #	0	1	2	3	0	1	2	3	0	1	2	3
max	150	60	30	70	18	15	20	15	103	76	248	34
min	50	-100	-60	10	-39	-49	-34	-23	-135	-85	-158	-38

Table 6.1: Value ranges of the arm joint positions, velocities and accelerations.

example, for 20 neurons, 4 inputs, 6 outputs and both layers with the usual hyperbolic tangent, the flops count (computed with the Lightspeed Matlab toolbox v.2.2. [Minka, 2009]) is 2766.

6.1.3 Results and Discussion

The three previously discussed methods have been evaluated experimentally on a common data set that has been gathered during a sequence of random arm movements, performed in joint space. Every movement brings the arm from a starting position $\vec{q}_\mathrm{s} \in \mathbb{R}^4$ to a final position $\vec{q}_\mathrm{f} \in \mathbb{R}^4$, which subsequently becomes the starting position for the next movement. Each of these positions is defined by a vector of joint angles, which are chosen randomly using a uniform distribution within the admissible range of the respective joint. Joint velocity profiles during motion are *bell-shaped* with a predefined maximum velocity, which causes the absolute velocities to vary from zero to the maximum value during any movement. Trivially, the sign of the velocity depends on the direction of the motion. The joint accelerations (i.e. actual slope of the velocity profiles) depend on the distance between $\vec{q}_\mathrm{s}$ and $\vec{q}_\mathrm{f}$, since the time duration of the movement is kept constant. Joint positions, velocities and accelerations were retrieved from the DSP boards at 50 Hz. Velocities and accelerations were computed via numerical differentiation on the DSP boards at a higher frequency (1 kHz). A simple collision avoidance strategy was used during the experimental data acquisition, in order to ensure that the arm would not collide with the body or the environment. The complete data set of 40000 samples has been shuffled and split in two equal parts. The set of the first 20000 samples is used for training and is subsequently subsampled to obtain smaller sized training sets, whereas the second half is used as a common test set. The reported performance measure on the test set is the

average Normalized Mean Squared Error (NMSE) over all 6 output dimensions, where the NMSE is defined as the mean squared error divided by the variance of the respective output dimension. For the two machine learning approaches, the input dimensions have been rescaled (see original ranges Table 6.1) to be approximately within the range $[-1, +1]$, based on the maximum and minimum values found in each input dimension in the training set.

Number of Training Samples

In this initial experiment we measured the performance of each method when increasing the number of training samples. The results in Fig. 6.1 show clearly that the two learning methods have a strong dependency on the size of the training set. As more samples become available, they consistently continue to improve performance, eventually outperforming the model-based approach by an order of magnitude. Interestingly, the model-based approach appears to perform at a constant level, regardless of the number of samples. This is confirmed by further analysis on even smaller data sets, as demonstrated in Fig. 6.2. When considering only the joint positions, it shows the remarkable capability of achieving acceptable performance using only 5 training samples. This means that the model-based approach is the preferred approach when only very few samples are available. The machine learning methods require many more samples to achieve similar performance. This is not surprising considering that the model based approach takes advantage of additional information implicit in the structure of the model.

Contribution of Velocity and Acceleration on the Estimation

Another observation (Fig. 6.3) is that inclusion of joint velocities and accelerations does not always improve the generalization performance when training is done on a small number of samples. Intuitively, one might expect that adding relevant information could only improve the estimation. However, learning methods require an increasing amount of training samples to make effective use of this additional information (i.e. the *curse of dimensionality* [Duda et al., 2001]). This affects particularly the learning methods, since these need to construct their model based solely on training data. Fig. 6.3 shows that both LS-SVM and NN eventually use joint velocities to improve their predictions, given a sufficiently large training set.

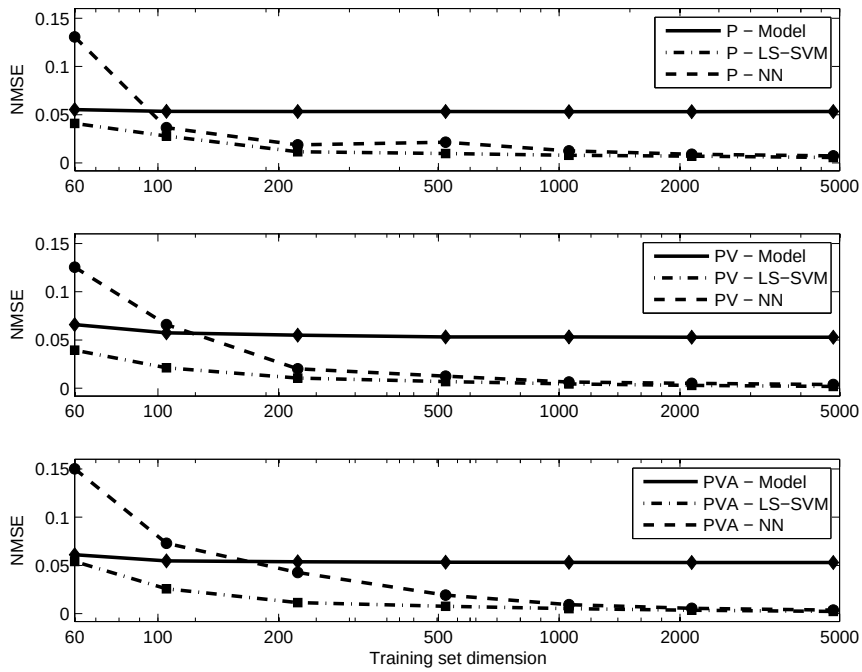


Figure 6.1: Comparison of the three methods on random training subsets of increasing dimension and three different input spaces. P denotes the input space containing only joint positions ($\vec{q} \in \mathbb{R}^4$), PV contains both joint positions and velocities ($\vec{q}, \dot{\vec{q}} \in \mathbb{R}^8$), and PVA contains joint positions, velocities and accelerations ($\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}} \in \mathbb{R}^{12}$).

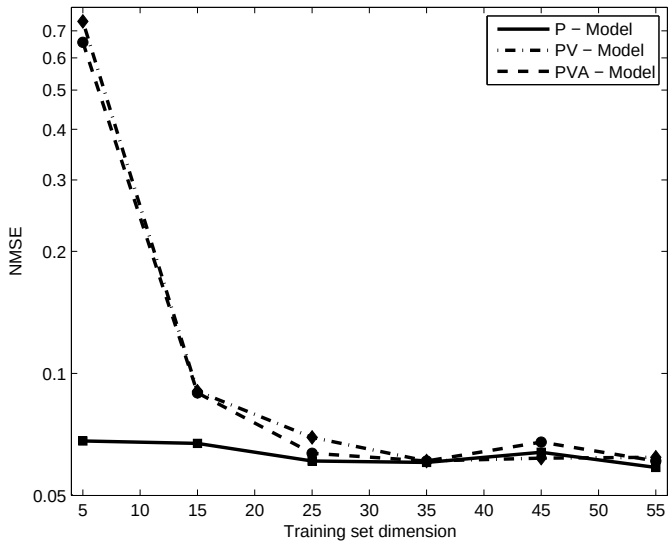


Figure 6.2: Performance of the model-based method on very small training sets.

Joint accelerations, however, do not improve prediction performance in any of the cases (cf. Fig. 6.4). This is probably due to the low signal to noise ratio for the acceleration and, in first place, to the robotic setup used to obtain the data set. In particular, the joint accelerations were not measured directly but were derived from positions. This causes the acceleration measurements to be much less precise and reliable than those for joint velocities and positions. Furthermore, the range of accelerations is relatively small[1] and within this range we observed that the contribution of $M(\vec{q})\ddot{\vec{q}}$ in equation (6.2) is relatively small compared to the contribution of the other terms ($C(\vec{q}, \dot{\vec{q}})\dot{\vec{q}}$ and $G(\vec{q})$).

Selective Subsampling

The data set we acquired is characterized by the fact that there is an abundance of training samples. Thus far, we have used a uniform random subsampling strategy, as to ensure that the qualitative properties of the subset approximate those of the original data set. However, with such an abundance of samples it is likely that the original data set is *oversampled* (i.e. additional samples do not further increase the generalization performance) and contains samples that are (nearly)

[1]The chosen motors produce limited torques, which reflects into relatively low accelerations.

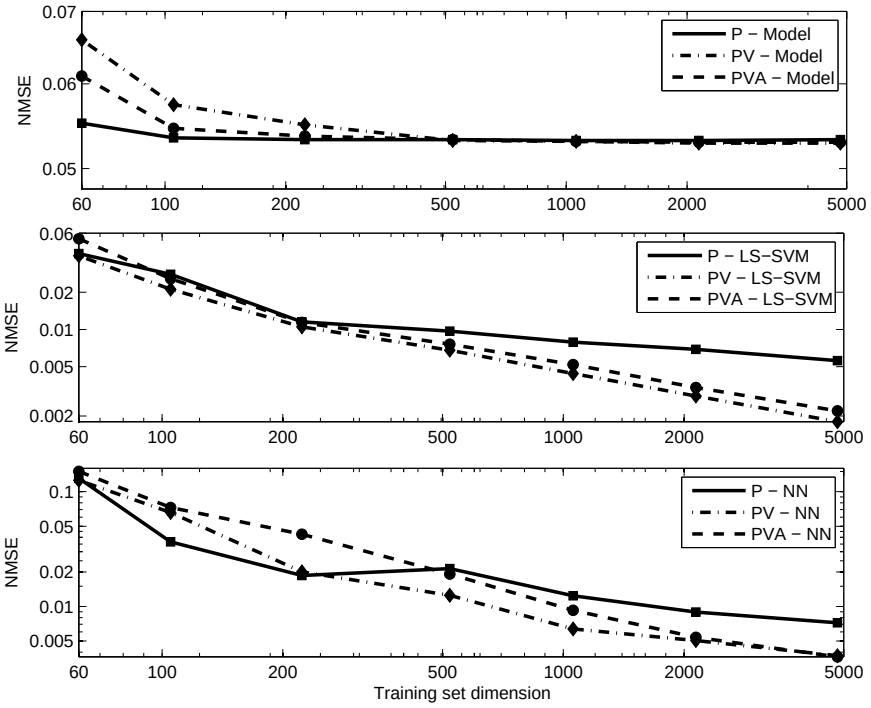


Figure 6.3: Comparison of the performance for all methods with increasing input spaces (i.e. P, PV and PVA; defined as in Fig. 6.1). Note that both axes are in logarithmic scale to accentuate differences in final performance.

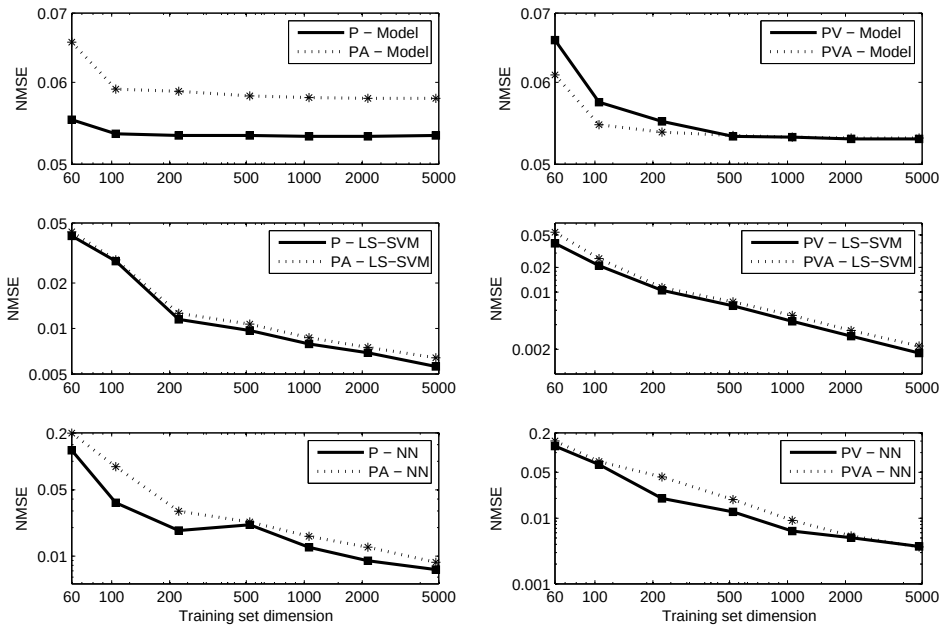


Figure 6.4: Performance after inclusion of joint accelerations. The figures on the left hand side compare the performance on P and PA (defined analogously to P, PV and PVA in Fig. 6.1), while those the right hand side compare PV and PVA.

identical to each other. This similarity of input samples particularly affects LS-SVM, as this method describes the prediction function in terms of inner products with respect to all training samples.

We can guarantee a certain "sparsity" of the training set by taking a subset, such that the inter-sample distance is at least a threshold t. Let us define a distance measure $D(\vec{x}_i, \vec{x}_j) = \sqrt{(\vec{x}_i - \vec{x}_j)\Sigma^{-1}(\vec{x}_i - \vec{x}_j)}$, where $\vec{x}_i, \vec{x}_j \in \mathbb{R}^n$ and Σ is an $n \times n$ matrix containing the variances of all input dimensions on its diagonal. This measure coincides with the Euclidean distance in the standardized input space. In order to construct a Euclidean subset $\mathcal{E}_t$, we iterate over a permutation of the original data set $\mathcal{S}$ using index $i = 1, \ldots, \ell$ and append only those samples to $\mathcal{E}_t$, for which $\min D(\vec{x}_i, \vec{x}) \geq t \; \forall \vec{x} \in \mathcal{E}_t$.

Fig. 6.5 shows the prediction performance of LS-SVM with random and Euclidean subsampling. The Euclidean subsets were generated by varying the threshold t, such that the size of the subsets were nearly identical to each of the random subsets. Whether selective subsampling based on Euclidean distance outperforms random subsampling depends on the input space that is used to determine the inter-sample distance, and the size of the training set. When this distance is determined solely based on the joint positions, then Euclidean subsampling results in a significant improvement for small data sets. In contrast, random subsampling performs better than Euclidean subsampling based on joint positions, velocities and accelerations. It is our belief that this difference is due to the Euclidean strategy attempting to create a uniform sampling distribution in all dimensions under consideration, effectively forming subsets that contain a wide range of velocities and accelerations (besides positions). Given the relatively low velocities and accelerations of the robot, the force and torques are primarily caused by gravity. In return, gravity is only dependent on the joint positions of the robot. It is therefore beneficial, for limited training sets, to select those samples that help LS-SVM to model this gravity component.

Further, we can note that the different subsampling strategies perform nearly identically for large training sets. This can easily explained by the fact that the size of $\mathcal{E}_t$ is inversely proportional to the chosen distance threshold t and, by definition, $\mathcal{E}_t$ becomes a random permutation of $\mathcal{S}$ as t approaches zero. In short, for large data sets, and thus a small inter-sample threshold, the Euclidean

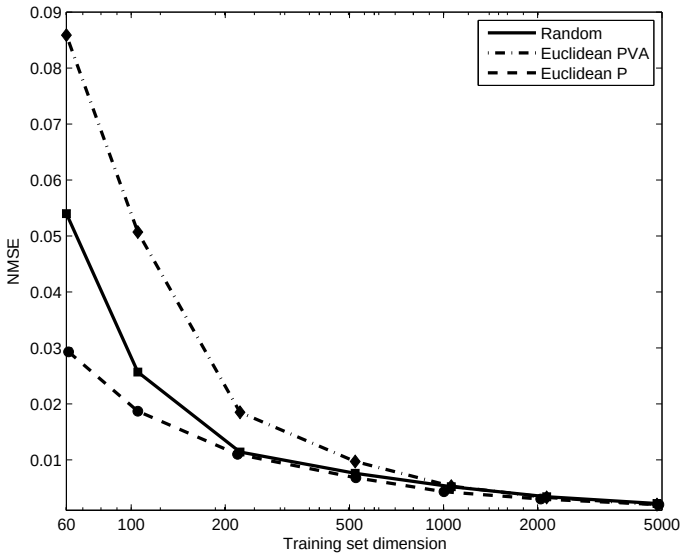


Figure 6.5: Comparison of random and selective subsampling based on standardized Euclidean distance. *Euclidean P* and *Euclidean PVA* denote subsampling based on Euclidean distance thresholds $t = \{1.35, 1.15, 0.88, 0.65, 0.5, 0.35, 0.18\}$ using position inputs and thresholds $t = \{6.0, 5.3, 4.5, 3.7, 3.1, 2.5, 1.8\}$ using position-velocity-acceleration inputs, respectively.

and random subsets have very similar sample distributions and therefore similar performance.

6.1.4 Online learning

As described in the beginning of section 6.1.3, the data used for training has been collected during an exploration phase in which the robot arm was moving toward randomly distributed targets in joints configuration. Anyway, such data can be potentially gathered during any arm movement (e.g. reaching movements), without the need of a particular explorative policy. Actually, retriving the data while performing meaningful actions (instead of random movements) could be even a better strategy in order to limit the exploration to the most useful part of the learning space. For example, considering that the input space is formed by arm joint positions and velocities, and that the exploration is performed through

reaching movements, there could be regions of the robot workspace which do not need to be explored (e.g. arm configurations bringing the hand behind the robot, outside its visual workspace) or arm velocities which are in reality not experienced. These part of the learning space could anyway be explored in a second moment, if a new acquired behavior requires it (e.g. to scratch its own back or to move with faster velocities). Considering our robot, the only missing component to realize such online learning in a completely autonomous way is a tactile sensor distributed over the arm surface. In fact, the robot must know if the forces and torques measured by the F/T sensor are due just to its own motion or also to external contribution. As already said, there are plans to mount a full body sensing skin [Cannata et al., 2008] in the future, which can be used to the purpose. So far, we carried on an online learning experiment collecting data during subsequent reaching trials, manually checking that the robot arm was not subject to any external force. Data is composed by arm joint positions and velocities ($\vec{q}, \dot{\vec{q}} \in \mathbb{R}^8$) and measured forces and torques ($\vec{x} = [\vec{f}^\top, \vec{\tau}^\top]^\top = \vec{x}_I \in \mathbb{R}^6$). Two neural networks have been trained online during data acquisition, based on RFWR and LWPR respectively. Then, NMSE has been computed on a common test set of 2000 samples collected during later reaching movements. Results are reported in figures 6.6 and 6.7.

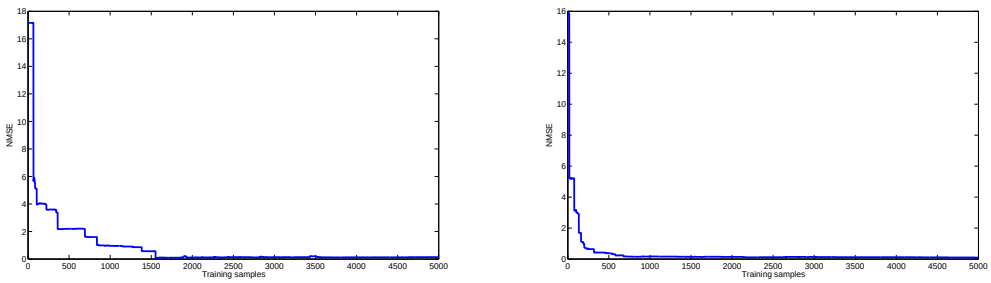


Figure 6.6: NMSE of the internal forces estimation, with increasing size of training set. In the left plot, the net is trained online with RFWR. In the right plot, LWPR is employed. The NMSE decreases exponentially as more samples are included in the training set.

These results, even though preliminary, show that internal forces can be estimated also exploiting online learning techniques, and that the estimation error

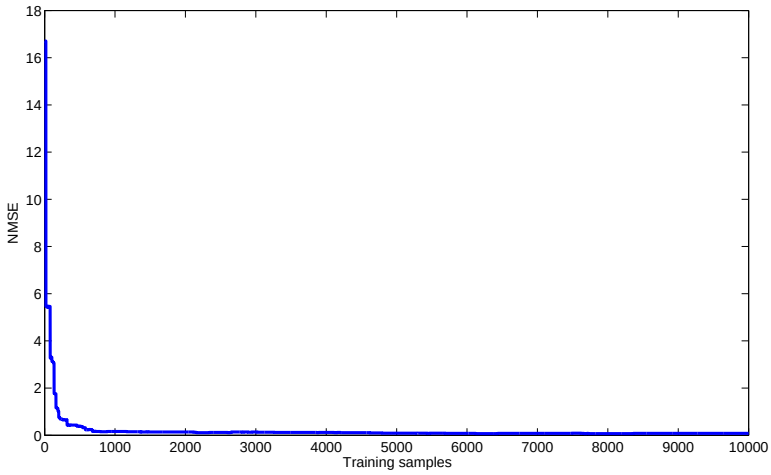


Figure 6.7: NMSE of the internal forces estimation, with a further increase of training set size. The net is trained online with LWPR. After a certain amount of data has been reached, it seems that adding more training samples do not improve the estimation consistently, even if a slight reduction of the NMSE is still present.

decreases as more training data are provided to the learning network. The final NMSE achieved with LWPR learning, after training with 10000 samples, is 0.07. Nevertheless, we cannot quantitatively compare these results to the ones obtained in the previously presented analysis, since both training and test set are different.

6.2 Obstacle avoidance

Once the internal forces have been estimated, the external forces can be simply obtained from the FT sensor measurement, as shown in equation 6.3. On the basis of this information, an obstacle avoidance strategy can be implemented to accomplish reaching tasks. Here we propose a solution using potential force field for motion generation and Incremental Least Squares Regression [Hosoda and Asada, 1994] to estimate the Jacobian of the contact (which provides possible directions to avoid the obstacle). This controller is integrated in the *armController* software module. A description of the control strategy here follows.
As already explained in section 3.5, the arm motion is handled by the *armController*, which defines a force field in the arm joints space. During the ballistic

part of the reaching action, the target position of the arm is modeled as an attractive pole in the force field. Conversely, any arm configuration which needs to be avoided can be modeled as a repulsive pole.
Once the robot has computed the necessary arm configuration to reach for the visually identified target, it only senses an actractive pole which brings the arm toward that configuration. A velocity command, depending on the distance from the target, is given to each joint. During the motion, the force/torque sensor is used to detect contacts with the environment (i.e. presence or not of external forces). If a contact occurs, an estimated Jacobian matrix of the contact is used to let the robot move in a safe position (i.e. a position in which both the external forces and torques are equal to zero).
Arm joint positions, $\mathbf{q} \in \mathbb{R}^4$, and external force/torque measurements[1], $\mathbf{x}_E \in \mathbb{R}^6$, are used to estimate online the Jacobian of the contact, $\hat{J}_c(t) \in \mathbb{R}^{6x4}$:

$$\hat{J}_c(t) = \hat{J}_c(t - \Delta t) + \Delta \hat{J}_c(t) \tag{6.24}$$

$$\Delta \hat{J}_c(t) = \begin{cases} \frac{\left\{(\Delta \mathbf{x}_E(t) - \hat{J}_c(t-\Delta t)\Delta \mathbf{q}(t)\right\}\Delta \mathbf{q}(t)^T W(t)}{\varrho + \Delta \mathbf{q}(t)^T W(t) \Delta \mathbf{q}(t)} & if \|\Delta \mathbf{q}(t)\| \neq 0 \\ 0 & if \|\Delta \mathbf{q}(t)\| = 0 \end{cases} \tag{6.25}$$

where $\Delta \mathbf{x}_E(t) \in \mathbb{R}^6$ is the vector of the external forces/torques variations at time t, and $\Delta \mathbf{q}(t) \in \mathbb{R}^4$ is the vector of arm joints positions variations at time t, $\hat{J}_c$ is the jacobian estimation, W is a weighting matrix and ϱ is a forgetting factor.
During the interaction, the robot is controlled giving as input the velocity $\dot{\mathbf{q}} \in \mathbb{R}^4$:

$$\dot{\mathbf{q}}(t) = -K \hat{J}_c(t)^T \mathbf{x}_E(t) \tag{6.26}$$

where $K \in \mathbb{R}^{4x4}$ is a constant positive gain matrix; this control makes the manipulator converge in a position where external force/torque measurements are zero. Then, the potential force field is updated adding a repulsive pole where the contact happened, in the arm joints space, $\mathbf{q}^c$. The presence of a repulsive

[1]In this experiment $\mathbf{x}_I$ was estimated using the model-based approach described in section 6.1.2.

pole generates a velocity $\dot{\mathbf{q}}$, as in equation 6.27, which is added to the velocity generated by the attractive pole.

$$\dot{\mathbf{q}}(t) = K(\mathbf{q}^c - \mathbf{q}(t)) \tag{6.27}$$

Now the arm motion is driven by the attractive action of a pole located in the arm target configuration and by the repulsive action of the new pole. In this way the robot is able to find a safe path to reach for the target without hitting the obstacle.

In figure 6.8 a joint-space trajectory followed by the arm during an obstacle avoidance movement is depicted (red solid line). The trajectory is compared to the situation in which the obstacle is not present (blue dashed line). The sensory stimuli that elicited the motor action are shown in figure 6.9. Since the evaluation of the external forces and torques is noisy, due to intrinsic sensor noise and internal forces and torques estimation errors[1], we put arbitrary upper and lower thresholds to determine the presence or not of a contact ($\pm 5N$ for forces and $\pm 1\frac{N}{m}$ for torques). If at least one component of x_E exceeds the threshold, the obstacle avoidance action starts.

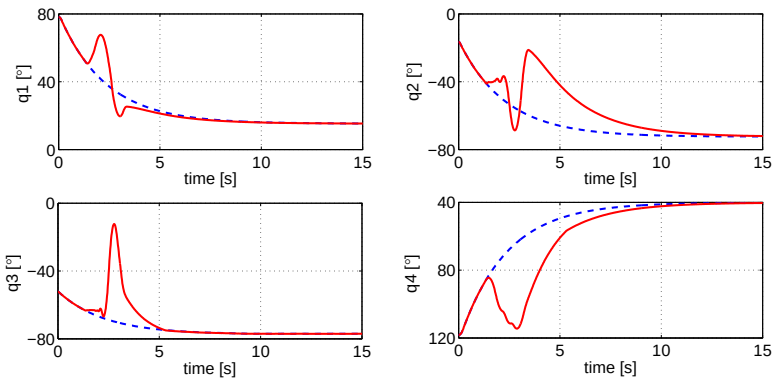


Figure 6.8: Arm joints trajectories during an obstacle avoidance movement. The red solid line is the trajectory followed when there is contact with the object. The blue dashed line shows the same movement performed without any contact.

[1]In this experiment, the internal forces are estimated using the model-based approach, as described in section 6.1.2.

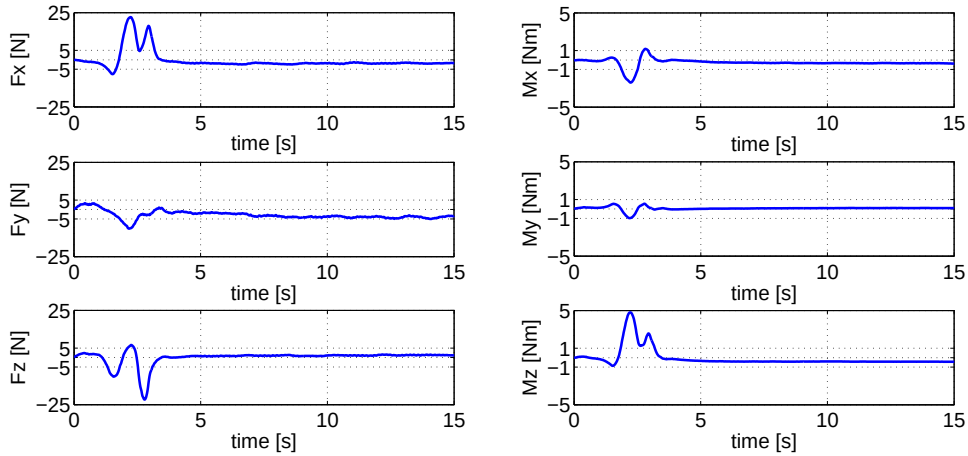


Figure 6.9: Estimation of the external forces and torques during the movement. When at least one of the components exceeds the upper or lower threshold the obstacle avoidance motion starts, driven by this information.

6.3 Tactile interaction

As described in section 3.3, the hand of James is equipped with a total of 17 tactile elements, distributed on the internal side of each finger. In this section we report experiments showing a possible way to exploit such sensing capability to realize a simple form of interaction. In particular, the tactile information is integrated with vision and external force/torque sensing, triggering different actions depending on this sensory information. The measure of the external forces and torques applied to the robot arm is obtained by subtracting the estimated internal forces/torques from the arm F/T sensor measurement, as in equation 6.3. In the following experiments, the estimation of the internal forces and torques is achieved through online learning, as discussed in section 6.1.4.
Three sets of graphs describe three different situations that could occur during a ballistic reaching action:

. ballistic reaching without any contact between the hand and the external environment. The arm simply moves to the target joints configuration.

. ballistic reaching with contact between the hand and the external environment while the hand is inside the visual field. The touched object is assumed

to be the target one; the reaching action is considered to be accomplished successfully and the arm motion stops.

. ballistic reaching with contact between the hand and the external environment while the hand is outside the visual field. The touched object is assumed not to be the target one; the robot moves the wrist in order to avoid the object (i.e. the obstacle).

Differently from previous sections, here we consider an additional DOF of the arm, namely the wrist pitch θ_{wp}. Therefore, the arm configuration is here defined as $\mathbf{q} = [q1\ q2\ q3\ q4\ q5]^T = [\theta_{sp}\ \theta_{sy}\ \theta_{sr}\ \theta_e\ \theta_{wp}]^T \in \mathbb{R}^5$. Anyway, this additional DOF is actually moved just in the third situation, while in the first two cases its position is controlled to zero.
The first set of graphs describes a ballistic reaching action without any hand contact. Figure 6.10 reports arm joints positions and velocities. The *armController* brings the arm to the target joints configuration, with bell-shaped velocity profiles. Figure 6.11 shows the activation of the hand tactile sensors during the experiment: clearly no touch is detected. Any output under an arbitrary defined threshold (the dashed line in the graph) is ignored, as it is generally due to sensor noise; note that even applying this threshold the sensors are able to detect very slight contacts, in the order of $10^{-1}N$ (normal component of the applied force). Figures 6.12 and 6.13 display the estimated external forces and torques. Upper and lower thresholds (dashed lines) are the same as in the previous experiment (see section 6.2).
The second set of graphs describes the situation in which a contact is sensed during the ballistic motion, while the hand is visually detected inside the visual field. This concurrence of visual and tactile stimulation causes the ballistic motion to suddenly stop: the target object is considered to be reached.
It is clear from figure 6.14 how the joints positions do not reach the desired value; on the contrary, they all rest at a different position, and all velocities istantaneously go to zero, as soon as one of the tactile sensors outputs exceeds the threshold (see figure 6.15).
The contact of the hand with the object is also reflected in the external forces and torques estimation (figures 6.16 and 6.17). Anyway, the contact is slight

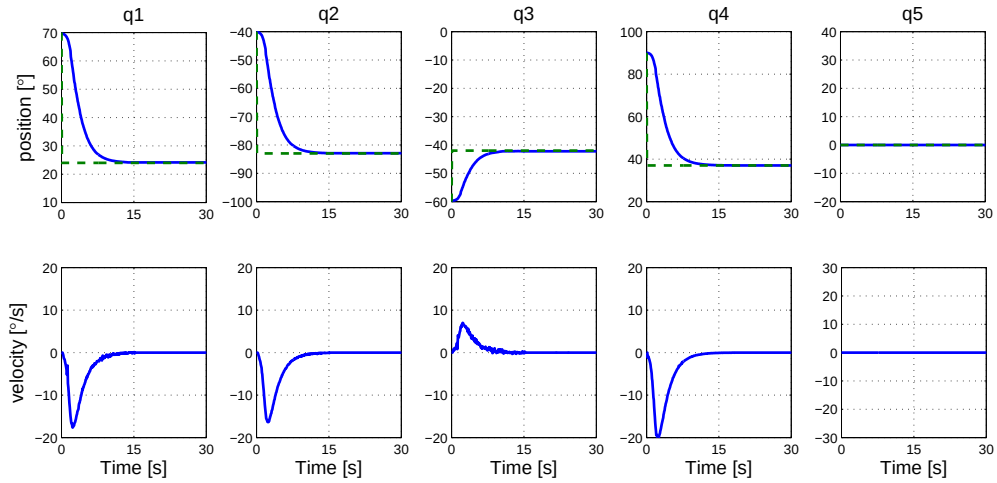


Figure 6.10: Arm joints positions and velocities during a ballistic reaching movement. No hand contact occurs.

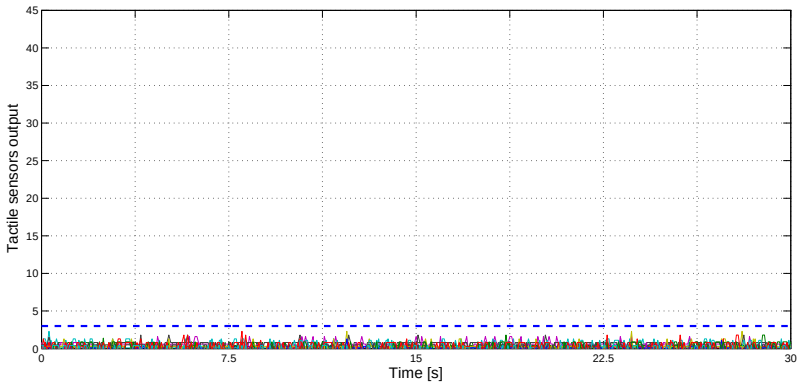


Figure 6.11: Hand tactile sensors measurements during a ballistic reaching movement. No hand contact occurs.

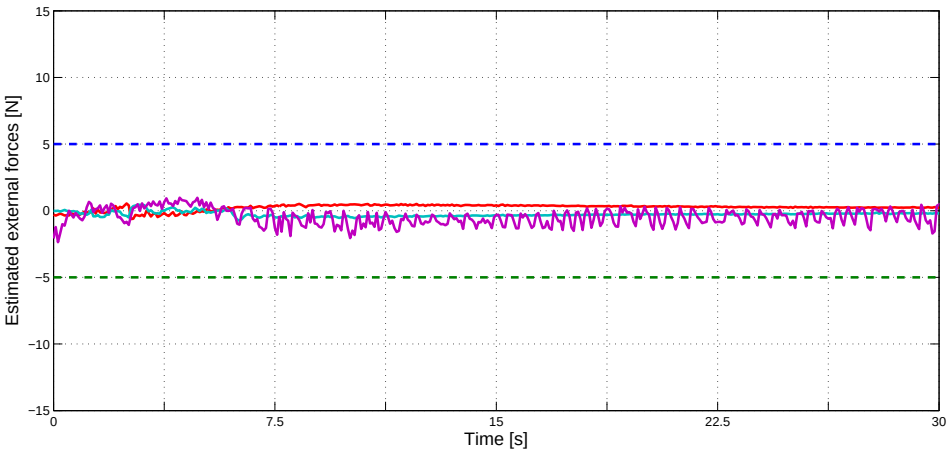


Figure 6.12: Estimation of external forces during a ballistic reaching movement. No hand contact occurs.

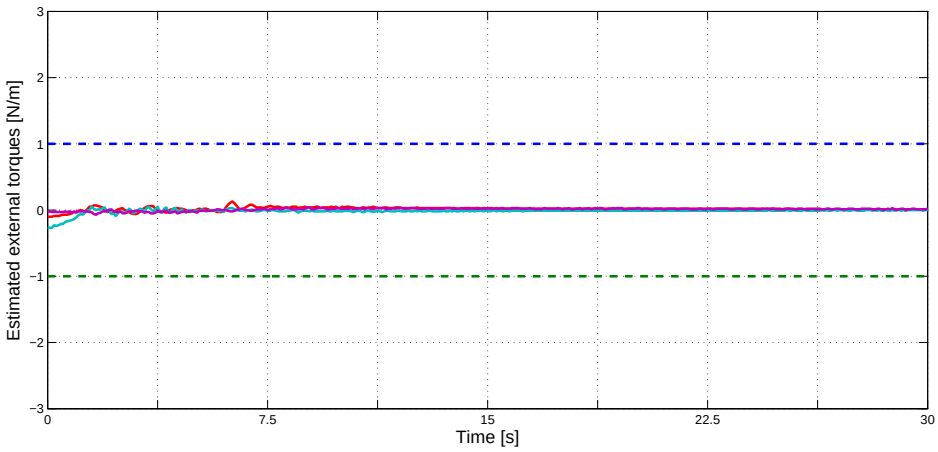


Figure 6.13: Estimation of external torques during a ballistic reaching movement. No hand contact occurs.

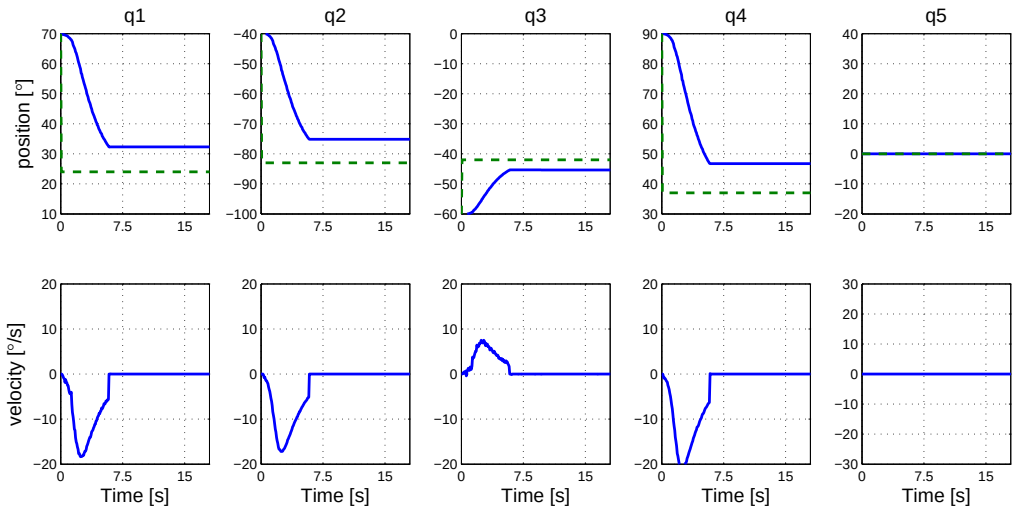


Figure 6.14: Arm joints positions and velocities during a ballistic reaching movement. Hand contact occurs while the hand is inside the visual field: the target is considered to be reached and the arm motion suddenly stops.

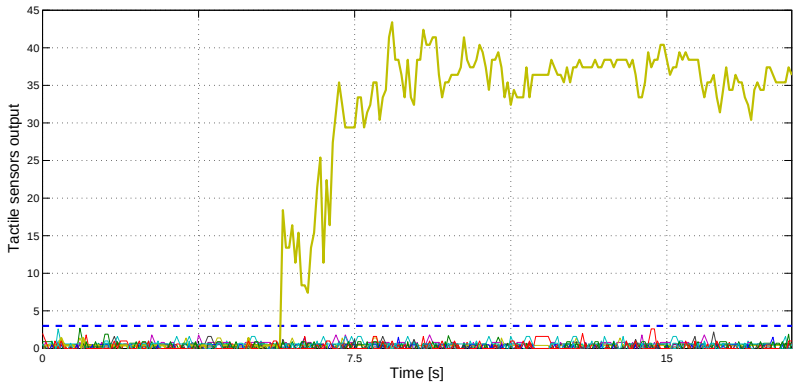


Figure 6.15: Hand tactile sensors measurements during a ballistic reaching movement. Hand contact occurs while the hand is inside the visual field: the target is considered to be reached and the arm motion suddenly stops.

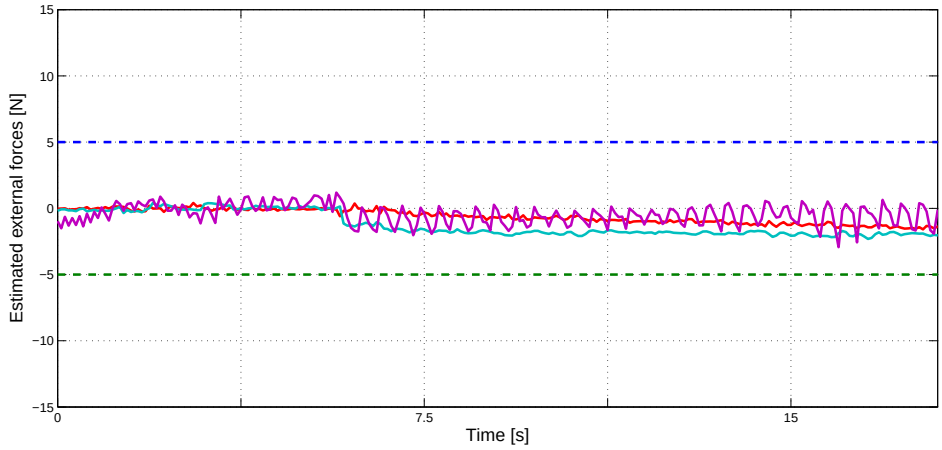


Figure 6.16: Estimation of external forces during a ballistic reaching movement. Hand contact occurs while the hand is inside the visual field: the target is considered to be reached and the arm motion suddenly stops.

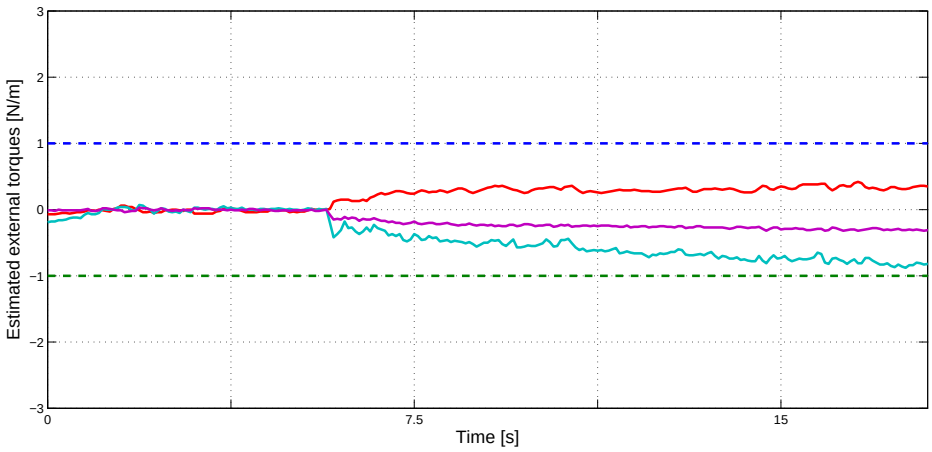


Figure 6.17: Estimation of external torques during a ballistic reaching movement. Hand contact occurs while the hand is inside the visual field: the target is considered to be reached and the arm motion suddenly stops.

and the estimated forces and torques do not exceed their thresholds. Indeed, the two sensory systems work at different scales. Consequently, no obstacle avoidance motion of the arm is generated. Furthermore, the integration of tactile and visual sensations enriches the perception of external forces/torques: the robot is aware that the contact is happening on the hand (which is also inside the visual field, and probably touching the target object) and can plan its actions on the basis of this more complete information. In this case, no obstacle avoidance motion is needed, since the hand has reached for the target object and the arm motion has been stopped.

Finally, the last set of graphs describes a different situation, in which the contact happens when the hand lies outside the visual field. In this case, the hand is probably not touching the target object, but something else (i.e. an unexpected obstacle). Therefore, the robot should avoid this obstacle while continuing the motion toward the target arm configuration. To achieve this, it can move the wrist trying to cancel the tactile stimulation (i.e. to eliminate the contact). Figure 6.18 depicts this action. It can be noticed that the motion of the first 4 DOF of the arm is the same as figure 6.10, where no contact was present; conversely, the wrist is moved back to avoid the obstacle (see figure 6.18, joint $q5$). Wrist motion is elicited by the tactile sensors stimulation shown in figure 6.19. Remarkably, this time more than one sensor output has exceeded the threshold. In particular, four tactile elements are stimulated in sequence: the first is the *phalangeal-sensor* of the index finger, the second and third are the *phalangeal-sensors* of the middle finger and the last one is the *fingertip-sensor* of the middle finger. This indicates that the hand slithered on the object during the motion. During the interaction the wrist is controlled with a velocity proportional to the intensity of the (overall) tactile stimulation. When there is no contact, wrist position is restored to zero with a proportional controller.

Figures 6.20 and 6.21 plot the estimated external forces and torques. Even in this case, the magnitude of the estimated external forces/torques during contact is not enough to generate an arm obstacle avoidance action. Anyway, differently from the previous situation, if the external forces/torques were larger (i.e. exceeding the thresholds) it could mean that wrist motion is not enough to avoid the touched obstacle, and that an obstacle avoidance movement of the arm is

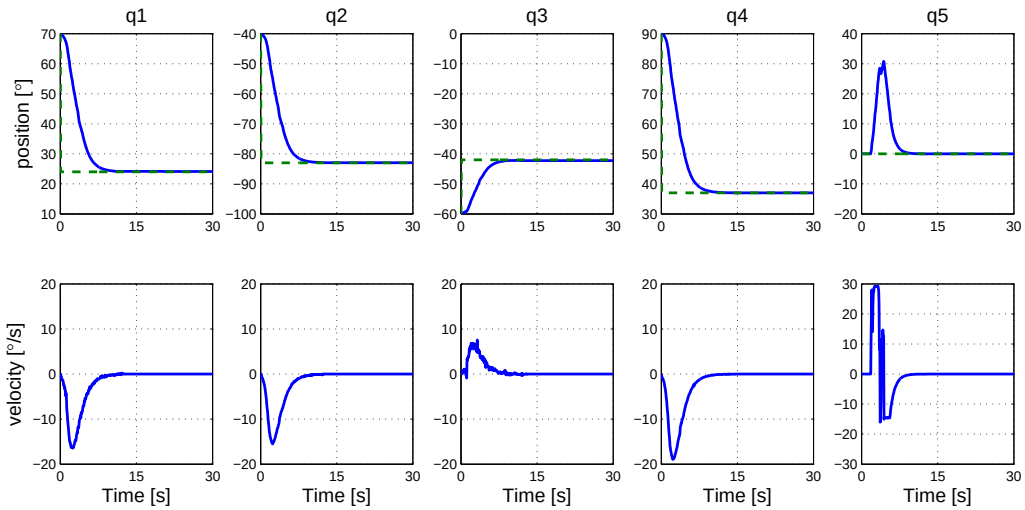


Figure 6.18: Arm joints positions and velocities during a ballistic reaching movement. Hand contact occurs while the hand is outside the visual field: an obstacle avoidance movement of the wrist is performed.

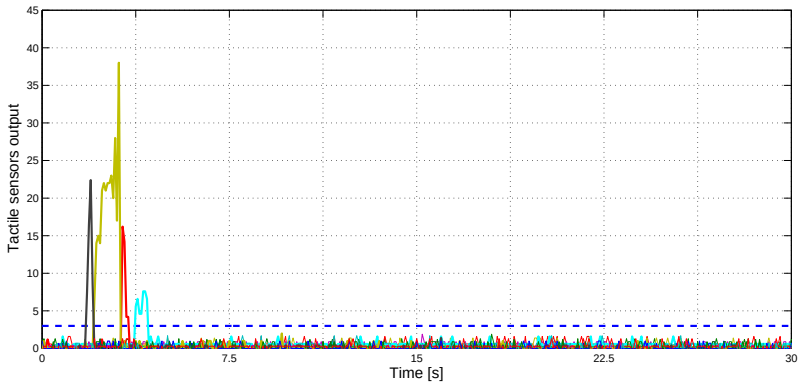


Figure 6.19: Hand tactile sensors measurements during a ballistic reaching movement. Hand contact occurs while the hand is outside the visual field: an obstacle avoidance movement of the wrist is performed.

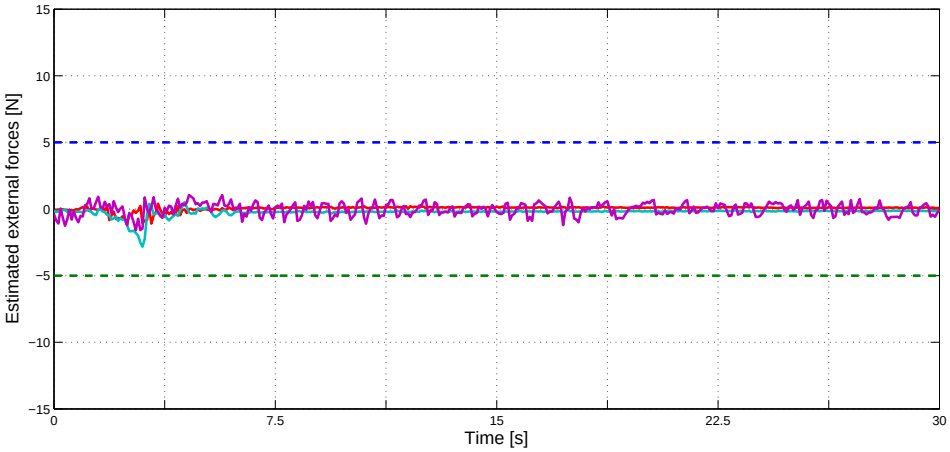


Figure 6.20: Estimation of external forces during a ballistic reaching movement. Hand contact occurs while the hand is outside the visual field: an obstacle avoidance movement of the wrist is performed.

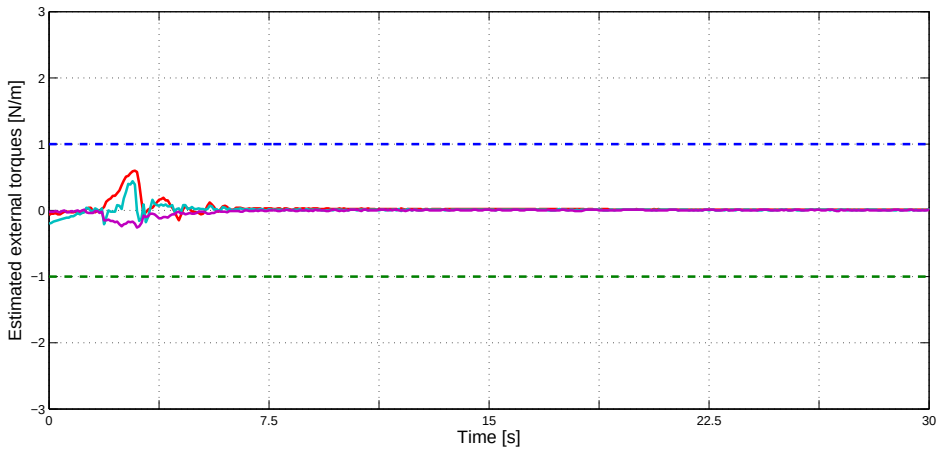


Figure 6.21: Estimation of external torques during a ballistic reaching movement. Hand contact occurs while the hand is outside the visual field: an obstacle avoidance movement of the wrist is performed.

required to complete the ballistic reaching successfully and safely. This could be a way to exploit the fact that the two sensory sytems (force and touch) can detect contacts at different scales.

7

Conclusions and future work

In this thesis we have presented a possible approach to achieve autonomous sensori-motor learning in a humanoid robot. We explained how the solution adopted by biological systems can be a source of inspiration, especially focusing on three aspects: the importance of the physical interaction with the environment, the necessity to find a smart balance between a-priori knowledge and acquired knowledge, the need for learning to be performed on-line.

We demonstrated the feasibility of our approach by providing experimental results concerning three different problems: learning how to control the neck orientation, learning how to reach for visually identified targets and learning the arm dynamics. Remarkably, the proposed learning strategies are general enough to be applied to any humanoid robot equipped with the necessary motor and sensory system. Indeed, the development of a general framework for autonomous learning, independent from the specific platform, can dramatically reduce the time spent for robot modeling and programming.

In the case of neck control (see chapter 4), learning has turned out to solve specifical control problems we encountered in previous work on the same platform. Moreover, the study provides potential hints for the control of similar tendon-driven parallel mechanical systems.

With respect to the problem of reaching (see chapter 5), a flexible and robust solution is sought both in industrial settings (e.g. to advance manufacturing automation, to work in hazardous environments) and in humanoid research laboratories. In particular, considering humanoid robots, efficient reaching can be

a basis for developing higher order cognitive processes (e.g. classify and label grasped objects [Jamone et al., 2006], learn about objects dynamics by tapping them [Fitzpatrick et al., 2003]) and social behaviors (e.g. by getting in physical contact with humans, by manipulating the same common-use objects). Moreover, preliminary studies have been carried out concerning the incremental building of a reachable space map based on motor information, which would allow the robot to evaluate its possibility to reach for an object by fixating it.
Regarding the arm dynamics (see chapter 6), we provide a comparison between different approaches to estimate the internal forces and torques generated by the arm during motion, as measured by a single six-axis F/T sensor, using both an analitical model with parameters estimation and two offline learning algorithms. Indeed, we believe that the results obtained can also be generalized to similar estimation problems in robotics. Such an estimation is required to extract a measure of the external forces/torques applied to the robot arm, which can be used to control the interaction with the external environment (e.g. obstacle avoidance). Furthermore, investigations have been carried out also concerning the online incremental learning of such a mapping during goal-directed movements.
Noticeably, the three learning problems we studied can be unified together in a developmental framework. In fact, the attainment of reaching behavior relies on the aquisition of neck orientation control, which allows fixation of a perceived object in the visual field; indeed, gaze direction when the object is fixated is used as a reference frame for the reaching action, as suggested by neurophysiological observations in humans. Moreover, as soon as the robot is able to properly detect external forces, this knowledge can be exploited to interact with the environment; preliminary experiments are presented which show the integration of tactile, visual and force information during the execution of reaching tasks, allowing a safe interaction with the external world.

7.1 Future work

The study of the development of an artificial human-like system is by definition a endless work. Indeed, this research opens up several interesting problems and offers opportunities for further studies. In the following we discuss some of the

possible extensions to this work, considering them as the starting point for future researches.

7.1.1 Grasping

Successful reaching is the main prerequisite for grasping. In this sense, learning how to grasp and manipulate the reached object is the first, obvious, potential extension to the present work.
Interestingly, hand synergies have been observed in human subjects performing reach-to-grasp movements [Mason et al., 2001] or mentally simulated grasps of familiar objects [Santello et al., 1998]. Moreover, a recent theory concerning the control of the movement of the limbs proposes the idea that a finite number of basic muscular synergies (*motion primitives*) are coded into the CNS, which is able to plan more sophisticated movements just by combining some of these primitives depending on the specific task and context.
Mussa-Ivaldi and Bizzi [Mussa-Ivaldi and Bizzi, 2000] describe results supporting this hypothesis, obtained from experiments with frogs and rats, whose muscular synergies have been evoked by the microstimulation of specific interneurons in the spinal cord.
It has also been shown [Nori and Frezza, 2004; Nori et al., 2006] that an appropriate set of motion primitives allows a controlled physical system to reach any admissible configuration through the *linear* combination of these same elementary modules. Although there were no evidence that the human spinal cord is in fact organized following this modularity, it is likely that evolution preserved a working solution across species (i.e. from rat to human). In particular we do not know whether hand movements follow the same modularity although in fact Rizzolatti and colleagues propose that a vocabulary of actions is available in the premotor cortex specifically dedicated to grasping and manipulation [et al., 1996]. Anyway, this strategy seems efficient and can be borrowed in designing solutions to the grasping problem. Some authors have recently proposed to exploit a small set of motion synergies to grasp common objects with articulated robotic hands, smartly reducing the complexity of the problem [Ciocarlie et al., 2007; Tsoli and Jenkins, 2007]; some draft ideas for the realization of a similar system, based on

learning and multisensory integration, are reported hereinafter.
As a consequence of the reaching motion, the hand should touch the target object. At this point, a grasping action must be chosen depending on the specific object and context of action (e.g. arm/hand configuration, position of the hand with respect to the object, object affordance), or, in other terms, on the sensory feedback (touch, vision, force, proprioception). Let's assume that the robot is bootstrapped with a limited set of hand motion synergies (i.e. coordinated movements of all the degrees of freedom), which can be seen as elementary modules that can be combined to create more sophisticated actions. An example of synergy can be the motion observed in human *palmar grasp reflex*. Within this framework, to choose a grasping action means to find a suitable set of coefficients to linearly combine the motor synergies. Following the experiments of [Fernandez and Walker, 1998; Jr. et al., 2002; Steffen et al., 2007], the implementation of a system which chooses the best grasping action through the linear combination of a set of basic motor synergies could proceed by maximizing a proper cost determined by the number of contact points obtained through haptic perception, given the current object and context.
Of course, in order to show this ability of evaluation and improvement of performances (i.e. move the fingers holding the unstimulated sensors toward the object) two informations are needed: a topological map of the sensors within the hand and a sensorimotor map which estimates the sensory consequence of a motor action. Following the developmental approach, we want these maps to be autonomously learned by the robot through interaction with the environment. It is reasonable to assume that a topological map can be built by analyzing the correlation between data recorded from different sensory channels [Olsson et al., 2005] (in this case, the different tactile elements). Furthermore, tactile information can be combined with proprioception, exploiting multisensory integration as suggested by generic neural science results [Kandel et al., 2000]. Others propose solutions to the grasping problem [Chinellato and del Pobil, 2005; Natale et al., 2005] mostly based on visual information, which is fundamental to gather data about objects before getting in contact with them.
A more comprehensive system must integrate all the available sensory channels to gather as much information as possible about the external world, retrieving

complex object features and building sensorimotor mappings through incremental learning. Within such a framework, it appears quite clear how action and perception could be linked together to form a single and unique tool of development. Indeed, gathering visual, proprioceptive, kinesthetic and haptic information allows the robotic system to make new associations between objects/contexts and sets of coefficients used to linearly combine the coded motion synergies, i.e. learning to handle new objects and situations. This acquired experience can even lead to reuse previously learned associations for similar objects/contexts perhaps extracting and exploiting a certain modularity of the world: e.g. to grasp and use a water jug I can exploit both my knowledge of how to grasp a cup and my knowledge of how to use a bottle of water, of course only if I have caught the similarity with the cup in terms of geometry of the handle and that with the bottle in terms of affordance. Eventually, the choice of the suitable coefficients to linearly combine the aforementioned motion synergies, in order to produce the desired motor output to grasp a visually identified object, will be determined addressing the learned sensorimotor mappings with this complete sensory information.

7.1.2 Full Body Reaching

When whole-body reaching is considered, the ballistic movement will become a motion toward a whole-body target configuration, and the closed-loop correction will exploit additional sensory modalities (e.g. force interaction, vestibular sensing). When dealing with such a complex motion, redundancy must be solved in a smart way, in order to keep balance and avoid obstacles while achieving the reaching task. This could be done [Sentis and Khatib, 2006] by projecting the motor commands related to the task (reaching) in the null space of the constraints (balance, potential obstacles), after having learned the necessary Jacobian mappings through experience. As suggested in section 5.2.3, the results of the robot reaching attempts can constitute a source of information in order to build a map of its reachable space. Exploiting such a map, the robot would be able to assess the probability of a perceived object to be reachable or not just by gazing at it.

7. CONCLUSIONS AND FUTURE WORK

8

Appendix

8.1 RFWR and LWPR

RFWR (Receptive Field Weighted Regression) is an supervised learning algorithm which has been developed by Schaal and Atkeson [Schaal and Atkenson, 1998]. They describe RFWR as a constructive incremental learning system for regression problems[1] that models data by means of locally linear experts [Schaal and Atkeson, 1996]. These experts are trained independently and do not compete for data during learning. Only when a prediction for a query is required the experts cooperate by blending their individual predictions. Each expert is trained by minimizing a penalized local cross validation error using second order methods. Therefore it could find a local distance metric by adjusting the size and shape of the receptive field RF (in our case gaussian) in which its predictions are valid, and also to detect relevant input features by adjusting its bias on the importance of individual input dimensions. RFWR is refered to be a lazy learning method [Atkenson et al., 1997a], as it defers processing of training data until a query needs to be answered. The basic principles of how a RWFR network looks like could be seen in figure 8.1.

Every local model consist of an underlying linear model ($\mathbf{b}_k$ and b_0) and a receptive field (Gaussian with center at $\mathbf{c}_k$ and a distance matrix D_k). The receptive field (RF) determines the region of the input space in which the expert's knowl-

[1] It can be remodeled to learn a classification task.

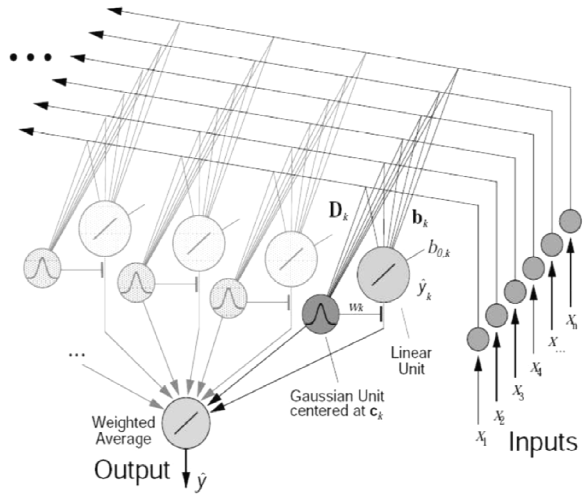


Figure 8.1: Depiction of a typical RFWR network.

edge is valid and weights its contribution to a query.
Every receptive field (RF) is trained independently of all other RFs by adjusting the parameter of the locally linear model, the size and shape of its gaussian RF and the bias on the relevance on its individual input dimensions. The linear model is updated to minimize the weighted squared error by standard recursive Newton training method. The shape and size of the gaussian RF is adjusted recursively by using a stochastic approximation of leave-one-out cross validation using gradient descent. The bias is updated by gradient descent as well. New RFs are allocated or pruned automatically as needed depending on the overlap between different RFs. A new RF is created if a training point does not activate any of the existing receptive fields by more than a fixed threshold (data is suprising). RFs are pruned if they overlap too much, again detected by comparing with a threshold. To query a point a weighted sum of all experts is calculated (see figure 8.1). Every single expert is weighted through it's gaussian kernel (RF) to the query point.
RFWR has several advantages, listed hereinafter.

. It is robust to interference. Since learning takes place only locally new

points far away (e.g. a new task) do not interfere with already learned models.

. It can deal with new input distribution (e.g. a new task). After having learned a mapping for a certain input distribution a new input distribution does not interfere, since there is no global approach.

. It deals automatically with the bias-variance dilemma (i.e. the tradeoff between overfitting and oversmoothing) by using local cross-validation.

. It performs local feature detection by ridge regression parameters, when dealing with locally rank deficient data or with irrelevant input dimensions (local dimensionality reduction). A better version in that sense is LWPR, as discussed later in this section.

. Adding new data is cheap in computational time.

Anyway, at least two drawbacks can be underlined.

. Querying points could be computationally expensive since every expert potentially contributes to the output. Therefore the memory and computation cost increases with the amount of RFs. Atkeson et al. [Atkenson et al., 1997b] claim to be able to reduce costs by using kd-trees. This and other similar algorithms look for nearest neighbours which contribute most to the query point. In general, it is important to limit the overall number of RFs generated by the network by properly adjusting its tunable parameters.

. Generalization depends intensively on the amount of data. Especially when dealing with large input dimensions, learning will not successfully approximate a particular function over the entire space of potential inputs, unless data set is really big. One way to overcome this problem could be local dimension reduction with LWPR. Otherwise, estimation errors in region of the input space which have not been explored intensively are bigger than using other standard machine-learning methods.

Vijayakumar and Schaal [Vijayakumar and Schaal, 2000] extended RFWR to a version called Locally Weighted Projection Regression (LWPR). In comparision to RFWR it's an $O(n)$ algorithm. It reduces locally the dimension with a Linear Projection Regression developed by them to project locally along relevant dimensions. It should outperform RFWR in presence of a big input space with irrelevant dimensions. Anyway, in the problems we faced in this thesis, RFWR and LWPR were behaving similarly.

8.2 Incremental Least Squares

We adapted an incremental least squares regression algorithm from [Hosoda and Asada, 1994], and we used it to estimate local Jacobian matrices both concerning neck control (4.2), reaching control (5.2.2) and obstacle avoidance (6.2). A description of the algorithm here follows. We consider the scenario in which we want to approximate a matrix M for which holds the relation $\mathbf{y}(t) = M\mathbf{x}(t)\ \forall t$. Let's assume we have a current estimation of our matrix $\hat{M}(t - \Delta t)$, or either an initialization matrix from which we start the incremental regression. The matrix estimation at time t is described by the following equation:

$$\hat{M}(t) = \hat{M}(t - \Delta t) + \Delta\hat{M}(t) \tag{8.1}$$

$$\Delta\hat{M}(t) = \begin{cases} \frac{\left\{(\mathbf{y}(t) - \hat{M}(t-\Delta t)\mathbf{x}(t)\right\}\mathbf{x}(t)^T W(t)}{\varrho + \mathbf{x}(t)^T W(t)\mathbf{x}(t)} & if \|\mathbf{x}(t)\| \neq 0 \\ 0 & if \|\mathbf{x}(t)\| = 0 \end{cases} \tag{8.2}$$

where $\mathbf{x}(t)$ and $\mathbf{y}(t)$ are a measured instance of $\mathbf{x}$ and $\mathbf{y}$ at time t, W is a weighting matrix and ϱ is a forgetting factor ($0 \leq \varrho\varrho 1$ must holds). If ϱ is nearly equal to 0, the system becomes sensitive to observed data, while if it is nearly equal to 1, the system becomes insensitive and more stable. When the proposed scheme is applied to a real system, like in our case, ϱ must be tuned properly according to the system characteristics (e.g. noise, sensitivity).

8.3 Hand detection

Hand detection is achieved using a marker: a green ball attached to the robot wrist. The $< u, v >$ position of the ball center on both eyes defines the hand

position in visual field: a cascade of filters is applied to each image in order to determine such position. Here follows a description of the different steps.

. Color segmentation. A HSV model of the ball color is used to segment regions of the image having the same color. After this filter the image becomes binary (i.e. each pixel could be 0 or 1, white or black).

. Elimination of salt & pepper noise. Isolated black pixels are removed, as well as small white holes inside black blobs.

. Edge detection. Standard edge detection (with OpenCV Bradski and Kaehler [2008]) is employed to end up with an image composed of just lines (instead of blobs).

. Hough transform [Duda and Hart, 1972]. An OpenCV algorithm is used to detect circles (and their center) using the hough transform.

8.4 Attention system

James attention system has been adapted from the one developed for the iCub robot [Ruesch et al., 2008]. Anyway, some of its original features are not used in our system. The attention system detects interesting visual regions within the robot visual workspace and sends references to the *gazeController* module, which actuates James neck and eyes in order to gaze toward the interesting point in space. A saliency map is created from images applying a bank of weighted filters (mainly detecting intensity, saturation and motion). Then a module called Attention Selection searches for the maximum salience in the saliency map, defining a target object. Finally, a Binocular Vision system extracts from the current RGB right image the natural cluster (based on first and second order statistics on spatial, 2D, and chromatic, 3D RGB, features) corresponding to the object with the maximum salience. The left part of the vision system searches in the left RGB image the natural cluster having the maximum similarity with respect to the right one. The centres of the two clusters are thus extracted and sent to the *gazeController* module.

References

Albers, A., Brudniok, S., and Burger, W. (2003). The mechanics of a humanoid. In *International Conference on Humanoid Robots*, Karlsruhe, Germany. 28

ATI (1982). F/t sensor: Mini45. `http://www.ati-ia.com/products/ft/ft_models.aspx?id=Mini45`. 38, 101

Atkenson, C. G., Moore, A. W., and Schaal, S. (1997a). Locally weighted learning. *Artificial Intelligence Review*, (11):11–73. 137

Atkenson, C. G., Moore, A. W., and Schaal, S. (1997b). Locally weighted learning for control. *Artificial Intelligence Review*, (11):75–113. 139

Bellman, R. E. (1956). *Dynamic Programming*. Princeton University Press. 18

Berenson, D., Diankov, R., Nishiwaki, K., Kagami, S., and Kuffner, J. (2007). Grasp planning in complex scenes. In *International Conference on Humanoid Robots, Pittsburgh, USA*. IEEE-RAS. 20

Bertenthal, B. I. (2001). *Developmental changes in postural control during infancy*, pages 19–22. Advances in motor developmentand learning in infancy. Amsterdam: Print Partners Ipskamp, j. van der kamp, a. ledebt, g. savelsberg and e. thelen (eds.) edition. 12

Bertenthal, B. I. and von Hofsten, C. (1998). Eye, head and trunk control: The foundation for manual development. *Neuroscience and Biobehavioral Reviews*, (22(4)):515–520. 12

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford Press. 17

Blackburn, M. R. and Nguyen, H. G. (1994). Learning in robot vision directed reaching: A comparison of methods. In *ARPA Image Understanding Workshop*, Monterey, CA, USA. 21

Bloch, H. and Carchon, I. (1992). On the onset of eye-hand coordination in infants. *Behavioral Brain Research*, (49):85–90. 12

Bower, T. G. R. and Broughton, J. M. (1970). The cordination of visual and tactual input in infants. *Perception and Psychophysics*, (8):51–53. 13

Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Press. 141

Brooks, R. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15. 6

Bushnell, E. W. (1981). *The Ontogeny of Intermodal Relations: Vision and Touch in Infancy*, pages 5–37. Intersensory Perception and Sensory Integration. New York: Plenum Press, r. walk and h. pick edition. 15

Cannata, G., Maggiali, M., Metta, G., and Sandini, G. (2008). An embedded artificial skin for humanoid robots. In *Proc. of IEEE Int. Conf. on Multisensor Fusion and Integration for Intelligent Systems*, pages 434–438. 101, 117

Cawley, G. C. (2006). Leave-one-out cross-validation based model selection criteria for weighted ls-svms. In *IJCNN-2006: Proceedings of the International Joint Conference on Neural Networks*, pages 1661–1668, Vancouver, BC, Canada. 106

Chinellato, E. and del Pobil, A. P. (2005). Vision and grasping: Humans vs. robots. In J. Mira, J., editor, *LNCS*, volume 3561, pages 368–377. Springer-Verlag. 134

Ciocarlie, M., Goldfeder, C., and Allen, P. (2007). Dimensionality reduction for hand-independent dexterous robotic grasping. In *International Conference on Intelligent Robots and Systems*, San Diego, CA, USA. IEEE. 133

Clarkson, H. (2000). *Musculoskeletal Assessment*. Lippincott Williams & Wilkins, second edition. 28

Clifton, R., Muir, M. C. D. W., and Ashmead, D. H. (1993a). Is visually guided reaching in early infancy a myth? *Child Development*, 64(4):1099–1110. 21

Clifton, R., Muir, M. C. D. W., and Ashmead, D. H. (1993b). Visual guidance in infants reaching toward suddenly displaced targets. *Child Development*, 64(4):1111–1127. 21

Clifton, R. K., Muir, D. W., Ashmead, D. H., and Clarkson, M. G. (1993c). Is visually guided reaching in early infancy a myth? *Child development*, (64):1099–1110. 13

Coello, Y., Bartolo, A., Amiti, B., Devanne, H., Houdayer, E., and Derambure, P. (2008). Perceiving what is reachable depends on motor representations: Evidence from a transcranial magnetic stimulation study. `http://www.plosone.org/article/info862`. 65

De Luca, A., Albu-Schaffer, A., Haddadin, S., and Hirzinger, G. (2006). Collision detection and safe reaction with the dlr-iii lightweight manipulator arm. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1623–1630. IEEE. 97

Demers, D. and Kreutz-delgado, K. (1992). Learning global direct inverse kinematics. In *Advances in Neural Information Processing Systems*, pages 589–595. Morgan Kaufmann. 17

Denavit, J. and Hartenberg, R. S. (1955). A kinematic notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics*, 23:215–221. 87, 103

Duda, R. O. and Hart, P. E. (1972). Use of the hough transformation to detect lines and curves in pictures. *Communications of Association for Computing Machinery*, (15(1)):11–15. 141

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. Wiley-Interscience. 110

Eibl-Eibesfeld, I. (1970). *Ethology: the Biology of Behavior (E.Klinghammer trans.)*. Holt, Rinehart and Winston. 15

et al., G. R. (1996). Localization of grasp representations in humans by pet: 1. observation versus execution. In *Exp. Brain Res.*, volume 111, pages 246–252. 133

Fernandez, J. J. and Walker, I. D. (1998). Biologically inspired robot grasping using genetic programming. In *IEEE International Conference on Robotics & Automation*. 134

Fitzpatrick, P., Metta, G., Natale, L., Rao, S., and Sandini, G. (2003). Learning about objects through action: Initial steps towards artificial cognition. In *Proceedings of IEEE International Conference on Robotics and Automation*, Taipei, Taiwan. 132

Flanders, M., Daghestani, L., and Berthoz, A. (1999). Reaching beyond reach. *Experimental Brain Research*, 126(1):19–30. 21

Frankenburg, W. K., Dodds, J., and Archer, P. (1992). The denver ii: a major revision and restandardization of the denver developmental screening test. *Pediatrics*, (89):91–97. 12

Fumagalli, M., Gijsberts, A., Ivaldi, S., Jamone, L., Metta, G., Natale, L., and Nori, F. (2010). *Learning How to Exploit Proximal Force Sensing: a Comparison Approach*. Springer-Verlag, from motor to interaction learning in robots edition. 100

Fumagalli, M., Jamone, L., Metta, G., Natale, L., Nori, F., Parmiggiani, A., Randazzo, M., and Sandini, G. (2009). A force sensor for the control of a human-like tendon driven neck. In *International Conference on Humanoid Robots*, Paris, France. 38, 45, 57

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. 40

Gaskett, C. and Cheng, G. (2003). Online learning of a motor map for humanoid robot reaching. In *International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS)*, Singapore. 21

Gould, J. L. (1982). *Ethology: the Mechanism and Evolution of Behavior*. Norton. 15

Guenter, F., Roos, L., Guignard, A., and Billard, A. (2005). Design of a biomimetic upper body for the humanoid robot robota. In *International Conference on Humanoid Robots*, Tsukuba, Japan. IEEE. 24, 28

Hagan, M. T. and Menhaj, M. B. (1994). Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5:989–993. 108

Hay, A. and Snyman, J. (2005). Optimization of a planar tendon-driven parallel manipulator for a maximal dextrous workspace. 37(3):217–236. 27

Hofsten, C. V. (1979). Development of visually directed reaching: the approach phase. *Journal of Human Movement Studies*, (5):160–178. 13

Holland, O. and Knight, R. (2006). The anthropomimetic principle. In *Symposium on Biologically Inspired Robots, Bristol, England*. AISB. 24

Hornik, K., Stinchombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366. 107

Hosoda, K. and Asada, M. (1994). Versatile visual servoing without knowledge of true jacobian. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 48, 118, 140

Hutchinson, S., Hager, G. D., and Corke, P. I. (1996). A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670. 20

Jamone, L., Natale, L., Fumagalli, M., Nori, F., Metta, G., and Sandini, G. (2010). Machine-learning based control of a human-like tendon driven neck. In *International Conference on Robotics and Automation*, Anchorage, Alaska, USA. IEEE-RAS. 46

Jamone, L., Nori, F., Metta, G., and Sandini, G. (2006). James: A humanoid robot acting over an unstructured world. In *International Conference on Humanoid Robots*, Genova, Italy. 23, 132

Jr., R. P., Fagg, A. H., and Grupen, R. A. (2002). Nullspace composition of control laws for grasping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 134

Kandel, E., Schwartz, J., and Jessell, T. (2000). *Principles of Neural Science*. McGraw-Hill, fourth edition. 134

Kendall, F., McCreary, E., Provance, P., Rodgers, M., and Romani, W. (2005). *Muscles: Testing and Function with Posture and Pain*. Lippincott Williams & Wilkins, fifth edition. viii, 29, 30

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98. 42

Kim, H., York, G., Burton, G., Murphy-Chutorian, E., and Triesch, J. (2004). Design of an anthropomorphic robot head for studying autonomous development and learning. In *International Conference on Robotics and Automation (ICRA)*, New Orleans, LA, USA. IEEE. 24, 28

Kinney, H., Brody, B., Kloman, A., and Gilles, F. (1988). Sequence of central nervous system myelination in human infancy. ii.patterns of myelination in autopsied infants. *Journal of Neuropathology and Experimental Neurology*, (47 (3)):217–234. 67

Konczak, J. (2004). Neural development and sensorimotor control. In *Fourth International Workshop on Epigenetics Robotics*, Lund University Cognitive Studies. IEEE. 21

Konczak, J. (2005). On the notion of motor primitives in humans and robots. In *In Proceedings of the 5th International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, volume 123, pages 47–53. 43, 64

Konczak, J., Borutta, M., and Dichgans, J. (1979). Development of goal-directed reaching in infants: Ii. learning to produce task-adequate patterns of joint torque. *Experimental Brain Research*, (113):465–474. 13

Konczak, J., Borutta, M., Topka, H., and Dichgans, J. (1995). Development of goal-directed reaching in infants: Hand trajectory formation and joint force control. *Experimental Brain Research*, (106):156–168. 13

Konczak, J. and Dichgans, J. (1997). Goal-directed reaching: development toward stereotypic arm kinematics in the first three years of life. *Experimental Brain Research*, (117):346–354. vii, 14

Krakauer, J. W., Ghilardi, M. F., and Ghez, C. (1999). Independent learning of internal models for kinematic and dynamic control of reaching. *Nature neuroscience*, 2(11):1026–1031. 11

Krzysztof, K. (1998). *Modelling and Identification in Robotics*. Springer-Verlag New York, Inc. 99, 103, 104

Lagarde, M., Andry, P., Gaussier, P., Boucenna, S., and Hafemeister, L. (2009). Proprioception and imitation: on the road to agent individuation. In *From Motor to Interaction Learning in Robots*. Springer.

Laschi, C., Asuni, G., Teti, G., Carrozza, M. C., Dario, P., Guglielmelli, E., and Johansson, R. (2006). A bio-inspired neural sensory-motor coordination scheme for robot reaching and preshaping. In *The First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics*, pages 531–536. 21

Levenberg, K. (1944). A method for the solution of certain problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168. 107

Liu, G., K.Iagnemma, Dubowsky, S., and Morel, G. (1998). A base force/torque sensor approach to robot manipulator inertial parameter estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*. 97, 104

Lu, S., Chung, J. H., and Velinsky, S. A. (2005). Human-robot collision detection and identification based on wrist and base force/torque sensors. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3796–3801. 97

Maravita, A. and Iriki, A. (2004). Tools for the body (schema). *Trends in cognitive sciences*, 8 (2):79–86. 21

Marjanovic, M., Scassellati, B., and Williamson, M. (1996). Self-taught visually guided pointing for a humanoid robot. In *International Conference on Simulation of Adaptive Behavior*, MA. 21

Marquardt, D. (1963). An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11:431–441. 107

Mason, C. R., Gomez, J. E., and Ebner, T. J. (2001). Hand synergies during reach-to-grasp. *Journal of Neurophysiology*, 86(6):2896–2910. 133

Mathew, A. and Cook, M. (1986). The control of reaching movements by young infants. *Child development*, (61):1238–1257. 13

McIntyre, J., Stratta, F., and Lacquaniti, F. (1989). Viewer-centered frame of reference for pointing to memorized targets in three-dimensional space. *Journal of Neurophysiology*, 62:582–594. 21

Metta, G., Fitzpatrick, P., and Natale, L. (2006). Yarp: yet another robot platform. *International Journal on Advanced Robotics Systems*. Special Issue on Software Development and Integration in Robotics. 39

Metta, G., Sandini, G., and Konczak, J. (1999). A developmental approach to visually-guided reaching in artificial systems. *Neural Networks*, 12(10):1413–1427. 20, 21

Minka, T. (2009). Lightspeed toolbox. `http://research.microsoft.com/en-us/um/people/minka/software/lightspeed/`. 109

Mizuuchi, I., Yoshikai, T., Sodeyama, Y., Nakanish, Y., Miyadera, A., Yamamoto, T., Niemel, T., Hayashi, M., Urata, J., Namiki, Y., Nishino, T., and Inaba, M. (2006). Development of musculoskeletal humanoid kotaro. In *International Conference on Robotics and Automation (ICRA)*. IEEE. 24

Mussa-Ivaldi, F. A. and Bizzi, E. (2000). Motor learning through the combination of primitives. *Philosophical Transactions of the Royal Society: Biological Sciences*, 355:1755–1769. 133

Natale, L., Nori, F., Metta, G., and Sandini, G. (2007). Learning precise 3d reaching in a humanoid robot. In *International Conference of Development and Learning*, London, UK. IEEE. 17, 21

Natale, L., Orabona, F., Metta, G., and Sandini, G. (2005). Exploring the world through grasping: a developmental approach. In *6th CIRA Symposium, Espoo, Finland*. 134

Nguyen, D. and Widrow, B. (1990). Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *Proc. of the Int. Joint Conference on Neural Networks*, volume 3, pages 21–26. 108

Nguyen-Tuong, D., Seeger, M., and Peters, J. (2009). Real-time local gp model learning. In *From Motor to Interaction Learning in Robots*. Springer. 106

Nori, F. and Frezza, R. (2004). Biologically inspired control of a kinematic chain using the superposition of motion primitives. In *Proceedings of 2004 Conference on Decision and Control*, pages 1075–1080. 133

Nori, F., Jamone, L., Metta, G., and Sandini, G. (2007). Accurate control of a human-like tendon driven neck. In *International Conference on Humanoid Robots*, Genova, Italy. 45

Nori, F., Metta, G., Jamone, L., and Sandini, G. (2006). Adaptive combination of motor primitives. In *Workshop on Motor Development part of AISB'06: Adaptation in Artificial and Biological Systems. University of Bristol, England*. 133

Olsson, L., Nehaniv, C., and Polani, D. (2005). Discovering motion flow by temporal-informational correlations in sensors. In *Epigenetic Robotics 2005*. 134

Park, J., Lim, B., Haan, J., and Park, F. C. (2007). Optimal reaching and balancing for humanoid robots. In *Workshop on Robot Manipulation: Sensing and Adapting to the Real World*, Georgia Tech, Atlanta, Georgia, USA. RSS. 20

Penrose, R. (1955). A generalized inverse for matrices. In *Proc. Cambridge Phil. Soc.*, number 51, pages 406–413. 47, 68

Piaget, J. (1954). *The Construction of Reality in the Child*. Basic Books Inc. 10

Rougeaux, S. and Kuniyoshi, Y. (1998). Robust tracking by a humanoid vision system. In *International Workshop on Humanoid and Human Friendly Robotics*, Tsukuba, Japan. 17, 20, 21

Ruesch, J., Lopes, M., Hornstein, J., Santos-Victor, J., and Pfeifer, R. (2008). Multimodal saliency-based bottom-up attention - a framework for the humanoid robot icub. In *International Conference on Robotics and Automation (ICRA)*, pages 962–967. 141

Samson, C., Espiau, B., and Borgne, M. L. (1991). *Robot Control: the Task Function Approach*. Oxford University Press. 47

Santello, M., Flanders, M., and Soechting, J. F. (1998). Postural hand synergies for tool use. *Journal of Neuroscience*, 18(23):10105–10115. 133

Schaal, S. and Atkenson, C. G. (1998). Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084. 17, 48, 72, 137

Schaal, S., Atkenson, C. G., and Vijayakumar, S. (2000). Real-time robot learning with locally weighted statistical learning. In *International Conference on Robotics and Automation (ICRA)*, page 288293. 17

Schaal, S. and Atkeson, C. G. (1996). From isolation to cooperation: An alternative view of a system of experts. *Advances in Neural Information Processing Systems*, (8):605–611. 137

Schmidt, D. C. (2003). The adaptive communication environment. `http://www.cs.wustl.edu/~schmidt/ACE.html`. 40

Schmidt, D. C. and Huston, D. H. (2002). *C++ Network Programming: Mastering Complexity Using ACE and Patterns*. Addison-Wesley Longman. 40

Sciavicco, L. and Siciliano, B. (1996). *Modeling and control of robot manipulators*. MacGraw-Hill. 98, 100

Sentis, L. and Khatib, O. (2006). A whole-body control framework for humanoids operating in human environments. In *Proceedings of IEEE International Conference on Robotics and Automation*, Orlando, Florida, USA. 135

Shinya, M. and Kazuhiro, K. (2003). Collision detection system for manipulator based on adaptive impedance control law. In *International Conference on Robotics and Automation (ICRA)*, pages 1080–1085. IEEE. 97

S.H.Lee and D.Terzopoulos (2006). Heads up! biomechanical modeling and neuromuscular control of the neck. In *SIGGRAPH Conference, Boston, MA*. ACM. 28

Smola, A. J. and Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222. 105

Soechting, J. F. and Flanders, M. (1989). Sensorimotor representations for pointing to targets in three-dimensional space. *Journal of Neurophysiology*, 62:582–594. 21

Steffen, J., Haschke, R., and Ritter, H. (2007). Experience-based and tactile-driven dynamic grasp control. In *International Conference on Intelligent Robots and Systems*, San Diego, CA, USA. IEEE. 134

Suykens, J. A. K., Van Gestel, T., De Brabanter, J., De Moor, B., and Vandewalle, J. (2002). *Least Squares Support Vector Machines*. World Scientific Publishing Co. Pte Ltd., Singapore. 105

Swevers, J., Ganseman, C., Tukel, D. B., Schutter, J. D., and Brussel, H. V. (1997). Optimal robot excitation and identification. *IEEE Trans. on Robotics and Automation*, 3(5):730–740. 98

Thelen, E., Corbetta, D., Kamm, K., Spencer, J. P., Schneider, K., and Zernicke, R. F. (1993). The transition to reaching: mapping intention and intrinsic dynamics. *Child development*, (64(4)):1058–1098. 13

Tikhanoff, V., Fitzpatrick, P., Metta, G., Natale, L., Nori, F., and Cangelosi, A. (2008). An open source simulator for cognitive robotics research: The prototype of the icub humanoid robot simulator. In *Workshop on Performance Metrics for Intelligent Systems*, National Institute of Standards and Technology, Washington DC. 40

Ting, J., Mistry, M., Peters, J., Schaal, S., and Nakanishi, J. (2006). A bayesian approach to nonlinear parameter identification for rigid body dynamics. *Robotics: Science and Systems (RSS)*. 98

Tsai, L. W. (1999). *Robot Analysis: The Mechanics of Serial and Parallel Manipulators*. Wiley-Interscience. 28

Tsoli, A. and Jenkins, O. C. (2007). 2d subspaces for user-driven robot grasping. In *Robotics, Science and Systems Conference: Workshop on Robot Manipulation*. 133

Van der Meer, A. L. H. (1997). Keeping the arm in the limelight: advanced visual control of arm movements in neonates. *European Journal of Paediatric Neurology*, (4):103–108. 18

Van der Meer, A. L. H., Van der Weel, F. R., and Lee, D. N. (1995). The functional significance of arm movements in neonates. *Science*, (267):693–695. 43

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer, New York. 17

Verhoeven, R. and Hiller, M. (2000). Estimating the controllable workspace of tendon-based stewart platforms. In *7th International Symposium on Advances in Robot Kinematics (ARK)*, pages 277–284, Portoroz, Slovenia. 27

Vijayakumar, S. and Schaal, S. (2000). Locally weighted projection regression : An o(n) algorithm for incremental real time learning in high dimensional space. In *International Conference on Machine Learning (ICML)*, pages 1079–1086. 79, 140

Von Hofsten, C. (1982). Eye-hand coordination in newborns. *Developmental Psychology*, (18):450–461. 18, 21

Von Hofsten, C. (1991). Structuring of early reaching movements: a longitudinal study. *Journal of Motor Behavior*, (23(4)):280–292. 13

Wachter, A. and Biegler, L. T. (2006). On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57. 67

White, B. L., Castle, P., and Held, R. (1964). Observations on the development of visually guided reaching. *Child development*, (35):349–364. 15

Wolfe, J., Kluender, K., and Levi, D. (2005). *Sensation and Perception*. Sinauer Associates Incorporated. 36

Yisheng, G. and Yokoi, K. (2006). Reachable space generation of a humanoid robot using the monte carlo method. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1984–1989. 74

Yoshida, E., Laumond, J. P., Esteves, C., Kanoun, O., Sakaguchi, T., and Yokoi, K. (2008). *Whole-Body Locomotion, Manipulation and Reaching for Humanoids*, pages 210–221. Lecture Notes in Computer Science. Springer Berlin, motion in games edition. 20

Zacharias, F., Borst, C., and Hirzinger, G. (2007). Capturing robot workspace structure: representing robot capabilities. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 3229–3236. 74