Scuola di Dottorato in Ingegneria dell'Informazione
Indirizzo in Scienza e Tecnologia dell'Informazione

XXIII Ciclo

Humanoid robot motion creation based on touch interpretation: A new programming paradigm

Dottorando
Fabio Dalla Libera

Supervisore:
Chiar.^{mo} Prof. Emanuele Menegatti

Direttore della Scuola:
Chiar.^{mo} Prof. Matteo Bertocco

Anno Accademico 2010/2011

Abstract

Generating motions for humanoid robots is to this day a very challenging problem. These robots often features a high number of degrees of freedom, 20 or more, even for small-sized ones. This strongly limits the applicability of classical artificial intelligence approaches, and requires the design of specialized solutions.

Some specific tasks, like locomotion or grasping, have been deeply studied, and important concepts like the Zero Moment Point (ZMP) for stable walking or the Grasping Shape Primitives were introduced. Previous relevant work also explored planning strategies for these two important tasks and ongoing research keeps improving the algorithms in terms of speed, robustness and applicability.

When dealing with more generic tasks, a demonstration from a human can be used to derive an initial movement, which is then often optimized by the robot itself. In particular, one of the most common approaches is to directly acquire the movement of a human performer and adapt it to the robot. Ideally, this technique can yield very natural looking motions. This is a strong advantage, since human likeness of motions is very important for communication between human and robots, but no general mathematical formulation has been provided to date.

The retargeting of human motions to robot motions presents indeed several inconveniences. First of all, the instrumentation for acquiring motion data is expensive, requires a careful setup and may not be accessible to most of the users. Second, a human actor who is able to perform the desired movement may not be available. Finally, the differences in the shape, degrees of freedom, power and weight distribution between humans and robots may require an intensive modification of the human motion in order to adapt it to the robot. During this adaptation process, the quality of movement appearance can end up to be significantly reduced.

An alternative approach that does not suffer from these issues is the direct physical interaction between the robot and a human teacher. In a typical setup the instructor moves the humanoid's limbs, showing the robot how the task needs to be accomplished. The idea behind this technique, called "kinesthetic demonstration", actually appeared very early in robotics. Although under different names like "teach-in", "guiding", "play back", "direct teaching" or "walk-through programming", it constitutes one of the most effective methodologies for programming industrial robot arms.

When teaching a robot using this technique the joints are usually let free to move. This approach has the disadvantage that the joints move under the force of gravity, so some setups include gravity compensation or workarounds to make the joints passive only when the teacher touches the robot. Apart from applying forces that compensate for gravity, when kinesthetic demonstration is used the robot usually just responds passively to the forces applied.

In order to ease the teaching process, this thesis proposes a new paradigm, termed

“teaching by touching”. This approach consists in having the robot interpret the meaning of the tactile instructions it receives and move based on its own understanding of the user’s will, instead of limiting its behavior to a mere passive movement.

For instance, if the robot is squatting, and the user pushes the sensors on both sides of the robot, the robot will guess that the user wants it to stand up and will apply by itself forces on its knees. Using the classical approach, instead, the robot would not move when touched on both sides, and would require the user to actually lift up the robot’s body.

In other terms, the proposed methodology changes the way of interacting between humans and robots: with the classical, kinesthetic demonstration approach, the robot moves passively under the user forces like a puppet, while within the teaching by touching approach, touch is considered a way of communication between humans and robots that allows the robot to take an active role in the teaching process.

Among the many ways of communication between humans, touch is actually one of the least studied but most powerful. Touch is often used by instructors in sport or dance classes to adjust the student’s posture in a very intuitive way. Tactile instructions thus appear to be a very appealing modality for developing humanoid robot motions as well.

Interpretation of tactile instructions spontaneously given by human teachers reveals to be a complex task for artificial systems. In fact, the way users employ touch to communicate their intention is still completely unexplored even in the interaction between humans, and models are not available. This thesis reports the first results obtained in the realization of a system for robot motion creation based on tactile interaction.

In particular, it will be shown that the meaning of tactile instructions is both context dependent and user dependent. An example of context dependence of tactile instructions can be readily provided. If a user presses the upper part of the leg when the robot is standing he or she could imply that the robot should bend the leg backwards. However, when the robot is squatting, the same touch on the leg could mean that the robot should bend its knees further.

In regard to user dependence, experiments showed that when asked to interact freely with the robot, different people tend to give different meaning to similar tactile instructions. In detail, preliminary results seem to suggest that the differences in the way of teaching could be partially explained by the diverse abstraction level at which teachers provide their instructions. For instance, some subjects decided to use a nearly direct mapping from a small set of sensors to the joints. Other users adopted a mapping between sensors and joints that appears to be derived from physical considerations. Expressly, the relationship between the pressed part and the way the robot should move according to the user can be estimated by imagining each joint to have a spring inside and to move according to the applied force. Finally, other people employed a mapping between a single touch and a complete movement. Using this approach, for instance, a step with one leg is symbolized by a tap on the knee of that leg.

The experiments were conducted initially with simulated touch sensors. A technique that permits the interaction with virtual sensors displayed on a touch screen while making the real robot execute the motion will be introduced. The strongest advantage of this setup is that it allows applying teaching by touching even to cheap humanoids on the market that are usually lacking tactile sensors.

A second set of experiments was then conducted with M3-Neony, a new humanoid robot equipped with touch sensors over the whole body, that will be shortly introduced. Data analysis confirmed the complexity of the mapping between touch instructions and the movement to be executed. On the other hand, it will be shown that experimental data suggest that the way in which the robot should respond to touch instructions can be described by using a low-dimensional subspace of the complete joint space.

Interestingly, this subspace seems to be highly correlated to the subspace where the motion being developed lies. This fact suggests that the postures assumed by the robot during the movement could be used to improve the interpretation of tactile instructions.

The final part of the thesis will briefly deal with very simple techniques used to improve the robustness of the motions taught to the robot. In detail, two approaches that use sensory information to automatically make small modification to the movement during its execution will be introduced. The first, graph based one, simply tries to make the robot return to a known state as soon as possible when a perturbation brings the robot state far from the expected one. The second approach, inspired from the chemotaxis of bacteria like Escherichia Coli, works by the addition of random noise of intensity that depends on whether the robot state evolves as expected or not.

Sommario

Nonostante la ricchezza di soluzioni proposte in letteratura, la generazione di movimenti per robot umanoidi si rivela ancora un problema non risolto. I robot umanoidi presentano spesso oltre 20 gradi di libertà, anche nel caso di umanoidi di dimensioni ridotte. Questo rende impossibile applicare gli approcci classici dell'intelligenza artificiale, e richiede lo sviluppo di soluzioni specifiche per ciascun problema.

Alcuni task particolari, come la locomozione o il grasping, sono stati studiati approfonditamente, e sono stati introdotti importanti concetti come lo Zero Moment Point (ZMP) per la camminata stabile o le grasping shape primitives per il grasping.

Quando si ha che fare con task più generici, una dimostrazione da parte di un essere umano può essere utilizzata per derivare un movimento preliminare, che può poi venir ottimizzato dal robot stesso. In particolare, uno degli approcci più comuni é far eseguire il movimento direttamente ad un attore umano, registrarlo ed adattarlo al robot. Questa tecnica si rivela efficace nel produrre movimenti che appaiono naturali, e viene infatti spesso utilizzata nel campo della computer grafica. Questo è un grosso vantaggio, in quanto, sebbene avere movimenti naturali risulti importante per la comunicazione tra utenti e robot, al momento non esistono formalizzazioni matematiche.

Il trasferire i movimenti da esseri umani a robot presenta tuttavia alcuni inconvenienti. Innanzitutto, la strumentazione per l'acquisizione di movimenti è costosa, richiede particolare cura per l'allestimento del sistema e non è fruibile alla maggior parte degli utenti. In secondo luogo, un attore in grado di compiere il movimento che si vuole far eseguire al robot potrebbe non essere disponibile. Infine, le differenze di forma, gradi di libertà, forza e distribuzione del peso tra esseri umani e umanoidi potrebbe richiedere di modificare in modo sostanziale il movimento. Durante questo processo di adattamento, caratteristiche importanti del movimento potrebbero venir modificate con una conseguente perdita di naturalezza del movimento.

Un approccio alternativo che non presenta questi problemi è ottenere il movimento mediante interazione diretta tra il robot ed un insegnante umano. Un modo tipico di effettuare questo trasferimento di conoscenza è prevedere che l'insegnante muova direttamente gli arti del robot, mostrando come il task debba essere eseguito. L'idea di base di questa tecnica, denominata "kinesthetic demonstration", è comparsa molto presto in robotica, e sebbene sotto diversi nomi come "teach-in", "guiding", "play back", "direct teaching" o "walk-through programming", costituisce uno degli approcci più efficaci e diffusi per la programmazione di bracci robotici industriali.

Solitamente quando viene impiegato questo approccio i giunti vengono lasciati liberi di muoversi. Se non vengono presi opportuni accorgimenti, quindi, vi è lo svantaggio che il robot si muova per il semplice effetto della forza di gravità. Sistemi per compensare automaticamente la forza di gravità sono stati proposti in letteratura. Altre soluzioni adottate sono rendere i giunti passivi solo localmente, quando l'utente tocca il robot.

A parte eventuali compensazioni della gravità, quando viene impiegata la kinesthetic demonstration il robot si muove in modo totalmente passivo sotto effetto delle forze applicategli.

Allo scopo di facilitare l'insegnamento di movimenti, questa tesi propone un nuovo paradigma di programmazione, chiamato "teaching by touching". Questo approccio consiste nel prevedere che il robot interpreti il significato delle istruzioni tattili che riceve, e si muova in base alla stima dell'intenzione sottostante all'istruzione tattile, anziché limitarsi a muoversi passivamente sotto l'effetto delle forze applicate.

Per esempio, se il robot è accovacciato, e l'utente preme i sensori sui fianchi del robot, il robot potrebbe intuire che l'utente desideri che il robot si alzi, e potrebbe applicare forze agli arti inferiori in modo da assumere una posizione eretta. Viceversa, se viene utilizzata la kinesthetic demonstration classica, il robot non si muove quando gli vengono applicate forze ad ambo i lati del corpo, e per fare in modo che il robot si alzi l'utente deve esplicitamente sollevare il robot. In altri termini, la metodologia proposta cambia il modo di interagire tra esseri umani e robot: con la kinesthetic demonstration classica il robot si muove passivamente come farebbe una marionetta, mentre con il teaching by touching, le istruzioni tattili sono considerate come una forma di comunicazione tra uomo e robot ed il robot assume un ruolo attivo nel processo di apprendimento.

Tra le varie forme di comunicazione tra esseri umani, il tatto è infatti una di quelle meno studiate ma più efficaci per trasferire una grande varietà di informazioni. Si pensi ad esempio a come, con semplici tocchi, istruttori di sport o di danza riescano a suggerire ai loro allievi come modificare il proprio movimento. Osservazioni di questo tipo, suggeriscono come il tatto risulti quindi una forma di comunicazione interessante anche per la comunicazione tra esseri umani e robot.

L'interpretazione di istruzioni tattili fornite in modo spontaneo da insegnati umani risulta però essere complessa per sistemi artificiali. Infatti, il modo in cui gli esseri umani usano il tatto per comunicare le loro intenzioni è un campo completamente inesplorato, e modelli di come le istruzioni debbano essere interpretate sono completamenti assenti in letteratura. Questa tesi riporta i primi risultati ottenuti nella realizzazione di un sistema di sviluppo di movimenti per robot umanoidi basato su interazione tattile.

In particolare viene dimostrato come il significato delle istruzioni tattili dipenda sia dal contesto che dall'utente che le fornisce. Un semplice esempio di dipendenza dal contesto può essere facilmente fornito: se il robot è in posizione eretta e l'utente preme la parte superiore della gamba allora ci si può aspettare che l'utente desideri portare indietro la gamba, mentre se il robot è accovacciato si può pensare che l'utente desideri che il robot pieghi le ginocchia per accovacciarsi ulteriormente.

Per quanto riguarda la dipendenza dall'utente, esperimenti hanno dimostrato che, quando lasciati liberi di interagire a piacimento con il robot, utenti diversi tendono talvolta ad associare significati sostanzialmente differenti a istruzioni molto simili. In particolare, i risultati riportati in questa tesi sembrano suggerire che le differenze nel modo di insegnare di utenti diversi possano essere interpretate come differenze nel livello di astrazione usato da diverse persone. Per esempio, alcuni utenti hanno deciso di utilizzare un mapping fisso tra sensori e giunti. Altri soggetti hanno adottato una relazione tra sensori premuti e movimenti che sembrano derivare da considerazioni fisiche, ed in particolare dall'immaginare che i giunti siano elastici e che il robot si muova sotto l'effetto delle forze applicate dall'utente o dalla reazione del pavimento. Infine, altri utenti impiegano un mapping tra un singolo tocco ed un movimento completo. Con questo approccio, ad esempio, la pressione di un ginocchio è associata ad un passo effettuato con la relativa gamba.

Gli esperimenti presentati sono stati condotti inizialmente con un sensori di tatto simulati. Una tecnica che permette di interagire con sensori simulati, visualizzati su un touch screen, è introdotta nella tesi. Il vantaggio di questo tipo di approccio sta nel fatto che permette di applicare la tecnica del teaching by touching anche agli umanoidi di basso costo tipicamente disponibili sul mercato.

Un secondo insieme di esperimenti è stato condotto con M3-Neony, una nuova piattaforma robotica dotata di sensori tattili su tutto il corpo. L'analisi dei dati ha confermato la complessità della struttura del mapping tra istruzioni tattili e modifiche del movimento. D'altro canto, i risultati sperimentali indicano che i movimenti attesi come riposta alle istruzioni tattili giacciono in un sottospazio dello spazio dei giunti.

In particolare, il fatto interessante è che tale sottospazio coincide con in sottospazio in cui può essere proiettato il movimento che l'utente sta sviluppando. Questa constatazione suggerisce che le posture assunte dal robot durante l'esecuzione del task possono essere impiegate per migliorare la stima del significato delle istruzioni tattili.

La parte finale della tesi presenta brevemente due semplici tecniche che possono essere utilizzate per migliorare la robustezza a perturbazioni esterne dei movimenti generati. La prima, basata sulla costruzione di in grafo, tenta di far ritornare il robot in uno stato conosciuto il prima possibile dopo che una perturbazione porta lo stato lontano da quello aspettato. Il secondo approccio, derivato dalla chemiotassi di un batterio denominato Escherichia Coli, si basa sull'aggiunta di rumore al controllo, con un intensità che dipende dal fatto che lo stato evolva nel modo atteso o meno.

Acknowledgements

I would like to thank my supervisor, Prof. Emanuele Menegatti, and Prof. Enrico Pagello. This thesis would not have been possible without their support, the winter schools they suggested me to attend or the Prof. Hiroshi Ishiguro laboratory they introduced me to. I would like to show my gratitude to them for all the freedom they gave me in conducting my research. I am grateful to Alessandra Calore, for her extreme kindness and her efficiency in dealing with all the Ph.D. school bureaucracy. I would like to thank Prof. Monica Reggiani, Marco Antonelli, Stefano Ghidoni, Alberto Pretto, Daniele Masato, Fabrizio Lana, Paolo Repele and Salvatore Anzalone for their help as well.

I owe my deepest gratitude to Prof. Hiroshi Ishiguro for all the support he gave me in conducting my research. It is an honor for me to being working with him. I am very grateful for his comments on my work, that allowed me to steer the research toward unexplored and interesting areas. I am indebted to Prof. Takashi Minato for guiding me since the beginning of my Master's thesis. He is the person who mostly influenced my way of conducting research, having taught me how to use the research equipment, how to conduct experiments, how to present my ideas, how to prepare papers, posters or slides. The idea of using touching for developing robot motions, developed in this thesis, is actually a suggestion from Prof. Minato as well. I really enjoyed conducting my research with him, and I wish to work with him again in the future.

Most of the work presented in this thesis would have not been possible without the JST ERATO Asada Project. It is therefore a pleasure to thank Prof. Minoru Asada for all his support and kindness. I owe much to ERATO members and Asada laboratory as well. I would like to thank Prof. Koh Hosoda, Prof. Yuichiro Yoshikawa and Prof. Masaki Ogino for their warm kindness. I would like to express my sincere gratitude to the secretaries, and in particular to Yoshimi Yamada, Ayako Fukunaga, Kumiko Shingu and Hideo Maeda for their gentleness. They always helped me much more than it was required by their own work, and with no doubt I feel to list them as nice friends much more than as coworkers.

I am glad to thank Shuhei Ikemoto and Toshihiko Shimizu for their true friendship, and for helping me countless times, in research as well as in daily life. I really enjoyed the time spent together, and I wish to keep working together in the future. Similarly I want to thank Tomoyuki Noda, Hidenobu Sumioka, Katsushi Miura and Kenichi Narioka for all their help and nice time spent together. I owe much to the RoboCup team members, and in particular to Yuki Sasamoto, Ayaka Ikejima, Masanori Morizono and Keita Ogawa who timely helped me a lot of times. I also feel indebted to Tsukasa Nakano, Hisashi Ishihara, Takanori Nagura, Sawa Fuke, Shinya Fujiki, Kazuki Takahei, Joschka Boedecker, Norbert Michael Mayer, Mai Nguyen, and Woori Bae.

I thank very much Takeshi Arabiki for all his help with my Japanese, as well as for the interesting discussions on software or data mining. I would like to thank very much Koki

Sone for his help, too. I owe much to Richard Christopher Keely, Jacob Pyke and Dale Thomas for checking my English as well. I am glad to thank Li Zi Xi for his kindness. Special thanks go to Fransiska Basoeki as well, for the effort she puts in conducting her research and for her help on this thesis.

Finally, I thank my family for their constant and invaluable support.

Table of Contents

1 Introduction

The field of robotics nowadays comprises a huge variety of robot types: industrial robot arms, wheeled robots, quadrupeds, submarine robots and flying robots, just to cite some. Among these many categories, humanoid robots draw particular attention.

Several reasons motivate research on humanoid robotics. First of all, much of the tools, buildings, infrastructures are built for humans, with size and features suitable to people. Stairs, knobs, vehicle handles and pedals are designed for humans. Robots with a structure close to the human shape would be able to move in human environments and use commonly available tools, without the need to create new facilities or devices for the robots [1].

Secondly, robots are supposed to closely interact with humans in the near future. Humans find easier to understand and communicate with artifacts with anthropomorphic shape. As an example, just notice that most of Disney characters assume human form, although they're animals. In fact, human-human communication is very rich, and contains a lot of non-verbal clues, like pointing gestures [2, 3] or face expressions [4].

Last but not the least, by studying humanoid robots it is possible to understand more about humans. In particular in recent years particular attention is given to Cognitive Developmental Robotics [5], a new discipline that aims at studying mechanisms of human intelligence, cognition, and development with robots and computer simulations. Usually, in this approach, models of infant development deriving from brain science or psychology are implemented in a humanoid robot with an anthropomorphic structure and a human-like perceptual system. The actual implementation allows verifying the hypothesis by comparing the phenomena observed in the resultant robot behaviors with the real infant development. The growing interest in CDR and the consequent need of new baby robot platforms seems to be confirmed by the recent development of Child robots like CB^2, Child-robot with Biomimetic Body [6], and iCub [7]. This thesis will briefly introduce a new, small size child robot, called M3-Neony, in Section 4.1.

Much research is being conducted using small size humanoids. Although these humanoids cannot employ tools designed for humans, they present advantages in terms of reduced cost, eased maintenance and, thanks to the limited torques of their motors, safety.

The control of humanoid robots reveals to be very difficult due to their high number of degrees of freedom. Even small humanoid robots often present over 20 degrees of freedom (DOFs). When dealing with humanoids, therefore, the curse of dimensionality problem [8] rules out the straightforward application of most classical Artificial Intelligence algorithms.

Specific solutions for humanoids were thus presented. For instance, the use of complete algorithms like [9] reveals unfeasible so the Rapidly-exploring Random Trees [10] (RRTs) were proposed as a possible solution for motion planning [11]. When using these algorithms, the space of configurations is sampled randomly, with a heuristic that allows a rapid exploration of large unsampled areas and sample with increasing density in all the regions of the configuration space.

Similarly, classical reinforcement learning [12] cannot be used to cope with humanoids movements, and specific solutions were thus proposed. In fact, reinforcement learning shows very good performances in low-dimensional, noise-free grid world. However, when dealing with real world problems, the state and action spaces become highly dimensional and often continuous. The approach usually taken is defining the value function and the actions as a parametric function, for instance as a linear combination of nonlinear functions. Research in that directions can be found in the humanoid field, for example, in [13], where predefined basis functions are used to express the movement and in [14], where the learned parameters of a dynamical systems are used to generate an appropriate control signal.

Other approaches to reduce the state space when dealing with real world problems are extracting high level features, like the presence or absence of an object, from sensory data. This, however, both decreases the generality and, depending on the feature extractor, could make the system decisions sensitive to noise. More general techniques, like the G algorithm [15] or the U-tree [16], try to split the space only when necessary, but require the acquisition of many data for having significant statistics.

In addition to the dimensionality problems, real world settings present the problems of perceptual aliasing [17], i.e. qualitatively different world states are perceived as the same by the sensory input. Furthermore, while in simulated worlds changes are usually assumed to occur just as a consequence of robot actions, in the real world setup events could change the world condition even if no actions are taken by the robot. In other terms, therefore, while often grid worlds can be considered as Markov Decision Processes (MDPs), real world problems often require to explicitly consider the robot-world interaction as a Partially Observable Markov Decision Process (POMPD).

Furthermore, real world tasks often involve achieving a sequence of subgoals. In order to have reasonable learning times, specific solutions for dealing with subgoals must be taken. Examples of these heuristics are the construction of a different state space for each of the subgoals [18], the realization of hierarchical architectures [19], or the use of "reward shaping" [20], i.e. specifically giving rewards for the subgoals, in terms of "progress metrics".

Given all these problems, to have reasonable learning times when dealing with humanoids, the approach often taken to learn a good movement is to define the motion parametrically and optimize the free parameters. Plenty of parameterizations can be found, for instance, for walking patterns [21, 22]. Since smoothness is often a desirable aspects, and since often the actuators present a low-pass filtering effects in their response, the usage of splines for defining the movement is one of the most diffused approaches [23, 24]. Definition of the movement in the frequency domain can be also found in literature [25].

Another approach often adopted is defining the movement as a composition of "basis functions" or "motion primitives" [26]. Actually, living organisms seems to adopt a similar strategy. Studies on frogs and rats [27] reveal in fact that the movements may be generated by a superposition of basic primitives. Similarly, studies on humans show that, despite the extremely high number of degrees of freedom, locomotion [28] or hand movements [29] can be approximated very well even in very low dimensional subspaces. Actually, similarity between humans and humanoid robots also motivated the extraction of motion primitives from human movements, for their usage in control [30, 31] or even in hardware design [32].

One important aspect of the motion representation is whether it is designed to be used open-loop or if it can include feedback from the sensors. Among the motion representation that are designed to undergo on-line modifications depending on the sensory information, we can cite the Nonlinear Dynamical Systems movement primitives [33]. In this setup, essentially, the movement is encoded as a low dimensional dynamical system of predictable behavior, whose trajectory is mapped, by a learned nonlinear function, to actual motor commands. The dynamical system has the role of giving a "phase" or "virtual time" to the whole system. By modifying the evolution of this phase variable, the whole robot movement is changed. For discrete movements, a dynamical system that converges to an attractor point [34] as a leaky integrator is employed, while for periodic movements, oscillators can be used [35].

Humanoid robot periodic motions are actually often generated by oscillators, or better interconnection of oscillators [36, 37]. This approach takes inspiration from "Central Pattern Generators" (CPGs), group of cells present in animals that exhibit periodic activity and are responsible of the generation of periodic movements like gait [38]. CPGs present the great advantages of being able to adapt to external perturbation and synchronize with the environment once feedback is introduced. See Section 5.1 for a more comprehensive introduction.

Given the difficulty of automatically learning movements, a lot of research exploit help from humans, an approach also termed as "coaching" [39] when the interaction between the robot and the user is an important factor of the transfer of knowledge. This support from humans given to humanoids, often eased by the similarity of the morphology, is exploited in the so-called learning from demonstration approaches [40, 41]. In this setup a human instructor directly executes the task, or controls the robot, in order to collect information that can be used by the robot to generate a representation of a solution for the task. This solution can also be automatically improved by the robot, once an evaluation function is specified. In fact, often, even a movement shown by a user that does not achieve the task can be a good starting point for an optimization that leads to satisfactory performances [42].

In fact, one of the most direct approaches of teaching a motion to a humanoid is making a person execute the motion. The movement can be recorded, for instance, by using a motion capture system that stores the position of markers attached to the performer's body. The movement is then adapted to the robot's structure so that the motion can be replayed by the humanoid [43]. This motion retargeting approach is often used in computer graphics as well.

In fact, generating human-like motions still reveals to be an open problem, mainly due to the difficulty in providing mathematical formulations. Motion capture systems are thus used to easily obtain human-like motions both for computer graphic character animation [44] and for humanoids [43, 45].

However, this technique presents several inconveniences. First of all, the instrumentation for acquiring motion data is expensive, it requires a careful setup and it may not be accessible to most of the users. Secondly, a human actor who is able to perform the desired movement may not be available. For instance, users may want to teach the robot to execute a acrobatic flip even if they are not able to do it themselves. Finally, the differences in the shape, degrees of freedom, power and weight distribution between humans and robots require an intensive modification of the human motion to make it executable by the robot. During this adaptation process, the quality of the movement appearance may be strongly degraded.

Another very diffused motion generation methodology is direct manipulation of the robot limbs. The trajectories are recorded and then used for motion playback. This technique appeared very early in robotics, in the field of arm manipulators, and was denoted with a great variety of terms such as teach-in, guiding, play back, direct teaching or walk-through programming [46]. The approach is still largely employed and keeps drawing attention [47, 48]. The idea of equipping mobile robots with tactile sensing for recognizing gestures as an intuitive human interface is recurrent in literature [49, 50] as well. In the humanoids field, this technique, recently diffused under the name of "kinesthetic demonstration", appears to be a simple and valid solution for motion development [51]. If several demonstrations are provided, then a probabilistic model of the movement can be obtained [52] to know, for instance, whether when moving an object the final position of the trajectory must be accurate or not.

Surprisingly, nowadays small humanoid robots are still programmed, even in the consumer market, in a very basic, low level way [53, 54, 55, 56, 57, 58]. In particular, the robot motion is defined by a sequence of keyframes. Each keyframe defines the position that the robot must assume at a specified time, and the position that the robot assumes between two keyframes is defined by their interpolation. The position is usually specified by setting the angle of each of the joints, using Graphical User Interfaces (GUIs) that usually consist in simple sliders, which are moved to change the angles. The process is time consuming and counter-intuitive, since inexperienced users often get confused on how the joints should be moved to obtain a posture they have in mind.

Commercial editors, like Robovie Maker for VStone's robot, HeartToHeart for Kondo's robots or RoboPlus for Bioloid robots[1], usually follow this approach, apart from some functions like the possibility of defining conditional or loop execution of motions, modifying the joint posture depending on the accelerometers information or solving some inverse kinematic. Bigger robots, like the HRP-4c, are often programmed in a similar way, although the editors present further facilities like automatic adjustment of the postures to maintain equilibrium [59].

The approach presented in this thesis, called "Teaching by touching", relies on direct physical interaction between the user and the robot. The approach is inspired from the way sport coaches or dance instructors use simple touches to correct their trainee's postures.

In detail, within our setup, the user touches the robots, and the touch is *interpreted* to modify the movement. The concept will be discussed extensively in Chapter 2. In particular, we will present two ways of editing motions. The first approach is, for simplicity, based on frames. In this case, however, the GUI is reduced essentially to a simple time-line and play/stop buttons. The users select the time of the posture they want to modify, and then they touch the robot to communicate how they want to modify the robot's posture. The second approach, presented in Chapter 5, is based on CPGs. In this setup, the users are able to to modify whole periodic motions by simple touches.

The second part of the thesis, and explicitly Chapter 6, deals with simple and computationally lightweight techniques that can be used to improve the robustness of movement execution. Their applicability is not restricted to movements taught with the teaching by touching approach, nor to humanoid robots. However, the combination of the teaching by touching approach and these techniques with minimal hardware requirements appear to be an interesting solution for improving the usability of small humanoid robots that are increasingly available on the consumer market.

Finally, Chapter 7 concludes the thesis by summarizing the main ideas and results presented in the thesis and by illustrating future works.

[1]See `www.vstone.co.jp/` for VStone robot's, `http://www.atr-robo.com/product/maker/robo-maker.html` for Robovie Maker, `http://kondo-robot.com/` for the popular Kondo robots, and `http://www.robotis.com/xe/bioloid_en` for the Bioloid robots.

2

Interpretation of touch instructions

This chapter provides an introduction on the way touch can be used for communication. In particular, Section 2.1 will provide a description on the the role of touch as a communication means between humans. References to works on how touch can be used to improve the communication in difficult environments, that require short response times, will be given. Additionally, devices developed for replacing other sensory modalities, like vision or audition, will be shortly described. The biological structures that permit tactile sensing in humans will be briefly listed.

Section 2.2 will briefly explain the difficulties that artificial agents must face when interpreting the tactile instructions given spontaneously by humans. In detail, examples of why the meaning of the instructions is both user and context dependent will be provided. The section then analyzes the possible alternatives that can be adopted to enable humans and robot to communicate through touch.

In particular, the idea of learning the mapping between tactile instructions and their meaning will be discussed. The possible ways to learn this mapping, and the respective advantages and disadvantages will be considered. Specifically, the section will indicate the reasons that lead to the choice of adopting an online, supervised learning setup for the prototype system described in the next chapter.

2.1 Communication through touch

Abstractly, we regard communication as a process by which a sender encodes a concept into a format suitable for transmission though a medium, and sends this information to a receiver, which then reconstructs (or decodes) it. Human-to-human communication is traditionally divided into verbal, i.e. when concepts are encoded in the form of words, or non-verbal, when they are not [60]. Communication can also be broken down according to the transmission channel used, such as communication via vision, sound, smell [61] or touch. There are a great variety of studies dealing with characteristics of visual communication, for example recognition of human gestures, see for instance [2, 3, 62], or analysis of how a robot's appearance affects human-robot interaction, i.e. the Uncanny valley phenomena [63].

However touch as a communication medium has received considerably less attention. Touch is an important sense in animals and humans. It is the earliest developed sensory system in all animals, in fact at two months of gestation, humans' fingers already make a grasping movement when the palm is touched and during the first year of life, infants learn many things by touching objects with their mouth. Strictly speaking, touch is the stimulation of the skin by mechanical, thermal chemical or electric stimuli that gives as sensations pressure, warmth, vibration, pain, or muscle movements. The outermost layer of the skin is the epidermis, whose surface is constituted by dead cells. Under this layer, the dermis contains connective and nutritive tissues, and specialized nerve cells:

- Meissner's corpuscles, which respond to the lightest form of stimulation (located on hairless parts like fingertips or tongue);
- Pacinian corpuscles, which respond to pressure, vibrations, high frequency sounds (located near the joints and mammary glands);
- Merkel's disks, which respond to constant pressure;
- Ruffini endings, which register pressure and temperature.

Any stimulation of the skin is carried to the spinal cord on nerve fibers that are small for pain and temperature information, and large if they carry mechanical information. The information traveling to the brain then crosses the sensory cortex to the opposite side of the brain where it is processed.

On the cortex, the space reserved for the body parts is a function of the density of the nerves and not of the area of the body. For instance, a great area is reserved for the abundant amount of information coming from fingers and lips. As a proof of different sensitivity of different areas, if the skin is touched with one or two brush hairs and people are asked how many points are felt, in sensitive areas (fingertips, lips, etc), two points are perceived as distinct more readily than in other parts. The temperature sensitiveness is very high, in fact an increase of only three-four degrees of the skin temperature causes feeling of extreme heat, while even a decrease of one or two degrees is perceived as deep cold. Experiments also show that the sensitivity is generally the same for both the left and right sides of the body, but that women are more sensitive than men to skin stimulation [64]. For a more complete introduction on the topic, refer, for instance, to [65].

Touch can also replace other sensory systems in impaired people. For example, devices like the Optacon (OPtical TActile CONverter) have been developed to replace vision. The Optacon permits blind people to read printed material that has not been transcribed into Braille; it consists of an electronics unit connected to a lens module. The main electronics unit contains a "tactile array" onto which the blind person places his/her index finger. The Optacon user moves the lens module across a line of print, and the image under the lens module is transmitted via the connecting cable to the main electronics unit. The tactile array in the main electronics unit contains a matrix of tiny metal rods which are vibrated to form a magnified tactile representation of the image being viewed by the lens module. As the user moves the lens module along the print line, an image roughly the size of one print letter is felt moving across the tactile array from right to left under the user's finger. Similarly, images acquired from a camera can be converted into a tactile representation. In [66] it was shown that blind subject, after few hours of training, were even able to acquire concepts like perspective, shadow, shape distortion, etc.

Devices to convert sounds into tactile stimulation have been developed as well [67]. In [68] a study on the possibility of recognizing words with one of such devices is reported. The experiment shows that subjects learned to identify 50 words in less than 50 hours of training.

Touch is a primitive but very powerful mean of communication. Simulation experiments where subjects are asked to aim at a target with a mouse [69] show that tactile cues can be used to nearly halve the time to initiate the movement toward the target. Similarly [70] shows the feasibility of vibrating units located on a belt for navigating pedestrians or pilots. In particular, experiments showed that the solution remains applicable also in case of external perturbation, as in the case of people piloting fast boats or helicopters. Plenty of applications of such devices are being outlined. For instance, they could reveal a good solution for guiding firefighters in low visibility conditions.

Further examples of possible usages of touch based devices in the sports domain are reported in [71]. The paper shows, for instance, that cues on where to go can be given to soccer players. Hints on how to adjust the posture to maximize the performance can be given to skaters and cyclists. Finally, the work highlights the possibility of using tactile feedback to correct the timing of the movement of rowers.

Touch has an important role for humans since infancy. Plenty of studies show the importance of the tactile interaction between infants and their caregivers [72]. Touch is used to communicate both emotions, like happiness, fear or anger [73], and to convey specific information, like the presence or absence of a caregiver or the identity of the person touching the infant. Studies with toddlers [74] show that touch is so rich and informative that it can provide by itself information on very high level and abstract aspects like the "quality of interaction". At older ages, tactile communication maintains its importance. Tactile interaction can for instance be observed in dance, where it assumes an important role in the coordination between the partners [75, 76]. Similarly sport instructors intuitively correct the movement of their trainee by simple touches.

The intuitiveness and richness of touch makes it an appealing way of interacting with robots as well. Direct physical interaction between robot arms and humans appeared early in robotics, termed as teach-in, guiding, play back, direct teaching or walk-through programming [46]. The approach is still largely employed and keeps drawing attention [47, 48]. The idea of equipping mobile robots with tactile sensing for recognizing gestures as an intuitive human interface is recurrent in literature [49, 50] as well.

Among robots, humanoids are the ones that can probably benefit the most from exploiting haptic communication. In fact, their anthropomorphic shape facilitates social interaction, as people spontaneously tend to attribute human characteristics to inanimate objects, animals or other entities trying to rationalize their actions [77]. The capability of sensing forces can thus constitute a natural interface for human-robot interaction. An example of the intuitiveness that can be achieved in robot programming by exploring direct physical interaction is given by kinesthetic demonstration [52]. Within this approach, users directly grasp and move the robot limbs, providing demonstrations of the task that are sufficient for the robot to extract a model of the movements to be executed.

This teaching method assumes the robot to move passively under the forces applied to it. In fact, kinesthetic demonstration is often used with small humanoids whose motors are switched off. As possible alternatives, a subset of the motors can be made passive only when necessary [78], or compliant actuators can be employed [79, 80].

The approach presented in this thesis is based on the *interpretation* of tactile interactions. In other terms, the robot tries to understand the meaning underlying the touch, and moves accordingly. Within this paradigm, the robot responds actively, and is not limited to a simple passive movement consequent to the external forces. Such an approach presents several advantages in comparison to kinesthetic demonstration. The robot could, for example, apply all the corrections assumed useful. For instance, when receiving instructions on how to modify the motion, the robot could apply small modification in order to satisfy criteria like dynamical stability, or the minimization of the torso oscillations. Similarly, the robot could try to satisfy the estimated user intention while minimizing the load on the servomotors of the knees and the ankles. When the user appear to be trying to set the right part of the body in a posture similar to the left one, the robot could estimate the intention and impose perfect symmetry between the right and left joints, which is difficult to be realized by the user's direct manipulation.

Other advantages of the proposed technique over kinesthetic demonstration can be readily provided. A single touch could be interpreted and correspond to a simultaneous movement of both arms and legs, while it would be very difficult, if not impossible, to move the four limbs of the robot simultaneously with classical kinesthetic demonstration. Additionally, with big robots like HRP-2 [81], kinesthetic demonstration may reveal to be unfeasible if solutions for compliance and gravity compensating are not adopted. The teaching by touching approach does not suffer from these drawbacks since it assumes an active movement performed by the robot.

Kinesthetic demonstration could be considered as a particular case of teaching by touching. In fact, a robot controlled by the teaching by touching paradigm can behave exactly as in the kinesthetic demonstration setup if the algorithm for mapping sensor pressure to movements is set according to inverse kinematics. For this reason, we could state that the teaching by touching approach is at least as much powerful as kinesthetic demonstration.

The concept of teaching by touching is very general. Touches could be converted to body posture modifications, but in some cases, they could correspond to changes in the movement velocity or timing. In a similar way, the approach does not put constraints in the motion representation underlying the robot movement. For instance, Section 3.1 reports a possible implementation when the motion is modeled by a set of keyframes, while Chapter 5 shows how touch instructions can be employed to edit the parameters of a CPG responsible for the robot's motion.

Similarly, touch is a very articulated communication means, exhibiting many features such as the location of its application, its force direction, intensity and time. Also in this case, although the specific implementations we will present take in considerations only some of the features of tactile information (pushing force or pressure time), the idea of teaching by touching is not constrained to a particular feature.

In order to interpret tactile patterns, humanoid robots must be equipped with a model of the way users provide these instructions. Touch interpretation is in fact complex, user dependent and context dependent, as will be highlighted in Section 2.2.

Unfortunately, models of how humans exploit touch to communicate precise information like motion teaching are completely missing in literature. In fact, most of the aspects of tactile communication in humans are still largely unexplored, as recent works in the field of psychology point out [82]. Performing quantitative measurements in the interaction between humans is very complicated, given the richness of tactile information. On the other hand, acquiring data during the interaction with a humanoid robot is much easier. For this reason, one of the ideas presented in this thesis is the possibility of using the teaching by touching approach to investigate people's way of using touch to communicate. One may argue that human-human interaction may differ from human-robot interaction. However, we should notice that often people unconsciously treat robots that appear human-like in a way very close to other humans. For instance, the way people direct their gaze to humans and to mechanical objects is different, but when interacting with androids, people's gaze follows the behavior taken in the interaction with other humans [83].

The teaching by touching approach can therefore be seen also as a tool for measuring features of the tactile interaction between humans and humanoids. Once data from a sufficient number of users is collected, it will be possible to construct a model for the interpretation of touch instructions. This model will shed light on a completely unexplored field of human science, the way humans employ touch to communicate precise information. From an engineering point of view, the model will allow the building of humanoids that are able to understand the meaning of touches, which can be provided spontaneously by users that are not familiar with the robots.

This thesis reports preliminary results on the analysis on the way people utilize touch. In particular, it will be shown that as easily predictable, some basic motions, like closing the arms, are converted by most of the users into similar instructions and that usually users touch the limb they want to move. This appears however not to be true when the users want to convey higher level behaviors to the robot. In the case considered in this paper, for instance, a subject touched the head of the robot to make it squat or touched its side to express the desire to turn the leg and bring the knee outwards. Furthermore, in general, different users give different levels of abstraction to the touch meanings, ranging from "when I press sensor s turn joint j" to "when I touch your knee, execute a step with that leg". Other findings regard the fact that usually the posture modifications desired by the users lie in a (linear) subspace of the motor space. Interestingly this subspace seems to be highly correlated to the subspace where the motion being developed lies. This fact suggests us that the motion itself could be used to improve the interpretation of tactile instructions. More detailed results can be found in Section 4.3.

2.2 Ambiguities in tactile interaction

As seen in the previous section, touch is an important and powerful means of communication. Observing a sport coach or a dance instructor, we often notice that the teacher can convey much information with a single touch. The learners are able to interpret the meaning in a spontaneous way. The same means of communication, naturally shared between human beings, should allow inexperienced users to teach robots new motions.

As previously stated, however, no models of this interpretation are available. Interpretation is complex because the meaning of tactile instructions is both context dependent and user dependent. Figure 2.1 provides an example of how the context can be sufficient to completely change the meaning of the same touch. If users press the upper part of the leg when the robot is standing they could imply that the robot should bend the leg backwards. However, when the robot is squatting, the same touch on the leg could mean that the robot should bend its knees further.

Figure 2.2 shows an example of user dependence. When the robot is standing, as previously stated, for a user touching the upper part of the leg the meaning of the touch could be that the leg should be brought backwards. However, for another user, the same touch could mean that the touched leg should be risen. Actually, this kind of interpretation was actually given spontaneously by a novice user, as reported in Section 3.5.

One simple way to allow humans to communicate with robots through touch would be to design a fixed protocol, and ask the user to learn it and use it. This solution presents several advantages. It is simple to implement, the responses can be easily predicted and the protocol can be optimized with respect to certain criteria. For instance, it would be possible to test a set of different fixed protocols and choose the one that allows faster programming.

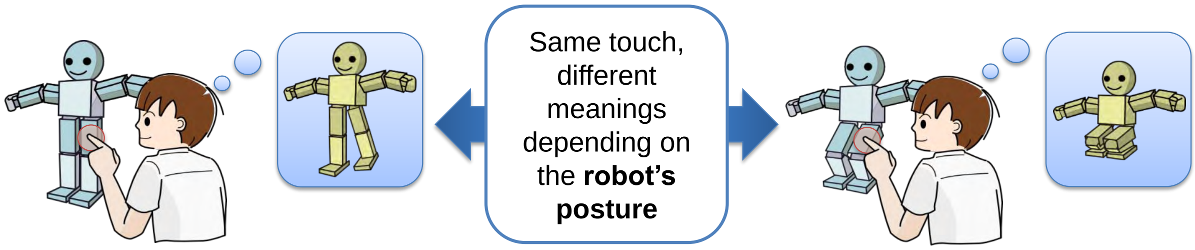


Figure 2.1: Different touch meaning in different contexts. The same touch corresponds to two different intentions depending on the context.

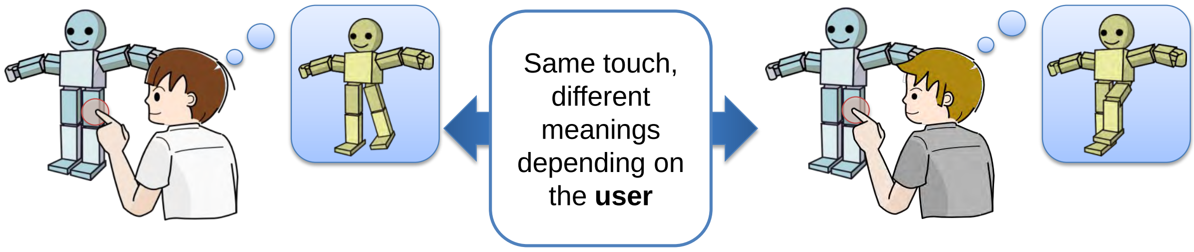


Figure 2.2: Different touch meaning in different contexts. The same touch corresponds to two different intentions depending on the context.

This may be a good solution for robot programmers: an optimized "sensor based language" that allows easy programming. However, as shown in Fig. 2.3 this requires an initial effort in learning the protocol. At least in the beginning phase, each time the users want to communicate with the robot, they must put effort in expressing their idea on how to modify the motion using the protocol available. In other terms, users have an idea on how they want to modify the motion. If they were to interact with another human, they would just touch the trainee without any need on thinking how to express their intention. Conversely, with a fixed protocol, they need to mentally convert their own motion modification intention into a set of touch instructions defined by the protocol. This initial learning effort may be unacceptable for users who teach motions very occasionally. On the robot's side, the interpretation is clearly very easy, since the instructions just need to be converted into a motion modification using the preestablished protocol.

The complexity of the process can however be moved from the human side to the robot side, as depicted in Fig. 2.4. In particular, if the robot is equipped with a better estimator of the user intention, that mimics the interpretation unconsciously performed by a human trainee, then the inexperienced teachers effort is reduced to the minimum. In this setup, users are given no constraints on the way they provide their instructions, and the robot estimator must be able to interpret them, possibly adapting to each particular user. This parallels with what happens between humans: trainees understand their teacher more and more as they spend time together.

One interesting question is whether, for robot programmers, the instructions given

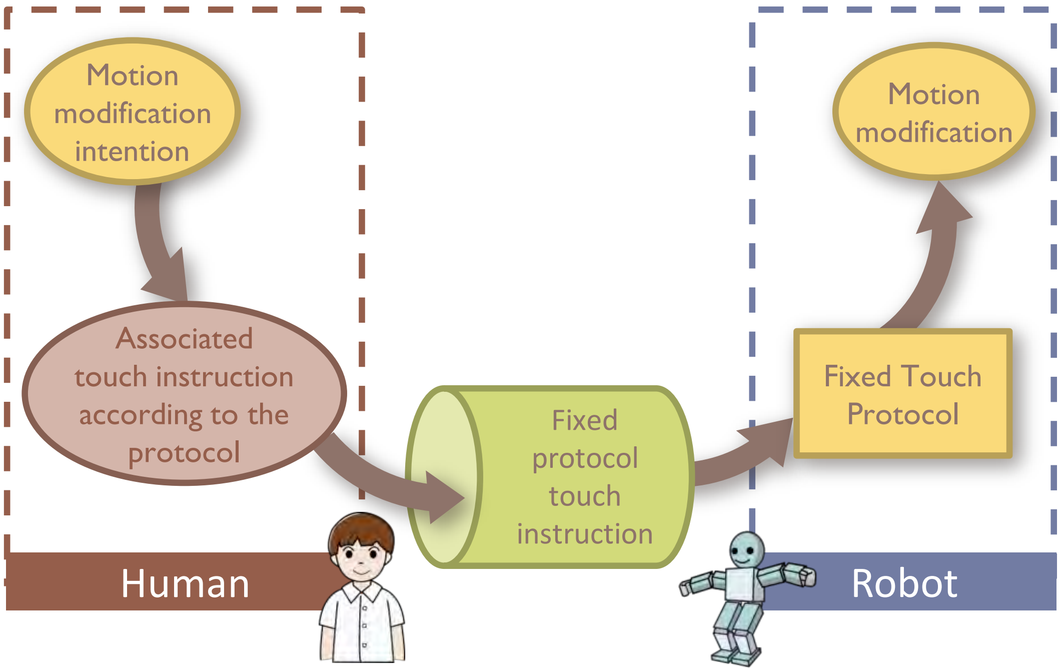


Figure 2.3: Touch interpretation by a fixed protocol. The communication occurs due to the effort of the user who converts its own motion modification intention into instructions that the robot knows.

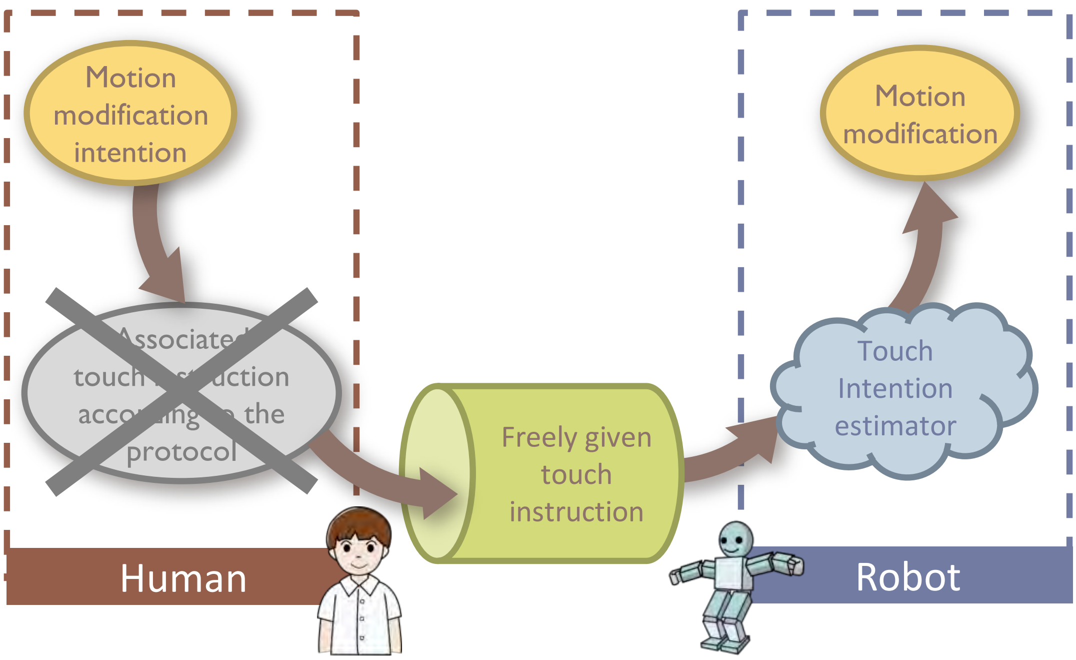


Figure 2.4: Touch interpretation by natural provision of tactile instructions. The communication occurs due to the effort of the robot that converts the naturally provided instructions into actual motion modifications.

unconsciously are the faster way to program the robot or if it would be possible to design a protocol that, once learned, is optimal for programming. In a similar fashion, we could imagine to have several robot programmers interact freely with the robot, and to compare their motion development time, to find out the protocol of "good teachers". It would then be possible to force the programmers to master the protocol of "good teachers", and observe whether such a protocol is in general good for everyone or if unconsciously given instructions are really the best way to program robots not only for inexperienced users but for experienced ones as well.

Given the lack of models for the interpretation of natural instructions, two approaches can be taken. The first one is to actually collect data of the interaction between humans, and derive a model. To collect data, we can design a simple experiment. Imagine having two people. A video of a complex movement, for instance a dance, is shown to one of the subjects, which we call the teacher. The other subject, who acts as a learner, is provided no information on the movement he has to perform. The teacher is required to employ exclusively touch to teach the motion to the learner. In other words, we want to see how a human would behave if it were the robot of our current system. The movements of the two subjects could be easily recorded by a motion capture system, and a data glove could be worn by the teacher to accurately measure the forces he applies. Once we collect data from human-human interaction, we can study an appropriate representation for the touch and context information and derive a model of the interpretation of touch instructions used by the human learners.

The second approach is to directly use a robot, and make it learn the protocol. For learning, there are several aspects to be decided. The first is *when* the touch interpretation should be learned. One approach is to have a special preliminary session, in which the robot is taught the protocol, and then, during motion development, such protocol is used.

While this strategy presents the advantage of clearly separating the two concepts of "teaching the meaning of a tactile instruction" and "teaching a motion", several difficulties arise. Concretely, one possible setup could be to define several couples of robot postures, and ask the users to touch the robot as they would do to bring the robot from the first to the second posture of the couple. This setup presents the problem that it is difficult to imagine beforehand all the possible contexts that the robot will face and for which the meaning of the touch instructions should be taught. Secondly, it may be difficult for users to provide instructions in a natural way when they are requested to bring the robot from a predefined posture to another one. Users may not be able to observe the initial and target posture in a sufficiently accurate way, and may not grasp the actual meaning of that movement, ending up providing an unnatural instruction. To mitigate this effect, the users may be asked to freely teach meaning of tactile instructions. This setup would assure that the instructions are given unconsciously. However, users may restrict their teaching to particular contexts, and the data collected may be unsuitable to learn a protocol that can be used afterward.

Another approach is to have a "touch instruction training" session, but to have teachers teach both the motion and the meaning of tactile instructions at the same time. This solution provides several advantages. The instructions are provided just when the user has a concrete idea on how he wants to modify the motion, and the instructions are provided in a natural way. Furthermore, there is no need of thinking at all possible contexts, because the meaning of touch instructions can be provided at any time. More in detail, such an approach allows teaching the meaning of tactile instructions just when required. Practically, this means that the mapping between tactile instructions and motion modification can be refined more and more just where needed.

The second aspect to be decided is *how* to teach the robot the meaning of tactile instructions. In particular, it must be decided whether to use supervised learning or reinforcement learning. Supervised learning corresponds to giving the robot examples of a tactile instruction in a particular context and the desired motion modification. This is the setup in which the user gives the most rich information to the robot.

Another approach is using reinforcement learning: the robot proposes a possible motion modification to the user, and the user just evaluates the goodness of the response by a scalar (possibly binary) evaluation. It is even not necessary to provide an evaluation at every step. The user could, for instance, provide just negative rewards when the motion modification is an undesired one. This approach requires the user to provide less information, but more advanced algorithms are required. In particular, a naive approach consisting in classical reinforcement learning appears unfeasible. If the robot tries random motion modifications, and the user must evaluate them, then we can expect that an impractically long time will be required before the system converges to good motion modifications for each touch instruction.

Additionally, a possible setup is obtaining the reward information internally, and not asking the user to provide it. Concretely, if the robot is equipped with an evaluation function of the task, then the robot can assume that in general the user instructions are always beneficial for the task, so when the task performance increases, the instruction interpretation was probably correct, and that, conversely, when the task performance decreases, the instruction was misunderstood. Although very appealing, this approach requires a very good task description, that is able to give a good evaluation even to partial improvements of the motion. For instance, if the task is crawling, the evaluation function should consider lifting the arm positively, even if such action alone does not directly reflect into an improvement of the task purpose, i.e. moving forwards.

Chapters 3 and 4 present the implementation of a system where the meaning of instructions is taught during motion development, in a supervised learning setup. In this setup the motion representation is trivial, and based on a sequence of frames, i.e. postures that the robot must assume over time. Touches are actually used to modify single posture. Chapter 5 focuses on the application of the teaching by touching approach to a more interesting motion representation, namely Central Pattern Generators. An implementation where a simple fixed mapping between touch instructions and motion modifications is described. Thanks to the CPG motion description, in this setup a single touch corresponds to a modification of the whole motion.

3

Supervised learning of tactile instructions

This chapter introduces an algorithm for tactile interpretation based on supervised learning. In the actual system implementation, the robot learns the meaning of tactile instructions on-line, i.e. during the motion development. As previously discussed, this presents several advantages. Indeed, the user can provide instructions in a very natural way. Furthermore, the mapping between tactile instructions and motion modifications can be refined where required, i.e. where the robot misunderstands the meaning of tactile instructions.

Figure 3.1 reports a schema of the motion development process in this framework. The motion is taught by successive refinements. At each cycle, the user observes the current state of the motion, decides what to change, and provides a tactile instruction. If the robot correctly understands the instruction ("Correct estimation" path in Fig. 3.1) then the user keeps editing the motion. Otherwise ("Wrong estimation" path in the figure) the user teaches the correct interpretation of the instruction, and then proceeds in editing the motion. When the correct meaning of the touch instruction is given, a new example of an input (tactile instruction) and the corresponding output (the desired motion modification) of the estimator is made available to the robot.

The robot can then use this piece of information to improve its touch instruction interpreter. Concretely, with a model-based method, this would correspond to an update of the model parameters. With a memory based method, as the algorithm presented in this chapter, the new input-output mapping example is simply stored into a database. To prevent any bias in the resulting mapping, the current system starts with an empty database for each new user. The user, however, can reuse the same database over time. This makes the tactile instruction interpreter more and more accurate over time, although specific for a single user. The increase in the accuracy of the interpreter, in turns, translates in a decrease over time of the need of teaching the meaning of touch instructions, as will be shown in Sections 3.5 and 4.3.

In our first implementation, the motion is represented as a set of keyframes. During the motion development, the user selects an instant in time at which the posture should be changed. This can be done by a Graphical User Interface (GUI) that displays a

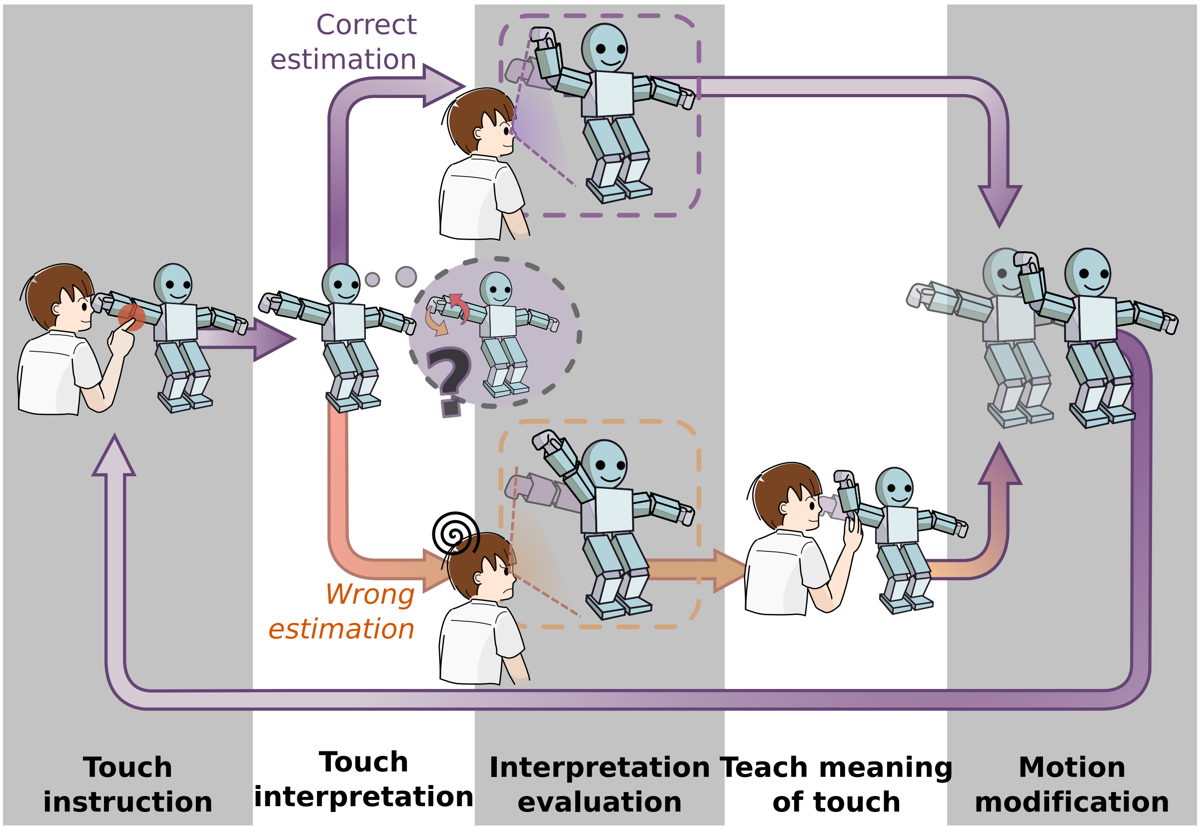


Figure 3.1: A schema of the motion development cycle. The user touches the robot to edit the motion. The robot interprets the meaning of the tactile instruction and changes the motion. The user then evaluates the motion change. If the change corresponds to the original intention then the user will continue to develop the motion, otherwise he or she will teach the robot the meaning of the touch instruction.

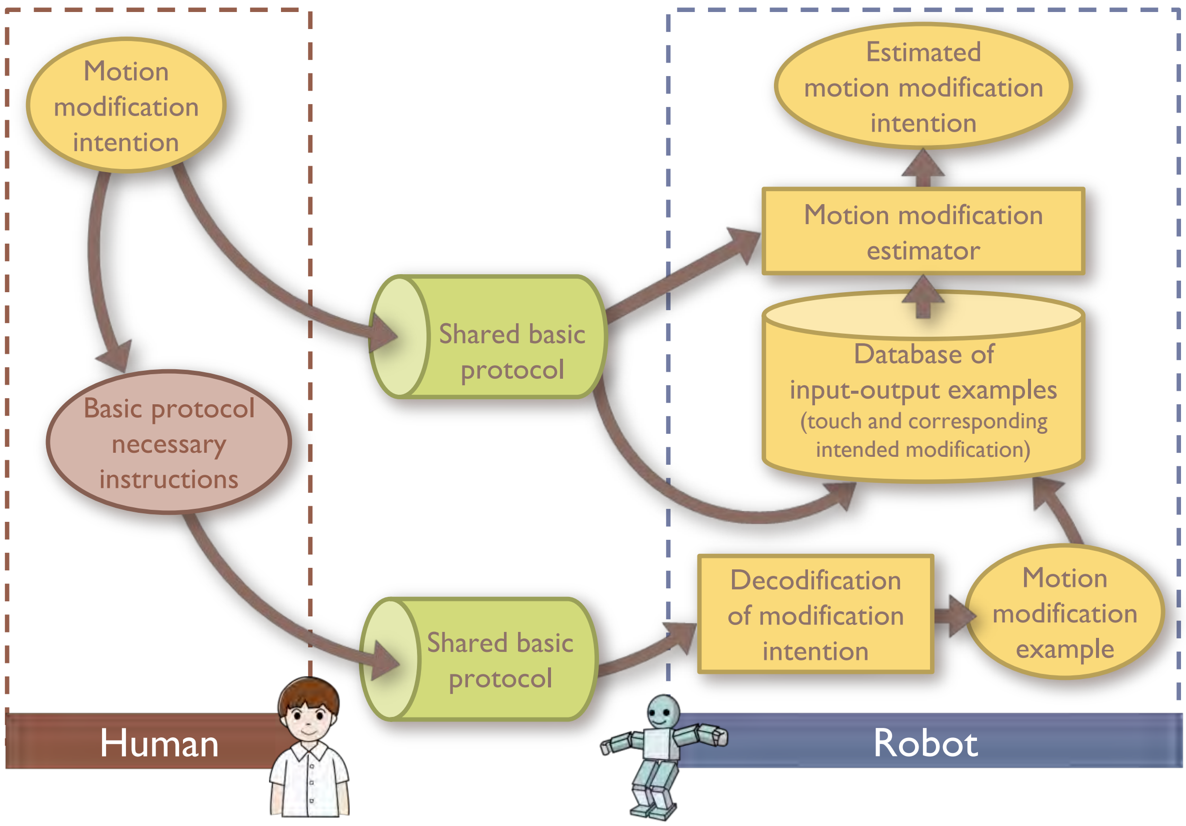


Figure 3.2: A conceptual schema of touch interpretation under the supervised learning framework.

timeline. The timeline consists of a slider, and by dragging the cursor the robot assumes the posture of the corresponding time.

Once the time is selected, the user can edit the posture of that time by touching. The interpreter maps the tactile instructions given to motion modifications. In particular, if a frame based representation is used, as in our case, the tactile instructions can be mapped to changes of the current posture.

If the posture change does not correspond to the desired one, the user provides the meaning of the tactile instruction, i.e. the expected posture change. Figure 3.2 provides a conceptual schema of the approach. In order to teach the correct interpretation to the robot, any communication means shared by the user and the humanoid could be used. Speech recognition with commands like “bring the hand backward” or “raise the knee more” could be employed. This appears as a very natural way of interaction with the robot. However, it would require a lot of effort from the user, who would need to learn the terms and grammar structures understood by the robot. Furthermore, it may be difficult to use words for fine tuning the postures. Learning by watching [84] could also be used, but in this case the complexity of the system would need to be strongly increased.

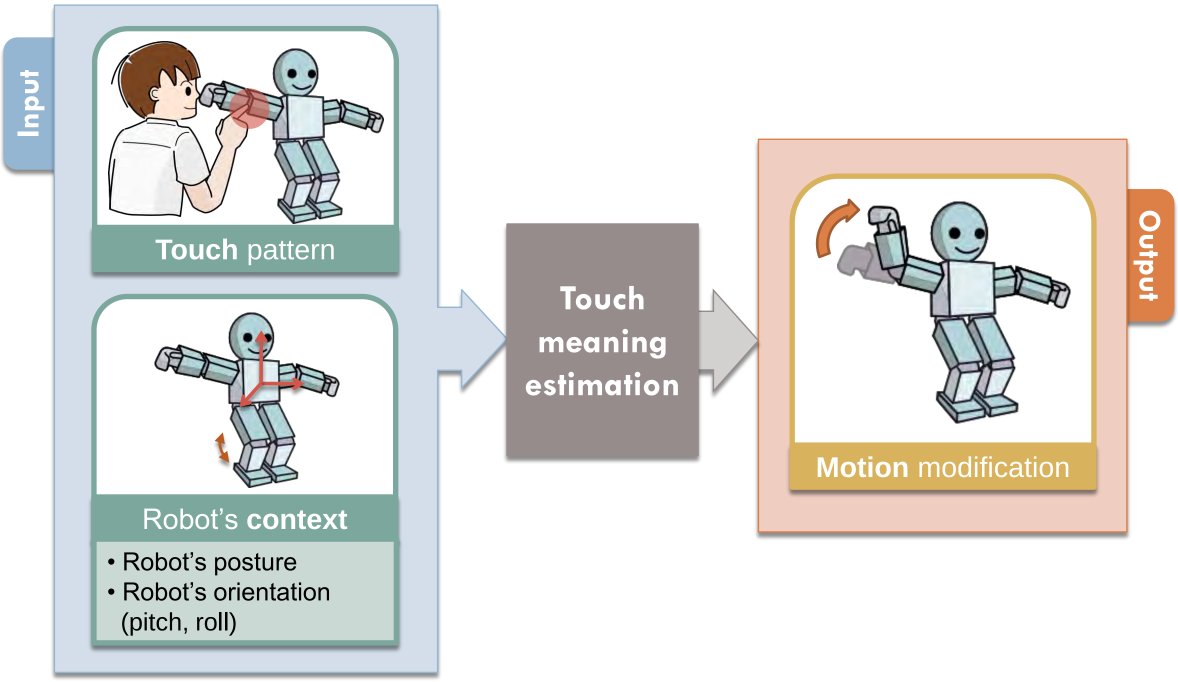


Figure 3.3: The input and the output of the touch instruction meaning estimation.

The system presented in this chapter provides the user with two simple communication modalities. The first is a classic GUI with a slider for each joint: the user can modify each of the joint angles by operating the corresponding slider. The second is kinesthetic demonstration [52]: the user can switch off some of the robot motors, move the joints and reactivate them. When the user finishes editing the posture, it just restarts editing the motion, and the association from the initial posture and the relative context to the provided posture change is stored in the database.

The algorithm used to map a generic touch instruction (given in a certain context) to a posture change is described in the following section. Essentially, the system uses a Locally Weighted Learning [85] with a specifically designed kernel. A proof of concept system implementation will be presented in Section 3.2. Finally, an analysis of user dependence in the meaning of tactile instructions will be provided in Section 3.5.

3.1 Algorithm

The touch instruction estimator, core of the teaching by touching approach, is a module that takes as inputs touch instructions and their context, and gives as output a motion modification, as shown in Fig. 3.3.

As previously stated, a wide spectrum of possible choices are available for its implementation. As discussed in Section 2.2, we decided to realize the mapping between the input and the output using machine learning. In particular, we decided to employ

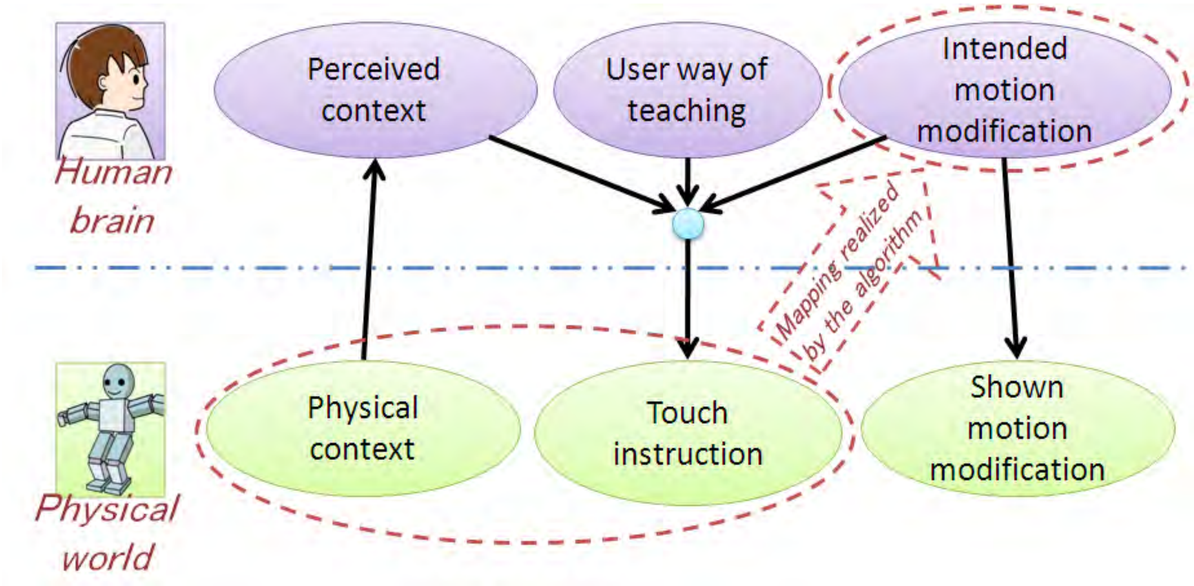


Figure 3.4: Conceptual schema of the teaching by touching setup.

supervised learning.

Figure 3.4 provides a conceptual schema of the elements involved in the teaching by touching approach. At each step of the motion refinement, the human teacher has an intention on how to modify the motion. The robot is in a determinate physical context, and the human user, unconsciously, evaluates some features of this context. Depending on this perceived context, on the personal way of teaching and on the modification intention, the user provides a touching pattern to express his or her will.

The role of the instruction interpreter algorithm is to estimate the user motion modification intention from the tactile instruction and the physical context, that consists, for instance, of the joint angles of the robot, its orientation in the space or the velocities of its limbs.

When the robot fails to estimate the meaning of the instruction, the user communicates the meaning of the instruction through a shared channel. What happens is that the mental image of the motion modification is transferred in the physical world through the alternative communication channel. In the supervised learning framework, this information provides the output of the input-output examples used for training the instruction interpreter.

Several alternatives are available under the supervised learning framework. The first choice is whether to use a parametric model or a memory based approach. A parametric model would require defining a model of the input/output, and a fitting of the parameters. One example would be employing a generative model [86]. The conceptual schema reported in Fig. 3.4 could in fact be readily translated into a generative model, as will be discussed in Chapter 7. This would need, however, some assumption on the family of the distributions of the various elements of the schema. Other possibilities would be, for instance, the training of a neural network [87] or a regression tree with linear models in the leafs [88].

In order to keep the mapping as general as possible, and for simplicity, we decided to employ a memory based learning. This category of approaches are also called instance based learning [89] or lazy learning. The latter name derives from the fact that usually no computation is performed when new input-output examples are given, and the processing occurs only when a query needs to be answered.

The most commonly used class of instance based learning methods is given by locally weighted learning algorithms [90]. These algorithms approximate input-output functions with local models around the query point. Usually this provides a better fitting than function approximations that try to fit all input-output examples with a global function. In the robotics field, these approaches have found interesting applications in challenging robot control problems. A survey can be found in [85].

More precisely, in locally weighted learning, the prediction of the output for a given query input is obtained by looking for examples in memory that have a similar input, by fitting a local model to those points, and by making a prediction based on the model. There are thus four components that define a memory based learner:

1. a distance metric, used for measuring how "similar" the stored inputs are to the current input

2. the number of nearest neighbors considered for the computation

3. a weighting function, or kernel function, that weights the importance of each example in the computation of the output

4. a local model, that is fitted to the points considered and used for the prediction of the output

Typical examples of distance metrics are the Manhattan distance and the Euclidean distance. When the examples are non-spherically distributed points, the variance and covariance between the components of the example inputs should be taken into consideration. In these cases, the Mahalanobis distance is usually employed.

The number of nearest neighbors considered usually depends on the problem. Particular cases are 1, that is considering only the closest example, and considering all the examples. Note that, actually, even when all the examples are considered, most of the weights could be set to 0 by the kernel function.

Weighting functions are usually chosen as decreasing functions of the distance between the current query input and the input of the example. A widespread weighting function is the Gaussian[1]

$$K(d) = e^{-\alpha d}$$

[1] Constant multiplying factors can be omitted as they do not influence the ratio between the weights.

where d is the distance and α is a constant. Another example is the tricube kernel,

$$K(d) = (1 - |d|^3)^3 I_{|d|<1}(d)$$

where I is the indicator function.

The most employed local models are the polynomial ones. In particular, degree 0 for kernel regression, degree 1 for locally weighted linear regression and degree 2 for fitting of quadratic models. More precisely, a widespread model of kernel regression is the one proposed independently by Nadaraya [91] and Watson [92],

$$f(x) = \frac{\sum_{i=1}^{n} K(x - x_i) y_i}{\sum_{i=1}^{n} K(x - x_i)} \tag{3.1}$$

where x is the current input query, n is the number of examples, x_i and y_i are the input and the output of the i-th example, respectively.

For locally weighted linear regression, the model can be easily obtained from classical linear regression. Given the set of examples with input x_i and output y_i, the purpose is finding the best matrix B, in terms of mean square error, such that

$$Y \approx \bar{X} B$$

where the i-th row of Y is given by y_i and the i-th row of X is given by x_i followed by a 1, to make the last row of B represent the constant terms. It can be readily found that

$$B = (\bar{X}^T \bar{X})^{-1} \bar{X}^T Y$$

If different weights are given to the errors of the examples, then

$$B = (\bar{X}^T W \bar{X})^{-1} \bar{X}^T W Y$$

where W is the matrix having the weights on the diagonal.

In model for locally weighted linear regression, the W matrix depends on the query point, i.e. the i-th diagonal term is simply set to $K(x-x_i)$, and the prediction is calculated as

$$f(x) = \bar{x}^T (\bar{X}^T W(x) \bar{X})^{-1} \bar{X}^T W(x) Y \tag{3.2}$$

where again $\bar{x}^T$ is given by x^T followed by a 1 to sum the constant term.

It is possible to observe that Eq. 3.1 of kernel regression can be thought as locally weighted linear regression when only the constant terms (equal to 1) of x and X are considered. In fact, denoting by e_n the column vector of n ones, when considering only the constant terms, Eq. 3.2 becomes

$$f(x) = 1 \cdot (e_n^T W(x) e_n)^{-1} e_n^T W(x) Y$$

that, with the definition of W(x) given, is exactly Eq. 3.1.

For keeping things simple, we decided to adopt a kernel regression method, similar to Eq. 3.1. Given some features of the tactile instructions, a custom kernel was devised.

More specifically, in the current system implementation, the context consists of the angular positions of all the motors, the robot's orientation and its center of mass velocity vector.

This leads to the decision of describing the input feature of the tactile instruction as a vector of "touch intensities", with one element for each of the sensors. Clearly, extensions of the algorithm could consider other features, like a continuous value for the touch location, or discriminate the instructions also depending on the contact area.

Formally, let us denote by n the number of tactile sensors of the robot and by m the number of motors. Let us then indicate by o the number of components used to describe the robot's orientation[2], and by v the number of components that describe the center of gravity velocity, which is 3. The touch interpretation algorithm is a function that given as input $I_* \in \mathbb{R}^{n+m+o+v}$, i.e. the touch pattern and its context, provides an output $M_* \in \mathbb{R}^m$ that expresses an angle modification for each of the joints. Let us denote by E the number of examples of the mapping from touch instructions with their context to posture modifications. For each of these examples let us consider its input $I_i \in \mathbb{R}^{n+m+o+v}$ and the corresponding output $M_i \in \mathbb{R}^m$, $1 \leq i \leq E$. The estimation of the desired joint modification can be calculated as a function of the stored outputs:

$$M_* = \sum_{i=1}^{E} \omega(I_*, I_i) M_i \tag{3.3}$$

where the function $\omega(I_*, I_i)$ is a kernel that gives the similarity between the system input I_* and the input of the i-th example is I_i. The term being weighted, M_i, is the output of the i-th example. In other terms, $\omega(I_*, I_i)$ takes the role of K, I the role of x and M the role of y.

It appears reasonable to choose a function $\omega(I_*, I_i)$ that satisfies the following criteria:

1. The more the touch pattern and context of an example differ from the system input, the less the movement associated to that example contributes to the output.

2. The stronger the user pushes the sensors, the further the robot joints are rotated.

3. The examples whose touch pattern includes the pressure of sensors that are not pushed in the touch instruction I_* provide no contribution to the output.

The first requirement is satisfied by most of the usual kernels, which give a weight that decreases when the distance between the query input and the sample input increases.

[2]In our case $o = 2$, since the orientation is expressed as the inclination and the orientation around the vertical, two quantities easily derivable from the accelerometer and gyroscope data [93].

However, the second requirement, which appears to be very obvious, actually rules out any conventional kernel based on the distance. In fact, any distance function d on a given set M is, by definition, a function $d : M \times M \rightarrow \mathbb{R}$ that satisfies the following conditions:

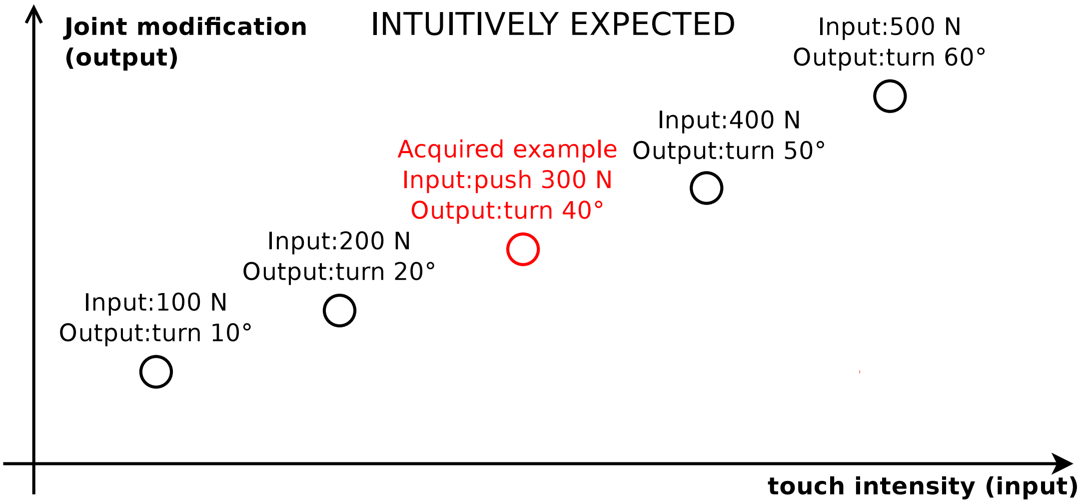
- $d(x, y) \geq 0$, and $d(x, y) = 0$ if and only if $x = y$;
- it is symmetric: $d(x, y) = d(y, x)$;
- it satisfies the triangle inequality: $d(x, z) \geq d(x, y) + d(y, z)$.

Due to the symmetry of the distance function, it is not possible to distinguish whether the current input sensor has a higher or lower pushing intensity than an example in the training set. This would lead, for any distance based kernel, to a very non-intuitive behavior of the touch interpreter.

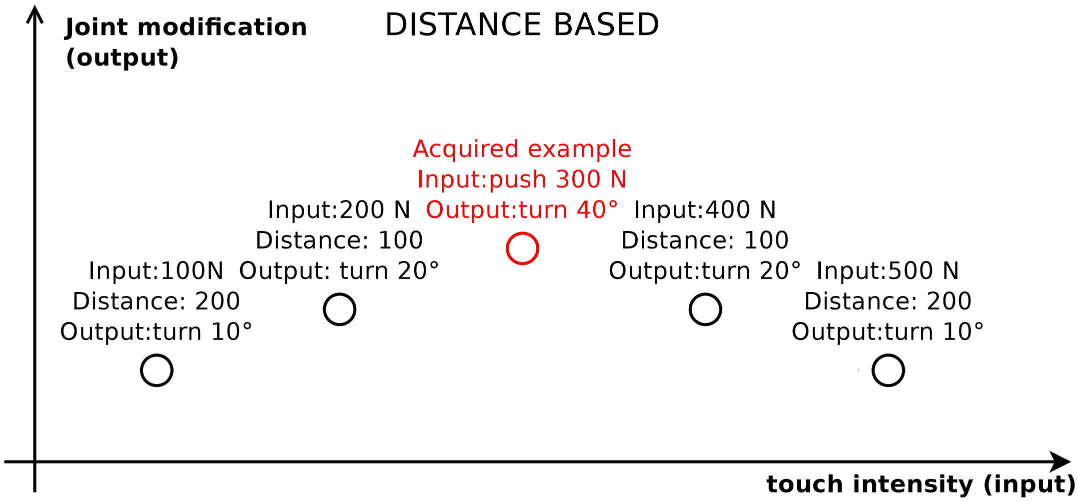
As a simplistic example, suppose to have just one training example, where a sensor was pushed with an intensity of 300 N, and this corresponded to a single motor joint change of 40 degrees. A user might naturally expect that pushing for less intensity will cause a smaller change in that joint, while a more intense press should produce a larger joint angle change. Conversely, in a system with a distance based kernel, any touch on that sensor with a pushing intensity different from 300 N, both higher or lower, would result in a smaller angle change. Figure 3.5 illustrates this problem.

The importance of the last requirement can be readily provided. Imagine the robot to be sat with the legs stretched forward and that a user pushes the foot toes and the heel of one leg simultaneously, teaching the robot to bend the knee and bring the legs close to the body. Suppose that at a later moment the robot is touched only on the foot toe. Many user could desire to associate this touch to a simple foot rotation. The third of the criteria specified prevents the complete leg for being moved, given the absence of the heel pressure in the query sensory input I_*.

Implicitly, the satisfaction of the last criterion also confers "priority" to the touch information over the context. This is to avoid the output to be determined mainly by the context instead of by the pushed parts, as it would happen with kernels that treat tactile and context information in the same way. As a trivial example, suppose a user is focusing on a leg motion and therefore only provides examples involving a leg. With such a training set, if the user pushed on an arm, this will cause the leg to move, in a way depending on the context. However, in such cases, the most reasonable choice would probably be to provide a null joint modification. If a rule like the one specified by the third requirement is not imposed, this kind of behavior is likely to happen since the input space is highly dimensional, so it is very difficult to have, for a particular kind of touch pattern, enough examples with different contexts, such that the touch pattern can have influence for most of the contexts. Figure 3.6 provides some instances of database examples that would be considered or discarded for a certain input.



(a)



(b)

Figure 3.5: The behavior that would be expected is that the more a sensor is pressed, the stronger is the motion modification, as shown in panel (a). However, any distance based metric would have a behavior similar to the one shown in panel (b): the motion modification would be maximum for a pressure intensity equal to the one provided in the database example, and lower in any other case.

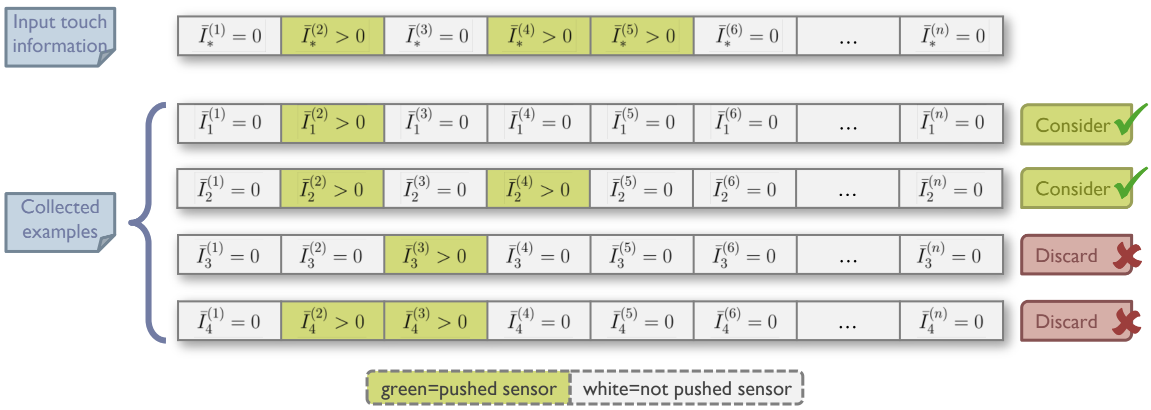


Figure 3.6: Instances of database examples considered and discarded for a particular touch information input.

Let us denote by $\bar{I}_i \in \mathbb{R}^n$ the components of the i-th input corresponding to the tactile sensors. Furthermore, let us indicate by $\bar{I}_i^{(s)} \in \mathbb{R}$ the force applied to the s-th tactile sensor, $1 \leq s \leq n$, normalized in the range $[0,1]$. Denote by $\tilde{I}_i \in \mathbb{R}^{m+o}$ the remaining components of I_i, that constitute the context and that we assume normalized by the respective variance in the dataset of E examples. Let us define by Ψ_i the set of sensors pushed in the i-th example, i.e. $\Psi_i = \{s : \bar{I}_i^{(s)} > 0\}$. Using this notation, after a preliminary empirical evaluation of different functions that satisfy the three criteria previously reported, we decided to set $\omega(I_*, I_i)$ as

$$\omega(I_*, I_i) = \begin{cases} 0 & if\ \exists s : s \in \Psi_i \wedge s \notin \Psi_* \\ \frac{\prod_{s\in\Psi_i} \bar{I}_*^{(s)}/\bar{I}_i^{(s)}}{1+\sqrt{\left\|\tilde{I}_*-\tilde{I}_i\right\|_2^2+\sum_{s:\notin\Psi_i}(\bar{I}_*^{(s)})^2}} & otherwise \end{cases}$$

Essentially, the condition $\omega(I_*, I_i) = 0$ if $\exists s : s \in \Psi_i \wedge s \notin \Psi_*$ is used to satisfy the third criterion. The numerator $\prod_{s\in\Psi_i} \bar{I}_*^{(s)}/\bar{I}_i^{(s)}$ is used to satisfy the second requirement, in fact the more a sensor is pushed the stronger the motion modification is. The denominator $1+\sqrt{\left\|\tilde{I}_* - \tilde{I}_i\right\|_2^2 + \sum_{s:\notin\Psi_i}(\bar{I}_*^{(s)})^2}$ is used to decrease the weight when the difference between the current input and the input of the stored example increases.

As will be better described in the following sections, the algorithm revealed to be applicable in real setups. In particular, Section 3.2 will describe an implementation with simulated touch sensors, and while Section 4.2 will explain the setup of experiments conducted with a real robot.

3.2 Simulated touch sensor implementation

As previously stated, the main idea presented in this chapter is using interpretation of tactile instructions for the creation of robot motions, in a classical "observe and correct" motion development process.

Literature provides solutions for several solutions for realizing a tactile sensor skin. For instance [94] proposes a very modular technique, while [95] presents a very original approach to realize a stretchable tactile distribution sensor that detects pressure by peripheral electrodes. Among humanoid robots with sensing capabilities on the whole body we can cite, for instance, Macra [78] or CB^2 [6]. Section 4.1 will briefly present a small humanoid covered with touch sensors as well.

However, most of the humanoid robots available on the market are usually not equipped with touch sensors for cost reasons. Customizing the robot by covering it with tactile sensors may be expensive and not very easy. In fact, most of the robots on the market are quite small and the wiring becomes complex. If the sensors provide an analog output, (multiplexed) A/D converters must be employed and buses with sufficient bandwidth must be designed.

If the robot is compliant, for instance if it uses pneumatic actuators, then it is possible to read the error between the target position and the actual position to estimate the force applied by the user. The readings are, however, restricted to the joints and estimating the location of the touch may be difficult, as well as identifying simultaneous touch of different parts.

Another alternative solution would be to use a shadow robot. This technique [96], consists in having two identical robots, placed in the same position. The user interacts with one of the two robots, and by comparing the torques with the ones of the second robot, it is possible to distinguish the force applied by the user from other forces like gravity or friction. This approach requires a careful setup, as well as a second robot, which in general may not be available to users.

Finally, although interacting with a physical robot is probably more intuitive, simulating the touch sensors is a very cost effective solution to allow tactile interaction with a robot. Employing simulated tactile sensors presents several advantages. In fact, when using virtual sensors, it is possible to simulate devices not currently available with the current technology in terms of size, bandwidth, signal to noise ratio, etc.

The technique is general, and applicable to any kind of robot. Simulation allows a safer interaction, both for the user and the robot. Interacting with a virtual world also allows viewing the robot's movement in slow play or stopping the motion with no effect on the dynamics, something not feasible in the real world. For instance, slowing down a jump motion to better observe the motion execution is possible in a virtual representation of the world, but not in the real world. Another advantage is that in simulated world additional information can be easily displayed. For example the displacement of the zero moment point [97] can be shown and considered by users developing the motion.

Obviously, the solution presents disadvantages as well. Interaction with 3D representations of the robot may not be intuitive for inexperienced users. Much information measurable by advanced touch sensors, such as the direction of the applied force, may not be obtained by virtual touch sensors which are simulated, for instance, by mouse clicks. Furthermore, while recent technological advances make multi-touch screen easily available, in case a standard mouse is employed, the user cannot touch multiple sensors simultaneously, as would be possible by employing a real robot.

For the advantages of low cost and possibility to easily realize a prototype system, the first experiments on teaching by touching interfaces were conducted using simulated touch sensors. In particular, we decided to simulate the touch sensors while employing the real robot to obtain the motion dynamic. This presents the advantage of removing simulation-reality gap, that would arise from developing the motion completely in simulation.

In other terms, the solution initially adopted is to provide an augmented reality [98] that enhances the existing robot by providing virtual touch sensors. Figure 3.7 shows a schema of the motion development using the proposed approach. In particular, the motion development is carried on by repeating the following steps

1. The motion is played by the real robot, and the position of the robot over time is acquired and recorded with a motion capture system. This allows recording the evolution of the motion in the real environment, preventing any simulator-reality gap. For instance, if a crawling motion is executed, then the real locomotion speed can be observed.

2. The interface allows watching the recorded motion performed by a model of the robot. Being a virtual representation slow-play, pausing and so forth are made possible, easing the user in the development process.

3. The user chooses a time instant at which the posture of the robot should be modified.

4. The user interacts with the virtual robot by touching the virtual robot. In the developed system, a touch screen was used for its intuitiveness, but other devices, like a simple a mouse could be used.

A more detailed description of a prototype system developed using this setup is provided in the next section. Section 3.4 will provide the experimental results of preliminary tests conducted on the system. These show the feasibility of the teaching by touching approach for developing motions. Analysis of the data collected confirms the complexity of the mapping from tactile instructions to motion modifications. In fact, even in the case of motion description by frames used for the experiments, linear models reveal to be inapplicable for explaining the mapping.

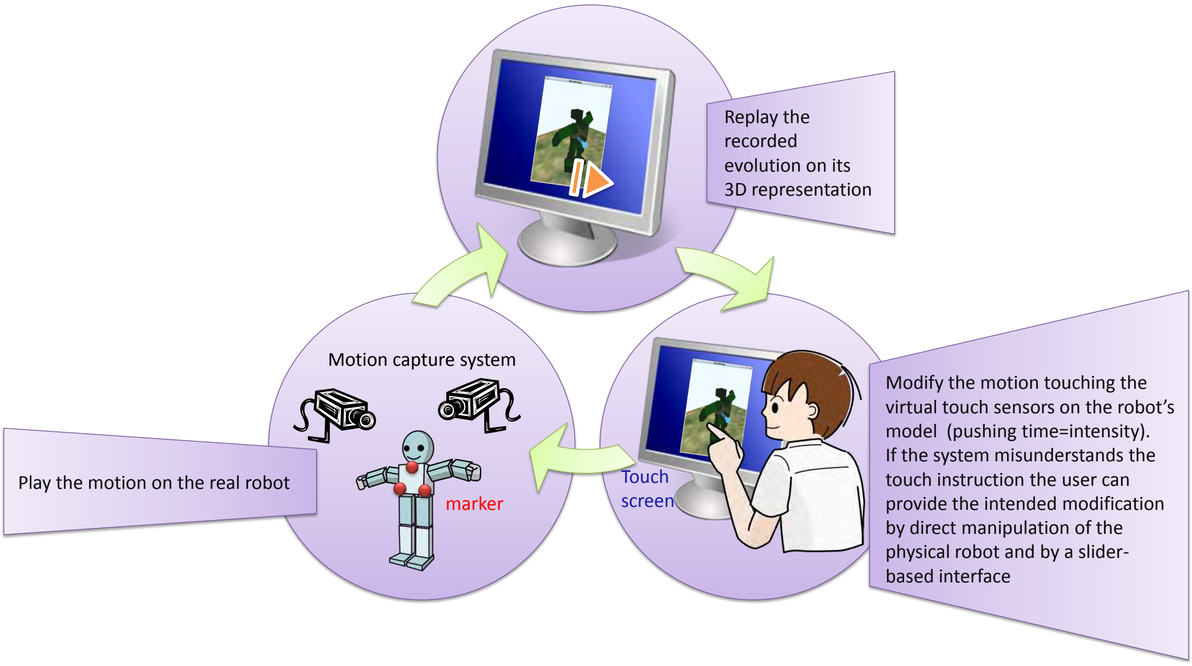


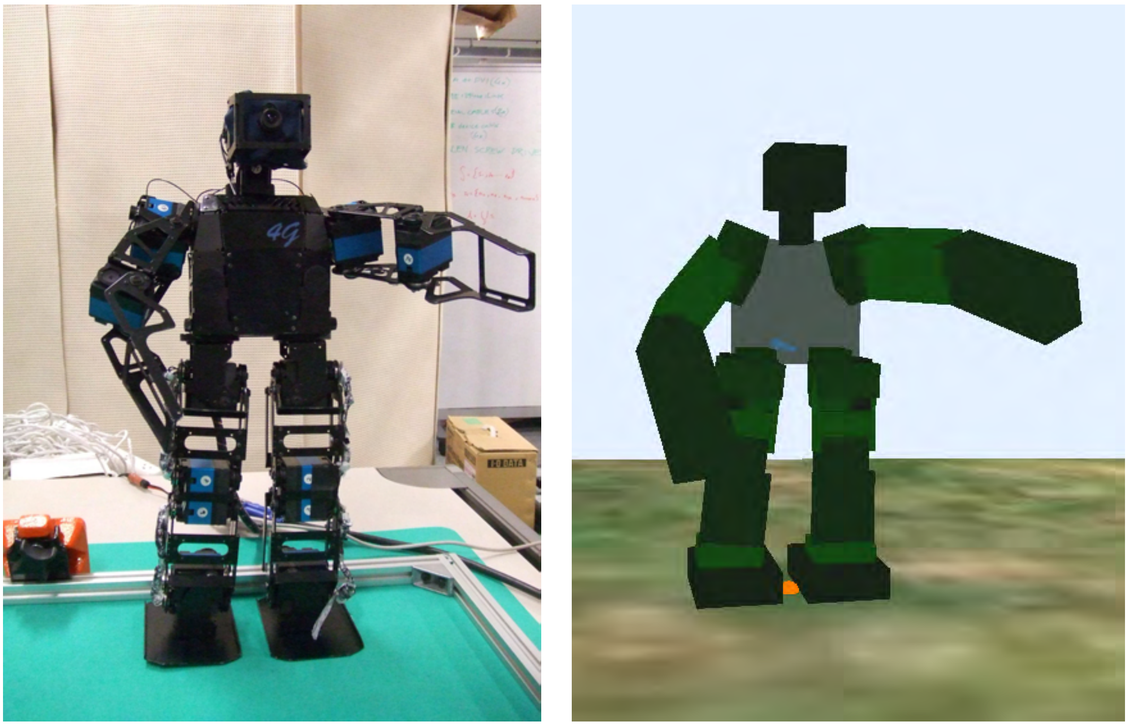
Figure 3.7: Schema of the motion development using virtual touch sensors.

3.3 Experimental setup

As previously stated, the first experiments were conducted using an augmented reality setup, where the real robot is used for playing the motion while a virtual model is used for tactile interaction. In order to replay the motion in the virtual world, the movement information is used to calculate the joint angles assumed by the servomotors during the motion execution. The robot's displacement and orientation assumed during the motion execution are instead captured from the real world with a motion capture system. As stated in the previous section, this permits avoiding any physical simulation and verifying, for instance, the actual velocity achieved by a walking motion.

In detail, the robot physical position was obtained utilizing an Eagle Digital RealTime System from Motion Analysis Corp[3]. More precisely, eight cameras were utilized: five placed just above the ground level, equally spaced on a 1-meter radius circle (that is, forming a pentagon) and pointed to the center of the circle, while the remaining three positioned 1.5 meters above forming an equilateral triangle (inscribable in a circle with a 1 meter radius) and directed toward the point headed by the lower cameras. To acquire the robot position three 2cm diameter markers were placed on its main body. To detect exchanges of the marker labels, a simple check was used: for each received data, the developed system verifies that the marker label assignment operated by the motion capture system is the one that, given the markers positions, minimizes the sum of the errors in the distance between each pair of markers. If the received data does not satisfy

[3]For details see `http://www.motionanalysis.com/html/animation/eagle.html`.



(a) (b)

Figure 3.8: The robot used in the prototype system (a) and its virtual model (b). In the virtual world an orange sphere indicates the projection of the center of gravity on the floor and a blue arrow depicts the velocity vector of the center of mass of the robot.

this property it is simply discarded.

The robot used in the experiment is VStone's VisiON 4G. A picture of the robot is available in Fig. 3.8. The schema of its 22-degrees of freedom is reported in Fig. 3.9.

The prototype interface, developed using Java 3D, models the robot's parts by cuboids, with size and joint positions corresponding to the real hardware. Each face of the parallelepiped simulates a touch sensor. Using devices like a mouse or a classical touch screen, however, it is not possible to push multiple parts simultaneously, and it is not possible to measure the applied force. To circumvent this, the user is allowed to click various parts of the 3D-model and the pushing time of each part is considered as a measure of the pushing intensity. A visual feedback of the intensity is given to the user, consisting of the pushed parts becoming redder and redder, as shown by figure 3.10.

To let the user teach the correct posture in case that the system fails to predict the desired modification, the interface gives several possibilities. In particular, if the response to the touch pattern is not the one desired, then the user can teach the expected modification by using a set of sliders, one for each of the servomotors, as visible in figure 3.11. The interface also allows switching off the desired servomotors, move them on the real robot and read their position. Once the expected resulting posture is set, it is sufficient to press a button (RecPoint) to make the robot remember the association

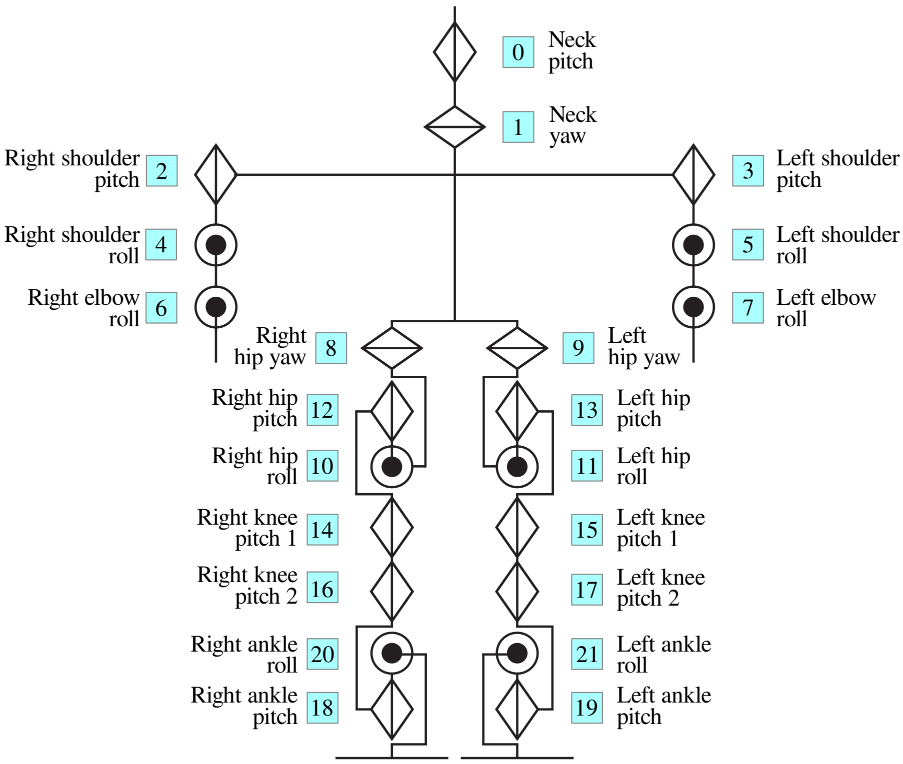


Figure 3.9: Degrees of freedom of VisiON 4G.

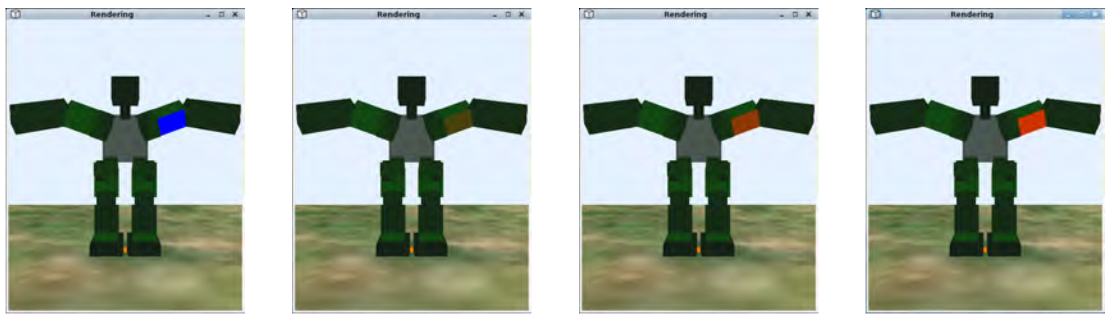


Figure 3.10: Visual feedback provided by the interface while pushing a robot's part.

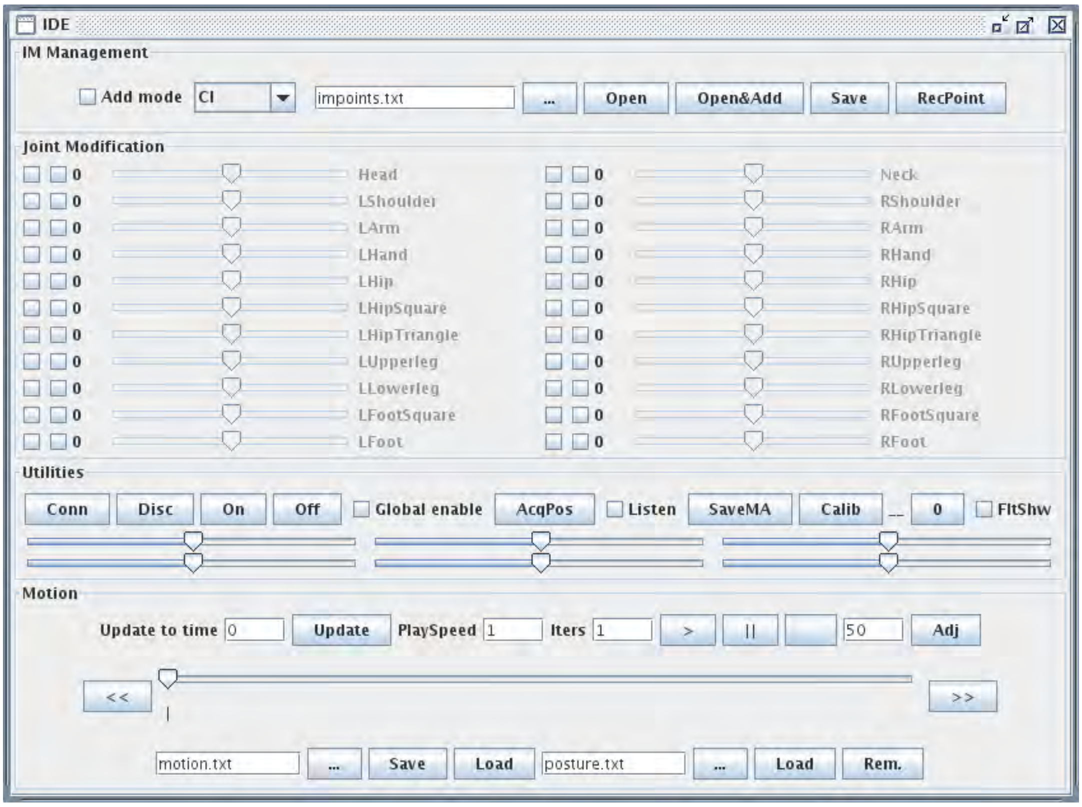


Figure 3.11: Screenshot of the developed prototype interface.

between the tactile pattern provided and the modification shown.

As a preliminary proof of concept, a jumping and walking motion were developed. Results, presented in the next section, show the feasibility of the teaching by touching approach for developing motions. In particular, it will be shown that the approach resulted in a faster motion development time compared to a classic, slider based interface. Furthermore, the input-output examples used to train the touch interpreter will be analyzed, providing a preliminary analysis of the complexity of the meaning of tactile instructions. Section 3.5 will then present experiments showing that the mapping is not only nonlinear and complex, but also strongly user dependent.

3.4 Results

This section reports the first preliminary experiment consisted in the creation of a jump motion. Actually, to make jumping feasible, a rubber band pulling the robot from the top was used, since the torque of the leg servomotor was not sufficient. Figure 3.12 provides an image sequence of the realized movement.

The jumping motion was developed by the same user first with the proposed teaching by touching interface, and then with a classic, slider based editor. In the former case the motion took 17 minutes, in the latter over 40 minutes. Although not statistically

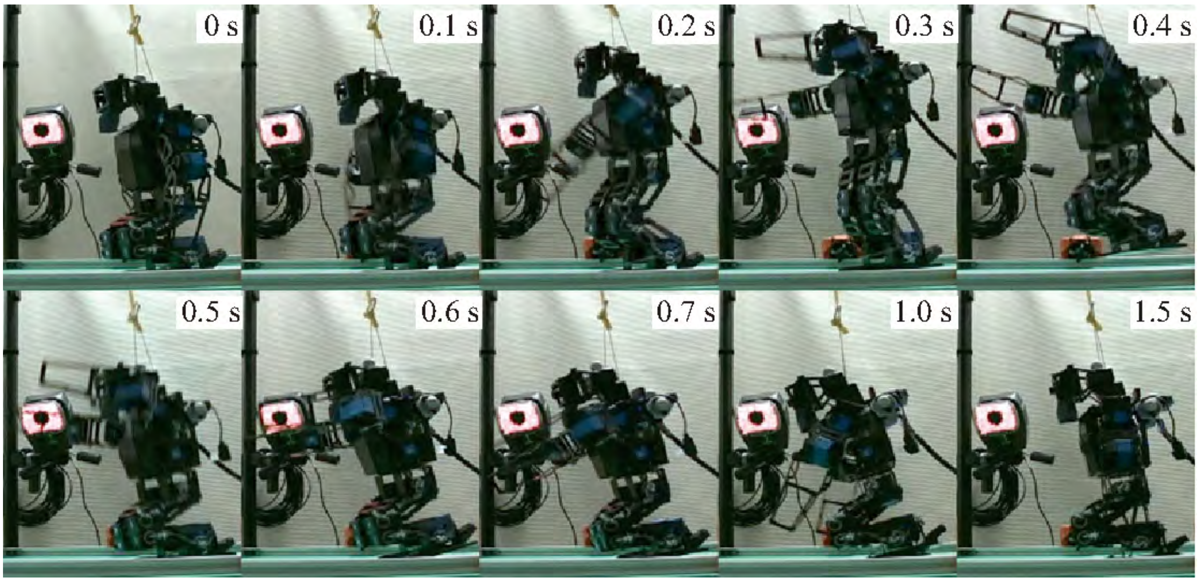


Figure 3.12: Image sequence of the jump motion.

significant, this result supports the possibility for the approach to be a good solution for motion development.

As already stated, one of the merits of using a supervised learning framework is that the input-output examples provided by the user can be analyzed, to get insights on the way humans use touch to develop motions. The next step was therefore developing another motion, walking, and studying the databases of data collected for the two motions. The collected datasets, consisting of examples of the mapping from touch pattern and context to joint modification, were studied first using linear regression, to verify how well a linear model could explain the collected data, and then with the C4.5 decision tree construction algorithm, to determine which are the features of the context that mostly influence the meaning of touching.

3.4.1 Mapping nonlinearity

As a first step, supposing that the collected data could be roughly approximated by a linear model, a simple linear regression was applied to each of the datasets, denoted in the following as $JUMP$ for the jumping motion and $WALK$ for the walking motion. Their union was also computed, and considered as a third database $COMBINED = JUMP \cup WALK$. Particularly, given the data set D, let us denote by X_D the matrix having as the i-th row the i-th example input I_i (touch information and context) followed by a 1 (to include the possibility of a constant term in the mapping) and similarly let us indicate by Y_D a matrix having in the i-th row the i-th example output M_i (the joint modification provided by the user), as done in Section 3.1.

Let us then define LD as the matrix such that

$$Y_D \approx X_D L_D$$

i.e.

$$Y_D = X_D L_D + \epsilon$$

where ϵ is a matrix expressing the error due to the linear approximation.

The matrices L_D (with D equal to $JUMP$, $WALK$ and $COMBINED$) were calculated using ridge regression, that is

$$L_D = (X_D^T X_D + \alpha I)^{-1} X_D^T Y_D \tag{3.4}$$

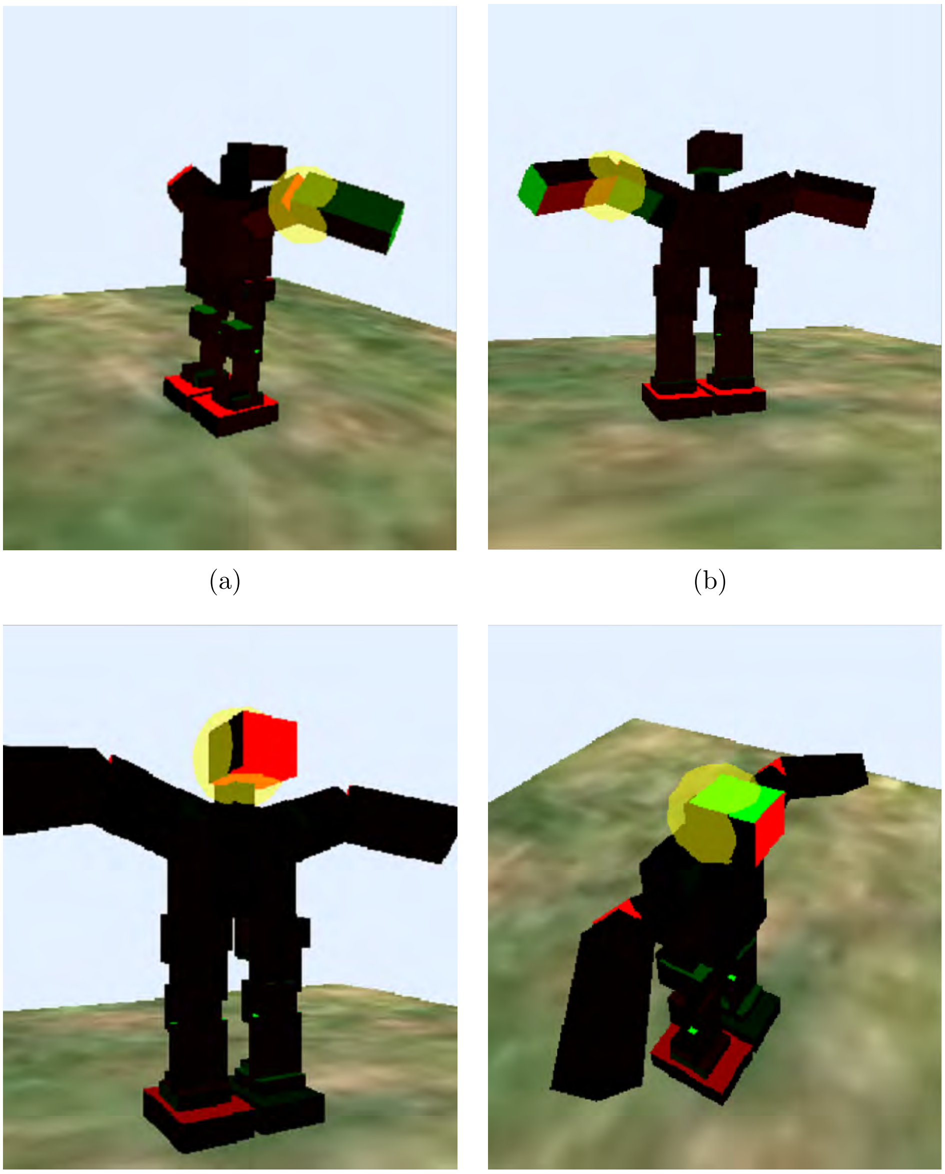
The term α is a small constant which can be interpreted as an estimate of the standard deviation of a Gaussian noise affecting the data [99], and which allows to prevent the resultant matrices L_D to have coefficients with huge absolute values. All the results reported here were obtained by setting it to 0.1.

The absolute values of the entries in L_D indicate the importance of each input features in determining the modification for each motor, while the sign permits understanding which features produce a similar effect and which produce an opposite effect. Figure 3.13 gives a visual representation of some of the entries of L_D: for the outputs corresponding to two of the robot's joints each sensor is colored depending on the value of the entry in L_{JUMP} that determines how much such sensor value contributes to the joint position variation.

Figure 3.13 shows that the importance of the input features in determining a joint modification calculated by linear regression, at least for the sensors, has a strong relationship with the common sense. In fact, for instance, the sensors identified as important for determining the head orientation are mainly the ones on the head. Nonetheless, the linear model seems to be not articulated enough to capture the structure of the tactile instructions and often overfit them. Actually, to get an idea of which features of the context assume higher importance, for each column of the matrices L_{JUMP}, L_{WALK} and $L_{COMBINED}$, the two features with the highest values were identified, but there was no feature that revealed to be the most or the second most important feature for a certain joint variation for all of the tree data sets. A comparison with the kernel regression algorithm showed that the latter performs a lot better on test data not used for the training. More specifically:

1. The matrices L_{JUMP}, L_{WALK} and $L_{COMBINED}$ were calculated using Eq. 3.4 with D equals to $JUMP$, $WALK$ and $COMBINED$, respectively, that is, using one of the three "training sets".

2. Each of the matrices L_D that was used to predict the output for each of the datasets



(a) (b) (c) (d)

Figure 3.13: Graphical visualization of some of the entries in L_{JUMP}. Panels (a) and (b) visualize the importance of each of the sensors in determining the variation of the head pitch angle. Panels (c) and (d) display the importance of each sensors and the variation of the elbow joint angle. The sensors are colored red or green depending on the sign of the relative coefficient, while the absolute value of the coefficient is represented by the intensity of color. A yellow sphere is used to highlight the position of the joint whose modification depends on the coefficients used to color the sensors.

Training Dataset	Test Dataset	Linear Reg. error	Kernel Reg. error
JUMP	JUMP	0.2846	0.1985
JUMP	WALK	13.717	0.5938
JUMP	COMBINED	3.7272	0.3006
WALK	JUMP	2.0567	1.0198
WALK	WALK	0.0314	0.0588
WALK	COMBINED	1.5462	0.7778
COMBINED	JUMP	0.3626	0.2223
COMBINED	WALK	0.3219	0.1425
COMBINED	COMBINED	0.3522	0.2028

Table 3.1: Average errors in the output prediction by linear regression and by kernel regression in rads.

(used as "test sets"), that is $\hat{Y}_{C,D} = X_C L_D$ was calculated for all the combinations of C and D (C and D can be $JUMP$, $WALK$ or $COMBINED$).

3. The differences $E_{C,D} = \hat{Y}_{C,D} - Y_C$ were calculated. Clearly, the i-th row of Y_C is the output, provided by the user, that should be given when the input is the i-th row of X_C. Similarly, the i-th row of $\hat{Y}_{C,D}$ is the output of the mapping predicted by the linear mode. Therefore, the i-th row of $E_{C,D}$ provides the error in the prediction of the i-th example of the test dataset C when the linear model is constructed based on the dataset D;

4. For each training dataset D and each test dataset C, the average error magnitude over all of the examples of the test dataset was determined. This was computed as the mean of the square roots of the sums of the squares of the values in each row of $E_{C,D}$.

Similarly, the kernel regression algorithm with the described weighting schema was tested on the three data sets. In each test, the algorithm was given one of the sets as training set, i.e. as the set of examples used to calculate the output, and using that knowledge the prediction errors for the three databases were computed. Table 3.1 provides a comparison of errors made by the two algorithms. As can be seen especially observing the second row, kernel regression strongly outperformed the linear model in the test.

3.4.2 Context feature importance

In order to try to determine if the mapping presents "cases" and "subcases", that is, context conditions under which the mapping from I_* to M_* changes, decision trees from each data set ($JUMP$, $WALK$ and $COMBINED$) were constructed. Usually decision trees are used for classification problems; in a decision tree each internal node corresponds to an attribute, and an arc to a child represents a possible value (or an interval of values, if the attribute is continuous) of such variable. A leaf represents the predicted class of an object, given the values of the attributes indicated by the path from the root.

In the experiments, decision trees were constructed using Quinlan's C 4.5 algorithm [100]. In detail, each collected example was regarded as an object with $|I| = n + m + o + v$ continuous attributes, where $|.|$ is used to indicate the cardinality of a vector, and n, m, o and v represent, using the notation of Section 3.1, the number of touch sensors, motors, components of the robot orientation and velocity, respectively.

In the construction of the decision tree the class to which each object belongs must be provided. Classes are discrete, therefore we operated a discretization of the outputs. In particular, we mapped each sensor output M_i to the class $C_i = \sum_{j=1}^{m} 3^j * q(M_i^{(j)}) \in \mathbb{N}$, where the function $q : \mathbb{R} \to \mathbb{N}$ is used to discretize each component of the output vector $M_i^{(j)}$. In particular, the function f returns 0 if the rotation magnitude is less than 5 degrees, 1 if the motor should rotate more than 5 degrees in the clockwise direction and 2 if the motor should rotate more than 5 degrees in the counterclockwise direction.

Clearly, the possible number of classes is very big, 3^m, with $m = 22$ in our case. However, few of them are significant and will really appear in the collected data, since many combinations have little meaning. For instance, it is improbable to collect an example where the user provides an instruction that corresponds to moving the head and one of the feet. In fact, the 238 examples collected during the development of the two motions belonged to 97 classes.

Once the tree is built, observing the level in the tree at which each of the context features appears it is possible to get insights on which features of the context are important in changing the meaning of the touching, because the most important features will appear at higher levels, near the root. Table 3.2 reports the levels in which each element of the context appears constructing a decision tree using each of the $JUMP$, $WALK$ and $COMBINED$ datasets. It should be noticed that the trees were constructed with all of the input information as training data, that is providing the touch pattern too, but just the context elements are reported in the table. Empty entries in the table means that for the training dataset indicated by the column header the feature reported as row header did not appear in the derived decision tree. The first 22 features correspond to the position of the robot's joints; the location of the various joints of the robot is shown in Fig. 3.9.

The most interesting thing is that for all of the constructed trees (with different training datasets) the position of the joints near the main body, that is the joints which determine the global position of the limbs, is used as a discriminating attribute in high levels of the tree, near the root. This fact, emerged from the data, is reasonable since the meaning of the touch probably strongly depends on the overall position of the limbs. It is also possible to note that the orientation of the robot and its center of gravity velocity vector do not appear as important features. The current hypothesis to explain

Feature	JUMP	WALK	COMBINED
Pos. joint 0	2		2
Pos. joint 1	17,21,22	9	19,23
Pos. joint 2	3	3	3
Pos. joint 3		5,11	
Pos. joint 4			
Pos. joint 5		2	
Pos. joint 6	20		22
Pos. joint 7	5,8,19		5,10
Pos. joint 8	1	3	1
Pos. joint 9			
Pos. joint 10		10	
Pos. joint 11	0	1	0
Pos. joint 12			
Pos. joint 13		3	
Pos. joint 14		2,8,10	
Pos. joint 15	5	3,4	7
Pos. joint 16			
Pos. joint 17			
Pos. joint 18			
Pos. joint 19			
Pos. joint 20			
Pos. joint 21			
pitch		0,2	
roll			
velocity x			
velocity y			
velocity z			

Table 3.2: Levels at which the various elements of the context appears in the decision trees constructed by the C4.5 algorithm using the three training datasets.

Training set	JUMP			WALK			COMBINED		
Test set	J	W	C	J	W	C	J	W	C
Linear regr.	0.25	1.00	0.44	0.99	0.00	0.74	0.43	0.82	0.53
C 4.5	0.38	0.97	0.53	0.96	0.39	0.82	0.39	0.40	0.40
Kernel regr.	0.10	0.69	0.25	0.88	0.00	0.66	0.11	0.18	0.13

Table 3.3: Error rates of the tree algorithms in predicting the output class expressed as fractions of unit (for example 0.1 means 10%). *J*, *W* and *C* stand, respectively, for *JUMP*, *WALK* and *COMBINED.*

this is that the developed motions are quite static, and probably during the development of more dynamical motions, the meaning will depend more on these features. Future works will investigate this fact more deeply, and will try to confirm the importance of the orientation of the joints near the body emerged in these tests.

One simple test was then carried on to see whether the kernel regression algorithm with the introduced weighting can outperform C4.5 in predicting the "class" associated to a certain touching pattern and context. This was done by translating the output of the algorithm into a class, using the function q previously described. As done in the comparison with the linear regression, the algorithms were tested using the datasets $WALK$, $JUMP$ and $COMBINED$ as training sets and test sets in all possible combinations. Table 3.3 reports the error rate for each of the algorithms in determining the belonging class. The output of linear regression was translated into classes too so a comparison between the three algorithms is possible. Even if C4.5 is specifically designed for classification problems, the kernel regression algorithm usually performs better, probably because its weighting schema was designed specifically for the touch instructions. As could be expected, kernel regression outperformed linear regression in this case too.

Comparisons with other machine learning algorithms that provide a continuous outputs, like regression trees [88] or neural networks [87], would be very interesting, and will be conducted as a future work.

In conclusion, results show that the mapping between tactile instructions and motion modifications cannot be realized through linear models, as could be expected. Furthermore, a simple application of C4.5 showed the potential of performing data-mining on the data collected by the system during user interactions. In detail, we identified that the position of the joints near the torso have great influence of the context. This information could be used when designing model based instruction interpreters.

The next section will present a preliminary analysis of user dependence conducted with six subjects. Also in this case, the application of simple data analysis techniques on the data shows the existence of clusters of users, that appear to be induced by the different levels of abstraction used in teaching.

Figure 3.14: Simulated VisiON 4G.

3.5 Analysis of user dependence

The fact that the meaning of tactile instructions depends on the user was highlighted in Section 2.2. To verify this hypothesis, a preliminary experiment with six subjects was conducted. Furthermore, the capability of the supervised learning algorithm was investigated. More precisely, the assumption that the need of teaching the meaning of tactile instructions decreases over time was verified.

In the experiment, subjects were requested to develop a walking and a kicking motion, two fundamental capabilities required by a humanoid soccer robot. The six subjects are all Italian male computer science students, and their age is in the range of 23-27 (mean 24.5, standard deviation 1.87). To assure the same conditions for all the users we asked them to develop motions using a simulated robot, instead of using the real robot as done in the previous experiments. We simulated VisiON 4G in a custom made simulator, shown in Fig. 3.14 and briefly described in appendix A.

We asked the subjects to freely touch the robot to give commands and to teach their meaning using the sliders when the robot did not understand them. Given the test subjects' similar background, all male computer science students, we expected them to have provided very similar touch instructions. However, the touching manner greatly varied, even in this very restricted setup.

As a first step of the analysis we studied the relationship between sensors and joints. In detail, for each of the users we identified the joints moved in response to the pressure of a sensor (possibly in combination with other sensors). This allows us to identify if there are direct mappings between sensors and joints that are intuitive for most of the users. In Fig. 3.15 (as well as in all the subsequent figures) each row represents a sensor and each column represents a joint. The rows and columns are divided into groups that represent the part of the body where the sensor or joint is located.

The figures report just the sensors that were used by at least one of the subjects. In each figure the headers of the sensors and joints, i.e. the row and column header, respectively, are colored lighter if they were employed by the user.

In Figures 3.15, 3.16, 3.17, 3.18, 3.19 and 3.20, each intersection between a row and a column is colored if in at least one touch the sensor corresponding to the row was pressed and then it was taught to move the joint corresponding to the column (and possibly other joints). We can observe that even if we fixed the task and employed subjects with similar background, the mappings between sensors and joints are very different.

To better grasp the similarities and dissimilarities of these relationships between the mappings, we plotted Fig. 3.21. In this table, the color of each entry indicates for how many users the sensor corresponding to the row was pressed to move the joint identified by the column. Figure 3.22 reports how many times each configuration was used by all of the users in total.

We notice that the combinations that appear more frequently are the ones which map sensors to joints of the same limb, that is, the sensor-joint combinations for which the row group and the column group coincide. It is in fact very reasonable that most of the users touch the limb they want to move. We can see a correlation between the body sensors and the joints of the legs. The reason is that the users pushed the robot's body to bend the trunk, and this is done by moving the hip joints. At first glance, the relatively high correlation between the sensors of one arm and the joints of the other is surprising. A deeper analysis of the data showed that this is because most of the users pushed both hands at the beginning to close the arms, which are opened in the initial position (see Fig. 3.10 or Fig. 3.13).

It should be noted that if, after providing these kinds of examples, the user pushes just one arm the other one is not moved, due to the third requirement given imposed to the regression kernel and discussed in Section 3.1.

Observing the lower part of Fig. 3.21 we notice that just one subject touched sensors on the left leg and moved the right one. By Fig. 3.19 we can see that this was done by subject E. Unlike other people, E's table shows a correlation between the body and the arms. A direct analysis of the touch data showed that user E pushed the robot's belly and made it close the arms. He then touched the left knee and taught the robot to move the left leg forward (and not backward, as one could imagine). Since we never observed such kind of touch instruction we contacted E again and learned that this subject gave a very high-level meaning to the touch instructions. For instance with a single touch on the belly he wanted to say "go to the normal posture", where "normal" is considered as

	neck	neckYaw	RshoulderPitch	RshoulderRoll	RelbowRoll	LshoulderPitch	LshoulderRoll	LelbowRoll	RhipYaw	RhipPitch	RhipRoll	RkneePitch1	RkneePitch2	RankleRoll	RanklePitch	LhipYaw	LhipPitch	LhipRoll	LkneePitch1	LkneePitch2	LankleRoll	LanklePitch
bodyFront	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	1	0	0	1	0
bodyLeft	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	0	0	0	0	1
bodyBack	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	1	0	0	1	0
bodyRight	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
bodyUp	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	1	0	0	1	0
neckBack	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	1	0	0	1	0
headFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headLeft	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headRight	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandFront	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandLeft	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandBack	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandRight	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LshoulderLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandFront	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandLeft	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandBack	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandRight	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RlowerLegFront	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	0	0	0	0	0
RlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
RupperLegFront	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	1	0	0
RupperLegBack	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	0	0
RhipOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhipOuterRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeBack	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
RkneeUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RfootFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RfootUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleInnerBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleOuterUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LlowerLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LupperLegFront	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0
LupperLegBack	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0
LhipOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhipOuterLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LkneeFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LkneeBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LfootFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LfootRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LankleOuterFront	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.15: Relationship between sensors (rows) and joints (columns) for subject A. The color indicates the part of the robot where the sensor or joint is located: green for the body, yellow for the head, red for the right arm, blue for the left arm, magenta for the right leg and cyan for the left leg.

	neck	neckYaw	RshoulderPitch	RshoulderRoll	RelbowRoll	LshoulderPitch	LshoulderRoll	LelbowRoll	RhipYaw	RhipPitch	RhipRoll	RkneePitch1	RkneePitch2	RankleRoll	RanklePitch	LhipYaw	LhipPitch	LhipRoll	LkneePitch1	LkneePitch2	LankleRoll	LanklePitch
bodyFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
neckBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandFront	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LshoulderLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandFront	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RlowerLegFront	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	0	1
RlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RupperLegFront	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0
RupperLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhipOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhipOuterRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RfootFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RfootUp	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
RankleInnerBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleOuterUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LlowerLegFront	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0	0	1
LlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LupperLegFront	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	1	0	1	1
LupperLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhipOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhipOuterLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LkneeFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LkneeBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LfootFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LfootRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LankleOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.16: Relationship between sensors (rows) and joints (columns) for subject B. The notation is the same as Fig. 3.15.

	neck	neckYaw	RshoulderPitch	RshoulderRoll	RelbowRoll	LshoulderPitch	LshoulderRoll	LelbowRoll	RhipYaw	RhipPitch	RhipRoll	RkneePitch1	RkneePitch2	RankleRoll	RanklePitch	LhipYaw	LhipPitch	LhipRoll	LkneePitch1	LkneePitch2	LankleRoll	LanklePitch
bodyFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyBack	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	1	1	1	0	1	1
bodyRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
neckBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderFront	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
RshoulderRight	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
RhandFront	0	0	1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
RhandLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandBack	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandRight	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LshoulderLeft	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	1
LhandFront	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandLeft	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandBack	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
LhandRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RlowerLegFront	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
RlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RupperLegFront	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
RupperLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhipOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhipOuterRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeBack	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
RkneeUp	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
RfootFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RfootUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleInnerBack	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
RankleOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleOuterUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LlowerLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LupperLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LupperLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhipOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhipOuterLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LkneeFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LkneeBack	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0
LfootFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LfootRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LankleOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.17: Relationship between sensors (rows) and joints (columns) for subject C. The notation is the same as Fig. 3.15.

	neck	neckYaw	RshoulderPitch	RshoulderRoll	RelbowRoll	LshoulderPitch	LshoulderRoll	LelbowRoll	RhipYaw	RhipPitch	RhipRoll	RkneePitch1	RkneePitch2	RankleRoll	RanklePitch	LhipYaw	LhipPitch	LhipRoll	LkneePitch1	LkneePitch2	LankleRoll	LanklePitch
bodyFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
neckBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandFront	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandRight	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LshoulderLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandFront	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandLeft	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RlowerLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RupperLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RupperLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhipOuterFront	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
RhipOuterRight	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
RkneeFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RfootFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RfootUp	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	0	1	0	0
RankleInnerBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleOuterUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LlowerLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LupperLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LupperLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhipOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0
LhipOuterLeft	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
LkneeFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LkneeBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LfootFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LfootRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LankleOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.18: Relationship between sensors (rows) and joints (columns) for subject D. The notation is the same as Fig. 3.15.

	neck	neckYaw	RshoulderPitch	RshoulderRoll	RelbowRoll	LshoulderPitch	LshoulderRoll	LelbowRoll	RhipYaw	RhipPitch	RhipRoll	RkneePitch1	RkneePitch2	RankleRoll	RanklePitch	LhipYaw	LhipPitch	LhipRoll	LkneePitch1	LkneePitch2	LankleRoll	LanklePitch
bodyFront	0	0	1	1	0	1	1	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0
bodyLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
neckBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headFront	0	0	1	1	0	1	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0
headLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LshoulderLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandRight	0	0	1	1	0	1	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0
RlowerLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RupperLegFront	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	1	1	0	0	1	0
RupperLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhipOuterFront	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	1	1	0	0	1	0
RhipOuterRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeFront	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	1	1	0	0	1	1
RkneeBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RfootFront	0	0	1	1	0	1	1	0	0	1	1	1	1	1	1	0	1	0	0	0	0	1
RfootUp	0	0	1	1	0	1	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0
RankleInnerBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleOuterFront	0	0	1	1	0	1	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	0
RankleOuterUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LlowerLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LupperLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LupperLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhipOuterFront	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	1	1	1	0
LhipOuterLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LkneeFront	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	1	1
LkneeBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LfootFront	0	0	1	1	0	1	1	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0
LfootRight	0	0	1	1	0	1	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	1
LankleOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.19: Relationship between sensors (rows) and joints (columns) for subject E. The notation is the same as Fig. 3.15.

	neck	neckYaw	RshoulderPitch	RshoulderRoll	RelbowRoll	LshoulderPitch	LshoulderRoll	LelbowRoll	RhipYaw	RhipPitch	RhipRoll	RkneePitch1	RkneePitch2	RankleRoll	RanklePitch	LhipYaw	LhipPitch	LhipRoll	LkneePitch1	LkneePitch2	LankleRoll	LanklePitch
bodyFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bodyLeft	0	0	1	0	0	1	0	0	0	1	1	0	0	1	1	1	1	0	0	0	0	1
bodyBack	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	1	0	1	1	1
bodyRight	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
bodyUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
neckBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headFront	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmFront	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmBack	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmRight	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandFront	0	0	1	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
RhandLeft	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandBack	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandRight	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmFront	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
LarmLeft	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmBack	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
LarmRight	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
LshoulderLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandFront	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandLeft	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RlowerLegFront	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
RlowerLegBack	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	1	0
RupperLegFront	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
RupperLegBack	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
RhipOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhipOuterRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeFront	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
RkneeBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RkneeUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RfootFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RfootUp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleInnerBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RankleOuterUp	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
LlowerLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0
LupperLegFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LupperLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhipOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhipOuterLeft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LkneeFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LkneeBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LfootFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LfootRight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LankleOuterFront	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.20: Relationship between sensors (rows) and joints (columns) for subject F. The notation is the same as Fig. 3.15.

	neck	neckYaw	RshoulderPitch	RshoulderRoll	RelbowRoll	LshoulderPitch	LshoulderRoll	LelbowRoll	RhipYaw	RhipPitch	RhipRoll	RkneePitch1	RkneePitch2	RankleRoll	RanklePitch	LhipYaw	LhipPitch	LhipRoll	LkneePitch1	LkneePitch2	LankleRoll	LanklePitch
bodyFront	0	0	1	1	0	1	1	0	0	2	1	2	1	2	2	0	1	1	0	0	1	0
bodyLeft	0	0	1	0	0	1	0	0	0	2	1	1	1	2	2	1	2	0	0	0	0	2
bodyBack	0	0	0	0	0	0	0	0	0	3	2	3	2	3	3	0	3	3	1	1	3	2
bodyRight	0	0	0	0	0	0	0	0	0	2	0	0	0	0	2	0	2	0	0	0	0	2
bodyUp	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	1	0	0	1	0
neckBack	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	1	0	0	1	0
headFront	0	0	1	1	0	2	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0
headLeft	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headRight	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmFront	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmBack	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmRight	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderFront	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
RshoulderRight	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
RhandFront	0	0	5	3	2	3	2	1	0	0	2	1	1	2	0	0	0	0	0	0	0	0
RhandLeft	0	0	1	2	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandBack	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandRight	0	0	3	4	2	3	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmFront	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
LarmLeft	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmBack	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
LarmRight	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
LshoulderLeft	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	1
LhandFront	0	0	3	2	1	5	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandLeft	0	0	2	3	2	2	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandBack	0	0	0	0	0	2	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
LhandRight	0	0	1	1	0	1	1	1	0	1	1	1	0	1	1	0	0	0	0	0	0	0
RlowerLegFront	0	0	1	0	0	1	0	0	0	2	3	3	2	4	2	0	2	0	0	0	0	1
RlowerLegBack	0	0	0	0	0	0	0	0	0	0	1	1	2	2	0	0	0	1	0	0	1	0
RupperLegFront	0	0	1	0	0	1	0	0	0	2	4	3	3	5	2	1	2	2	1	1	1	0
RupperLegBack	0	0	0	0	0	1	0	0	0	0	2	1	1	2	0	1	1	1	1	1	0	0
RhipOuterFront	0	0	0	0	0	0	0	0	0	1	2	0	2	2	1	0	1	1	0	0	1	0
RhipOuterRight	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
RkneeFront	0	0	1	0	0	1	0	0	0	1	2	0	1	2	1	0	1	1	0	0	1	1
RkneeBack	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0
RkneeUp	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
RfootFront	0	0	1	1	0	1	1	0	0	1	1	1	1	1	1	0	1	0	0	0	0	1
RfootUp	0	0	1	1	0	1	0	0	0	2	3	1	0	3	2	0	1	1	0	1	0	0
RankleInnerBack	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
RankleOuterFront	0	0	1	1	0	1	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	0
RankleOuterUp	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
LlowerLegFront	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0	0	1
LlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0
LupperLegFront	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	2	2	1	1	1
LupperLegBack	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0
LhipOuterFront	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	1	2	1	2	2	0
LhipOuterLeft	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
LkneeFront	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	1	1
LkneeBack	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0
LfootFront	0	0	1	1	0	1	1	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0
LfootRight	0	0	1	1	0	1	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	1
LankleOuterFront	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.21: Number of subjects that used each sensor-joint couple. Each row corresponds to a sensor, each column to a joint. The value in the entry indicates the number of users that associated the pressure of the corresponding sensor (and possibly of others) to the rotation of the corresponding joint (and possibly of others). The color of the row and column headers indicates the part of the robot where the sensor or the joint is located.

	neckPitch	neckYaw	RshoulderPitch	RshoulderRoll	RelbowRoll	LshoulderPitch	LshoulderRoll	LelbowRoll	RhipYaw	RhipPitch	RhipRoll	RkneePitch1	RkneePitch2	RankleRoll	RanklePitch	LhipYaw	LhipPitch	LhipRoll	LkneePitch1	LkneePitch2	LankleRoll	LanklePitch
bodyFront	0	0	2	2	0	2	1	0	0	2	1	2	1	2	2	0	1	1	0	0	1	0
bodyLeft	0	0	1	0	0	1	0	0	0	3	1	1	1	2	3	1	3	0	0	0	0	2
bodyBack	0	0	0	0	0	0	0	0	0	5	4	3	2	5	5	0	5	5	1	2	5	2
bodyRight	0	0	0	0	0	0	0	0	0	2	0	0	0	0	2	0	2	0	0	0	0	2
bodyUp	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	1	0	0	1	0
neckBack	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	1	0	0	1	0
headFront	0	0	1	1	0	2	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0
headLeft	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headRight	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmFront	0	0	1	2	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmBack	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RarmRight	0	0	2	4	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RshoulderFront	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
RshoulderRight	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
RhandFront	0	0	9	6	3	5	3	1	0	0	2	1	1	2	0	0	0	0	0	0	0	0
RhandLeft	0	0	1	2	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandBack	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RhandRight	0	0	3	6	3	3	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmFront	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
LarmLeft	0	0	2	3	0	2	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LarmBack	0	0	2	0	0	2	0	0	0	0	2	0	0	2	0	1	0	0	0	0	0	0
LarmRight	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
LshoulderLeft	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	1
LhandFront	0	0	5	4	1	9	5	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandLeft	0	0	2	3	2	3	6	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LhandBack	0	0	0	0	0	2	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
LhandRight	0	0	1	1	0	1	2	2	0	1	1	1	0	1	1	0	0	0	0	0	0	0
RlowerLegFront	0	0	1	0	0	1	0	0	0	3	5	4	3	6	3	0	3	0	0	0	0	2
RlowerLegBack	0	0	0	0	0	0	0	0	0	0	1	2	3	2	0	0	0	1	0	0	1	0
RupperLegFront	0	0	2	0	0	2	0	0	0	2	5	4	4	7	2	1	2	2	1	1	1	0
RupperLegBack	0	0	0	0	0	1	0	0	0	0	2	3	3	3	0	1	1	1	1	1	0	0
RhipOuterFront	0	0	0	0	0	0	0	0	0	1	2	0	2	2	1	0	1	1	0	0	1	0
RhipOuterRight	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
RkneeFront	0	0	1	0	0	1	0	0	0	1	2	0	1	2	1	0	1	1	0	0	1	1
RkneeBack	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0
RkneeUp	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
RfootFront	0	0	3	3	0	3	2	0	0	3	3	3	2	1	3	0	2	0	0	0	0	1
RfootUp	0	0	1	1	0	1	0	0	0	2	3	1	0	3	2	0	1	1	0	1	0	0
RankleInnerBack	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
RankleOuterFront	0	0	1	1	0	1	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	0
RankleOuterUp	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
LlowerLegFront	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	3	0	0	1
LlowerLegBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0
LupperLegFront	0	0	0	0	0	0	0	0	0	2	0	1	1	1	2	0	2	3	2	1	2	1
LupperLegBack	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0
LhipOuterFront	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	1	2	1	2	2	0
LhipOuterLeft	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
LkneeFront	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	1	1
LkneeBack	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0
LfootFront	0	0	2	2	0	2	1	0	0	2	2	2	1	1	2	0	1	0	0	0	0	0
LfootRight	0	0	1	1	0	1	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	1
LankleOuterFront	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.22: Number of times that each sensor-joint couple appeared in a tactile instruction. Each row corresponds to a sensor, each column to a joint. The value in the entry indicates the number of instructions in the databases for which the pressure of the corresponding sensor (and possibly of others) was associated to the rotation of the corresponding joint (and possibly of others). The color of the row and column headers indicates the part of the robot where the sensor or the joint is located.

placing the arms along the body sides. He then tried to teach with a single touch on a knee to make one step with that leg, i.e. to bring that leg forward (swing leg) and adjust the other one to keep equilibrium.

Paying attention to the sensors of the body, which are quite particular because they do not directly correspond to any joint, we can observe that subject A is the one who used them more extensively. More specifically, he pushed the robot sides to make it swing, imagining the joints to be "elastic" and supposing that by applying a force, the robot will move accordingly given the constraints posed by the ground. Actually, this is the approach we expected for most users. In fact, this approach is similar to kinesthetic demonstration [52, 51]. This also bears a strong resemblance to Yamane and Nakamura's "pin and drag" model, presented for the animation of computer graphics characters in [101]. Subjects B and D used a similar approach, though they seemed to treat the interaction more abstractly. In detail, they identified one sensor which could be associated with the joints they wanted to move and pressed that part any time they needed to use that joints. This fact is easily observed in Fig. 3.23 that shows the sensors used by each of the subjects. Comparing the six panels, we soon notice that subjects B and D employed very few sensors to develop all the motions.

This strict correspondence between the joints to move and the pushed sensor was not maintained by subjects C and F, who used more sensors to express similar joint modifications.

These observations can be verified in a more quantitative way. More specifically, it is possible to calculate a "distance" between the teaching ways, and see the formation of clusters.

Figure 3.24 illustrates the process followed. First of all we created a test set containing the touch instruction and relative context of all the examples provided by the users. For each I_i, consisting of a touch instruction and its context, we calculated the corresponding predicted joint modification using the estimators trained with the data of a single user at a time. Let us denote by $M_{i,u}$ the output of the estimator when I_i is given as input and the data used for training are the ones of user u. We calculated the correlation between the mapping of each couples of users u_1 and u_2 as the average of the cosines of the outputs, i.e.

$$corr(u_1, u_2) = \underset{i}{avg}\left(\frac{M_{i,u_1}^T \cdot M_{i,u_2}}{\|M_{i,u_1}\| \, \|M_{i,u_2}\|}\right)$$

We derived a "distance" between a couple of user u_1 and u_2 taking the negative log of the correlation absolute value:

$$dist(u_1, u_2) = -\log(|corr(u_1, u_2)|)$$

The choice of this function comes from the results of practical experiments conducted in [102]. However, other decreasing functions should provide similar results. Finally we

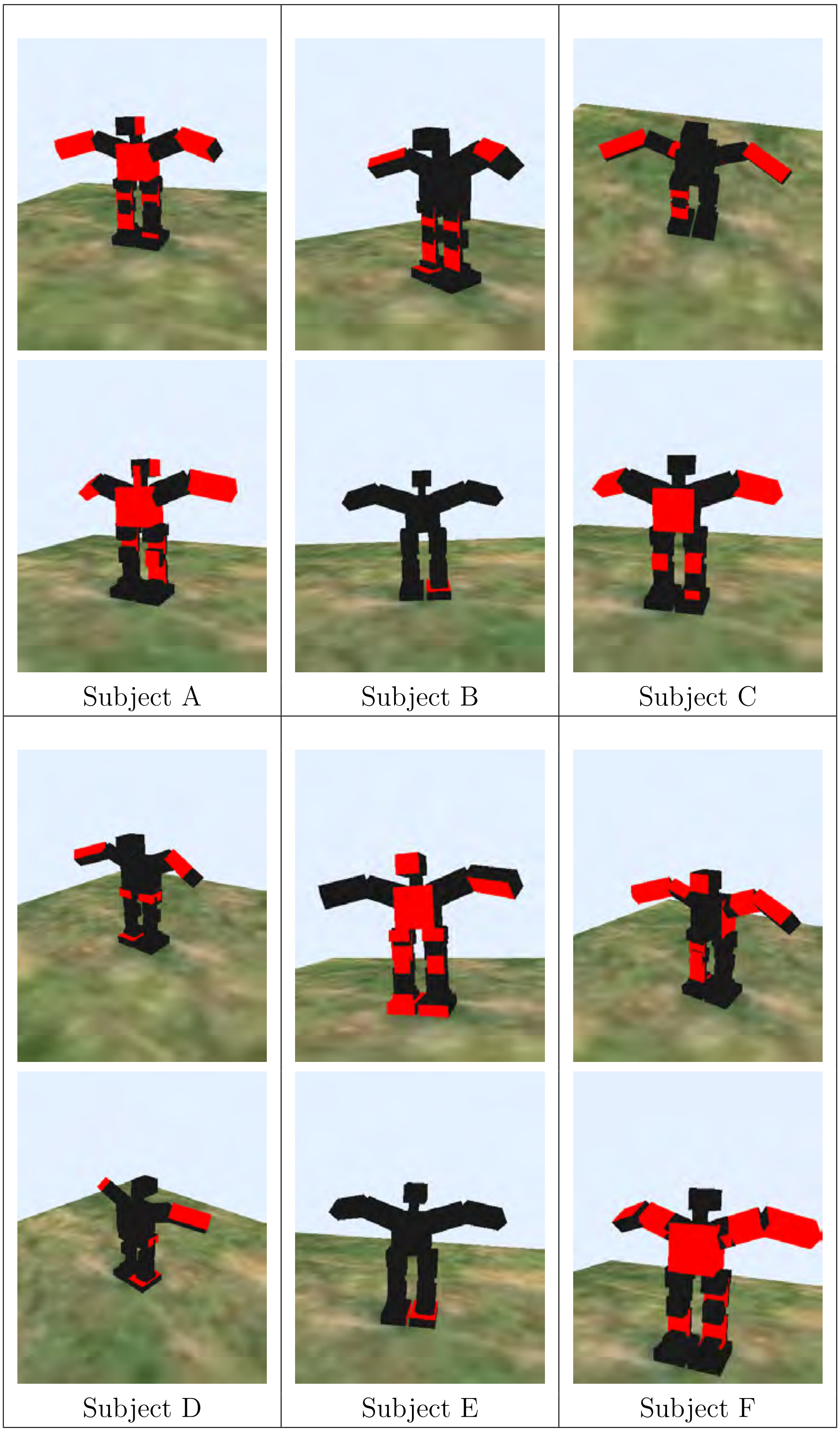


Figure 3.23: For each subject, the sensors touched are displayed by coloring them red. The robot is shown both from the front and from the back.

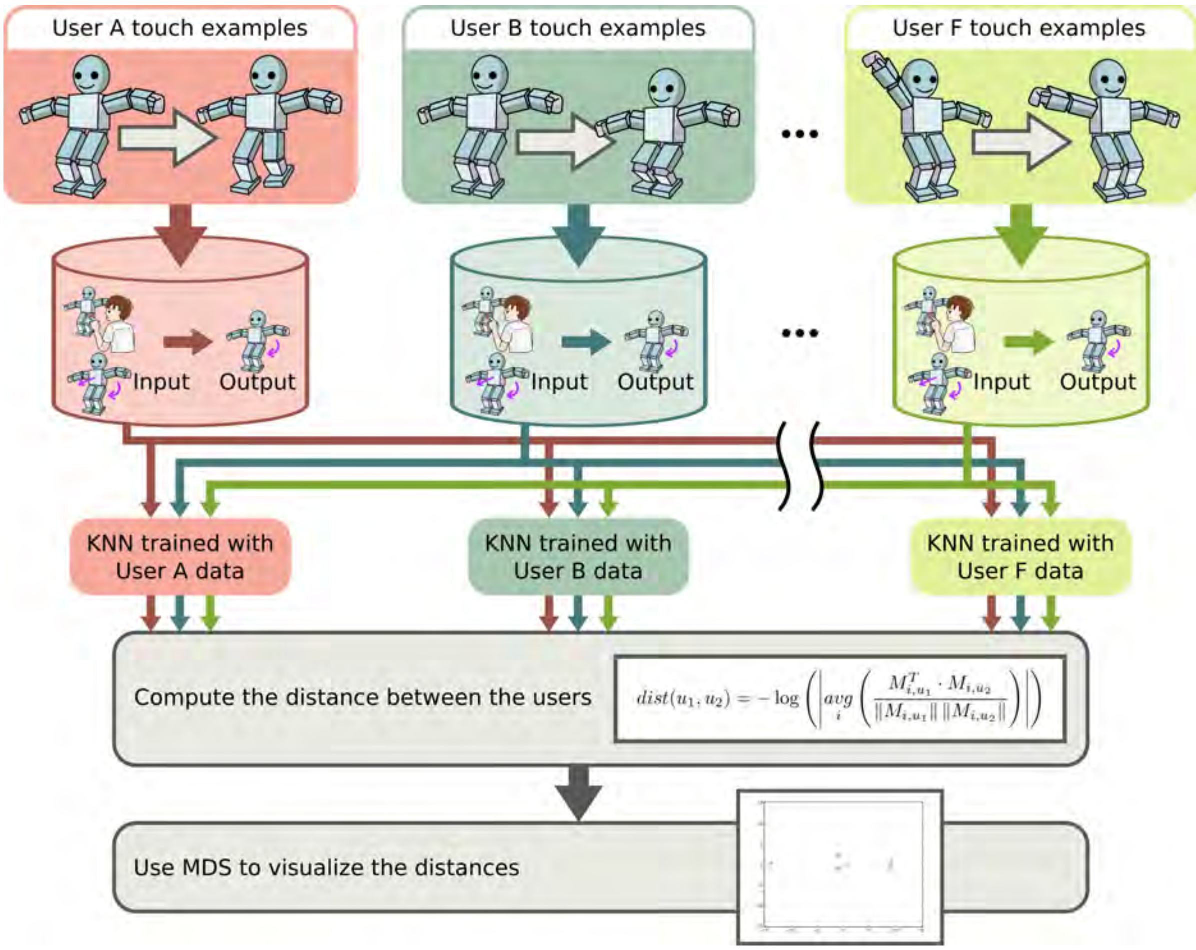


Figure 3.24: Schema of the analysis conducted on the data of the six subjects.

applied multidimensional scaling [103] and obtained Fig. 3.25, a 2D representation of the differences between the users' mappings. We can observe that the locations of subject B and subject D are close, subject E is far from all of the others and subjects C and F are placed in another area. This quantitative analysis seems to confirm our intuitions based on the direct analysis of the tables representing the relationships between sensors and joints.

In detail, the horizontal axis, along which the distance is much bigger, has a role similar to what could be interpreted as the complexity of the mapping:

- A nearly fixed mapping from a small set of sensors to the joints (subjects B and D) or a mapping based on physical considerations (subject A).
- A loose mapping between sensors and motors located on the same limb (subjects F and C).
- a very high level representation of the motion, where, for instance, the only information given through touch is the limb that should be moved; at this level of abstraction a single touch corresponds to a motion primitive(subject E).

The vertical axis appears to represent, for the same complexity, variations in the mapping. We can suppose that A appears near B and D in the low dimensional plot because for many sensors, users B and D decided a fixed mapping that is very similar to the application of force to the robot, i.e. the consideration done by subject A.

Given the clear partition in at least three groups of users we can assume the existence of strong differences in the touching manners. Providing a customized model for each of them would allow us to improve the system's ability to interpret the touch instructions of each group of users. In future works, the number of subjects will be increased to assure statistical significance.

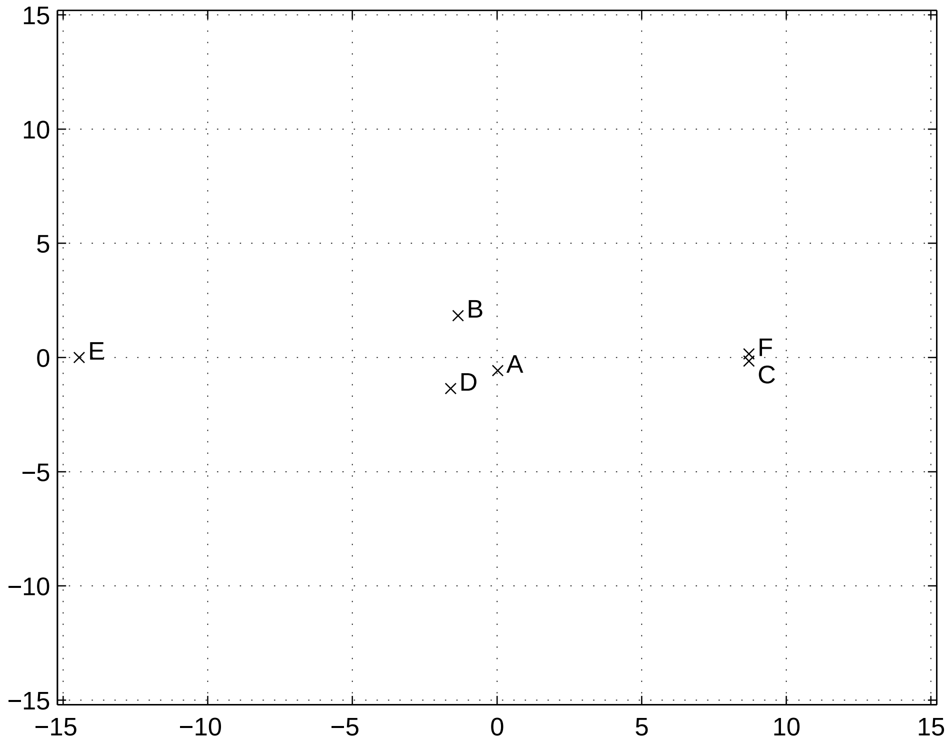


Figure 3.25: Representation of the difference between the user's mapping obtained by employing multidimensional scaling.

4

Direct interaction with a real robot

The previous chapter presented a prototype implementation of the teaching by touching approach that uses simulated touch sensors. Experimental results suggest the potentialities of using the teaching by touching approach for motion programming. Furthermore, Sections 3.4 and 3.5 show that the data collected during the interaction can be used for studying the way humans employ touch to communicate.

This chapter will present results obtained by implementing the teaching by touching on a robot equipped with touch sensors over the whole body. Allowing the users to touch a real humanoid improves the intuitiveness, making the interaction even closer to the one between people. In fact, while with virtual sensors users need to rotate the 3D view and click on the part they'd like to push, with a real robot they just need to touch the part they want to, exactly as they would do with another human.

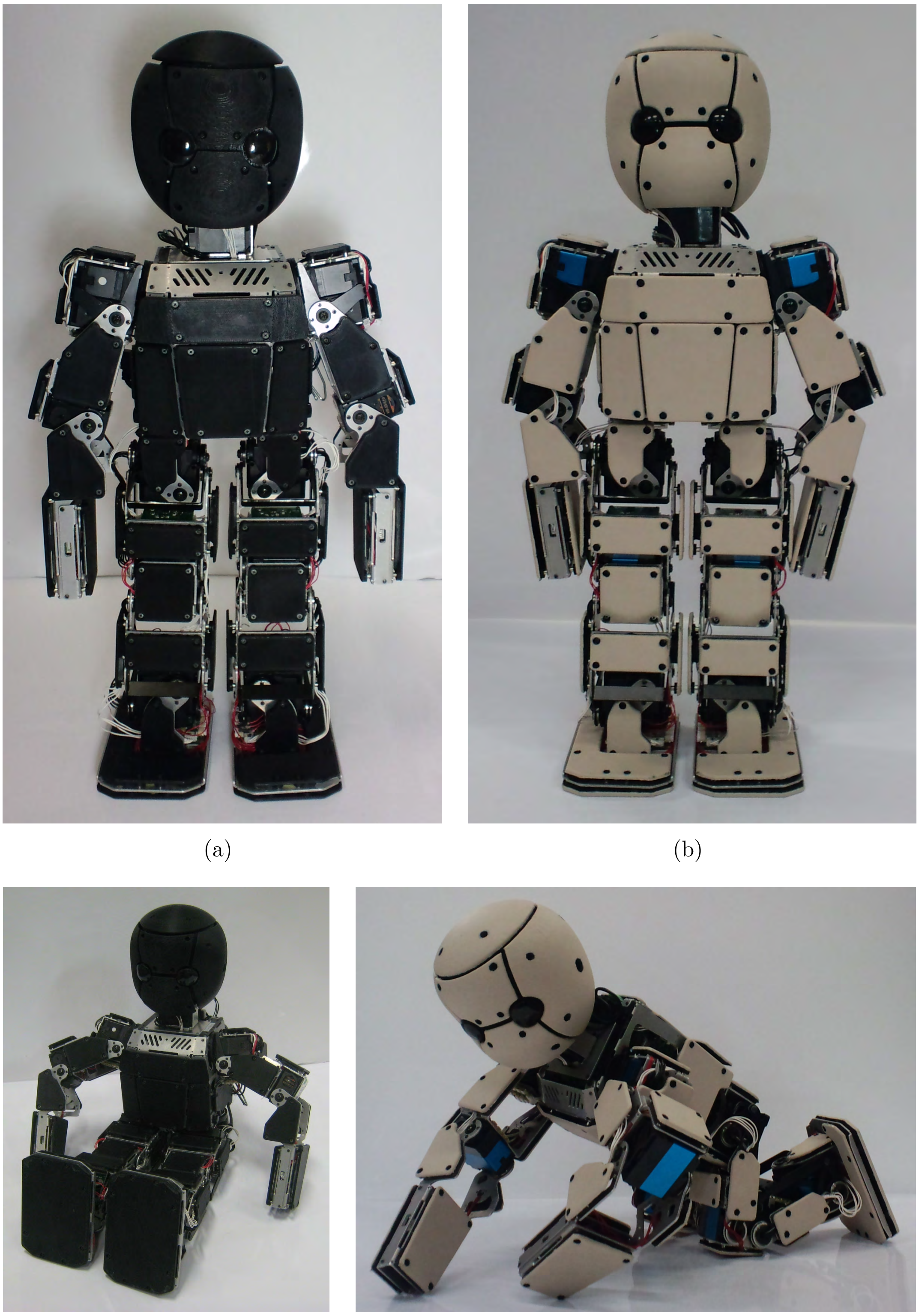
More precisely, the next section will briefly introduce M3-Neony, a new robotic platform used in the experiments. Section 4.2 will briefly describe the implementation of the teaching by touching approach on the real robot. Section 4.3 will conclude the chapter by analyzing the data of four test subjects that interacted with the robot.

4.1 M3-Neony

In order to verify the feasibility of teaching motions to a real humanoid by touching it, and to study how humans intuitively use touch to convey instructions, after experiments limited to simulated touch sensors, we started the design of a small humanoid with touch sensors on the whole body.

The developed robot, called *M3-Neony*, is visible in Fig. 4.1. M3-Neony is a completely autonomous humanoid, developed by the Socially-Synergistic Intelligence group of JST ERATO Asada Project.

The target of the platform are researches in the field of Cognitive Developmental Robotics (CDR)[5]. CDR is a new approach to study mechanisms of human intelligence, cognition, and development with robots and computer simulations. In this approach we usually hypothesize a model of infant development based on existing knowledge in



(a) (b)

(c) (d)

Figure 4.1: Photos of M3-Neony. Two copies of the robot were developed. The two robots are identical, the only difference is in their color.

psychology and brain science, implement the model in a humanoid robot that has human-like kinematics and a perceptual system similar to humans, and verify the hypothesis by comparing the resultant robot behaviors with the actual infant development. This requires a humanoid robot which can behave in a baby-like manner and naturally interact with people, and, therefore, the development of appropriate humanoid robot platform is important. In order to cover a wider area of research issues on infant development, it is desirable that the robot platform has a baby-sized body, a multimodal perception system, and a motor system able to perform dynamic whole-body movements. Another important factor in the development of a platform is modularity and ease of use, which is necessary for sharing the platform among researchers who work in CDR.

The growing interest in CDR and the consequent need of new baby robot platforms seems to be confirmed by the recent development of child robots like CB^2, Child robot with Biomimetic Body [6], and iCub [7]. CB^2 is a humanoid robot driven by 56 actuators most of which are pneumatic to allow a safe interaction with humans. This robot is equipped with encoders on all of the joints, accelerometers and gyroscopes on the three axis, two cameras located in the eyeballs, two microphones mounted in the head and with PVDF (Polyvinylidene Difluoride [104]) film based tactile sensors distributed over the whole body. Similarly, iCub presents a high number of degrees of freedom, precisely 53, which are however powered by electrical motors. This platform is also equipped with encoders, a binocular vision system, gyroscopes and accelerometers, microphones and force/torque sensors. A sensor skin, based on a mesh of sensors interconnected, is currently under development [105]. Both of these two robots are quite big and heavy, expressly CB^2 is about 130cm high and weights 33Kg while iCub is 94cm tall and weights 22Kg.

In order to decrease the cost, to increase the user safety by diminishing the necessary motor torques and to allow close interactions like hugging, M3-Neony was developed.

The developed platform is completely autonomous, and can be powered both by an external supply or by batteries, two Nickel-metal hydride battery packs lodged in the robot's torso that provide 14.8 V and 1400 mAh. The robot weights about 3 kg and is approximately 50 cm tall.

Despite its contained size and cost, it is equipped with a high number of sensors. In detail, each joint includes a potentiometer and a temperature sensor. The torso mounts two gyroscopes (ENC-03R by Murata Manufacturing Co.) and a three axis accelerometer (Freescale Semiconductor's MMA7260Q). Two cameras (Elecom's UVC camera UCAM-DFL30, see 4.1) and two microphones are located in the head (see Fig. 4.2(b)) and 90 touch sensors are distributed over the whole body.

Computation is provided by two boards. The first board, intended for high level computation, is a generic low consumption 500 Mhz Geode based CPU board, PNM-SG3 from Pinon[1]. As a result of having an x86 compatible Geode LX800/CS5536 processor

[1] `http://www.pinon-pc.co.jp/`

Table 4.1: Elecom UCAM-DFL30 characteristics

Pixel Count	300K Pixels
Photoreceptor	1/6 inch CMOS sensor
Max. resolution	640 × 480 pixels
Max. frame rate	30fps (when up to 640×480 pixels)
Number of colors	16,770,000(24bit)
Max. Power consumption	Waiting: 0.25W, Working: 0.55W
Interface	USB 2.0

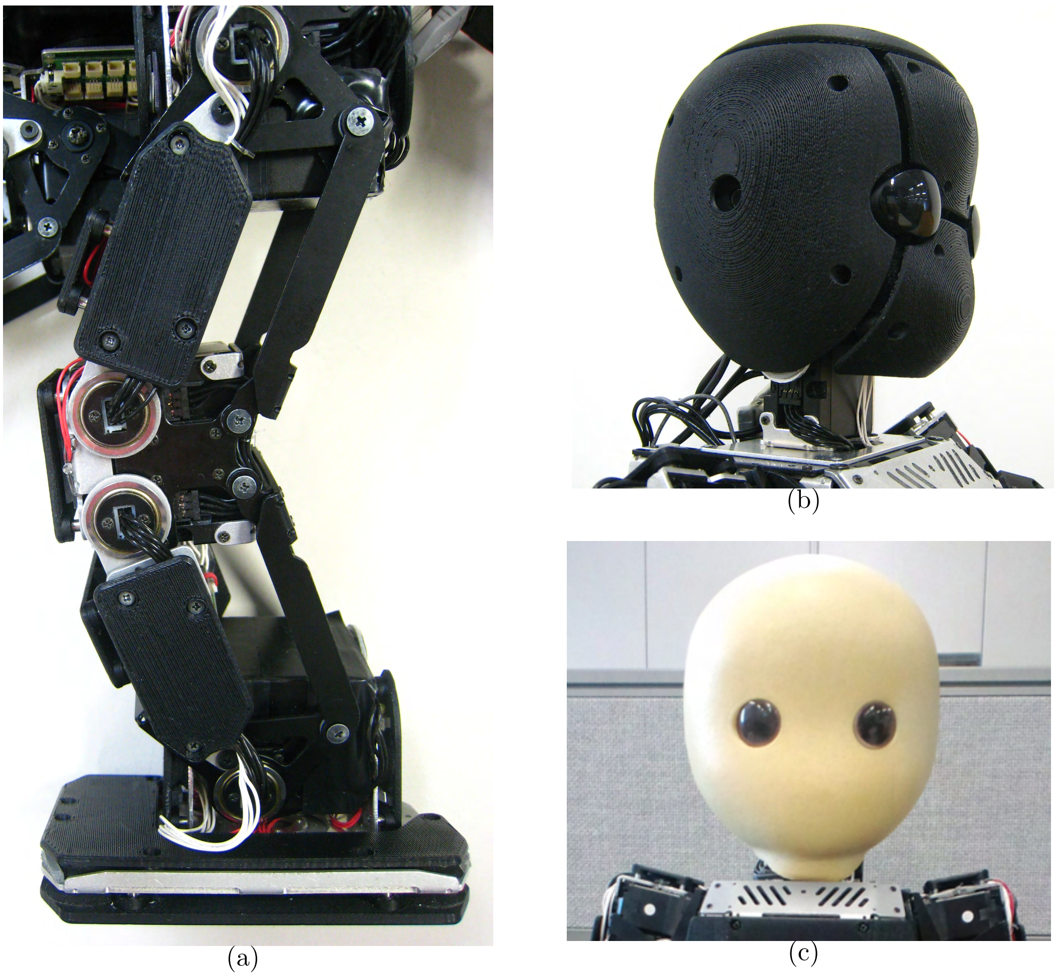


Figure 4.2: Particulars of M3-Neony. (a) Parallel links give robustness to the robot legs. (b) M3-Neony's head, equipped with two microphones, two cameras and 15 tactile sensors. (c) An alternative head, covered by polyurethane.

and 512Mb of DDR400 RAM, the board is able to run a conventional operating systems like Windows XP or Linux on the robot while keeping the power consumption very low (only 5W). The board also features a conventional VGA output (resolution 1920x1440, 32bit color depth), a USB bus and two serials. Storage is provided by a compact flash disk, and a supplementary compact flash slot allows plugging in a network card for connection to conventional IEEE 802.11b and 802.11g wireless LANs.

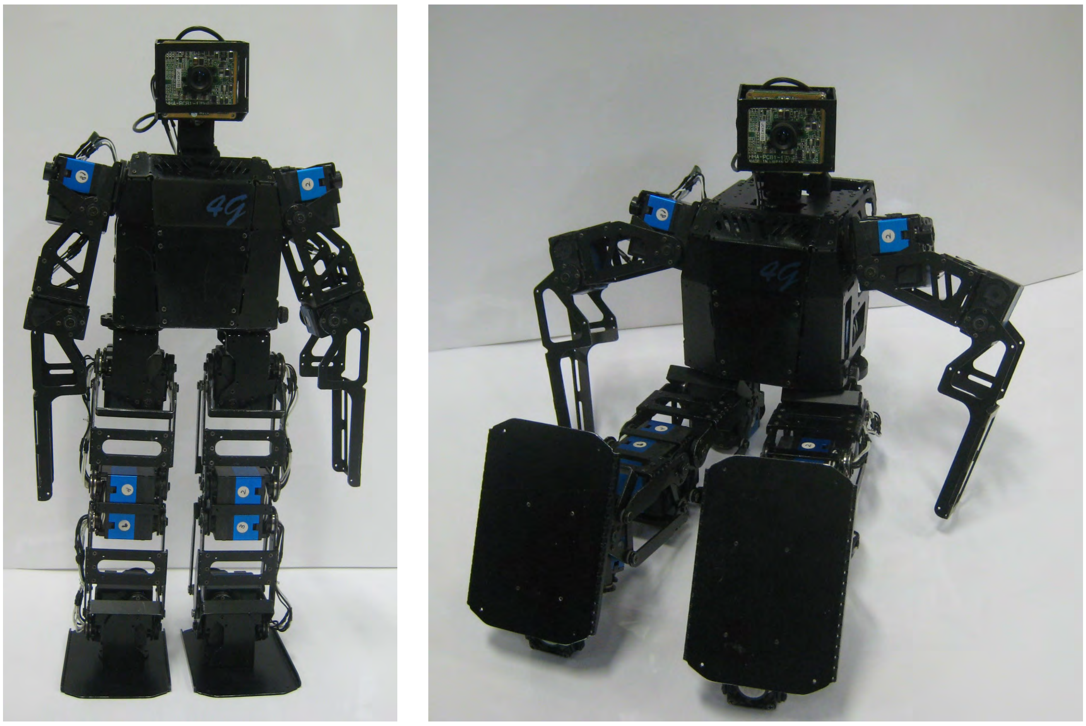
The second board, dedicated to motor control, is a VS-RC003, a 60 MHz ARM7 powered control board from VStone[2]. This board, that is usually controlled by the GEODE board through a serial connection, presents a USB mini B connector for direct communication with an external PC. The USB communication protocol is managed by VStone SDK that allows development of motions and acquisition of gyroscope and accelerometer information by a set of high level API. These API are fully integrated with the Robovie Maker development environment released by VStone and it is possible, for instance, to create robot motions by a classical slider based interface and to execute it by few lines of code. Simple signals can be sent to the ARM board also by a wireless bluetooth Playstation 3 controller (SCPH-98040, trademarked "SIXAXIS"), permitting, for instance, real time execution of different sets of motions.

The mechanical structure of M3-Neony is based on VisiON 4G, a commercial humanoid manufactured by VStone. VisiON 4G is a high performance humanoid robot used by Team Osaka, a RoboCup [106] team that won the kid-size humanoid soccer competition three times and the Louis Vuitton Best Humanoid Award for five consecutive years.

VisiON 4G, shown in Fig. 4.3, was designed to resist impacts with opponents during the RoboCup games, and therefore reveals to be a very robust robot. It is completely built in aluminum and its legs are realized by parallel links. M3-Neony maintains these technical solutions, as shown in Fig. 4.2(a). Furthermore, its touch sensors were developed aiming at maintaining this robustness, and actually the presence of elastic elements in their structure even improves the absorption of shocks.

The 22 degrees of freedom (DOFs) of VisiON 4G are maintained unchanged in M3-Neony, except for minor modifications in the head structure. In detail, the two robots present 7 DOFs for each leg, 3 for each arm and 2 DOFs for the head, as shown in Fig. 4.4. The joints are actuated by VS-SV410 servomotors, metal gear PID controlled motors that are able to provide a torque of 41 kgf·cm. Each of the servomotors includes a micro-controller, namely C8051F411 by Silicon Laboratories, that enables communication through a serial line. A simple protocol permits to set the joint target position, modify the PID parameters, read the position, the temperature, the voltage and so forth. All the servomotors are actually connected to an ARM based motor board by a single serial bus, a solution that strongly simplifies the wiring and improves the modularity of the system. Table 4.2 provides a summary of the motor characteristics, while

[2] http://www.vstone.co.jp



(a) (b)

Figure 4.3: VisiON 4G. Apart from the head and the hands, M3-Neony's metallic frame is identical to the one of VisiON 4G.

Figs. 4.5 and 4.6 reports the measured response of the motor.

A modular structure, analogous to the ones employed for the motors, is used for the tactile sensors. Expressly, groups of up to 8 analog tactile sensors are connected to boards based on Silicon Lab's C8051F411, a tiny microcontroller equipped with a 20 channels, 12-bit, 200 ksamples/s analog to digital converter (ADC). 19 of these boards are distributed over the whole body and interconnected by a serial bus. A protocol identical to the one of the servomotors provides access to each of the 90 sensors, whose location is reported in Fig. 4.7.

Fig. 4.8 reports a schema of the structure of M3-Neony's touch sensors. The working principle is analogous to the force sensing adopted for the micro-mechanical gripper

Table 4.2: VS-SV410 characteristics

size	$40.5 \times 21.0 \times 32.9$mm
weight	62g
torque	41Kgf cm (at 16.8V)
speed	0.14s/60°
range	180°
power supply	min 10V max 18V

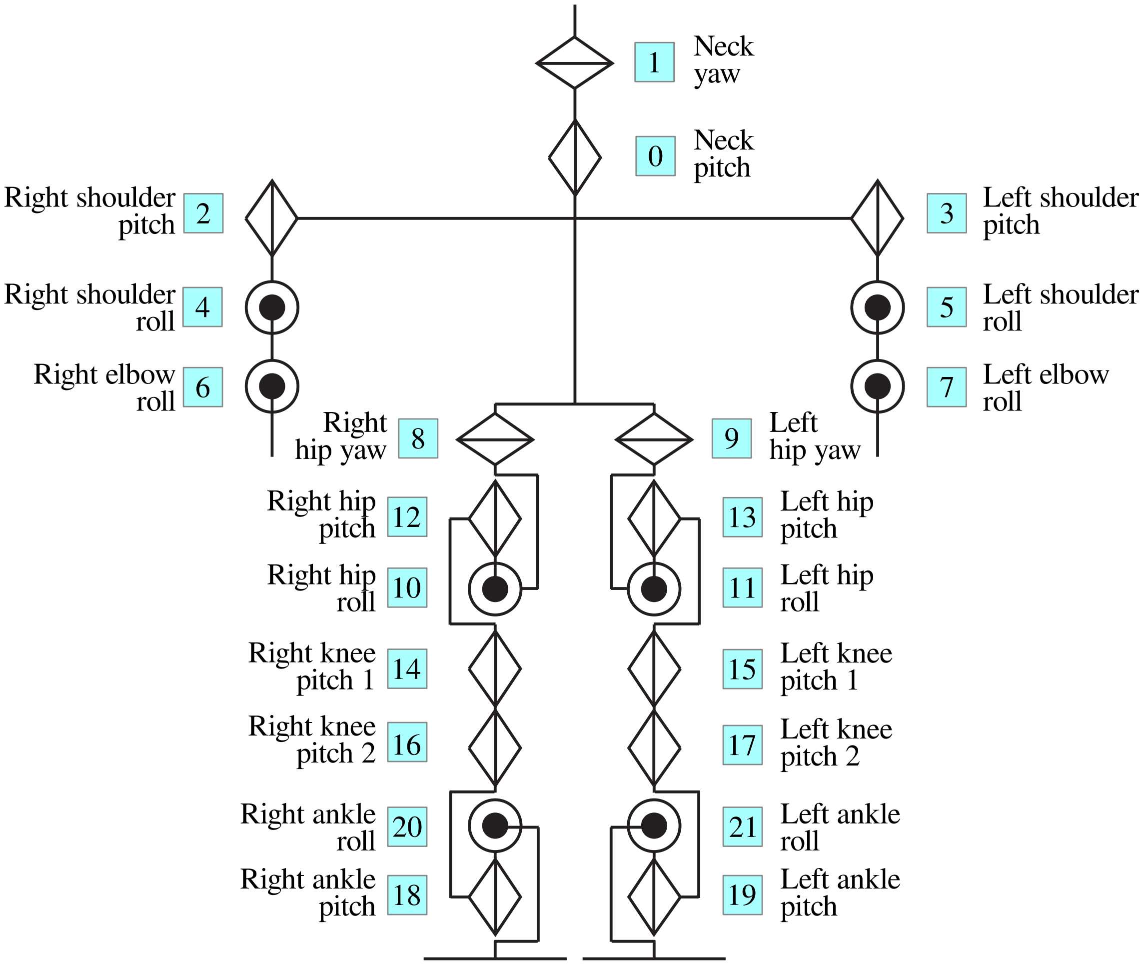


Figure 4.4: Degrees of freedom of M3-Neony. As can be seen, the structure is analogous to the one of VisiON 4G (see Fig. 3.9) except for the head.

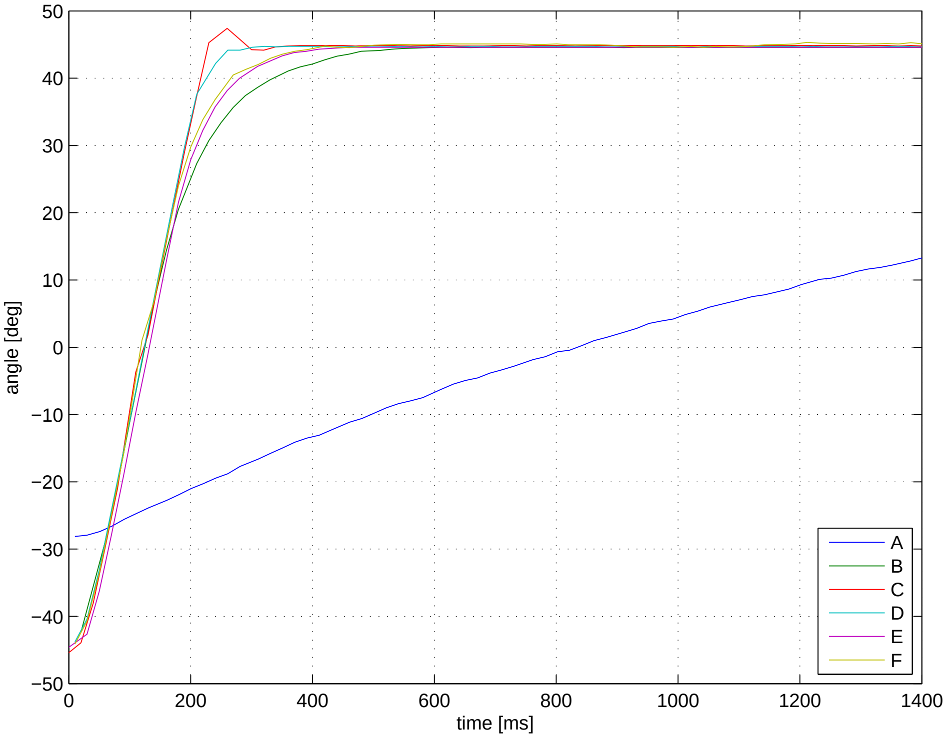


Figure 4.5: Measured responses of the servomotor for a rotation of 90 degrees and different configurations of the PID parameter. See the VS-SV410 manual for a complete list of the parameters. The X axis reports the time from the provision of the command in milliseconds, and the Y axis reports the angle read. Each plot corresponds to one of the following configurations (unspecified parameters are kept to their default value).

Config.	PID P	PID D	Back-EMF Coeff	PID I
A	1	0	0	0
B	16	0	0	0
C	32	0	0	0
D	32	32	0	0
E (default)	32	32	4	0
F	32	32	4	32

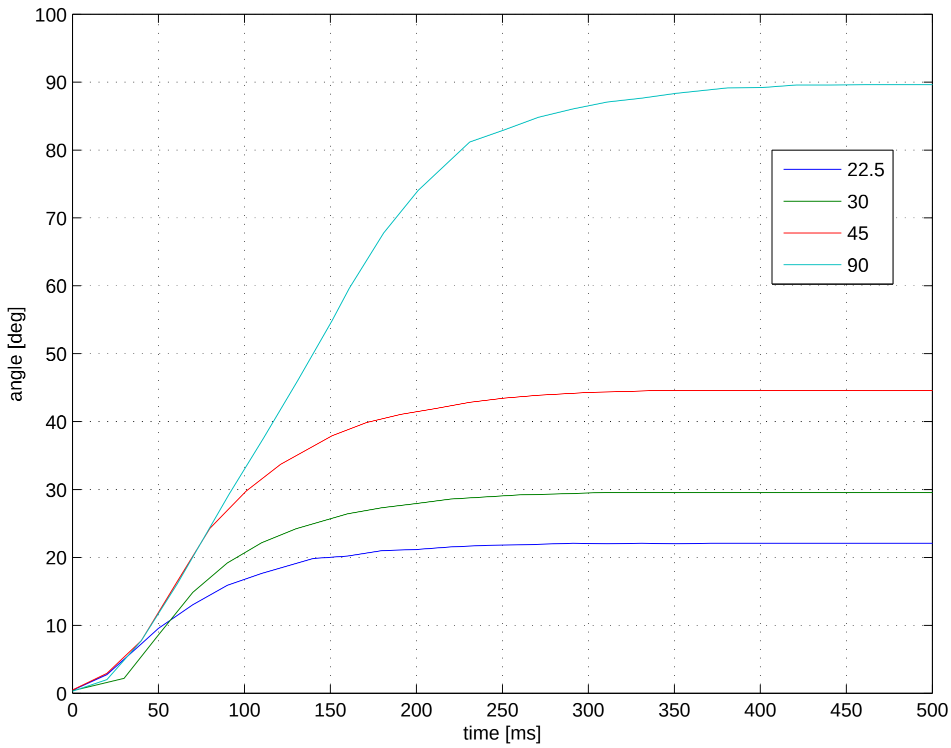


Figure 4.6: Response for different target angles (22.5, 30, 45 and 90 degrees). The X axis reports the time from the provision of the command in milliseconds, and the Y axis reports the angle read.

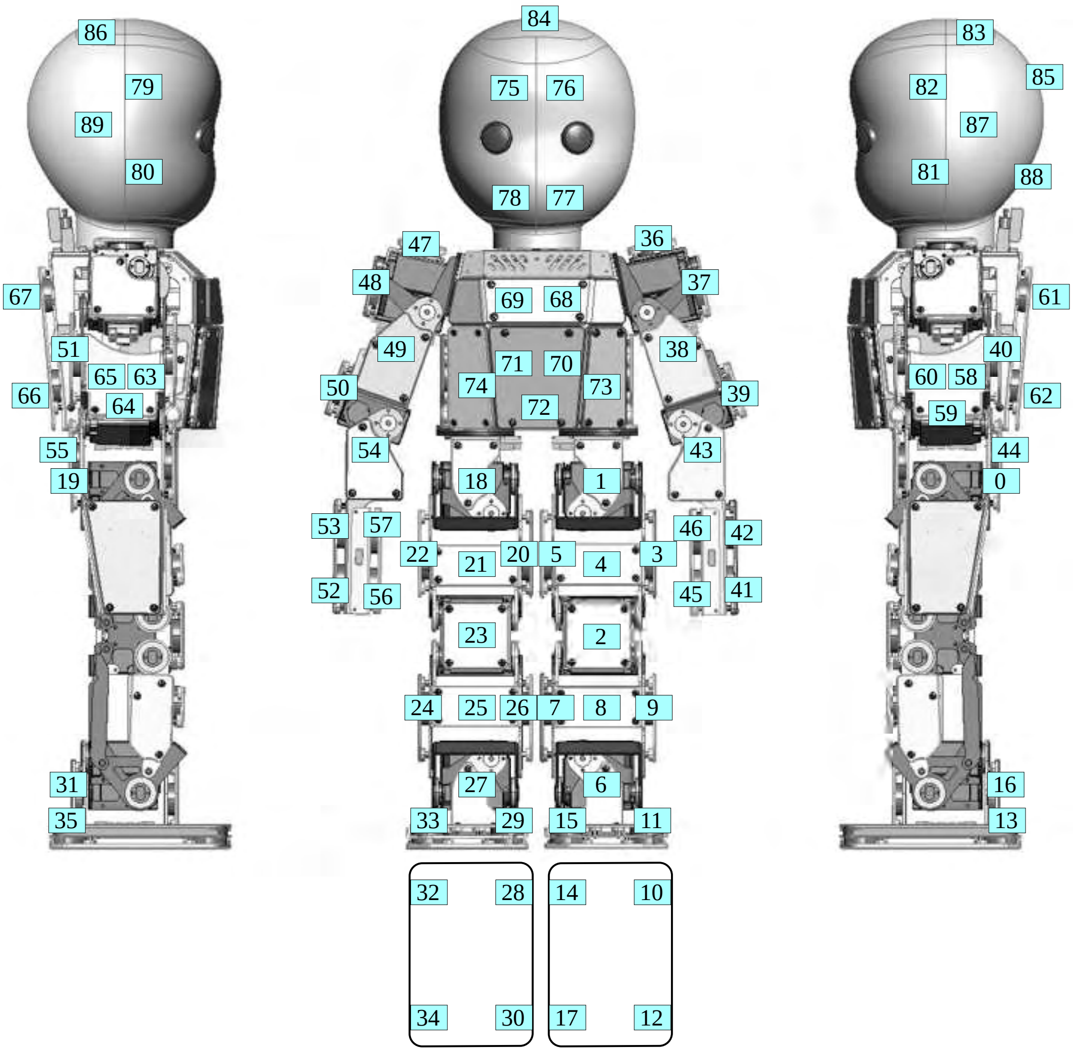


Figure 4.7: Arrangement of the touch sensors of M3-Neony.

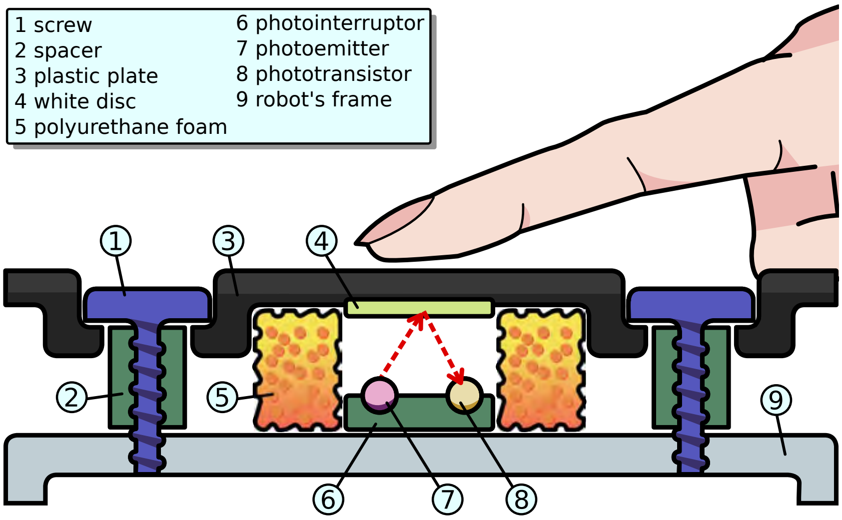


Figure 4.8: Schema of the touch sensors.

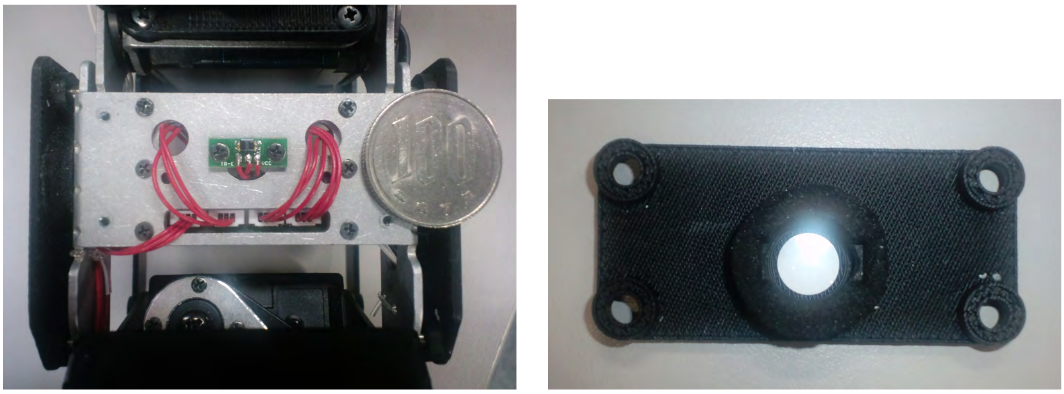
in [107]. Concretely, a photointerrupter (Sharp's GP2S60) is mounted on a small (12x4 mm) board fixed to the robot's aluminum frame, as shown in Fig. 4.9(a). A plastic plate, visible in Fig. 4.9(b), is placed over the photointerrupter and kept at a distance of 5mm by layer of polyurethane foam.

The light coming from the photoemitter of the photointerrupter is reflected by a white disc attached to the plate and intercepted by the phototransistor of the photointerrupter. When pressure is applied to the plastic plate the polyurethane foam acts as a spring that compresses, and the white disc gets closer to the photointerrupter. Since change in the distance translates in a variation of the reflected light, the amount of light captured by the phototransistor can be used as an indicator of the applied force.

Although the relationship between the applied force and the reflected light measured by the phototransistor is strictly monotonically increasing, it is nonlinear and the small differences in the elasticity of the materials make calibration necessary for each of the sensors. In particular, the range of the values read by each of the photointerrupters is different. Note that, however, in human-robot interaction studies, categorizing the touch in two or three classes like no pressure, soft pressure or strong pressure often suffices. In these cases, a simple calibration that determines the offset value and the range of each sensor reveals to be sufficient.

The access time for a single sensor is on average 10 ms (min 6 ms, max 19 ms), for the 8 sensors of a board is 20 ms (min 15 ms, max 32 ms) and for the whole 90 sensors is 110 ms (min 106, max 130 ms). The speed bottleneck is given by the serial communication, so speed improvements could easily be obtained by increasing the number of serial buses, at the expense of a more complex wiring.

Actually, these sampling times completely meet the teaching by touching requirements, as practical experiments showed. Next section will briefly introduce the teaching



(a) (b)

Figure 4.9: Structure of the tactile sensors. (a) The photo interrupter board. (b) One of the plastic plates placed over a photointerrupter.

by touching system developed for the real robot, and Section 4.3 will present results of preliminary experiments conducted with M3-Neony.

4.2 System description

Exploiting the interpretation of tactile instructions appears to be a possible solution for motion design. In fact, the pilot study results in Section 3.4 showed promising results in terms of reduction of motion development time. The intuitiveness of direct, physical interaction with a real robot is expected to ease the realization of robot motions even more. Consequently, we devised and implemented a system that allows developing robot movements using tactile instructions.

The system was designed to keep the development process as simple as possible for novice users. Expressly, as described in Chapter 3, the users first select a certain moment of the motion using a time-line on the Graphical User Interface (GUI) shown in Fig. 4.10. Then, using tactile instructions, they are able to set the position that the robot should assume at that time. As was shown in Fig. 3.1, when users provide a touch instruction, the robot may respond in ways that do not correspond to what the users expected. In this case the users can employ kinesthetic demonstration to show the robot the posture modification that they expected. The robot memorizes the association between the haptic interaction and the desired movement, and gradually improves its capacity to estimate the meaning of touch instructions.

More concretely, the interface presents two operating modes, the *motion development* mode and the *touch meaning provision* mode. The system is usually in the *motion development* mode, waiting for user touches that edit the robot's posture at the selected keyframe. When the robot does not understand correctly the meaning of the touch pattern, the operator can switch the system to the *touch meaning provision* mode by stepping a USB pedal, a device commonly used in racing video games (see Fig. 4.11). When the pedal is stepped, the robot moves back to the posture it had before misinterpreting the tactile instruction, and the users are able to show the robot the intended movement by kinesthetic demonstration, i.e. they can move the robot to the posture it should have gone by the provided touch instruction. In particular, during this mode, when the sensors on both sides (palm and top) of one hand are pressed, the motors of that arm are switched off so that the complete limb can be moved freely. When the sensors are released, the power is turned back on and the new posture of the arm is maintained. Similarly, the servos of each leg can be turned off by pressing the top and bottom side of the corresponding foot and the motors of the head can be powered off by pushing the front and back sensors of the head. Once the users want to terminate demonstrating the robot how it should have moved, all they have to do is just to release the pedal. This brings the system back to the *motion development* mode and the users can continue their movement creation [3].

Actually, when the pedal is released, a new association between the touch pattern given at the beginning of the *touch meaning provision* mode and the movement provided is stored in a database used by the supervised learning algorithm described in Section 3.1.

Only a little difference exists between the algorithm used for the real robot and the algorithm presented in Section 3.1. More precisely, given the limited importance of the center of gravity velocity vector found in the experiments, we decided to remove it from the context features, allowing the development of a system that does not require using a motion capture system during the motion development. The orientation of the robot was maintained as a context feature, but instead of being computed from the motion capture data, it was estimated using the accelerometer and gyroscope data using the algorithms in [93].

All of the other features of the prototype system are maintained. For instance, as previously done, initially the database is empty and the robot does not respond to any touch instruction. Using the *touch meaning provision* mode, the database is incrementally populated and the interpretation is improved more and more. This presents the advantages discussed in Chapter 3. More precisely, thanks to the supervised learning approach the system completely adapts to the user, who is completely free in creating associations between touch patterns and movement modifications. Additionally, as in the previous setup the user is not requested to teach the meaning of the touch instructions in a specifically devised session, but can provide them in contexts that naturally appear during the motion development. Finally, as for the prototype system, data collected during the interaction can be analyzed, and general policies underlying the user's way of

[3]Note that the configuration at the end of the *touch meaning provision* mode is stored as a keyframe of the motion, i.e. the robot does not return to the posture it had at the beginning of the *touch meaning provision* mode. This is because, in most cases, after teaching the desired movement, the robot will probably be in a state closer to the one desired for that moment.

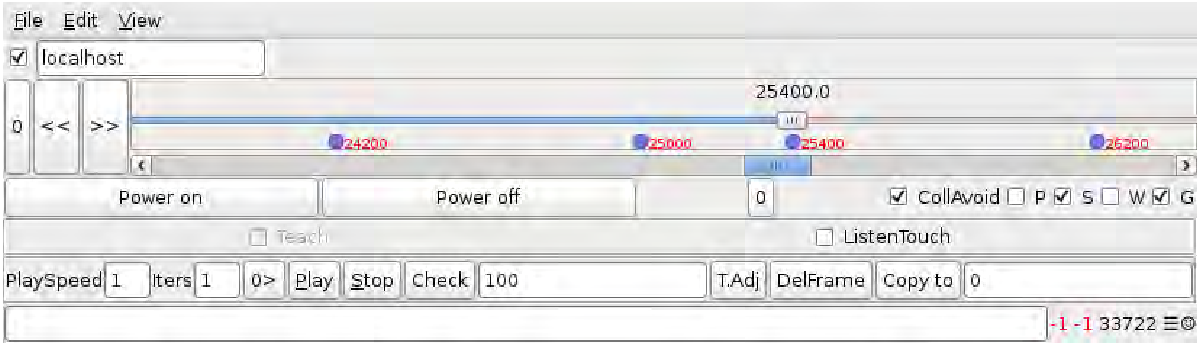


Figure 4.10: The graphical interface of our system. Exploitation of the touch sensors reduces the interface essentially to a timeline.

associating touch patterns to robot movements can be extracted.

Actually, identifying the features of the mapping from touches to motor changes will allow us to design new algorithms for the interpretation of haptic instructions. Hopefully, algorithms specifically devised for the task will require the provision of less examples of the mapping and will be more accurate in estimating the correct meaning of the instructions.

Although, to keep things simple, both the prototype system and the real robot implementation use a key-frame based motion representation, we note once more that the idea of interpreting the meaning of tactile instructions to modify the motion is not restricted to a keyframe based motion representation. Surely, the usage of Nonlinear Dynamical Systems [34], or of other motion representation that make use of the sensory feedback or that keep a probabilistic representation of the motion [52], will improve the applicability of the taught motions. In fact, Chapter 5 will present an implementation of the teaching by touching approach where a Central Pattern Generator (CPG) is used to define the movement.

We also need to note that the current system implementation uses kinesthetic demonstration for teaching the meaning of touches. We previously compared kinesthetic demonstration and the teaching by touching approach, and discussed the advantages of the teaching by touching paradigm. In particular, we noticed that the teaching by touching paradigm allows a single touch to be associated to a whole body movement, while when using kinesthetic demonstration, moving more than one limb at a time may be impossible. Furthermore, if no expedients for compensating the gravity are taken, interacting with big robots may be problematic.

We could be puzzled on whether this could bring in the drawbacks of kinesthetic demonstration. However, we note that

1. The kinesthetic demonstration is used just to teach the meaning of tactile instructions, and not for motion development.

2. The meaning of a touch needs to be taught just once, and can thereafter be reused.

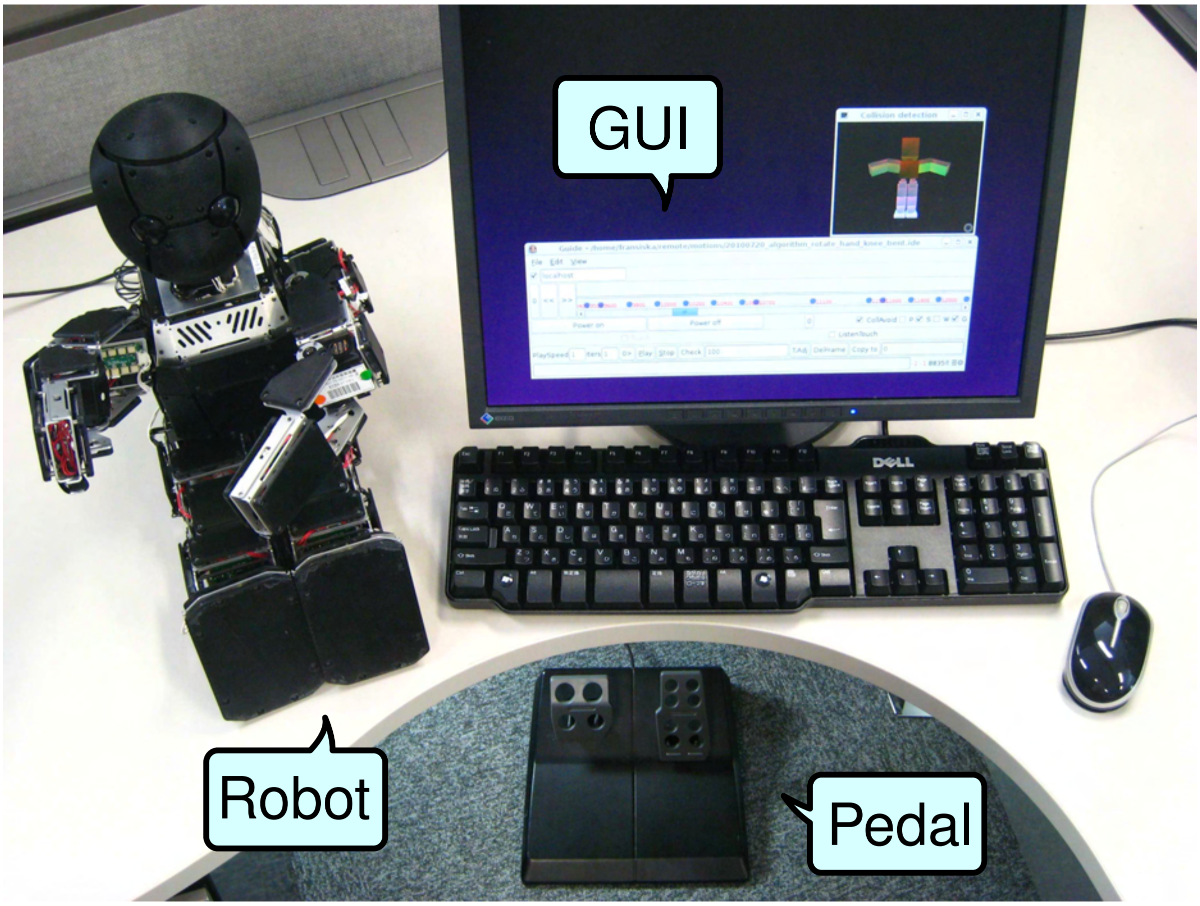


Figure 4.11: Experimental setup. A pedal device, commonly used for racing games, is used to switch between the *motion development* mode and the *touch meaning provision* mode. A pedal allows the users to easily switch the mode without having to move their hands away from the robot. Other modalities, like voice recognition, could be introduced in future versions of the system.

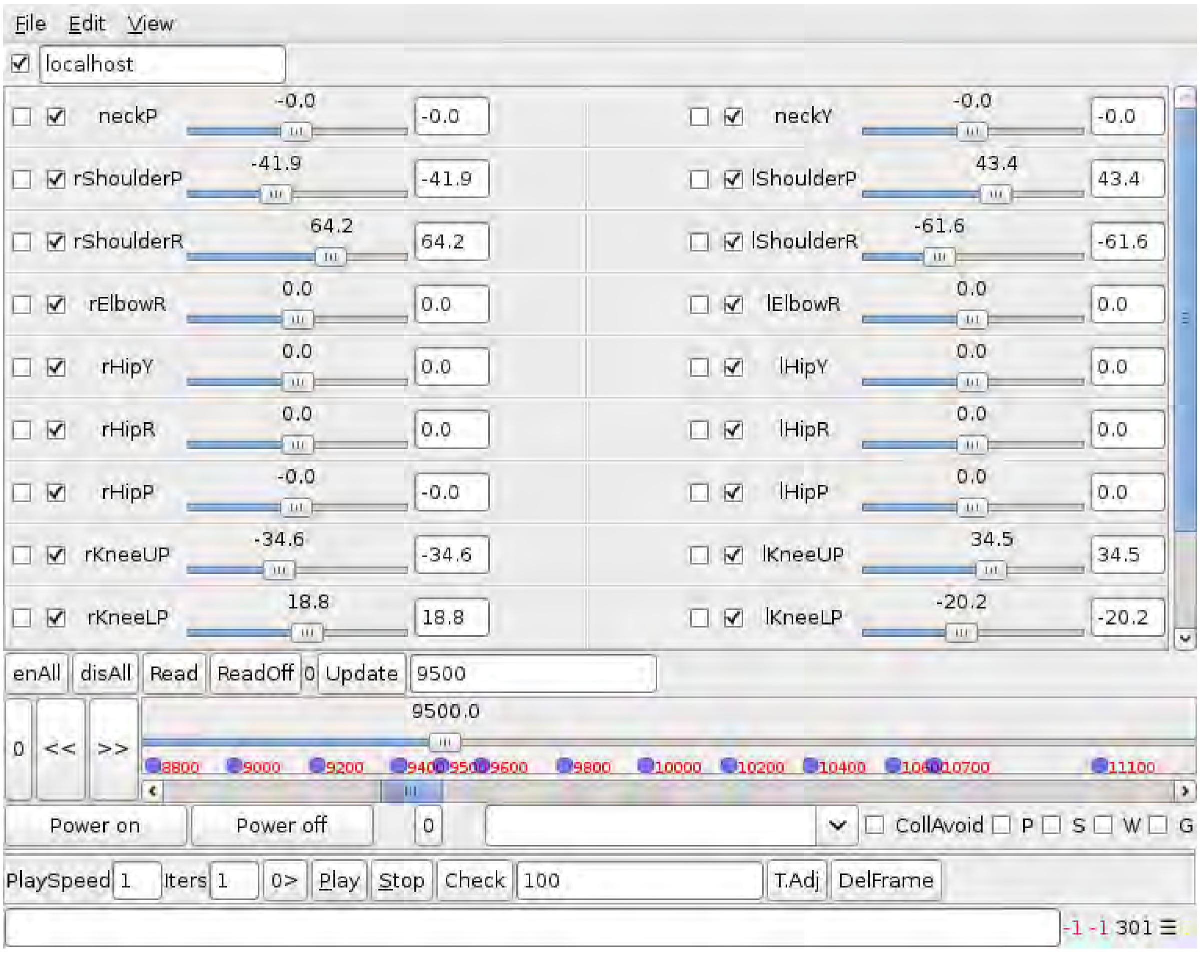


Figure 4.12: The graphical interface of the system when the sliders for direct joint control are shown.

This means that the user needs to face the drawbacks much less frequently than with kinesthetic demonstration.

3. Kinesthetic demonstration is just one way to teach the robot the meaning of the touch. In fact, the interface already includes the possibility of displaying the panel shown in Fig. 4.12. As with the prototype interface, the sliders in the GUI can be used to accurately move the joints. Voice recognition, or any other human-robot interface, could be used as well.

The next section will present preliminary experiments conducted with the system. Results showed that users who were unfamiliar with the system could soon interact with the robot through the teaching by touching approach, and successfully developed a complex dance motion. Once more, the system revealed to be able to learn the meaning of tactile instructions and require the user to teach less and less instructions over time. Additionally, simple analysis of the data acquired during the interaction provided new interesting insights on the interpretation of tactile instructions.

4.3 Experimental results

This section will present an experiment conducted with the system introduced in the previous section, and will provide an analysis of the touch instruction data gathered during the interaction. In particular, the analysis confirms that the mapping between touch instructions and the movement to be executed is not represented by a trivial correspondence between sensors and posture changes. On the other hand, analysis of the experimental data will suggest that the way in which the robot should respond to touch instructions can be described by using a low-dimensional subspace of the complete joint space. Interestingly, this low-dimensional subspace can be used to describe the movements that should be performed in response to tactile instructions as well. This allows us to advance the hypothesis of the possibility of exploiting knowledge on the motion being taught to improve the interpretation of tactile instructions.

In the experiment, four users were asked to teach a humanoid robot a motion using the teaching by touching algorithm. Expressly, the subjects were requested to develop the first half of *Algorithm Exercise*. Algorithm Exercise is a famous dance appearing in a TV show for kids aired by NHK, the Japanese national public broadcasting organization[4]. This motion involves changing the facing direction and a series of simple hand and leg movements, as shown in Fig. 4.13.

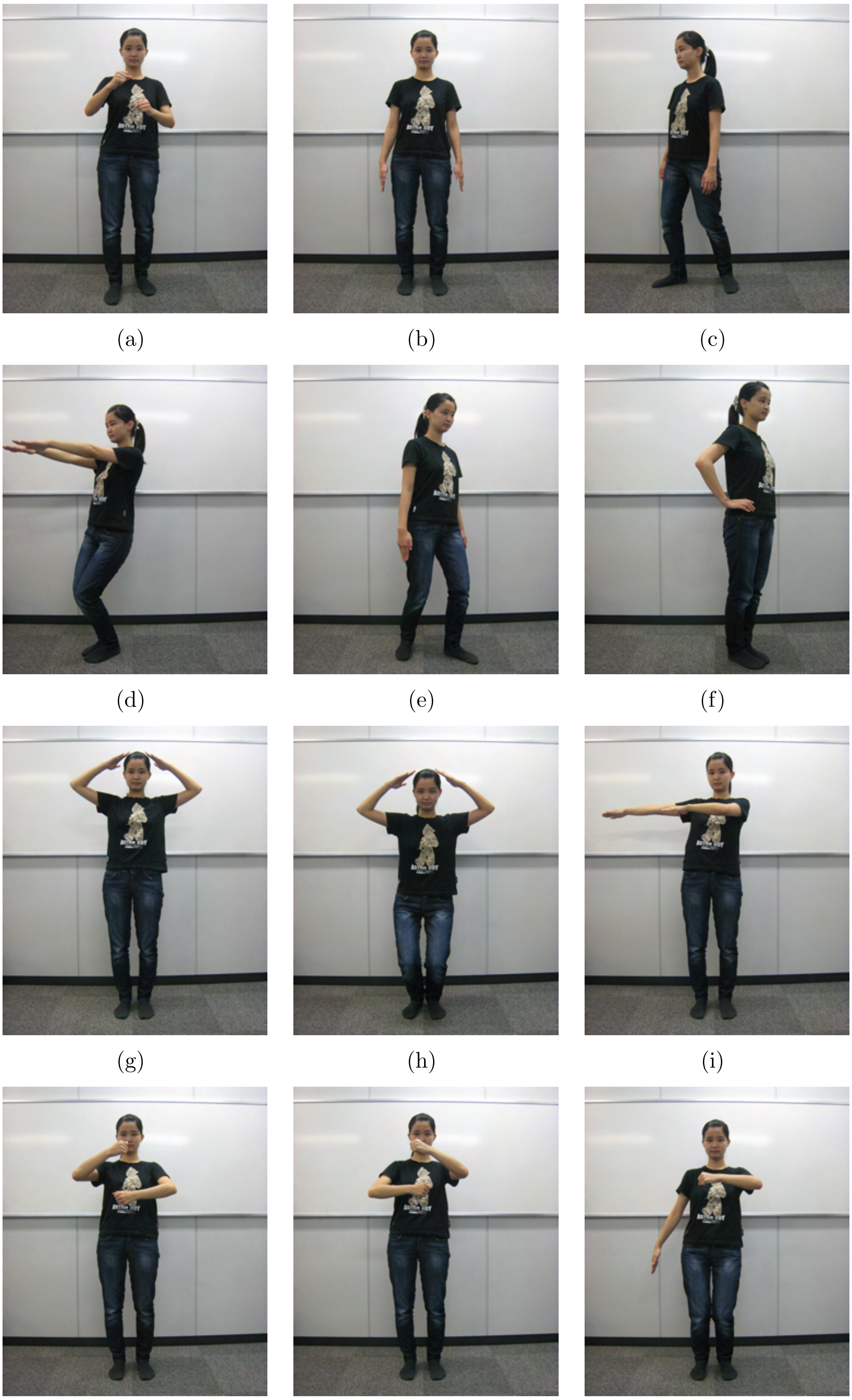
This dance was chosen because on the one hand it is complex enough to require the teaching of a large number of different postures to the humanoid and on the other hand it is simple from the viewpoint of robot balancing.

The four test subjects who participated in the experiments are Japanese Engineering students at Osaka University, three males (subjects A, C and D), one female (subject B), all right handed, age ranging from 23 to 25 years. These users are familiar with VisiON 4G, the robot lacking tactile sensors that was employed for the experiments reported in Section 3.3. The subjects, however, never used the teaching by touching interface nor knew its underlying concepts. All of the users could effectively employ our system to develop the motion, with an explanation of the system usage that required less than 5 minutes. Snapshots of the realized motion[5] are reported in Fig. 4.14.

During the motion development, the users provided the meaning of an average of 95.75 (standard deviation 6.55) tactile instructions. Actually, once instructions are taught, these can not only be reused, but also combined, simply by pressing multiple sensors simultaneously. For instance, imagine to teach the robot to look downwards when the chin is pressed and then to instruct it to turn the head leftwards when the right cheek is touched. If the chin and cheek are pushed together, the robot will head its gaze to its own left foot. In the following let us denote by the word *instruction* a basic association between sensors and angle changes taught, and by the term *touch* a tactile

[4]See `http://www.nhk.or.jp/kids/program/pitagora.html` for the details.

[5]A video is available at `http://robotics.dei.unipd.it/~fabiodl/video.php?tbt`



(a) (b) (c)
(d) (e) (f)
(g) (h) (i)
(j) (k) (l)

Figure 4.13: Snapshots of a human performer executing the dance chosen for the experiment.

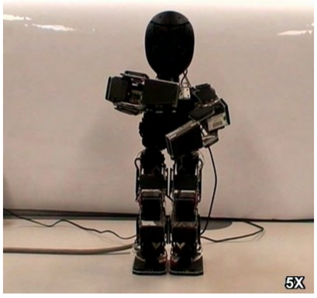
pattern applied to the robot, that, as in the previous example, can consist of multiple *instructions*.

The subjects provided an average of 867.5 *touches*, that were translated into an average of 1181.2 *instructions*, showing that the users actually exploited the superposition of several tactile *instructions* in the same *touch*.

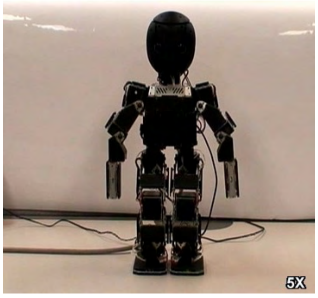
As explained before, when a tactile instruction is not understood, the user can teach the robot the meaning of such instruction for further use. In this way, the robot's mapping is enriched and refined continuously. Figures 4.15, 4.16, 4.17 and 4.18 report the ratio between the number of touch meanings taught and the number of touches provided. It evidently appears that the users need to teach less and less instructions over time because they can effectively reuse the ones already taught. The ratio between the number of touch meanings taught and the number of instructions provided clearly decreases even faster, since a single *touch* can activate multiple *instructions* previously taught. In most of the cases there is a tendency in providing the meaning of many touch instructions at the beginning and a progressive reduction in the need of teaching the meaning of the tactile patterns, as clearly appears for subject C.

In order to get insights on the touch protocol spontaneously employed by the users, we analyzed the examples of mapping between touch instructions and motor posture changes taught by each user. In particular we calculated the mutual information between each of the sensors and the rotation given for each of the motors. To compute the mutual information, we initially discretized the data. Each sensor information was set to 0 if its value was less than 20% of the maximum force measurable by the sensor and 1 otherwise. Each motor change information was set to 0 if the user moved the motor less than 5 degrees, to -1 if the user moved the motor more than 5 degrees in clockwise direction and +1 if the user moved the motor in counter clockwise direction more than 5 degrees. The threshold of 20% for the sensor was chosen empirically, observing that when users touch a sensor they commonly apply a force strongly higher than the 20% of the maximum measurable force, while noise is far below this threshold. Similarly, the threshold of 5 degrees for the motor rotations was selected observing that during the *touch meaning provision* in the great majority of the cases, intentional joint angle changes exceed 5 degrees while unintentional ones are usually contained under this threshold.

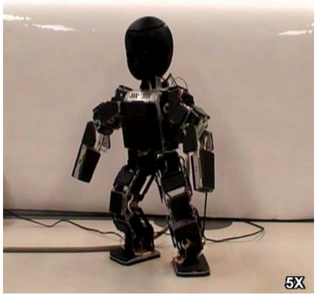
Let us denote the probability that the s-th sensor value is σ as $p_s(\sigma)$. The probability $p_s(\sigma)$ was estimated from the collected data as $p_s(\sigma) = \left|\bar{I}_i^{(s)} = \sigma\right|/E$, where E is the number of provided examples and $\left|\bar{I}_i^{(s)} = \sigma\right|$ denotes the cardinality of the set of collected examples where the sensor s assumed a value of σ. Notice that due to our discretization σ can assume only the values 0 and 1. Similarly, let us denote by $p_m(\mu)$ the probability that a touch instruction corresponds to a change in the position of the m-th motor of μ, $\mu \in \{-1, 0, 1\}$. Formally, we estimated $p_m(\mu)$ by setting $p_m(\mu) = \left|M_i^{(m)} = \sigma\right|/E$, where $M_i^{(m)}$ denotes the position change for the m-th motor provided in the i-th example.



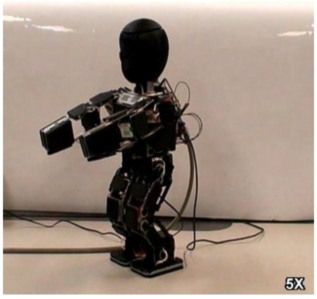
(a) t=13 s



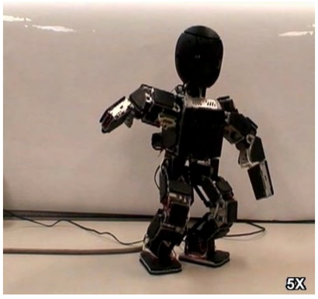
(b) t=17 s



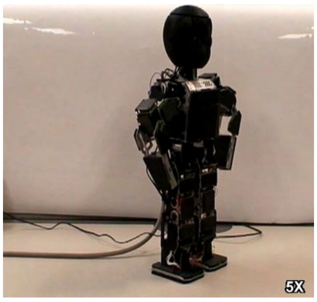
(c) t=33 s



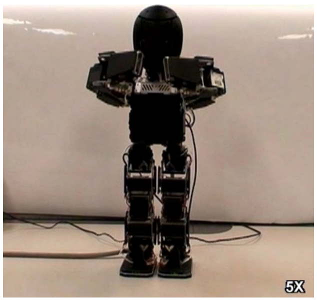
(d) t=39 s



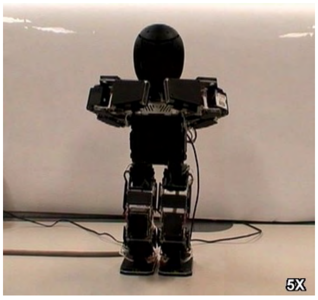
(e) t=52 s



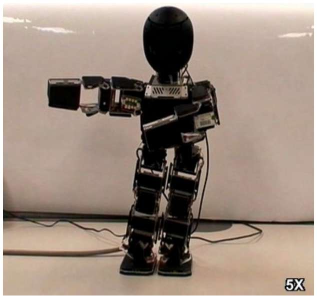
(f) t=56 s



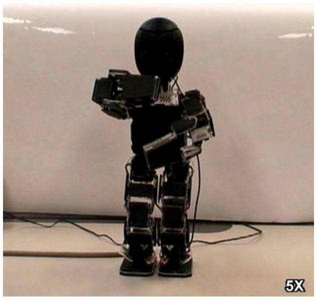
(g) t=104 s



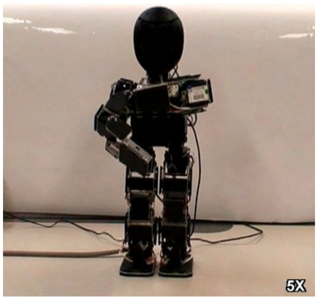
(h) t=109 s



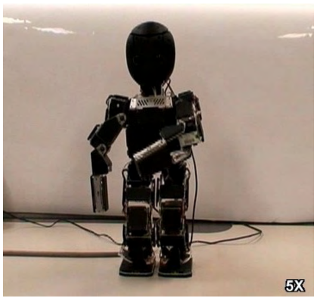
(i) t=127 s



(j) t=173 s



(k) t=175 s



(l) t=183 s

Figure 4.14: Snapshots of the robot performing the dance chosen for the experiment.

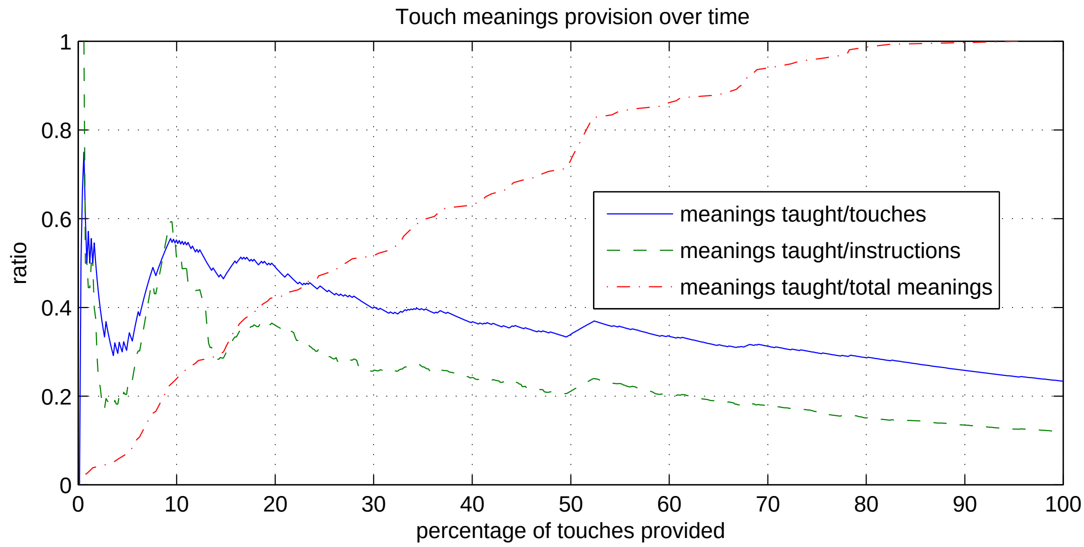


Figure 4.15: Touch meanings provision over time for subject A. The continuous line indicates the ratio between the touch meanings taught and the number of touches applied to the robot. The dashed line indicates the ratio between the number of touch meanings taught and the number of touch instructions given. The dash dot line indicates the ratio between the number of meanings taught before applying a certain touch and the total number of meanings taught.

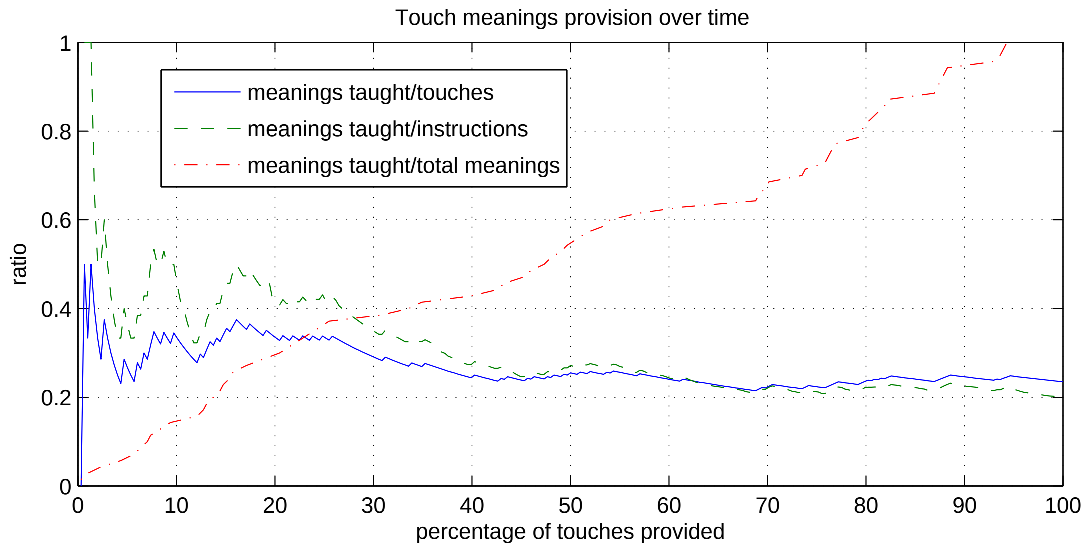


Figure 4.16: Touch meanings provision over time for subject B. The notation is the same as Fig. 4.15.

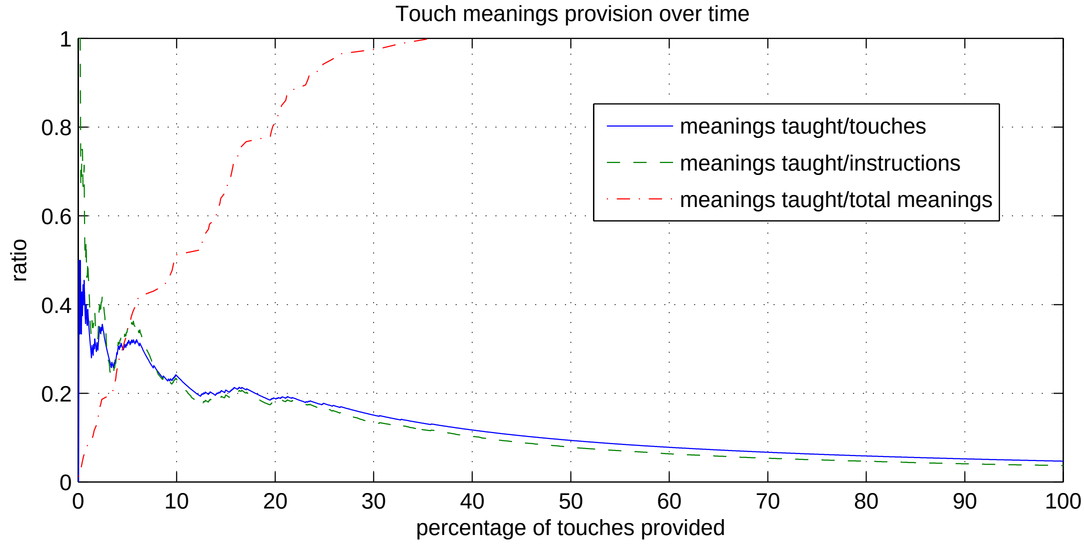


Figure 4.17: Touch meanings provision over time for subject C. The notation is the same as Fig. 4.15.

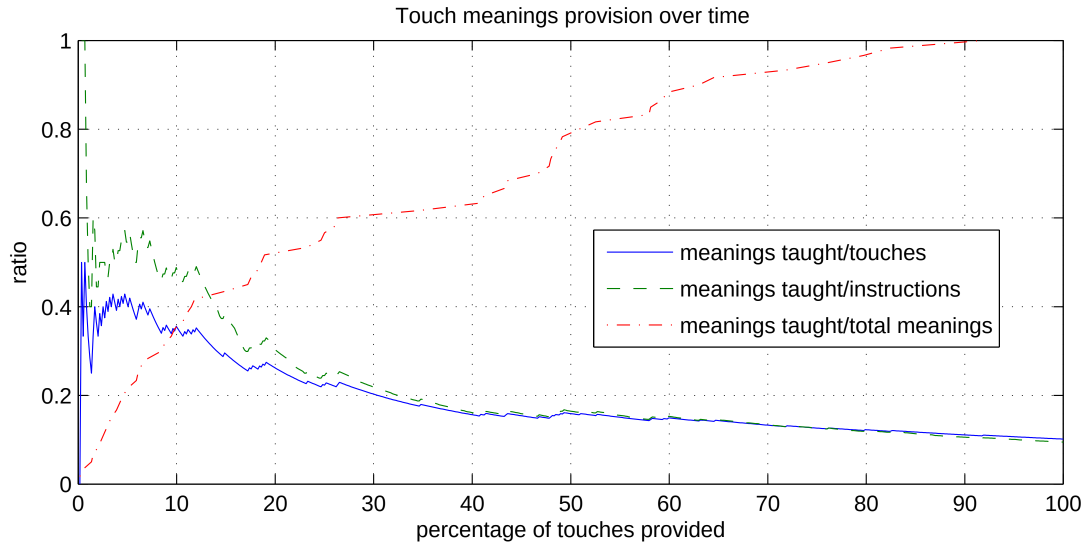


Figure 4.18: Touch meanings provision over time for subject D. The notation is the same as Fig. 4.15.

Finally, let us denote by $p_{s,m}(\sigma, \mu)$ the joint probability of the s-th sensor value being σ and the change of the m-th motor being μ for the same example. The mutual information between a sensor s, $0 \leq s \leq 89$, and the change of the position of a motor m, $0 \leq m \leq 21$, was computed as

$$\hat{I}_{s,m} = \sum_{\sigma} \sum_{\mu} p_{s,m}(\sigma, \mu) \log_2 \left(\frac{p_{s,m}(\sigma, \mu)}{p_s(\sigma) p_m(\mu)} \right)$$

The normalized mutual information $I_{s,m}$ was then computed considering the entropies [108], expressly $H_s = -\sum_{\sigma} p_s(\sigma) \log_2 p_s(\sigma)$, $H_m = -\sum_{\mu} p_m(\mu) \log_2 p_m(\mu)$ and

$$I_{s,m} = \frac{\hat{I}_{s,m}}{\sqrt{H_s H_m}}$$

Figures 4.19, 4.20, 4.21 and 4.22 illustrate the results. We notice that mainly the users touched the sensors on a limb to move motors on the same limb. However, the subjects didn't restrict themselves to a one to one correspondence between joints and sensors. Several sensors are used to actuate the same joint and conversely the same sensor actuates several joints. Furthermore, we notice that different users tend to provide different mappings, even if the experimental conditions (task, robot, etc.) are the same. This confirms the user dependence of the touch protocol, and therefore the need of using different mappings for different users.

We notice statistical dependence between the sensors on the top of the head and the leg motors for some users. Direct inspection of the touch data of subject A show that, for instance, this was done to teach the robot to squat when it is touched on the head. Similarly subject D used the sensors on the back of the robot's head to make the robot lean forwards and sensors on the front of the head to make the robot lean backwards. Even more interestingly, we can notice a statistical dependence between the sensors placed on the side of the robot's body and the corresponding leg for the first user. Direct examination of the data shows that in these cases the user employed the sensors on the side to tell the robot to rotate the corresponding leg and bring the knee outwards on that side (see figures 4.14(c) and 4.14(e)). Statistical Dependence between the sensors on the upper part of the left leg ($s00.lHipB$ and $s01.lHipF$) and motors of the right leg ($m10.rHipP$) also emerges, since often when the posture of one leg was changed the other leg was moved as well to maintain the balance.

To ease the interpretation of the data, tables 4.3, 4.4, 4.5 and 4.6 report the meaning of touch instructions provided by the users in terms of rules. In particular, assume to define each acquired example I_i and M_i in terms of a set Ξ_i of sensors that were pressed and a set Ψ_i of motors that were moved more than five degrees. A rule is defined as a couple of sets $\Sigma_j \subseteq \Xi_i$ and $\Theta_j \subseteq \Psi_i$ such that in all of the data acquired for that user, when sensors Σ_j are pressed then also all of the motors Θ_j were rotated. In other terms, we say that the set of motors is always rotated when some sensors are pushed if there is no counterexample in the database such that the same set of sensors were pushed but some of the motors that appear in the rule were not rotated.

Formally, we define the rule $\Sigma_j \to \Theta_j$ as valid if $\nexists i : \Xi_i \supseteq \Sigma_j \wedge \Psi_i \subsetneq \Theta_j$. Clearly, by the definition of a valid rule, if a rule $\Sigma_j \to \Theta_j$ is valid then any rule $\Sigma'_j \to \Theta_j$ where $\Sigma'_j \supseteq \Sigma_j$ is valid, since the "matches on the sensors" in the database will be a subset of the matches of the original rule. Similarly, if a rule $\Sigma_j \to \Theta_j$ is valid then any rule $\Sigma_j \to \Theta'_j$ where $\Theta'_j \subseteq \Theta_j$ is valid.

Intuitively, we are thus interested in the smallest set of sensors pushed for which a set of motors is always rotated. In other terms, the rules we aim to extract are the ones with the smallest sets on the left side of the arrow and the bigger sets on the right side of the arrow. Formally, we are interested in the rules $\Sigma_j \to \Theta_j$ such that $\nexists \Sigma'_j \subsetneq \Sigma_j$ with $\Sigma'_j \to \Theta_j$ valid rule and such that $\nexists \Theta'_j \supsetneq \Theta_j$ with $\Sigma_j \to \Theta'_j$ valid rule.

Furthermore, in some cases there exist a rule stating that the pressure of a set of sensors always leads to the rotation of a set of motors, and there exist another rules saying that the pressure of subsets of these sensors always leads to the rotation of subsets of the motors. In these cases, what is of interest for the first rule is the set of motors that are rotated just when all of the sensors are pressed. With the set notation, for the rule $\Sigma_j \to \Theta_j$ we are interested in the set of motors

$$\bar{\Theta}_j = \Theta_j \setminus \bigcup_{k:\Sigma_k \subsetneq \Sigma_j \wedge \Sigma_k \to \Theta_k \text{is a valid rule}} \Theta_k$$

To write the rules in a more compact form, each row in the tables groups the rules having the same set Θ of motors moved. The first column indicates the sensors that must be pressed, using a notation close to the propositional logic. For instance, with such notation the sets $\{\alpha, \beta\gamma\}$ and $\{\alpha, \beta\delta\}$ are summarized using the expression $\alpha \wedge \beta \wedge (\gamma \vee \delta)$. For the motors and sensors both their numerical ID and a shorthand name is reported. Refer to figures 4.4 and 4.7 for their actual location on the robot.

The second column indicates the motors rotated. In particular, when subsets of the motors rotated can be predicted by other rules that use a subset of the sensors, then the set $\bigcup_{k:\Sigma_k \subsetneq \Sigma_j \wedge \Sigma_k \to \Theta_k \text{is a valid rule}} \Theta_k$ of motors that can be predicted by other rules are reported in parentheses.

The last column reports the total number of N_j examples in the dataset for which a superset of the sensors Σ_j indicated by the rule was pushed and a superset of the motors Θ_j indicated by the rule was rotated. Intuitively, this figure, corresponding to the support of association rules [109], indicates the statistical significance of the concept expressed by the rule.

Table 4.3: Rules for subject A

Sensors	Motors	N_j
^{34}rFpalm$\wedge^{09}$lLeglowerL $\wedge$ (^{28}rFpalm $\vee$ ^{30}rFpalm)	^{09}lHipY	9
^{32}rFpalm$\wedge^{07}$lLeglowerR	^{19}lAnkleP	7
^{84}HeadtopC $\wedge$ (^{28}rFpalm $\vee$ ^{30}rFpalm $\vee$ ^{32}rFpalm)	^{14}rKneeUP ^{15}lKneeUP ^{19}lAnkleP	5
^{40}lArmupperB $\wedge$ (^{28}rFpalm $\vee$ ^{30}rFpalm $\vee$ ^{32}rFpalm $\vee$ ^{34}rFpalm $\vee$ ^{12}lFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{07}lLeglowerR $\wedge$ (^{28}rFpalm $\wedge$ (^{14}lFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{30}rFpalm $\wedge$ (^{14}lFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{12}lFpalm) $\vee$ ^{20}rLegupperL $\wedge$ (^{14}lFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{51}rArmupperB $\vee$ ^{45}lHandpalmB	^{02}rShoulderP	5
^{84}HeadtopC$\wedge^{12}$lFpalm	^{14}rKneeUP ^{16}rKneeLP ^{13}lHipP ^{15}lKneeUP ^{17}lKneeLP ^{19}lAnkleP	4
^{34}rFpalm$\wedge^{07}$lLeglowerR	^{09}lHipY ^{15}lKneeUP ^{17}lKneeLP ^{19}lAnkleP	4
^{86}HeadtopR $\wedge$ (^{28}rFpalm $\vee$ ^{30}rFpalm $\vee$ ^{32}rFpalm $\vee$ ^{14}lFpalm) $\vee$ ^{84}HeadtopC$\wedge^{14}$lFpalm	^{14}rKneeUP ^{15}lKneeUP ^{19}lAnkleP	4
^{42}lHandtopT $\vee$ ^{09}lLeglowerL$\wedge^{17}$lFpalm	^{05}lShoulderR ^{07}lElbowR	4
^{24}rLeglowerR $\wedge$ (^{28}rFpalm$\wedge^{12}$lFpalm $\vee$ ^{10}lFpalm$\wedge^{14}$lFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{20}rLegupperL$\wedge^{28}$rFpalm $\vee$ ^{23}rKnee$\wedge^{02}$lKnee	^{20}rAnkleR	4
^{86}HeadtopR $\wedge$ (^{10}lFpalm $\vee$ ^{12}lFpalm) $\vee$ ^{84}HeadtopC$\wedge^{10}$lFpalm	^{14}rKneeUP ^{16}rKneeLP ^{13}lHipP ^{15}lKneeUP ^{17}lKneeLP ^{19}lAnkleP	3
^{20}rLegupperL $\wedge$ (^{32}rFpalm $\vee$ ^{34}rFpalm) $\vee$ ^{24}rLeglowerR$\wedge^{32}$rFpalm $\vee$ ^{07}lLeglowerR$\wedge^{09}$lLeglowerL	^{10}rHipR ^{14}rKneeUP ^{16}rKneeLP ^{20}rAnkleR	3
^{20}rLegupperL $\wedge$ (^{10}lFpalm $\vee$ ^{12}lFpalm)	^{14}rKneeUP ^{16}rKneeLP ^{20}rAnkleR	3
^{28}rFpalm $\wedge$ (^{20}rLegupperL $\wedge$ (^{14}lFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{22}rLegupperR$\wedge^{03}$lLegupperL $\vee$ ^{09}lLeglowerL$\wedge^{14}$lFpalm) $\vee$ ^{14}lFpalm $\wedge$ (^{24}rLeglowerR$\wedge^{12}$lFpalm $\vee$ ^{30}rFpalm$\wedge^{09}$lLeglowerL) $\vee$ ^{46}lHandpalmT	^{05}lShoulderR ^{07}lElbowR	3

^{07}lLeglowerR $\wedge$ (^{10}lFpalm $\wedge$ (^{14}lFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{61}BackupperL $\vee$ ^{62}BacklowerL $\vee$ ^{66}BacklowerR $\vee$ ^{67}BackupperR $\vee$ ^{26}rLeglowerL $\vee$ ^{15}lFoottopR) $\vee$ ^{15}lFoottopR $\wedge$ (^{61}BackupperL $\vee$ ^{66}BacklowerR $\vee$ ^{67}BackupperR $\vee$ ^{26}rLeglowerL $\vee$ ^{10}lFpalm $\vee$ ^{12}lFpalm) $\vee$ ^{24}rLeglowerR $\wedge$ (^{17}lFpalm $\wedge$ (^{30}rFpalm $\vee$ ^{10}lFpalm $\vee$ ^{12}lFpalm) $\vee$ ^{34}rFpalm) $\vee$ ^{26}rLeglowerL $\wedge$ (^{61}BackupperL $\vee$ ^{62}BacklowerL $\vee$ ^{66}BacklowerR $\vee$ ^{67}BackupperR) $\vee$ ^{30}rFpalm $\wedge$ (^{20}rLegupperL $\vee$ ^{23}rKnee$\wedge$^{34}rFpalm $\vee$ ^{02}lKnee$\wedge$^{10}lFpalm) $\vee$ ^{40}lArmupperB $\wedge$ (^{10}lFpalm $\vee$ ^{14}lFpalm) $\vee$ ^{38}lArmupperF $\vee$ ^{41}lHandtopB$\wedge$^{45}lHandpalmB $\vee$ ^{28}rFpalm$\wedge$^{02}lKnee$\wedge$^{10}lFpalm	^{19}lAnkleP	3
^{66}BacklowerR$\wedge$^{02}lKnee	^{08}rHipY ^{10}rHipR ^{18}rAnkleP ^{20}rAnkleR ^{09}lHipY ^{13}lHipP ^{15}lKneeUP	2
^{73}lRib $\vee$ ^{22}rLegupperR$\wedge$^{30}rFpalm$\wedge$^{03}lLegupperL $\vee$ ^{31}rAnkleB	^{09}lHipY ^{13}lHipP ^{15}lKneeUP ^{17}lKneeLP	2
^{23}rKnee$\wedge$^{02}lKnee $\wedge$ (^{28}rFpalm $\vee$ ^{30}rFpalm $\vee$ ^{10}lFpalm) $\vee$ ^{20}rLegupperL$\wedge$^{30}rFpalm $\wedge$ (^{14}lFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{24}rLeglowerR$\wedge$^{09}lLeglowerL	^{08}rHipY ^{10}rHipR ^{16}rKneeLP (^{14}rKneeUP ^{20}rAnkleR)	2
^{62}BacklowerL $\wedge$ (^{52}rHandtopB $\vee$ ^{53}rHandtopT $\vee$ ^{56}rHandpalmB $\vee$ ^{57}rHandpalmT) $\vee$ ^{22}rLegupperR$\wedge$^{03}lLegupperL $\wedge$ (^{32}rFpalm $\vee$ ^{34}rFpalm $\vee$ ^{14}lFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{28}rFpalm $\wedge$ (^{20}rLegupperL $\wedge$ (^{10}lFpalm $\vee$ ^{12}lFpalm) $\vee$ ^{24}rLeglowerR$\wedge$^{17}lFpalm) $\vee$ ^{23}rKnee$\wedge$^{02}lKnee $\wedge$ (^{34}rFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{49}rArmupperF	^{02}rShoulderP ^{06}rElbowR	2

^{34}rFpalm ∧ (^{23}rKnee ∧ (^{30}rFpalm ∧ (^{28}rFpalm ∨ ^{32}rFpalm ∨ ^{10}lFpalm ∨ ^{12}lFpalm) ∨ ^{14}lFpalm) ∨ ^{20}rLegupperL ∧ (^{30}rFpalm ∨ ^{10}lFpalm ∨ ^{12}lFpalm) ∨ ^{24}rLeglowerR∧^{14}lFpalm ∨ ^{07}lLeglowerR∧^{12}lFpalm) ∨ ^{45}lHandpalmB ∧ (^{41}lHandtopB ∧ (^{10}lFpalm ∨ ^{14}lFpalm) ∨ ^{42}lHandtopT ∧ (^{10}lFpalm ∨ ^{14}lFpalm)) ∨ ^{20}rLegupperL∧^{32}rFpalm ∧ (^{30}rFpalm ∨ ^{10}lFpalm ∨ ^{12}lFpalm) ∨ ^{46}lHandpalmT ∧ (^{10}lFpalm ∨ ^{14}lFpalm) ∨ ^{23}rKnee ∧ (^{61}BackupperL ∨ ^{62}BacklowerL) ∨ ^{66}BacklowerR∧^{51}rArmupperB ∨ ^{33}rFoottopR ∨ ^{07}lLeglowerR∧^{09}lLeglowerL∧^{12}lFpalm	^{12}rHipP	2

Table 4.4: Rules for subject B

Sensors	Motors	N_j
^{22}rLegupperR	^{10}rHipR ^{12}rHipP ^{20}rAnkleR	5
^{22}rLegupperR∧^{14}lFpalm	^{16}rKneeLP (^{10}rHipR ^{12}rHipP ^{20}rAnkleR)	4
^{20}rLegupperL	^{10}rHipR ^{20}rAnkleR	3
^{62}BacklowerL∧^{54}rArmlowerF	^{02}rShoulderP	3
^{24}rLeglowerR	^{16}rKneeLP ^{20}rAnkleR	2
^{41}lHandtopB ∧ (^{30}rFpalm ∨ ^{34}rFpalm ∨ ^{10}lFpalm ∨ ^{12}lFpalm ∨ ^{14}lFpalm) ∨ ^{20}rLegupperL ∧ (^{32}rFpalm ∨ ^{34}rFpalm ∨ ^{14}lFpalm) ∨ ^{51}rArmupperB ∨ ^{19}rHipB ∨ ^{00}lHipB ∨ ^{03}lLegupperL	^{02}rShoulderP	2

Table 4.5: Rules for subject C

Sensors	Motors	N_j
^{41}lHandtopB ∨ ^{42}lHandtopT	^{07}lElbowR	6
^{23}rKnee	^{12}rHipP ^{14}rKneeUP	4
^{25}rLeglowerF	^{14}rKneeUP ^{16}rKneeLP	3
^{21}rLegupperF	^{12}rHipP	3
^{62}BacklowerL∧^{23}rKnee	^{16}rKneeLP ^{18}rAnkleP (^{12}rHipP ^{14}rKneeUP)	2

^{66}BacklowerR $\wedge$ (^{32}rFpalm $\vee$ ^{34}rFpalm $\vee$ ^{14}lFpalm) $\vee$ ^{28}rFpalm $\wedge$ (^{32}rFpalm $\vee$ ^{10}lFpalm $\vee$ ^{14}lFpalm) $\vee$ ^{12}lFpalm $\wedge$ (^{32}rFpalm $\vee$ ^{34}rFpalm $\vee$ ^{14}lFpalm) $\vee$ ^{50}rElbow $\vee$ ^{51}rArmupperB $\vee$ ^{52}rHandtopB $\vee$ ^{53}rHandtopT $\vee$ ^{56}rHandpalmB $\vee$ ^{57}rHandpalmT $\vee$ ^{37}lShoulderL $\vee$ ^{38}lArmupperF $\vee$ ^{40}lArmupperB $\vee$ ^{45}lHandpalmB $\vee$ ^{46}lHandpalmT $\vee$ ^{18}rHipF $\vee$ ^{20}rLegupperL $\vee$ ^{22}rLegupperR $\vee$ ^{30}rFpalm $\vee$ ^{03}lLegupperL $\vee$ ^{06}lAnkleF$\wedge$^{16}lAnkleB $\vee$ ^{07}lLeglowerR$\wedge$^{09}lLeglowerL $\vee$ ^{17}lFpalm	^{04}rShoulderR	2

Table 4.6: Rules for subject D

Sensors	Motors	N_j
^{43}lArmlowerF $\wedge$ (^{14}lFpalm $\wedge$ (^{28}rFpalm $\vee$ ^{30}rFpalm $\vee$ ^{12}lFpalm) $\vee$ ^{17}lFpalm $\wedge$ (^{28}rFpalm $\vee$ ^{30}rFpalm $\vee$ ^{12}lFpalm)) $\vee$ ^{54}rArmlowerF$\wedge$^{17}lFpalm $\wedge$ (^{28}rFpalm $\vee$ ^{30}rFpalm)	^{04}rShoulderR ^{06}rElbowR	4
^{44}lArmlowerB $\wedge$ (^{14}lFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{63}rSideF $\vee$ ^{47}rShoulderT	^{21}lAnkleR	4
^{83}HeadtopL $\vee$ ^{84}HeadtopC $\vee$ ^{86}HeadtopR	^{12}rHipP ^{14}rKneeUP ^{16}rKneeLP ^{13}lHipP ^{17}lKneeLP	3
^{59}lSideC	^{10}rHipR ^{20}rAnkleR ^{11}lHipR ^{21}lAnkleR	3
^{63}rSideF$\wedge$^{64}rSideC	^{10}rHipR ^{20}rAnkleR ^{11}lHipR (^{21}lAnkleR)	3
^{54}rArmlowerF $\wedge$ (^{14}lFpalm $\wedge$ (^{28}rFpalm $\vee$ ^{30}rFpalm) $\vee$ ^{17}lFpalm $\wedge$ (^{10}lFpalm $\vee$ ^{12}lFpalm)) $\vee$ ^{10}lFpalm $\wedge$ (^{43}lArmlowerF $\wedge$ (^{14}lFpalm $\vee$ ^{17}lFpalm) $\vee$ ^{62}BacklowerL$\wedge$^{23}rKnee) $\vee$ ^{81}lJaw $\vee$ ^{87}lEar	^{14}rKneeUP ^{16}rKneeLP	3

^{51}rArmupperB ∧ (^{28}rFpalm ∨ ^{30}rFpalm ∨ ^{32}rFpalm ∨ ^{34}rFpalm ∨ ^{14}lFpalm ∨ ^{17}lFpalm) ∨ ^{34}rFpalm ∧ (^{49}rArmupperF ∨ ^{43}lArmlowerF∧^{10}lFpalm ∨ ^{08}lLeglowerF) ∨ ^{32}rFpalm ∧ (^{43}lArmlowerF∧^{10}lFpalm ∨ ^{08}lLeglowerF) ∨ ^{62}BacklowerL∧^{55}rArmlowerB∧^{17}lFpalm ∨ ^{49}rArmupperF∧^{30}rFpalm ∨ ^{36}lShoulderT ∨ ^{37}lShoulderL ∨ ^{40}lArmupperB ∨ ^{18}rHipF ∨ ^{19}rHipB ∨ ^{16}lAnkleB	^{06}rElbowR	3
^{77}ChinL ∨ ^{78}ChinR	^{12}rHipP ^{14}rKneeUP ^{16}rKneeLP ^{13}lHipP ^{15}lKneeUP ^{17}lKneeLP	2
^{79}rTemple ∨ ^{80}rJaw ∨ ^{82}lTemple ∨ ^{89}rEar	^{10}rHipR ^{16}rKneeLP ^{13}lHipP ^{15}lKneeUP ^{19}lAnkleP	2
^{02}lKnee ∧ (^{62}BacklowerL ∧ (^{23}rKnee ∨ ^{28}rFpalm ∨ ^{30}rFpalm ∨ ^{10}lFpalm ∨ ^{12}lFpalm ∨ ^{14}lFpalm ∨ ^{17}lFpalm) ∨ ^{34}rFpalm ∧ (^{23}rKnee ∨ ^{28}rFpalm ∨ ^{30}rFpalm ∨ ^{10}lFpalm ∨ ^{12}lFpalm ∨ ^{14}lFpalm ∨ ^{17}lFpalm)) ∨ ^{60}lSideF ∨ ^{23}rKnee∧^{34}rFpalm∧^{10}lFpalm	^{16}rKneeLP ^{20}rAnkleR ^{11}lHipR ^{21}lAnkleR	2
^{63}rSideF ∧ (^{62}BacklowerL ∨ ^{30}rFpalm ∨ ^{32}rFpalm) ∨ ^{81}lJaw ∧ (^{10}lFpalm ∨ ^{14}lFpalm) ∨ ^{87}lEar ∧ (^{10}lFpalm ∨ ^{14}lFpalm) ∨ ^{44}lArmlowerB ∧ (^{54}rArmlowerF ∨ ^{43}lArmlowerF)	^{10}rHipR ^{20}rAnkleR ^{11}lHipR (^{21}lAnkleR)	2
^{19}rHipB ∧ (^{30}rFpalm ∨ ^{34}rFpalm ∨ ^{10}lFpalm ∨ ^{12}lFpalm ∨ ^{14}lFpalm ∨ ^{17}lFpalm) ∨ ^{18}rHipF ∧ (^{10}lFpalm ∨ ^{12}lFpalm ∨ ^{14}lFpalm ∨ ^{17}lFpalm) ∨ ^{62}BacklowerL ∧ (^{43}lArmlowerF ∧ (^{14}lFpalm ∨ ^{17}lFpalm) ∨ ^{54}rArmlowerF∧^{17}lFpalm) ∨ ^{16}lAnkleB ∧ (^{28}rFpalm ∨ ^{30}rFpalm ∨ ^{34}rFpalm) ∨ ^{54}rArmlowerF∧^{14}lFpalm ∧ (^{10}lFpalm ∨ ^{12}lFpalm) ∨ ^{08}lLeglowerF ∧ (^{25}rLeglowerF ∨ ^{30}rFpalm) ∨ ^{31}rAnkleB ∨ ^{09}lLeglowerL	^{04}rShoulderR ^{06}rElbowR	2

^{08}lLeglowerF ∧ (^{32}rFpalm ∧ (^{62}BacklowerL ∨ ^{10}lFpalm ∨ ^{12}lFpalm ∨ ^{14}lFpalm ∨ ^{17}lFpalm) ∨ ^{34}rFpalm ∧ (^{62}BacklowerL ∨ ^{10}lFpalm ∨ ^{12}lFpalm ∨ ^{14}lFpalm ∨ ^{17}lFpalm)) ∨ ^{62}BacklowerL ∧ (^{44}lArmlowerB ∧ (^{55}rArmlowerB ∨ ^{14}lFpalm ∨ ^{17}lFpalm) ∨ ^{47}rShoulderT ∨ ^{54}rArmlowerF∧^{43}lArmlowerF ∨ ^{55}rArmlowerB∧^{14}lFpalm ∨ ^{36}lShoulderT) ∨ ^{54}rArmlowerF∧^{43}lArmlowerF ∧ (^{28}rFpalm ∨ ^{30}rFpalm ∨ ^{10}lFpalm ∨ ^{12}lFpalm) ∨ ^{51}rArmupperB ∧ (^{40}lArmupperB ∨ ^{10}lFpalm ∨ ^{12}lFpalm) ∨ ^{49}rArmupperF ∧ (^{10}lFpalm ∨ ^{14}lFpalm) ∨ ^{38}lArmupperF ∧ (^{30}rFpalm ∨ ^{34}rFpalm) ∨ ^{85}Headback ∨ ^{48}rShoulderR ∨ ^{55}rArmlowerB∧^{44}lArmlowerB∧^{32}rFpalm ∨ ^{01}lHipF ∨ ^{03}lLegupperL	^{03}lShoulderP (^{02}rShoulderP)	2

The collected data show that multiple motors are usually moved with a single touch instructions. It is hence interesting to observe if there are consistencies in the relationship between the motor changes of different motors in the set of modifications provided as meaning of touch instructions. Computation of the mutual information between couples of motors yields the results reported in figures 4.23(a), 4.23(b), 4.24(a) and 4.24(b). We notice a very strong statistical dependence between motors that belong to the same limb. Statistical dependence between the two legs is also observable, in particular for the fourth user (Fig. 4.24(b)), or for the first user (Fig. 4.23(a)), between the pitch joints of the hips ($m10.rHipP$ and $m17.lHipP$) and of the ankles ($m13.rAnkleP$ and $m20.lAnkleP$). As previously reported, these statistical dependencies may be explained by the desire of the test subjects to keep the robot balanced over the whole experiment.

These high statistical dependencies suggest that the motor changes given by the user could actually be located in a low-dimensional manifold of the whole 22-dimensional motor space. This fact could be exploited in the estimation of meaning of touch instructions. For simplicity we focused on linear subspaces as possible manifolds. In particular, we analyzed how well the motor change relative to the e-th example fits in a subspace of dimension q constructed from the previous $e-1$ examples. More formally, we took the motor changes specified in the first $e-1$ examples $M_1 \dots M_{e-1}$. We subtracted the mean, and applied Principal Component Analysis (PCA). We then considered the e-th example motor change M_e. We subtracted the same mean, and projected the resultant vector on the subspace defined by the first q principal components $v_1 \dots v_q$. Finally, we

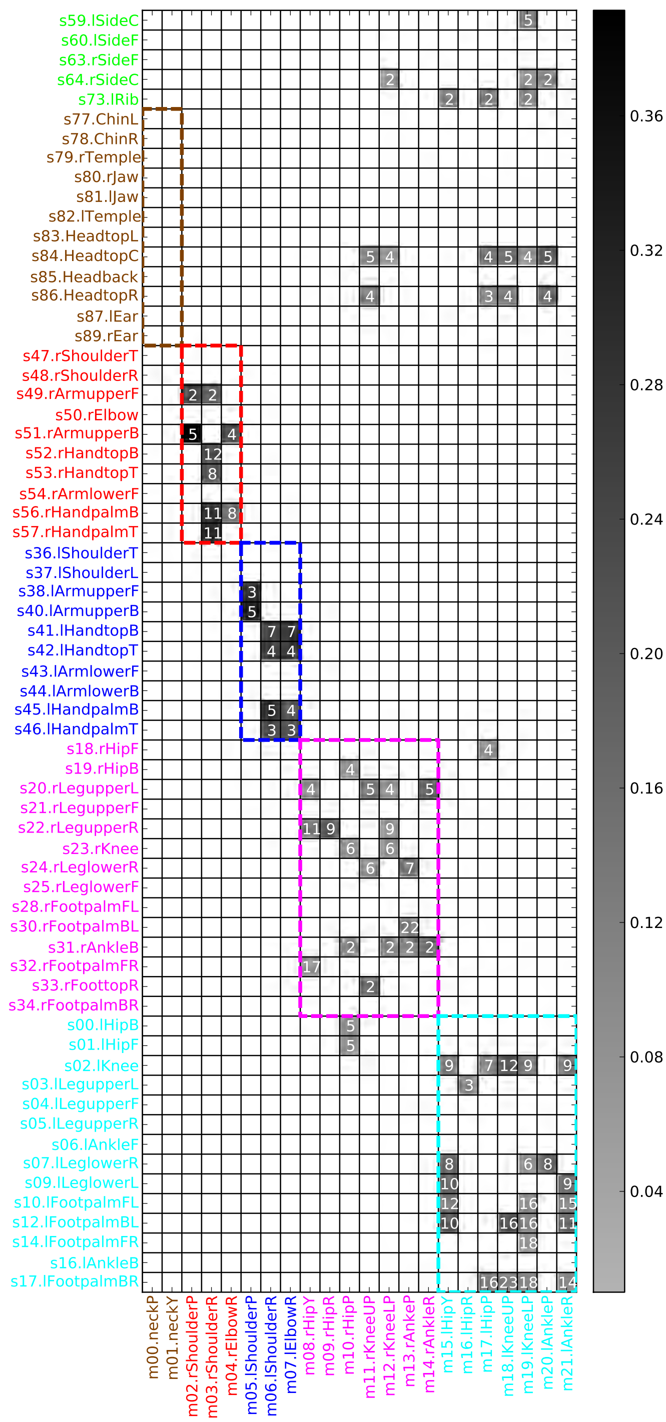


Figure 4.19: Mutual information between sensors (rows) and motors (columns) for subject A. The color of the intersection indicates the normalized mutual information value. Only mutual information values higher than 0.01 are indicated, and only sensors that have mutual information value higher than 0.01 with at least one motor are reported. The number inside each cell indicates the number of examples for which the corresponding sensor was pressed and the corresponding motor was moved. The colors of the labels indicate the robot parts. Sensors and motors of the same robot part are highlighted by a dashed rectangle.

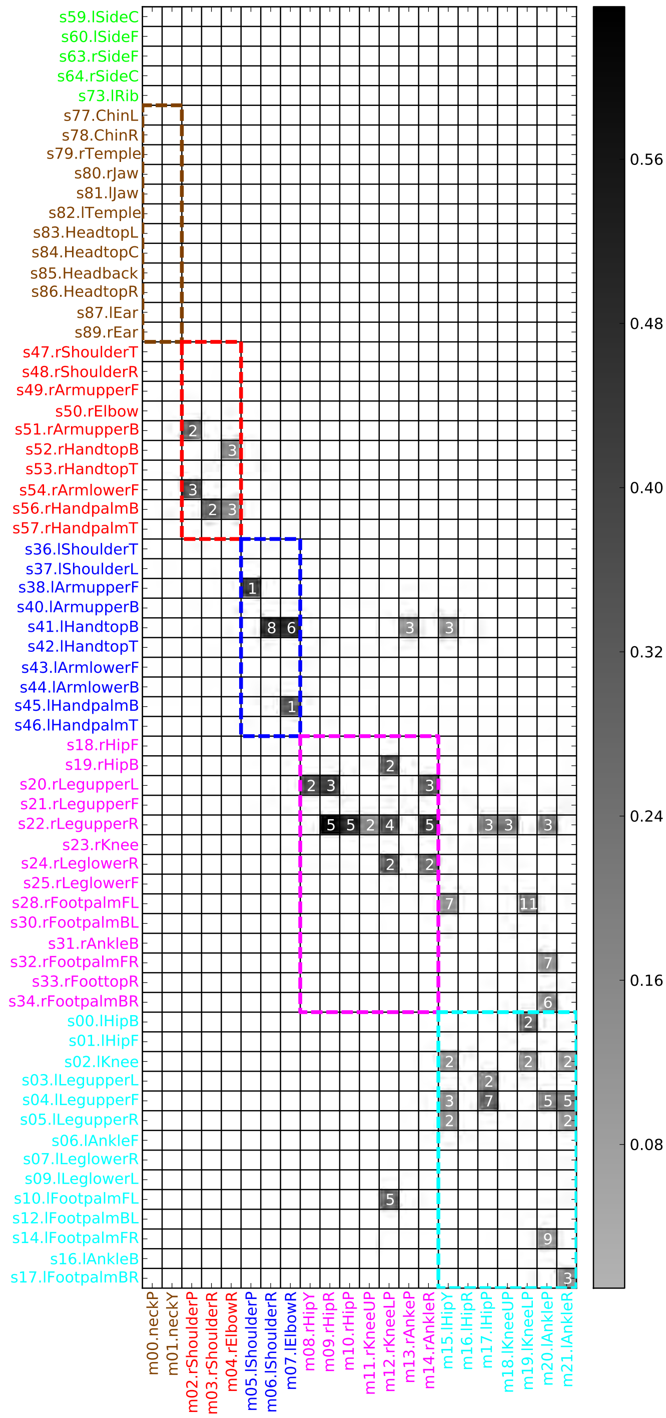


Figure 4.20: Mutual information between sensors (rows) and motors (columns) for subject B. The notation is the same as Fig. 4.19.

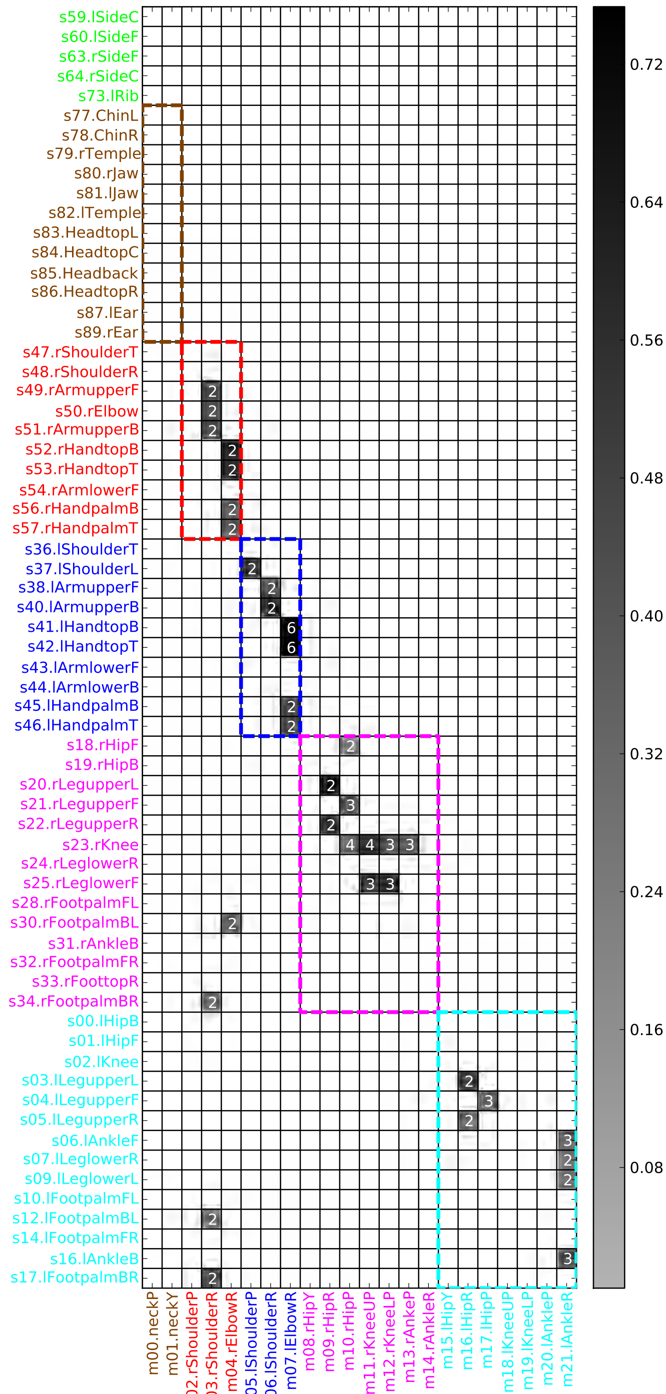


Figure 4.21: Mutual information between sensors (rows) and motors (columns) for subject C. The notation is the same as Fig. 4.19.

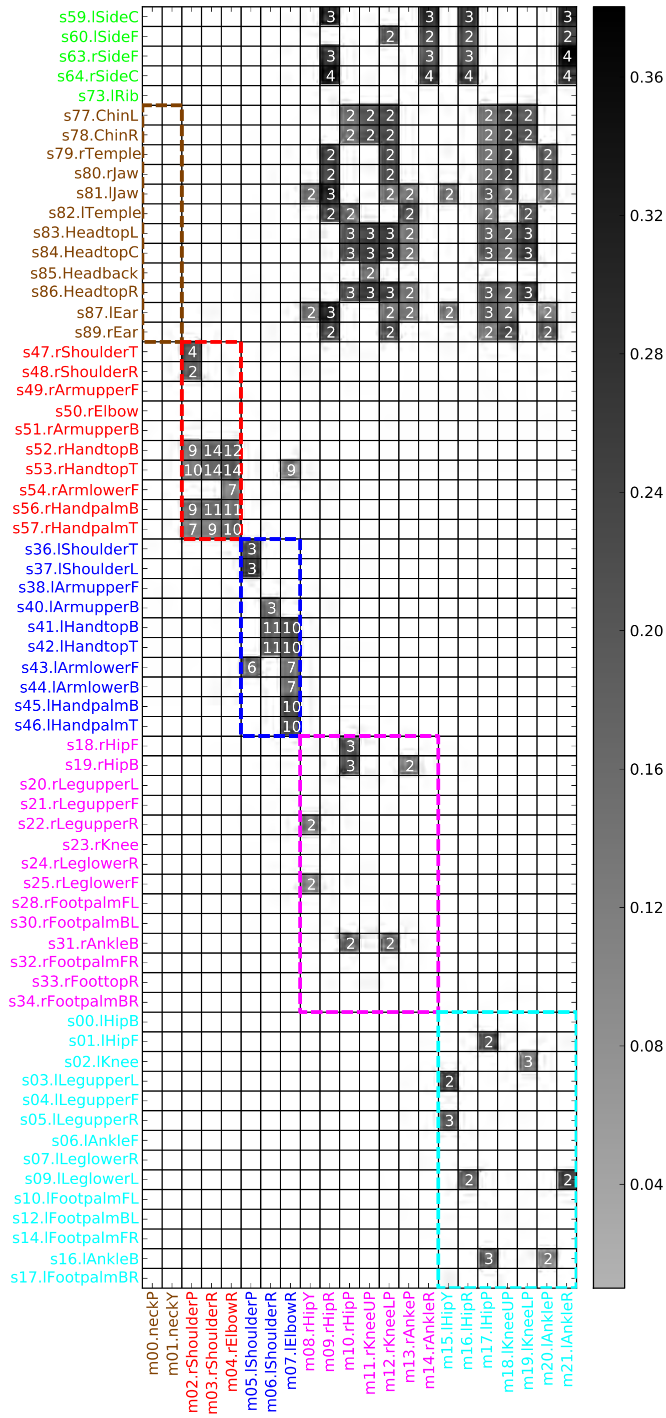
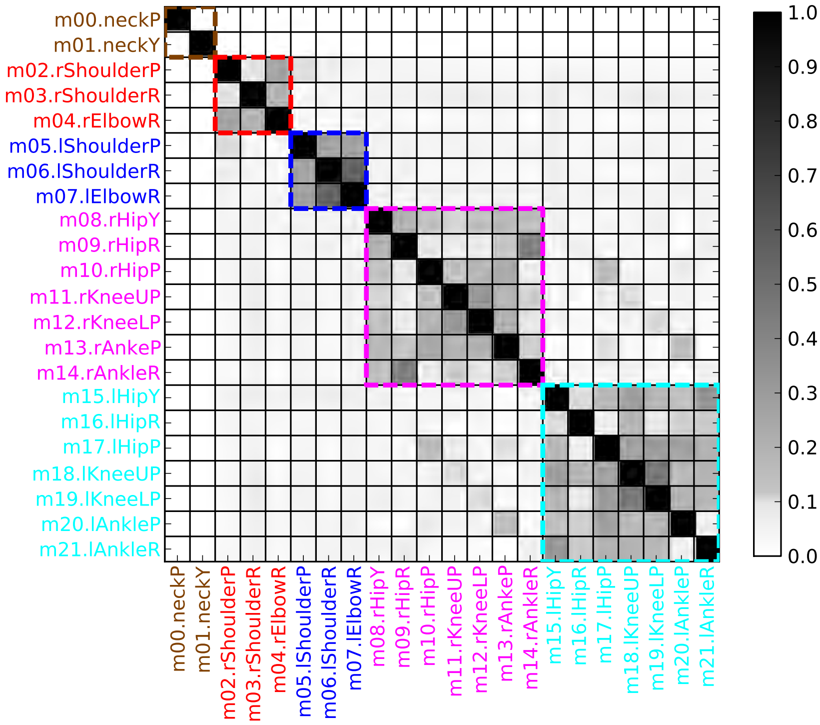
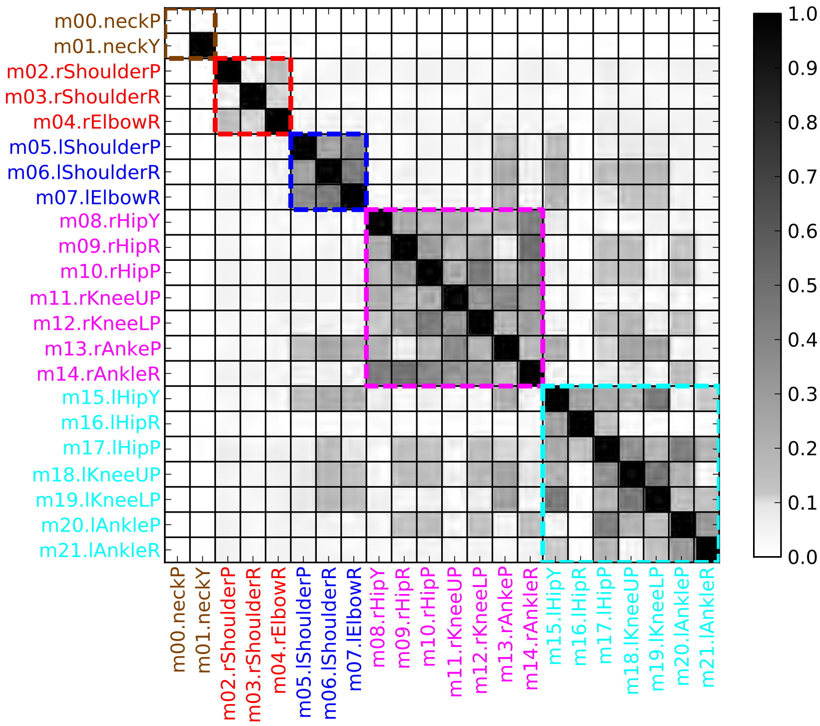


Figure 4.22: Mutual information between sensors (rows) and motors (columns) for subject D. The notation is the same as Fig. 4.19.

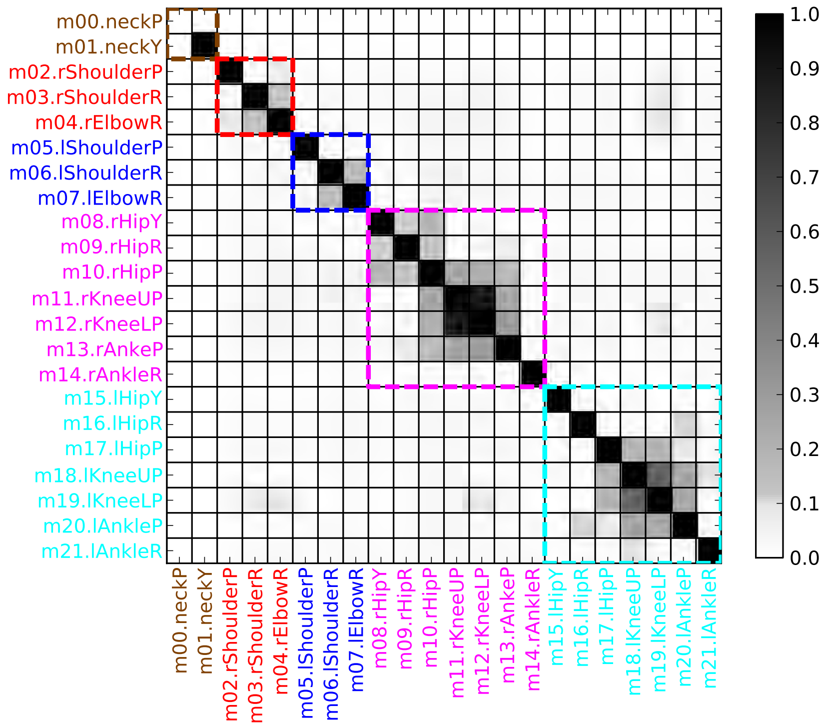


(a) Subject A

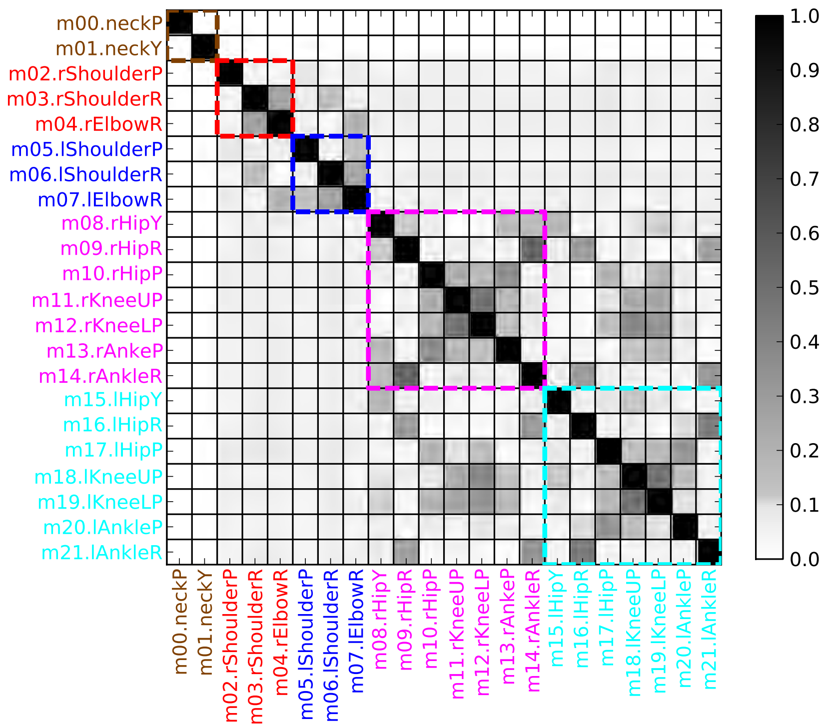


(b) Subject B

Figure 4.23: Mutual information between couples of motors for subjects A and B. Each label is given by a numerical id followed by a shorthand name separated by a dot. The color of the intersection indicates the normalized mutual information value. The areas of the map that correspond to motors of the same robot part (e.g. same limb) are denoted by a dashed square.



(a) Subject C



(b) Subject D

Figure 4.24: Mutual information between couples of motors for subjects C and D. The notation is the same as Fig. 4.23.

calculated the infinity norm of the reconstruction error[6]:

$$\epsilon_q(e) = \left\| M_e - \sum_{i=1}^{q} M_e^T v_i v_i \right\|_\infty$$

Figures 4.25, 4.26, 4.27 and 4.28 report the reconstruction error for different settings of q, $1 \leq q \leq 22$, averaged over all of the examples $e = 1 \ldots E$. For comparison, the error obtained by applying PCA on the whole set of examples, i.e. $M_1 \ldots M_E$ is also reported. We notice that the difference between the reconstruction error obtained using just the first $e-1$ examples is not much higher than the one obtained using the whole data set $M_1 \ldots M_E$.

As briefly stated in the introduction, it is a well known fact that for many tasks, the complete movement of the robot lies on a small subspace [31]. It is therefore interesting to analyze whether the subspace where motions can be projected with little errors is related to the motor changes desired as response of touch instructions. For each of the E examples provided by the user, we collected all the postures that the user brought the robot to before teaching M_e. For each of these sets of postures we then subtracted the mean and applied PCA to determine the principal components $\bar{v}_1 \ldots \bar{v}_q$. The average reconstruction error norm obtained by projecting the touch examples M_e on the subspace of dimension q defined by $\bar{v}_1 \ldots \bar{v}_q$ is also reported in figures 4.25 to 4.28 . We notice that except for low values of q ($q < 5$) the reconstruction error is comparable to the one of the projection on the subspaces constructed using the motor change information. These are preliminary results, and intensive verification must be made. However, it appears that for all of the four subjects the posture modification expected by the users in response to touch instruction could be limited to movements that lie in the subspace defined by the motion the subjects want to develop.

In other terms, the users were requested to develop a motion, "Algorithm exercise", and realized it by defining a set of postures that lie in a certain subspace of the motor space. When setting these postures that realize the task, instead of setting them in a completely free manner using motor changes in the whole motor space, they restricted their instructions to movements similar to the ones that compose the target motion.

This fact could be exploited to improve the touch interpretation. Expressly, knowledge of the frames set by the users during the teaching could be used to constrain the output of the mapping from touch instruction to motor movements.

Actually, this restriction in the movements used for teaching could partially be a consequence of the task choice. In particular, it could be caused by the fact that for dances, gestures are often more important than the actual postures taken. Conversely,

[6]We decided to employ the infinity norm because we are interested in the maximum reconstruction error over all of the joints. If the Euclidean norm were used, the information on the maximum reconstruction error would be lost. In fact, $\epsilon_q(e)$ could be small because the error is small for most of the joints and is high for very few of them.

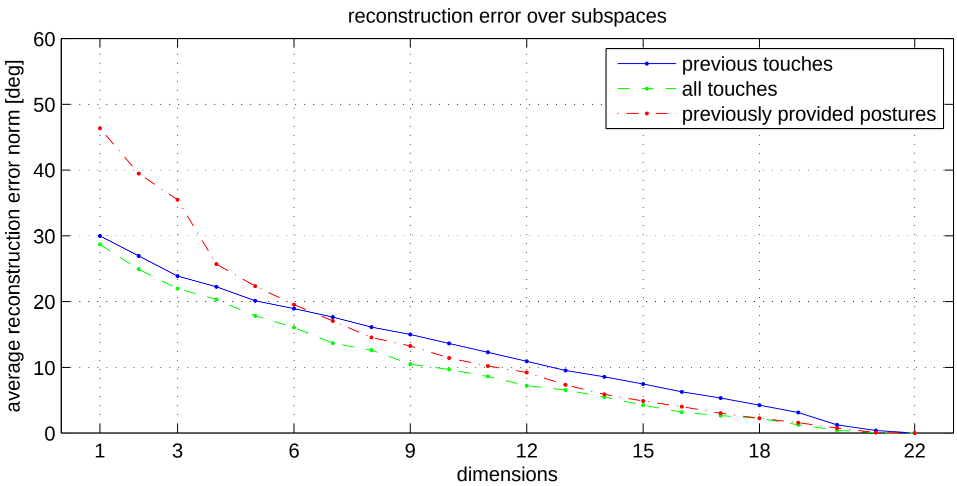


Figure 4.25: Reconstruction error for different number of dimensions of the subspace for subject A.

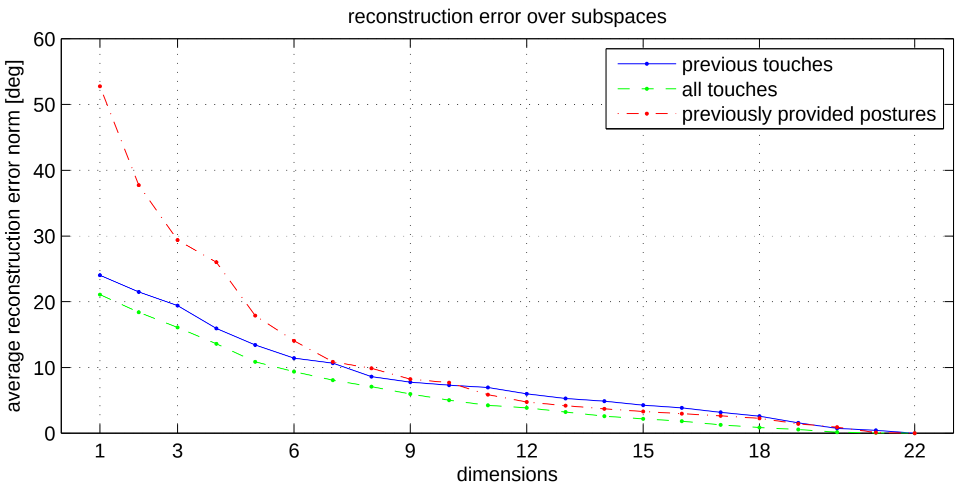


Figure 4.26: Reconstruction error for different number of dimensions of the subspace for subject B.

imagine a robot task to consist of grasping an object and rising it as high as possible. We can imagine the robot motion to essentially lie on a subspace that makes the robot arms move vertically. However, users would probably concentrate their instructions in adjusting the hands distance in order to achieve an adequate grasp, and therefore it would be likely that they provide a high number of instructions in a subspace orthogonal to the one of the motion. Future works will need to include the analysis of data from different types of tasks in order to verify these hypotheses.

Summarizing, all of the four subjects were able to develop the motion without problems, and could use the teaching by touching system without any training. Further tests will need to validate the approach with completely naive users, that are not even familiar with the robot. Data analysis showed that specific algorithms for interpretation of tactile

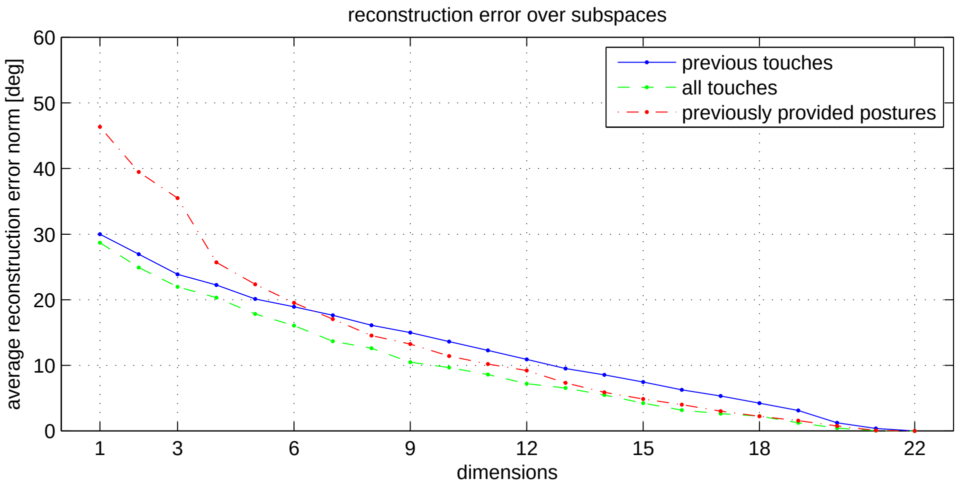


Figure 4.27: Reconstruction error for different number of dimensions of the subspace for subject C.

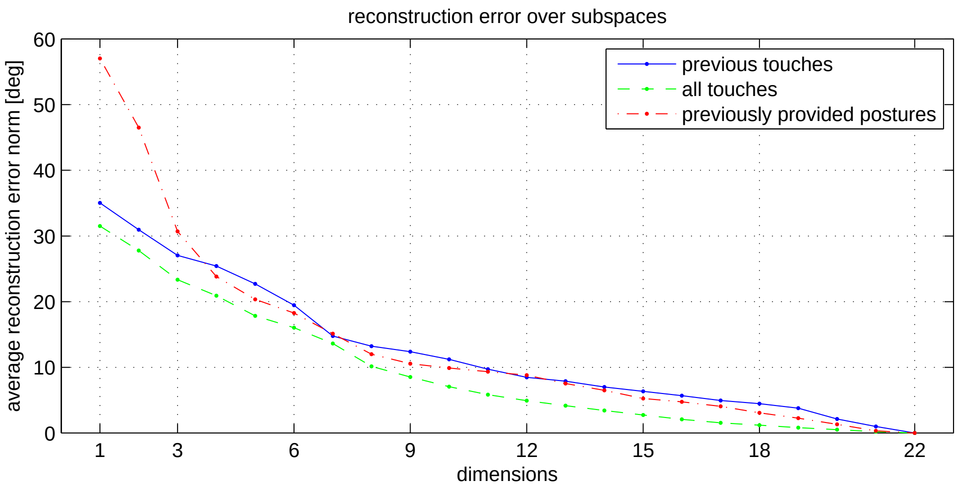


Figure 4.28: Reconstruction error for different number of dimensions of the subspace for subject D.

instructions are required, since it cannot be reduced to a simple mapping between sensor and joint angle modifications. Inspection of the mappings provided by the subjects also shows that, as expected, the mappings are user dependent. The approach of refining the mapping online revealed to be a feasible solution. In fact, experimental results supported the assumption that the need of teaching the meaning of tactile instructions decreases over time when interacting with the real robot, as we saw for simulated touch sensors in Section 3.5. We also identified general features of the mappings:

1. Sensors on one limb are very frequently employed to move joints of the same limb.
2. Associations between different parts, e.g. sensors on the head to make the robot bend the knees, should also be considered by the touch instruction interpreter.
3. The angle modifications that should be performed in response to a tactile instruction can be constrained to a linear subspace of the whole motor command space.
4. The subspace generated by the keyframe of the motions under development is (at least for some tasks) a good subspace for the representation of the movements expected in response to a touch instruction.

This knowledge on the way users use touch to develop motions will allow the future design of more advanced touch instruction predictors, that require the users to teach the meaning of fewer touch instructions in the first phases of the interaction.

The next chapter will focus on the motion representation, and show that the teaching by touching approach does not require the use of a posture based representation. In particular, tactile instructions will be used to edit periodic motions produced by a Central Pattern Generator.

5

Editing full motions with single touches

The previous chapter presented an implementation of the teaching by touching approach that employs a description of the motion based on a sequence of frames. However, as we stressed several times, the teaching by touching paradigm does not impose the use of this kind of motion representation. This chapter will focus on showing that the approach can be applied to a completely different motion representation, Central Pattern Generators (CPGs). The concept of sequence of postures and modification of postures, used in the previous chapters, will be abandoned. In the system presented in this chapter, parameter changes have effect on the whole signal generated by the CPG, thus a single touch translates to a modification of the whole motion.

Central Pattern Generators are a biologically inspired approach for motor control. Many of the periodic movements of animals, like swimming, walking or chewing, are controlled by groups of neurons called Central Pattern Generators (CPGs) that can produce a rhythmic activity even in the absence of external inputs. When motor and sensory feedback is included, the resulting entrainment between the neural circuit and the environment brings several advantages in terms of stability and ability to adapt to environmental changes.

The inherent capability of synchronizing the motion to external perturbation and adapting to the environment reveals to be very appealing in the robotics field, hence artificial CPGs were proposed as a solution for motion control. Furthermore, their computational cost is very limited, since the signal sent to the motors is generated by dynamical system, i.e. concretely by the evolution of a differential equation. Actually, even hardware implementations can be found in literature, for instance [110] or [111]. In other terms, CPGs allow the development of extremely simple, inexpensive, low power consumption, compact and flexible controllers for motion control. Therefore, they represent an interesting choice for the control of small humanoids, which are one of the main targets of the teaching by touching approach.

Often artificial CPGs consist of weakly coupled oscillators, where each oscillator consists of a couple of neurons representing extensor and flexor neurons [112]. An introduction on various oscillator models that can be found in literature will be presented in Section 5.1. The section will also briefly list the most diffused network types, i.e. the way of interconnecting the neurons.

As reported in [14] the determination of the numerous CPG parameters is very difficult because their value depends both on the robot and on the environment. Usually CPGs parameters are set automatically by genetic algorithms [112], policy gradient [113], reinforcement learning [14], or similar techniques.

Although there are obvious advantages in terms of minimal user effort, these approaches gives the motion developer little control over the resulting movement. In fact, the only strategy available for controlling the resultant motion is changing the fitness/evaluation/reward function. Some features like the similarity to human movement cannot be easily expressed mathematically. For instance, when developing a walk motion for humanoid robots if the speed is used as a reward, it is difficult to prevent the resultant motion from being fast and awkward.

Conversely manually setting each of the parameters, for instance from a console or a slider-based graphical user interface, allows the user to finely control the motion. This is, however, time-consuming, and, especially in the case of oscillators which are not easy to predict, like Matsuoka's neuron [114], it can be very unintuitive and difficult.

In this chapter, we propose the use of tactile interaction to set the CPG parameters and create periodic motions. As in the previous chapter, we imagine the user to watch the movement of a robot equipped with touch sensors. The user touches the robot, and depending on the tactile instruction the CPG parameters, and thus the motion, are modified. The user then observes the new motions and repeats the editing process until the desired robot movement is satisfactory.

To allow this interactive process we need a highly predictable network of oscillators. In fact, if similar touches could lead to quite different consequences we can imagine that the usability of the system would be strongly negatively affected. Section 5.2 therefore presents the highly predictable oscillator network we employed.

A strategy to convert user touches to CPG-parameter modifications must then be defined. In Section 5.3 we illustrate a very simple fixed protocol. Obviously a supervised learning setup, as the one presented in Chapter 3, could be used, but we preferred to keep things as simple as possible and focus only on the motion representation.

Section 5.4 will briefly describe the practical implementation used for verifying the feasibility of the approach. Finally, Section 5.5 will describe experiments on the realization of humanoid robot motions with the proposed CPG programming by touching.

5.1 Central Pattern Generators

This section briefly describes Central Pattern Generators, with particular attention to their robotic applications. In detail, after a brief introduction on the CPGs found in nature, a survey on the most common types of oscillators used in robotics will be provided.

The equations for sinusoidal, Hopf, adaptive Hopf, Rayleigh, Van Der Pol, FitzHugh-Nagumo, Hopfield, Ekeberg and Matsuoka oscillators will be given. Then, the oscillator network structures recurrent in literature will be briefly listed. Finally, considerations on the predictability, a fundamental requirement for the teaching by touching approach, will be reported.

5.1.1 Biological CPGs

Central Pattern Generators (CPGs) are neuronal circuits that can be found in living beings. Many parts of their nervous system produce patterns independent of their sensory input. One way for a neuronal circuit to oscillate is to have a neuron that generates periodic activity and influences the neighboring neurons through the synaptic interactions. In this case, the intrinsically oscillating neuron is called a "pace-maker" because it provides the pace for the whole network. In many cases, however, CPGs are composed only by neurons that are unable to oscillate when isolated, but that exhibit periodic activity when interconnected.

A single neuron or a network of neurons and, in general, any system of coupled oscillators which is able to exhibit rhythmic activity in the absence of sensory input is defined as a Central Pattern Generator. Examples of central pattern generators found in nature are the neuron networks that control periodic movements like the eels' motion, the fish swimming [115], the rhythmic movement of the wings of crickets during song production [116], insects' wing movements for flight, humans' chewing [117] and walking [118]. Animal locomotion is usually characterized by rhythmic activity and the use of multiple degrees of freedom, i.e. multiple joints and muscles. In fact, all types of vertebrate locomotion rely on some kind of rhythmic activity to move forward: undulations of the body and/or oscillations of fins, legs or wings. By rhythmically applying forces to the environment (ground, water, or air), reactions forces are generated and these forces move the body forward.

This type of locomotion is in contrast to most man-made machines which usually rely on few degrees of freedom (e.g. a limited number of powered wheels, propellers, or jet engines), and continuous, rather than rhythmic actuation [119]. Animal locomotion is significantly more difficult to control than most wheeled or propelled machines, since the oscillations of the multiple degrees of freedom need to be well coordinated to generate efficient locomotion. However, as can be observed from the swimming of a dolphin or from the running of a goat in an irregular terrain, animal locomotion presents many interesting features, such as energy efficiency and adaptability.

Despite the diversity in types of locomotion, the general organization of vertebrate locomotor circuits appears to be similar in most animals. Locomotion is just started and modulated by descending pathways from the brain's locomotor centers. Since the sixties, it is in fact known that simple electrical stimulation of the brain stem[1] initiates the walking gait in a decerebrated cat [38, 120], and progressive increase of the amplitude of the stimulation leads to an increase of the oscillation frequency accompanied by a switch from walking to trotting and eventually to galloping. This demonstrates that the brain stem and the spinal cord contain most of the circuitry necessary for locomotion, including complex phenomena such as gait transitions. In other terms, just a command signal is produced in the brain, and this basically sets the central pattern generator going. The CPG, by the motor neurons then makes the appropriate muscles contract and a motion is generated. It is worth to note that since a single muscle can only move in one direction and it cannot expand again without the help of an opposing muscle that contracts and pulls the first muscle back, in order to create rhythmic movement there must be more than one muscle, usually a flexor-extensor pair, connected to the CPG.

Although experiments with completely isolated spinal cords showed that no signal is necessary to generate rhythmic patterns, usually CPG networks receive signals from sensory feedback. Sensory feedback is essential for shaping and coordinating the neural activity with the actual mechanical movements. The main sensory feedback to the CPGs is provided by sensory receptors in joints and muscles, and in fact rhythmically moving the tail or a limb of a decerebrated vertebrate is often sufficient to initiate the rhythmic patterns of locomotion. The frequency of oscillations then matches the one used to stimulate the limb movement. This kind of coupling permits, for instance, to maintain a posture during locomotion.

5.1.2 Artificial CPGs

Although CPGs are usually discussed in terms of biological entities, as seen in the previous subsection, they can also be employed for a biologically inspired robot controller. This allows to bring some of the desirable features of CPGs found in living beings, like robustness and adaptability, to the robotics field.

Systems similar to animal CPGs, often realized as weakly coupled oscillators, have been proposed for the control of many kinds of robots, such as hexapods [121], quadrupeds [122, 123], bipeds [36, 124, 125, 126], snake robots [127], etc. CPGs offer several advantages in terms of simplicity and the ease with which sensory feedback can be introduced [128] as well as in terms of the adaptability and robustness of the resultant system [14].

Many alternatives are available in the choice of the type of oscillator/neurons employed for the realization of the CPG. One of the simplest oscillators is the sinusoidal one, in which each of the oscillators of the network is controlled by the equation

[1]The brain stem is the lower part of the brain, the rostral continuation of the spinal cord.

$$\dot{\theta}_i = \omega_i + \sum_i W_{ij} sin(\theta_j - \theta_i - (i-j)\phi)$$

where θ_i is a variable representing the internal state of the i-th oscillator, ω_i is its natural frequency, ϕ is the phase between two successive oscillators (i.e. the phase between the i-th oscillator and the $i-1$-th one) and W_{ij} is the weight of the connection from the j-th oscillator to the i-th. This kind of oscillator is often employed for snake robots like in [129]. In [124], the controller and robot dynamics, represented respectively with the two differential equations

$$\dot{\phi}_c = \omega_c + K_c sin(\phi_r - \phi_c)$$

$$\dot{\phi}_r = \omega_r + K_r sin(\phi_c - \phi_r)$$

are coupled to oscillate with a common phase difference and frequency equal to

$$\Psi^* = \phi_r - \phi_c = \frac{\omega_r - \omega_c}{K_c + K_r}$$

$$\omega^* = \frac{K_r \omega_c + K_c \omega_r}{K_c + K_r}$$

Another choice is the Hopf oscillator, which can be expressed by the equations

$$\dot{x}_i = \gamma_i(\mu_i - r_i^2)x_i - \omega_i y_i + \sum_j W_{ij} x_j$$

$$\dot{y}_i = \gamma_i(\mu_i - r_i^2)y_i - \omega_i x_i$$

$$r_i = \sqrt{x_i^2 + y_i^2}$$

where as before ω_i is its natural oscillation frequency and μ_i is its natural amplitude. γ_i is a parameter used to control how fast the system approaches its natural amplitude after a perturbation. The two real valued state variables x_i and y_i can be expressed using a single variable z_i assuming complex values. The resulting formulation, used for instance in [130], is:

$$\dot{z}_i = \gamma_i(\mu_i - |z_i|^2)z_i + i\omega_i z_i + \sum_j W_{ij} z_j$$

In [131] it is also proposed to create a more complex oscillator connecting two oscillators in "parallel", i.e. giving the same input to both of them and getting as output a weighted sum of the outputs of the two oscillators. An extension to this approach is used in [125], where a set of modified Hopf oscillators are connected in parallel to construct an oscillator that can produce a high variety of outputs (this is not possible with the common Hopf oscillators). In detail, in this work the classical Hopf oscillator is modified by making the natural frequency a state variable too, as done in [132]:

$$\dot{x}_i = \gamma_i(\mu_i - r_i^2)x_i - \omega_i y_i + \epsilon F(t)$$
$$\dot{y}_i = \gamma_i(\mu_i - r_i^2)y_i - \omega_i x_i$$

$$\dot{\omega} = -\epsilon F(t)\frac{y_i}{r_i}$$

where ϵ and γ are constants and $F(t)$ is a periodic signal given as input to the oscillator. Due to the modification, the oscillator is able to adapt its frequency to the main frequency component of the input signal. If n oscillators of this type are connected in parallel, given a periodic input signal, each of the oscillators will adapt to one of the frequency components. In detail, each oscillator will assume one of the main n frequency components of the signal, or, in other terms, the set of oscillator performs a sort of online Fourier transform of the input signal (note that it is not necessary to know any of the features of the signal, like the main period). If the signal is composed by less than n sinusoidal components, more than one oscillator can assume the same oscillation frequency, but the various amplitudes will be such that the total amplitude is equal to the one of the input signal.

Another kind of oscillator, used for instance in [126], is Rayleigh one, controlled by the equation

$$\ddot{x}_i - \delta(1 - q_i\dot{x}_i{}^2)\dot{x}_i + \omega^2 x_i - \sum_j W_{ij}\dot{x}_j = 0$$

which can also be expressed, giving a two dimensional representation, by the set of equations

$$\dot{v}_i = \delta(1 - q_i v_i^2)v_i - \omega^2 x_i + \sum_j W_{ij} v_j$$
$$\dot{x}_i = v_i$$

By differentiating the Rayleigh oscillator equation with respect to time and taking the first derivative as the main variable, it is possible to derive the Van Der Pol oscillator:

$$\ddot{x}_i + \mu(x_i^2 - 1)\dot{x}_i + \omega^2 x_i + \sum_j W_{ij} x_j$$

This kind of oscillator was used in robotics, for instance, in [133] and [134].

Other oscillator models were derived directly from the field of biology. In [135], by studying the giant axon of a squid, Hodgkin and Huxley derived a model of how action potentials in neurons are initiated and propagated. This model was simplified by Richard FitzHugh, to conceptually isolate the essential mathematical properties of excitation and propagation from the electrochemical properties of sodium and potassium ion flow. The model was in fact reduced to a two variable model for which the phase plane analysis applies, and an equivalent electric circuit was developed by Jin-ichi Nagumo, Suguru Arimoto and Shuji Yoshizawa in [136]. One possible formulation of this FitzHugh-Nagumo model is:

$$\tau_i \dot{u_i} = -v_i + u_i - \frac{u_i^3}{3} + \sum_j W_{ij} u_j$$

$$\dot{v_i} = \tau_i (u_i - \beta_i v_i + \gamma_i)$$

where τ_i is a time constant that can be used to change the oscillation frequency. An example of a quadruped controlled by this kind of oscillator is given in [123].

A very diffused neuron model is the Hopfield's one, also called leaky integrator from the form of its equation:

$$\tau_i \dot{u_i} = -u_i + \sum_j W_{ij} \sigma(u_j) + c_i$$

where σ is the standard logistic function, or sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

This simple model can be found in many applications, for instance [137, 138, 121, 139, 140]. In [141], the Hopfield model is modified to introduce synaptic depression. This is a kind of adaptation, which is a common natural phenomena by which a constant stimulus is perceived less and less. The modified model consists of the following two equations

$$\dot{u_i} = -u_i + \sum_j (1 - d_j) W_{ij} \sigma(u_j) + c_i$$

$$\tau_i \dot{d_i} = -d_i + \frac{1}{2} \sigma(u_i)$$

where the variable d_i is used to represent the depression value.

Another biologically inspired model comes from Grillner and colleagues' work on the lamprey spinal cord and the model of the lamprey central pattern generator then developed by Örjan Ekeberg in [142]. The equations for each of the neurons are

$$\tau_D \dot{\xi_{i+}} = -\xi_{i+} + \sum_j W_{ij} u_j$$

$$\tau_D \dot{\xi_{i-}} = -\xi_{i-} + \sum_j W_{ij} u_j$$

$$\tau_A \dot{\vartheta}_i = -\vartheta_i + u_i$$

$$u_i = [1 - e^{\Gamma(\Theta - \xi_{i+})} - \xi_{i-} - \mu\vartheta_i]^+$$

where the notation $[]^+$ is used to indicate the positive part of the argument, that is

$$[x]^+ = \begin{cases} 0 & if\, x \leq 0 \\ x & otherwise \end{cases}$$

This model was used in the robotics field by A. J. Ijspeert [143].

One of the most utilized CPG is the one proposed by Matsuoka in 1985 [114]. The equations for this model are

$$\tau_u \dot{u}_i = -u_i - \beta v_i - \sum w_{ij}[u_j]^+ + c_i$$

$$\tau_v \dot{v}_i = -v_i + [u_i]^+$$

Among its direct applications, it is possible to cite [144] or [145]. Similarly, in [112] Matsuoka oscillators are used for generating walking patterns, while in [146] they are used to make the robot jump entrained with the environment feedback.

Among all of the possible structures for the connection between the oscillators, we notice that essentially five structures are present in the literature, as shown in Fig. 5.1:

1. chain [147, 128, 148, 149], used mainly for snake robots
2. star [150, 151, 124, 152, 153], that is a "pacemaker"/ "clock" oscillator which provides a synchronizing signal to the other oscillators
3. tree [154, 155], where essentially the oscillators are connected as a tree, from proximal to distal joints
4. connection between homologous joints [156, 157, 158, 159, 160, 161, 37, 162, 163], i.e. joints with a similar function
5. full connection between the oscillators [164, 165, 166]

5.1.3 CPG predictability

As can be seen from the equations given in the previous subsection, most of the oscillators found in literature are highly non linear. Often, this strong nonlinearities cause the existence of multiple limit cycles. A very simple example can be given using a network of Matsuoka oscillators. Suppose to connect four neurons as shown in Fig. 5.2. Neurons N_1 and N_2 are interconnected and form an oscillator. This is a typical configuration, with one neuron assumed to have the role of controlling the extension and another one assumed to be controlling the flexion. When dealing with hinge joints, therefore, the

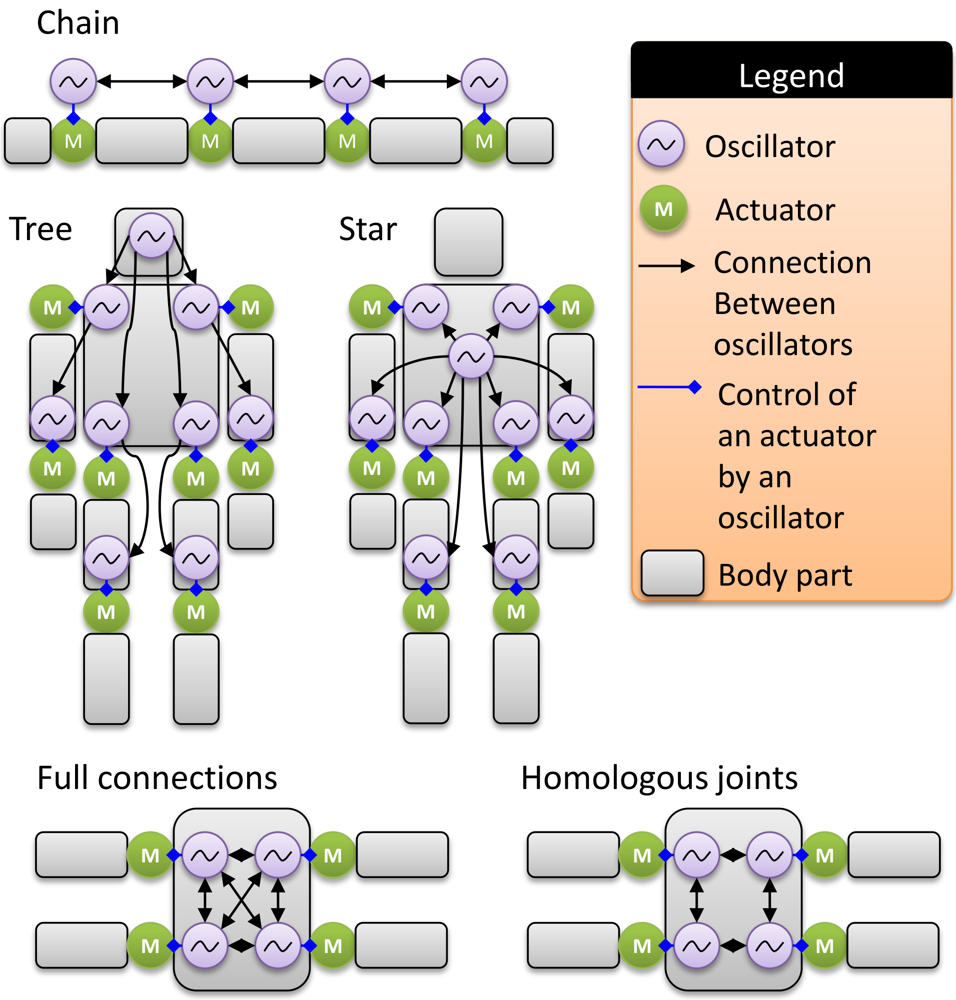


Figure 5.1: Types of network of oscillators found in literature. Chain of oscillators, found for instance for the control of a snake robot in [148]; tree shaped connection, found for instance for the control of the gait of a humanoid robot in [155]; star configuration, where a clock oscillator is used to synchronize the system, employed for instance for a drumming task presented in [150]; full connection between the oscillators, as proposed in [166] for the control of a quadruped robot; connection between the homologous joints, as done in [158].

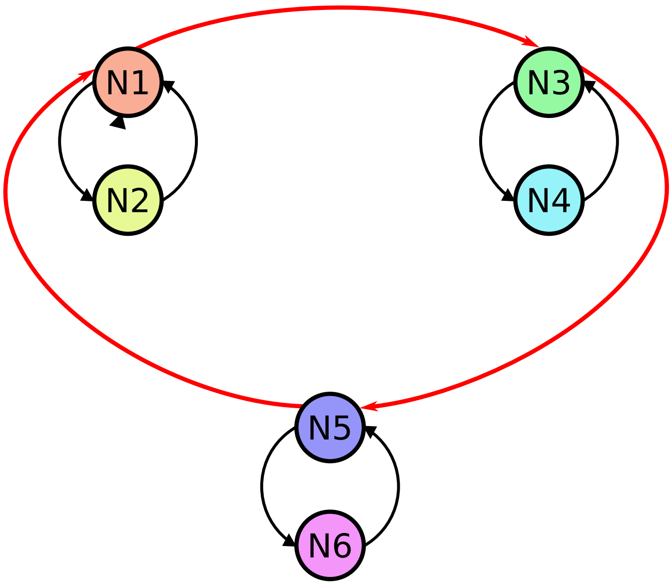


Figure 5.2: Simple configuration exhibiting multiple limit cycles. Weight $W_{i,j}$ is indicated by an arrow from neuron N_j pointing to neuron N_i. Black is used for connections between the neurons of the same oscillator, while red arrows represent connections between neurons of different oscillators.

torque or angle is usually set as the difference of the values of the two neuron outputs, i.e. in this case $u_1 - u_2$. In a completely analogous manner, neurons N_3 and N_4 form another oscillator, whose output $u_3 - u_4$ is assumed to control a second joint, and neurons N_5 and N_6 control a third joint. The oscillators are then synchronized by connections, shown in red in Fig. 5.2, that to using non null values for weights $w_{3,1}$, $w_{5,3}$ and $w_{1,5}$.

Even this simple structure exhibits multiple limit cycles, that depend on the initial condition. Figures 5.3 and 5.4 report the evolution of the outputs for identical parameter settings and different initial conditions. We notice that the evolutions are qualitatively different, and they don't differ, for instance, just by a simple permutations of the outputs.

When employing the teaching by touching approach to change the CPG parameters, such a behavior is unacceptable. In fact, if the parameters are changed during the motion execution, the resulting robot behavior depends on the timing at which the parameters are changed, and not solely on the parameters. This means that the same tactile instruction would be translated into completely different motion modifications depending on slight differences in the timing at which it is given.

One possibility to avoid this effect would be to use adaptive Hopf oscillators [125]. The limit cycle evolution preceding the tactile instruction could be stored in memory, modified according to the instruction, and then used as a perturbation to make the Hopf oscillator approximate the new motion. However, practical experiments show that the adaptation of adaptive Hopf oscillators is too slow for online usage. Furthermore, the convergence to one of the frequencies of the perturbations is guaranteed, but the

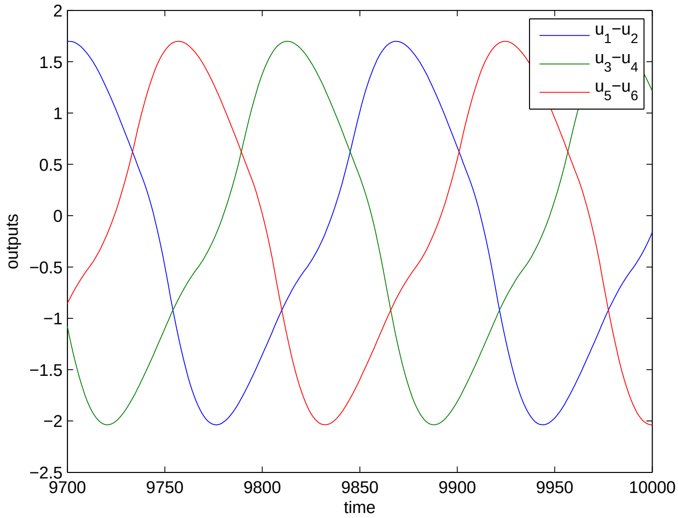


Figure 5.3: Outputs of the three oscillators of Fig. 5.2 when the limit cycle is reached from the initial conditions $u = [0.0999, 0.5046, 0.7471, -0.7059, -0.9883, 0.9757]^T$ and $v = [0.0768, 0.5865, 0.6040, 0.1839, 0.2886, 0.4419]^T$. Parameters were set to $c = 2$,

$$\tau_u = 0.2,\ \tau_v = 0.4, \beta = 2.5,\ W = \begin{bmatrix} 0 & 2.5 & 0 & 0 & 0 & 0 \\ 2.5 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 2.5 & 0.5 & 0 \\ 0 & 0 & 2.5 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 2.5 \\ 0 & 0 & 0 & 0 & 2.5 & 0 \end{bmatrix}.$$

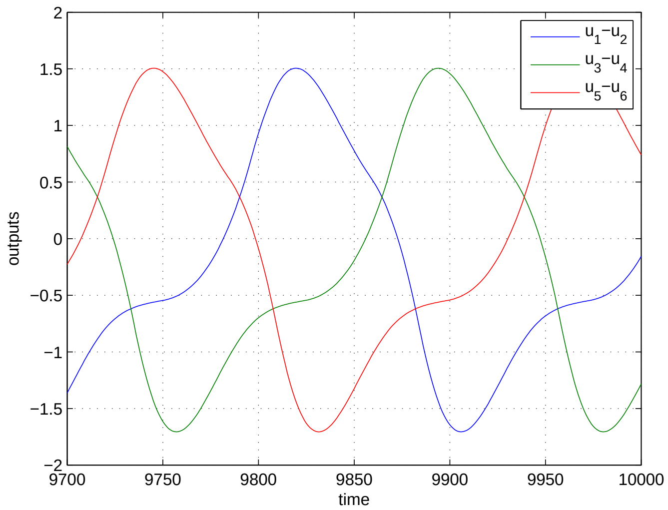


Figure 5.4: Outputs of the three oscillators of Fig. 5.2 when the limit cycle is reached the initial conditions $u = [0.8981, -0.6008, -0.0319, 0.5365, -0.4061, 0.9221]^T$ and $v = [0.4888, 0.2295, 0.0564, 0.5740, 0.3706, 0.3309]^T$. Parameters are identical to the ones reported for Fig. 5.3

frequency to which the oscillator converges depends on the initial conditions as well.

For these reasons, we excluded from our choice neurons that present the possibility of having multiple attractors. We adopted the easily predictable classic Hopf oscillator, with slight modifications to improve the synchronization of oscillators that have strongly different limit cycle periods. Similarly, among the possible interconnection architectures we decided to use the "star" one, because of its robustness to variations in the initial conditions. Next section will describe in detail the CPG network designed for the teaching by touching approach, and Section 5.3 will introduce a simple tactile protocol that can be used to change its parameters.

5.2 A predictable CPG network

As seen in the previous section, many alternatives are available in the choice of the type of oscillator/neurons employed for the realization of the CPG, as well as for the interconnection between oscillators. Our purpose is to let the user modify the behavior of the robot by changing the oscillator parameters through touching. We therefore need to have predictable changes in the behavior of the oscillators, i.e. in the movement of the robot, consequent to a change in the parameters. For these reasons, for its simplicity and robustness, the Hopf oscillator thus appears to be a good choice.

The predictability of the system is then strongly influenced by the oscillator interconnections. In the previous section we saw that the structures found in literature can be roughly categorized in five types: chain, star, tree, connection between homologous joints and full connection. We decided to employ the star configuration, i.e. a network with a "pacemaker" oscillator that provides a synchronizing signal to the other oscillators. This network is highly predictable, since the pacemaker oscillator gives a synchronizing signal to all of the oscillators controlling the joints but no unexpected effects occur due to the interaction between groups of oscillators as in the case of connection between homologous joints. Precisely, our implementation uses a directional coupling from the clock oscillator to the others. This assures the maximum system predictability. Tests on whether a bidirectional coupling can improve entrainment with the environment without affecting the ability to foresee the system behavior will be performed in future works.

Furthermore, this configuration is very general and task independent since no assumptions are made about which joints should be synchronized, a feature required by our system. Star structure is also highly predictable, since the pacemaker oscillator gives a synchronizing signal to all of the oscillators controlling the joints, but no unexpected effects occur due to the interaction between groups of oscillators, as in the case of connection between homologous joints.

Formally, in our implementation each of the n degrees of freedom of the robot is controlled by one oscillator and a further "clock" oscillator provides reference signals for these oscillators. Let us identify by C_0 the reference oscillator and by C_j, $1 \leq j \leq n$, the oscillators controlling the robot joints.

Using the complex number representation for the Hopf oscillator [130] we have for the j-th oscillator, $0 \leq j \leq n$

$$\begin{aligned} \dot{z}_j &= \gamma \left(\mu_j - |z_j|^2\right) z_j + i\omega_j z_j + F_j(t) \\ m_j &= \Re\left\{z_j\right\} + o_j \end{aligned} \tag{5.1}$$

In detail

- $z_j \in \mathbb{C}$ is the state of the oscillator
- $m_j \in \mathbb{R}$ is the control signal for the actuator
- γ is a coefficient for the speed of recovery after perturbation [125]
- $\mu_j \in \mathbb{R}, \mu_j \geq 0$ controls the amplitude of the oscillation
- $\omega_j \in \mathbb{R}, \omega_j \geq 0$ controls the oscillation frequency
- $F_j(t)$ is an external perturbation signal
- o_j is an offset value used to set the position around which the joint oscillates

Clearly CPGs are aimed at the development of periodic motions. Therefore, it appears natural to have a frequency for the whole motion, and setting the frequency of the movement of single joints to an integer multiple of this frequency. For this reason, we set

$$\omega_j = p_j \omega_0 \tag{5.2}$$

where $1 \leq j \leq n$ and $p_j \in \mathbb{N}$. In the current prototype implementation no feedback signal is introduced, so $F_0(t)$ is zero (the main clock is not influenced by the external world), while for $1 \leq j \leq n$ the feedback is solely given by

$$F_j(t) = we^{i\phi_j} z_0^{p_j} \tag{5.3}$$

In other terms, $F_j(t)$ consists essentially of the perturbation from the clock oscillator that permits a synchronization of the whole system. The reference signal z_0 is elevated to the power p_j, so the frequencies of the oscillator and of the perturbation are close. The similarity of the frequencies leads to an easier synchronization and a predictable phase between the j-th oscillator and the reference oscillator [167]. The coefficient w determines the coupling strength between C_0 and the other oscillators while the term $e^{i\phi_j}$ changes the phase difference between the clock oscillators and the others.

In the current implementation we set $\mu_0 = 1$, $w = 0.1$ and $\gamma = 1000$. The other parameters are set by exploiting user's tactile instructions. Their initial values are $\mu_j = 0$, $p_j = 1$, $\phi_j = 0$, and $o_j = 0$, $1 \leq j \leq n$.

The behavior of the network is easily predictable. In fact, the signal generated by each oscillators consists essentially of a sinusoidal waveform. From the equations, it clearly appears how the parameters affect the signal generated by the j-th oscillator: μ_j affects the amplitude, p_j the frequency, ϕ_j the timing of the movement and o_j the location around which the joint is moved. The way in which the parameters are changed in response to tactile instructions will be provided in the next section.

5.3 Touch protocol

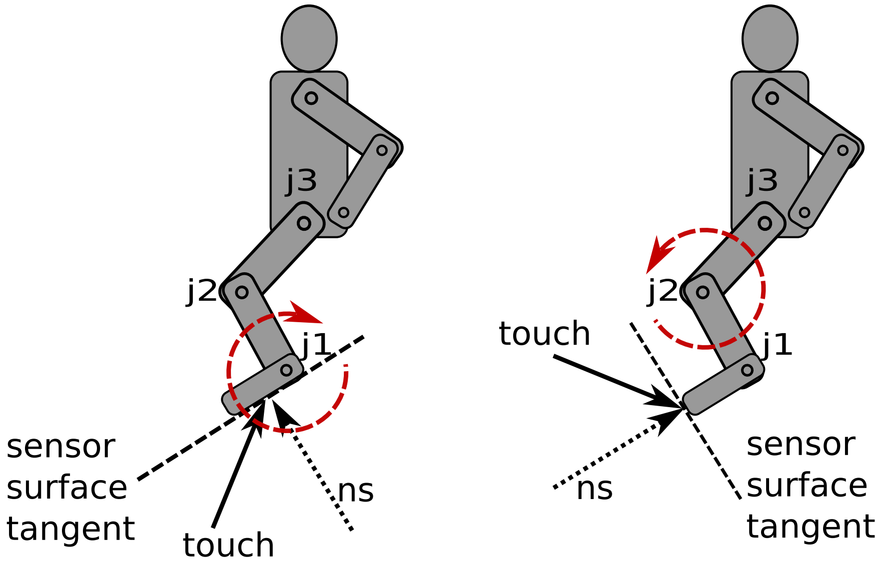
This section describes the protocol used to map tactile instructions into changes of the CPG parameters. Since this work aims exclusively at validating the feasibility of the approach, we decided to simplify the system as much as possible and opted for a static mapping between the user actions and the parameter changes. Analysis of adaptable systems that better fit the user's way of touching will be conducted in future works.

As explained in the previous section, each joint of the robot is controlled by an independent CPG, so it is possible to modify the movement of a single joint by changing its parameters. In particular, from what has been shown in the previous section, for each joint we can control

- the amplitude of the oscillation of each joint, by μ_j
- the frequency of the movement of each joint, by p_j
- the phase of the movement, with respect to the main oscillator, through the parameter ϕ_j
- the zero position (offset) around which the joint moves, by o_j

Given this possibility of controlling the individual joints, we based our protocol on the assumption that touching the part whose movement is to be modified should be intuitive for the users.

Therefore, depending on the pressed touch sensor, we decide which is the oscillator whose parameters should be modified. This is done by selecting the most distal joint that causes a movement of the pressed sensor in the direction normal to the sensor surface. We then decide which feature of the movement (amplitude, frequency, phase or offset) is affected depending on the pressure pattern. Expressly, if the user keeps pushing the sensor for a very long time, the offset is modified. If the user pushes for a shorter time, the amplitude is modified, while a single tap is used to change the phase. Two consecutive taps are used to change the frequency, and in particular if the touch is applied to the robot's torso, then the frequency of the whole motion is changed. An increase in the frequency is obtained by tapping the robot two times, with both of the taps of short



(a) (b)

Figure 5.5: Example of the determination of the joint whose parameters should be modified when a sensor is pressed. In the first case the parameters of joint j_1 are modified. In the second case, since the $\rho_1 =< n_s, d_{j_1} >$ is zero, j_2 is selected.

length. A frequency decrease is set by tapping the robot two times, with the duration of the second touch significantly longer then the first tap.

More formally, when a sensor is pushed, the joint for which the parameters should be modified is chosen in the following way. Suppose that the robot's torso is fixed in space, and denote by d_j the derivative of the position in the space of the center of the pushed sensor when the j-th joint is rotated. Identify n_s as the vector perpendicular to the pushed sensor surface. Denote by $j_1, j_2, ...j_q$ the indices of the q joints that connect the robot's main body to the link where the sensor is located, in order from the most distal to the most proximal, as exemplified in Fig. 5.5. We identify the joint j_s such that $\rho_s = n_s^T d_{j_s} \neq 0$ and $n_s^T d_{j_k} = 0$ for $s < k \leq q$. If this joint doesn't exist we simply ignore the sensor pressure, unless it is on the torso and the touch instruction aims at changing the global motion frequency.

Once j_s is determined, the type of motion modification is chosen depending on the number of times the sensor is pushed and on the pressure times, as previously described. For measuring the pressure times, the phase of joint j_s ($\angle z_{j_s}$) is used as a time reference. Expressly, the pushing time τ_{j_s} is measured in terms of phase difference between the release time and the pushing time, counting for the phase resets, i.e. the phase difference is considered a non-negative, monotonically increasing quantity that can be larger than 2π.

As previously stated and as shown in Fig. 5.6, we change

- the offset if the user keeps pushing for a very long time

- the amplitude if the pressure lasts a shorter time
- the phase if the user operator taps one time
- the frequency if two taps are provided.

More precisely:

- If $\tau_{j_s} > \Theta_O$ (the user pushes for a very long time) the offset parameter is changed according to the direction of the applied force. i.e. $o_{j_s,new} = o_{j_s,old} + sgn(\rho_{j_s})\Delta_O$, where sgn is the sign function.
- If $\Theta_A < \tau_{j_s} \leq \Theta_O$ the amplitude parameter is updated by the value $sgn((\rho_{j_s} * m_{j_s})\Delta_A$ where m_{j_s} is the value of the output at the pushing time.
- If the user pushes for a time $\tau_{j_s} \leq \Theta_A$, releases the sensor and doesn't push it for a time Θ_P then the phase parameter ϕ_{j_s} is updated such that in the following cycles the closest maximum of oscillation occurs at the pushing time, i.e. the quantity $-\angle(m_{j_s} * z_{j_s})$ is added to ϕ_{j_s}, where z_{j_s} and m_{j_s} are considered at the pushing time.
- If the user pushes for a time $\tau_{j_s} \leq \Theta_A$, releases the sensor and before a phase change of Θ_P pushes the sensor again then p_{j_s} is incremented or decremented respectively if this second pushing time $\tau_{j_s,2}$ is greater or lower than Θ_A.
- Similarly, if the user pushes the robot main body for a time $\Delta\phi_0 \leq \Theta_A$ releases the sensor and before a phase change of Θ_P pushes the sensor again then ω_0 is increased or decreased by the quantity Δ_{ω_0} respectively if the second pushing time is greater or lower than Θ_A.

In our implementation, all the Δ and Θ values are constants, expressly $\Theta_O = \pi$, $\Theta_A = \frac{\pi}{6}$, $\Theta_P = \frac{2\pi}{3}$, $\Delta_O = \Delta_A = \frac{\pi}{12}$, $\Delta_{\omega_0} = 1$.

Although the protocol is arbitrary and fixed, we can expect a system that utilizes tactile interaction to be more intuitive than designing the motion by a simple manual parameter setting. For instance, if users want to change the center of oscillation of a joints they have to consider the position of the various motors in the kinematic chain, and decide which motor offsets should be modified. Then, depending on how each motor is mounted on the robot and on the established conventions, they need to evaluate whether the offset parameters of each joint should be increased or decreased. Conversely, with our approach, the users are just required to keep pushing the robot parts in the direction they want the centers of oscillation to be moved.

The next section will present a prototype implementation of the approach in a simulated environment, and will provide results of experiments on motion development with the proposed teaching by touching paradigm. The way of developing motions with the

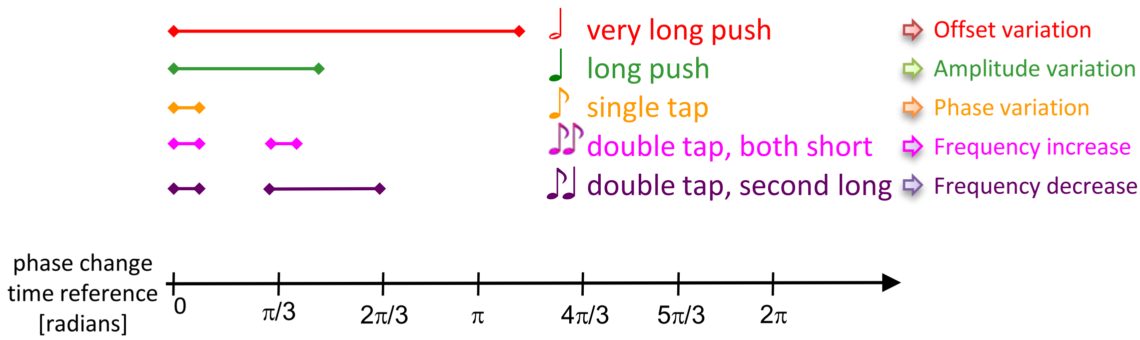


Figure 5.6: Representation of the touch patterns recognized by the system. The presence of a line indicates the pressure of a sensor. The phase change axis (time reference) is also reported.

CPG based system will differ substantially from the one of Chapters 2 and 4. In fact, in the system presented in the next section, a single touch changes a global feature of the motion, like the oscillation frequency of a joint or the timing differences between different joints.

5.4 Prototype system

In order to validate the feasibility of the approach we developed a prototype system that uses the network designed in Section 5.2 and the protocol devised in Section 5.3. We decided to conduct experiments with a simulator for reducing the system development and debug time. More precisely, M3-Neony, described in Section 4.1, was simulated using the same simulator of the experiments in Section 3.5. The user is able to interact with the 3D rendering of the robot. In particular, each link face simulates a touch sensor that can be clicked with a mouse. Further details of the simulator can be found in appendix A.

Although the results presented here are obtained using a simulated robot, we aim our technique to be applicable to real robots. When dealing with real robots, some CPG parameter configurations set by the user during motion development could lead to robot self-collisions that damage the actuators. Hence, these collisions must be predicted online. The collision detection should therefore be very fast, but it can however be inaccurate. More precisely, while we should assure collision positions to be avoided, we are not required to recognize each collision free posture as such, i.e. we can reduce the set of allowed postures to increase the computation speed. In our implementation, each link is approximated by a set of slightly bigger cuboids, and the computation is limited to checking the collisions between those cuboids.

In detail, the system memorizes the current motor positions as an n-dimensional vector $Y_{current}$. Given a vector Y_{target} representing the desired motors position, that is the outputs m_j of the oscillators, the collision detection module calculates what is the maximum rotation of each joint toward Y_{target} such that there's no collision between the

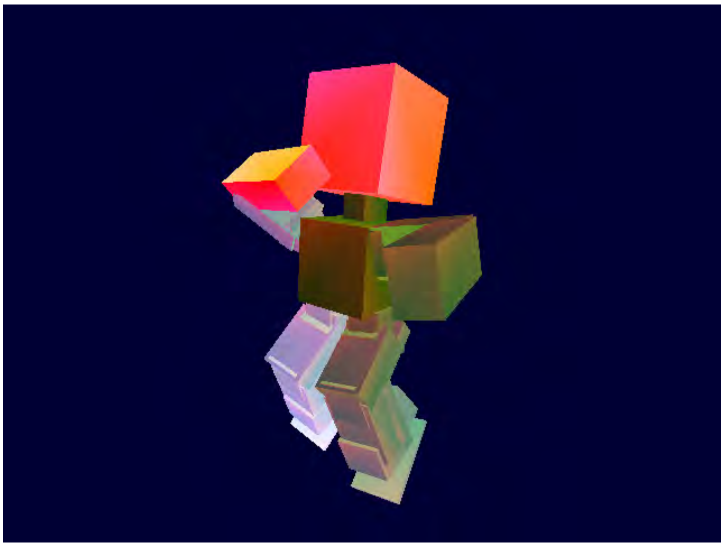


Figure 5.7: Output of the collision prevention system. The figure shows the motion that would be generated if the collision prevention system were not enabled and highlights the colliding parts by a red color.

cuboids. More precisely, for each joint, in a specified order, which is the joint ID in the current implementation, the system calculates the maximum rotation from the current angle to the target angle, with steps of 3 degrees, that doesn't cause any collision between the links moved by the rotation and the others. The position obtained in this way is sent to the motors and stored in $Y_{current}$. As most of the movements are periodic, collision detection computation is reduced further by employing a direct mapped cache. More precisely, the content of the cache is a binary value expressing whether the configuration causes self collisions or not. The position of each motor, discretized with granularity of one degree, is used as the full address that should be accessed. The mapping function between a generic address and the cache index is the hash function

$$f(Y) = mod(\sum_{j=1}^{n} Y^{(j)} \varrho_j, S)$$

where $Y^{(j)}$ is the angle of the j-th joint, ϱ_i is the j-th prime number, S is the cache size and mod is the modulo function. In the current implementation we have $n = 22$ and $S = 27077$. The system also includes a graphical representation of the collision detection computation and highlights the parts that would collide without the intervention of the prevention collision system, as depicted in Fig. 5.7. This information can be exploited by the user to refine the robot movement and get a motion that does not require any collision prevention.

Figure 5.8 provides a schema of the development cycle under the framework presented in this chapter. Similarly to what was seen in Chapters 2 and Chapter 4, the user observes

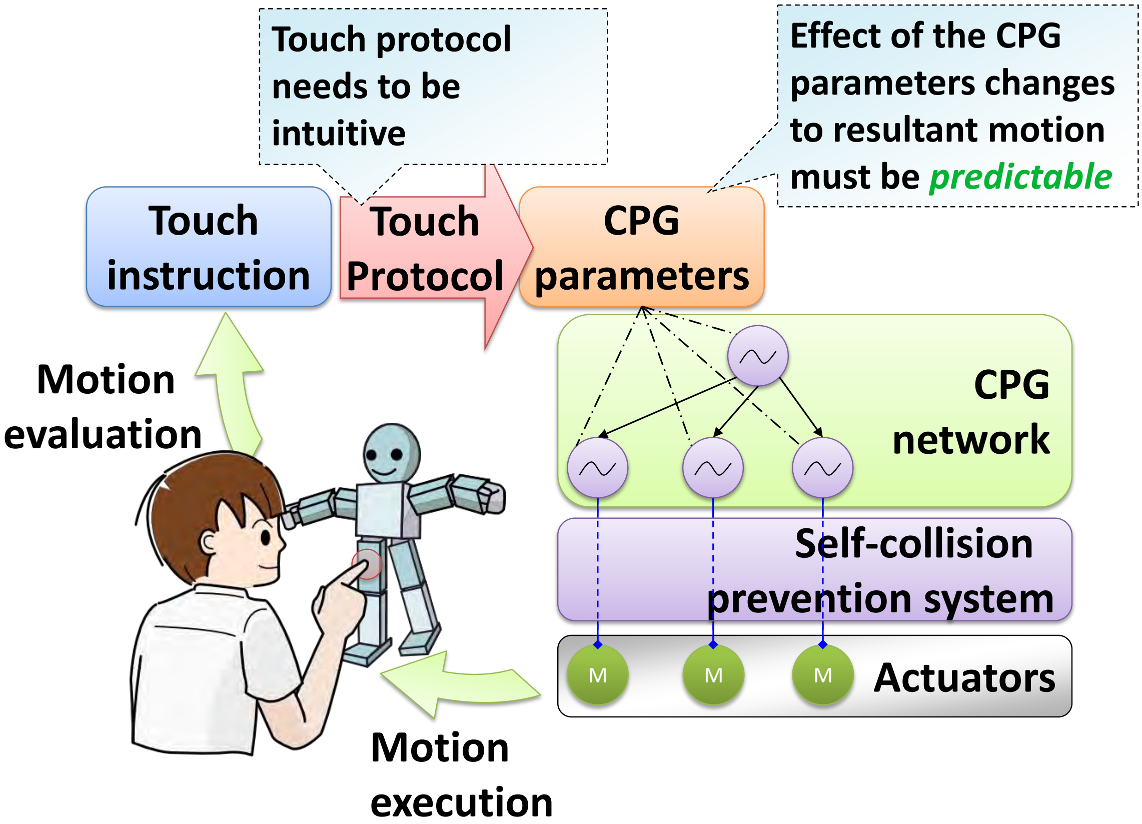


Figure 5.8: The teaching by touching approach applied to a CPG based motion representation.

the motion, touches the robot, the robot translates the tactile instruction into a motion modification and the cycle is repeated. We should note that when the users touch the robot, they may apply forces that interfere with the robot's motion. However, these forces are ignored since we imagine that the user touches the robot while it is moving, waits for the movement to stabilize after the parameter change and then decides the next modification.

Given the motion representation chosen, a network of Hopf oscillators, the system clearly focuses on the development of periodic motions. The main peculiarity of the approach presented here is clearly that single touches modify the whole motion. Furthermore, no Graphical User Interfaces as the one presented in Section 3.3 are used. The only way the user interacts with the robot is through touch sensors that, in this case, are simulated.

Next section will present experiments conducted with the developed system. In particular, it will be shown that using a genetic algorithm in a naive way to automatically set the CPG parameters produces awkward motions. Conversely, when interacting with the robot, the user unconsciously optimizes several parameters of the motion, as will be suggested by a quantitative analysis that compares the motion realized by the user and the one obtained by the genetic algorithm.

5.5 Experimental Results

In order to test the system described in the previous section, a crawling movement, a walking gait and a side-step motion were developed. Screenshots[2] of the resultant motions are reported, respectively, in figures 5.9, 5.10 and 5.11.

The crawling motion was developed by a single user in 56 minutes, by providing 57 amplitude changes, 39 phase changes, 22 offset changes and 2 frequency changes. The same user then realized a side-step movement in 29 minutes, using 31 amplitude changes, 4 frequency changes, 18 phase changes and 56 offset changes. Finally, a walking motion was obtained in 34 minutes. This required 60 amplitude changes, 15 frequency changes, 28 phase changes and 132 offset changes. Although a single experiment with a single user cannot provide any statistical evidence, we can notice that the number of commands per minute was highly increased during the motion development. In detail, in the first experiment, 2.14 commands/minute were provided, in the second, the frequency raised to 3.7 commands/minute and by the third motion the number of command/minute reached 6.9. This suggests that human operators can easily get accustomed the system. We can in fact exclude learning of the robot dynamics, since the same simulator and robot models were previously used by the user in plenty of experiments.

Table 5.1 reports the final values obtained for the CPG parameters for the three motions. The final value for ω_0 is 3.256 for the crawling, 5.256 for the side step and 1.256 for the walking motion. A further tuning, for instance to assure perfect movement symmetry or to maximize the crawling or walking speed, could be performed, but this is outside the scope of this paper.

We would also like to stress that the purpose of the experiment is not to achieve the fastest locomotion speed that is possible, but to validate the feasibility of the approach by developing a motion that is satisfactory for the user (in terms, for instance, of similarity to human movements). We therefore didn't even measure the walking velocity and ignored speed comparisons with walking gaits obtained with other methodologies.

For comparison, however, Fig. 5.12 reports a crawling movement obtained with a genetic algorithm (population size 20, 60 generations, real value encoding, roulette wheel selection, mutation probability 1). The average speed over one minute was employed as the evaluation function. We can see that the algorithm finds a shape for the legs that minimizes the friction with the ground, and uses the head as a support point to proceed forward by large arm movements. While this solution can lead to a good speed, it definitely looks awkward to humans. A difference in the smoothness of the two motions can be deduced quantitatively by observing the roll and pitch in the two cases (see Fig 5.13). The ranges of variation for the roll and the pitch of the robot are 15.4 and 14.5 degrees respectively for the motion obtained by direct interaction with the CPG and 34.7 and 23.9 for the motion optimized by the genetic algorithm.

[2]Videos are available at `http://robotics.dei.unipd.it/~fabiodl/video.php?cpg`

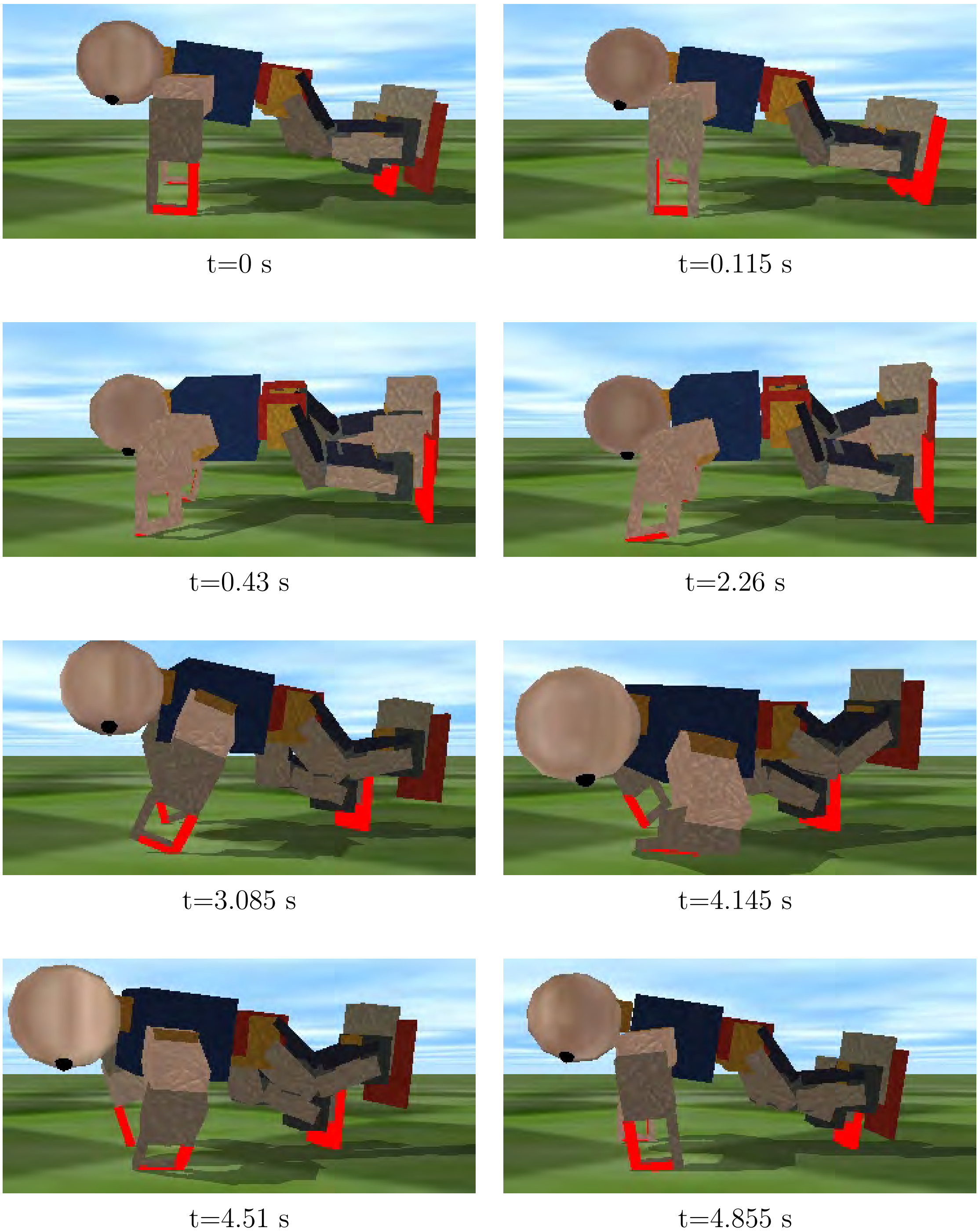


Figure 5.9: Execution of the crawling movement.

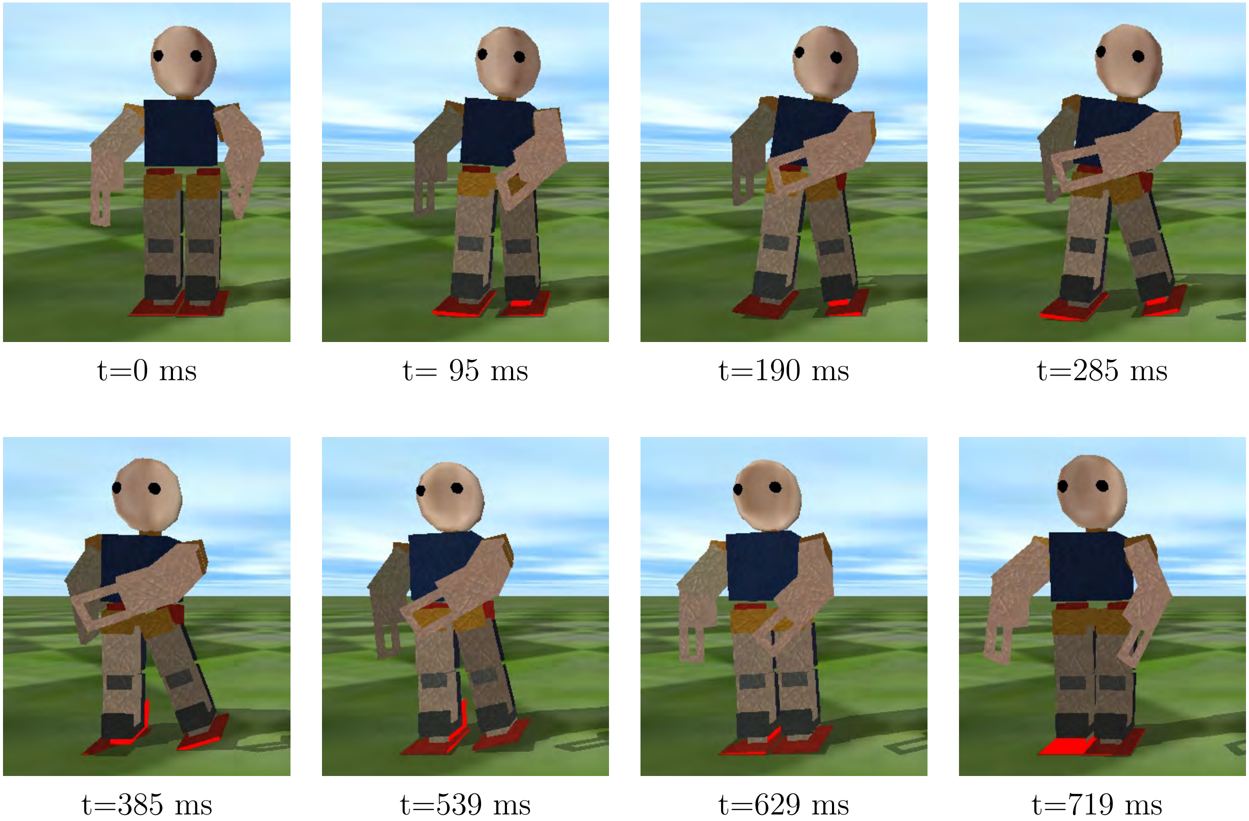
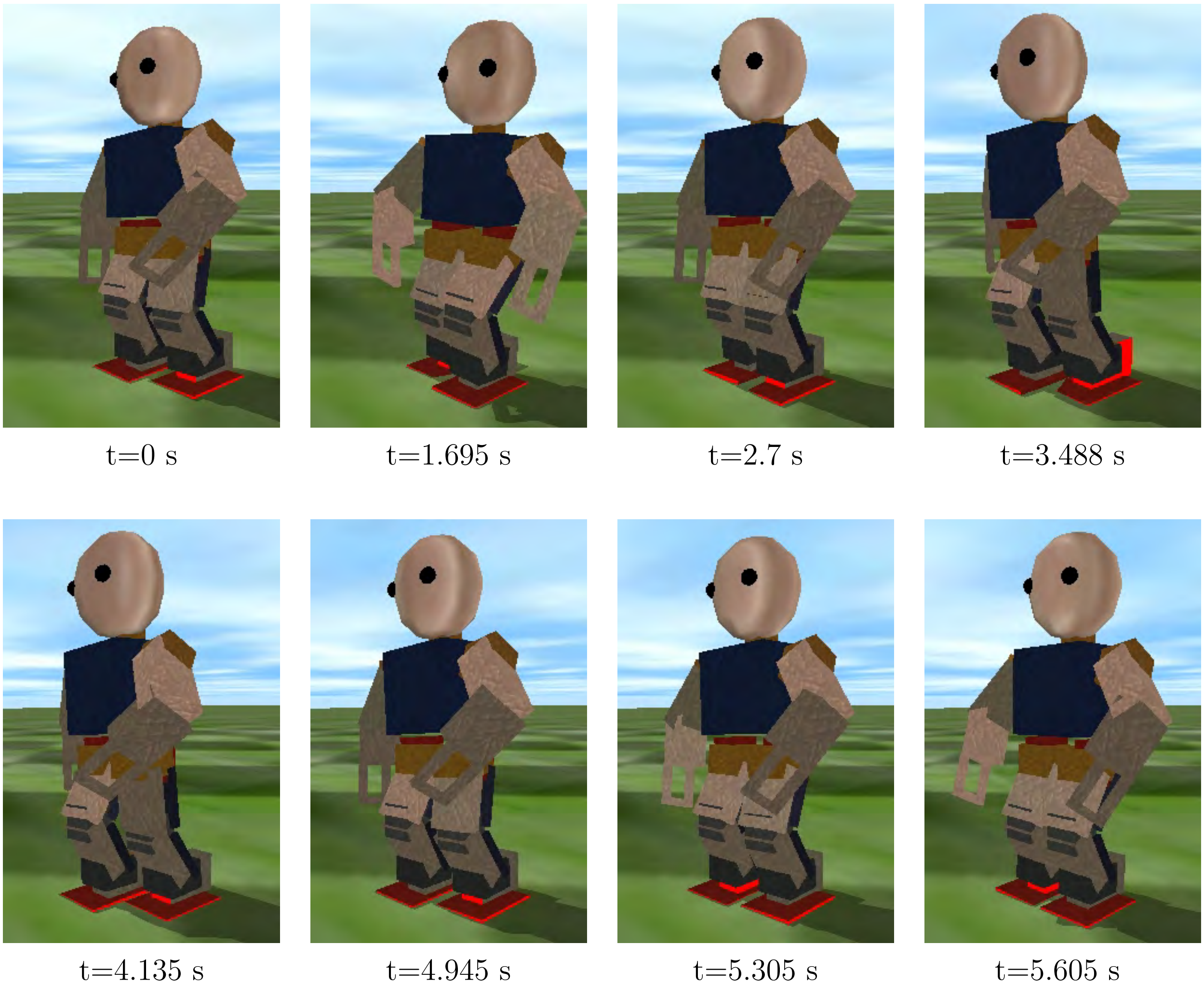


Figure 5.10: Execution of the sidestepping movement.

Table 5.1: CPG parameter settings obtained for the three motions.

Parameters / Joint	Crawling				Walking				Side Step			
	μ_j	p_j	ϕ_j	o_j	μ_j	p_j	ϕ_j	o_j	μ_j	p_j	ϕ_j	o_j
0	0	1	0	0	0	1	0	0	0	1	0	0
1	0	1	0	0	0	1	0	0	0	1	0	0
2	0.52	1	0.66	0.78	0.17	1	0.4	-0.7	0.35	1	0	-0.7
3	0.26	1	1.61	-0.52	0	1	0	0.35	0.17	1	1.86	0.17
4	0	1	9.72	0.52	0	1	0	1.22	0	1	0	1.05
5	0.52	1	2.31	0	0	1	0	-1.05	0.35	1	4.4	-1.4
6	0.26	1	2.20	0.78	0.17	1	0.49	-0.17	0	1	0	0
7	0.78	1	5.69	-0.52	0.17	1	0	-0.35	0	1	5.5	0
8	0	1	0	0	0	1	0	0	0	1	0	0
9	0	1	0	0	0	1	0	0	0	1	0	0
10	0	1	5.68	0	0	1	1.64	0	0	1	0	0
11	0	1	0	0	0	1	0	0	0.17	1	0	0
12	0	1	0	0	0	1	0	0	0	1	0	0
13	0	1	0	0	0	1	0	0	0	1	0	0
14	0.26	1	1.28	-1.05	0	1	0	-0.52	0	1	0	0
15	0.26	1	0	1.05	0.17	1	0.31	0.35	0	1	0	0
16	0.26	1	0	0	0	1	0	0.52	0	1	0	0
17	0.26	1	0	0	0	1	0	-0.52	0	1	0	0
18	0	1	0	0	0	1	0	0	0	1	0	0
19	0	1	0	0	0	1	0	0	0	1	0	0
20	0	1	0	0	0	1	0	0	0	1	0	0
21	0	1	0	0	0	1	0	0	0	1	0	0



t=0 s | t=1.695 s | t=2.7 s | t=3.488 s

t=4.135 s | t=4.945 s | t=5.305 s | t=5.605 s

Figure 5.11: Execution of the walking movement.

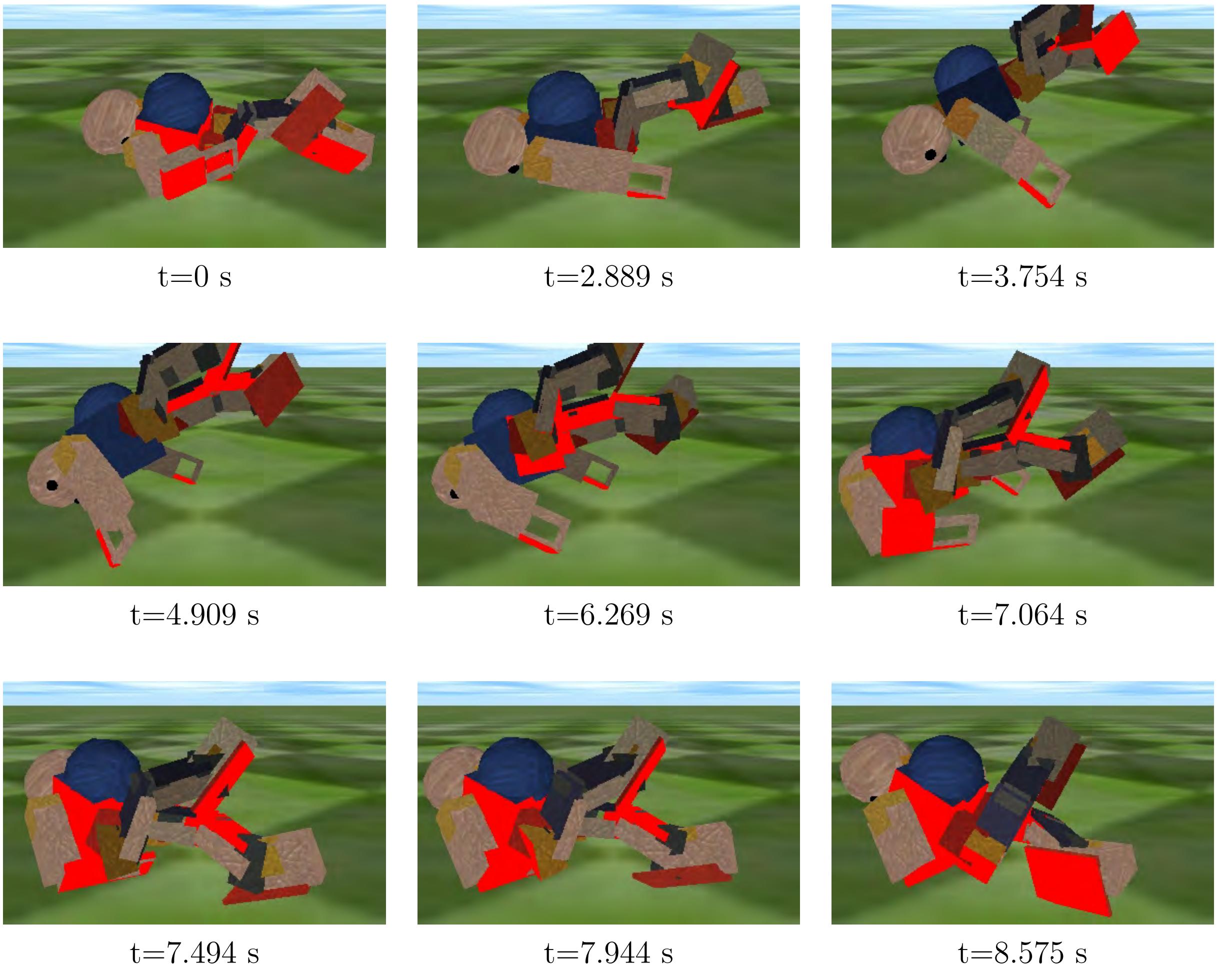
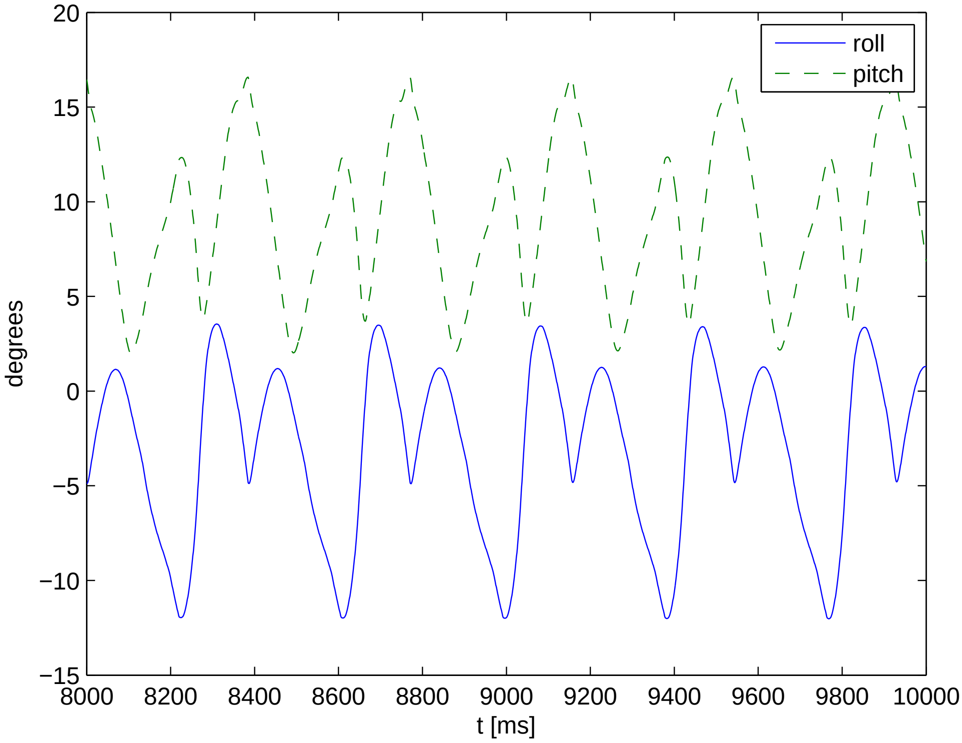


Figure 5.12: Execution of a crawling movement obtained with a genetic algorithm.

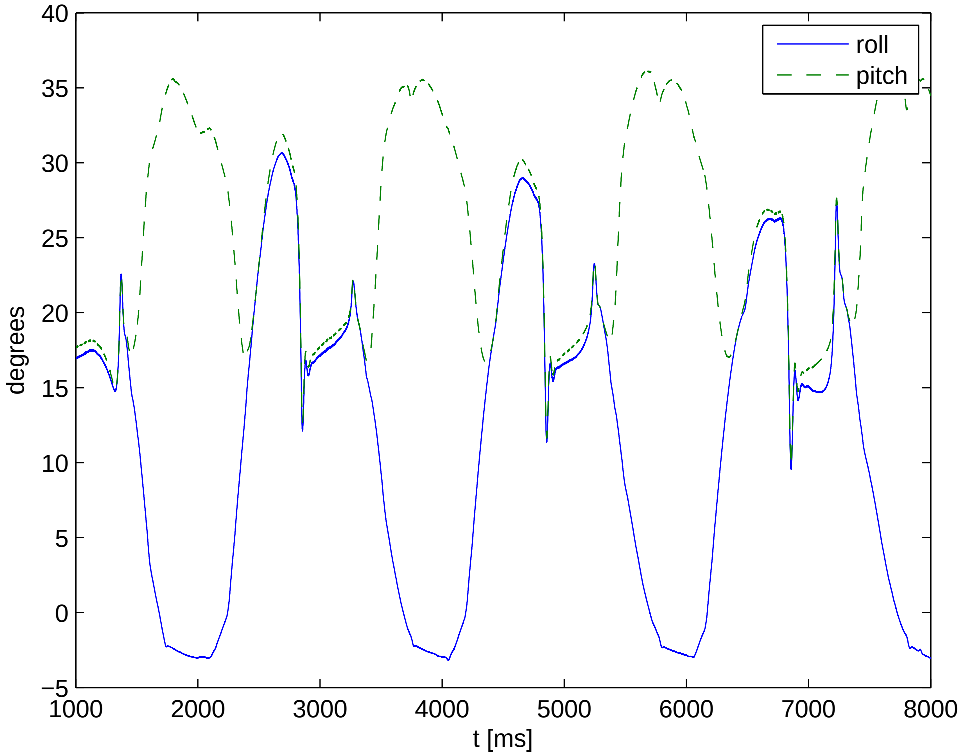
Obviously the pitch could be improved by introducing another term into the evaluation function that penalizes the high pitch and roll excursions, although deciding the weighting between the two terms is far from being a trivial task.

Furthermore we can imagine that users desire symmetries in the motion of the limbs. This would require another evaluation term, whose weight would be again difficult to guess. The difficulty in choosing the evaluation function weights highlights the richness and complexity of the evaluation that human user can provide: many factors are intuitively weighted, with priorities that are difficult to express as a weighted combination of evaluation functions. Indeed, developing systems that are able to directly exploit the human user evaluation appears very appealing, and the proof of concept results presented in this section seems to suggest that CPG parameter tuning by touch could represent a way to intuitively realize motions using a very simple and robust systems.

The actual implementation does not include feedback from the sensors or the gyroscope. Since we can expect entrainment with the environment and therefore a better



(a)



(b)

Figure 5.13: Pitch and roll for the crawling motion obtained with the proposed approach (a) and for the crawling motion obtained by a genetic algorithm (b).

stability and motion variation, future research will aim at including feedback without reducing the approach generality. The next chapter will provide two simple solutions to include feedback in the motion in a very general way. Obviously, for specific problems, custom use of the sensory feedback can probably give the best performances. However, we will show that with a very low computational cost, robustness during motion execution can be highly increased.

6 Increasing the robustness of motions

As previously stated, generation of humanoid robot motions is a complex task, mainly due to the high number of degrees of freedom, usually 20 or more, i.e. 4-5 times the number of degrees of freedom of a classical industrial robot arm. Furthermore in the case of small humanoid robots the on board computing power is often quite limited. For these reasons, usually at least part of the motion is generated off-line, by planning [168], optimization of parametric trajectories [169, 170, 13], motion retargeting of human motions [43], design from scratch [58, 53, 54] or other techniques, like the one introduced in this thesis.

To execute the motion, the most trivial approach consists in precomputing the joint trajectories or torques and replaying them while ignoring the sensory feedback [171]. Despite its disadvantages, feed-forward control is still largely employed for small humanoid robots, whose stability is usually improved by large footprints and a low center of mass position.

For tasks like walking [172], running [173] or crawling [174] specific techniques to exploit the sensory feedback in order to improve the robustness were developed. One commonly used approach is to modify the robot movement to control the Zero Movement Point [175], with the complexity of the controller ranging from the injection of torque into the ankle joints [176] to whole body movements [177]. These techniques allow robots which would otherwise be too unstable to be stabilized [178].

As previously seen, a biologically inspired approach consist of generating the motion using a dynamical system, termed as Central Pattern Generator [112, 113]. Usually CPGs are obtained by the interconnection of oscillators, which in turn often consist of an extensor and a flexor neuron pair [112] that controls one of the actuators. In this case the motion is implicitly described by the weight of the connections between the neurons (appropriate weights are usually set automatically by the policy gradient [113] or genetic algorithms [112]). Adaptation to the environment is obtained by interconnecting, with proper weights, the sensory inputs to the neurons that generate the motion: this feedback entrains the robot in stable limit cycles that depend both on the robot structure and the environment.

A recent, very appealing approach based on dynamical systems can be found in [179] while [180] shows how locally weighted regression can be used to exploit demonstrations to generate a suitable movements on-line for a large range of situations.

As can be easily imagined, generally speaking employing the feedback improves the performance over open-loop playback, as experimentally shown in [181]. In short, with open-loop the only information available when taking actions is the internal representation of time. Conversely, using sensory inputs the control algorithm can have a more complete knowledge of the current state and can take appropriate actions to achieve the desired robot state.

This however requires a design (or at least learning) of the relations between the sensory inputs and the control action to be taken. The approaches adopted are often both strongly task and robot dependent. Our aim is to devise general techniques that exploit the information coming from the sensors to improve the system robustness with minimal computational power requirements. Any comparison with specifically designed controllers (e.g. ZMP based) will be omitted, since we can easily guess that the task and robot specific knowledge exploited in the design phase can strongly improve the performance of those algorithms compared to our completely general techniques. The next sections describe two possible approaches to achieve the goal of an increased robustness with a minimal computational cost. The first approach is based on a graph that stores the motion as desired transition between states, and is used to try to bring the system to a known state as soon as possible. The second approach, inspired from Escherichia coli chemotaxis, works by random exploring the motor command space when things do not behave as expected. The second approach could be seen as a generalization of the first. In fact, the motion graph is based on the assumption that it should be easy to bring the system to a state that has small Mahalanobis distance from the current state but the second approach removes this heuristic.

6.1 A simple, graph based approach

The aim of the technique presented in this chapter is to correct deviations from the expected sensory inputs of an essentially stable systems due, for instance, to environmental changes. Drastic changes of the movement, inverse dynamics computation or online replanning such as using the hands for support during walking [182] are completely outside of our scope. We therefore assume to have perturbations in a range such that trying to track the sensory states is sufficient to stabilize the motion.

Specifically, we assume the robot to have the body covered by force sensors, as in [183] and the actuators equipped with potentiometers/encoders. Our approach consists in two phases. In the first one, the off-line phase, we play the motion in open-loop in a reference environment. We record the sensory information and store the transitions between sensory states as a graph. This graph becomes our motion representation. In the second phase, the online one, we use the information contained in the graph to replay the motion.

This is very similar to the StateNet approach [184] and presents the same advantages, namely, quantitative comparability between the states, easily extensible robot's knowledge representation, and capability of online search of correction strategies. However, while in [184] a transition is a complete movement like “Stand up”, our edges represent small state transitions and so a motion like standing up can be decomposed into dozens of states. This difference in granularity reflects our aim to make small corrections to cope with environment disturbances. In fact, [184] deals with complete behaviors and, for instance, if the robot falls down while walking then it will initiate a roll over, sit and stand up sequence. In our case, we want to restrict the recovery actions to small modifications of the motion to respond to disturbances.

Subsection 6.1.1 provides the details of the graph representation we designed. Subsection 6.1.1 explains the construction of the graph for a given motion, while Subsection 6.1.3 describes how the graph is exploited during the motion execution. Subsection 6.1.4 reports results obtained by experiments with a simulated robot, which suggest the capability of the proposed approach to strongly improve the motion robustness over plain motion playback.

6.1.1 Graph representation

The idea of storing multiple execution of a motion as a set of states is not new, and we can find examples both for the Computer Graphics field [185, 186] and in the description of robot behaviors [184]. Within our approach we assume to have a precomputed motor command sequence, obtained for instance through human-robot interaction. The motion is played open-loop in a controlled environment, the sensory information are sampled and a transition graph between the sensory states is constructed. Specifically, each node represents a hypercube in the sensory space, since we assume two sensory states to be mapped to the same node if their infinity norm[1] distance does not exceed a threshold Δ (in other terms, a node corresponds to a point in the sensory space with a confidence interval of size Δ). Each edge represents a motor command that causes a transition between states. Every edge is labeled by a weight that indicates the number of times that the transition was executed in the off line phase. In the online phase, at each sampling time we identify the node corresponding to the sensory information. If the state belongs to one of the nodes identified in the off line phase, we just execute the transition with the highest weight. If the node is unknown, we determine the nearest node explored in the off line phase and issue a command that should bring the robot to that state. Note that since we simply compare the postures of the two states and

[1] We chose to employ the infinity norm for its low computational cost. Any other norm should also be valid.

avoid any dynamics consideration, the system is not guaranteed to move to the desired state. As done in [184] we assume that moving between close states has a strong success likelihood. In case the resulting state is different from the planned one, the system will simply search once more the closest known state in the next step.

This is an important constraint in the choice of the sensory information contained in the nodes. In fact, we need to be able to estimate on-line a command able to bring the robot to the desired state (nearest known node hypercube center). One possible choice, which is what we actually did, is to include proprioception information (state of the actuators, e.g. joint angles) in the sensory state, so that the motor command to be issued to reach the desired state can be obtained by a transition from the current joint angles to the desired joint angles.

6.1.2 Graph construction

Let us denote the nodes and the edges of the graph $G = \{N, E\}$ respectively by $N = n_1, \ldots, n_P$ and $E = e_1, \ldots, n_Q$ where P is the number of nodes and Q the number of edges. Let us then indicate using s_i the sensory information corresponding to the center of the hypercube defined by the node n_i and by w_j the weight of edge e_j. Initially, the graphs consists of a single node, n_0, corresponding to the robot's initial state s_0 with no edges. The previous node n_* is initialized to n_0. At each sampling instant, the following algorithm

Read the sensory information $\hat{s}$
if $\|s_* - \hat{s}\|_\infty > \Delta$ **then**
    if n_k such that $\|s_k - \hat{s}\|_\infty \leq \Delta$ exists **then**
        if edge e_j between n_* and n_k **then**
            $w_j \leftarrow w_j + 1$
        else
            Add edge e_{Q+1} between n_* and n_k
            $w_{Q+1} \leftarrow 1$
        end if
        $n_* \leftarrow n_k$
    else
        Add node n_{P+1}, a hypercube centered at $\hat{s}$
        Add edge e_{Q+1} between n_* and n_{P+1}
        $w_{Q+1} \leftarrow 1$
        $n_* \leftarrow n_{P+1}$
    end if
end if

Note that self loops are avoided by the first control of the algorithm. In our implementation, the motor command information associated to edge e_j from node n_A to node n_B includes a histogram of the time spent on node n_A before taking the transition between node n_A and node n_B. This allows the representation of motions where the robot keeps still for some time, which would otherwise not be possible. For simplicity, we assume the sampling frequency to be high and the value of Δ to be small enough so that motor commands can be approximated by linear transitions between the angles stored in the nodes at the head and the tail of the edges. Under these simplifications the motor command associated with each edge can be simply derived by the angle difference between the centers of the hypercubes of the head and tail nodes. To avoid perceptual aliasing [187] occurring if we had motions that contain repeated sub-movements, we included time as state information stored in the nodes (note that for periodic motions the time returns to 0 at every cycle). Summarizing, the motion description we obtain from the off-line phase is a graph where

- Nodes represent the robot state. Each node contains information on
  - joint angles (as measured by the simulated encoders)
  - forces acquired by the tactile sensors (simulated Force-Sensing Resistors that provide the perpendicular forces applied to each surface)
  - time instant of the motion execution
- Edges represent state transitions. Each edge contains information on
  - the number of times the transition was taken (stored as the node weight)
  - a histogram of the time taken to perform the transition

Given the heterogeneity of the units of the node, these are normalized by normalizing factors. In detail, joint angles and forces are divided by 3 times the standard deviation of the sensor errors and the time instant by 50 ms (assuming this is a lower bound under which no perceptual aliasing occur).

6.1.3 Motion reproduction

The graph obtained by the off-line phase constitutes a robust motion representation that contains in a unified framework sensory information together with motor commands. During the motion execution, a "virtual time" is used to avoid forcing the motion evolution to have the same period as the trajectories used to generate the graph. To state it more exactly, when a motor command is issued, the current time is assumed to be the time of the target node, whether the target state was actually reached or not. This virtual time is also used to avoid loops: assume the robot is in state A and tries to reach state B, but ends up in state C. Imagine then that B is the closest known node from C, and that trying to reach B makes the robot remain in state C. Forcing the algorithm to choose the closest node with a time higher than the virtual time is sufficient to avoid the robot to get stuck in such loops. The simple algorithm used for motion reproduction at each sensor sampling time is reported in listing 1. t^* denotes the virtual time, while n_i^t indicates the time of node n_i. To allow periodic motions, the transitions are accepted also toward nodes with a highly smaller times, so that periodic motions can "restart" (case $or\ n_i^t - t^* < -\Theta$ in the algorithm). The value Θ is set to the 90% of the period in our implementation.

Algorithm 1 Motion reproduction algorithm

Read the sensory information $\hat{s}$
$c \leftarrow \arg_i \min \|s_i - \hat{s}\|_\infty$
if $\|s_c - \hat{s}\|_\infty \leq \Delta$ **then**
  $h \leftarrow \arg_j \max\{w_j : e_j\ departs\ from\ n_c\}$
  execute the motor command e_h
  $t^* \leftarrow n_b^t$ where n_b is the node pointed by e_h
else
  $d \leftarrow \arg_i \min \|s_i - \hat{s}\|_\infty\ s.t.\ n_i^t > t^*\ or\ n_i^t - t^* < -\Theta$
  issue a motor command to reach state s_d
  $t^* \leftarrow n_d^t$
end if

As described in the previous subsection, in our implementation each edge stores a histogram of the time spent to execute the state transition. When edge e_j is chosen the reference angles sent to the motor's PID controllers vary (linearly) from the position specified in the tail node to the position specified in the head node. The time over which this variation is performed is equal to the mode of the time statistics saved in e_j.

When the system is in a state not belonging to the nodes generated during learning (else branch in the previous algorithm), a motor command corresponding to a variation between the current angles and the angles specified by the desired state s_c is issued. In this case, the motor transition is executed over a constant time T_C (50 ms in the current implementation).

We can soon notice that it is possible to prune the graph before the motion replaying, leaving a single outgoing edge from every node (the one with the highest weight) and storing just the mode instead of the complete waiting time statistics. In this case, the complexity of the algorithm becomes $\mathcal{O}(P)$, i.e. linear in the number of nodes, since at each sampling time we need to search the closest node. In our experiments, nodes were always in the order of few hundreds, so the computation is feasible even with low computational power. If P increases and the computation becomes too slow, the Best Bin First algorithm [188] can be employed.

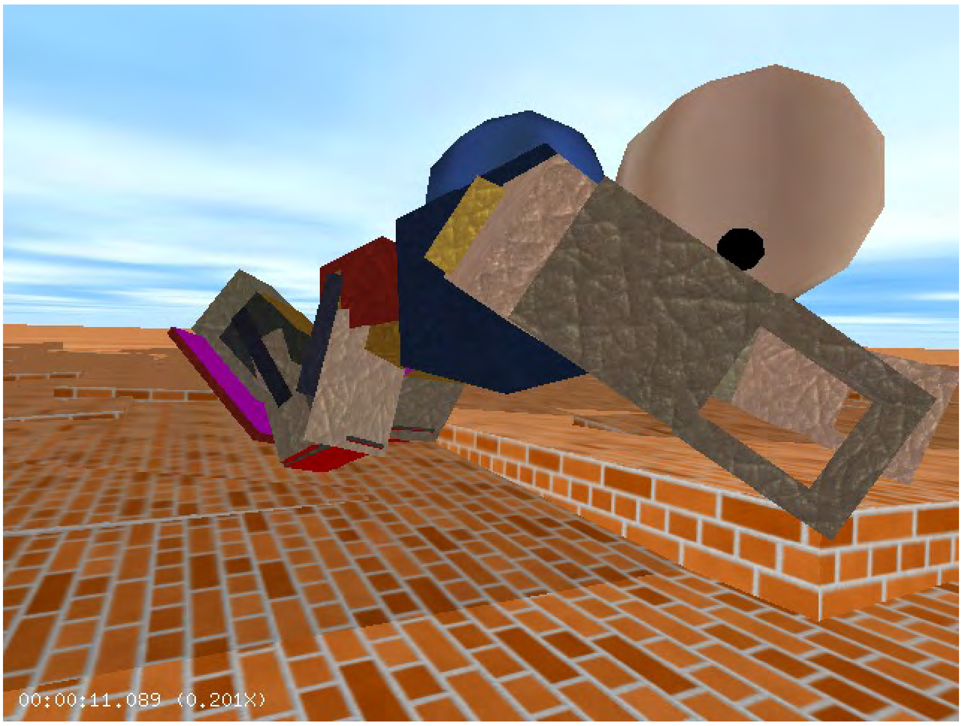


Figure 6.1: Simulation of M3-Neony, a robot with 22 degrees of freedom and touch sensors over the whole body, crawling on a rough terrain.

6.1.4 Experiments

The described algorithm has been tested by obtaining a graph representation of a crawling motion performed by a humanoid robot. The experiments were conducted by employing a physics simulator, shortly described in Appendix A, and a model of M3-Neony, as visible in Fig. 6.1.

Specifically, we developed a motion from scratch using a frame based motion editor (i.e. the robot motion is defined as a set of key-frames, that is, a set of instants in time for which the position of each and every joint is provided). We executed the motion 100 times on a flat terrain and constructed a graph with the algorithm described in Subsection 6.1.2. Even if the simulator does not include an explicit generation of sensor noise, numerical approximations, random generation of the contact points between surfaces and a damping of the lateral oscillation generated by the first step prevents the various executions from being identical. The resulting graph, obtained for $\Delta = 2$, consists of 66 nodes and 158 edges. Figure 6.2 reports its structure. We then conducted three experiments

1. We verified whether the graph representation is able to store the original, trajectory based representation. In other terms, we replayed the motion using the graph in the environment used for graph construction.

2. We verified the robustness of the approach under several environmental changes.

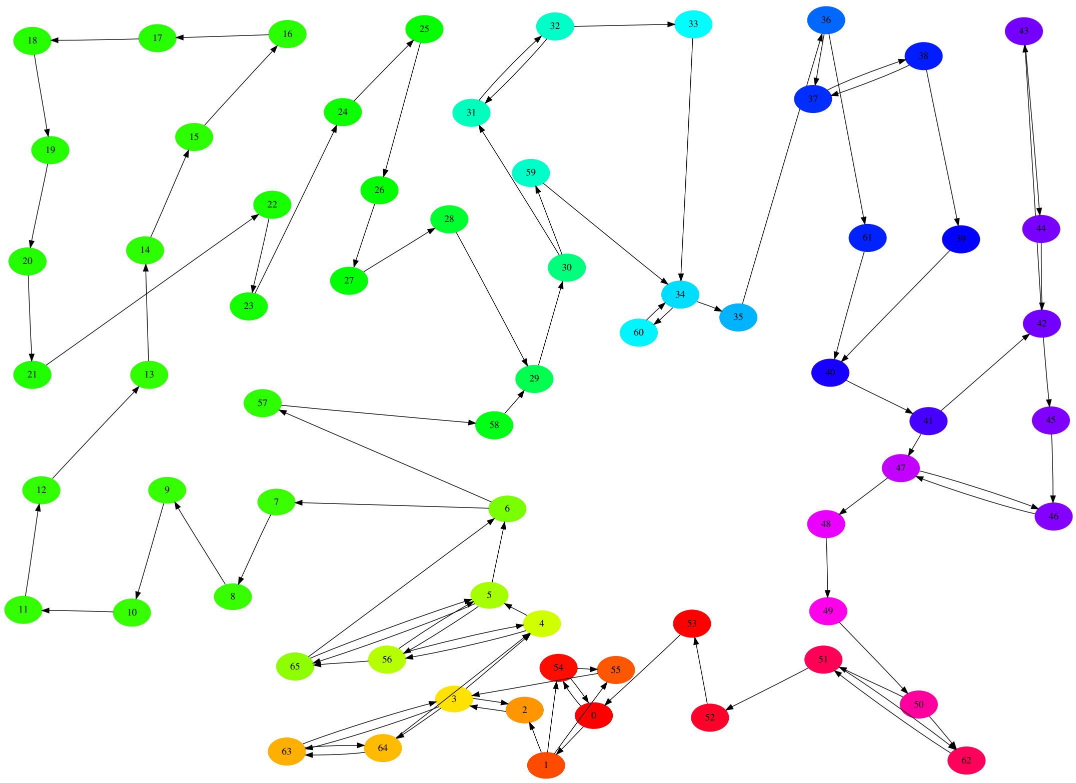


Figure 6.2: Graph representation obtained for the crawling motion. The color hue indicates, using the HSV scale, the virtual time stored in each node.

3. We added noise to the servomotor signals, in order to get the robot unstuck on very rough terrains, and verify that also in this case we can take advantage from the graph representation over the feed-forward solution.

We defined the crawling motion performance as the average velocity of the robot. Expressly, since during the execution of the crawling motion, lateral swings are present and therefore instantaneous velocity of the center of mass is not very meaningful, we defined the velocity at time t as

$$v(t) = \frac{\left\| \int_{\tau=0}^{T} X(t-\tau)d\tau - \int_{\tau=T}^{2T} X(t-\tau)d\tau \right\|}{T}$$

where X(t) is the (3D) position of the robot's center of mass at time t, T is a constant (4 seconds) and Euclidean norm is used.

6.1.4.1 Motion replay

We verified practically that the graph representation is sufficient to store the crawling motion, as well as other motions like sitting down, turning over, crawl turning left and right. In particular, unexpectedly, we could see that the use of the graph representation strongly improves the performance even if the environment used to replay the motion is identical to the one used to construct the graph: the average velocity (measured over 4 minutes) is 5.25 cm/s for the open loop execution and 7.86 cm/s for the graph based approach (150%). This is quite surprising if we consider that the motions was hand tuned for this environment.

The performance improvement is essentially due to two factors. First, oscillations are quickly damped using the graph, in fact if we calculate the variance of the roll angle during the motions we get 0.0084 rad^2 for the open-loop case and 0.0047 rad^2 for the graph based control. Secondly, the average crawling period is reduced. Both these effects are due to "shortcuts" on the graph, which are performed when the robot reaches unexpected states, that make the robot skip small parts of the graph. Figure 6.3 shows the nodes and paths visited in the online phase. Figure 6.4 shows part of the graph, relative to the transition between states 53 and 5. For instance, the open-loop control visited states 53, 0, 54, 0, 1, 54, 55, 3, 63, 64, 63, 3, 2, 3, 4 and 5. The short loops that start and end in states 0 and 3 lasted, respectively 6 and 39 milliseconds, and the complete transition between states 53 and 5 took 297 ms. Using the graph, the robot arrived at state 53 and moved to state 54 (reached after 46 ms), planned to go to state 0 but ended up in state 4 (at time 96 ms) and then proceeded to state 5 (reached 158 ms after the reaching of node 53). In this case, therefore, the small loop among nodes 3 63 and 64, that consists of nodes where the robot doesn't move the motors but oscillate laterally due to the dynamics, was skipped. This didn't affect the motion, but, conversely, improved the performance. Note that the inclusion of time in the nodes to

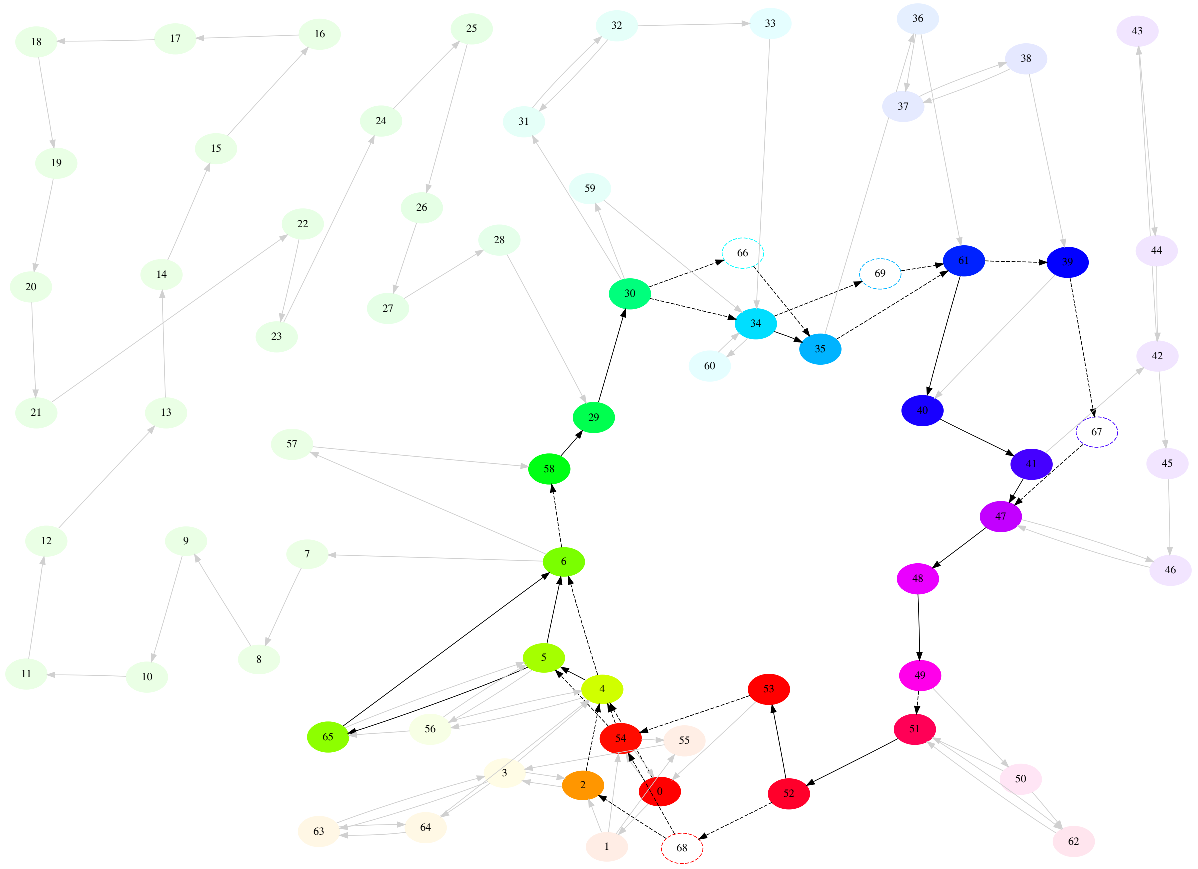


Figure 6.3: States visited during the online phase in the environment used for graph generation. Filled nodes and solid line edges belong to the knowledge generated offline, while empty nodes are states reached only in the online phase. Color (hue) indicates, using the HSV scale, the creation time of each node. Vivid color (high saturation) for the filled nodes indicates that the node was visited during the online phase, faded color indicates that it was not.

prevent state aliasing also prevents clamping big loops that describe meaningful parts of the movement (for example one complete step with a foot) since no "shortcuts" will be created between nodes that have similar touch sensor and angle information but very different times. However, future works will investigate whether the assumption that "small" loops can be neglected is true, and will possibly introduce on-line learning that automatically identifies the necessary loops.

6.1.4.2 Motion replay under different conditions

To verify the robustness of the approach, we compared the performance of the open loop motion replay and the graph based execution under several environmental changes:

- Reduction of the friction coefficient with the floor to the 10% of the reference value: open loop crawling velocity 4.84 cm/s, graph based 8.12 cm/s (168%). Fig. 6.5

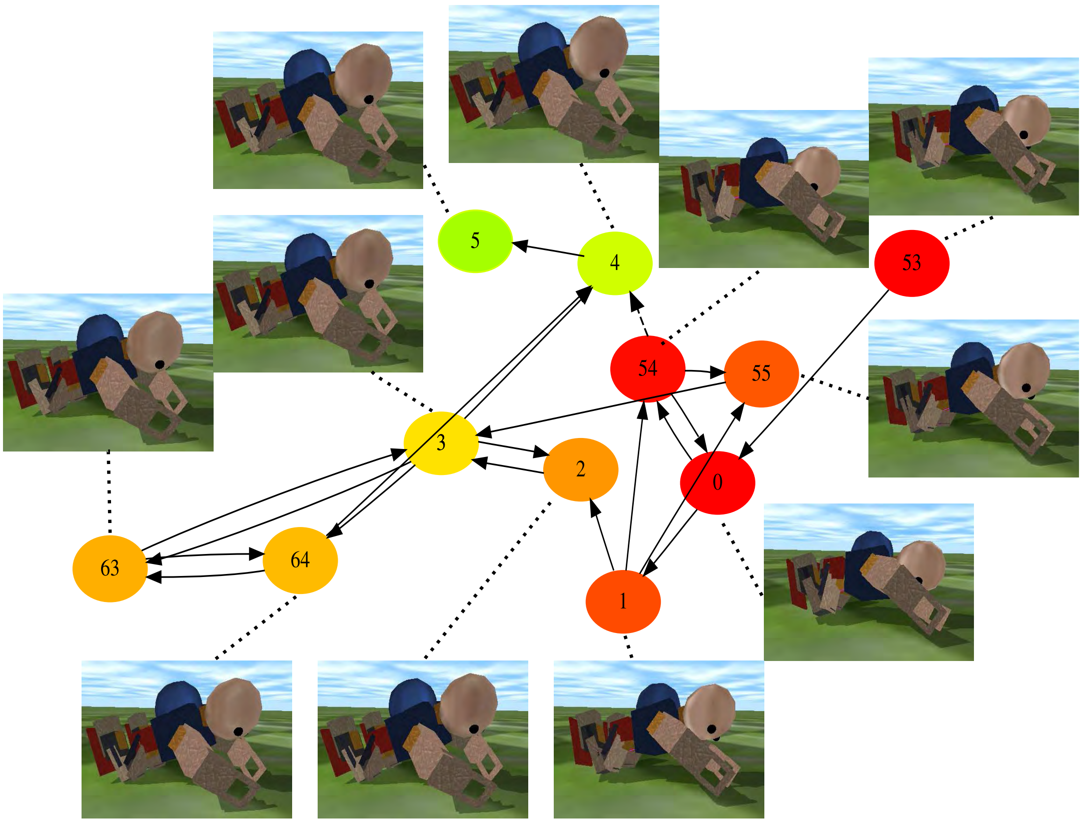


Figure 6.4: Part of the graph, executed when replaying the motion in the environment used during the graph construction. Postures associated to each node are displayed by a picture linked by a dotted line.

reports the nodes visited during the movement execution.

- Halve the robot's weight: open loop crawling velocity 4.62 cm/s, graph based 7.22 cm/s (156%). Fig. 6.6 reports the nodes visited during the movement execution.

- Walk on an ascending slope of 20 degrees: open loop 2.81 cm/s, graph based 5.79 cm/s (206%). Fig. 6.7 reports the nodes visited during the movement execution.

- Walk on a rough terrain: open loop 0.54 cm/s, graph based 1.37 cm/s (249%). Fig. 6.8 reports the nodes visited during the movement execution.

The rough terrain, depicted in Fig.6.1, is obtained in the following way:

1. construct a square grid of 50 cm interspaced points on the floor

2. place a tile of $70 \times 70 \times 14$ cm centered in each of the grid points

3. apply a random translation on the X-Y (floor) plane of each tile (uniformly distributed in the range $[0, 30]$ cm)

4. change the pitch and roll of each tile by a random angle (uniformly distributed in the range $[0, 3]$ degrees)

6.1.4.3 Addition of noise to avoid stuck conditions

The performances on very rough terrains like the one previously described can be strongly improved if noise is added to the servomotor commands when the robots gets stuck. We decided to add Gaussian noise with standard deviation that is a decreasing function of velocity. This is very similar to what happens in biological systems: the actions taken are more and more deterministic the better the conditions are, and the more stochastic the worse the conditions are. For instance, E. Coli proceeds by alternating forward movements and random tumbles, and decreases the frequency of random tumbles when the environment conditions are favorable [189, 190] (see Subsection 6.2.1 for further details). In particular, we decided to use a simple piecewise linear function to map velocities to the noise standard deviation

$$\eta(v) = \begin{cases} \alpha & if\ v \leq v_\alpha \\ \alpha + \frac{v - v_\alpha}{v_\beta - v_\alpha} * (\beta - \alpha) & if\ v_\alpha < v \leq v_\beta \\ \beta + \frac{v - v_\beta}{v_\beta - v_\gamma} * (\gamma - \beta) & if\ v_\beta < v \leq v_\gamma \\ \gamma & if v > v_\gamma \end{cases}$$

We determined a good set of values for this piecewise function, namely $\alpha = 0.035$, $\beta = 0.014$, $\gamma = 0.0064$, $v_\alpha = 3.8$, $v_\beta = 5.9$ and $v_\gamma = 6.4$ using a genetic algorithm (population size 20, 50 generations). Units are cm/s for velocities and radians for the standard

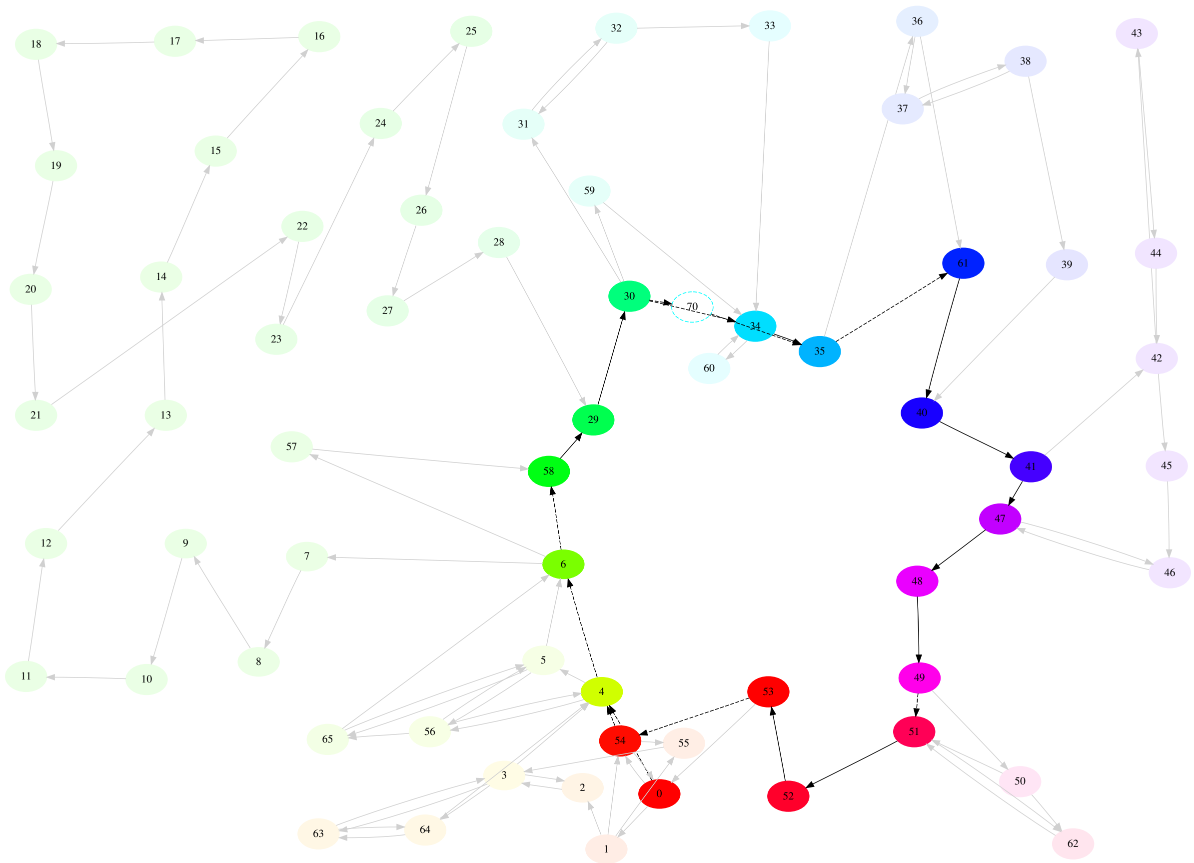


Figure 6.5: States visited during the online phase in an environment with different ground friction. Filled nodes and solid line edges belong to the knowledge generated offline, while empty nodes are states reached only in the online phase. Color (hue) indicates, using the HSV scale, the creation time of each node. Vivid color (high saturation) for the filled nodes indicates that the node was visited during the online phase, faded color indicates that it was not.

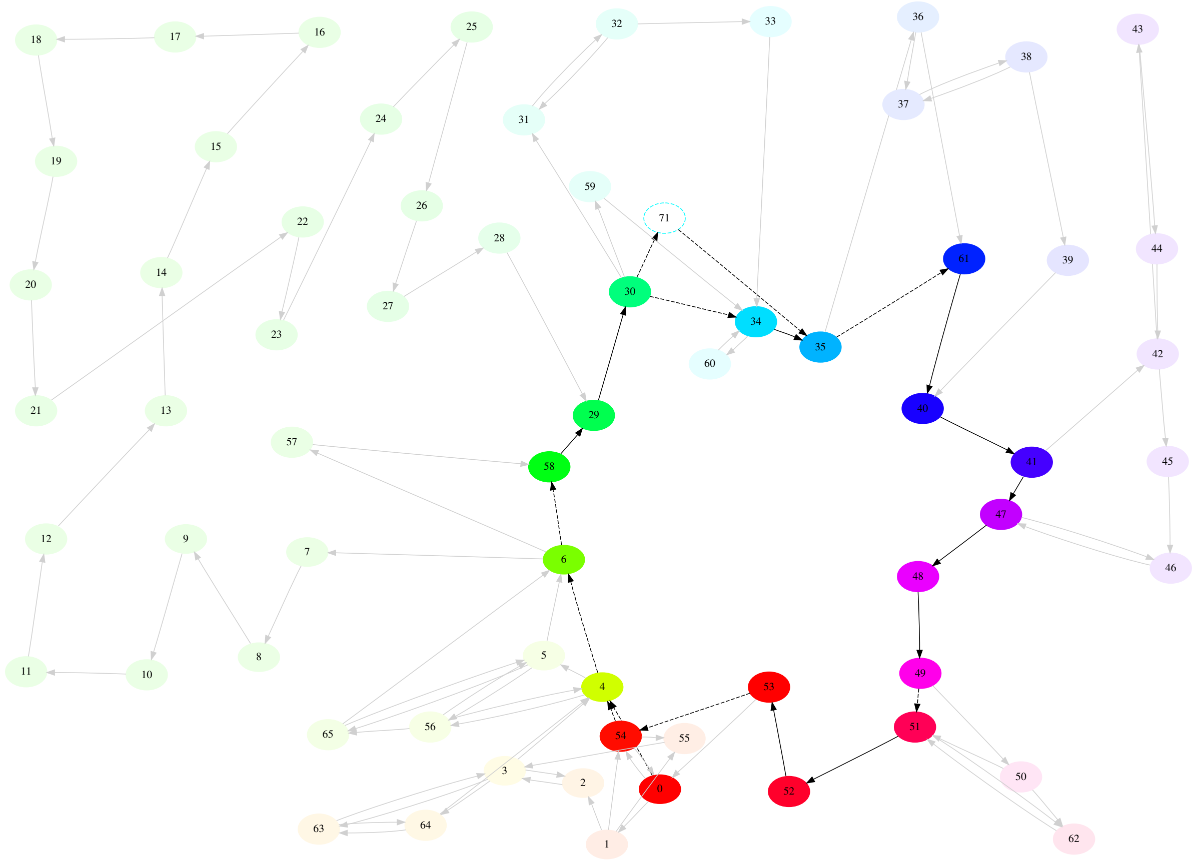


Figure 6.6: States visited during the online phase when the robot's weight is halved with respect to its weight in the graph generation phase. The notation is the same as Fig. 6.5.

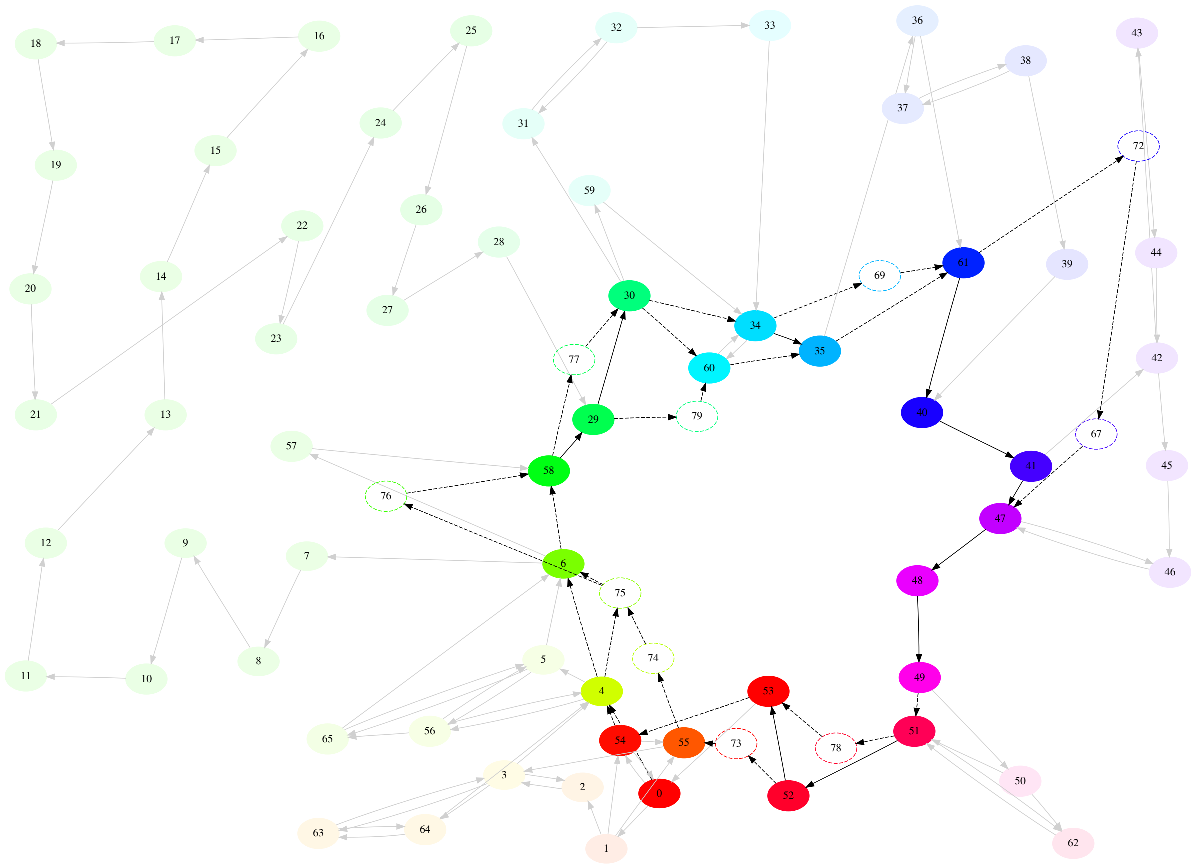


Figure 6.7: States visited during the online phase when the robot crawls over a slope. The notation is the same as Fig. 6.5.

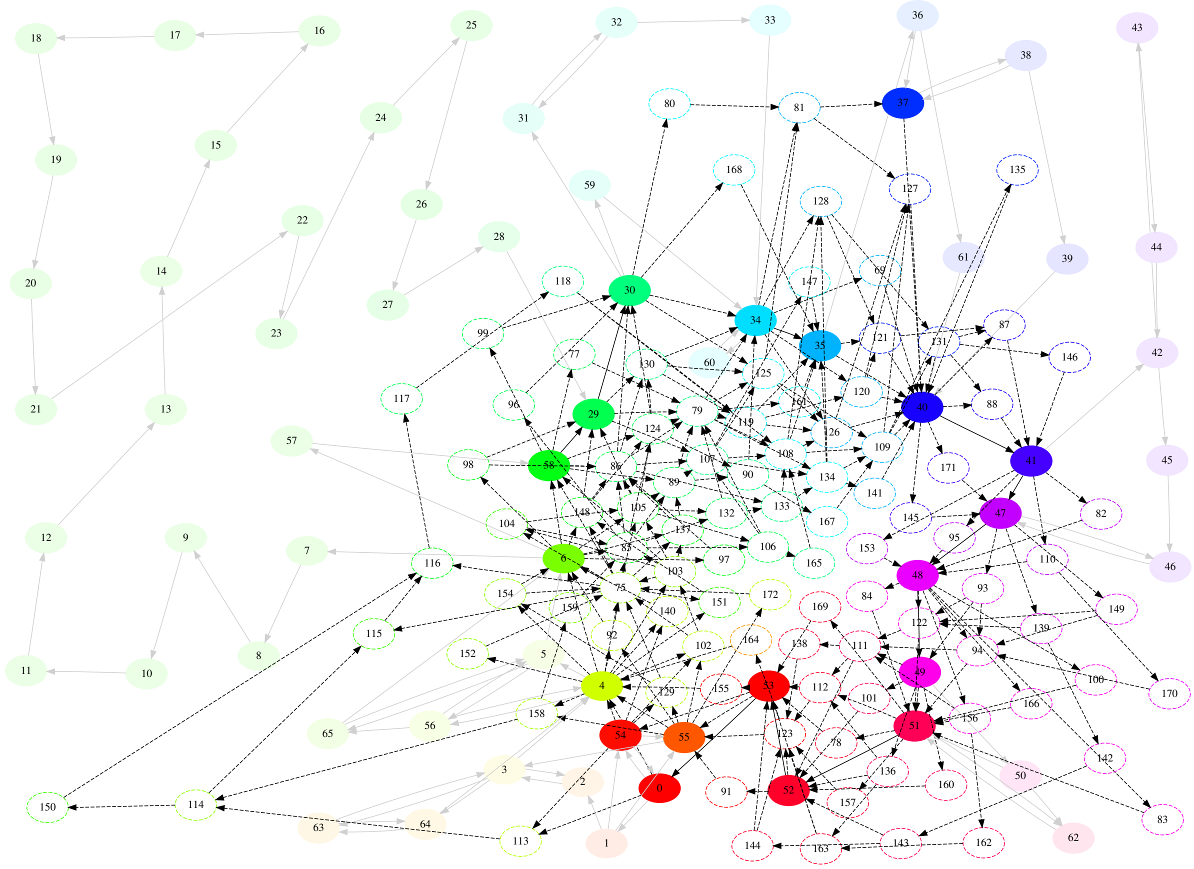


Figure 6.8: States visited during the online phase when the robot crawls over the rough terrain described. The notation is the same as Fig. 6.5.

deviation (we apply noise to the motor target angle). Before executing each motor command the robot's velocity v_t is evaluated (with the definition of instantaneous velocity previously provided) and the target angles are perturbed with Gaussian noise of zero average and standard deviation $\eta(v_t)$. To have a fair comparison between the graph based representation and the open loop execution we measured the average state transition time for the graph case and this resulted to be $t_T = 45ms$. We then played the open loop motion approximating it by linear transitions (over time spans of $t_T = 45ms$) between angle postures, where in each interval the target posture is given by the posture specified by the "nominal" trajectory plus a noise given by the $\eta(v_T)$ function. We repeated the experiment for 30 times both for the open loop and graph based representation, obtaining mean velocities of 1.2cm/s (standard deviation 0.94) and 1.9cm/s (standard deviation 0.94), respectively.

In conclusion, the presented technique provided a big difference in the performance with respect to the open-loop system in several tests. We do not believe the technique has better performances than specifically designed controllers, e.g. ZMP based stabilizing controllers. Nonetheless, the approach here presented is general and does not require any knowledge of the robot's structure or dynamics. Future works will need to test whether good performances are achieved in other tasks as well, and how much the tuning of the piecewise linear function that adds noise to the system influences the task performances.

A possible improvement of the current system is introducing learning during the motion execution, for instance in order to adapt to a different environment. This could be realized by avoiding the graph pruning and updating the graph structure during the on-line phase. However, if such modification is included, care must be taken not to "forget" the previously acquired knowledge, because, for instance, changes on the edge weights could prevent the robot from executing parts of the motion.

Finally, it is often very reasonable to assume that states at a short Euclidean distance are easily reachable from each other. However, this heuristic constitutes one of the strongest generality limitations of the approach. The next section presents two new algorithms, based on bacteria chemotaxis, that do not require this assumption to be true.

6.2 A biologically inspired algorithm

In the previous section, an algorithm for improving motion robustness based on motion graph was presented. Within the presented approach, when the robot faces an unknown state, it tries to recover to the closest known state. The algorithm presents some limitations. In particular, the definition of "close" is simply "close in terms of Euclidean distance", while, depending on the task, specific measures could be required. Furthermore, the recovery is possible thanks to an appropriate choice of the sensory space, that includes the proprioceptive information coming from the joint potentiometers. This section presents two new algorithms, that remove this assumptions and provide much more general approaches. In particular, the following subsection will present Escherichia Coli chemotaxis, the natural phenomenon that underlies the design of two algorithms. Subsection 6.2.2 will describe a version of the algorithm based on biased random walk. Experimental results on the approach will follow in Subsection 6.2.3, showing that the algorithm is very robust to hardware faults and sensors noise. Some possible improvements in the generality of the algorithm will be reported in Subsection 6.2.4, and a new version of the algorithm will be provided in Subsection 6.2.5. Comparisons of the two algorithms will be provided in Subsection 6.2.6. Finally, Subsection 6.2.7 will present preliminary experiments obtained with a simulated humanoid robot.

6.2.1 Bacteria chemotaxis

One of the simplest behaviors found in nature is *chemotaxis*, the process by which bacteria [189, 191, 192] or eukaryotic cells [193, 194, 195, 196, 197] sense chemical gradients and move with directional preference toward food sources.

Among the most thoroughly studied organisms with regards to chemotaxis we can certainly cite Escherichia coli (E. coli), often taken in consideration due to its well-characterized physiology, its simple chemotaxis signaling pathway, its capacity to sense small concentration gradients of a chemoattractants, and its possible use as either a natural host or a surrogate host for plasmids coding of desired features or products [198].

This bacterium moves by alternating clockwise and counter-clockwise rotations [189]. When it rotates counter-clockwise the rotation aligns its flagella into a single rotating bundle and it swims in a straight line. Conversely, clockwise rotations break the flagella bundle apart and the bacterium tumbles in place. The bacterium keeps alternating clockwise and counterclockwise rotations. In absence of chemical gradients, the length of the straight line paths, i.e. the counter-clockwise rotations, is independent of the direction. The movement thus results in a random walk.

In case of a positive gradient of attractants, like food, E. Coli instead reduces the tumbling frequency. In other terms, when the concentration of nutrients increases, the bacterium proceeds in the same direction for a longer time. This strategy allows to bias the overall movement toward increasing concentrations of the attractant. Such a simple mechanism works despite the difficulties in precisely sensing the gradient. Actually, the spatial gradients in concentration cannot be sensed directly due to the small dimensions of the bacteria, so temporal difference in the concentration is used to estimate the nutrient distribution.

A wide spectrum of models, from a very abstract point of view to the modeling of the protein interactions, is available in literature [199, 200, 201].

Few studies however regard the effect of noise on the chemotactic behavior. Furthermore, the noise is seen mainly as a nuisance in sensing that should be filtered out by the bacterium [202, 198], similarly to what is usually done in control theory.

Using two very general models, instead, we show that external noise can actually increase the performance. The first model, described in the next subsection, is based on biased random walk, with a bias that is a function of the previous bias and of whether the conditions are improving or not. The second one, designed to overcome some limitations of the first biased random walk algorithm, is presented in Subsection 6.2.5 and analyzed in Subsection 6.2.6.

6.2.2 Algorithm

When considering challenging environments like forests [203], planetary explorations [204] or game fields where the robots can collide to each other, it is almost impossible to predict in advance all the problems which could arise in the task execution and the possible failures the robot hardware might encounter. However, the current technology seems still to be lacking in providing reliable robots able to cope with hardware failure or uncertainties. Most techniques require a previous identification of the possible accidents that can occur to the robot and the design of specific workarounds for each of them. Some advanced self-modeling techniques were presented [205] but they request intensive computation to estimate the current status of the robot from the sensor information.

Conversely, as highlighted in the previous subsection, very simple living beings like bacteria can cope with very complex and variable environments, and often present a highly adaptive and robust behavior despite their structural simplicity.

A simple possible modeling of the bacteria behavior is by random walk. Actually this model has already been applied for controlling a mobile robot [206, 207]. The experiments in [207] show that while gradient descent is faster for tracking a single source, the biased random walk performs better in the presence of multiple and dissipative sources and noisy sensors and actuators. Demonstrations of the effectiveness of introducing a random term in the algorithm for preventing the robot from ending up in local minima are also provided.

However, the robustness to hardware damages and noisy sensory information achievable by biased random walk are not fully exploited. Expressly, in [207] the hardware already provides two basic movements, proceed straight and change direction randomly, and the biased random walk is performed at the behavior level. This approach limits the robustness for unexpected hardware failures. In fact, if due to hardware failures one of these basic movements does not operate as expected in many cases it will not be possible to accomplish the tasks. Imagine for instance to have a mobile robot with two wheels powered by independent motors and that due to an encoder problem one of the motors starts to rotate in the opposite direction. In this case when the "go forward" motor command is provided the robot spins around itself and the target will never be reached, as schematized in Fig. 6.9.

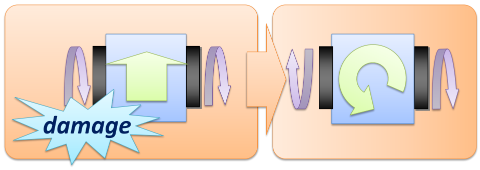


Figure 6.9: Example of damage for which the robot will not be able to reach the goal if the control bases on fixed low level behaviors. If a robot encoder damages, and the wheel starts to rotate in the opposite way, then the "go forward" behavior will become a spinning behavior, which is unable to bring the robot to the goal.

We propose, instead, to perform a biased random walk directly in the *motor command space*, i.e. the behaviors themselves are determined online through the random walk. This gives great robustness in case of hardware failures since new behaviors that exploit the current hardware behavior are found online in an automatic way.

In general, performing a random walk in the motor command space allows determining at runtime how to exploit the dynamics of the hardware, which can change due to hardware failures. By using this approach there is no need to explicitly identify the failure and use preprogrammed alternative behaviors, which can be difficult to design beforehand [208]. Despite the extreme simplicity and the absence of any information on the hardware this proposed bio-inspired method is able to provide high adaptability and robustness. We would like to stress that other, more task specific techniques could probably be designed to achieve higher performance for specific task, but what we are interested in achieving is a biologically inspired approach that gives robustness to hardware failures and sensor noise.

Concretely, the control of the robot can be reduced to a single equation

$$u_t = u_{t-1} + b_t + \beta\eta_t = u_{t-1} + \alpha sgn\left(\Delta A_t\right) \frac{u_{t-1}}{\|u_{t-1}\|} + \beta\eta_t \tag{6.1}$$

where $u_t \in \mathbb{R}^m$ is the motor command control signal provided at time t. The modification applied to the control signal is composed of two terms, a bias term $b_t = sgn\left(\Delta A_t\right) \frac{u_{t-1}}{\|u_{t-1}\|}$ and a purposely added random perturbation $\eta_t \in \mathbb{R}^m$, multiplied respectively by two scalar coefficients α and β.

The quantity $\Delta A_t \in \mathbb{R}$ appearing in the bias term expresses the variation of the "quality" of the robot state during the t-th time step. For instance, if the task consists in reaching a target with a mobile robot, the quantity ΔA_t represents how much the robot got closer to the target from time $t-1$ to time t. Therefore, essentially, the bias term states that if the robot got closer to the target using command u_t then command u_{t+1} should be similar. The second term is the most interesting part. In fact, it can be shown that by adding random perturbations of opportune magnitude to the control input, it is possible improve the performances.

Figure 6.10 reports an example. Let us suppose to have a holonomic robot initially

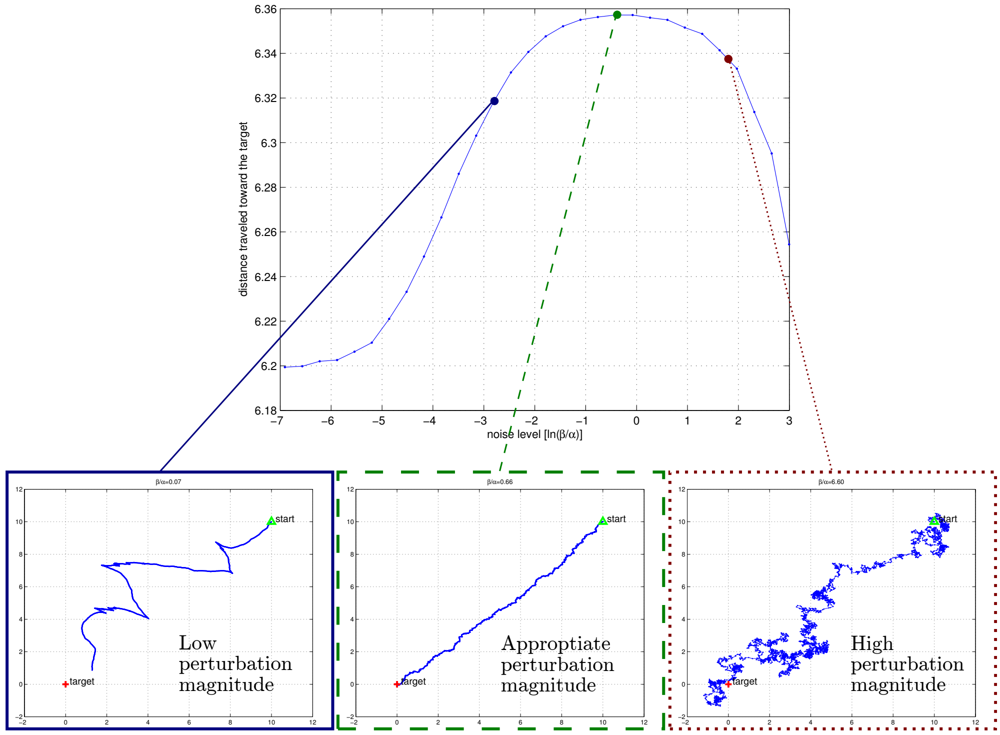


Figure 6.10: Performance for different perturbation magnitudes. The top panel reports the performances for different values of β when $\alpha = 10^{-2}$. The performance is measured as the average distance traveled toward the goal in 1000 time steps, calculated over 10000 simulations. The bottom panels report examples of trajectories obtained for different values of β. Precisely, the first trajectory was obtained for a low perturbation level, $\beta = 0.07$, the second trajectory corresponds to an opportune level of perturbation, $\beta = 0.66$ and finally the bottom right panel reports a trajectory generated with $\beta = 6.6$.

placed at $[10, 10]^T$ (arbitrary units) that has to reach a target placed at $[0, 0]^T$, and the motor command to be simply the translations along the two axes. If the perturbation is too small, then the effect of the bias will be strong, and even nearly tangential movements that bring the robot slightly closer to the goal will be used for a long time. If the perturbation is too big then the robot changes direction too frequently, even when the movement is headed straight to the goal. If the perturbation amplitude is appropriate, then the robot will reach the goal with a good trajectory [2]. This result is confirmed observing the distribution of the robot heading direction compared to the optimal direction in Fig. 6.11. When the perturbation is too little, essentially all headings that do not face backwards are chosen with uniform distribution. When the noise is too big, the robot tries all directions, with a small bias on the good ones. Finally, when the perturbation level is appropriate the distribution has a peak in the optimal heading.

Actually, the performance depends essentially just on the ratio between the two

[2]A stochastic resonance effect can be observed. For details see [209, 210, 211].

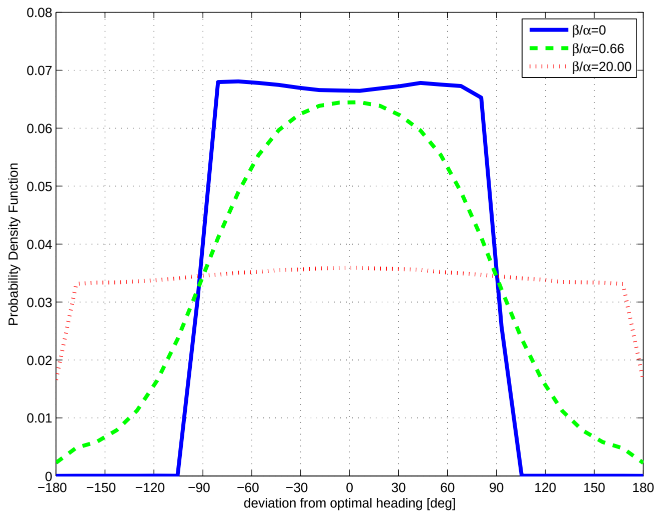


Figure 6.11: Distribution of the robot heading, compared to the optimal heading, for different perturbation levels. The blue continuous graph reports the distribution for a low perturbation magnitude, the green dashed graph represents the probability density function obtained for an opportune perturbation coefficient ($\beta = 0.66$) and the third, red dotted curve, shows the distribution of the heading for an high perturbation level ($\beta = 20$).

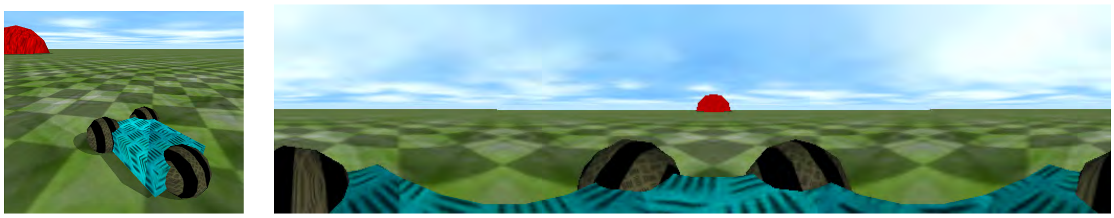
coefficient α and β, and not on their values itself. However, the optimal ratio depends on the hardware and environment conditions, as shown in the following subsection.

6.2.3 Experiments

In order to verify the feasibility of the algorithm, simulation and real world experiments were conducted. In detail, to keep the problem simple and ease the result analysis, wheeled robots were employed. Using ODE[3], we simulated a mobile robot equipped with three spherical wheels. The two front wheels are directly actuated by two independent motors whose maximum velocity is 0.5 rad/s while the rear wheel is free to rotate in any direction. The task is to reach a red hemisphere having radius of 4 m and placed at a distance of 30m. The robot is equipped with an omni-directional camera (see Fig. 6.12), and the number of red pixels perceived by the camera is used as an estimation of the distance to the goal. In particular, given the RGB components of a pixel, this is considered red if its R component is more than double the maximum of the G and B components. The number of pixels considered red is used as the term A_t in Eq. 6.1. We notice that since A_t appears just in the term $sgn\,(\Delta A_t)$, the only information used by the algorithm is whether the red pixels in the camera image are increasing or not.

The algorithm of Eq. 6.1 is run with a frequency of 0.2Hz and provides a 2 dimensional

[3]Open Dynamics Engine, a free library for simulating rigid body dynamics. For details see `http://www.ode.org`.



(a) (b)

Figure 6.12: Scene rendered by the simulator and the same scene as seen by the robot camera.

velocity command u representing the velocities of the two wheels. We chose to employ such a low sampling frequency to validate the robustness of the method even in the case of low cost hardware with very poor performances, as in the case of a cheap webcam connected to an outdated notebook.

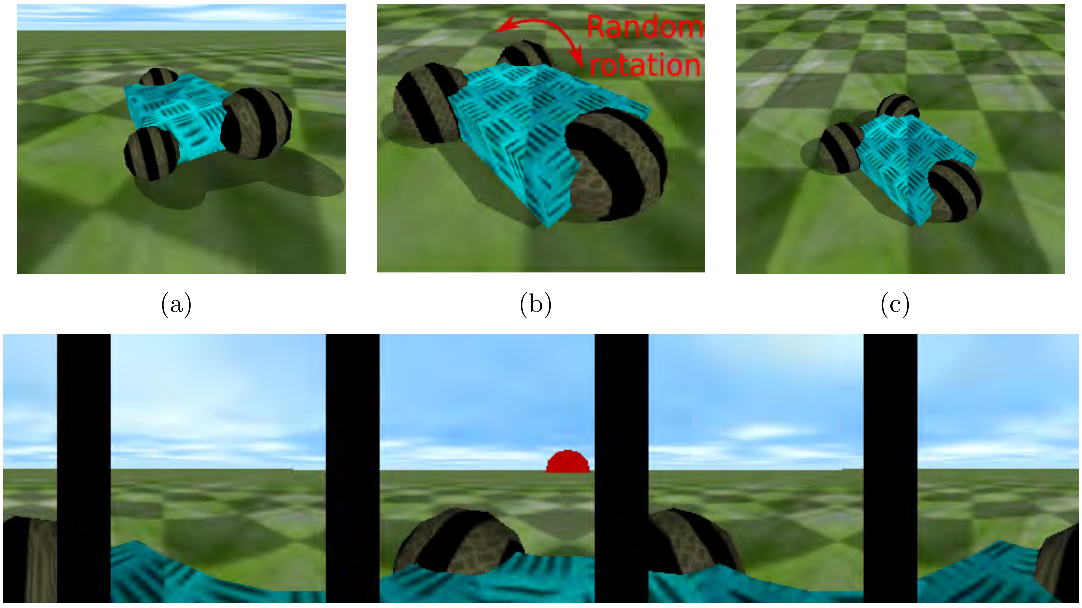
To test the robustness of the algorithm in driving the robot to the goal we simulated four types of damages (see Figure 6.13):

1. the right wheel size becomes 1.5 its normal size;
2. the right wheel becomes uncontrollable, i.e. its movement is completely random;
3. the right wheel rotation axis direction is turned 90 degrees along the Z axis and becomes parallel to the longitudinal axis, i.e. instead of making the robot move forward or backward, the rotation of the wheel causes the robot to move left or right, sideways;
4. 20% of the camera image becomes black.

We assumed $\eta \sim \mathcal{N}(0,1)$ as a Gaussian variable of variance one and studied the behavior for several values of α (the scaling coefficient of the bias) and β (the scaling coefficient of the random perturbation). In particular, for each condition (no damage or one of the damages listed) we determined the time spent contacting with the goal in 20000 simulated seconds. Each condition is simulated one time for 128 different positions of the target, in particular assuming the robot's chassis is placed at (0,0) we set the target in each of the positions as $(R \cdot cos(\theta_i), R \cdot sin(\theta_i))$, $\theta_i = \frac{i \cdot 2\pi}{128}$, $i = \{0, \ldots, 127\}$ where $R = 30$ m.

In analogy with the examples reported in Fig. 6.10, the bias should be strong enough to drive the robot toward the target in short time but small enough to avoid performing the same action if the reduction of the distance to the goal is too little. We can expect therefore the existence of an optimal value of the bias.

In other terms, if the α/β ratio is too high, then even if the velocity becomes nearly tangential to the path that leads to the target, the robot will tend to proceed in that direction, which results in traveling long paths that only achieve a short reduction of the



(a) (b) (c)

(d)

Figure 6.13: The four damages simulated. (a) Variation in the size of one wheel. (b) Uncontrollability of one wheel (c) Rotation of the axle of one wheel (note the direction of the stripes) (d) Obscuration of part of the acquired image.

distance to the goal. Conversely, if the α/β is too low, then although the bias eventually drives the robot to the target, a long time may be required to reach the goal because of unnecessary path deviations.

We can ask ourselves whether the optimal bias is constant or changes depending on the hardware damage. The reaching experiment was therefore repeated for different settings of the values of α and β both for the undamaged robot and for the four robot conditions described beforehand.

Figure 6.14 depicts the results for the undamaged robot and for the four damages previously listed. The x and y axis indicate the values of α and β respectively, while the color represents the performance, in terms of ratio between the time spent touching the target and the total simulation time (20000 seconds). For all damages the graphs presents non zero values, i.e. the robot is able to reach the target and touch it. It is worth to notice that some damages, especially the rotation of the axle of one wheel, completely changes the effect of motor commands. However, our simple algorithm is able to identify new efficient motor commands on the fly, without the need of any failure detection.

As expected, a completely deterministic behavior ($\beta = 0$) is often not able to drive the robot to the target, since, without "exploration" of motor commands done by the random part, the system can just provide a single type of motor command. Similarly, when $\alpha = 0$ the probability of touching the goal with a completely random movement is so low that in the experiment none of the simulations ended up with the robot reaching the target by the end of the simulation time.

We can see that the color zones are approximately triangles departing from the origin, i.e. the performance depends just on the ratio between α and β and not on their value.

Figure 6.14(f) shows the average performance for various $\frac{\alpha}{\beta}$ ratios. We notice that for the first type of damage (reduced wheel size) a ratio close to 2 gives the best performances, while in the case of changed rotation axis the best performance is obtained with $\frac{\alpha}{\beta} \approx 2.5$. The undamaged robot and the robot with damaged camera instead performs best with $\frac{\alpha}{\beta} \approx 3$. For the uncontrollable wheel, higher values for $\frac{\alpha}{\beta}$, around 5, gives the best performance. In this case, probably the noise introduced by the hardware itself reduces the noise required in the control signal.

Observing Fig. 6.14, we notice that some damages seems easier to recover than others. In detail, the performance decreases more abruptly when the size of one wheel is changed and when the rotation axis is changed by 90 degrees, than when the camera is partially obscurated or when one wheel become uncontrollable. In these cases, a lower $\frac{\alpha}{\beta}$ is more beneficial, i.e. intuitively speaking, it appears that when the task is difficult the more stochastic the control is the better it is. These preliminary results provide us an interesting hypothesis that will be verified by future works: the damage entity determines the optimal $\frac{\alpha}{\beta}$ ratio, with lower values for "harder" damages.

A second set of tests regards experiments with a real robot. Often the effect of noise, environment uncertainties and modeling errors prevents algorithms to work in a real world setup. We therefore decided to measure the performance of our algorithm in a real world setup.

We validated the practical applicability of our algorithm by using a mobile robot, namely B12 by Real World Interface[4]. The robot was equipped with a 640×480 Logitech webcam placed on an omnidirectional mirror and an off-the-shelf mobile PC. A red blanket placed on a chair was used as target, as visible in Fig. 6.15.

As in the simulation experiment, the control loop consists in acquiring the image, counting the number of red pixels, performing the biased random walk and setting the motor velocities. Once again, to show the applicability of our approach even in case of very low-cost hardware, no particular hardware choice or software optimization were performed, leading to a quite slow and jittery control loop time, 861 ms with a standard deviation of 7.3 ms.

It is interesting to notice that the motor command space of the robot used in the experiment is quite different from the one of the simulated robot. Expressly, while the simulated robot has two driven-wheels independently actuated and a third rear caster wheel, the real robot is a Synchro Drive platform, in which one motor drives the wheels, while the other changes the wheel orientation. However, since our approach makes no assumption on the hardware, no change in the algorithm is required when passing from the simulation setup to the real robot experiment. The only parameter adjustment required was the α/β ratio. For the real world experiment, we adopted $\alpha = 0.8, \beta =$

[4]A video is available at `http://robotics.dei.unipd.it/~fabiodl/video.php?navigation`

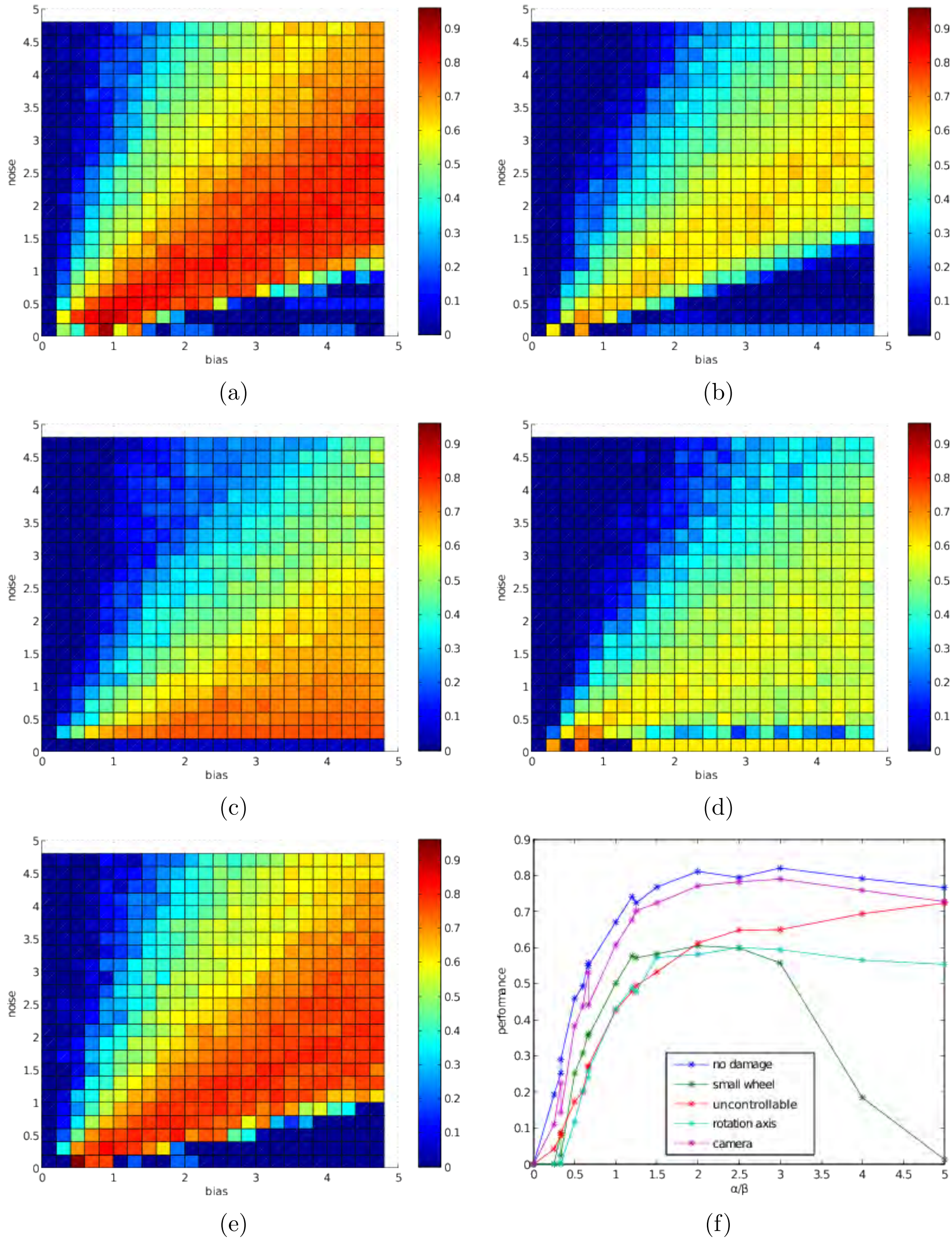


Figure 6.14: Performances (ratio between the time spent touching the target and the total simulation time) of the robot for various settings of α and β: (a) Without any damage, (b) Change in the size of one wheel, (c) Uncontrollability of one wheel, (d) Rotation of the axle of one wheel, (e) Obscuration of part of the image, (f) plot of α/β vs. average performance.

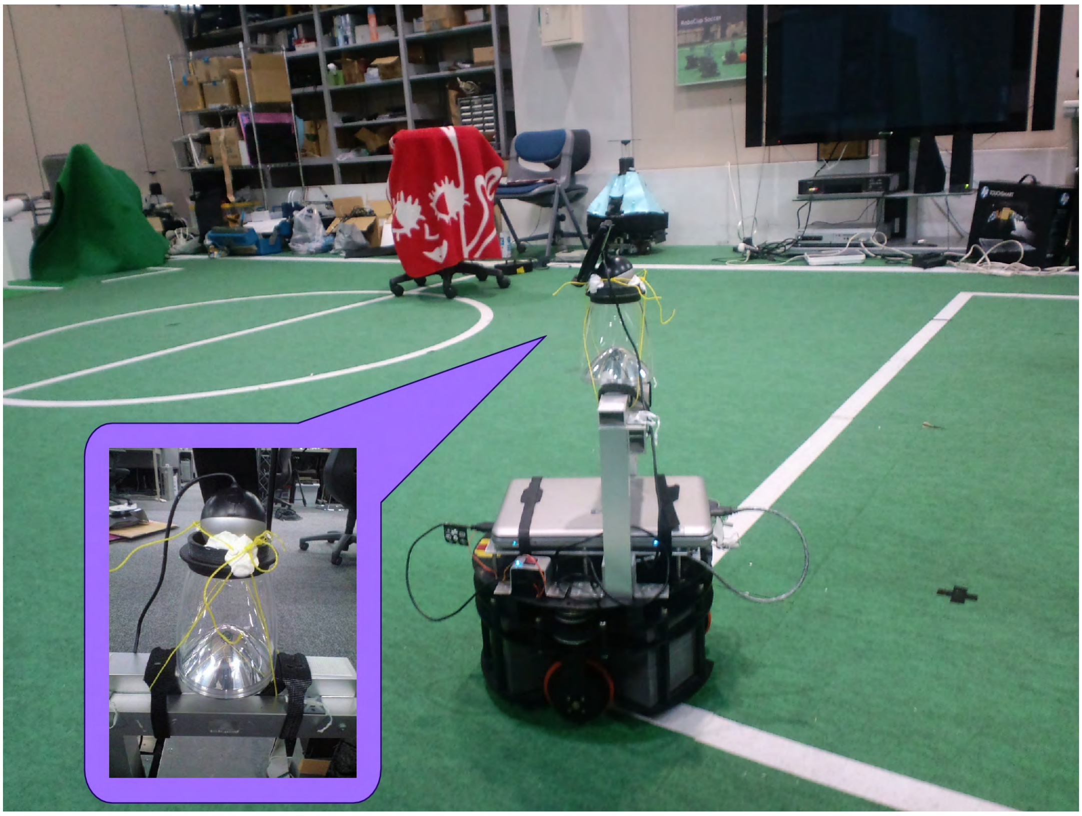


Figure 6.15: Setting of the real world experiment. A B12 mobile robot was equipped with a webcam placed on an omnidirectional mirror. A red blanked was used as the target.

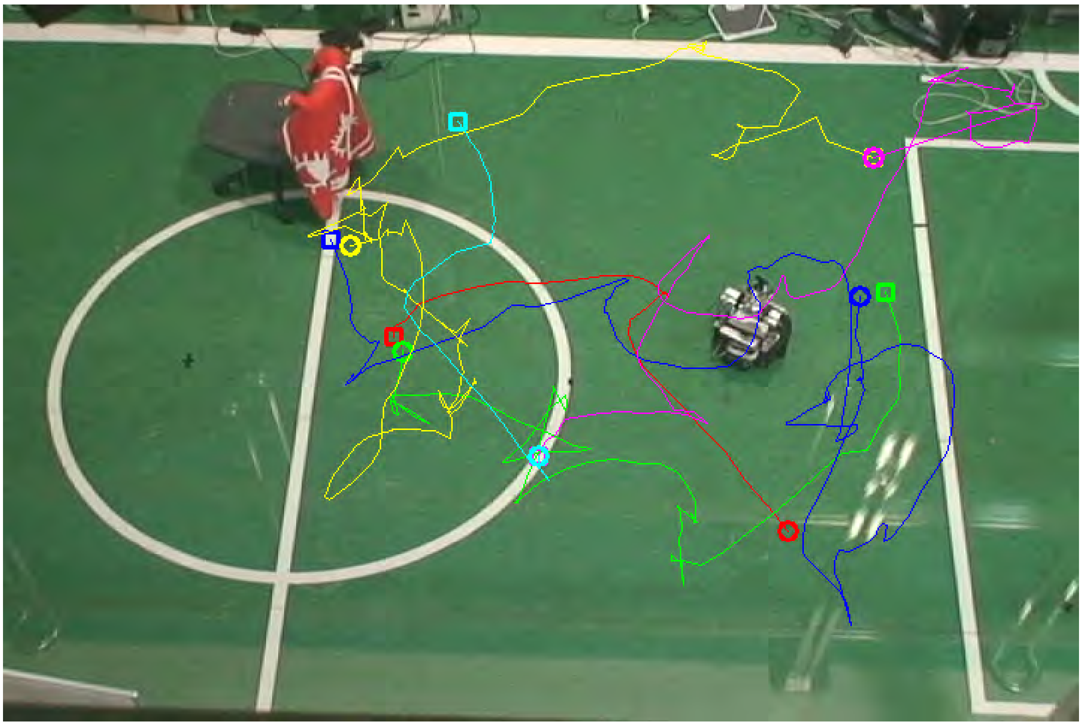


Figure 6.16: Paths covered by the robot for six different initial positions and goal settings. Each path is drawn with a different color. Circles indicate starting points and squares indicate end points (for some subsequent paths the starting point is the ending point of the previous path).

0.025, determined experimentally with four trials to understand the order of magnitude.

The motors are equipped with encoders, which were used exclusively for data logging and reconstruction of the robot path. The short length of the paths covered by the robot in the experiments actually make the error accumulated by the encoders negligible. A validation was nonetheless performed using image processing on a video captured from the top of the experiment field.

Figure 6.16 shows some of the paths covered by the robots for different goal and robot initial position settings. Initial distance were set with values from 164 to 364 cm, and for all the 11 tests performed by the robot achieved target reaching, with times ranging from 32 to 167 seconds. The measured path lengths were in average 3.38 times the optimal path. Notice that the performances could be improved by tuning the α/β ratio.

To illustrate the robustness to the real world noise Fig. 6.17 reports the relationship between the distance in a straight line to the goal, obtained off-line from the encoder values, and the number of red pixels measured during the robot motion. We can notice that, although the number of pixels is approximately a decreasing function of the distance, for similar distances we often have very different values for the number of pixel and, conversely, for a similar number of pixels the distances can be very different. Impressively, despite these strong uncertainties in the actual target distance, our simple

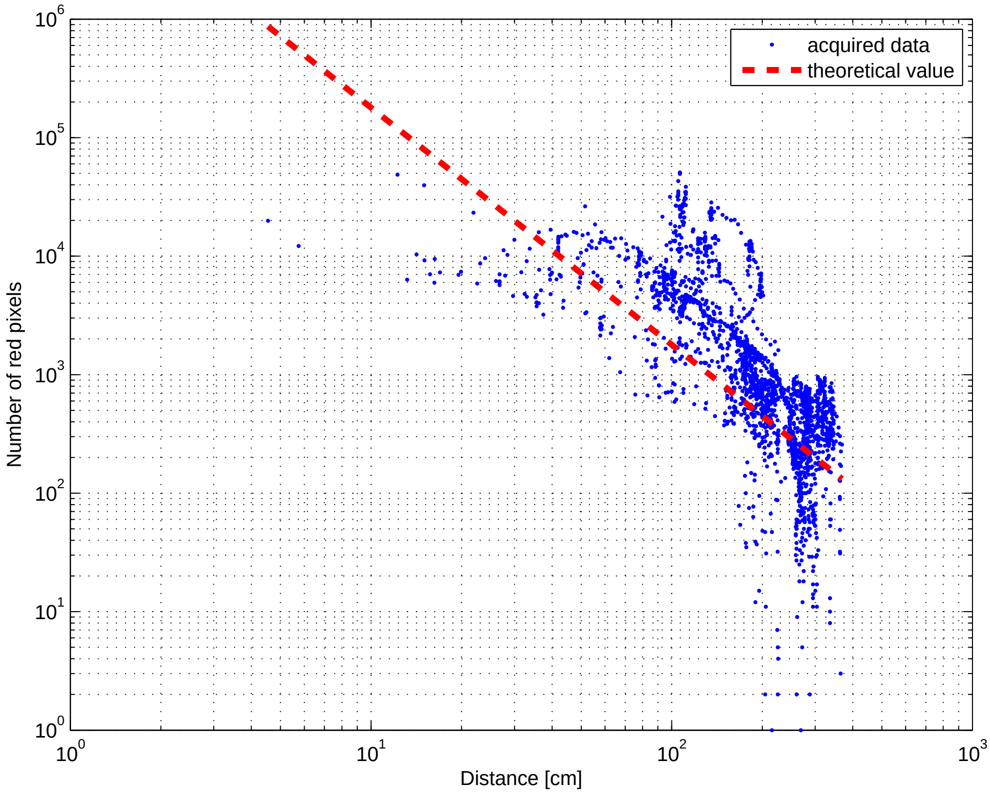


Figure 6.17: Relationship between the distance from the goal and the number of red pixels measured during the experiments.

algorithm was always able to drive the robot to the goal in a reasonably short time.

In conclusion, experiments showed that our algorithm can be sufficient in many cases to provide robustness to hardware damages and unexpected events. Practical results were obtained using a mobile robot equipped with an omnidirectional camera that has to reach a goal with very noisy information and while undergoing hardware damages. We provided an experimental study on the optimal bias in the random walk, and showed that performances can be optimized by adjusting the ratio between α and β. The results suggest that the optimal bias is proportional to the random term with a coefficient dependent on the hardware. Initial tests seem to suggest that the bias should be higher in case of stronger damages, but this should be investigated more systematically with a clear definition of the "strength" or "hardness" of damages. Limitations of the algorithm will be highlighted in the next subsection, and a new algorithm, which comprises automatic adaptation of the random perturbation size, will be described in Subsection 6.2.5.

6.2.4 Algorithm shortcomings

The previous subsection reported that our biased random walk based algorithm performed well both in simulation and with the real robot. Extensive tests were performed, and the algorithm was shown to work in most of the practical setups we tested. Actually, this comes from the fact that most of the real world system share common features, or can be easily transformed to very similar problem descriptions.

For instance, in most systems, opposite motor commands generate an opposite effects, at least in some regions of the motor command space, and scaled versions of the same motor command provide a similar effect with different intensity. By observing carefully Eq.6.1, it is possible to observe that the algorithm previously presented exploits these facts. In fact, it assumes that if command u_t is beneficial for the robot state at time t, then $\frac{u_t}{\|u_t\|}$ will probably be a good bias for the following command u_{t+1}. Furthermore, if u_t worsened the conditions during time t then it is assumed that $-\frac{u_t}{\|u_t\|}$ will be an appropriate bias for u_{t+1}. These considerations however do not hold for a generic motor command spaces. Suppose to have a holonomic robot whose coordinates are $x_t \in \mathbb{R}^2$. Assume now $u_t \in \mathbb{R}^2$ and its two components $u_t^{(1)}$ and $u_t^{(2)}$ to represent the velocity in polar coordinates, i.e. $x_{t+1} = x_t + \left|u_t^{(1)}\right| \cdot \begin{bmatrix} \cos(u_t^{(2)}) \\ \sin(u_t^{(2)}) \end{bmatrix}$. The nonlinearity near the origin introduced by the cosine prevents the algorithm from being able to drive the robot to the goal.

Another disadvantage of using $-u_t$ as bias when the conditions worsen is that the control algorithm performs badly if there are dead times in the response. Imagine, for simplicity, a unidimensional case where the goal is at +10 and the position x_t, initially 0, changes by $x_t = x_{t-1} + u_{t-2}$, $x_t, u_t \in \mathbb{R}$. In this case, the performance increases when the bias is positive, i.e. ΔA is positive for a positive u and negative otherwise. Suppose to start with u_0 and u_1 negative, and suppose the random perturbations to be small enough that the sign of the motor command u_t is determined by the sign of the bias b_t. Since u_0 is negative ΔA_2 will be negative, and the bias b_2 and signal u_2 will become positive. However, in the next step, the effect of u_1 will lead to a negative ΔA_3, which in turn will bring the bias b_3 and u_3 to become negative again. The effect of u_2 will provide a positive ΔA_4, so the bias b_4 will have the same sign of u_3, i.e. it will be negative. At this point the evolution of the bias will repeat, in a loop that contains two negative biases and a positive one. If the magnitude of the biases is similar for the positive and negative case, in general the bias will tend to bring the robot farther from the target instead of bringing it closer, and the system will never reach the target +10. Table 6.1 reports other examples of bias sequences that reveal to be a nuisance instead of being beneficial to reach the target.

For similar reasons, the bias can become deleterious if the system includes delays introduced by low pass filters. This can be a serious disadvantage of the algorithm, since many physical systems present this kind of behavior. A simple example can be provided by introducing an Infinite Impulse Response (IIR) filter in the example of a holonomic robot moving on the plane. Concretely, assume the robot to change its position by $x_{t+1} = x_t + v_t$ where $v_t = (1 - 10^{-\rho}) \cdot v_{t-1} + 10^{-\rho} \cdot u_t$. Let us define a "bad bias" a bias that has a heading that differs more than 90 degrees from the optimal one and

Table 6.1: Bias sequences leading to performance decrease

	t					
	0	1	2	3	4	$\cdots$
ΔA_t			-	-	+	$\cdots$
u_t	-	-	+	-	-	$\cdots$
ΔA_t			-	+	-	$\cdots$
u_t	-	+	-	-	+	$\cdots$
ΔA_t			+	-	-	$\cdots$
u_t	+	-	-	+	-	$\cdots$

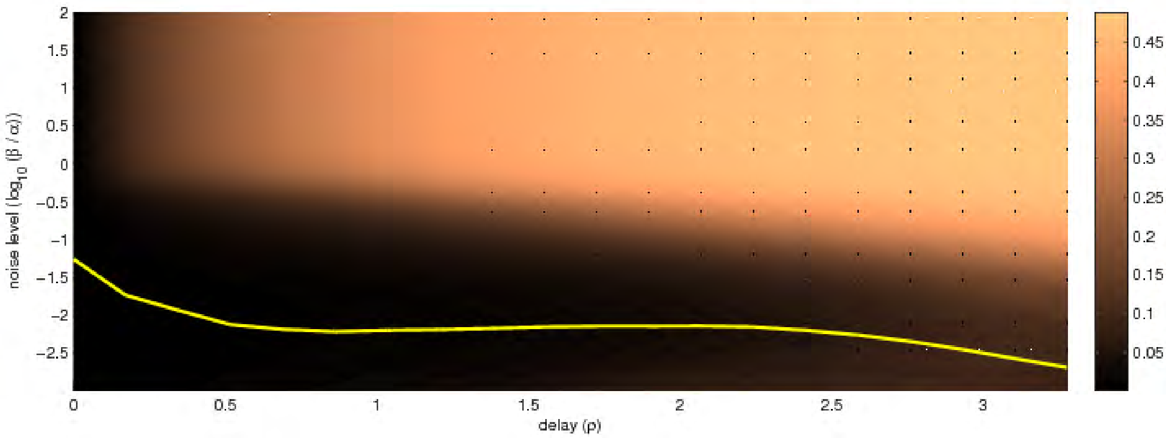


Figure 6.18: Probability of biases that would bring the robot further from the goal. The X axis represents the delay level (ρ), the Y axis represents the perturbation level (β/α) and the color indicates the probability of bad biases (lighter color indicates a higher probability). The yellow line indicates the noise level that gives the lowest bad bias for each delay level.

would therefore bring the robot further from the goal. Figure 6.18 reports how the probability of bad biases changes by varying the random perturbation level (β/α ratio) and the delay level (ρ). Although the probability of bad biases can be minimized by changing the random perturbation level, it can be observed that as the delay increases, the probability increases. For instance, for a value of ρ equal to 3 the probability of bad biases cannot be lower than 0.09.

It is worth noting that for a wide range of problems it is often possible to find expedients that allow mitigating the weak points presented here and that allow the use of the previous version of the algorithm. In fact, an adequate parametrization of the motor command space reveals to be sufficient in most of the cases. However, the algorithm presented in the next subsection strongly improves the generality without increasing the algorithm complexity.

6.2.5 Improved algorithm

As reported in Subsection 6.2.1, Escherichia Coli proceeds by movements in random directions, but when moving toward increasing concentrations of nutrients, the movement in that direction is prolonged. A similar behavior can be obtained by taking $u_{t+1} = u_t$ if $\Delta A_t \geq 0$ and selecting u_{t+1} randomly[5] if $\Delta A_t < 0$. As in the previous version of the algorithm, a random perturbation can improve the performances. In particular, it is sufficient to add a perturbation to each of the components of the input when $\Delta A_t \geq 0$, i.e. $u^i_{t+1} = u^i_t + \eta^i R$, $R \sim \mathcal{N}(0, 1)$ for each of the components of the input $(1 \leq i \leq m)$.

Choosing the bias at random when the system is getting further from the goal removes any assumption of linearity of the system. Clearly this leads to a performance decrease for systems that are effectively linear, but considerably improves the generality of the algorithm. Furthermore, in case of dead times periodic bias sequences with negative effects are unlikely generated. For instance, in the case of the unidimensional example provided in the previous subsection, if by chance a positive bias is followed by another positive bias, then the system will keep a positive bias and reach the target[6]. Similarly, better performances are expected when delays arising from low pass filter effects are present. Imagine in fact to have a sequence of good motor commands that are not recognized as such because their effect comes later. In the meanwhile, new commands will be generated. If the system responds with an opposite behavior when the input is negated (as for all linear systems and many other setups), then choosing a random command is less deleterious than choosing the negated motor command.

Intuitively, the algorithm operates in a very simple way. It keeps using the same motor command as long as the command is beneficial, otherwise it picks up a new one at random. This provides intuition on how to adjust the magnitude of the random perturbations. Expressly, if the random perturbations are appropriate and in general good inputs are selected, these will be used for a long time. By observing the variance of the produced motor commands, we can thus have an idea of the quality of the motor command. In order to dynamically adapt η^i_t we can therefore estimate the variance of u^i_t by picking some samples, slightly increase(/decrease) η^i_t, and estimate the variance of u^i_t again. If the variance decreased then we increase(/decrease) η^i_t once more, otherwise we decrease (/increase) it. This kind of effect can be showed by a simple example. Suppose, as previously done, to have a holonomic robot that must approach a target located in $[0, 0]^T$. To reduce the problem to a system with a unidimensional motor command, assume the robot to move by steps of fixed length s, along the angle indicated by u_t. Formally, let $x \in \mathbb{R}^2$ be the robot position, $u_t \in \mathbb{R}$, $x_{t+1} = x_t + s \cdot \begin{bmatrix} \cos(u_t) \\ \sin(u_t) \end{bmatrix}$. Assume

[5] In the following we assume to select $u_t \in \mathbb{R}^m$ using a uniform distribution over the whole motor command space, but the results remain essentially the same using different distributions.

[6] As previously stated, we assume the perturbations to be small enough. Notice, however, that a sequence of two positive biases is sufficient for recovery if perturbations change the sign.

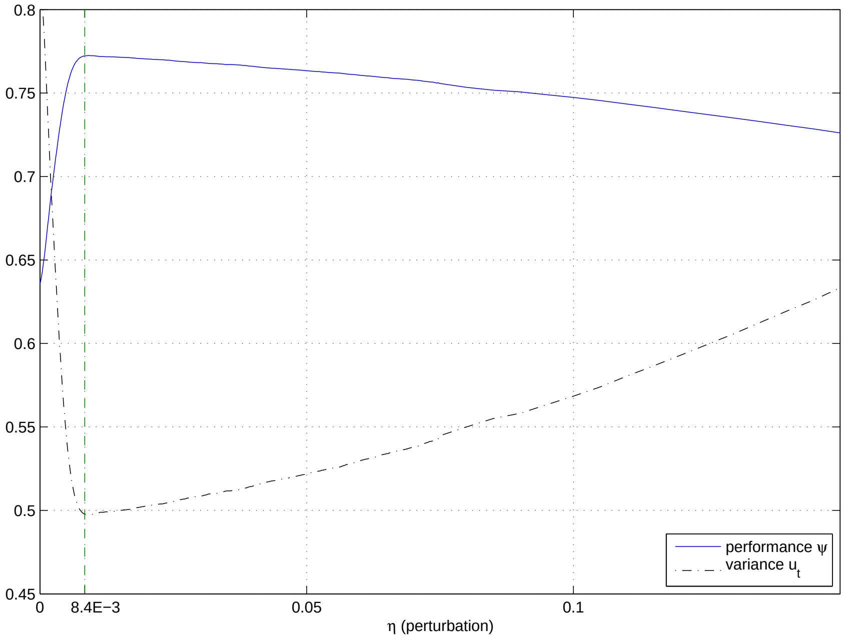


Figure 6.19: Average performance and input variance obtained for different values of η. The graphs were obtained placing the robot in $x_0 = [10, 10]^T$ with $s = 10^{-6}$, simulating $N = 10^3$ steps and repeating the test 10^5 times.

then to express the performance ψ as the average decrease in the distance to the goal for a single step over N steps, i.e. $\psi = \frac{\|x_0\| - \|x_N\|}{N \cdot s}$.

Fig. 6.19 reports the average performance ψ and variance of u_t for different values of η. We notice that the maximum performance corresponds to the minimum variance. For more complex setups the two peaks could not coincide, but choosing the perturbation that gives the lowest motor command variance appears to be a reasonable choice in most cases.

Assuming to estimate the variance using just two samples[7] we derive the following algorithm

$$u^i_{t+1} = \begin{cases} u^i_t + \eta^i_t R & if\ \Delta A_t \geq 0 \\ random\ selection & otherwise \end{cases} \tag{6.2}$$

$$\delta^i_0 = 1.1 \tag{6.3}$$

$$\sigma^i_t = \frac{(u^i_t - u^i_{t-1})^2}{2} \tag{6.4}$$

$$\delta^i_{t+1} = \begin{cases} 1/\delta^i_t & if\ t\ odd \wedge \sigma^i_t \geq \sigma^i_{t-2} \\ \delta^i_t & otherwise \end{cases} \tag{6.5}$$

[7] A higher number of samples provides a better estimate of the variance and therefore of the variance change, but slows down the adaptation. Note that, however, two samples are always sufficient to guarantee a right estimation of whether the variance increased or not with a probability higher then 0.5.

$$\eta_{t+1}^i = \begin{cases} \eta_t^i \delta_{t+1}^i & if\ t\ odd \\ \eta_t^i & otherwise \end{cases} \tag{6.6}$$

In this way we obtain an algorithm that is parameterless, is very robust and works even with highly nonlinear systems. The next subsection will present experiments on the algorithm, showing both its generality and performance in reasonable settings.

6.2.6 Comparison experiments

This subsection presents the results on the comparisons between the biased random walk algorithm presented in Subsection 6.2.2 and the version presented in the previous subsection. In particular, as a first step we compared the performances of the two algorithms when coping with nonlinear systems. In this experiment, the movement of the robot was set to $x_{t+1}^i = x_t^i + s \cdot f^i(u_t)$ where $f^i(x) = \frac{1}{\pi} \arctan\left(\frac{(sin(2\pi x+\xi^i))^T Q^i sin(2\pi x+\xi^i)}{(sin(2\pi x+\zeta^i))^T P^i sin(2\pi x+\zeta^i)}\right)$. In this expression the sin function is applied element-wise and $Q^i, P^i \in \mathbb{R}^{2\times 2}$ and $\xi^i, \zeta^i \in \mathbb{R}^2$ were randomly initialized. Figure 6.20 shows the distance traveled toward the goal in different trials. In detail the robot was placed in 6 different initial positions ($10 \cdot [sin\,(2k\pi/6)\, cos\,(2k\pi/6)]^T$, $k \in \mathcal{N}, 0 \leq k \leq 5$) and for each position the experiment was repeated 10^4 times. We notice that as expected the performance is generally higher for the newer version of the algorithm. The newer version of the algorithm is able to drive the robot to the goal even in case of the highly nonlinear mapping introduced in the experiment.

The second test deals with dead times in the system. Expressly, we simulated the case $x_{t+1} = x_t + u_{t-d}$, $d \in \mathcal{N}$, $x_t, u_t \in \mathcal{R}^2$, $-s \leq x_t^i \leq s$ for $N = 10^4$ time steps. As visible in Fig. 6.21, with the previous version of the algorithm the distance traveled toward the goal drops off as soon as there is a dead time d. The performance of the new version of the algorithm degrades as the dead time d increases, but does not reach 0, i.e. the algorithm is still able to drive the robot to the goal whatever the dead time is.

We then tested the versions of the algorithm on a system that includes a low pass filter as the one described in Subsection 6.2.4, i.e. we assumed the movement to be given by $x_{t+1} = x_t + v_t$ where $v_t = (1 - 10^{-\rho}) \cdot v_{t-1} + 10^{-\rho} \cdot u_t$. Figure 6.22 reports the distance traveled toward the goal for different values of the filtering effect ρ. We can observe that in the previous version of the algorithm, the distance traveled toward the goal decreases as ρ increases and becomes nearly 0 for a value of $\rho = 3$. The newer version has better performance for all of the ρ settings, and interestingly the performance increases for $\rho > 2.4$. This is due to a smoothing effect introduced by the low pass filter that makes the trajectories more straightly headed to the goal.

The tests on a mobile robot that undergoes hardware damages, described in Subsection 6.2.3, were repeated for the new algorithm.

As a result, the newer algorithm provides faster reaching times than the previous

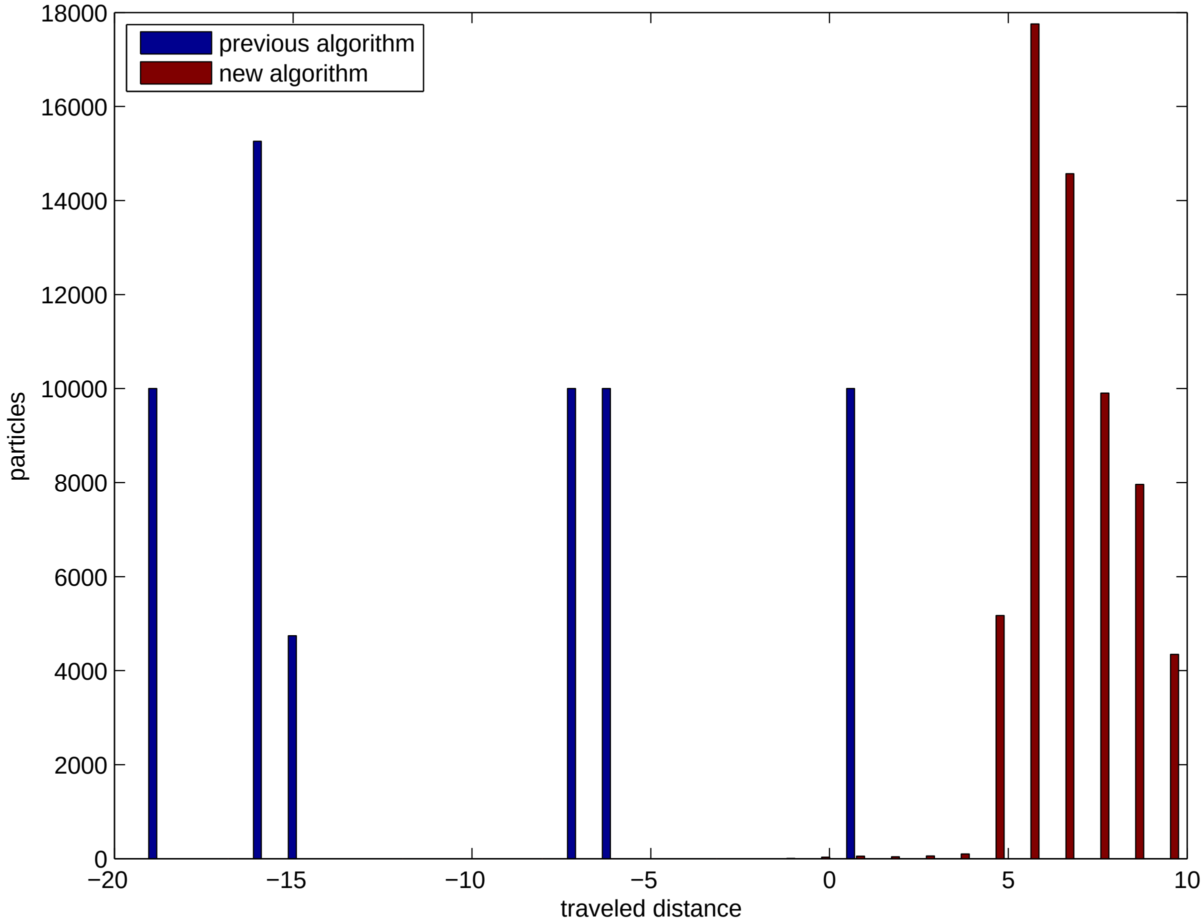


Figure 6.20: Distribution of the distance toward the goal traveled in $N = 10^4$ steps of size $s = 10^3$ using the nonlinear functions $f^i(x)$. The robot was placed in 6 different positions and for each position the test was repeated 10^4 times.

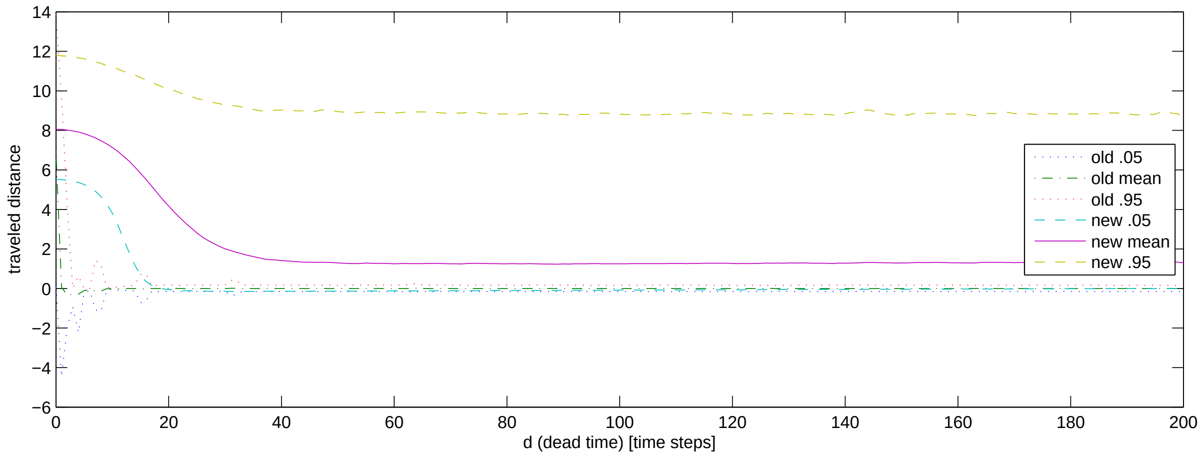


Figure 6.21: Average movement toward the goal for different dead time values. The plot reports the average distances traveled using the two algorithms, as well as the 0.05 and 0.95 quantiles, i.e. the distances traveled toward the goal are reported with their 90% confidence interval. The graphs were obtained setting the maximum velocity $s = 10^{-3}$, placing the robot at $x_0 = [100, 100]^T$, simulating the movement for $N = 10^4$ steps and repeating the experiment 10^4 times.

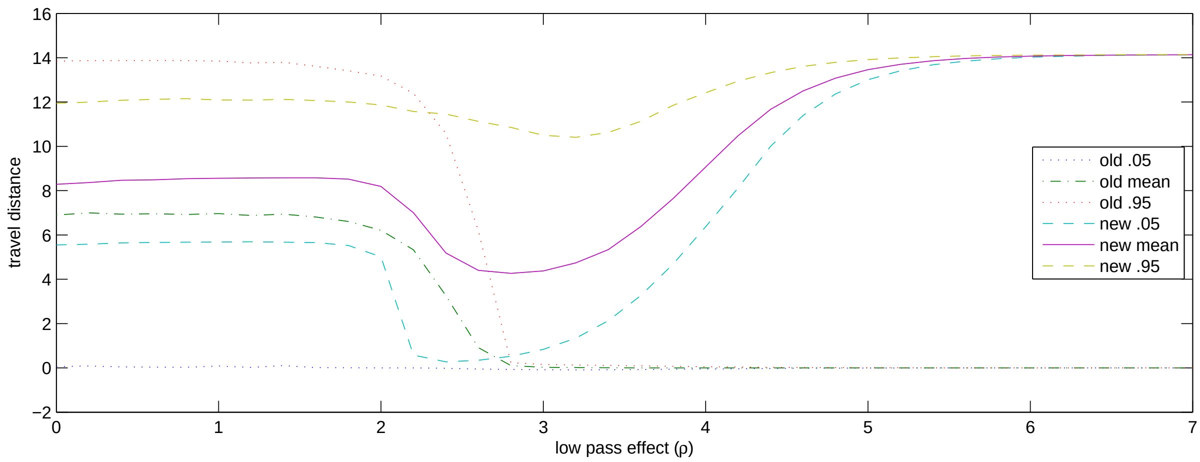


Figure 6.22: Average movement toward the goal for different values of the low pass filter entity ρ. The plot reports the average distances traveled using the two algorithms, as well as the quantile function for 0.05 and 0.95, i.e. the distances traveled toward the goal are reported with their 90% confidence interval. The graphs were obtained setting the maximum velocity $s = 10^{-3}$, placing the robot at $x_0 = [100, 100]^T$, simulating the movement for $N = 10^4$ steps and repeating the experiment 10^4 times.

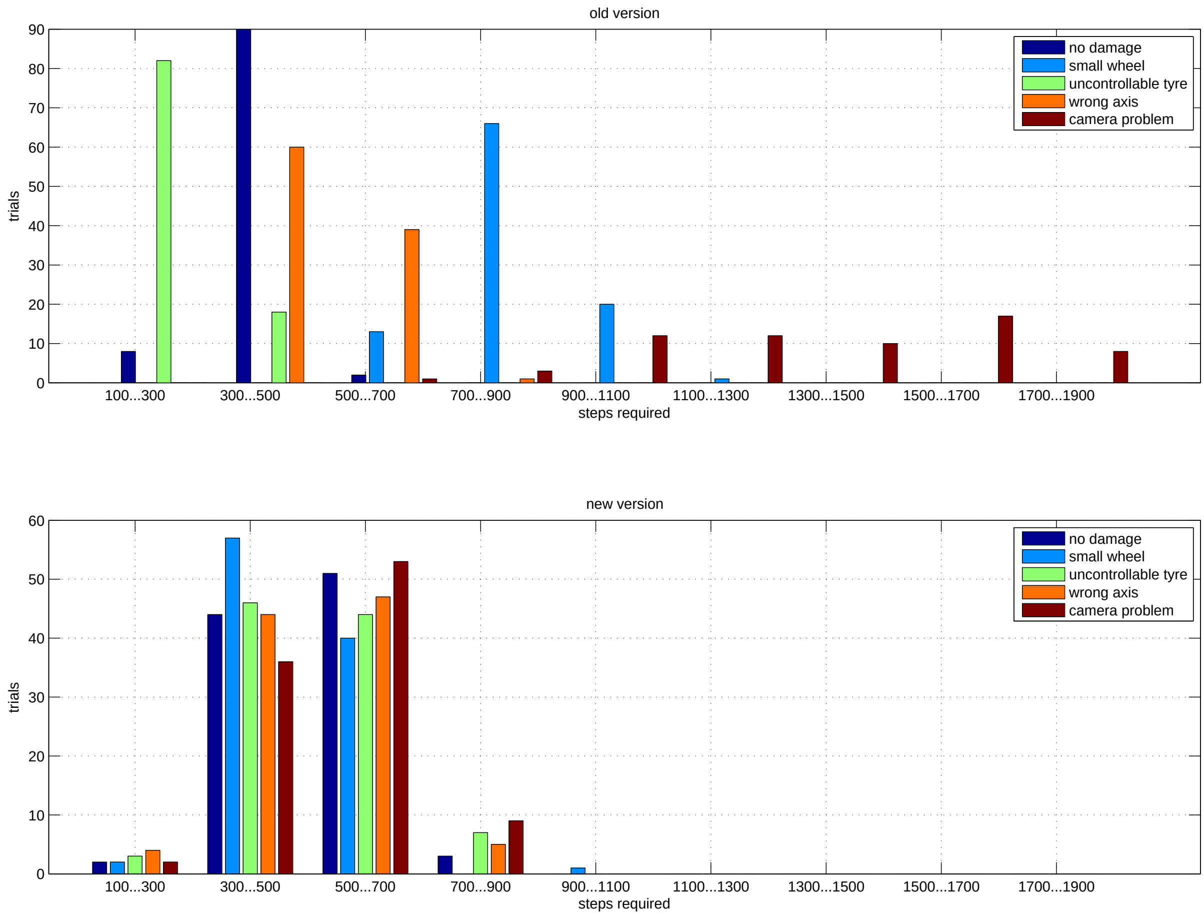


Figure 6.23: Distribution of the time steps required to reach the target. The time step was set to 5s, and the experiment was repeated 100 times for each robot condition. For the previous version of the algorithm, the test was repeated for β/α equal to 0.01 to 0.91 with steps of 0.1 and the best performing noise level was chosen.

version (with α/β set as to maximize the performances) in most of the cases. In the uncontrollable wheel case, however, the previous version obtained better performances, because with that setup automatically setting the noise level becomes difficult. Figure 6.23 reports the distribution of the time steps required to reach the goal.

6.2.7 Humanoid robot results

Subsection 6.2.3 described experiments on the first versions of the algorithm inspired from E. Coli chemotaxis. The previous subsection reported a comparison of the two versions of the algorithm, showing the improvements over the previous version. In order to reduce the time required for the experiments and easing the analysis, mobile robots were used, both in simulation and in real world experiments. This subsection presents preliminary experiments with a humanoid robot.

In detail, before applying the second version of the algorithm on humanoid robot, the

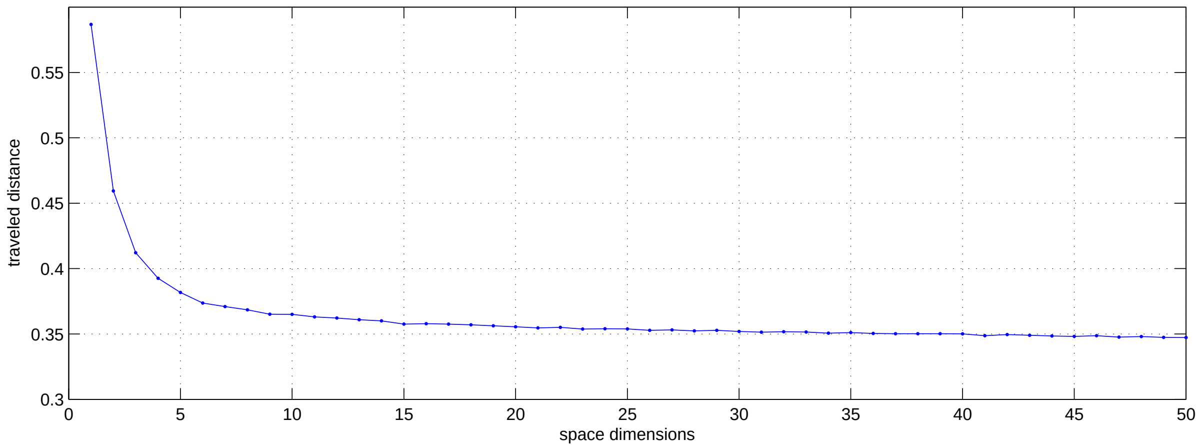


Figure 6.24: Average movement toward the goal by the particles for different settings of the space dimension p.

behavior of the algorithm in highly dimensional spaces was tested. The experiment was conducted placing 10^4 particles in a p dimensional space at $x_0 = [-1, 0, \ldots, 0]^T \in \mathbb{R}^p$ and assuming their position to be modified by $x_{t+1} = x_t + s \cdot u_t$ where $s = 10^{-4}$. Fig. 6.24 reports how much the particles got closer to the goal in $N = 10^4$ steps as p varies. Although we would expect an exponential decrease of the distance traveled toward the goal, we notice that the performance is reasonably high even for high values of p. This ability to cope with high dimensional spaces makes the algorithm appealing for the control of the humanoid robots.

When applying the algorithm to practical problems, the first thing to decide is how to define the quantity ΔA. If a specific task is given beforehand, then defining A as an indicator of the progress in the task achievement, as suggested in [20], and taking ΔA as its derivative appears to be a good choice.

Since we are interested in the development of simple, task independent algorithms, we assume the setup of having a generic motion, developed, for instance, by the teaching by touching approach, that we can use as reference. Let us denote the motion as a function that, given a time, returns the corresponding posture assumed by the robot $M(t) : \mathbb{R} \to \mathbb{R}^n$, where n is the number of motors. Note that, in order to simplify the notation, in this definition and hereafter, the time is assumed continuous.

A possible way to define A in a generic way is by simply taking the Euclidean distance between the position actually assumed by the motors, that we denote by $x(t) \in \mathbb{R}^n$, and the position specified by the motion, $M(t)$. In particular, in order to allow our algorithm to generate online interesting variations of the motions, we decided to compute $M(t)$ at a frequency lower than the frequency of the application of the algorithm. In other terms, we sampled the motion at a set of time instants $t_k \in \mathbb{R}$, $k \in \mathbb{Z}$, and set the algorithm "goals" as $M(T_1)$ during the time $[0, T_1)$, $M(T_2)$ during the time $[T_1, T_2)$, etc. Formally, let us define $\bar{M}(t, T_q)$ as the next target position, sampled with a step T_q i.e.

$\bar{M}(t, T_q) = M(T_q)$, $q : T_{q-1} \leq t < T_q$. With this definition, we can express A as

$$A = \|x(t) - \bar{M}(t, T_q)\|_2$$

and, as usually, define ΔA as the variation of A between successive executions of the control algorithm.

We notice that the same scalar, ΔA, is used to control all the motors as a whole, and not a single joint. This permits, for instance, to start generating random movements on all of the joints when one of them is stuck and cannot reach its target position. Clearly if further sensory information is available, this can be introduced in the computation of the distance between the expected robot's state and the current one. The simplest approach to know the expected state could be, for instance, playing the motion in a controlled environment and storing the data, as done in Section 6.1.

In the experiment we decided use a crawling movement, developed by touching, as a reference motion. The motion lasts 1162 ms. Its first and last posture are identical, so that it can be repeated continuously. As in Subsection 6.1.4, a simulated M3-Neony was employed. The motion execution was simulated in two environments. The first is a flat floor, and the second is the rough terrain made by randomly inclined tiles that was presented in Subsection 6.1.4.

The control algorithm, i.e. the computation of Eq. 6.2, was run every Δ_T =3ms. Various settings of the sampling interval of the motion were tested. In particular, we conducted experiments with the sampling values $T_q = 2^k \Delta_T$, $0 \leq k \leq 9$. To better study the effect of the purposely added random perturbations, we decided to fix the perturbation magnitude coefficients η_t^i to constant values, and evaluate the performance for different settings. In particular we decided to adopt the same constant value for all the joints, and measured the system performances for $\eta = 0.01/2^h$, $0 \leq h \leq 8$.

Figure 6.25 shows the results obtained on the flat terrain, while Fig. 6.26 reports the performance for the rough terrain, for the various settings of k and h previously described. The value used as index of the performance is the distance traveled in 5 simulated minutes, averaged over 50 trials.

We notice that in both cases the performance has a peak for $\eta = 0.005$ and $2^k = 64$. When the motion is replayed in open loop, without using the proposed algorithm, the average traveled distance is 5.40 m on flat terrain and 1.16 m on rough terrain. When the proposed algorithm is used with $\eta = 0.005$ and $2^k = 64$, the average performance is 3.76 m for the flat terrain and 2.07 m for the rough terrain.

As could be expected, we see that with the flat terrain, where essentially the environment is predictable, using a random perturbation for the control decreases the performance. When, instead, the environment changes, as in the case of the rough terrain, the random component helps to adapt to the environment. As expected, from the graphs we see that there exists a value of the random perturbation magnitude for which the

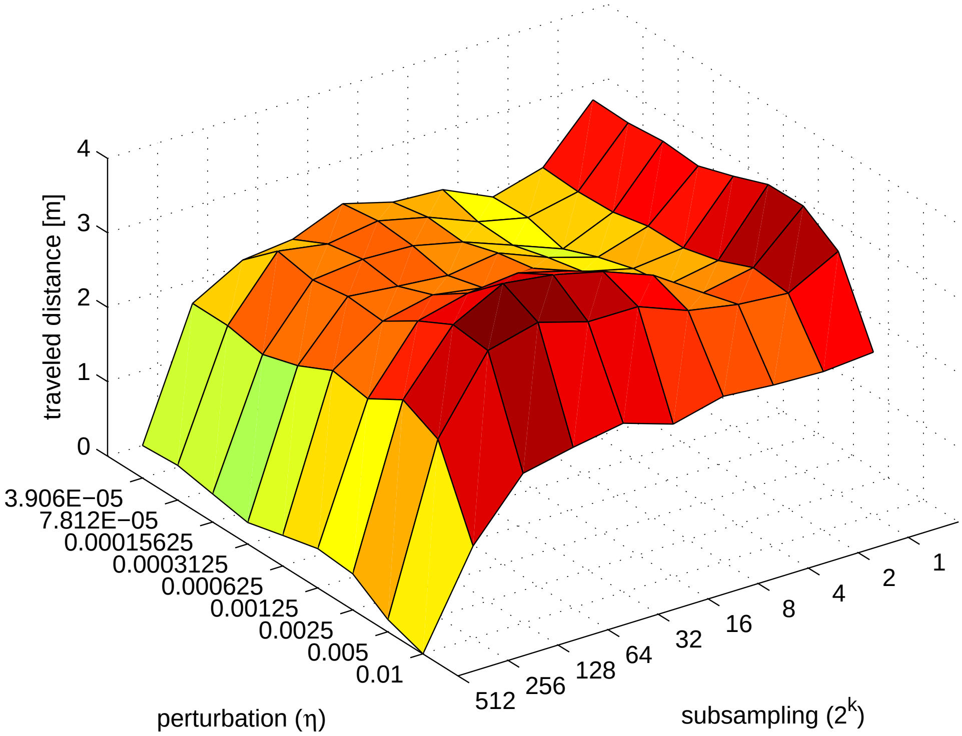


Figure 6.25: Average distance traveled in 5 minutes on the flat terrain for different settings of the random perturbation magnitude and motion subsampling.

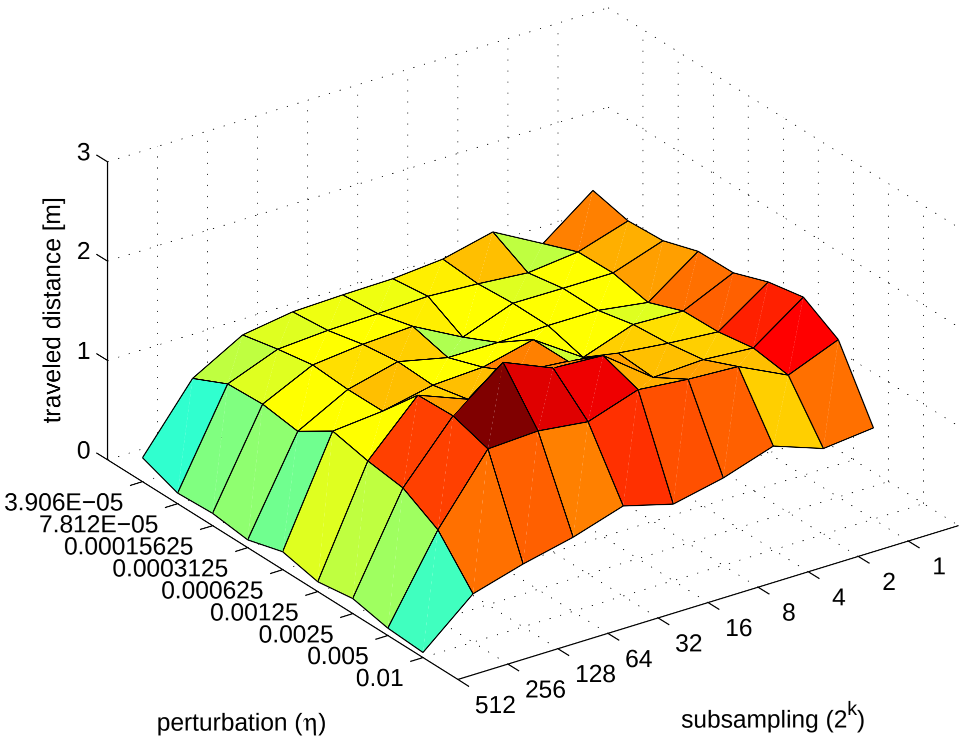


Figure 6.26: Average distance traveled in 5 minutes on the rough terrain for different settings of the random perturbation magnitude and motion subsampling.

performance is the highest, i.e. $\eta = 0.005$. Interestingly, there is also a value $2^k \neq 1$ of the subsampling of the motor commands for which the performance is optimal. Actually, we see that in both cases the performance decreases when 2^k goes from 1 to 2, but then increases, until 2^k=64. This seems to confirm our idea that if the target posture is set with a frequency lower than the one of the control algorithm, then the algorithm has freedom to find good solutions for the movement execution. If the sampling time T_q is too long, however, too much information on the original motion is lost and the resultant movement looses performance in achieving the task.

Actually, the performances of the algorithm could be increased by choosing better functions for A, or by introducing different functions for different limbs. The result presented here is just a first preliminary experiment, but they suggest the feasibility of the algorithm for controlling humanoid robots with a high number of degrees of freedom.

7

Conclusions and future works

The idea of using interpretation of tactile instructions for programming humanoid robot motions was presented in this thesis.

In general, when dealing with robots in real world setups, the curse of dimensionality problem rules out any naive application of classic automated learning. For these reasons, much works resort to solutions focused on a particular task. Another approach often taken is the exploitation of the help of a teacher.

Our programming paradigm belongs to this second category. The help of a teacher is used to develop the robot motion. This gives the user fine control on the resulting motion. The problem of the emergence of awkward motions, that appears with naive automatic optimizations of the movement, is thus removed, at the expense of an increased effort from the teacher.

A wide spectrum of techniques that benefit from the coaching of a human teacher can be found in literature. The methodologies differ under several aspects. From the point of view of the richness of the information transferred from the teacher to the robot, we find several levels, ranging from a simple critique of the performance to a detailed set of instructions.

From the communication perspective, literature presents a huge variety of modalities, among which we can cite physical interaction, vision, speech or physical interaction.

Another criteria for the classification of the modalities is the interactivity of the interaction. Some approaches assume the robot to be passive, or to practice the movement and learn by itself once the user demonstration is ended. In other setup, conversely, the robot and the user mutually interact. The robot "proposes" a solution that the user evaluates and criticizes or modifies.

In the case of whole body humanoid robot motions, which were considered in this thesis, two approaches are commonly used. The first one is motion retargeting, a technique often employed in computer graphics for the animation of characters. The movement of a human performer is recorded, and adapted to the robot. The approach is appealing because human-like motions can be easily obtained.

However, the technique presents several inconveniences. The instrumentation for acquiring motion data is bulky, expensive and usually not available to average customers. Moreover, a human actor who is able to perform the desired movement may not be available. In some cases, users could desire to make the robot do movements impossible to be done human beings. Last but not least, when porting the motion from a human to a humanoid, it must be heavily modified due to the differences in the shape, degrees of freedom, power and weight distribution between humans and robots. During this adaptation process, the quality of movement appearance can be strongly degraded.

A second widely used approach for humanoid robot motion development is kinesthetic demonstration. Within this setup, the user directly manipulates the robot, which responds passively to the forces applied by the user. The robot stores the information on the movement, and possibly derives a statistical model if multiple examples of the movement are given.

The teaching by touching approach proposed in this thesis resembles kinesthetic demonstration because of the direct, tactile interaction between user and robot. However, the way of employing touch is different. Within the classical kinesthetic demonstration approach, the impedance of the joints determines the way the robot responds to the applied force. In the teaching by touching setup, instead, the robot tries to interpret the meaning of the tactile instruction, and moves depending on its estimation of the meaning of the tactile instruction.

The teaching by touching approach tries to make the robot training closer to the interaction between humans. Ideally, when a robot is programmed by kinesthetic demonstration, the interaction is close to the one with a puppet, while teaching by touching attempts to mimic the interaction between a human coach and a human trainee.

Advantages of the presented approach over kinesthetic demonstration can be readily provided. When interpreted, a single touch could be made to correspond to the simultaneous movement of both arms and legs, while it would be very difficult, if not impossible, to move the four limbs of the robot simultaneously with classical kinesthetic demonstration.

Additionally, with big robots, kinesthetic demonstration may reveal to be unfeasible if solutions for compliance control and gravity compensating are not adopted. The teaching by touching approach does not suffer from these drawbacks.

From another perspective, the presented approach is more general than kinesthetic demonstration, and the latter can be considered as a particular case of the teaching by touching approach. In fact, our system could behave exactly as in the kinesthetic demonstration setup by choosing an interpretation of the tactile instructions that corresponds to the passive, compliant movement.

This generality derives from the fact that the teaching by touching considers touch as a means of communication, instead of a simple way to transfer forces from the teacher to the learner. If we observe humans learning sport or dance skills, we notice that touch is intuitively employed as a communication means between the instructor and the learner. Indeed, humans can convey much information trough simple touches. However, the interpretation of tactile instructions is complex for a humanoid robot. In fact, their meaning is both user dependent and context dependent. The same touch can have different meanings depending on the robot's posture, body orientation, or on the personal way of teaching of the user.

Literature provides no models for the meaning of tactile instructions. This thesis presents a supervised learning approach that constructs a model for the interpretation of tactile instructions through the interaction with the user.

A supervised learning setup reveals to be interesting for two different reasons. From an engineering point of view, it constitutes a feasible solution for realization of practical systems that employ touch as a communication mean between users and robot. From the humans sciences perspective, the analysis of data collected during the interaction can provide useful insights on the way humans use touch to communicate, a largely uninvestigated research topic.

The thesis presents three variants of the implementation of the teaching by touching paradigm. The first implementation consists of an augmented reality setup. A real robot is used, but virtual tactile sensors are employed. This allows using the teaching by touching approach also with robots that lack tactile sensors. The second implementation shows that the approach can be easily ported to robots equipped with physical sensor. Both of these two implementations use a classic motion representation, based on a sequence of key-postures that are interpolated over time. The tactile instructions are translated into changes of the posture assumed by the robot at a determinate time.

The third implementation shows that the teaching by touching approach can be used to change the whole motion with a single touch. In particular, in the third implementation the motion is generated by a CPG. When the robot is touched, the tactile instruction is translated into a change of the CPG parameters that, in turn, corresponds to a modification of the whole movement.

As the most common humanoids on the market are small size humanoids, the thesis focused the application of the methodology to this platforms. For this reason, one of the main priorities of the algorithms developed was to keep the computational cost low.

Along these lines, the last part of the thesis presents simple, general solutions to improve the motion robustness while keeping the computational cost low. The approaches are completely general, and do not assume the motion to be developed by touching. However, teaching by touching combined with these methods appears to be a solution for developing motions and replaying them on small humanoid robots with limited computational power in a robust way.

The topics presented in this thesis can be found in various publication. In particular, Chapter 2 provided a conceptual analysis of the idea of interpreting tactile instructions for motion development. After a short survey on the research that deals with touch as a mean of communication, that was provided in Section 2.1, the difficulties in the estimation of the tactile instructions were given in Section 2.2. A first implementation of the algorithm, based on simulated touch sensors, was given in Chapter 3.

More precisely, Section 3.1 provided the details on the algorithm used, kernel regression with a specifically designed kernel. Section 3.2 described the system, Section 3.3 explained the experimental setup and Section 3.4 reported an analysis of the data collected in the experiment. Feasibility of the approach was demonstrated by the realization of motions. Furthermore, data analysis showed that globally linear models are unable to predict the mapping between tactile instructions and motion modifications. The section also shows that classic data mining techniques can be easily applied to collected data. In particular, extraction of the context features that are important for the tactile interpretation was performed using the C4.5 algorithm.

These findings were initially published in [212]. In [213], the advantages of the simulated touch sensors were highlighted, and a more detailed description of the simulator was provided. The theoretical formalization of the teaching by touching process, and the considerations on the user effort reported at the beginning of the chapter were first given in [214]. The same paper also provided the user dependence study reported in Section 3.5.

Chapter 4 reported the experiments conducted with the real robot. In particular, Section 4.1 described M3-Neony, the humanoid robot developed by out research group. This robotic platform was introduced in [215]. The system description was provided in Section 4.2 and the analysis of the experimental results was given in Section 4.3.

In particular, it was reported that the data suggest that usually a limb is moved by touching touch sensors on the same limb, as could be expected, but, interestingly, this appears not to be true when the users want to convey higher level behaviors to the robot. Further analysis showed that usually the posture modifications desired by the users lie in a linear subspace of the motor space. Interestingly this subspace seems to be highly correlated to the subspace where the motion being developed lies. This fact suggests us that the keyframes of the motion that the user is creating could be used to improve the interpretation of tactile instructions. All these findings are reported, in a concise form, in a paper under review at the time of writing.

Chapter 5 showed how to apply the teaching by touching paradigm to the programming of CPGs. In particular Section 5.1 provided a survey on the types of oscillators and coupling found in literature. Section 5.2 discussed the design of a predictable network, and in Section 5.3 a simple protocol to translate tactile instructions to CPG parameter modification is introduced. The chapter concludes with a very brief description of the system implementation, reported in Section 5.4, and an analysis of the experimental results, given in Section 5.5. Comparison of the motion developed by the user with a motion obtained by automatic parameter optimization suggested that users, unconsciously, optimize many different criteria, that are not optimized by automatic algorithms if not provided explicitly in the evaluation function. Part of these concepts were reported in [216]. A more complete discussion can be found in [217].

Chapter 6 presented two solutions for increasing the robustness of motion. In particular, Section 6.1 reported a solution based on a motion graph. The idea, implementation details and experimental results were published in [218].

Section 6.2 provided an alternative solution based on biologically inspired approach. The solution removes heuristics used in the motion graph approach, improving the approach generality. More precisely, Subection 6.2.1 described bacteria chemotaxis, the behavior mimicked by the algorithm we presented. Subsection 6.2.2 presented a first version of the algorithm and Subsection 6.2.3 reported the related experiments. These results can be found in [219].

Subsection 6.2.4 analyzed generality limitations of the algorithm, and Subsection 6.2.5 proposed a new version of the algorithm. This second version was compared to the first one in Subsection 6.2.6, showing improved generality and performance. The same results were published in [220]. Finally, Subsection 6.2.7 presented preliminary experiments conducted with a humanoid robot.

Appendix A will give details on the simulator and on a library developed for the realization of the systems presented in the thesis. A short description of demonstration programs built using these libraries will be given. A very brief overview of the concept underlying this toolset can be found in [221].

The work presented here can be improved and extended in multiple ways, as alternative solutions can be readily thought for most of the problems tackled here. One of the points that present the most wide variety of possible alternatives is the touch interpreter algorithm.

The version presented here uses a memory based approach, kernel regression. Testing other memory based solutions, like local linear regression [90], and performing prediction accuracy comparisons is of high interest. Additionally, the application of model based approaches is worth investigating. More precisely, directed acyclic graphical models appears to be a feasible solution for the estimation of tactile instruction.

Section 3.1 provided a conceptual formalization of the tactile instructions, schematized in Fig. 3.4. Briefly, users have an intended motion modification they want to apply. Depending on some of the features of the physical context that they unconsciously perceive and on their own way of teaching, they provide a tactile instruction. The touch instruction interpreter aims at constructing the inverse mapping from touch instruction and physical context to intended motion modification, by using examples consisting in tuples that include the touch instruction, the context and a directly communicated motion modification.

The translation of such conceptual formalization to a generative model is straightforward. Figure 7.1 provides a possible model. In brief, two multinomial random variables, C and U, model the perceived context and the level of abstraction adopted by the user, respectively. The multivariate Gaussians B and T represent the robot's posture and

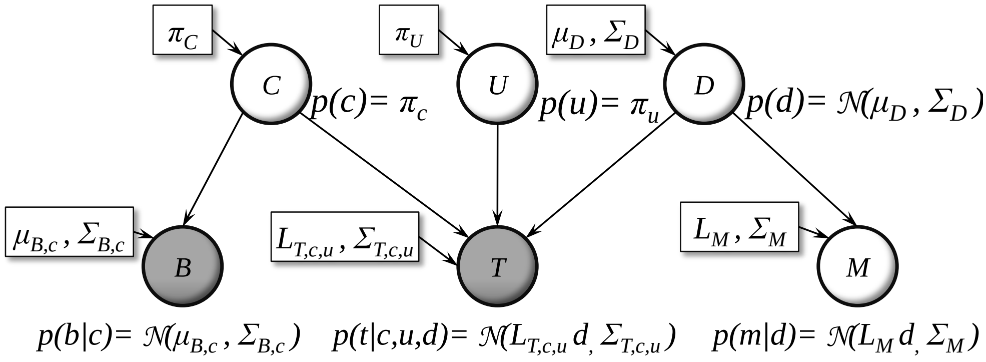


Figure 7.1: A possible generative model for the interpretation of tactile instructions.

orientation in the physical world and the touch applied by the user, respectively. The multivariate Gaussians M and D correspond to the movement taught as the response to a touch instruction and to the movement actually desired when providing the touch. Finally, the matrices $L_{T_{c,u}}$ and L_M, model how the desired movement is mapped to touches and actual movements.

As for other research direction that can be taken in future works, the main shortcoming of the present, supervised learning solution is the need to provide the meaning of tactile instructions. In fact, the supervised learning approach is a good solution if the user has to teach many tasks to the robot, because initially the teacher will need to teach the meaning of many tactile instructions, but, as time goes on, he or she will be able to teach the robot new tasks by mainly reusing previously taught tactile instructions.

However, if the user needs to teach movements to the robot only occasionally, teaching the meaning of tactile instructions could constitute a non-negligible overhead. Furthermore, since human learners do not need to be taught the meaning of tactile instructions, this process could be unnatural for users who interact for the first time with a humanoid robot.

Therefore, removing the need for every user to teach the meaning of all tactile instructions, making the teaching process essentially identical to the one used between instructors and their human trainees, appears of great interest.

As explained beforehand, devising a model of the meaning of tactile instructions in human-human communication and porting it to the robotics domain is very difficult. Touch is a very articulated communication means, exhibiting many features such as the location of its application, its force direction, intensity and time. Performing quantitative measurements in the interaction between humans is very complicated. However, as seen in this thesis, acquiring data during the interaction with a humanoid robot is much easier. In particular, the current supervised learning technique could be used to collect data from a statistically significant set of users.

The idea is therefore to employ a humanoid for acquiring tactile interaction data from a sufficient number of users, construct a model for the interpretation of touch instructions from these data and use the resulting model for the interpretation of touches of new users.

As we saw in Section 3.5, the way the instructions should be interpreted depends on the user. When trained with the data from multiple users, the current supervised learning system would just merge all the data together, and the result would be probably unsatisfactory for many new users.

Conversely, if the way the user teaches is modeled explicitly, then when a new user faces the robot, its teaching way can be estimated, and a correct interpreter could be performed. A possible and elegant solution for this explicit modeling would be, for instance, the described generative model. Examples of body posture, tactile pattern and the desired joint movement could be collected using M3-Neony. These data could be used to determine the model parameters, depicted by rectangles in Figure 7.1, using the Expectation Maximization algorithm, in a way analogous to [222].

Such a system could also be made adaptive by imposing that when the confidence on the meaning of the tactile instruction is low, the humanoid will automatically make its actuators passive, similar to a trainee that relaxes his muscles telling his coach "please show me how I should do it". Detecting the forces applied by the user in a way similar to kinesthetic demonstration and updating the model parameters to include the new data will make the robot adapt to its user over time, similar to how humans understand a teacher more and more as the time spent together gets longer.

Last but not least, future versions of the system should explicitly include heuristics for features usually desired by the users, like balance, posture symmetries or movement smoothness. These heuristics could be activated with weights that depend on the robot's state and used to apply small corrections to a first, "naive" interpretation.

For instance, when the user applies a correction to the right arm posture and this is going to reach a position close to the one of the left arm, the robot could estimate that the user is trying to make the arms assume the same position. In such case, the robot could try to apply small variations to the first interpretation of the tactile instruction in order to make the robot assume a completely symmetric arm position. Conversely, when the arm positions is completely different, the robot could disable the heuristic and do not apply any correction to reach the symmetry.

In conclusion, the implementations of the teaching by touching presented in this thesis should be considered as a starting point for a new, powerful and user friendly human robot communication based on direct tactile interaction.

A

A toolset for humanoid robot simulation

This chapter describes very briefly the main design principles of the software developed for the research topics previously presented. Particular attention will be given to *pplayer*, a software used both for simulation and control of M3-Neony. The purpose of this chapter is threefold. First of all, it allows a more concrete comprehension of the tools used for the experiments reported in the previous chapters. Secondly, it provides a brief introduction of the software available to users interested in using M3-Neony. Finally, it is intended for relieving other users from the development of similar software tools, since the ones presented here can be easily reused for newly developed robots.

A.1 Introduction

In the case of mobile robots we notice that some projects reached the maturity level and are employed by many researchers. For instance Player and Stage [223] nearly constitute a de facto standard, and their usage allows fast prototyping of new algorithms without the need of reimplementing graphical user interfaces, and, for many commercial robots, the software for the robot control.

For humanoid robots, instead, up to date no architecture seems to have spread among different research groups. Most of the works are in fact based on custom-made simulators. Sometimes these simulators are released as open source, as in the case of Gazebo [224] and SimRobot [225], or are sold as a commercial product, as for Webots [226] and Microsoft Robotics Studio, but none of them has reached the popularity that Player and Stage have in the mobile robot community.

In many works [227, 228, 229], a library for simulating the dynamics of rigid bodies called ODE (Open Dynamics Engine), which is already the simulation engine of existing simulators (Gazebo, SimsPark, Webots, etc.), is directly employed. The success of this library is surely due to its simplicity. In fact, in a few hours, the complete API description[1] can be mastered, and the examples provided with the code are a very good starting point of simulations required for fast prototyping of new algorithms. Conversely,

[1] `http://www.ode.org/ode-latest-userguide.pdf`

projects that provide more sophisticated simulations (see [230] for a review) are often not employed by researchers because of the complexity of their API. For instance, Bullet is a very complete library for simulations. It allows simulations of rigid bodies as well as soft bodies and that its integration with Blender [231] allows photorealistic renderings. However, realizing even simple projects is quite difficult, so most of the researches resort to simpler libraries such as ODE.

Nonetheless, ODE is designed for generic simulation of rigid bodies, and most of the researchers reinvent the wheel by implementing wrapper classes used for modeling humanoid robots. Expressly, usually researchers simulate the servomotors by an hinge joint and develop a parser that converts a description of physical dimensions of the robot to a set of rigid bodies in ODE. A rudimentary library, called drawstuff and distributed with the ODE examples, is usually employed for visualization. While the library is very simple to use, its design usually requires mixing code for visualization of the objects and simulation steps. Moreover the library does not implement basic interaction, like picking objects in the simulated world, which are usually desired by many researchers. Every time such interaction is needed, the drawstuff library is hacked or reimplemented by the various research groups, spending time on an extension that other people already developed in their projects.

Our purpose is therefore to implement basic functions required for humanoid robot simulations, while keeping the code simple enough to be usable in a short time. We even aspire at having the code to be easily understandable so that researchers are not forced to use it as a black box but can customize it for their own purposes without difficulties.

Our library does not therefore aim at substituting articulated projects, like SimSpark, presented in [232], which provides very advanced functions like a path-name space mapping for management of objects or the simulation of a complete match between robots. The target of our project is thus similar to the one of Simbad [233], although we stress the fast prototyping aspect more than the educational purpose. Furthermore, we preferred to employ C++ instead of Java.

The main reason lies in the fact that we intend to seamlessly port our code between the simulation environment and the real world, but while Java Virtual Machines could not be available on real robots, C++ compilers are available for all the platforms we are interested in. We also preferred to employ ODE as the simulation physics library, which is widely used and therefore debugged, instead of employing custom simulation engines as in Simbad or MuRoSimF [234]. Since the simulation completely relies on ODE, our toolset does not present particular difference in terms of simulation speed and accuracy compared to other ODE based simulators, like Webots or Simspark. The simulator was used in a wide set of experiments, among which we can cite, for instance, the tests on user dependence described in Section 3.5.

The code was developed to be as self-contained as possible and we therefore reduced the number of employed libraries to a minimum set of widespread ones. In fact, we considered reducing the dependencies as a key point, because most users may find very cumbersome installing many libraries just to be able to compile a simulator. Furthermore, in the open source world backward compatibility is not guaranteed. These incompatibilities could prevent the code from compiling in the future and could require the users to install on their systems older versions of the libraries or to wait for code fixes. Section A.2 describes the basic functions. Section A.3 reports some of the ideas underlying its design and Section A.4 provides a short list of examples of programs that use the library.

A.2 Functionalities

Our library for robot simulations is a part of a bigger set of software tools for robot control, robot simulation and motion development. The main software, called pplayer, is essentially a server that listens for commands on a socket and actuates the real robot or a simulated robot, depending on the compilation directives.

We decided to employ commands that consist of simple strings in ASCII format. This slightly reduces the efficiency of the communication, but allows a lot easier debugging and permits controlling the robot by a simple telnet connection. The commands include basic functions like switching on and off the motors or reading the robot sensors. Expressly, the system provides two ways of reading the sensor values, polling or a proactive way in which sensory data are sent as soon as they are collected.

Commands can be launched both in a synchronous and asynchronous way, i.e. in the synchronous mode the next command is executed when the previous one has finished its execution while in the asynchronous mode commands are executed in parallel. In our implementation, in order to resemble the Linux bash, every command is launched in an asynchronous way by simply placing an *&* after the command. A *ps*-like command, as in the Unix world, allows seeing the commands running in a determinate moment, in order to ease debugging.

Besides the basic sensor reading and actuators activation, pplayer server provides functions for execution of movements as well. Expressly, most of the open loop humanoid motion executions are based on the concept of key-frames: the angles of all the joints are defined for certain time instants, termed keyframes, and intermediate postures are calculated by interpolation. The software comprises commands for transferring the keyframes to the robot, as well as for playing the motion with an arbitrary playing speed, starting time and number of iterations.

The software toolset includes a GUI for the development of the motions, shown in Fig. A.1. The interface provides all the functions common to classical commercial editors like VStone's Robovie Maker[2] or Kondo's Heart2Heart[3]. Notice that all of the

[2]`http://www.vstone.co.jp`
[3]`http://www.kondo-robot.co.jp`

elements, like the sliders that allow modifying the joint angles of a frame, are generated automatically by simply passing an object that describes the physical structure of the robot. The same robot description is used for generating an ODE model. The simulator is capable of simulating multiple robots, as shown in Fig. A.2, and allows easy insertion of objects in the environment, as visible in Fig. A.3.

Rendering of the simulation can be enabled or disabled. When rendering is enabled it is possible to interact with the mouse to move and rotate robots and objects in the simulated environment. The rendering can be saved as a video by simply setting a filename parameter in the function that enables the drawing. The simulator simulates virtual cameras, which can be easily attached to any part of a robot, and virtual touch sensors. In particular, while ODE provides contact forces between bodies, our API allows detecting the forces that act on a single face.

The software toolset is composed by the following components

- A library for OS dependent functions, e.g. retrieving the date (in milliseconds) or creating a socket
- Math utilities (matrix computations, generation of random numbers with Gaussian distribution, integration of differential equations, description of graphs, etc.)
- Networking utilities (client sockets, multi-client server socket, classes for simple job dispatching over the network, etc.)
- Utility classes for the creation of windows, keyboard and mouse callbacks, OpenGL renderings and recording of videos of the generated renderings
- Classes for robot modeling
- Classes for the simulation (Object Oriented wrapper of ODE) and its visualization
- A graphical interface developed with gtkmm [4], that allows the development of robot motions

A.3 Design policies

A.3.1 Basic classes

An important element in our system is the *Motors* superclass. The *Motors* class represent a set of motors, which can correspond to real servomotors, simulated ones or more abstract objects. Among the subclasses of *Motors* we can cite

- *PrintMotors*: saves the postures to a log file

[4] A C++ wrapper of gtk, `http://www.gtkmm.org/`

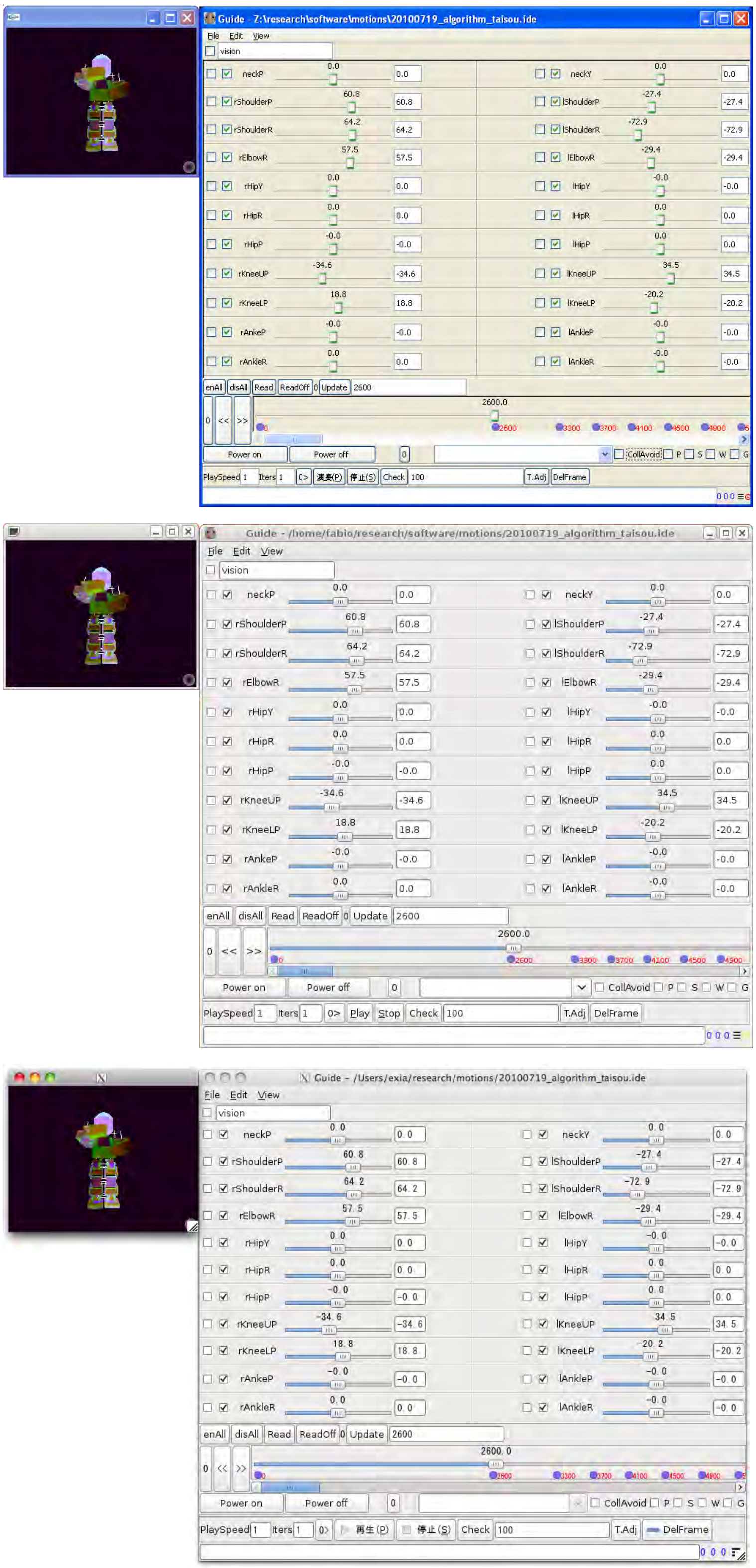


Figure A.1: A classical slider based interface automatically generated for a specific robot using our library, running in Windows XP, Ubuntu 9.10 and Mac OSX Snow Leopard. The interface includes collision detection algorithms which display graphically the parts that would collide when moving the actual robot.

Figure A.2: Two M3-Neony robots in the simulated world during a one vs one soccer simulation.

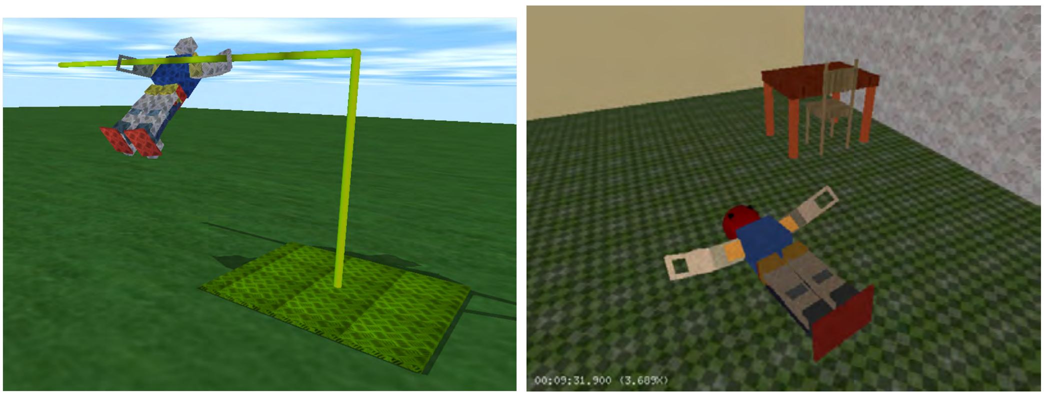


Figure A.3: Views of the robot and some objects in the simulated world.

- *HubMotors*: allows to attach multiple *Motors* objects to the *HubMotors* so that when the *HubMotors* is rotated, all the attached *Motors* are rotated simultaneously.
- *CollPrevMotors*: when a *Motors* is attached to a *CollPrevMotors* and the *CollPrevMotors* object is rotated, a fast collision detection computation is performed at each rotation, and each motor of the *Motors* object is rotated to the maximum extent that does not perform any self-collision of the robot (the computation is performed as if the motors were turned one at a time, in an order specified by the user).

Another basic class underlying our system is the *Policy* class. A policy is essentially a parametrized function that is able to return joint angles for any time instant. For instance, a classical key-frame based representation have as parameters the various frames (with their time) and the values returned for intermediate times are the interpolation of the previous and following keyframe (*FramePolicy* class). Another subclass of policy, *Sinpolicy*, moves the joints using sine waves. Although very simple, by appropriately setting the frequencies, amplitudes, phases and offset positions, simple sinusoidal movements are able to generate a great variety of movements. In our code, Central Pattern Generators were represented as a subclass of *Policy* as well (*HopfPolicy*), by introducing variables describing the internal state.

A.3.2 Robot modeling

An important set of classes is used for modeling robots. In detail, any robot is modeled by describing its kinematic chain in terms of *SkeletonNode* elements. The nodes are attached in a graph structure, similarly to what is done in OpenSceneGraph [235]. Parts, described by *Part* objects, are attached to the graph to describe the shape and mass distribution of the robot. The *Part* objects can be described by a set of *Element* objects, that provide the description for basic shapes like spheres or cuboids. The *Element* objects provide easy customization of the appearance of the robot. For instance, simply specifying the texture filename allows having photorealistic renderings.

Several functionalities can be added to the *Element* objects by using subclasses of the *Customization* class. For example, by the simple insertion of a *ForceSensorBoxCustomization* object (subclass of *Customization*) in an *Element* class, it is possible to describe the presence of a touch sensor on each face of the element. The representation in terms of *SkeletonNode*, *Part*, *Element* and *Customization* objects can be obtained automatically by an XML file, allowing easy development and debugging of new robot models. The modeling is completely independent from the libraries used for simulation, so it would be possible to easily include the support for other physics simulation libraries like Bullet. In particular, once the PAL project[5] is mature enough, we could employ

[5] Physics Abstraction Layer (`http://www.adrianboeing.com/pal/index.html`), a project aiming at

PAL in place of ODE to have a completely transparent way to change the underlying physics simulation. Expressly, in our current library, the model in terms of ODE primitive objects is automatically generated when a *SkeletonNode* graph is inserted into an *OdeWorld* object by the *addRobot* function. Multi-robot simulation is therefore trivial. Adding static objects to the world, like the horizontal bar in Fig A.3 is very simple as well, it is in fact sufficient to pass a *SkeletonNode* graph to the *addStaticObject* function of the *OdeWorld* class. During the simulation, all the information regarding the robot can be easily obtained by functions like *getPartRototranslation(double *rotoTra)* that returns in *rotoTra* the roto-translation matrix of a part, expressed in the absolute reference frame.

A.3.3 Visualization

Creation of windows to display, for instance, the rendering of the simulation from a third point of view or from a robot-mounted camera, is done using *GlPrimitiveCanvas* objects. To add a window, it is in fact sufficient to add an element to the vector of *GlPrimitiveCanvas* objects passed to the *glDraw* function. Each window allows chaining a series of mouse (*MouseInteraction* objects) and keyboard managers (*KeyboardInteraction* objects) that are called in sequence to deal with the user actions.

The library provides callbacks that allow picking and placing of objects in the simulated 3D world as well as movements of the camera observing the scene. A keyboard managers that allow pausing the simulation, restoring the robot position to a standard state, switching the visualization to a wireframe mode, activating a rotating camera and so forth is provided as well.

Simulations without the scene rendering (to decrease the computational cost or allow distributed simulations over command line sessions) can be obtained by simply avoiding to invoke the *glDraw* function. In the cases when the simulation of the camera is necessary, the library switches to offline rendering by the usage of the OSMesa library, allowing simulations on systems that are not running any X-server or equivalent.

A.3.4 Command Parsing

As stated in the introduction, the *pplayer* program opens a TCP socket and listens for commands. This is done by instantiating an *AsciiServer* object, i.e. a multi-client server that expects non binary commands. The command parsing is very modular, to allow easy extension or reorganizations of the command set. The *AsciiServer* provides the *registerCommandParser* function, that enables adding a *CommandParser* object to a chain of parsers. Each command parser has to specify its name, and the commands it

---

providing a common interface to different dynamic engines.

manages. It can also expose some commands to the "shortcut" mode. Every command is assumed to be composed of

1. *Z*, followed by the name of the parser that should manage it
2. a space
3. an action (i.e. a string)
4. a variable number of parameters, separated by spaces
5. the terminating character, in our implementation ";"

When a command is exported to the "shortcut" mode, the command parser name does not need to be specified, i.e. the commands starts directly with the action.

The shortcut mode allows reducing typing for frequently sent commands, as well as to make two managers handle the same command. In this case, the parsing priority can be specified, as well as whether after using a manager to parse the command, other managers should be invoked as well. Table A.1 reports the commands currently available. Note that the syntax is highly variable, and designed mainly with the idea of providing a handy way to interact with the robot by a command line interface.

Table A.1: Listing of the commands available

parser	command	meaning
	H	display the commands summary
	HP	show the running processes (commands)
[ZMO]	P <joint> <0|1>	set the power of joint <joint> off (0) or on (1)
[ZMO]	R JR <joint> <angle>	rotate joint <joint> to angle <angle> (in radians)
ZMO	FR	bypass any optimization and force the rotation given by the following R JR command
[ZMO]	D JR <joint> [<times>]	read the position of joint <joint> controlling to get similar results <times> times.
[ZSM]	R JI <joint> <value>	rotate the joint with <joint> by setting the desired raw (potentiometer) position <value>
[ZSM]	D JI <joint> [<times>]	read the raw (potentiometer) position of joint <joint> controlling to get similar results <times> times.
ZMO	TD	stop continuous readings of the position

ZSM	TD	stop continuous readings of the raw position
	TD	stop any continuous motor position reading
[ZSM]	O <joint> <min> <max> <zero>	replace the calibration of the joint <joint> with values <min> <max> <zero>
ZSM	PID <joint> <p> <d> <e> <i>	set the PID parameters <p> <d> <e> <i> for joint <joint> (see sect. 4.1)
ZSM	VG VGET <joint> <addr> <size>	get <size> bytes from the ROM of joint <joint> starting from address <addr>
ZSM	VS VSET joint addr size value	set <size> bytes from the ROM of joint <joint> starting from address <addr> with the value <value> (little endian)
ZSM	FR Force Rotate joint	bypass any optimization and force the rotation given by the following R JI command
[ZTS]	U U<mode>[C] [<sensor ids>]	read the touch sensors <sensor ids>, all if none specified, continuously if C is present. <mode> is N for being notified of the value of all the sensors exceeding a threshold, P for powering off the limbs grabbed (see sec. 4.2) and I for retrieving the raw value measured by all the sensors
[ZTS]	TU	stop continuous readings of the touch sensors
[ZGY]	G G<mode>[C]	read the accelerometer and gyroscope information, continuously if C is present. <mode> is I for the raw readings and R for the estimated robot rototranslation matrix
[ZPP]	L <start time> <playing speed> [<iterations>]	play the current motion from <start time> at <playing speed> times the normal speed for <iterations> times (1 if unspecified).
[ZGY]	TG	stop continuous readings of the accelerometers and gyro
[ZPP]	T	stop playing a motion
[ZPP]	Y <action><type> [<params>]	operate on the current motion (policy). <type> is F for a frame based policy and S for moving all the joints by sine waves. <action> is S for selecting the policy <type> and P for setting its parameters (notion specification).
[ZPP]	S <joint> <time-position sequence>	set the positions (frames) of the joint <joint> for the policy type <F>

ZPP	RS[Z] <t> [<t2>] <target joint values> [<start joint values>]	rotate slowly (in <t> milliseconds) all the joints from the position <start joint values> (the current if unspecified) to the <target joint values>. If Z is specified, to avoid collisions, the robot assumes intermediately its zero position in <t> milliseconds and then moves to the target position in <t2> milliseconds.
ZMP	R <t> <id> <target joint values>	rotate all the motors slowly (in <t> milliseconds) to the position <target joint values> and return the <id> if no other ZMP R commands are provided in <t> milliseconds, otherwise accept the new ZMP R command and ignore the current one.
[ZSIM]	VG <variable name>	get the value of variable <variable name> of the simulator. Currently only MU (friction coefficient) is supported.
[ZSIM]	VS <variable name> <variable value>	set the value of variable <variable name> of the simulator. Currently only MU (friction coefficient) is supported.
		Notes: [.] indicates an optional value, < . > indicates a parameter. <joint> set to -1 has the meaning of "all joints". Similarly <times> set to -1 is used to read continuously and <iterations> set to -1 indicates infinite iterations.

A.3.5 Client applications

The interface for motion development, reported in Fig. A.1, as well as most of the software developed for experiments and demonstrations, acts as clients of the pplayer server. In detail, all the software uses the *AsciiClient* class, that provides functions for rotating the motors, reading the potentiometers or the touch sensor values and so forth. The TCP/IP communication between the *AsciiClient* class and *pplayer* is completely transparent to the client applications. All applications operate on the robot by simply invoking the *AsciiClient* functions without any knowledge of the underlying communication. In other terms, all applications invoke the *AsciiClient* functions as they were operating directly with the robot's hardware.

A.3.6 Dependencies

All of the software is written in standard C++. The compilation was tested under g++ versions 3.3, 3.4, 4.2, both in Linux and Cygwin environments and under Microsoft Visual Studio 2005. The code strongly uses the Standard Template Library (STL) library to improve its readability. Conversely, compiler dependent libraries like the Microsoft Foundation Class Library (MFC) are avoided to assure code portability.

Each module of the system can be enabled or disabled by compilation directives. This allows the exclusion of all things which are not relevant for particular applications, simplifying the code and reducing the libraries required. In detail, the minimal requirements consist of the pthread library on Linux and the Winsock2 library on Windows (when the source is compiled with Visual Studio). Enabling the simulation requires the ODE library. In particular, the current code assumes the ODE library to be compiled with the *–enable-double-precision* option.

Enabling the rendering requires the GLUT library. If the possibility to save videos is included, then the Intel OpenCV library (and the related highgui) must be included. Furthermore, if offline rendering is desired, then the OSMesa library should be installed. When the possibility to parse models in the XML format is activated by the compilation directives, the TinyXml[6] library is required. The TinyXml is a minimal C++ XML parser that can be easily integrated into other programs by simply including its object files at compilation time.

Finally, if the graphical user interfaces are compiled, the gtkmm library, a C++ wrapper of Gtk, is required.

A.4 Usage Examples

The libraries were employed for the realization of all the experiments presented in the previous chapters. Notably, they are currently being used by the JEAP RoboCup team at Osaka University [236] as well.

Several demonstrations, that essentially consist of simple client programs that employ the *AsciiClient* class, were developed as well. Fig. A.4 shows some keyframes of a simple demonstration of the sensors available on the robot[7]. The program consists of a state machine (implemented with the state design pattern [237]). At the beginning of the demonstration the robot is sleeping. In this state, the robot waits for sounds of volume crossing a certain threshold, as shown in Fig. A.4(a). When the user claps his hands, as in Fig. A.4(b), the robots starts crying, as visible in Fig. A.4(c). When the accelerometers indicate that the user lifted the robot with the head upwards, as shown in Fig. A.4(d), then the robot stops crying. In this state the robot switches off the motors, and laughs

[6] See `http://sourceforge.net/projects/tinyxml/`

[7] A video is available at `http://robotics.dei.unipd.it/~fabiodl/video.php?sensorDemo`

when the sensors on the belly are pressed, as in Fig: A.4(e). Finally, after laughing for three times, the robot starts to follow red objects, as shown in figures A.4(f) and A.4(f).

Another demonstration is related to biological fluctuations, a very general phenomena found in nature [238, 239, 240]. For instance, some cells can adapt to environmental changes by altering their pattern of gene expressions and metabolic flux distribution. These adaptive responses are usually explained by the signal transduction mechanisms, a sort of a pre-wired logic circuit that modifies the gene expression depending on the environmental condition. However, not all of the adaptations can be explained in terms of transduction mechanisms and in [241] it was shown that cells can select states most favorable for their survival among a large number of other possible states simply because the cells that grow more (are more adapt to the environment) present a less stochastic behavior. Biological fluctuations are often modeled for continuous time systems as

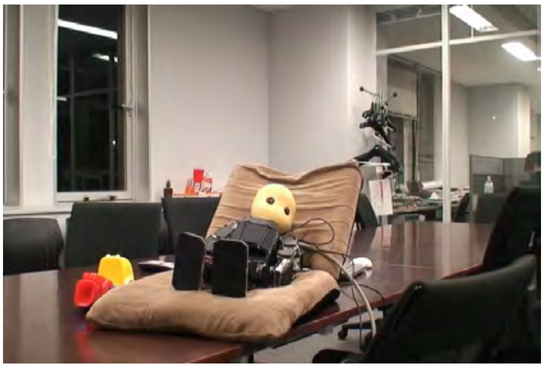
$$\dot{x} = Af(x) + \eta. \tag{A.1}$$

where $x \in \mathbb{R}^m$ is the control signal or represents the value of some parameter that determines the behavior (of the animal or, in our case, of the robot), $f : \mathbb{R}^m \to \mathbb{R}^m$ is a deterministic function of the current value of x, η is a random variable and $A : \mathbb{R}^n \to \mathbb{R}$ is a function, called "activity", that indicates the fitness, or "quality" of a particular state of the living being/robot. Intuitively, when the state is getting better, the value of A increases and the control actions follow f and become mainly deterministic, while when the conditions worsen the control becomes more and more stochastic.

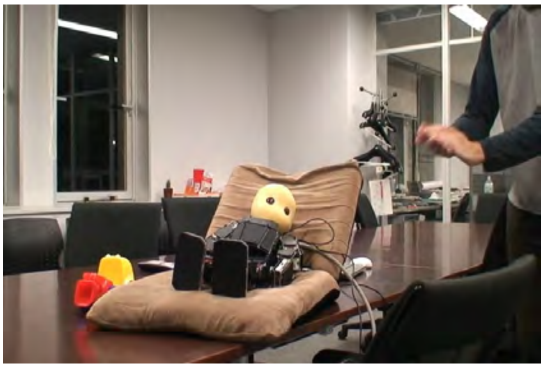
If the states are discrete, the same effect can be obtained using a Markov chain and assuming the transition probability from a state to itself as an increasing function of the state fitness. In particular, we conducted an experiment where we controlled each joint of a mobile robot by sine waves, and define the (crawling) velocity as the fitness of a state. The frequency, amplitude and an offset values along which the motors oscillate were set to a fixed value. The phases, i.e. the timing differences between the various servomotors, were instead varied for 4 of the motors (the shoulder and the hip pitch joints) to obtain different behaviors. In detail we prepared $N = 8$ states corresponding to 8 different phase settings, as shown in Fig. A.5.

We assumed the transition probability from a state to itself to be equal to a, and from to a state to another to be equal to $(1-a)/(N-1)$, with $a = \sigma\,(v - v_0)$, where σ is the sigmoid function, v is the robot velocity and v_0 is a constant. We verified that with this simple setup, the robot is able to find the phase setting most suitable to crawl and to change it when the performance decreases because, for instance, an obstacle prevents the robot movement[8]. Also in this case, the code is very limited, and is strongly based on calls to the *AsciiClient* class methods. In fact, the whole program consist of a single file of few hundreds of line of code.

[8]A video is available at `http://robotics.dei.unipd.it/~fabiodl/video.php?yuragiCrawl`



(a)



(b)

(c)

(d)

(e)

(f)

(g)

Figure A.4: Scenes of a demonstration of the robot sensors: the microphone, the accelerometer, the touch sensors and the cameras.

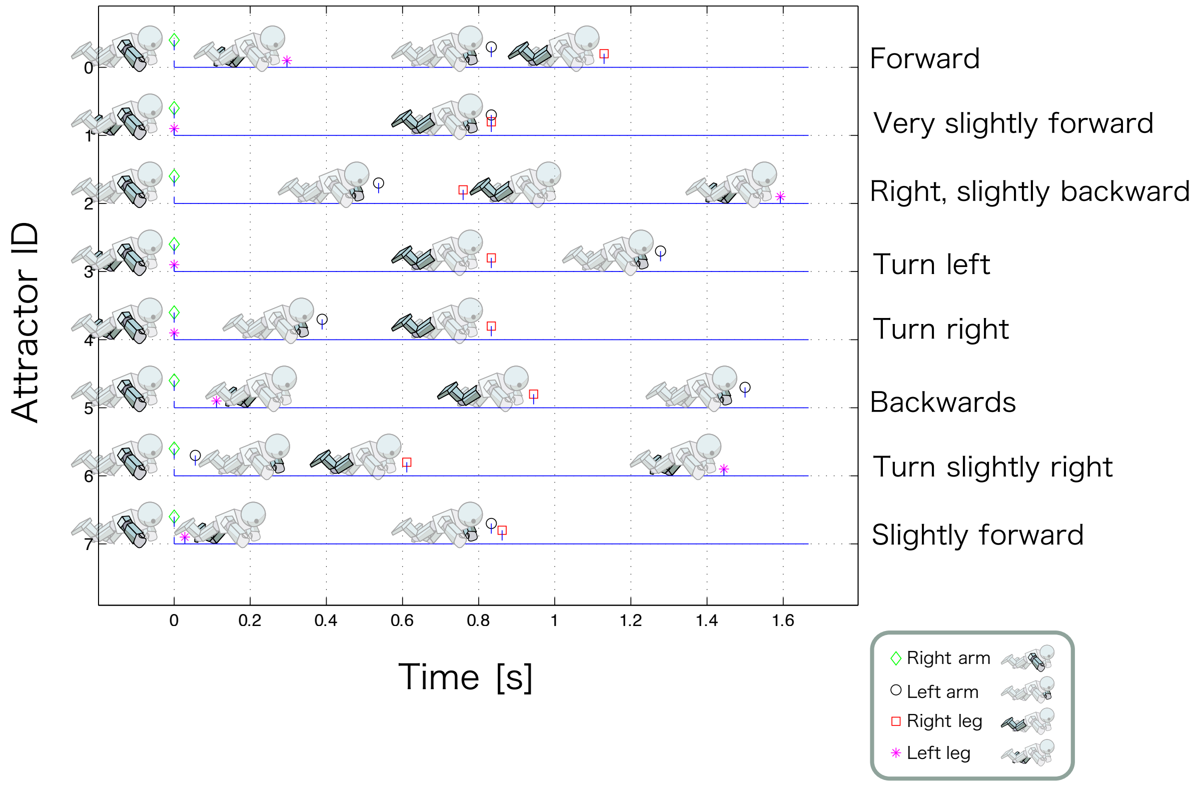


Figure A.5: By changing the phases of 4 motors (right shoulder pitch, left shoulder pitch, right hip pitch, left hip pitch) several behaviors can be obtained. The upper figure schematizes the different settings used in the experiment. In detail the graph reports, for each of the 8 configurations, the time instants for which each limb is brought in its foremost position.

Bibliography

[1] K. Yokoi, K. Nakashima and Y. Yanagihara, *A Tele-operated Humanoid Operator*, M. Ang and O. Khatib, eds., Experimental Robotics IX, *Springer Tracts in Advanced Robotics*, vol. 21, pp. 229–239, Springer Berlin / Heidelberg (**2006**).

[2] K. Nickel and R. Stiefelhagen, *Visual recognition of pointing gestures for human-robot interaction*, Image and Vision Computing, vol. 25, no. 12, pp. 1875–1884 (**2007**).

[3] O. Sugiyama, T. Kanda, M. Imai, H. Ishiguro and N. Hagita, *Humanlike conversation with gestures and verbal cues based on a three-layer attention-drawing model*, Connection science, vol. 18, pp. 379–402 (**2006**).

[4] J. Hirth, N. Schmitz and K. Berns, *Emotional Architecture for the Humanoid Robot Head ROMAN*, 2007 IEEE International Conference on Robotics and Automation (ICRA 2007), pp. 2150–2155, Roma, Italy (**2007**).

[5] M. Asada, K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, Y. Yoshikawa, M. Ogino and C. Yoshida, *Cognitive developmental robotics: a survey*, IEEE Transactions on Autonomous Mental Development, vol. 1, no. 1, pp. 12–34 (**2009**).

[6] T. Minato, Y. Yoshikawa, T. Noda, S. Ikemoto, H. Ishiguro and M. Asada, *CB2: A Child Robot with Biomimetic Body for Cognitive Developmental Robotics*, IEEE-RAS 7th International Conference on Humanoid Robots (Humanoids 2007), pp. 557–562, Pittsburg, PA, USA (**2007**).

[7] N. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A. Ijspeert, M. Carrozza and D. Caldwell, *iCub: the design and realization of an open humanoid platform for cognitive and neuroscience research*, Advanced Robotics, vol. 21, pp. 1151–1175 (**2007**).

[8] R. E. Bellman, Adaptive Control Processes: A Guided Tour, Princeton University Press (**1961**).

[9] J. T. Schwartz and M. Sharir, *On the "piano movers" problem. II. General techniques for computing topological properties of real algebraic manifolds*, Advances in Applied Mathematics, vol. 4, no. 3, pp. 298–351 (**1983**).

[10] S. M. LaValle and J. J. Kuffner, *Randomized kinodynamic planning*, 1999 IEEE International Conference on Robotics and Automation (ICRA 1999), vol. 1, pp. 473–479, Detroit, MI, USA (**1999**).

[11] J. J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba and H. Inoue, *Dynamically-Stable Motion Planning for Humanoid Robots*, Autonomous Robots, vol. 12, pp. 105–118 (**2002**).

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

[13] S. Iida, S. Kato, K. Kuwayama, T. Kunitachi, M. Kanoh and H. Itoh, *Humanoid robot control based on reinforcement learning*, Proceedings of the 2004 International Symposium on Micro-Nanomechatronics and Human Science and The Fourth Symposium Micro-Nanomechatronics for Information-Based Society, pp. 353–358, Nagoya, Japan (**2004**).

[14] Y. Nakamura, T. Mori, M. Sato and S. Ishii, *Reinforcement learning for a biped robot based on a CPG-actor-critic method*, Neural Networks, vol. 20, no. 6, pp. 723–735 (**2007**).

[15] D. Chapman and L. P. Kaelbling, *Input generalization in delayed reinforcement learning: an algorithm and performance comparisons*, Proceedings of the 12th International joint Conference on Artificial intelligence, vol. 2, pp. 726–731, Sydney, New South Wales, Australia (**1991**).

[16] A. K. Mccallum, *Learning to Use Selective Attention and Short-Term Memory in Sequential Tasks*, 4th International Conference on Simulation of Adaptive Behaviour (SAB 96), pp. 315–324, MIT Press, Cape Cod, MA, USA (**1996**).

[17] S. D. Whitehead and D. H. Ballard, *Active perception and reinforcement learning*, Proceedings of the Seventh International Conference on Machine learning, pp. 179–188, Austin, Texas, United States (**1990**).

[18] T. Nishi, Y. Takahashi and M. Asada, *Incremental Purposive Behavior Acquisition based on Modular Learning System*, Proceedings of the 9th International Conference on Intelligent Autonomous Autonomous Systems (IAS-9), pp. 702–709, Tokyo, Japan (**2006**).

[19] S. P. Singh, *Transfer of Learning by Composing Solutions of Elemental Sequential Tasks*, Machine Learning, vol. 8, pp. 323–339 (**1992**).

[20] M. J. Mataric, *Reward Functions for Accelerated Learning*, Proceedings of the Eleventh International Conference on Machine Learning, pp. 181–189, Rutgers University, New Brunswick, NJ, USA (**1994**).

[21] G. Dip, V. Prahlad and P. d. Kien, *Genetic algorithm-based optimal bipedal walking gait synthesis considering tradeoff between stability margin and speed*, Robotica, vol. 27, pp. 355–365 (**2009**).

[22] C. Niehaus, T. Röfer and T. Laue, *Gait Optimization on a Humanoid Robot using Particle Swarm Optimization*, E. Pagello, C. Zhou, E. Menegatti and S. Behnke, eds., Proceedings of the Second Workshop on Humanoid Soccer Robots in conjunction with the 2007 IEEE-RAS International Conference on Humanoid Robots, Pittsburgh, PA, USA (**2007**).

[23] A. Ude, C. Atkeson and M. Riley, *Planning of joint trajectories for humanoid robots using B-spline wavelets*, 2000 IEEE International Conference on Robotics and Automation (ICRA 2000), vol. 3, pp. 2223–2228, San Francisco, CA, USA (**2000**).

[24] A. Ude, C. G. Atkeson and M. Riley, *Programming full-body movements for humanoid robots by observation*, Robotics and Autonomous Systems, vol. 47, no. 2-3, pp. 93–108 (**2004**).

[25] N. M. Mayer, J. Boedecker and M. Asada, *Robot motion description and real-time management with the Harmonic Motion Description Protocol*, Robotics and Autonomous Systems, vol. 57, no. 8, pp. 870–876 (**2009**).

[26] B. Lim, S. Ra and F. C. Park, *Movement Primitives, Principal Component Analysis, and the Efficient Generation of Natural Motions*, 2005 IEEE International Conference on Robotics and Automation (ICRA 2005), pp. 4630–4635, Barcelona, Spain (**2005**).

[27] F. A. Mussa-Ivaldi, *Nonlinear Force Fields: A Distributed System of Control Primitives for Representing and Learning Movements*, Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation, pp. 84–90, IEEE Computer Society, Monterey, CA, USA (**1997**).

[28] Y. P. Ivanenko, G. Cappellini, N. Dominici, R. E. Poppele and F. Lacquaniti, *Coordination of locomotion with voluntary movements in humans.*, Journal of Neuroscience, vol. 25, no. 31, pp. 7238–53 (**2005**).

[29] M. Santello, M. Flanders and J. F. Soechting, *Postural hand synergies for tool use*, Journal of Neuroscience, vol. 18, no. 23, pp. 10105–10115 (**1998**).

[30] A. Nakazawa, S. Nakaoka and K. Ikeuchi, *Synthesize stylistic human motion from examples*, 2003 IEEE International Conference on Robotics and Automation (ICRA 2003), pp. 3899–3904, Taipei, Taiwan (**2003**).

[31] R. Amit and M. J. Mataric, *Parametric Primitives for Motor Representation and Control*, 2002 IEEE International Conference on Robotics and Automation (ICRA 2002), pp. 863–868, Washington, DC, USA (**2002**).

[32] J. Rosmarin and H. H. Asada, *Synergistic design of a humanoid hand with hybrid DC motor - SMA array actuators embedded in the palm*, 2008 IEEE International Conference on Robotics and Automation (ICRA 2008), pp. 773–778, Pasadena, CA, USA (**2008**).

[33] S. Schaal, J. Peters, J. Nakanishi and A. Ijspeert, *Learning Movement Primitives*, P. Dario and R. Chatila, eds., Robotics Research, *Springer Tracts in Advanced Robotics*, vol. 15, pp. 561–572, Springer Berlin / Heidelberg (**2005**).

[34] A. J. Ijspeert, J. Nakanishi and S. Schaal, *Movement Imitation with Nonlinear Dynamical Systems in Humanoid Robots*, 2002 IEEE International Conference on Robotics and Automation (ICRA 2002), pp. 1398–1403, Washington, DC, USA (**2002**).

[35] A. J. Ijspeert, J. Nakanishi and S. Schaal, *Learning rhythmic movements by demonstration using nonlinear oscillators*, 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002), vol. 1, pp. 958–963, Lausanne, Switzerland (**2002**).

[36] G. Taga and H. Yamaguchi, Y. Shimizu, *Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment*, Biological Cybernetics, vol. 65, pp. 147–159 (**1991**).

[37] G. Endo, J. Morimoto, J. Nakanishi and G. Cheng, *An Empirical Exploration of a Neural Oscillator for Biped Locomotion Control*, 2004 IEEE International Conference on Robotics and Automation (ICRA 2004), pp. 3036–3042, New Orleans, LA, USA (**2004**).

[38] D. M. Armstrong, *Supraspinal contributions to the initiation and control of locomotion in the cat.*, Progress in Neurobiology, vol. 26, pp. 273–361 (**1986**).

[39] M. Riley, A. Ude, C. Atkeson and G. Cheng, *Coaching: An Approach to Efficiently and Intuitively Create Humanoid Robot Behaviors*, IEEE-RAS 6th International Conference on Humanoid Robots (Humanoids 2006), pp. 567–574, Genoa, Italy (**2006**).

[40] B. D. Argall, S. Chernova, M. Veloso and B. Browning, *A survey of robot learning from demonstration*, Robotics and Autonomous Systems, vol. 57, no. 5, pp. 469–483 (**2009**).

[41] S. Calinon, Robot Programming by Demonstration: A Probabilistic Approach, EPFL/CRC Press (**2009**).

[42] J. Peters and S. Schaal, *Policy Gradient Methods for Robotics*, 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006), pp. 2219–2225, Beijing, China (**2006**).

[43] S. Nakaoka, A. Nakazawa, K. Yokoi, H. Hirukawa and K. Ikeuchi, *Generating whole body motions for a biped humanoid robot from captured human dances*, 2003 IEEE International Conference on Robotics and Automation (ICRA 2003), pp. 3905–3910, Taipei, Taiwan (**2003**).

[44] M. Gleicher, *Retargetting motion to new characters*, Proceedings of the 25th annual Conference on Computer graphics and interactive techniques, SIGGRAPH '98, pp. 33–42, Orlando, FL, USA (**1998**).

[45] S. Nakaoka, A. Nakazawa, F. Kanehiro, K. Kaneko, M. Morisawa, H. Hirukawa and K. Ikeuchi, *Learning from Observation Paradigm: Leg Task Models for Enabling a Biped Humanoid Robot to Imitate Human Dances*, The International Journal of Robotics Research, vol. 26, pp. 829–844 (**2007**).

[46] D. D. Ardayfio, Fundamentals of robotics, CRC Press, Dekker Mechanical Engineering (**1987**).

[47] G. Grunwald, G. Schreiber, A. Albu-Schffer and G. Hirzinger, *Touch: The Direct Type of Human Interaction with a Redundand Service Robot*, Proceedings of the 10th IEEE International Workshop on Robot and Human Interactive Communication (ROMAN 2001), pp. 347–352, Bordeaux/Paris, France (**2001**).

[48] Y. Maeda, T. Ushioda and S. Makita, *Easy Robot Programming for Industrial Manipulators by Manual Volume Sweeping*, 2008 IEEE International Conference on Robotics and Automation (ICRA 2008), pp. 19–23, Pasadena, CA, USA (**2008**).

[49] R. Voyles and P. Khosla, *Tactile Gestures for Human/Robot Interaction*, 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 1995), pp. 7–13, Pittsburg, PA, USA (**1995**).

[50] T. Tsuji and T. Ito, *Command recognition by haptic interface on human support robot*, 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009), pp. 3178–3183, St. Louis, MO, USA (**2009**).

[51] H. Ben Amor, E. Berger, D. Vogt and B. Jung, *Kinesthetic Bootstrapping: Teaching Motor Skills to Humanoid Robots through Physical Interaction*, B. Mertsching, M. Hund and Z. Aziz, eds., KI 2009: Advances in Artificial Intelligence, *Lecture Notes in Computer Science*, vol. 5803, pp. 492–499, Springer Berlin / Heidelberg (**2009**).

[52] M. Hersch, F. Guenter, S. Calinon and A. Billard, *Dynamical System Modulation for Robot Learning via Kinesthetic Demonstrations*, IEEE Transactions on Robotics, vol. 24, no. 6, pp. 1463–1467 (**2008**).

[53] G. Pierris and M. G. Lagoudakis, *An interactive tool for designing complex robot motion patterns*, 2009 IEEE International Conference on Robotics and Automation (ICRA 2009), pp. 4013–4018, Kobe, Japan (**2009**).

[54] J. Baltes and J. Anderson, *Complex AI on Small Embedded Systems: Humanoid Robotics Using Mobile Phones*, G. Hoffman, ed., Proceedings of the AAAI Spring Symposium on Intelligence in Embedded Systems, Stanford, CA, USA (**2010**).

[55] J. D. Uk, C. I. Sung, K. Y. Yong, L. D. Hun, K. M. Jong, K. J. Yeong, P. H. Woo and O. J. Hwan, *Kwangwoon University ROBIT 2010 (Kid Size League) Team Description Paper*, RoboCup 2010: Robot Soccer World Cup XIV Proceedings, Singapore, Singapore (**2010**).

[56] C. C. Wong, K. H. Huang, Y. Y. Hu, H. M. Chan, H. C. Chen, C. H. Hung, C. C. Jiang and W. N. Hsu, *Humanoid Soccer Robot Design by TKU Team for Humanoid League of RoboCup 2010*, RoboCup 2010: Robot Soccer World Cup XIV Proceedings, Singapore, Singapore (**2010**).

[57] J. Baltes, S. McCann and J. Anderson, *Humanoid Robots: Abarenbou and DaoDan*, RoboCup 2006: Robot Soccer World Cup X Proceedings, Bremen, Germany (**2006**).

[58] T. Wama, M. Higuchi, H. Sakamoto and R. Nakatsu, *Realization of tai-chi motion using a humanoid robot*, R. Jacquart, ed., IFIP Congress Topical Sessions, pp. 59–64, Kluwer, Toulouse, France (**2004**).

[59] S. Nakaoka, S. Kajita and K. Yokoi, *Intuitive and Flexible User Interface for Creating Whole Body Motions of Biped Humanoid Robots*, 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009), pp. 1675–1682, Taipei, Taiwan (**2009**).

[60] P. A. Andersen, J. P. Garrison and J. Andersen, *Implications of a neurophysiological approach for the study of nonverbal communicaton*, Human Communication Research, vol. 6, pp. 4–89 (**1979**).

[61] A. Bodnar, R. Corbett and D. Nekrasovski, *AROMA: ambient awareness through olfaction in a messaging application*, Proceedings of the 6th International Conference on Multimodal interfaces, ICMI '04, pp. 183–190, State College, PA, USA (**2004**).

[62] C. Lee and Y. Xu, *Online, interactive learning of gestures for human/robot interfaces*, 1996 IEEE International Conference on Robotics and Automation (ICRA 1996), vol. 4, pp. 2982–2987, Minneapolis, MN, USA (**1996**).

[63] K. F. MacDorman, *Androids as an experimental apparatus: Why is there an uncanny valley and can we exploit it?*, Proceedings of the CogSci-2005 Workshop: Toward Social Mechanisms of Android Science, pp. 106–118, Stresa, Italy (**2005**).

[64] S. Weinstein, *Intensive and extensive aspects of tactile sensitivity as a function of body part, sex, and laterality.*, D. R. Kenshalo, ed., The Skin Senses, Proceedings of the First International Symposium on the Skin Senses, chap. 10, pp. 195–218, Charles C. Thomas, Springfield (**1968**).

[65] A. R. Møller, Sensory systems: anatomy and physiology, Academic Press, San Diego, CA, USA (**2002**).

[66] P. B. y Rita, C. C. Collins, F. A. Saunders, B. White and L. Scadden, *Vision Substitution by Tactile Image Projection*, Nature, vol. 221, pp. 943–964 (**1969**).

[67] C. M. Reed, N. I. Durlach and L. A. Delhorne, *Historical overview of tactile aid research*, Proceedings of the Second International Conference on tactile aids, hearing aids and Cochlear Implants, Stockholm, Sweden (**1992**).

[68] M. P. Lynch, R. E. Eilers, D. K. Oller and L. Lavoie, *Speech perception by congenitally deaf subjects using an electrocutaneous vocoder*, Journal of Rehabilitation Research and Development, vol. 25, no. 3, pp. 41–50 (**1988**).

[69] T. Oron-Gilad, J. Downs, R. Gilson and P. Hancock, *Vibrotactile Guidance Cues for Target Acquisition*, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, vol. 37, no. 5, pp. 993–1004 (**2007**).

[70] R. W. Lindeman, J. L. Sibert, C. E. Lathan and J. M. Vice, *The Design and Deployment of a Wearable Vibrotactile Feedback System*, Proceedings of the Eighth International Symposium on Wearable Computers, ISWC '04, pp. 56–59, Arlington, VA, USA (**2004**).

[71] J. B. V. Erp, I. Saturday and C. Jansen, *Application of tactile displays in sports: where to, how and when to move*, EuroHaptics 2006, pp. 105–109, Paris, France (**2006**).

[72] T. Field, Touch, MIT Press, Bradford Books (**2003**).

[73] M. J. Hertenstein, D. Keltner, B. App, B. A. Bulleit and A. R. Jaskolka, *Touch communicates distinct emotions*, Emotion, vol. 6, no. 3, pp. 528–533 (**2006**).

[74] F. Tanaka, A. Cicourel and J. R. Movellan, *Socialization between toddlers and robots at an early childhood education center*, Proceedings of the National Academy of Sciences, vol. 104, no. 46, pp. 17954–17958 (**2007**).

[75] T. Takeda, Y. Hirata and K. Kosuge, *HMM-based Error Recovery of Dance Step Selection for Dance Partner Robot*, 2007 IEEE International Conference on Robotics and Automation (ICRA 2007), pp. 1768–1773, Roma, Italy (**2007**).

[76] S. Gentry and R. Murray-Smith, *Haptic dancing: human performance at haptic decoding with a vocabulary*, IEEE International Conference on Systems Man and Cybernetics, pp. 3432–3437, Washington, DC, USA (**2003**).

[77] B. R. Duffy, *Anthropomorphism and the social robot*, Robotics and Autonomous Systems, vol. 42, no. 3-4, pp. 177–190 (**2003**).

[78] T. Yoshikai, M. Hayashi and M. Inaba, *Simultaneous Learning and Recalling System for Wholebody Motion of a Humanoid with Soft Sensor Flesh*, Intelligent Autonomous Systems 10 - IAS-10, pp. 229–237, Ottawa, Canada (**2008**).

[79] T. Yoshikawa and O. Khatib, *Compliant humanoid robot control by the torque transformer*, 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009), pp. 3011–3018, St. Louis, MO, USA (**2009**).

[80] O. Ly and P.-Y. Oudeyer, *Acroban the humanoid: playful and compliant physical child-robot interaction*, SIGGRAPH '10: ACM SIGGRAPH 2010 Emerging Technologies, pp. 1–1, Los Angeles, CA, USA (**2010**).

[81] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi and T. Isozumi, *Humanoid robot HRP-2*, 2004 IEEE International Conference on Robotics and Automation (ICRA 2004), vol. 2, pp. 1083–1090, New Orleans, LA, USA (**2004**).

[82] M. J. Hertenstein, J. Verkamp, A. Kerestes and R. Holmes, *The Communicative Functions of Touch in Humans, Nonhuman Primates, and Rats: A Review and Synthesis of the Empirical Research*, Genetic, Social, and General Psychology Monographs, vol. 132, no. 1, pp. 5–94 (**2006**).

[83] T. Minato, M. Shimada, S. Itakura, K. Lee and H. Ishiguro, *Evaluating human likeness of an android by comparing gaze behaviors elicited*, Advanced Robotics, vol. 20, pp. 1147–1163 (**2006**).

[84] Y. Kuniyoshi, M. Inaba and H. Inoue, *Learning by watching: extracting reusable task knowledge from visual observation of human performance*, IEEE Transactions on Robotics and Automation, vol. 10, no. 6, pp. 799–822 (**1994**).

[85] C. G. Atkeson, A. W. Moore and S. Schaal, *Locally Weighted Learning*, Artificial Intelligent Review, vol. 11, pp. 11–73 (**1997**).

[86] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer, 1 edn. (**2007**).

[87] F. Scarselli and A. C. Tsoi, *Universal Approximation Using Feedforward Neural Networks: A Survey of Some Existing Methods, and Some New Results*, Neural Networks, vol. 11, no. 1, pp. 15–37 (**1998**).

[88] W.-Y. Loh, *Regression Trees With Unbiased Variable Selection and Interaction Detection*, Statistica Sinica, vol. 12, pp. 361–386 (**2002**).

[89] D. W. Aha, D. Kibler and M. K. Albert, *Instance-Based Learning Algorithms*, Machine Learning, vol. 6, pp. 37–66 (**1991**).

[90] W. S. Cleveland and S. J. Devlin, *Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting*, Journal of the American Statistical Association, vol. 83, no. 403, pp. 596–610 (**1988**).

[91] E. Nadaraya, *On estimating regression*, Theory of Probability and its Applications, vol. 9, pp. 141–142 (**1964**).

[92] G. S. Watson, *Smooth regression analysis*, Sankhyā: The Indian Journal of Statistics, Series A, vol. 26, pp. 359–372 (**1964**).

[93] H. Luinge and P. Veltink, *Measuring orientation of human body segments using miniature gyroscopes and accelerometers*, Medical and Biological Engineering and Computing, vol. 43, pp. 273–282 (**2005**).

[94] Y. Ohmura, Y. Kuniyoshi and A. Nagakubo, *Conformable and Scalable Tactile Sensor Skin for Curved Surfaces*, 2006 IEEE International Conference on Robotics and Automation (ICRA 2006), pp. 1348–1353, Orlando, FL, USA (**2006**).

[95] H. Alirezaei, A. Nagakubo and Y. Kuniyoshi, *A highly stretchable tactile distribution sensor for smooth surfaced humanoids*, IEEE-RAS 7th International Conference on Humanoid Robots (Humanoids 2007), pp. 167–173, Pittsburg, PA, USA (**2007**).

[96] S. Katsura, K. Ohnishi and K. Ohishi, *Transmission of Force Sensation by Environment Quarrier Based on Multilateral Control*, IEEE Transactions on Industrial Electronics, vol. 54, no. 2, pp. 898–906 (**2007**).

[97] M. Vukobratovic, B. Borovac and V. Potkonjak, *ZMP: A review of some basic misunderstandings*, International Journal of Humanoid Robotics, vol. 3, no. 2, pp. 153–175 (**2006**).

[98] R. Azuma, *A Survey of Augmented Reality*, Presence, vol. 6, no. 4, pp. 355–385 (**1997**).

[99] A. Hoerl and R. Kennard, *Ridge regression: Biased estimation for nonorthogonal problems*, Technometrics, vol. 12, pp. 55–67 (**1970**).

[100] J. R. Quinlan, C4.5: programs for machine learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (**1993**).

[101] K. Yamane and Y. Nakamura, *Natural Motion Animation through Constraining and Deconstraining at Will*, IEEE Transactions on visualization and computer graphics, vol. 9, pp. 352–360 (**2003**).

[102] T. Noda, T. Miyashita, H. Ishiguro and N. Hagita, *Map acquisition and classification of haptic interaction using cross correlation between distributed tactile sensors on the whole body surface*, 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007), pp. 1099–1105, San Diego, CA, USA (**2007**).

[103] I. Borg and P. Groenen, Modern Multidimensional Scaling: Theory and Applications, Springer (**2005**).

[104] M. Tanaka, Y. Tanaka and S. Chonan, *Measurement and Evaluation of Tactile Sensations using a PVDF Sensor*, Journal of Intelligent Material Systems and Structures, vol. 19, pp. 35–42 (**2008**).

[105] G. Cannata, M. Maggiali, G. Metta and G. Sandini, *An embedded artificial skin for humanoid robots*, Proceedings of IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI2008), pp. 434–438, Seoul, Korea (**2008**).

[106] M. Asada and M. Mayer, *RoboCup Humanoid Challenge*, International Journal of Humanoid Robots, vol. 5, pp. 335–351 (**2008**).

[107] Y. Ansel, F. Schmitz, S. Kunz, H. P. Gruber and G. Popovic, *Development of tools for handling and assembling microcomponents*, Journal of Micromechanics and Microengineering, vol. 12, no. 4, p. 430 (**2002**).

[108] A. Strehl and J. Ghosh, *Cluster ensembles — a knowledge reuse framework for combining multiple partitions*, The Journal of Machine Learning Research, vol. 3, pp. 583–617 (**2003**).

[109] R. Agrawal, T. Imieliński and A. Swami, *Mining association rules between sets of items in large databases*, SIGMOD Record, vol. 22, pp. 207–216 (**1993**).

[110] M. Lewis, R. Etienne-Cummings, A. Cohen and M. Hartmann, *Toward biomorphic control using custom aVLSI CPG chips*, 2000 IEEE International Conference on Robotics and Automation (ICRA 2000), vol. 1, pp. 494–500, San Francisco, CA, USA (**2000**).

[111] K. Nakada, T. Asai and Y. Amemiya, *An analog neural oscillator circuit for locomotion controller in quadruped walking robot*, 2003 IEEE International Joint Conference on Neural Networks, vol. 2, pp. 983–988, Portland, OR, USA (**2003**).

[112] H. Inada and K. Ishii, *Behavior generation of bipedal robot using central pattern generator(CPG) (1st report: CPG parameters searching method by genetic algorithm)*, 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), vol. 3, pp. 2179–2184, Las Vegas, NV, USA (**2003**).

[113] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi and G. Cheng, *Learning CPG-based Biped Locomotion with a Policy Gradient Method: Application to a Humanoid Robot*, The International Journal of Robotics Research, vol. 27, no. 2, pp. 213–228 (**2008**).

[114] K. Matsuoka, *Sustained oscillations generated by mutually inhibiting neurons with adaptation*, Biological Cybernetics, vol. 52, pp. 367–376 (**1985**).

[115] K. Uematsu, *Central nervous system underlying fish swimming [A review]*, N. Kato and S. Kamimura, eds., Bio-mechanisms of Swimming and Flying, pp. 103–116, Springer Japan (**2008**).

[116] T. Forrest, *Temporal rhythms in the signals of insects*, Proceedings of the 16th International Congress on Acoustics and 135th meeting of the Acoustical Society of America, pp. 1803–1804, Seattle, WA, USA (**1998**).

[117] J. Lund and A. Kolta, *Generation of the Central Masticatory Pattern and Its Modification by Sensory Feedback*, Dysphagia, vol. 21, pp. 167–174 (**2006**).

[118] J. T. Choi and A. J. Bastian, *Adaptation reveals independent control networks for human walking*, Nature Neuroscience, vol. 10, pp. 1055–1062 (**2007**).

[119] M. A. Arbib, The Handbook of Brain Theory and Neural Networks, MIT Press, Cambridge, MA, USA, 1st edn. (**1995**).

[120] L. M. Jordan, *Initiation of Locomotion in Mammals*, Annals of the New York Academy of Sciences, vol. 860, pp. 83–93 (**1998**).

[121] M. A. Lewis, A. H. Fagg and G. Bekey, *Genetic Algorithms for Gait Synthesis in a Hexapod Robot*, Recent Trends in Mobile Robots, pp. 317–331 (**1994**).

[122] T. Ishii, S. Masakado and K. Ishii, *Locomotion of a quadruped robot using CPG*, 2004 IEEE International Joint Conference on Neural Networks, vol. 4, pp. 3179–3184, Budapest, Hungary (**2004**).

[123] J. J. Collins and S. A. Richmon, *Hard-wired central pattern generators for quadrupedal locomotion*, Biological Cybernetics, vol. 71, no. 5, pp. 375–385 (**1994**).

[124] J. Morimoto, G. Endo, J. Nakanishi, S.-H. Hyon, G. Cheng, D. C. Bentivegna and C. G. Atkeson, *Modulation of Simple Sinusoidal Patterns by a Coupled Oscillator Model for Biped Walking*, 2006 IEEE International Conference on Robotics and Automation (ICRA 2006), pp. 1579–1584, Orlando, FL, USA (**2006**).

[125] L. Righetti and A. Ijspeert, *Programmable Central Pattern Generators: an application to biped locomotion control*, 2006 IEEE International Conference on Robotics and Automation (ICRA 2006), pp. 1585–1590, Orlando, FL, USA (**2006**).

[126] A. C. de Pina Filho, M. S. Dutra and L. S. C. Raptopoulos, *Modeling of a bipedal robot using mutually coupled Rayleigh oscillators*, Biological Cybernetics, vol. 92, no. 1, pp. 1–7 (**2005**).

[127] A. Crespi, A. Badertscher, A. Guignard and A. Ijspeert, *AmphiBot I: An amphibious snake-like robot*, Robotics and Autonomous Systems, vol. 50, no. 4, pp. 163–175 (**2005**).

[128] A. J. Ijspeert and A. Crespi, *Online trajectory generation in an amphibious snake robot using a lamprey-like central pattern generator model*, 2007 IEEE International Conference on Robotics and Automation (ICRA 2007), pp. 262–268, Roma, Italy (**2007**).

[129] J. Conradt and P. Varshavskaya, *Distributed Central Pattern Generator Control for a Serpentine Robot*, Joint 13th International Conference on Artificial Neural Networks and 10th International Conference on Neural Information Processing (ICANN 03), Istanbul, Turkey (**2003**).

[130] A. Kern and R. Stoop, *Nonlinear Dynamics of Cochlear Information Processing*, Proceedings of the 12th International Workshop on Nonlinear Dynamics of Electronic Systems (NDES-04), pp. 190–193, Evora, Portugal (**2004**).

[131] J. Nicolas, *Artificial evolution of controllers based on non-linear oscillators for bipedal locomotion*, Master's thesis, EPFL, Lausanne, Switzerland (**2005**).

[132] J. Buchli and A. Ijspeert, *A simple, adaptive locomotion toy-system*, 8th International Conference on Simulation of Adaptive Behaviour (SAB 04), pp. 153–162, Los Angeles, CA, USA (**2004**).

[133] P. Veskos and Y. Demiris, *Developmental acquisition of entrainment skills in robot swinging using van der Pol oscillators*, Proceedings of the Epigenetic Robotics 2005, pp. 87–93, Nara, Japan (**2005**).

[134] D. Kurabayashi, K. Okita, T. Funato and R. Sakaematsu, *Obstacle avoidance of a mobile robot group using a nonlinear oscillator network*, 2006 IEEE International Conference on Robotics and Automation (ICRA 2006), pp. 186–191, Orlando, FL, USA (**2006**).

[135] A. L. Hodgkin and A. F. Huxley, *A quantitative description of membrane current and its application to conduction and excitation in nerve*, Journal of Physiology, vol. 117, pp. 500–544 (**1952**).

[136] J. Nagumo, S. Arimoto and S. Yoshizawa, *An Active Pulse Transmission Line Simulating Nerve Axon*, Proceedings of the Institute of Radio Engineers, vol. 50, no. 10, pp. 2061–2070 (**1962**).

[137] B. Mathayomchan and R. D. Beer, *Center-crossing recurrent neural networks for the evolution of rhythmic behavior*, Neural Computation, vol. 14, pp. 2043–2051 (**2002**).

[138] H. J. Chiel, R. D. Beer and J. C. Gallagher, *Evolution and analysis of model CPGs for walking: I. Dynamical modules.*, Journal of Computational Neuroscience, vol. 7, pp. 99–118 (**1999**).

[139] A. Billard and M. J. Mataric, *Learning human arm movements by imitation:: Evaluation of a biologically inspired connectionist architecture*, Robotics and Autonomous Systems, vol. 37, no. 2-3, pp. 145–160 (**2001**).

[140] E. D. Vaughan, E. D. Paolo and I. R. Harvey, *The Evolution of Control and Adaptation in a 3D Powered Passive Dynamic Walker*, 9th International Conference on the Simulation and Syntesis of Living Systems (ALIFE9), pp. 139–145, Boston, MA, USA (**2004**).

[141] A. L. Taylor, G. W. Cottrell and W. B. Kristan, *Analysis of Oscillations in a Reciprocally Inhibitory Network with Synaptic Depression*, Neural Computation, vol. 14, pp. 561–581 (**2002**).

[142] Ö. Ekeberg, *A combined neuronal and mechanical model of fish swimming*, Biological Cybernetics, vol. 69, pp. 363–374 (**1993**).

[143] A. J. Ijspeert and J. Kodjabachian, *Evolution and development of a central pattern generator for the swimming of a Lamprey*, Artificial Life, vol. 5, no. 3, pp. 247–269 (**1999**).

[144] M. Williamson, *Rhythmic robot arm control using oscillators*, 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 1998), vol. 1, pp. 77–83, Victoria, Canada (**1998**).

[145] Q. Han, Z. Qin, X. Yang and B. Wen, *Rhythmic Swing Motions of a Two-link Robot with a Neural Controller*, International Journal of Innovatice Computing, Information and Control (IJICIC), vol. 3, no. 2, pp. 335–342 (**2007**).

[146] M. Lungarella and L. Berthouze, *Learning to bounce: First lessons from a bouncing robot*, Proceedings of the 2nd International Symposium on Adaptive Motion in Animals and Machines (AMAM 2003), Kyoto, Japan (**2003**).

[147] A. H. Cohen, *Control principles for locomotion - looking toward biology*, Proceedings of the 2nd International Symposium on Adaptive Motion in Animals and Machines (AMAM 2003), pp. 41–51, Kyoto, Japan (**2003**).

[148] K. Inoue, S. Ma and C. Jin, *Neural Oscillator Network-based Controller for Meandering Locomotion of Snake-like Robots*, 2004 IEEE International Conference on Robotics and Automation (ICRA 2004), pp. 5064–5069, New Orleans, LA, USA (**2004**).

[149] T. Williams, *Phase coupling by synaptic spread in chains of coupled neuronal oscillators*, Science, vol. 258, no. 5082, pp. 662–665 (**1992**).

[150] S. Schaal, S. Kotosaka and D. Sternad, *Nonlinear dynamical systems as movement primitives*, IEEE-RAS 1st International Conference on Humanoid Robots (Humanoids 2000), pp. 117–124, Cambridge, MA, USA (**2000**).

[151] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal and M. Kawato, *A framework for learning biped locomotion with dynamical movement primitives*, IEEE-RAS 4th International Conference on Humanoid Robots (Humanoids 2004), pp. 925–940, Santa Monica, Los Angeles, CA, USA (**2004**).

[152] S. Aoi, K. Tsuchiya and K. Tsujita, *Turning Control of a Biped Locomotion Robot using Nonlinear Oscillators*, 2004 IEEE International Conference on Robotics and Automation (ICRA 2004), pp. 3043–3048, New Orleans, LA, USA (**2004**).

[153] M. A. Lewis, F. Tenore and R. Etienne-Cummings, *CPG Design using Inhibitory Networks*, 2005 IEEE International Conference on Robotics and Automation (ICRA 2005), pp. 3682–3687, Barcelona, Spain (**2005**).

[154] S. Degallier, C. P. Santos, L. Righetti and A. Ijspeert, *Movement generation using dynamical systems: a humanoid robot performing a drumming task*, IEEE-RAS 6th International Conference on Humanoid Robots (Humanoids 2006), Genoa, Italy (**2006**).

[155] L. Jalics, H. Hemami and Y. F. Zheng, *Pattern Generation Using Coupled Oscillators for Robotic and Biorobotic Adaptive Periodic Movement*, 1997 IEEE International Conference on Robotics and Automation (ICRA 1997), vol. 1, pp. 179–184, Albuquerque, NM, USA (**1997**).

[156] P. Roy and Y. Demiris, *Analysis of biped gait patterns generated by van der Pol and Rayleigh oscillators under feedback*, Proceedings of the 3rd International Symposium on Adaptive Motion in Animals and Machines (AMAM 2005), Ilmenau, Germany (**2005**).

[157] M. Yano, *Purposive Locomotion of Insect in Indefinite Environment*, Proceedings of the 2nd International Symposium on Adaptive Motion in Animals and Machines (AMAM 2003), pp. 31–40, Kyoto, Japan (**2003**).

[158] H. Kimura and Y. Fukuoka, *Biologically Inspired Dynamic Walking of a Quadruped Robot on Irregular Terrain - Adaptation at Spinal Cord and Brain Stem -*, International Symposium on Adaptive Motion in Animals and Machines (AMAM 2000), Montreal, Canada (**2000**).

[159] H. Kimura, Y. Fukuoka, Y. Hada and K. Takase, *Three-Dimensional Adaptive Dynamic Walking of a Quadruped - Rolling Motion Feedback to CPGs Controlling Pitching Motion*, 2002 IEEE International Conference on Robotics and Automation (ICRA 2002), pp. 2228–2233, Washington, DC, USA (**2002**).

[160] K. Nakada, T. Asai and Y. Amemiya, *An Analog CMOS Circuit Implementing CPG Controller for Quadruped Walking Robot*, Proceedings of the 2nd International Symposium on Adaptive Motion in Animals and Machines (AMAM 2003), pp. WeP–II–2, Kyoto, Japan (**2003**).

[161] H. Takemura, J. Ueda, Y. Matsumoto and T. Ogasawara, *The Influence of Friction on Gait and Energy Efficiency of the Walking Robot Based on Rhythmic Control*, Proceedings of the 2nd International Symposium on Adaptive Motion in Animals and Machines (AMAM 2003), p. ThA–I–4, Kyoto, Japan (**2003**).

[162] J. S. Bay and H. Hemami, *Modeling of a neural pattern generator with coupled nonlinear oscillators*, IEEE Transactions on Biomedical Engineering, vol. 34, no. 4, pp. 297–306 (**1987**).

[163] M. Cao and A. Kawamura, *A Design Method of Neural Oscillatory Networks for Generation of Humanoid Biped Walking Patterns*, 1998 IEEE International Conference on Robotics and Automation (ICRA 1998), pp. 2357–2362, Leuven, Belgium (**1998**).

[164] K. Tsujita, K. Tsuchiya and A. Onat, *Decentralized Autonomous Control of a Quadruped Locomotion Robot*, Artificial life and robotics, vol. 5, pp. 152–158 (**2003**).

[165] S. Miyakoshi, G. Taga, Y. Kuniyoshi and A. Nagakubo, *Three dimensional bipedal stepping motion using neural oscillators–towards humanoid motion in the real world.*, 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 1998), pp. 84–89, Victoria, Canada (**1998**).

[166] W. Ilg, J. C. Albiez, H. Jedele, K. Berns and R. Dillmann, *Adaptive Periodic Movement Control for the Four Legged Walking Machine BISAM*, 1999 IEEE International Conference on Robotics and Automation (ICRA 1999), pp. 2354–2359, Detroit, MI, USA (**1999**).

[167] K. J. Pikovsky A., Rosenblum M., Synchronization: A Universal Concept in Nonlinear Science, Cambridge University Press, Cambridge (**2001**).

[168] K. Harada, M. Morisawa, K. Miura, S. Nakaoka, K. Fujiwara, K. Kaneko and S. Kajita, *Kinodynamic Gait Planning for Full-Body Humanoid Robots*, 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2008), pp. 1544–1550, Nice, France (**2008**).

[169] F. Yamasaki, K. Endo, H. Kitano and M. Asada, *Acquisition of Humanoid Walking Motion using Genetic Algorithm - Considering Characteristics of Servo Modules*, 2002 IEEE International Conference on Robotics and Automation (ICRA 2002), pp. 3123–3128, Washington, DC, USA (**2002**).

[170] J. Peters, S. Vijayakumar and S. Schaal, *Reinforcement learning for humanoid robotics*, IEEE-RAS 3rd International Conference on Humanoid Robots (Humanoids 2003), Karlsruhe, Germany (**2003**).

[171] K. D. Mombaur, H. G. Bock, J. P. Schlöder and R. W. Longman, *Stable Walking and Running Robots Without Feedback*, Proceedings of the 7th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines 2004, CLAWAR 2004, pp. 725–735, Madrid, Spain (**2005**).

[172] Y. Okumura, T. Tawara, K. Endo, T. Furuta and M. Shimizu, *Realtime ZMP compensation for biped walking robot using adaptive inertia force control*, 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), vol. 1, pp. 335–339, Las Vegas, NV, USA (**2003**).

[173] S. Kajita, K. Kaneko, M. Morisawa, S. Nakaoka and H. Hirukawa, *ZMP-based Biped Running Enhanced by Toe Springs*, 2007 IEEE International Conference on Robotics and Automation (ICRA 2007), pp. 3936–3969, Roma, Italy (**2007**).

[174] F. Kanehiro, T. Yoshimi, S. Kajita, M. Morisawa, K. Fujiwara, K. Harada, K. Kaneko, H. Hirukawa and F. Tomita, *Whole Body Locomotion Planning of Humanoid Robots based on a 3D Grid Map*, 2005 IEEE International Conference on Robotics and Automation (ICRA 2005), pp. 1072–1078, Barcelona, Spain (**2005**).

[175] M. Vukobratovic and J. Stepanenko, *On the stability of anthropomorphic systems*, Mathematical Biosciences, vol. 15, pp. 1–37 (**1972**).

[176] V. Prahlad, G. Dip and C. Meng-hwee, *Disturbance rejection by online zmp compensation*, Robotica, vol. 26, pp. 9–17 (**2008**).

[177] T. Sugihara, Y. Nakamura and H. Inoue, *Realtime Humanoid Motion Generation through ZMP Manipulation based on Inverted Pendulum Control*, 2002 IEEE International Conference on Robotics and Automation (ICRA 2002), pp. 1404–1409, Washington, DC, USA (**2002**).

[178] K. Yokoi, F. Kanehiro, K. Kaneko, S. Kajita, K. Fujiwara and H. Hirukawa, *Experimental Study of Humanoid Robot HRP-1S*, The International Journal of Robotics Research, vol. 23, pp. 351–362 (**2004**).

[179] A. J. Ijspeert, J. Nakanishi and S. Schaal, *Learning Attractor Landscapes for Learning Motor Primitives*, S. Becker, S. Thrun and K. Obermayer, eds., Advances in Neural Information Processing Systems (NIPS), pp. 1523–1530, MIT Press (**2002**).

[180] A. Ude, M. Riley, B. Nemec, A. Kos, T. Asfour and G. Cheng, *Synthesizing goal-directed actions from a library of example movements*, IEEE-RAS 7th International Conference on Humanoid Robots (Humanoids 2007), pp. 115–121, Pittsburg, PA, USA (**2007**).

[181] J. Baltes, J. Anderson and S. McGrath, *Model-free active balancing for humanoid robots*, RoboCup 2008: Robot Soccer World Cup XII Proceedings, pp. 544–555, Suzhou, China (**2008**).

[182] K. Hauser, T. Bretl and J.-C. Latombe, *Non-gaited humanoid locomotion planning*, IEEE-RAS 5th International Conference on Humanoid Robots (Humanoids 2005), pp. 7–12, Tsukuba, Japan (**2005**).

[183] T. Yoshikai, M. Hayashi, Y. Ishizaka, T. Sagisaka and M. Inaba, *Behavior Integration for Whole-body Close Interactions by a Humanoid with Soft Sensor Flesh*, IEEE-RAS 7th International Conference on Humanoid Robots (Humanoids 2007), pp. 109–114, Pittsburg, PA, USA (**2007**).

[184] T. Ogura, K. Okada, M. Inaba and H. Inoue, *Behavior network acquisition in multisensor space for whole-body humanoid*, Proceedings of IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI2003), pp. 317–322, Tokyo, Japan (**2003**).

[185] L. Kovar, M. Gleicher and F. H. Pighin, *Motion Graphs*, ACM Transactions on Graphics (SIGGRAPH2002), vol. 21, no. 3, pp. 473–482 (**2002**).

[186] H. Sidenbladh, M. J. Black and L. Sigal, *Implicit Probabilistic Models of Human Motion for Synthesis and Tracking*, ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I, pp. 784–800, Springer-Verlag, London, UK (**2002**).

[187] P. Crook and G. Hayes, *Learning in a state of confusion: Perceptual aliasing in grid world navigation*, Proceedings of Towards Intelligent Mobile Robots (TIMR2003), UWE, Bristol (**2003**).

[188] J. Beis and D. G. Lowe, *Shape indexing using approximate nearest-neighbour search in high-dimensional spaces*, Conference on Computer Vision and Pattern Recognition, pp. 1000–1006, San Juan, Puerto Rico (**1997**).

[189] J. Adler, *The sensing of chemicals by bacteria*, Scientific American, vol. 234, pp. 40–47 (**1976**).

[190] S. G. Nurzaman, Y. Matsumoto, Y. Nakamura, S. Koizumi and H. Ishiguro, *Yuragi-Based Adaptive Searching Behavior in Mobile Robot: From Bacterial Chemotaxis to Levy Walk*, 2008 IEEE International Conference on Robotics and Biomimetics (ROBIO 2008), pp. 806–811, Bangkok, Thailand (**2008**).

[191] J. E. Segall, S. M. Block and H. C. Berg, *Temporal comparisons in bacterial chemotaxis*, Proceedings of the National Academy of Sciences of the United States of America, vol. 83, no. 23, pp. 8987–8991 (**1986**).

[192] C. V. Rao, J. R. Kirby and A. P. Arkin, *Design and Diversity in Bacterial Chemotaxis: A Comparative Study in Escherichia coli and Bacillus subtilis*, PLoS Biology, vol. 2, no. 2, p. e49 (**2004**).

[193] S. Zigmond, *Mechanisms of sensing chemical gradients by polymorphonuclear leukocytes*, Nature, pp. 450–452 (**1974**).

[194] P. Devreotes and S. Zigmond, *Chemotaxis in eukaryotic cells: a focus on leukocytes and Dictyostelium*, Annual review of cell biology, vol. 4, no. 1, pp. 649–686 (**1988**).

[195] C. Parent and P. Devreotes, *A cell's sense of direction*, Science, vol. 284, no. 5415, p. 765 (**1999**).

[196] C. Parent, *Making all the right moves: chemotaxis in neutrophils and Dictyostelium*, Current opinion in cell biology, vol. 16, no. 1, pp. 4–13 (**2004**).

[197] P. J. M. Van Haastert and P. N. Devreotes, *Chemotaxis: signalling the way forward*, Nature Reviews Molecular Cell Biology, vol. 5, no. 8, pp. 626–634 (**2004**).

[198] P. R. Patnaik, *A feasibility analysis of bacterial chemotaxis under the influence of external noise*, Journal of Biochemical Technology, vol. 2, pp. 119–125 (**2009**).

[199] M. J. Schnitzer, *Theory of continuum random walks and application to chemotaxis*, Physical Review E, vol. 48, no. 4, pp. 2553–2568 (**1993**).

[200] M. D. Baker, P. M. Wolanin and J. B. Stock, *Systems biology of bacterial chemotaxis*, Current Opinion in Microbiology, vol. 9, no. 2, pp. 187–192 (**2006**).

[201] L. Jiang, Q. Ouyang and Y. Tu, *Quantitative Modeling of Escherichia coli Chemotactic Motion in Environments Varying in Space and Time*, PLoS Computational Biology, vol. 6, no. 4, p. e1000735 (**2010**).

[202] B. W. Andrews, T.-M. Yi and P. A. Iglesias, *Optimal Noise Filtering in the Chemotactic Response of Escherichia coli*, PLoS Computational Biology, vol. 2, no. 11, p. e154 (**2006**).

[203] Y. M. Saiki, E. Takeuchi and T. Tsubouchi, *Vehicle localization in outdoor woodland environments with sensor fault detection*, 2008 IEEE International Conference on Robotics and Automation (ICRA 2008), pp. 449–454, Pasadena, CA, USA (**2008**).

[204] C. Plagemann, D. Fox and W. Burgard, *Efficient Failure Detection on Mobile Robots Using Particle Filters with Gaussian Process Proposals*, Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, pp. 2185–2190, Hyderabad, India (**2007**).

[205] J. Bongard, V. Zykov and H. Lipson, *Resilient Machines Through Continuous Self-Modeling*, Science, vol. 314, no. 5802, pp. 1118–1121 (**2006**).

[206] O. Holland and C. Melhuish, *Some adatpive movements of animats with single symmetrical sensors*, From Animals to Animats: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, pp. 53–64 (**1996**).

[207] A. Dhariwal, G. S. Sukhatme and A. A. G. Requicha, *Bacterium-inspired Robots for Environmental Monitoring*, 2004 IEEE International Conference on Robotics and Automation (ICRA 2004), pp. 1436–1443, New Orleans, LA, USA (**2004**).

[208] M. Scheutz and J. Kramer, *Reflection and Reasoning Mechanisms for Failure Detection and Recovery in a Distributed Robotic Architecture for Complex Robots*, 2007 IEEE International Conference on Robotics and Automation (ICRA 2007), pp. 3699–3704, Roma, Italy (**2007**).

[209] R. Benzi, A. Sutera and A. Vulpiani, *The mechanism of stochastic resonance*, Journal of Physics A: mathematical and general, vol. 14, pp. 453–457 (**1981**).

[210] L. Gammaitoni, P. Hänggi, P. Jung and F. Marchesoni, *Stochastic resonance*, Reviews of Modern Physics, vol. 70, no. 1, pp. 223–287 (**1998**).

[211] K. Wiesenfeld and F. Moss, *Stochastic resonance and the benefits of noise: from ice ages to crayfish and SQUIDs*, Nature, vol. 373, no. 6509, pp. 33–36 (**1995**).

[212] F. DallaLibera, T. Minato, I. Fasel, H. Ishiguro, E. Menegatti and E. Pagello, *Teaching by touching: an intuitive method for development of humanoid robot motions*, IEEE-RAS 7th International Conference on Humanoid Robots (Humanoids 2007), pp. 352–359, Pittsburg, PA, USA (**2007**).

[213] F. DallaLibera, T. Minato, H. Ishiguro, E. Pagello and E. Menegatti, *Developing Robot Motions by Simulated Touch Sensors*, Proceedings of the 1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots, pp. 242–253, Venice, Italy (**2008**).

[214] F. DallaLibera, T. Minato, I. Fasel, H. Ishiguro, E. Pagello and E. Menegatti, *A new paradigm of humanoid robot motion programming based on touch interpretation*, Robotics and Autonomous Systems, vol. 57, no. 8, pp. 846–859 (**2008**).

[215] T. Minato, F. DallaLibera, S. Yokokawa, Y. Nakamura, H. Ishiguro and E. Menegatti, *A Baby Robot Platform for Cognitive Developmental Robotics*, Workshop on "Synergistic Intelligence" at the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009), St. Louis, MO, USA (**2009**).

[216] F. DallaLibera, T. Minato, H. Ishiguro and E. Menegatti, *Developing Central Pattern Generator based periodic motions using tactile interaction*, IEEE-RAS 9th International Conference on Humanoid Robots (Humanoids 2009), pp. 105–112, Paris, France (**2009**).

[217] F. DallaLibera, T. Minato, H. Ishiguro and E. Menegatti, *Direct programming of a central pattern generator for periodic motions by touching*, Robotics and Autonomous Systems, vol. 58, pp. 847–854 (**2010**).

[218] F. DallaLibera, T. Minato, H. Ishiguro, A. Ferreira, E. Pagello and E. Menegatti, *Humanoid motion representation by sensory state transitions*, 2009 6th Latin American Robotics Symposium (LARS2009), pp. 1–8, Valparaiso, Chile (**2009**).

[219] F. DallaLibera, S. Ikemoto, T. Minato, H. Ishiguro, E. Menegatti and E. Pagello, *Biologically inspired mobile robot control robust to hardware failures and sensor noise*, RoboCup 2010: Robot Soccer World Cup XIV Proceedings, Singapore, Singapore (**2010**).

[220] F. DallaLibera, S. Ikemoto, T. Minato, H. Ishiguro, E. Menegatti and E. Pagello, *A Parameterless Biologically Inspired Control Algorithm Robust to Nonlinearities, Dead-Times and Low-Pass Filtering Effects*, Proceedings of the 2nd International Conference on Simulation, Modeling, and Programming for Autonomous Robots, p. 362–373, Darmstadt, Germany (**2010**).

[221] F. DallaLibera, T. Minato, H. Ishiguro, E. Pagello and E. Menegatti, *A software toolset for quick humanoid motion prototyping*, Proceedings of the 4th Workshop on Humanoid Soccer Robots, workshop of the 2009 IEEE-RAS International Conference on Humanoid Robots (Humanoids 2009), pp. 45–51, Paris, France (**2009**).

[222] M. Beal, H. Attias and N. Jojic, *Audio-Video Sensor Fusion with Probabilistic Graphical Models*, A. Heyden, G. Sparr, M. Nielsen and P. Johansen, eds., Computer Vision — ECCV 2002, *Lecture Notes in Computer Science*, vol. 2350, pp. 736–750, Springer Berlin / Heidelberg (**2002**).

[223] B. P. Gerkey, R. T. Vaughan and A. Howard, *The Player/Stage Project:Tools for Multi-Robot and Distributed Sensor Systems*, Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003), pp. 317–323, Coimbra, Portugal (**2003**).

[224] N. Koenig and A. Howard, *Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator*, 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004), pp. 2149–2154, Sendai, Japan (**2004**).

[225] T. Laue and T. Röfer, *SimRobot: Development and Applications*, Workshop Proceedings of SIMPAR 2008 International Conference on Simulation, Modeling and Programming for Autonomous Robots, pp. 143–150, Venice, Italy (**2008**).

[226] O. Michel, *Webots: Symbiosis Between Virtual and Real Mobile Robots*, VW '98: Proceedings of the First International Conference on Virtual Worlds, pp. 254–263, Paris, France (**1998**).

[227] K. Wolff and P. Nordin, *Learning Biped Locomotion from First Principles on a Simulated Humanoid Robot using Linear Genetic Programming.*, Proceedings of the 2003 International Conference on Genetic and evolutionary, GECCO'03, pp. 12–16, Chicago, IL, USA (**2003**).

[228] J. L. Lima, J. C. Gonçalves, P. J. Costa and A. P. Moreira, *Realistic Humanoid Robot Simulation with an Optimized Controller: A Power Consumption Minimization Approach*, Proceedings of the 11th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines 2008, CLAWAR 2008, pp. 1242–1248, Coimbra, Portugal (**2008**).

[229] N. Sugiura and M. Takahashi, *Development of a Humanoid Robot Simulator and Walking Motion Analysis*, Workshop Proceedings of SIMPAR 2008 International Conference on Simulation, Modeling and Programming for Autonomous Robots, pp. 151–158, Venice, Italy (**2008**).

[230] A. Boeing and T. Bräunl, *Evaluation of real-time physics simulation systems*, Proceedings of the 5th International Conference on Computer graphics and interactive techniques in Australia and Southeast Asia, GRAPHITE '07, pp. 281–288, Perth, Australia (**2007**).

[231] S. Wouters and C. Wartmann, The Official Blender 2.0 Guide, Premier Press, Incorporated (**2001**).

[232] O. Obst and M. Rollmann, *Spark - A Generic Simulator for Physical Multi-Agent Simulations*, Computer Systems Science and Engineering, vol. 20, no. 5, pp. 347–356 (**2004**).

[233] L. Hugues and N. Bredeche, *Simbad : an Autonomous Robot Simulation Package for Education and Research*, Simulation of Adaptive Behavior (SAB 2006), pp. 831–842, Rome, Italy (**2006**).

[234] M. Friedmann, K. Petersen and O. von Stryk, *Simulation of Multi-Robot Teams with Flexible Level of Detail*, S. Carpin, I. Noda, E. Pagello, M. Reggiani and O. von Stryk, eds., Proceedings of the 1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots, vol. 5325, pp. 29–40, Venice, Italy (**2008**).

[235] D. Burns and R. Osfield, *Open Scene Graph A: Introduction, B: Examples and Applications*, IEEE Virtual Reality Conference 2004 (VR 2004), p. 265, Chicago, IL, USA (**2004**).

[236] Y. Sasamoto, A. Ikejima, M. Morizono, K. Sakai, P. Sung, M. Ogino and M. Asada, *JEAP Team Description*, RoboCup 2010: Robot Soccer World Cup XIV Proceedings, Singapore, Singapore (**2010**).

[237] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley Professional (**1995**).

[238] I. Fukuyori, Y. M. Yutaka Nakamura and H. Ishiguro, *Flexible Control Mechanism for Multi-DOF Robotic Arm Based on Biological Fluctuation*, 10th International Conference on Simulation of Adaptive Behaviour (SAB 08), pp. 22–31, Osaka, Japan (**2008**).

[239] T. Yanagida, M. Ueda, T. Murata, S. Esaki and Y. Ishii, *Brownian motion, fluctuation and life.*, Biosystems, vol. 88, pp. 228–242 (**2006**).

[240] K. Leibnitz, N. Wakamiya and M. Murata, *Biologically inspired self-adaptive multipath routing in overlay networks*, Communications of the ACM, vol. 49, pp. 63–67 (**2006**).

[241] C. Furusawa, K. Kaneko and H. Shimizu, *A noise-driven mechanism for adaptive growth rate regulation*, BIONETICS '08: Proceedings of the 3rd International Conference on Bio-Inspired Models of Network, Information and Computing Sytems, pp. 1–5, Hyogo, Japan (**2008**).