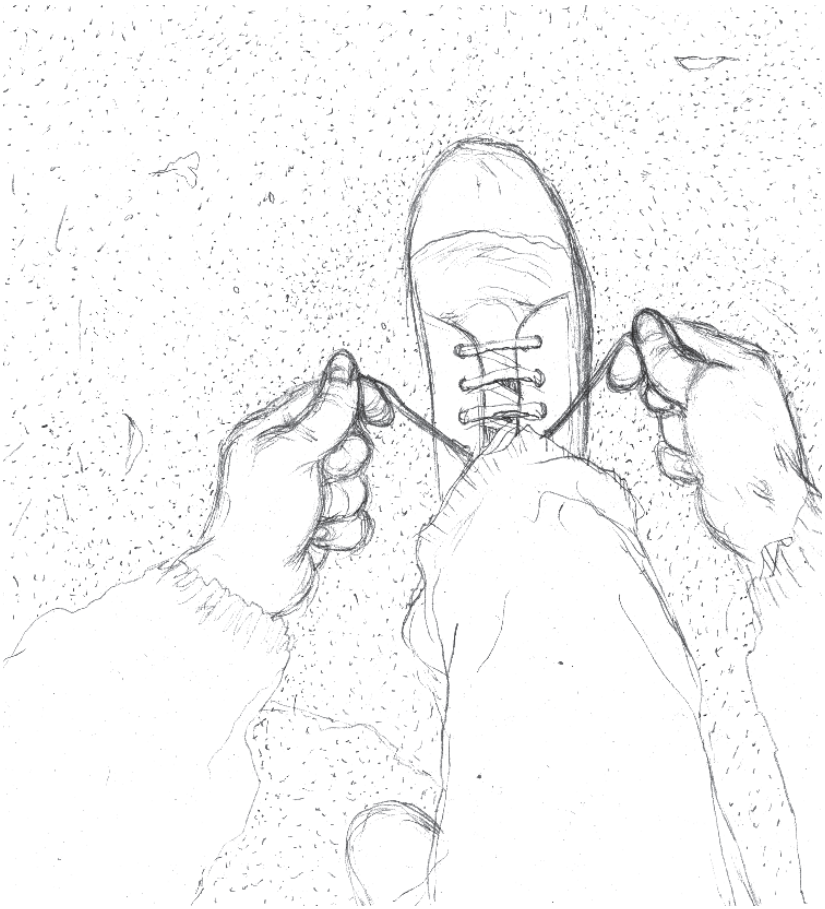
complexityinmotion

a dissertation on human machine interaction



ignazio aleo

complexity in motion
a dissertation on human machine interaction
ignazio aleo

university of catania
XXIV doctoral school

coordinator: prof. luigi fortuna

tutor: prof paolo arena
ing. luca patanè

cover from a drawing of sebastiano aleo 2011

none of us is as smart as all of us
(japanese proverb)

table of contents

introduction

exploiting motion and control of living beings:
from data analysis to feedback control

algorithms for motor and motion control:
from reflexes to complex coordination

perceiving the world:
action-oriented perception

algorithmic solutions for actions and more:
sequence learning toward given objectives

a general framework for robot control and system integration:
from structure to complex algorithms

advanced motion platform for inertial sensing:
beyond the pure motion control research

conclusion

acknowledgment

Academic path is a very hard path. As it is possible to imagine what follows in these pages is fruit of my studies and of my work but it is important to highlight that, more than this, it is fruit of several form of support I have recieved in this years.
Fortunately, this work has been deeply supported. First of all, most of the funds needed to investigate these opportunities comes from STMicroelectronics and from European projects.
I am really grateful to the Automation Robotics and Trasportation group (ART). Furthermore several people from the University of Catania helped me a lot through every difficult subject. Last but not least I would like to thank all the people who still trusts in me and in my work despite my sharp nature.

preface

In the last few years my interest in motion of living beings started to grow day after day, month after month...without my control.

It all started during the complex systems course attended within my master degree (in automation engineering and complex systems control) at the university of catania. At that time my professors, Luigi Fortuna and Paolo Arena (respectively the coordinator and the turor of my Ph.D. course), introduced me to the world of non-linerity, complexity and reaction-diffusion and showed me how it is possible to consider a lot of natural phenomena under this frame: from waves to fur spots and walk patterns in animals. I decided that I would have spent time (even years) studying this field. As often happens, the decision was taken without consider-

ing difficulties and without a focused working strategy.
With this point in mind, day after day I took all the incoming opportunities to develop this research. After the final test of control of complex systems I worked on my master thesis on complex dynamics in population dynamics inside the city and than I come across Ph.D. course in human machine interaction in collaboration with STMicroelectronics: the opportunity to make (or at least, to try to make) things interesting for a big (very big, indeed) international company.

Almost impossible...

...but that was a huge opportunity and so I decided to take it, without considering difficulties and without a focused working strategy, again.

The years in this joint work with STMicroelectronics were very interesting and fruitful but as expected they have been very hard.
My project was slowed by a lot and even stopped sometimes. Nevertheless, month by month, I learned several technical things at the front edge of the technological research and how to merge company driven interest with my project.

After a couple of years I could now say that the aim work (and of this thesis) is to address the human machine interaction problem from analysis to synthesis.
It is important to notice that, both completeness and a rigorous mathematical (and analytical) treatment are out of the scope of this thesis. Moreover I have been lucky enough to have time to study, and work, on very different fields under this view I hope that this project could serve as a first information gatherer of multiple interconnected disciplines. Last but not least this is a work in progress and so stay tuned and don't be too hard in judgment.

A lot of works have been done in the field of motor control. Several different hypothesis have been described and reviewed to understand living beings on motor coordination (Latash 2008).
What is commonly referred to as motor control is indeed, an articulated problem that is, at least from a robotic perspective, often more suitably divided in: *sensing* (perception, cognition), *deliberation*, *planning*, *kinematic control* and *dynamic control* (as described in Fig 1).
Moreover all these things could, in line of principle, be represented by a flexible and learnable structure and so, as commonly described, they could be learned.

Robotic and machine learning communities are performing signifi-

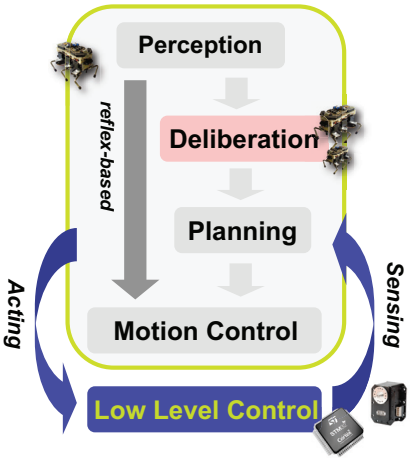


Fig 1. Block diagram for motion control. The high level control task is divided into several, hierarchical descending, sub-problems. Each problem has to be addressed almost separately and than their interaction must be considered.

cant effort to provide tools and framework in which it is possible at the same time to face classic robotic control problems and more deeply understand what so far is known on biological motor control (Peters 2007; Schaal 2010).
On the other side standard optimal control strategy, together with efficient learning strategy, are under investigation to cope with agile and under-actuated platforms (Tedrake 2005) to shorten the mechanical technological gap with the living counterparts.

Embodiment and environmental niche are two of the most cited words in this, quite new, emerging field (Pfeifer and Bongard 2006). Living beings (animals) grow and change experiencing at the same time their body, their control system and the environment (Fig 2). Under this point of view the motion control and the overall learning process have to be considered in a situated body: both innner system dynamical properties and environmental constraints are strictly related to the learned behaviours.

System dynamics includes all subsystems together with sub-prob-

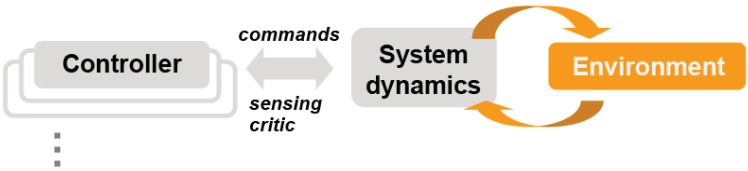


Fig 2. Under the ecological niche hypothesis the controller (i.e. natural or artificial system that performs parameter control on the hierarchical lower structure) has to learn control in a situated body. It is clear that the environment plays a fundamental role: it is impossible to consider the system dynamics without including it.

lem decomposition (*e.g.* low level motion control, muscle synergies). Under this point of view the learning process is achieved in a situated body: both dynamical properties and environmental constraints are strictly related to the learned behaviors.
It is almost useless to consider the system without considering the environment interaction and the whole environmental condition.

Moreover, in several recent publications is common sense that Central Nervous System (CNS) only give parameter control on a reflex-based (self-stabilized) complex system (Bizzi 1998; Pilon, De Serres et al. 2007) instead of trying to finely control the complex nonlinear structure.
The very complex control of motor capabilities in animals are achieved trough the hierarchical decomposition of task into simpler-and-simpler problems (Pilon, De Serres et al. 2007; Latash 2008) (as depicted in Fig 3).

In this understanding, a huge part of the learning process is at level of reflexes tuning and in an unconscious level.
Complex algorithmic solution are therefore proposed based on neural networks, spiking neuron model and reinforcement learning paradigm (Fig 4 and Fig 5)

Recent advantages in motor control and learning discourage the previously adopted hypothesis on internal model as a detailed kinematic perspective of body representation (Latash, 2008).

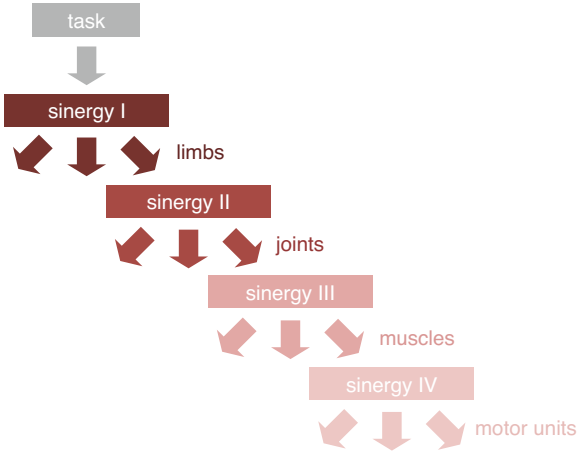


Fig 3. Task decomposition into simpler and simpler tasks identifying motor synergies (Bernstein, 1967) (Bizzi, 1998)

As described in (Feldman 2009) the learning process can be performed without knowing the exact relation between parameters, adopted policy and environment. Nevertheless critic sensing lets the system to correctly adjust parameters adopting, for instance, a descending gradient of the cost function.

The monitoring of real limbs trajectories in day life experiences for human motion control is certainly a big help to understand underling strategies.

Common used strategies are based on multiple cameras and markers. Emerging solution are now developed with inertial modules and sensors fusion techniques (Roetemberg, 2007).
In this quite new field the possibility of using low cost MEMS sensor (Vlasic, 2008) creates interests for the consumer electronics market, drastically shortening development time.

In the following sections, through the pages of this work, several different problems related to motion control, to living beings motion analysis and to its robotic counterpart will be addressed.
The strong underling motif of all the proposed algorithms and ar-

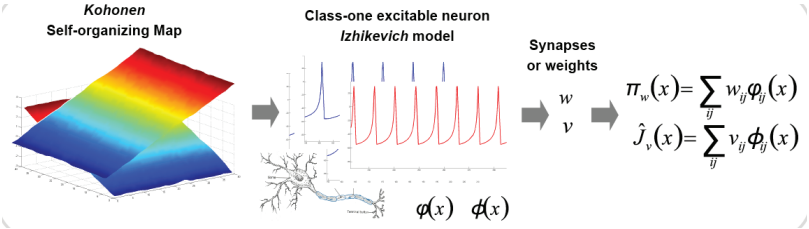


Fig 4. Example of a multi-layer solution based on Self-Organizing Maps (SOM) for the control of a dynamical system (e.g. torque limited pendulum swing-up). Inner layer are also based on realistic spiking neuron model.

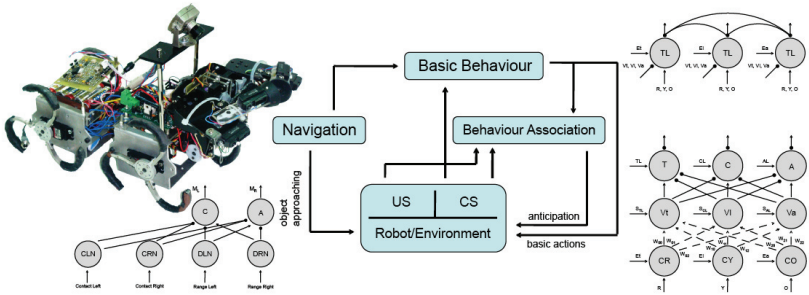


Fig 5. Complex robot control loop trough a classic biologically relevant algorithmic solution.

chitectures (both software and hardware) is the presence of a real environment interaction.

In particular sections will be dedicated to sensors, kinematic of the human body and its application to industrial robotics, mobile platform design and robotic architecture implementation, algorithms for motion planning, central pattern generation and perception.

The first step, explored in chapter 1, is an attempt to understand basis of motion starting form motion data analysis and closing the feedback of the eye-hand (sensor and actuators) coordination. After that, in chapter 2, several bio-inspired algorithms for motion control will be investigated. These algorithms explore different problems of control: from hard-wired reflexes implementation to complex high level motion planning learning trough reward function.

Fig 6. iNEMO™ evaluation board from STMicroelectronics: 10 DoF demonstration board based on accelerometer, gyroscope and magnetometer. It integrates a common quaternion based Kalman filter.

The problem of perception is faced in chapter 3.
The problem of sequences learning and of delayed reward is addressed in chapter 4.
A general framework for robot control and modular system integration is proposed in chapter 5.
The development of an advanced motion platform for inertial sensing is presented in chapter 6.

Let's start from the beginning.

chapter 1

exploiting motion and control of living beings:
from data analysis to feedback control

The incoming necessity of fast and reactive gesture recognition, for Human-Machine Interaction, and the diffusion of cheap and reliable MEMS multi-axial accelerometers, gyroscopes and digital compasses (*i.e.* magnetometer) are introducing a new discipline called "Inertial Motion Detection".
In this field already known problems, native for aeronautics, are the Inertial Navigation System (INS) and Attitude and Heading Reference System (AHRS).
Under this kind of open problems we are going to analyze all the difficulties about the implementation of an Inertial Mouse platform using accelerometers.
In order to understand underling dynamics in human motion and behind eye-hand coordination in humans, a complete sensor platform

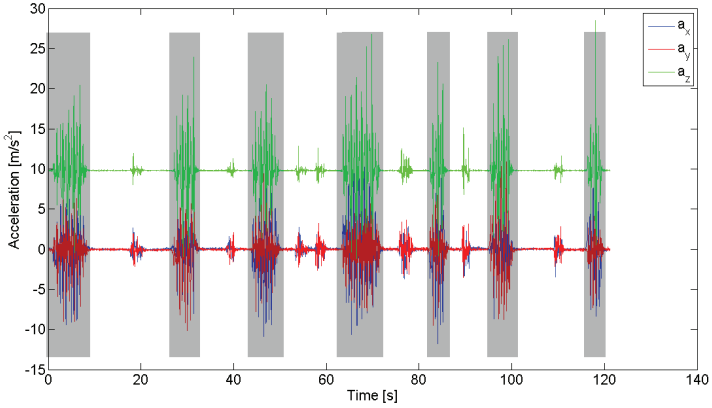


Fig 7. Acceleration pattern during a common walk on a flat floor. Acceleration data have been aligned using the computed orientation (obtained using a kalman filter). A motion detection algorithm has been used to identify motion contion to avoid long term integration. Highlight zones identify motion.

has been realized and acceleration data from it has been acquired.

Though the analyzed device is the LIS3LV02DL (a tri-axial digital accelerometer), all the considerations can be extended to any kind of analog and digital device.
This chapter is organized in two different parts. In the first one most of the common theoretical and numerical problems are discussed. In the second part algorithmic procedures based on proposed solution to those problems are presented.

Reference frame
Consider now a personal computer on a table (Fig 8). The accelerometer local reference system (O_{xyz_2}), in which accelerations are measured, is rigid with the mobile platform (O_{xyz_1}) and any translation and/or rotation between them could be considered constant.

For such a defined body the acceleration of the point O_2, expressed in the inertial reference system , can be, in general, evaluated as follows:

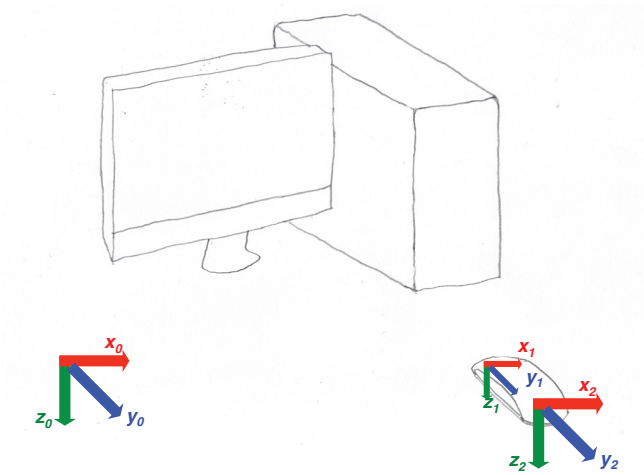


Fig 8. Simple sketch of the all adopted reference systems. The inertial platform and the accelerometer sensor are in different frames of reference.

$$\vec{a}_{O2} = \vec{a}_{O1} + \vec{\omega} \times (\vec{\omega} \times (O_2 - O_1)) + \dot{\vec{\omega}} \times (O_2 - O_1) + 2\vec{\omega} \times \vec{v}_{02}^{(r)} \quad (1)$$

where $\vec{\omega}$ is the angular velocity of the body (rotation of the O_{xyz_1} reference system) and $\vec{v}_{O2}^{(r)}$ is the relative velocity of the point O_2 with respect to the point O_1.
The accelerometer will, therefore, measure that acceleration $\vec{a_P}$ in the O_{xyz_2} coordinate system.
As in any common pointing device (*e.g.* optical mouse) motion should be interpreted as in a local reference system.
As described by the equation (1), the distance between O_{xyz_1} and O_{xyz_2} ($r = (O_2 - O_1)$) will cause the measurement of both centripetal/centrifugal force and Euler force due to the dynamic rotation of the system O_{xyz_1} (with $\vec{\omega}$ angular velocity) in an absolute reference system.
However these accelerations can be considered as a part of the motion of the accelerometer reference system (O_{xyz_2}). Furthermore their effects, on the adopted local system, are very similar (for very different reasons) to those present in any other pointing device (*e.g.* optical mouse off-lens-axis rotation in plane) and they are easily

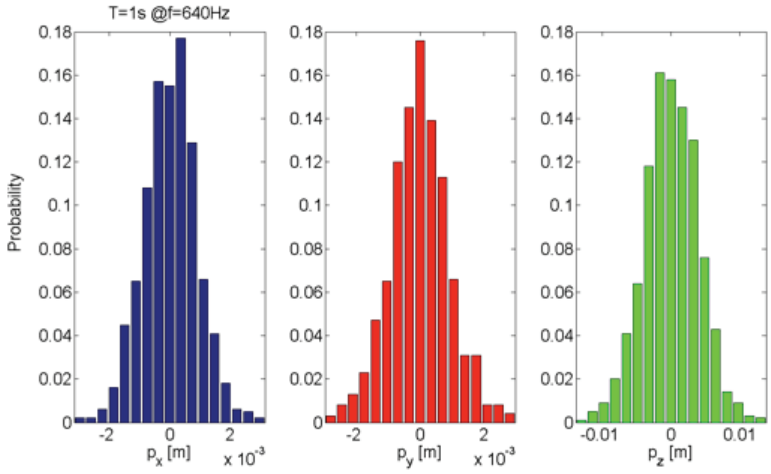


Fig 9. Statistical characterization of a dataset of N=1000 points with T=1*s* f=640*Hz* with respect to random walk of position in the absence of external signal and with offset calibration.

understood and compensated by the feedback control of the user (eye-hand coordination).
Coriolis effect, due to the motion of the accelerometer frame (O_{xyz_2}), with relative velocity $\vec{v}_{O2}^{(r)}$ within the rotating mobile platform system (O_{xyz_1}), is hard to compensate and needs to be analyzed in the accelerometer mechanical design.
The angle between the local reference (*i.e.* accelerometer frame, O_{xyz_2}) on the mobile platform (O_{xyz_1}) and the absolute reference (O_{xyz}) could be discarded.

Nevertheless, in order to establish a user friendly cursor control, the angles between the accelerometer reference (O_{xyz_2}) and the mobile platform reference (O_{xyz_1}) must be kept as small as possible or compensated with static rotation matrices $R_x(\psi)$, $R_y(\theta)$ and $R_z(\phi)$ as follows:

$$\vec{x}_1 = R_x(\psi)R_y(\theta)R(\phi)\vec{x}_2 \qquad (2)$$

where $\vec{x}_i = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T$ and

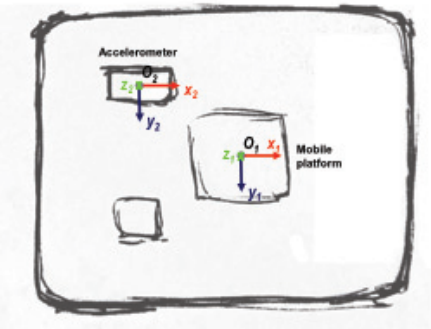


Fig 10. Mobile platform and accelerometer frame.

$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\psi) & -sin(\psi) \\ 0 & sin(\psi) & cos(\psi) \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix}$$

$$R_z(\phi) = \begin{bmatrix} cos(\phi) & -sin(\phi) & 0 \\ sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3)$$

As already defined all these matrices are constant and can be easily evaluated in a first calibration step. However if the structure is built to minimize this mismatch the overall rotation can be discarded for most applications.

Gravity acceleration

Due to the gravity force the accelerometer will measure approximately 9.806 $\frac{m}{s^2}$ (standard gravity acceleration is equal to 1*g*) even in a rest condition.

Though, in a first approximation, this component is on the inertial *z-axis* (*z*) and is (almost) perpendicular to the mobile platform motion plane (motion surface) and parallel to its *z-axis* (z_1) in a deeper analysis a different condition arises: this is due to the misalignment among the motion surface (*e.g.* table) and the absolute *xy* plane and due to the accelerometer frame imperfections.

Therefore the accelerometer offset values in all three dimensions must be updated through the entire path following.

Though the dynamic estimation of these offset values is not possible with just a *tri-axial* accelerometer, they can be considered constant

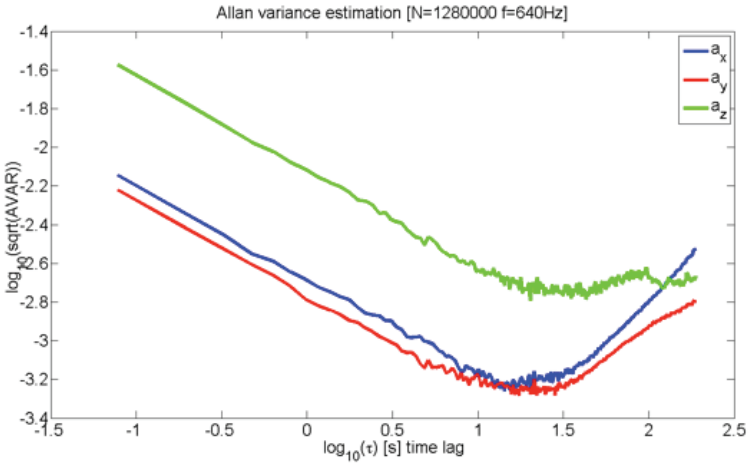


Fig 11. Plot of the square root of the Allan variance with respect to the time lag τ (seconds in logarithmic scale).

in a motion *xy* plane-constrained and under this hypothesis they can be estimated through time considering, in a first approximation, the off-plane motion as a disturbance.
Under all the analyzed constraints the numerical integration of the accelerometer readings (*i.e.* the acceleration vector $\vec{a} = \begin{bmatrix} a_x & a_y \end{bmatrix}$) should be, in line of principle, the velocity ($\vec{v} = \begin{bmatrix} v_x & v_y \end{bmatrix}$) and the numerical integration of this one should be the position ($\vec{p} = \begin{bmatrix} p_x & p_y \end{bmatrix}$).

$$\vec{v}_t = \vec{v}_0 + \int_0^t \vec{a}(\tau)d\tau \simeq \vec{v}_0 + \sum_{i=1}^{N} \vec{a}_i \Delta t \qquad (4)$$

$$\vec{p}_t = \vec{p}_0 + \int_0^t \vec{v}(\tau)d\tau \simeq \vec{a}_0 + \sum_{i=1}^{N} \vec{v}_i \Delta t \qquad (5)$$

Where $\Delta t = \frac{1}{f}$ and f is the sampling frequency.

From a computational point of view it is easier to consider the iterative form of the integration as follows:
$\vec{v}_{k+1} = \vec{v}_k + \vec{a}_k \Delta t$
and

$$\vec{p}_{k+1} = \vec{p}_k + \vec{v}_k \Delta t \qquad (6)$$

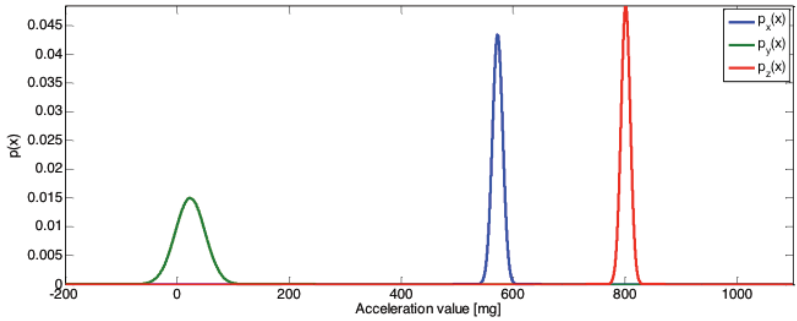


Fig 12. Example of density probability distribution of acceleration in static condition. It is important to notice that acceleration values show a sharp distribution with low variance.

Accumulation

The working data set consists on approximately 167 minutes ($10^4 s$) of continuous data acquisition. As described in the following sections the same data were, for different common statistical tools, split into dataset of various length.

Noise analysis: the Allan Variance method

The Allan VARiance (AVAR) technique differentiates the various noise sources for the sensor under test: quantization noise, Speed Random Walk (SRW)/white noise, correlated noise, bias stability and Acceleration Random Walk (ARW).

As described by the reference the AVAR has been estimated with respect to the time lag to indentify major noise component.

As it is possible to derive from Fig 11 (with common AVAR analysis) there are three main noise components: random noise, measure bias and ARW. With low averaging time τ the main component is the random noise SRW.

The effect of the random noise decreases with longer averaging time until a minimum is reached when the best bias estimation is achieved. As the averaging time increases, the variance starts to increase again (clearly evident in *x* and *y* components) since the

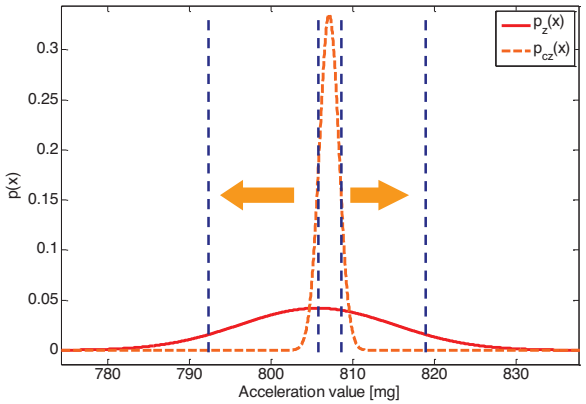


Fig 13. Density probability distribution spreading during motion (or shock) condition.

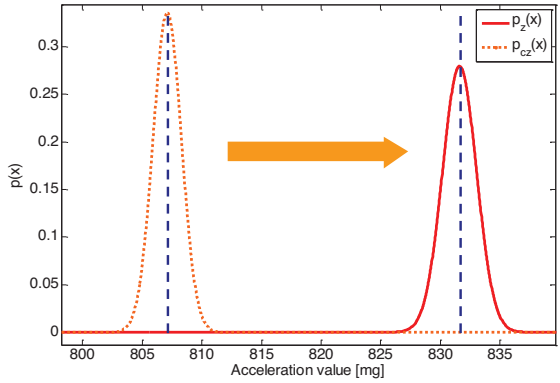


Fig 14. Average shift in off-plane axes z_1 during off-plane notion.

presence of the bias instability of the sensor takes the lead (ARW).

Noise Integration

The presence of random noise in all the accelerometer measurements cause both in the first integration (velocity) and in the second integration (position) a well known drift problem called “Random Walk” (RW) due to the not perfectly zero average (in a finite number of samples).

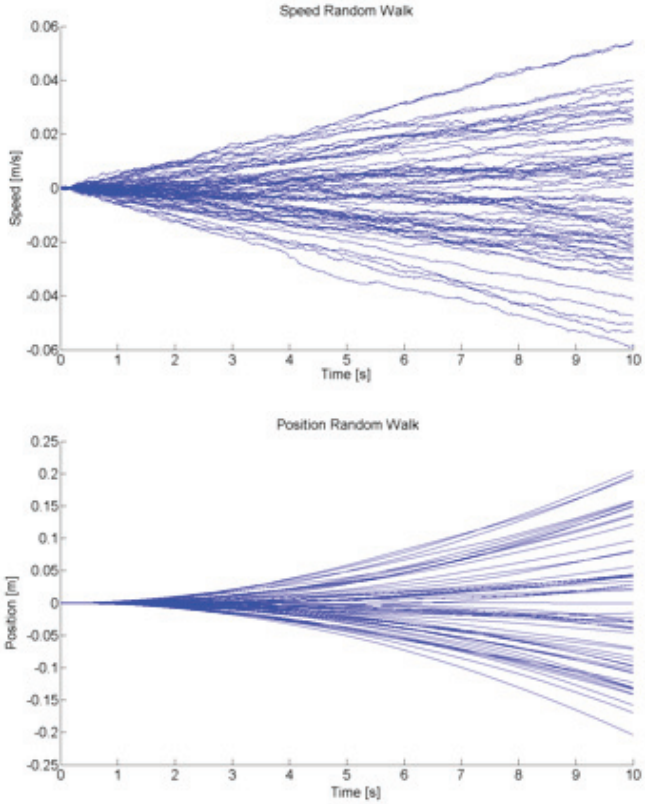


Fig 15. Example of N=70 dataset in which Speed Random Walk (SRW) (a) and Position Random Walk (PRW) (b) are evaluated trough time (T=10*s*).

One of the possible strategies to evaluate the effect of the random noise in the random walk behaviour is to identify its stochastic model in terms of average $\vec{\mu}_P$ and standard deviation $\vec{\sigma}_P$: the position $\vec{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$ is evaluated with acceleration integration in a time interval T in the absence of motion.

For the considered device (LIS3LV02DL) at a sampling frequency of f=640*Hz* the standard deviation of PRW is

$$\vec{\sigma}_p = \begin{bmatrix} 0.877 \cdot 10^{-3} & 0.925 \cdot 10^{-3} & 4 \cdot 10^{-3} \end{bmatrix} m \qquad (7)$$

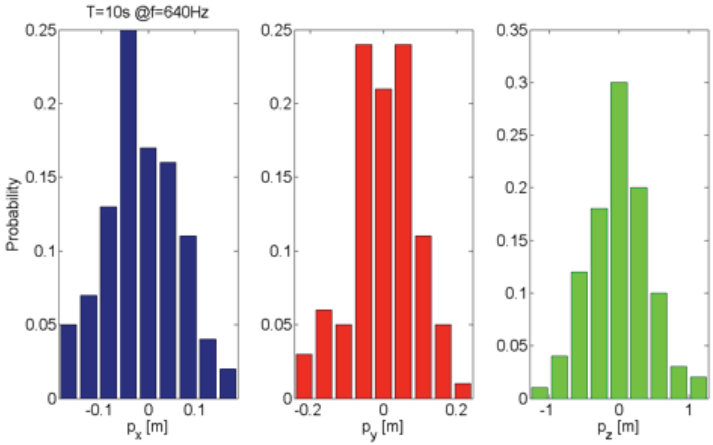


Fig 16. Statistical characterization of *N =100* dataset with T =10s f =640*Hz* with respect to random walk of position in absence of external signal and with offset calibration.

for a time interval T=1s (see Fig. 4) and it is

$$\vec{\sigma}_p = \begin{bmatrix} 77.3 \cdot 10^{-3} & 89.3 \cdot 10^{-3} & 0.432 \end{bmatrix} \; m \qquad (8)$$

for a time interval T=10*s* (Fig. 5).

It can be noticed that, as described from the literature, while the of the Speed Random Walk (SRW) increases almost linearly with the time the of the Position Random Walk (PRW) increases approximately with the square of the time.

Algorithmic procedures

Under all of these considerations the main problem in acceleration integration is to dynamically set to zero the integration (*i.e.* the velocity) periodically to avoid very large time drift. From this point of view, it must be taken into account that the application is mostly independent from “small” position errors (*i.e.* double integration of the acceleration) thanks to the strong stabilizing action of the human eye-hand coordination.

Motion reset and active offset

As described, thanks to the closed loop control, errors in position are

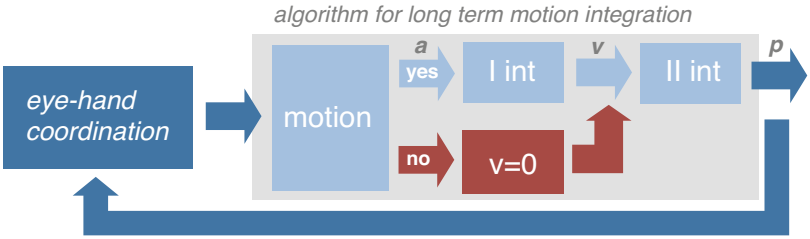


Fig 17. Functional block diagram of the eye-hand closed loop control of position with integration reset

in large part corrected, nevertheless even a small drift in speed will cause a numerical divergence during the second integration that is impossible to compensate. For this reason a “no-motion” condition recognition ($\vec{v} = 0$) is a critical feature for an efficient control algorithm. A such defined feature should also allow to correct all long term drift errors on acceleration due to instability and/or temperature thanks to an active offset evaluation during “rest” periods.

How does an accelerometer detect an integration-free “no-motion” condition?

Though from a mathematical point of view the problem has no

$$\vec{\sigma}_p = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\vec{p}_i - \vec{\mu}_p)^2} \text{ with } \vec{\mu}_p = \frac{1}{N}\sum_{i=1}^{N}\vec{p}_i \qquad (9)$$

solution because of the constant-velocity motion in the real world it is possible to discriminate “motion” condition with a time interval statistical analysis.

The density probability function of all the three axis is evaluated and, similarly to equation (9), a non zero variance (or standard deviation) is revealed due to the accelerometer random noise (Fig 11). Due to the motion of the mobile platform the density probability distribution spreads along abscissa as shown in Fig 13.

$$\vec{\sigma}_a = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\vec{a}_i - \vec{\mu}_a)^2} \text{ with } \vec{\mu}_a = \frac{1}{N}\sum_{i=1}^{N}\vec{a}_i \qquad (10)$$

and $\vec{a} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T$

It is important to remark that false positive motion condition is detected in case of acceleration shocks because they have almost same impact on the data distribution. Nevertheless, while in our application false “rest” conditions are disruptive for acceleration integration, the integration of shock acceleration data does not cause any further error beyond the already known SRW and PRW.

The presented approach needs a vector acceleration buffer of length N (one for each axes) and therefore the state of the mobile platform is evaluated after seconds. This time interval is seen as response delay from the user. As in any other statistical approach the error decreases with the number of samples. Therefore it can be greatly reduced increasing the sampling time (*i.e.* using more CPU time) and a delay-consumption trade-off of the algorithm must be performed.

Off-plane motion detection
A very similar problem could be defined to distinguish between On-plane (within the working two-dimensional surface) and Off-plane (outside that surface) motion. In this case the statistical value of interest is the average value of off-plane axes (z_1) within the N samples dataset. In an ideal on-plane motion this value should be almost constant and equal to the active offset value. However as in the case of standard deviation, previously analyzed, it happens that additional motion-induced “almost zero-average” noise is present as depicted in Fig 13 (e.g. high frequency shock or low frequency mechanical bend) and described in the previous section. Therefore the motion can be considered on-plane if the difference with offset value is below a threshold δ_z.

$$| \mu_z - off_z | \leq \delta_z \qquad (11)$$

Where off_z is the current active offset for z_1 axes.

Position-Cursor function
Any pointer device has a non-linear function that assigns a cursor velocity for a given pointer-case velocity. In other words different variation in cursor position is assigned for the same variation in

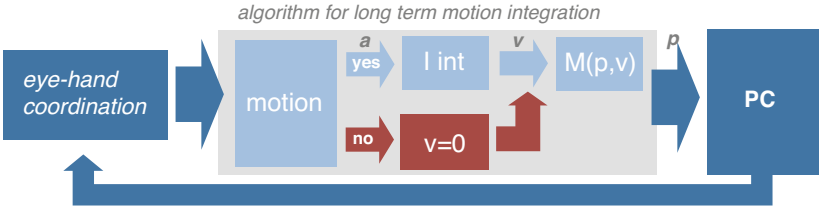


Fig 18. Overall functional block diagram of the eye-hand closed loop control of position with integration reset and with Position-Cursor transformation.

pointer-case position with respect to the time this variation occurs.

$$\vec{p}_{k+1} = \vec{p}_k + G_p(\vec{v}) \cdot \vec{v}_k \Delta t \text{ with } G_p(\vec{v}) = \alpha \mid \vec{v}_k \mid^{\beta} \quad (12)$$

and

$$\| \vec{p}_{k+1} \| \leq \vec{D} \text{ with } \vec{(D)} = [\ \tfrac{W}{2} \quad \tfrac{H}{2}\ ] \quad (13)$$

In which α is a constant dependent on pixel/inches ratio of the monitor, and are respectively the monitor width and the monitor height [pixel], useful to introduce a saturation in the position along both *x* and *y* monitor axis, and β is user defined distortion factor. From experimental data it has been found that typically $20 \leq \alpha \leq 100$ and $0 \leq \beta \leq 1$.

Enhanced functionality

Thanks to the developed algorithm, based on accelerometer data, it is possible to enhance the classical two-dimensional mouse operation in case of different orientation detected within the mobile platform rest condition or in case of particular acceleration pattern (*e.g.* shocks in a particular axes).

The first straightforward application of this consideration is to implement an orientation-free mouse and a shock triggered click.

Another simple application could be a tri-dimensional mouse operating mode that can be implemented in case of reversed *z-axes*.

With the add-on of a yaw gyroscope it is also possible to develop a tri-dimensional pointer.

In particular referring to the a multiple circles test (where the number of consecutive circles is n = 5, *i.e.* typical test for hand pointer

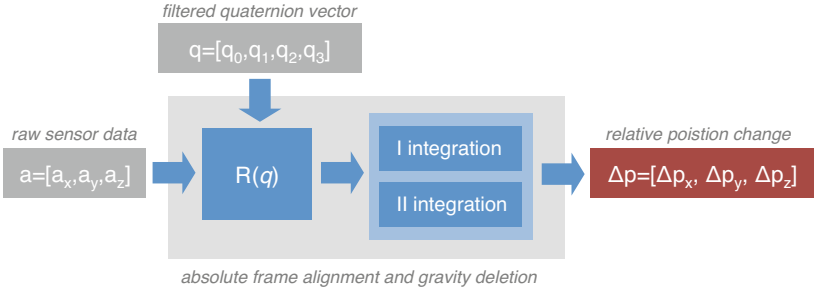


Fig 19. Block diagram for absolute referenced acceleration integration. Output from Kalman filter (*i.e.* rotation matrix or quaternion) is used for data alignment and gravity deletion. Two step of integration are used to first achieve speed and then to get position. Zero Velocity Update is used with motion detection algorithm.

devices) trajectory tested at different speed:
- the test requires high accuracy in position estimation within a very long time interval (several minutes) while real mouse movements are typically performed in couple of seconds;

$$r = 20mm \text{ and } s = n \cdot 2\pi r = 628mm \text{ so } y = 1\frac{mm}{s}$$

and

$$s = \frac{s}{y} = 628s \sim 10.5min$$

The test does not consider the strong feedback correction naturally performed by the user coordination (eye-hand coordinator).
Under these assumptions, even if we consider the acceleration noise as the only present technological constraint (and this is clearly not true), in order to apply a trivial double integration of the accelerations to estimate motion, the required device specifications are far away from the real device specifications.
In fact, assuming a normal distribution model, the numerical simulations show that to achieve less than 2 *mm* (10%) of error in T = 628*s*, as shown in Fig. 11, less than $\sigma_a = 10^{-6}mg$ (standard deviation) is required with a sampling frequency of $f_s = 600Hz$ (while real measured $\sigma_a > 4.5 \cdot 10^{-4}mg$).

However it can be shown that those condition reports about motion estimation tests and not about real user driven mouse operations. Moreover it is important to understand that, from a theoretical point of view, the PRW proportionally decrease both with the square root of the number of accelerometers and with the frequency (*e.g.* 100 accelerometers have 10 times lower PRW, the same as 10 accelerometers with 10 times higher frequency). Under this point of view, array or matrices of accelerometer, sampled at higher frequency, could theoretically be used to improve overall system performances.

To summurize, in this chapter, the basis of accelerometer data processing algorithm has been presented.
The movements of a mobile platform actuated from a user have been recorded and analyzed in order to perform a feasibility study of a two dimensional pointer device.
Moreover it has been shown that the “no-motion” condition accuracy detection and the Position Random Walk (PRW) standard deviation are both function of the amplitude of the noise density of the accelerometer. Common short term motion (approximately $< 2s$) is up to now considered precise enough to permit a user-friendly device. Very long term drift is virtually eliminated thanks to the active offset evaluation.

Though very complex trajectories could be traced, the midterm PRW is still too big to achieve the user desired position. Infact, although the use of multiple accelerometer devices together with more complex data filtering algorithms (e.g. Kalman filters) has not be tested yet, it must be noticed that, as showed in the Test Analysis section, when no assumption could be done on the device movements and the performances are tested in a long-term run the achievable precision is still lower than what required in this kind of application. Specifications test on the PRW with the inertial sensors especially in the mid-range time interval (between $2s$ and $10s$) has been performed.

chapter 2

algorithms for motor and motion control: *from reflexes to complex coordination*

Let's start from the problem of motion at a level of joint coordination. If we think at the motion of the tip of one of our finger or at motion of a part of our body, the coordination of each muscles comes out as a reflex.

Well, it is all but trivial.

Multi-link structures (referred to as kinematic chains), move in a very complex and non linear way. At a joint level and from a robotic perspective this problem has been studied since decades and it is addressed, especially in industrial robotics, as kinematic problem.

The industrial robotics community, has divided it in two sub-prob-

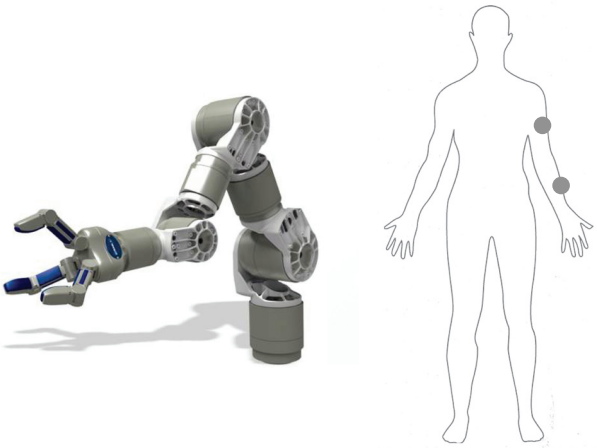


Fig 20. Considered kinematic chain on a human subject and a robotic counter part (from Schunk).

lems: forward kinematic problem and inverse kinematic problem. The former intends to understand the position of the end-effector (end, or tip, of the whole kinematic chain) given the joint angular position. The latter intends to find angular values for each joint for a given end-effector position.
The inverse kinematic problem is the one we have to face, even not consciously, when we want to grab a cup of coffee with our left hand.

For all these reasons we are going to consider now a industrial-like robotic structure and to look for different ways to control it and to solve, already solved problem, in a different and more flexible way. As entry point it is important to analyze the framework of industrial robotic and to understand why in the last decades these problem were solved in a particular way.

Robotic manipulators are widely used in several fields of applications: for mechanical structure assembly, in industries, on roving platforms for manipulation of heavy or dangerous objects, in humanoid robots for research and others.
These robotic structures, depending on the application, can present a potentially large number of degrees of freedom and can work either in a 2D-like or 3D environment. In some cases, a redundant

configuration can be used increasing the complexity needed to define a model of the structure and to develop a control algorithm. Depending on the application, an important performance index that can be considered is the computational power consumption.
In fact, it constrains the control frequency and therefore the possibility to develop an algorithm embedded on a low MIPS microcontroller-based board for real-time applications.

The robustness against singularities, the capability to face angular constraints in the joint space or other constraints in the operating workspace (*e.g.* obstacle avoidance) are important aspect that have to be considered.
The three-dimensional positioning task (inverse kinematic problem, IK) for a redundant serial manipulator is an already-solved problem through various classic approaches, like pseudo-inverse Jacobian and Jacobian transpose algorithms.
Generally the problem is first mapped onto the velocity domain by using the first-order instantaneous kinematic relation, which is given by the Jacobian matrix (J), and then solved in that domain.
For redundant manipulators J has a null space and therefore minimization algorithms can be used to achieve various subtasks like singularity (or obstacle) avoidance (Nenchev, 2004) or to cope with angular joint limits (Siciliano ed., 2008).
Several approaches have been proposed addressing the joint limit problem in the velocity domain (see for details Ahn, 2002 and Chan, 2005), nevertheless, as discussed in (Shimizu, 2008) , the best way to find a suitable solution consists into considering the problem in the position domain.

Shimizu and Kakuya (Shimizu, 2008) and, similarly Tondu (Tondu, 2006), analytically derive the inverse kinematic model for a 7-Degrees-of-Freedom (DoF) redundant anthropomorphic manipulator with joint limits. Nevertheless, their results are not easily generalizable to any kind of manipulator. Almost the same considerations can be done to the Lee and Bejczy's closed form IK solution based on joint parametrization technique (Moradi, 2005) and to Moradi and Lee redundancy resolution method for minimizing elbow movement

do not address neither all possible configurations of the manipulator (Moradi, 2005). Under these hypotheses, the aim of this chapter is to propose a low computational effort, numerical approach, that can provide, for any kind of robot manipulator configuration, a sub-optimal solution, for both forward kinematic (FK) and inverse kinematic (IK) problems. The proposed strategy makes the robot able to react to environmental changes and therefore to cope with dynamically changing joint limits.

The approach is based on the already developed Mean of Multiple Computations (MMC) Recurrent Neural Network (RNN) algorithm which provides a fast, flexible and robust to configuration singularities sub-optimal solution to the discussed problems (Cruse, 1993), (Arena, 2009).
Although several minimization algorithms can be used for DK and the IK problems, as discussed before, the proposed MMC approach (Cruse, 1993) is quite simple and bio-inspired (Cruse, 1998). In fact several recent studies in limb Neurophysiology indicate that in particular neural sites, like, for example the spinocerebellar neurons in vertebrates, proprioceptive sensory signals are processed in a flexible network organization which encodes functional relationships among the limb segments, resulting in a global representation of the limb parameters.

In movement planning, the weights of inputs from various limb segments might be biased toward an explicit representation of the whole limb (Bosco, 2001). This representation is also somatotopically distributed, allowing multiple (redundant) representations of the same limb parts. This rule adapts both to cerebral and to cerebellar cortex (Kandel, 2000). Also corticospinal neural cells in the Primary motor cortex of rhesus macaques are considered as representing different combinations of muscles that constitute functional synergies for the execution of both single and multijoint limb movements (Park, 2001).

Very recent studies suggest that that the cat motor cortex controls the musculature in an integrated manner rather than singly and separately and that there appears to be a substrate for the dynamic

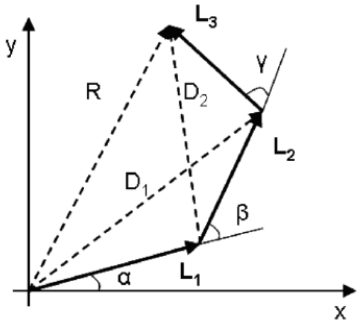


Fig 21. A planar manipulator consisting of three vector segments: L_1,L_2, L_3. The joint angles are: α, β and γ. The end-effector position is indicated by the vector R, D_1 and D_2 are two additional vectors describing the diagonals.

selection of motor output patterns, laying the basis for the idea that flexible selections of motor outputs occur on a moment-to-moment basis as a result of the motor cortex's activation state (Capaday, 2009).
Taking inspiration from these considerations, the MMC approach represents a candidate, yet simple model of such an integrated network organization. It provides a framework in which a global, distributed and redundant representation of the limb parameters is encoded, providing a concurrent natural solution to the IK and DK problem iteratively, with the addition of a robust real-time adaptation to external obstacles that, differently from other approaches (Seraji, 1999), (Cheng, 1998), (Kheradpir, 1988) are formulated as joint limits. This approach allows a kind of "on-the-fly" flexibility. From the pure algorithmic point of view the MMC approach provides also a selectable precision-speed trade-off.
Furthermore the other classical approaches taken into consideration for comparison, do not address the problem in a dynamically changing environment. In fact, the cited solutions consider at least joint limits only if they are known and constant through time.

This chapter is organized in four main sections. First the general MMC model is discussed. Then, multiple control strategies are described in order to achieve desired end-effector position and/or

orientation control. After that, some experimental results are presented and finally, results of the proposed model are analyzed and compared to the most common velocity domain algorithms in terms of timing performance, iterations per convergence and robustness to external disturbance.

Model description

Since many years neural networks have been largely used to solve manipulator control problems, in which the set of variables involved in the process are combined in a single pattern (Miller 1990), (Ritter 1992). The output values are retrieved by completion of an even partially defined pattern given as input.

TWithin the neural network approach, the Mean of Multiple Computation is an interesting method introduced in (Cruse, 1993), and further exploited in (Steinkuhler,, 1998), that can be used to create a model of multi-link, *m*-dimensional structure by using simple geometrical relations between arcs of the same complete graph.
Going deeper into details we can consider as a simple example, a planar manipulator with three degrees of freedom like the one shown in Fig 21.
The procedure herewith exposed can be further extended to more complex structures using simple principles of the graph theory. The modelling phase is important for successive formalization strategies. In fact the proposed control mechanisms are strictly related to the characteristics of the MMC-based model formulated for the redundant manipulator.

MMC manipulator model

The main idea followed to construct an MMC-based model is that, looking at the serial manipulator as a geometrical multi-link structure, it is possible to compute each geometric quantity (*i.e.* vectors $\in R^2$ in the example) in several ways, using different graph paths, and then average over them.
For instance for the link L_3 of the planar manipulator in Fig 21 all the following relationships can be considered:

$$\begin{aligned} \mathbf{L_3} &= -\mathbf{L_2} - \mathbf{L_1} + \mathbf{R}, \\ \mathbf{L_3} &= \mathbf{L_1} + \mathbf{D_2} - \mathbf{D_1}, \\ \mathbf{L_3} &= -\mathbf{L_2} + \mathbf{D_2}, \\ \mathbf{L_3} &= +\mathbf{R} - \mathbf{D_1}. \end{aligned} \qquad (14)$$

According to the MMC theory, looking at the equations in (14), the mean value of L_3 (L_{3m}) can be computed as:

$$\mathbf{L}_{3m} = \frac{1}{4}(-2\mathbf{L}_2 + 2\mathbf{R} + 2\mathbf{D}_2 - 2\mathbf{D}_1). \qquad (15)$$

In this way the MMC model provides a parallel computation of each variable that represents a key point to guarantee robustness to singularities in the IK problem formulation.
Referring to Fig 21 the complete geometrical structure of the manipulator, defined in terms of an MMC network, is constituted by vectors L_i (real links $\in R^2$) as well as by vectors D_i (virtual links, *i.e.* diagonals) and R (end-effector absolute position) that will be referred to as virtual links.
The complete pattern for the defined model can be expressed as:

$$\mathbf{P} = (\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3, \mathbf{R}, \mathbf{D}_1, \mathbf{D}_2). \qquad (16)$$

Therefore, it is possible to define the matrix M (weight matrix of the network) that summarizes all the relationships within the considered graph as shown in equation (17).

$$\mathbf{P}_m = \mathbf{M} \cdot \mathbf{P}, \qquad (17)$$

where, as for L_{3m} in equation (15), P_m is the vector of the average values of each quantity present in P.
The general form of M can be computed systematically using simple geometric relationships and principles of the graph theory (Diestel, 2005): for instance, for the simple three DoF manipulator shown in the previous section, M can be defined as in equation (18):

With the considered MMC model, defining input and output of the network (modifying the weights), several unknown elements of the input vector can be reconstructed, iteratively, starting from a reference input using the matrix M.

$$M = \frac{1}{4}\begin{pmatrix} 0 & -2 & 0 & 2 & 2 & -2 \\ -2 & 0 & -2 & 0 & 2 & 2 \\ 0 & -2 & 0 & 2 & -2 & 2 \\ 2 & 0 & 2 & 0 & 2 & 2 \\ 2 & 2 & -2 & 2 & 0 & 0 \\ -2 & 2 & 2 & 2 & 0 & 0 \end{pmatrix} \qquad (18)$$

Therefore the same model can be used, for instance, both for inverse and direct kinematic problem solving. In the first case a desired end-effector position R_d is given and L_1, L_2 and L_3 are retrieved and therefore α, β and γ. To impose as known value the vector R_d, the corresponding line in the weight matrix M (4th row) is modified deleting the relations with the other variables.
For such a defined problem the modified weight matrix M_{IK}, should be expressed as:

$$M_{IK} = \frac{1}{4+d}\begin{pmatrix} d & -2 & 0 & 2 & 2 & -2 \\ -2 & d & -2 & 0 & 2 & 2 \\ 0 & -2 & d & 2 & -2 & 2 \\ 0 & 0 & 0 & 4+d & 0 & 0 \\ 2 & 2 & -2 & 2 & d & 0 \\ -2 & 2 & 2 & 2 & 0 & d \end{pmatrix} \qquad (19)$$

Moreover, as described in (Cruse, 1993), (Cruse, 1998), the P vector for the current iteration i can be computed as in the following equation:

$$\mathbf{P}(i) = \mathbf{M}_{IK} \cdot \mathbf{P}(i-1). \qquad (20)$$

It must be noticed that in (18), a damping factor d has been introduced, in the diagonal elements, for a speed-accuracy trade-off: for large values of d the system is faster but can show overshooted behaviour while for small values the system dynamics is smoother but slower.
It is important to remark that equation (19), with the modified matrix M_{IK}, implies: R*(i)* = R*(i-1)*. This leads the system, through a certain number of iterations, toward the desired geometrically-feasible configuration in which results:

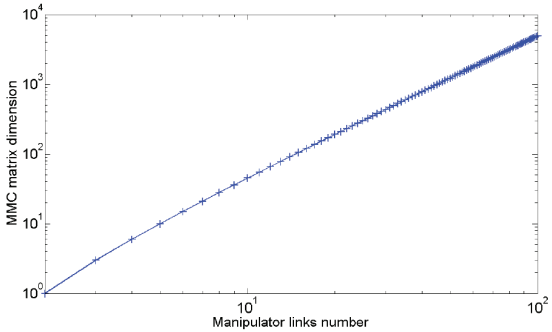


Fig 23. MMC complexity increases with number of links of the manipulator geometrical structure.

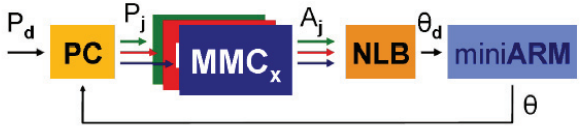


Fig 22. Algorithm block diagram for a given absolute reference input pattern, P_d. PC is the Pattern Constructor, MMC_x, MMC_y and MMC_z are the three-dimensional linear computational networks. NLB is the Non-Linear Block and miniARM is the manipulator itself (simulator or real robot). P_j are the MMC input pattern while Aj are the outputs. d are the desired angular values and are the read joints values.

$$\| \mathbf{R}_d - \mathbf{R}(i-1) \| \leq e_{tol}, \qquad (21)$$

where e_{tol} is the chosen position error tolerance.
In direct kinematic problem solving, α, β and γ and therefore L_{1d}, L_{2d} and L_{3d} are given while R is retrieved as output.

Graph extension
Thanks to the graph theory it is possible to extend the model to various architectures in order to address the problem for structures with a different number of degrees of freedom, eventually even for a parallel manipulator. As described in Fig 23, the dimensions of the MMC linear computational matrix (*NxN*) depend on the number of the arcs of the complete graph (Fig 23) and so it can be determined referring

to the number of manipulator links, n_l, as follows:

$$N = \frac{n_l \cdot (n_l + 1)}{2} \qquad (22)$$

Control strategies

In order to guarantee the redundancy for the 3D positioning and orientation task, a seven degrees of freedom serial manipulator structure has been considered. The modeled geometrical structure is a four link serial structure and therefore more than one degrees of freedom are computed with the same geometrical quantity (e.g. two relative joints associated with
the same link).
A complete pattern $\vec{P} = [\ P_x \quad P_y \quad P_z\ ]^T$ for this kind of structure, made up of all the arcs present in the graph (similar to the one seen in Fig.1), can be written as:

$$\mathbf{P}_j = (L_{j1}, ..., L_{j4}, R_j, D_{j1}, ..., D_{j5}), \quad j = x, y, z, \qquad (23)$$

where, as described before, both real and virtual links are present. Due to our hardware implementation , *i.e.* the servo motor features, also joint angle limits have to be considered in the model and therefore the linear model introduced in the equation (20) needs a further extension.
In this section three different problems will be considered, dealing with the end-effector position control with absolute reference input and relative error feedback. Moreover a general end-effector configuration control (*i.e.* both position and orientation) is proposed.

Position control with absolute reference input

The simplest extension of the two-dimensional positioning of a planar redundant serial manipulator (shown in Fig 21) introduced in the previous section (deeply described in Cruse, 1993) is the tridimensional positioning of the end-effector for the redundant serial manipulator in space (*i.e.* R^3).
Each vector component of the geometrical links is processed, iteration after iteration by a different linear MMC network and then given as input at the non linear block (NLB).
The single blocks of the control architecture are described below.

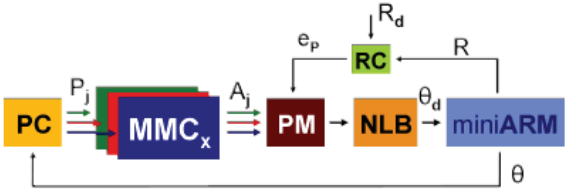


Fig 24. Algorithm block diagram with relative position error feedback (eP). PM is the Pattern Modifier block and RC is the Reference Comparator block.

PC (Pattern Constructor): it defines inputs and outputs within the pattern structure (see equation (23)) for the iteration i, modifying, similarly to (19), the weight matrix M to obtain relation (24).

$$R_j(i) = R_{jd}, \qquad j = x, y, z. \tag{24}$$

The PC block also sets input values ($P_j(i)$) for the network for the given vector reference $P_{jd} = (..., R_{jd}, ...)$, with $j = x, y, z$ and the actual angular positions $\vec{\theta}(i)$ (values read from the encoders or simulated): MMC (linear blocks): for a given vectorial input pattern P_j these structures compute separately and linearly the outputs A_j , in all the three dimensions.

$$\mathbf{A}_j(i+1) = \mathbf{M} \cdot \mathbf{P}_j(i), \qquad j = x, y, z. \tag{25}$$

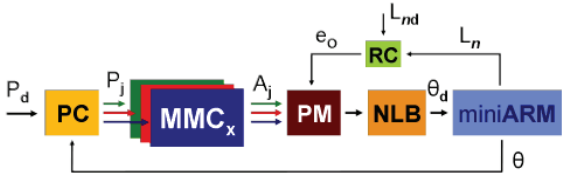


Fig 25. Algorithm block diagram with relative position error feedback (e_P). PM is the Pattern Modifier block and RC is the Reference Comparator block.

For example, the desired link configuration for L_{j3} of the planar manipulator (Fig 22) is obtained iteratively as follows:

$$L_{j3}(i+1) = \frac{1}{4+d}(-2L_{j2}(i) + 2R_j(i) + 2D_{j2}(i) + \\ -2D_{j1}(i) + d \cdot L_{j3}(i)), \qquad j = x, y, z. \tag{26}$$

NLB (Non-Linear Block): it defines the links inextensibility and other constraints known a priori (*e.g.* servo angular operating range), since these are not considered in the linear MMC block (Cruse 1993), (Cruse, 1998). So NLB transforms the networks output Aj in desired angular position $\vec{\theta}_{kd}$ for each joint (*k* = 1; 2; ...; 7).

$$\begin{aligned} \theta_{1d} &= g_1(\mathbf{L}_1) \\ \theta_{2d} &= g_2(f_1(\alpha_1, \theta_{1d}), \mathbf{L}_2) \\ \theta_{3d} &= g_3(f_2(\alpha_1, \alpha_2, \theta_{1d}, \theta_{2d}), \mathbf{L}_2) \\ &\dots , \end{aligned} \tag{27}$$

in which the functions f_k implement a rotational transformation of reference systems from absolute to particular local joint components while the functions g_k rebuild the angular value from given link components: in the simplest case, *i.e.* for the first link, it results

$$g_1(\mathbf{L}_1) = tg^{-1}(\frac{L_{y1}}{L_{x1}}). \tag{28}$$

As it is possible to see in equations (27) both 2d and 3d are computed with the same link, L_2 due to the fact that in a 3D problem more than one angle can be associated to a link.
It must be remarked that the introduction of the NLB, of primary importance for the control of the real manipulator, modifies the dynamics of the system affecting in some cases the global optimum convergence of the overall network.

Position control with relative error feedback
The most relevant difference with the control scheme described in the previous section is that the control action is given, with the addiction of the Reference Comparator (RC) block, by the feedback of the relative position error (e_P) instead of an external absolute reference (P_d), as follows:

$$\mathbf{e_P}(i) = \mathbf{R}_d - \mathbf{R}(i). \qquad (29)$$

Equation (29) computes the error value for the current iteration (e_P(i)) with respect to the absolute instantaneous position of the end-effector *R(i)*. In some application its value can be measured directly (*e.g.* with stereoscopic vision).
Each error component (e_{jP}) modulates a different MMC layer according to the following rule:

$$R_j(i+1) = R_j(i) + K_P \cdot e_{jP}(i) \qquad j = x, y, z, \qquad (30)$$

where (K_P) is a gain able to modulate a damping factor in the position control and to decide a speed-accuracy trade-off.
PM: Pattern Modifier, it modifies the R components in the network outputs (A_j) proportionally to the measured end-effector position error as described in equations (30).

Absolute position and orientation control with relative error feedback
The MMC network architecture maps the real manipulator links into vectors. Therefore each quantity results to be invariant (in R^3) to rotations around its extension direction.
Orientation control, under this point of view, can operate with two degrees of freedom following the reference as desired configuration for the last n-link (L_{nd}). It is possible to modify the weight matrix M to ensure the following relation through iterations:

$$L_{jn}(i+1) = L_{jnd}, \qquad j = x, y, z. \qquad (31)$$

Although (31), similarly to (24) is able, thanks to the NLB, to let the system relax towards the desired orientation it also gives an even more pressing constraint because the orientation would be kept through all the algorithm iterations. Therefore the feedback for the orientation error (e_O) is analyzed.

$$\mathbf{e}_O(i) = \mathbf{L}_{nd} - \mathbf{L}_n(i) \qquad (32)$$

PM changes the L_n components in the output patterns (A_j) proportionally to that component error (e_{jO}) in order to allow the network to relax toward the desired orientation as follows:

Link	Dimension $[m]$
L_1	0.109
L_2	0.109
L_3	0.125
L_4	0.05

TABLE I Lengths of the link of the considered manipulator

Angle	limits $[rad]$
θ_1	$[-\frac{\pi}{2}, \frac{\pi}{2}]$
θ_2	$[-2.2, 2.9]$
θ_3	$[-\frac{\pi}{2}, \frac{\pi}{2}]$
θ_4	$[-2.9, 2.2]$
θ_5	$[-\frac{\pi}{2}, \frac{\pi}{2}]$
θ_6	$[-2.2, 2.9]$
θ_7	$[-\frac{\pi}{2}, \frac{\pi}{2}]$

TABLE II Constraints of each joint of the considered manipulator

$$L_{jn}(i+1) = L_{jn}(i) + K_O \cdot e_{jO}(i), \qquad j = x, y, z, \qquad (33)$$

where (K_O) is a constant able to modulate a damping factor in the orientation control.

Simulation and experimental setup

The considered manipulator is custom built with seven degrees of freedom (revolute joints) in which the z-axis of j-link is 90° rotated around x-axis from the (j-1)-link.

The physical structure can be mapped, as requested for the MMC modeling, into the geometric theoretical model links whose lengths are reported in TABLE I.

Due to the chosen real robot architecture, angular limits are not equal for each joint (see TABLE II). As described in

the previous section, each presented control strategy has been

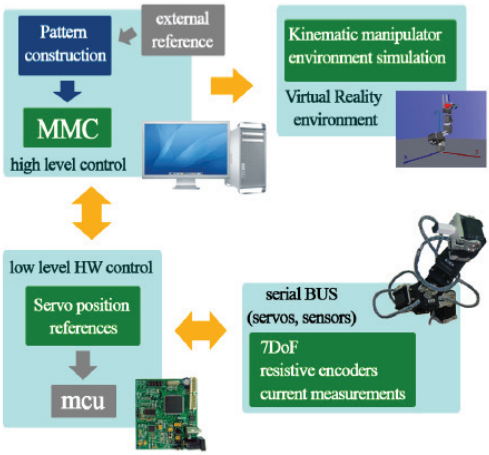


Fig 26. Hardware configuration of the experimental setup: low level microcontroller AVR32 board, actuators and sensors bus architecture and high level control host computer.

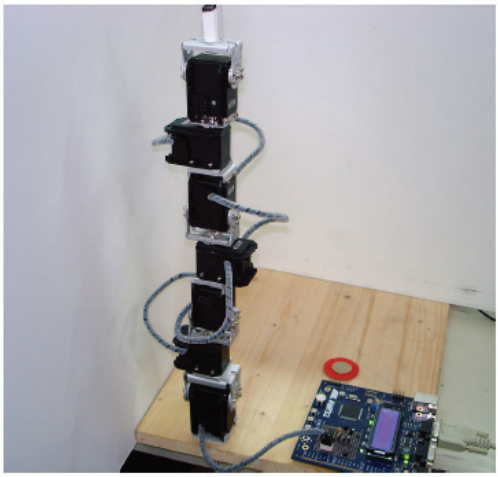


Fig 27. Experimental setup with the real robot (7-DoF manipulator) and the 32 bit microcontroller board.

tested both in simulation and on real robot (*i.e.* the miniARM shown in Fig 27).

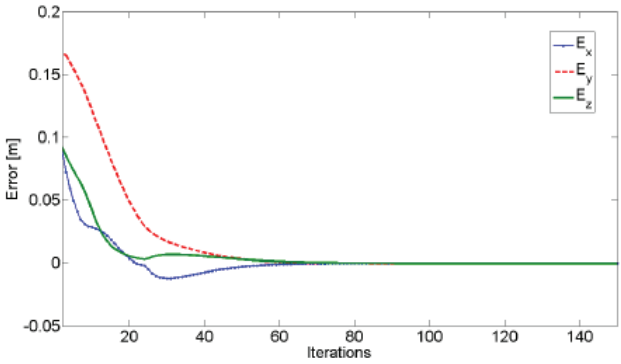


Fig 28. Position errors through iterations in trial with absolute reference:simulation results tested in the real robot using simulated angular values as real joint references.

The proposed algorithms have been implemented both in the PC and in a custom microcontroller unit (mcu) board.
Respectively a Dual Core Intel Centrino *2.2 GHz* host computer with 2GB of RAM and an UC3A AVR32 mcu with *66MHz* of maximum clock and only 64KB of RAM were used as comparison platforms (Fig 26). As sketched in Fig 26, in our first implementation the overall manipulator control can be split in two levels: low level (hardware) control and high level control. The low level control is, in all cases, achieved thanks to a mcu-based board that is used both to acquire information from the distributed sensory system and to control the actuators. Moreover a serial bus is used for low level communication purposes. The high level control and the data logging are made through a host PC connected to the board via serial interface (with a USB-to-serial transceiver). In the second (embedded) version of the control algorithm all the MMC-based calculations are directly executed locally in the mcu-based board while the PC is just used for data logging and for the Virtual Reality (VR) simulation environment.

Simulation and experimental results
Each of the proposed control schemes has been tested not considering the high level control platform (both mcu-based and PC) to compare tasks achievement and performances.
In order to guarantee a reliable and fast end-effector positioning,

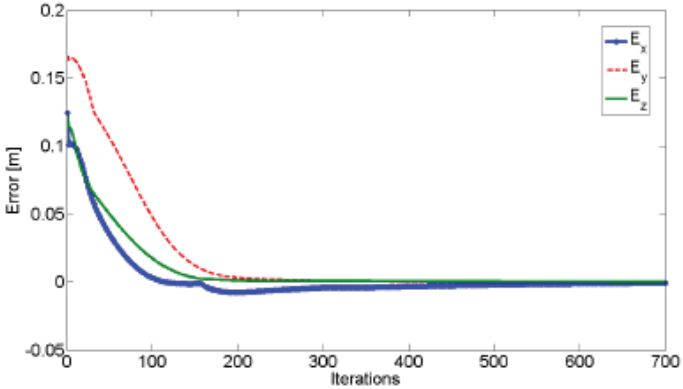


Fig 29. Errors through iterations in position and orientation control with absolute reference (in position) and relative feedback (in orientation): simulation results tested in real robot using simulated joint values as references for real joints.

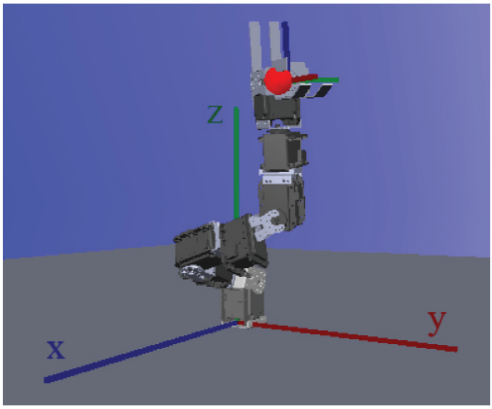


Fig 30. Last manipulator and target (circle) configuration in a simulated environment in the simple position control task.

in each trial, a damping factor proportional to the position error has been chosen as follows:

$$d(i) = d_S + K_S \cdot \|\mathbf{e_P}(\mathbf{i})\|, \qquad (34)$$

where d_s is the initial damping factor and K_s is a constant.

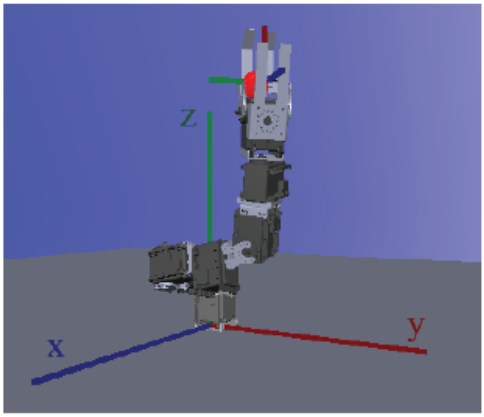


Fig 31. Last configuration of the manipulator and target in a simulated environment for position and orientation control task.

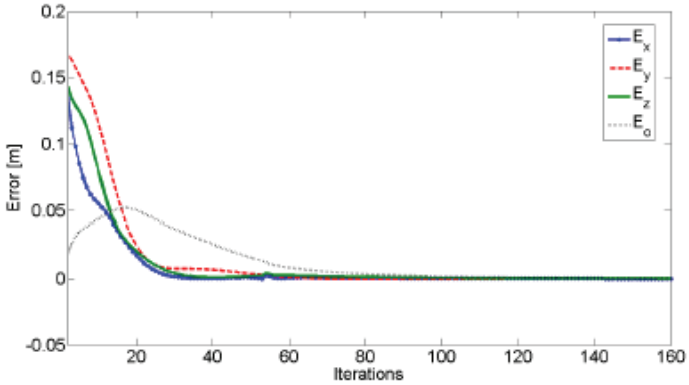


Fig 32. Errors through iterations in position and orientation control with absolute reference (in position) and relative feedback (in orientation): simulation results tested in real robot using simulated joint values as references for real joints.

A first trial a in simple position task, for a given absolute reference $R_d = (0.15, 0.2, 0.2)$ *m,*
has been realized with $d_S = 0.001$ and $K_S = 200$.

It must be noticed that reference is given in terms of absolute vector components.

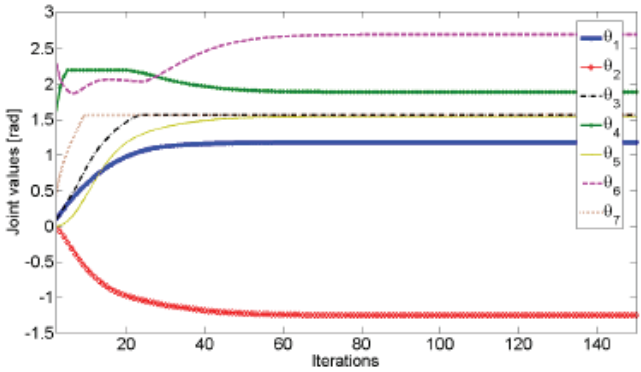


Fig 33. Joint values during iterations in position with absolute reference trial: simulations outcomes used as references for real robot joints positioning.

As shown in Fig 28, after less than 80 iterations the network converges toward the desired position and the residual error is $||e_P|| < 0{:}001$ *m.*
The final configuration of the manipulator is depicted in Fig 30.
In Fig 33 it is possible to see that, due to joint angular constraints (Table II), introduced in the NLB, the angular position of each servo is constrained. It is also important to notice that, as described in (Cruse, 1993), the performance of this approach outside the operating space are still very good: the manipulator reaches the minimum distance point inside the operating space.

End-effector positioning with relative error feedback
The same experimental setup has been kept to test the control scheme with relative error feedback instead of the absolute reference. In this trial the same d_S and K_S were used, while for feedback error a $K_P = 0.2$ was chosen in equation (30). Up to now the end-effector position error measurement is just estimated using, even in real robot implementation, the forward kinematic model.

The simulation results show a significant increase in the number of iterations needed to reach the same desired position.
In fact the same position error value is reached after 600 iterations

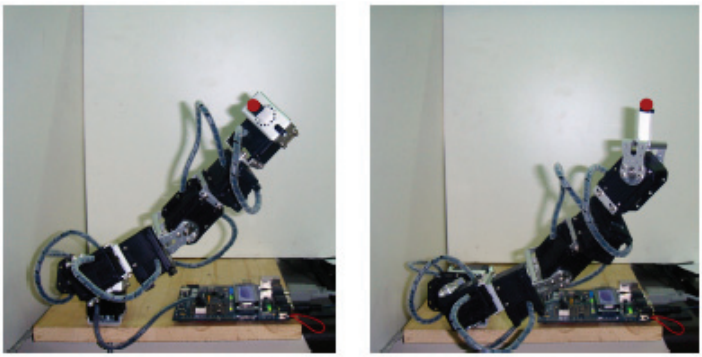


Fig 34. Configuration of the real robot and of the virtual target (circle) after the position control of the end-effector (on the left) and after the position and orientation control of the end-effector (on the right).

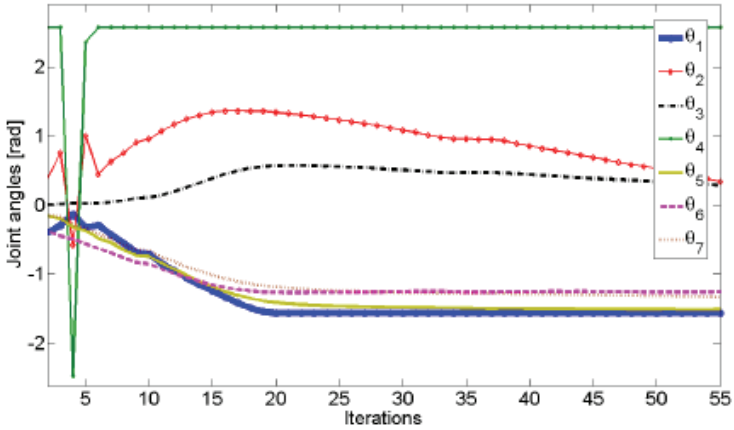


Fig 35. Joint encoder readings trough iterations in real robot simple position trial. Angular values are acquired in encoder values (10 bit resolution) within servo constraints.

(Fig 29). Further simulations show that quantitative results for the number of iterations needed for a complete convergence (*i.e.* within a defined error tolerance) depends on the K_P value and therefore an optimal speed-accuracy trade-off should be chosen (*e.g.* with $K_P = 0.7$ the same error value is reached after only 100 iterations

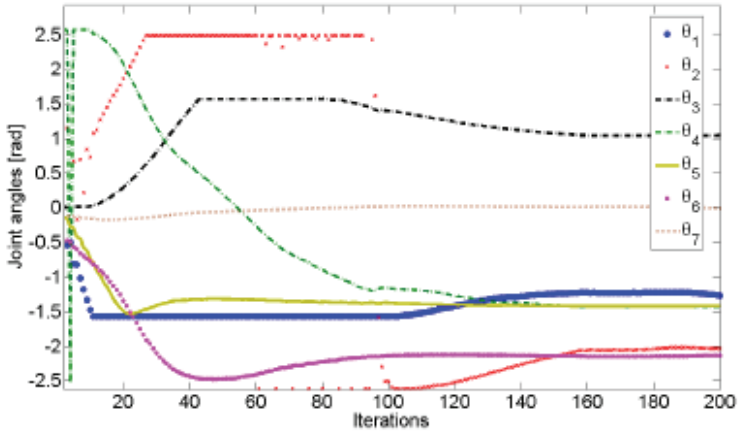


Fig 36. Joint encoder readings trough iterations in real robot trial with external disturbance. Angular values are acquired in encoder values (10 bit resolution) within servo constraints.

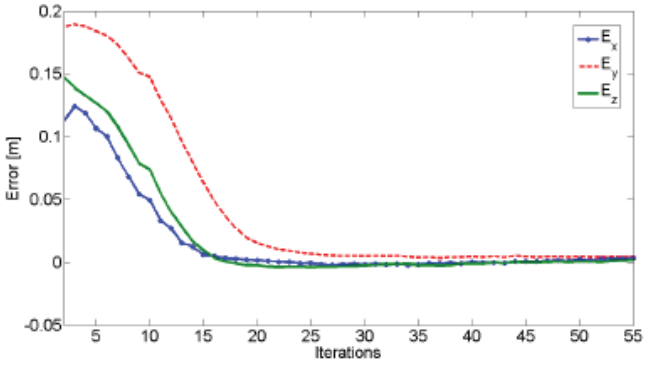


Fig 37. Estimated position error of the end-effector of the real robot trough the iterations in position control task.

without overshoot).

End-effector position and orientation control

In order to test performances of the complete model in an orientation constrained task, a $R_d = (0.15, 0.2, 0.2)\ m$ together with a desired $L_d = (0, 0, 0.05)\ m$ for the last link were given. As shown in Fig 32, it

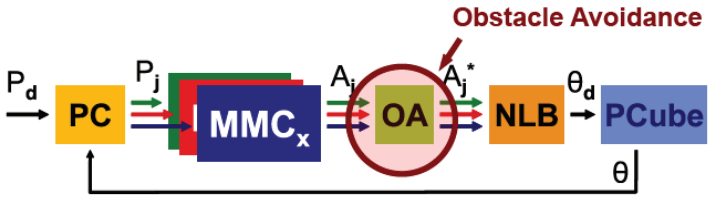


Fig 38. Modified MMC block diagram for Obstacle Avoidance (OA). Implemented for Power Cube (PCube) robotic structure (from Schunk)

results that, for $K_O = 0.6$ (equation (33)), after just 90 iterations the network is completely relaxed and the errors are $||e_P|| < 0.001\ m$ and $||e_O|| < 0.001\ m$ (where e_O is the last link relative error vector). The final configuration of the manipulator is depicted in Fig 31.
Position control and position-orientation control tasks have been performed on the real robot (Fig 34), without changing parameters, introducing the encoder readings as angular feedback values and estimating the errors trough iterations with forward kinematics (*e.g.* position task shown in Fig 35 and in Fig 36). Error tolerance (e_t) has been increased from $e_t = 0.001\ m$ to $e_t = 0.005\ m$ due to mechanical constraints and to the accuracy given by the low cost components of the experimental setup.

The mechanical finite acceleration and the finite speed of the real robot motion introduce a natural damping factor whose value depends on algorithmic time needs related to the joint speed.

Behaviour in front of dynamically changing constraints
The introduction of real angular feedback gives to the introduced algorithm an interesting added value, especially in dynamically changing environment. In fact, let us consider the case in which an external disturbance is added, for example, to the first degree of freedom of the manipulator .
The behavior is shown in Fig 36, where the same input reference and algorithm parameters as for the previous simulation (Fig 29) were used.
The external disturbance could be due to the effect of a moving obstacle, or simply by a mechanical fault.
This implies imposing a new, unforeseen constraint to the joint posi-

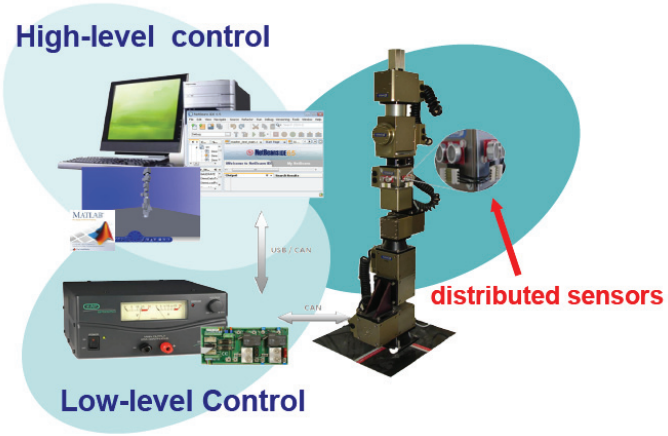


Fig 39. Hardware block diagram for Power Cube (PCube, from Schunk) high level control from a PC.

tion. The corresponding error feedback modifies the network behaviour, but the network nevertheless is able to relax toward a minimal position error, whose value depends on the entity of the constraint imposed, that could bring the target even outside the new operative space. This can be appreciated in Fig 36, where θ_1 is blocked by physically preventing its motion for about a half of the number of iterations.
It is also possible to appreciate that the joint driven by θ_6, though it shows some chattering, is able to overcome the angular limitation and the encoder dead zone. Videos of the described experiments together with high resolution pictures of the figures are available on the web (SPARK II website).

Moreover another explored possibility is to implement an Obstacle Avoidance block (namely OA) in order to produce a repulsive force in the near filed of each presented obstacle (Fig 38). From an hardware point of view, this can be simply realized adding a small network of range sensor (*e.g.* sonar) within the links of the manipulator (Fig 39)

Method	Iterations for convergence
MMC_{abs}	45
MMC_{rpf}	75
MMC_{nvd}	156
J^t	105
J^*	62

TABLE III Algorithm performance in position control task

Result analysis

Due to the lack of strong analytical proof of convergence and stability of the introduced method an extensive statistical simulation campaign has been performed.

A sampled version (with number of points *N = 107*) of the workspace of the described structure has been computed and the histogram has been analyzed.

Moreover random reachable targets have been selected from the sampled workspace and used to test the algorithm. Starting from the same initial condition $\vec{0} = [\ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ ]$ using a variable damping (function of the position error) and with a maximum number of iteration $i_{max} = 3 \cdot 10^3$ the algorithm converged in position task with joint limits with probability $p = 0.877$ (estimated in N = 105 trials).

Although the iteration number needed to reach the desired reference strongly depends on the chosen parameters and on the particular given reference, in order to estimate the algorithm performance, multiple comparisons with common algorithms for kinematic inversion, such as pseudo-inverse Jacobian (J) and Jacobian transpose (J^t), have been considered. It must be taken into account that in the J the joint limit constraints are introduced in form of an additional task to be completed while in the J^t no joint limitations are considered.

Free parameters were chosen to maximize convergence speed keeping a non-overshoot condition on end-effector positioning. As it is possible to see in TABLE III and in TABLE IV the performance in terms of number of iterations needed for convergence (*i.e.*

Method	Iterations for convergence
MMC_{abs}	65
MMC_{rpf}	80
MMC_{nvd}	160
J^t	450
J^*	190

TABLE IV Algorithm performance in position and orientation control task.

Method	Time/iterations [s]
$MMC_{abs}/MMC_{rpf}(PC)$	$3.7 \cdot 10^{-4}$
$MMC_{nvd}(PC)$	$2.1 \cdot 10^{-4}$
$MMC_{abs}(MCU)$	$1.8 \cdot 10^{-3}$
$J^t(PC)$	$4.1 \cdot 10^{-3}$
$J^*(PC)$	$4.6 \cdot 10^{-3}$

TABLE V Algorithm timing for different implementation on different platforms

$\| e_P \| \leq e_T$) are comparable with already known algorithms for the MMC model with relative position feedback (MMC_{rpf}) and for the MMC with a non-variable damping (MMC_{nvd}) while they are even lower for MMC with absolute position reference (MMC_{abs}) and relative orientation feedback (in orientation control task).

Thanks to its low computational effort, the proposed model architecture permits a simple and fast hardware implementation. TABLE V gives an idea of time needed for each iteration. For a complete comparison among all the different implementations of the proposed algorithm we have to consider that the MMC_{rpf} and the MMC with relative orientation feedback are almost the same as the MMCabs in terms of computational effort while the MMC_{nvd} is significantly faster because it is possible to compute the weight matrix M once for all the iterations needed even if more iterations are requested for convergence.

In this chapter, a fast, reliable algorithm able to solve at the same time both position and orientation tasks is proposed. Its implementation on an embedded mcu-based board has been tested and a real

robot application has been developed.
The convergence of the MMC network has been robustly extended in a position and orientation task inside a threedimensional operating space. Beyond the state of the art (Jacobian-based algorithms (Sciavicco ed., 2008) and other cited analytical approaches (Moradi, 2005), (Tondu, 2006)), though both problems are solved in a sub-optimal way the real-time implementation addresses the problem not only in presence of angle limits but also in a dynamically changing environment even in a mcu-based hardware. That results together with sensors readings have been investigated in order to effectively modify the trajectory of the manipulator toward a constrained optimum configuration. Further development should include a multi-legged robot application and a moving target followed trough visual feedback.

chapter 3

perceiving the world:
action oriented perception

The gather and the analysis of the distributed sensor information present in the body of a living beings is for sure the most important (and the most interesting) part of these studies and of this project.

In this very wide and complex field the most challenging part is how to address the dimensionality problem.
The number of dimension of the sensor information of a complex structure is quite huge. Whatever is the problem to solve and however you think to use this sensor information the dimension of the sensor space will be higher than necessary.

With the local Principal Components it is possible to represent sensor distributions locally constrained to sub-spaces with fewer dimen-

sions than the space of the sensed data. Under this point of view, as described in (Martinez, 1993), PCA is able to model distributions in those directions with almost zero variance exist.
In robotic applications data classification is extremely important and as far as the visual input is considered, the number of features and the complexity of in-class discrimination increase enormously. For these reason it is important to identify algorithm for classification that show characteristics like robustness.
The Neural Gas (NG) (Martinez, 1993) is a vector quantization technique that try to approximate a given manifold with a reduced number of points. With Principal Component Analysis (PCA) each of this data-rapresentative points (code-book vectors or particles) is extended into hyper-ellipsoid to better match the given data distribution. This technique has been applied in different fields as redundant inverse and forward kinematic (IK and FK) for serial manipulator and roving robot path planning (Martinez, 1993). Our purpose is to further develop a fully unsupervised classifier for real robotic applications. Under this point of view our algorithm has been compared, from the beginning, with Self-Organizing Map (SOM) (Kohonen, 1995) approach which is at the same flexible and simple.
As a key point for a real time application the data set is acquired during the experiment and not completely available at the beginning. Under this point of view, it is important to envisage a growing mechanism that allows the algorithm to cope with sample collection typical of a roving agent in unknown environment (*i.e.* dynamically changing training set).
The model architecture can be easily divided in a learning phase, in which the data distribution is approximated and in a recall phase in which an incomplete pattern is presented to the network in order to retrieve the output (complete pattern). As already discussed (Cruse, 1993 and Arena, 2009), similarly to Recurrent Neural Networks (RNNs), this model is able to cope with multiple solution tasks providing one of the possible solutions and it is also possible to choose the role of input and output neurons, after the training, simply modifying the recall phase.
A supervised classification extension of the algorithm, with visual cues variables in the input pattern portion, will be discussed.

Moreover an incremental on-line version of the architecture gives a action-oriented classification capability for roving robot applications (SPARK II project).

Model description

The Neural Gas algorithm is a variant of the soft-clustering vector quantization with the addiction of annealing. For a given pattern space $P \subseteq \Re^d$, the algorithm starts choosing m points c_j (with *j = 1,2,...,m*), in this hyperspace, from the N training set elements. During each step, a random pattern x is chosen from the training set. Then each j-point position c_j is updated for next iteration step relating to its rank r_j function of the distance from the selected pattern trough the learning rate " and the neighborhood range ρ as follows:

$$\mathbf{c_j}(t+1) = \mathbf{c_j}(t) + \alpha_j \cdot [\mathbf{x} - \mathbf{c_j}(t)], \qquad (58)$$

where α_j is defined by $\alpha_j = \varepsilon \cdot e^{-r_j/\varrho}$.

In order to force algorithm convergence through iterations both ϵ and ρ exponentially decrease from ϵ_{init} (ρ_{init}) to ϵ_f (ρ_f) as in the following.

$$\begin{aligned} \varepsilon(i) &= \varepsilon_{init}\left(\frac{\varepsilon_f}{\varepsilon_{init}}\right)^{\frac{t_i}{T}} \quad with \quad \varepsilon_{init} \geq \varepsilon_f \\ \varrho(i) &= \varrho_{init}\left(\frac{\varrho_f}{\varrho_{init}}\right)^{\frac{t_i}{T}} \quad with \quad \varrho_{init} \geq \varrho_f \end{aligned} \qquad (59)$$

where t_i is the current iteration number and T is the lat iteration number.

The local PCA extension of the NG considers hyper-ellipsoid units, with q principal components, instead of simple points and therefore the ranking of the units cannot depend on an Euclidean distance. One of the possible distance measure is the normalized Mahalanobis distance (Hoffmann, 2003) (Hinton, 1997), an elliptical distance that can be computed, for each j particle, as:

$$\begin{aligned} E(x) = \xi^T W \Lambda^{-1} W^T \xi + \tfrac{1}{\sigma^2}(\xi^T\xi - \xi^T W W^T \xi) + \\ + ln(det\Lambda) + (d-q) ln\sigma^2, \end{aligned} \qquad (60)$$

where $\xi = x - c$ is the deviation of vector x from the centre unit, W is the eigenvector matrix, Λ is a diagonal matrix containing the eigen-

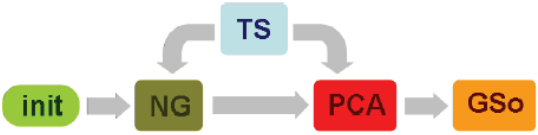


Fig 40. Algorithm block diagram for the learning phase of the Neural Gas with local Principal Component Analysis (NGPCA). The NG block performs centre updating for a given training vector, extracted by the Training Selector (TS), the PCA block executes one step of local principal components algorithm and GSo (Graham-Schmidt ortho-gonalization) is able to give eigenvectors orthogonality property.

values. The second term of equation (60) is the reconstruction error divided by 2 that depends on the total residual variance v_{res}, among all d-q minor dimensions ($d \geq q$), as in equation (61).

$$\sigma^2 = \frac{v_{res}}{d-q}. \qquad (61)$$

The total residual variance is updated according to

$$v_{res}(t+1) = v_{res}(t) + \\ + \alpha \cdot (\xi^T W \Lambda^{-1} W^T \xi - \xi^T W W^T \xi - v_{res}(t)). \qquad (62)$$

To modify principal components of existing ellipsoids, one step of the Robust Recursive Least Square Algorithm (RRLSA) (Moller, 2002), described in (Ouyang, 2000) is performed.

$$\{W(t+1), \Lambda(t+1)\} = \\ \mathbf{PCA}\{W(t), \Lambda(t), \xi(t), \alpha(t)\} \qquad (63)$$

Since the orthogonality of W is not preserved after each step, the Gram-Schmidt orthogonalization method has been introduced (Hoffmann, 2003).

The algorithm overall block diagram is shown in Fig 40 where the already introduced main blocks are: an initialization block (init) that initialize all learning parameters and variables; a Neural Gas block (NG) that update the position of the centers of the ellipsoids for a given pattern x chosen from the Training Selector (TS) from the training set T = T_1,T_2, ..., T_N as in equation (58); a Principal Component Analysis block (PCA) that update hyper-ellipsoids axes with

m	number of particles
N	data set dimension
d	pattern space dimension
q	number of principal components
$\mathbf{c}_j$	center of particle j
r_j	ranking of particle j
v_{res}	total residual variance
Λ_j	eigenvalues matrix of particle j
W_j	eigenvectors matrix of particle j

TABLE VI Notation summary for Neural Gas with Principal component analysis

RRLSA as in equation (63) and a orthogonalization block (GSo) that performs the so called Gram-Schmidt algorithm.

Relevant parameters and variables used in the model are summarized in table VI.

In order to reduce the dependence of the error in equation (60) from the volume of the considered ellipsoid and to avoid useless points (with weight almost zero) in the pattern space it is possible, as in [1], to modify the distance measure as follows:

$$\widetilde{E}(x) = (\xi^T W \Lambda^{-1} W^T \xi + \\ + \tfrac{1}{\sigma^2}(\xi^T\xi - \xi^T W W^T \xi))V^{2/d} \qquad (64)$$

where V is the volume of the considered ellipsoid unit and can be computed according to

$$V = \sigma^{d-q}\sqrt{det\Lambda}. \qquad (65)$$

Test phase

After the learning phase, the data distribution is represented by m hyper-ellipsoids with center c_j (with j = *1,2, ..., m*), semi-axes lengths $\sqrt{\lambda_j^k}$ (with *k = 1,2,...,q*), w_j^k principal component eigenvectors and a residual variance σ_j^2 .

In the recall phase and incomplete pattern $p^* \subseteq \mathbb{R}^{d-s}$, with *s* number of laking dimensions ($s \leq d$), is given as input and the algorithm would rebuild the *s* free dimensions and give the optimum complete pattern $\hat{z}_{j^*}$ as output.

In order to perform the recall, a potential function in the constrained subspace $E_i(z)$ can be computed for each ellipsoid as:

$$E_j(\mathbf{z}) = \mathbf{y}_j^T \Lambda_j^{-1} \mathbf{y}_j + \frac{1}{\sigma_j^2}(\xi_j^T \xi_j - \mathbf{y}_j^T \mathbf{y}_j) + \\ + ln(det\Lambda_j) + (d-q) ln\sigma_j^2, \quad (66)$$

where ξ_j is the displacement from the center $\xi_j = z - c_j$ and $y_j = W_j^T \xi_{j_1}$ is the representation in the local coordinate system of the ellipsoid.
The input to the network is given in form of an offset p in the constrained space $\mathbf{z}(\eta) \subseteq \Re^d$ as follows:

$$\mathbf{z}(\eta) = \mathbf{M}\eta + \mathbf{p}, \quad (67)$$

where M matrix aligns the constrained space to a particular parameters space while $\eta \in \mathbb{R}^s$ is a vector of free parameters. For each unit j, the point of constrained space with smallest potential bzj is determined, according to equation (67), and then the unit j^* that has the minimal potential among all $E_j(\hat{z}_j)$ is chosen as complete pattern. As shown in (Hoffmann, 2003), the function $E(\eta)$ is convex and it is possible to determine analytically the only minimum $\hat{\eta}_j$ computing:

$$\hat{\eta}_j = \mathbf{A}_j(\mathbf{p} - \mathbf{c}_j), \quad (68)$$

with

$$\mathbf{A}_j = -(\mathbf{M}^T D_j \mathbf{M})^{-1} \mathbf{M}^T \mathbf{D}_j, \\ \mathbf{D}_j = \mathbf{W}_j \mathbf{\Lambda}_j^{-1} \mathbf{W}_j^T + \frac{1}{\sigma_j^2}(\mathbf{I} - \mathbf{W}_j \mathbf{W}_j^T). \quad (69)$$

Classification problem
A simple supervised classification problem is one of the straightforward application of this kind of approach. Moreover the pattern is divided, as in equation in a feature hyperspace f and a class parameter n.

$$\mathbf{x} = [\mathbf{f}, n]. \quad (70)$$

During the training phase the complete pattern is presented as input to the algorithm. In the testing phase the class number portion of the vector is left free and is retrieved as output.

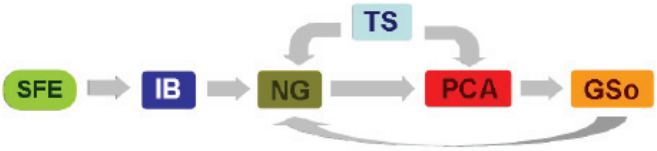


Fig 41. Algorithm block diagram for on-line incremental learning that uses pre-processed input vectors. The Segmentation and Feature Extraction block (SFE) performs a frame segmentation and extracts the feature vector f . The Incremental Block (IB) regulates the increase of the number of gas particles used in the classification.

Patterns

Simple visual cues have been chosen to build features part f of the input training pattern. Both a space-color representation and geometrical features extraction have been performed in a segmented image. The most present hue, saturation and value (respectively c_H, c_S and c_V) and some geometrical features *p, A,* e_m and e_M (respectively the perimeter, the area, the shorter and the longer edge of the minimum rectangle that containing the object) have been extracted from an image portion obtained with a Canny segmentation algorithm.

$$\mathbf{f} = [c_H, c_S, c_V, p, A, e_m, e_M], \qquad (71)$$

As often happens in real-time robotics, in our application, the training data-set is not fully available at the beginning of the learning: we suppose the robot to collect training images (*i.e.* complete training patterns) through its mission (*i.e.* iteration after iteration) and at the same time try to correctly classify them for better foraging task-oriented performances.

Incremental Learning

In order to achieve incremental learning several problems needs to be overcame with some modification to the presented algorithm. First of all the number of code-book vectors should be variable. Then the annealing of each ellipsoid (*i.e.* the capability to move) must be function of its own age (*i.e.* number of steps it is present of the training). The on-line supervised nature of the algorithm needs both recall and training phase to be achieved per each iteration. A

block diagram of the modified algorithm is shown in Fig 41.

When a new input is presented *(i.e.* a vector of features f with a class *n* from the Segmentation and Feature Extraction (SFE) pre-processing block) the Incremental Block (IB) performs a test phase and then check the distances e E(x) from for all the ellipsoids already instantiated.
Therefore when a pattern is chosen for learning purposes if inequality (72) is false a new instance of code-books is made. Moreover its ellipsoid is initialized and its Mahalanobis distance from that patter is set to zero.

$$max(\widetilde{E}(x)) \leq E_{th} \qquad (72)$$

The time variable is then no more an global iteration counter but becomes a variable of each particle. Therefore the ellipsoid *j* at iteration *i* would have age $a_j(i)$ and its specific learning parameter are ξ_j and ρ_j as in following equations.

$$\begin{aligned} \varepsilon_j(i) &= \varepsilon_{init}\left(\frac{\varepsilon_f}{\varepsilon_{init}}\right)^{\frac{a_{ji}}{a_{max}}} \quad with \quad \varepsilon_{init} \geq \varepsilon_f \\ \varrho_j(i) &= \varrho_{init}\left(\frac{\varrho_f}{\varrho_{init}}\right)^{\frac{a_{ji}}{a_{max}}} \quad with \quad \varrho_{init} \geq \varrho_f \end{aligned} \qquad (73)$$

where a_{max} is the maximum ellipsoid age.
This mechanism allow to selectively force the convergence of each code-book ellipsoid and therefore it is possible to cope with a dynamically changing training set.

Pruning
One of the problem introduced with the incremental learning mechanism is the possible growing of the number of particles that can appear during the learning time but that cannot In order to achieve a fast convergence maintaining the forgetting capability and performances increasing the over long time simulation, a pruning strategy has been implemented. Thanks to the supervised classification process each time one particle is selected as the nearest to the presented input pattern, its distances is under a given threshold, a

Fig 42. Image of all laboratory tools used to create the input data set. Image has been acquired at VGA resolution *(640px x 480px)*

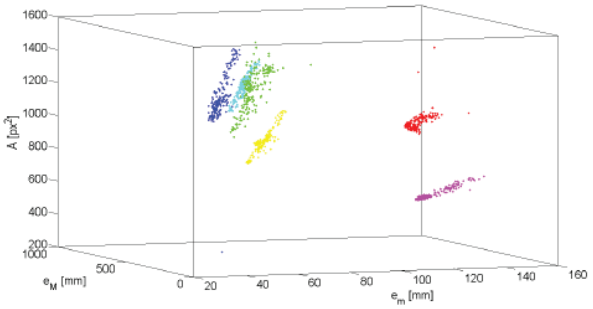


Fig 43. Reduced feature-space obtained after segmentation. Depicted dimensions have been reduced from six to three (*i.e.* e_m, e_M and A). Each grey-scale value stands for a different class. It can be noticed that patterns from all three scree-drivers have reduced Euclidean distance in this feature sub-space (on the left).

recall phases is performed and the resulting class is compared with the real one.

Considering that the class number is a discrete variable while algorithmic outcomes are continuous, a classification error means that the distance between reconstructed class number n_z and test pattern class number n_t exceed a user defined threshold ($e_{th} \geq 0$)

$$|n_z - n_t| \geq e_{th} \qquad (74)$$

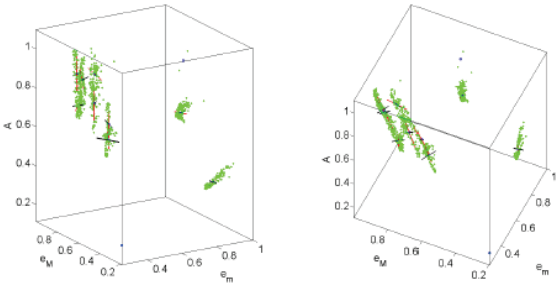


Fig 44. Two different views of the reduced normalized features space (e_m, e_M and A) together with code-book ellipsoids after learning.

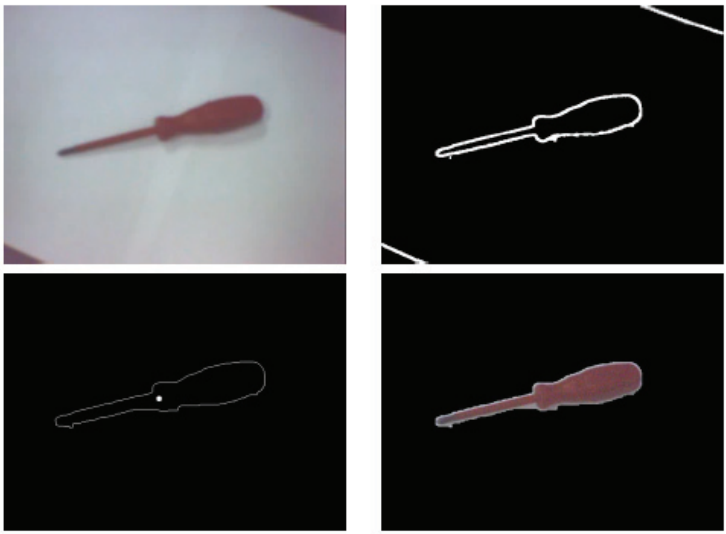


Fig 45. Flow of the segmentation process. First a gaussian filter is applied (top-left), then the Canny algorithm generates a binary image (top-right) and finally meaningful contours are isolated (bottom-left). Segmented object area is determined and the same portion of the original image is extracted for color space based statistical analysis (bottom-right).

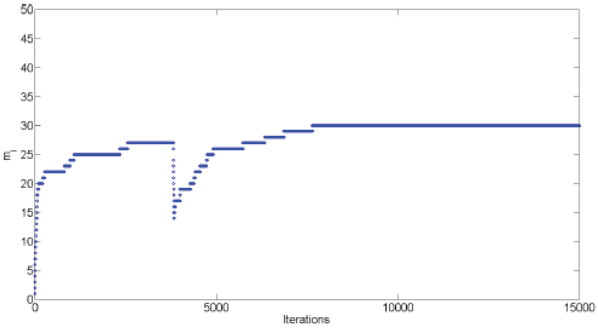


Fig 46. Number of code-book ellipsoids (m) through learning with respect to the time (algorithm iteration number). The number of particles decrease for a while, around iteration 3500, due to the pruning strategy that allows each j particle to be erased when $a_j \geq a_{max}$.

In this way per each particle *j*, at iteration *i*, it is possible to compute an error rate $e_j(i)$ as it follows:

$$e_j(i) = \frac{n_{err}(i)}{n_{wins}(i)}, \qquad (75)$$

where n_{err} is the number of times the particle is the nearest and recalls phase leads to a classification error and $n_{wins,j}$ is the total number of times it is the nearest.
After code-book *j* has expired its life (*i.e.* $a_j \geq a_{max}$) if its error rate (e_j) is over a tolerance threshold (r_{th}) the corresponding particle be erased. In other case it is held freezed with minimum learning parameter (*i.e.* $\xi = \xi_f$ and $\rho = \rho_f$).

Experimental setup
Experimental images have been acquired to test algorithm performances. In order to consider a realistic visual flow on autonomous robot both resolution and overall quality of the images have been kept low. Testing objects have been chosen from common laboratory tools, see an example if Fig 42.

Visual cues have been extracted with OpenCV library (OpenCV website) and a custom made simple segmentation algorithm through

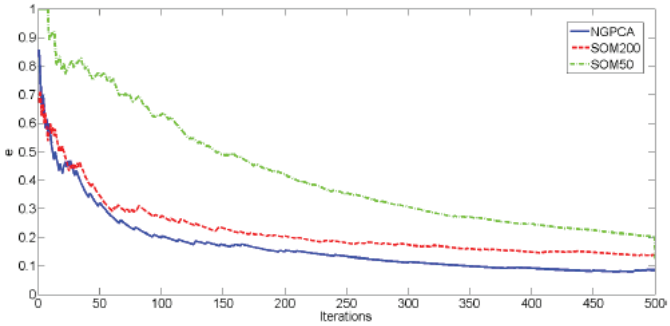


Fig 47. Error indexes e_T comparison between implemented NGPCA (solid line) and Self-Organizing Map approaches with 50 by 50 neurons (dashedline) and 200 by 200 neurons (dotted-lint) through learning iteration.

a host PC with low-resolution webcam (*640px x 480px*). It must be remarked that considering a frame rate of *30fps*, a that resolution, simple segmentation routine can be easily performed in less than t_s = 1/60 *s* in modern PC leaving at least t_l = 1/60 *s* for classification learning. A reduced feature space of the outcomes of the segmentation process is shown in Fig 45 with one gray-scale value per each class.

Experimental results

The input data set T = {i_1, i_2, ..., i_N} has been built of N = 1300 different images in which one of the object belongs to one class was present. Six different classes were chosen.

During a testing phase, at the iteration i, the considered performance index eT is the sum of each particle error rate e_j.

$$e_T(i) = \sum_{j=1}^{m_i} e_j(i) \qquad (76)$$

In order to be comparable even in supervised and semi-supervised problem solving the classic SOM has been trained with a discrete class parameter and a test phase has been introduced at each iteration.

For a given pattern x, the Best Matching Unit j* (BMU, *i.e.* the near-

est unit under Euclidian metric) is chosen just on distance on the features part of the x vector, f as follows.

$$j^* = argmin_j(\mathbf{f}_j - \mathbf{f}_x) with j = 1, 2, ..., m_S, \quad (77)$$

with ms number of neurons in map. The classification error is then evaluated on the class part of the vector (*i.e. n* parameter) between the BMU and the presented pattern as in inequality (74).

For a $\xi_{init} = 0.5$, $\xi_f = 0.02 \cdot \xi_{init}$, $\rho_{init} = 0.01$, $\rho_f = 0.02 \cdot \rho_{init}$ and a_{max} = 3000, if we consider each frames to have a meaningful different pattern configuration, with an error $e_T \leq 0.07$ in less than 500 iteration with an average computational time of $t_i = 5 \cdot 10^{-3}$ (on a laptop Core 2 Duo 2.2 *GHz* with 2GB of RAM).
As briefly introduced in previous sections starting from a number of ellipsoid m(0) = 0 as initial condition, m(i) increases through iteration as shown in Fig 46.

As it is possible to see in Fig 46, starting from N = 1300 input patterns the number of code-book ellipsoids is far less (maximum reached value m = 30). Therefore considering one learning algorithm iteration at each acquired frame with an image frame rate of *30fps*, an elaboration time t_e 28x10^{-3} s can be used for segmentation and other control algorithm.

Moreover same parameters lead toward a e_T<0.015 in 2500 iterations (t = 12.5s in our hardware setup) and e_T<0.005 in 15000 iterations (*i.e.* t = 75s) as summarized in Table II. SOM with a relatively large number of neurons (*i.e.* S50 with 5050 neurons and S20 with 200200) have been tested. It is possible to observe that, in our simulations, larger is the number of neurons of the map smaller is the overall classification. SOM with 40000 neurons performs largely worse than NGPCA. The complexity of the NGPCA algorithm is O(Nq) with N number of code-book ellipsoid and q is the number of principal components.

In this chapter an on-line classification application of the Neural Gas with Principal Component Analysis is presented.

i	m_i	t [s]	e_T [%]	e_{S50} [%]	e_{S200} [%]
500	22	2.5	7	21	14
2500	26	12.5	1.5	14.5	7
15000	30	75	0.5	11	2

TABLE VI Performance analysis table

Both the algorithm and the whole experimental setup have been chosen for straightforward porting of the architecture to a PC-based object manipulation in a roving platform. Under this point of view, the error-to-performances trade-off of the learning process is selectable through parameter tuning and the overall computational cost of the algorithm can be kept low, even for a embedded version of the classifier.

The model input-output variables can be changed at any time for a given learning. A first comparison with standard structures, like Self-Organizing Maps, shows a higher level of flexibility without error nor complexity increase. Nevertheless, for a complete classification capability, focused on robot action-perception closed loop application, the proposed algorithm should be further developed in a fully unsupervised classifier.

chapter 4

algorithmic solutions for actions and more: *sequence learning toward given objectives*

As already introduced, actions toward a given objective can be decomposed in multiple ways, at multiple levels of abstraction.
From a behavioural point of view (*i.e.* a high level of abstraction) and considering as a atomic action the complex motor coordination for motion of a multi-link structure (as almost any living beings limb) from one point to another, several considerations can be done on motion sequences.
Nevertheless the complexity of this trivial coordination can be very hard and not so trivial. Consider now, for instance, the quite simple task of the pendulum swing up.

It is clearly easy and straightforward to imagine the solution of the problem at a joint level (and eventually to derive equations for the

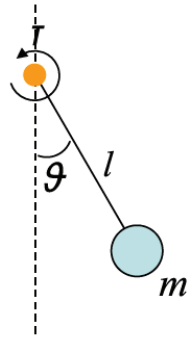
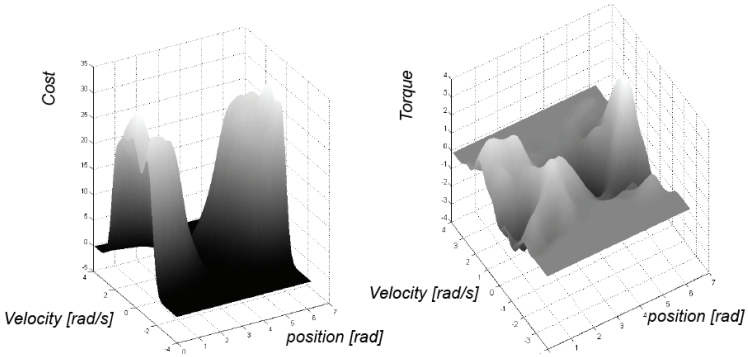


Fig 49. Very simple pendulum model used to demonstrate situated controller learning. In particular the considered task was a torque limited pendulum and a spiking Self-Organizing Map was used to learn both the optimal policy and the estimated cost function.



$\hat{J}_v(x)$ *estimated cost function* $\pi_w(x)$ *learned policy*

Fig 48. Torque limited pendulum swing-up. Example of a learned policy (on the right) and estimated cost function (on the left), with the help of a multi-layer spiking Self-Organized Map (SOM).

control of an actuated structure).
It can be still easy to imagine the solution of the problem in terms of agonist and antagonist muscles that can concurrently control the joint. It not so simple and not so straightforward to imagine (and eventually to derive equation) of this pendulum swing-up in case of limited torque.

Moreover, as it is possible to imagine, problem can be even harder if

you want to start from a general purpose structure (like a Motor Map or a more general Self-Organizing Map) and want to learn a poilicy and estimate the cost function if every point of the domain (*i.e.* two dimensional space made up of angular joint position and angular joint velocity).

For all this difficulties and for many others not even cited, let us consider, it this chapter the synthetic problem of sequences of actions and let us face with this problem employing a Reinforcement learning approach..

Reinforcement Learning (RL) is the straightforward learning paradigm for bio-inspired architectures. In the last few decades the state of the art for robot learning has moved towards RL (Lee, 2006) (Braga, 2003). Unfortunately in robotics the common trial-and-error practice is not so trivial: the lowcost mechanical structures have very different compliance and timing capabilities with respect to their biological counterpart. A large number of different approaches have been proposed to overcome this kind of problems (Bakker, 2006) (Abbeel, 2008). Generally speaking, the underling idea is to use computational power to minimize necessity of real environment interaction.

Eligibility traces, model based approaches (like Prioritized Sweeping in Moore, 1993) and probabilistic methods (like Ant-Colony Optimization in Dorigo, 2004) increase computational costs in order to reduce the number of environment examples needed (Kaelbling, 1996) (Barto, 2003). A commonly used strategy is to perform multiple iterations based on past observations between two real experiences (Sutton, 1998).

In this chapter an application of a modified version of a simple State-Action-Reward-State-Action (SARSA) algorithm is presented in order to cope with a discrete shortest path problem with a redundant serial manipulator.
The task taken into consideration includes a robotic arm equipped with a pointer as end-effector. The robot, hereafter referred to as the

agent, should learn which is the best way to position the end-effector on all the black squares in a given custom checkerboard. Multiple levels have been used to hierarchically control the hardware. A high level, behavioral, control is first performed in a host computer. The algorithm output (*i.e*. a discrete checkerboard position) is then given as input to a kinematic inversion algorithm able to cope with the redundant serial structure. A low level control is then performed both in a custom designed control board and in the distributed control system of the robot.

As in any other Q-learning based algorithms, for a given state and a policy, the next action is chosen, reward is evaluated, and therefore the action-value (Q-function) for state-action pair is updated iteratively. The particular reward function determines the overall behaviour of the agent.
In contrast to what previously discussed, for a given state the most suitable action (*i.e.* the one that leads to the estimated best next state) is simulated and through the uncomplete model the estimated reward is assigned. Furthermore the learning process is strongly accelerated with action simulation performed by the agent based on what previously learnt: starting from a particular state the agent explores multiple parallel simulated trials and respectively evaluates rewards creating a tree of all chosen possibilities. It is clear that the problem is better achieved via n-step prediction algorithms(Sutton, 1998).
Eligibility traces have been used in order to further keep trace of meaningful state-action pairs.

The system performances have been evaluated using the Ant-Colony Optimization meta-heuristic method (Dorigo, 2004), that is one of the most suitable approaches for this kind of problem. In the first section, the considered general SARSA model is introduced. In the second section, an improvement with a model-based prediction is discussed. Finally, in the other two sections the experimental architecture is presented and the experimental results are shown.

Model description

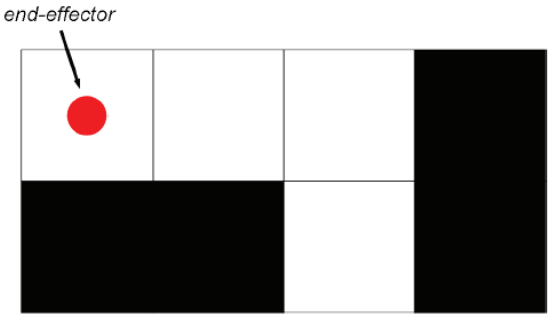


Fig 50. Example of four by two checkerboard together with the end-effector position representation (circle).

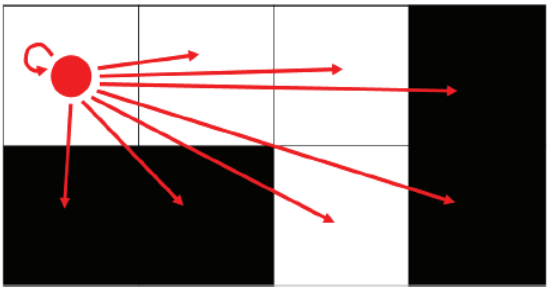


Fig 51. Example of all possible actions (arrows) in a four by two state-space checkerboard by the agent (circle).

The problem of the shortest path is herewith introduced in a unconventional task-driven way. The environment is a modified checkerboard with white and black squares: when the agent is on a black ($s_v = 0$) square, the selected area changes its state to white ($s_v = 1$). The goal is reached when all the squares become white. Both the state and the action spaces are discrete. In particular, state set S includes all the bistable *MxN* checkerboard squares configurations together with agent position for a total of n_s possible states:

$$S = \{s_1, s_2, ..., s_{n_s}\} \qquad \text{with } n_s = M \cdot N \cdot 2^{M \cdot N}. \qquad (35)$$

The actions from action set A can drive the end-effector to one of all possible squares:

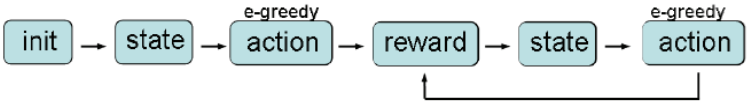


Fig 52. Block diagram of single episode cycle. After an initialization phase implemented in the init block, the (s, a, r, s*, n*) vector is built at each iteration.

$$A = \{a_1, a_2, ..., a_{n_a}\} \qquad \text{with } n_a = M \cdot N. \tag{36}$$

A representation of one possible state from the state-space set S is depicted in Fig 50.
As previously described, for a given state all the possible actions $a \in A$ are those sketched in Fig 51. Therefore considering the actual state *s* and what since now learned, the most rewarding action a is chosen, using Q-learning,with the following method:

$$a^* = \text{argmax}_a(Q(s, a)). \tag{37}$$

In contrast to what happens in Q-learning, shown in equation (37), in SARSA-based algorithms the performed action is not always the one with the highest value in Q-function (for a given state).
For instances, the so called *e-greedy* policy can be followed by the agent as described in the next pseudo-code block.

```
...
if (rand < ε)
then a* ← randint([1, n_A]);
else a* ← argmax_a(Q(s*, a));
endif
...
```

The simplest SARSA block diagram is shown in Fig. 3 where, after parameters and variables initialization performed by the init block, the (s, a, r, s*, n*) vector is collected iteration after iteration. Moreover the action value function Q is updated at each iteration as in the following assignment:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s^*, a^*) - Q(s, a)), \tag{38}$$

where α is the step-size of the learning process, r is the current step

Fig 53. Cost function part for initial state depicted in Fig 50 in the case of movement towards black squares with s_v = 1 (on the left). State-dependent overall cost function used as reward (on the right).

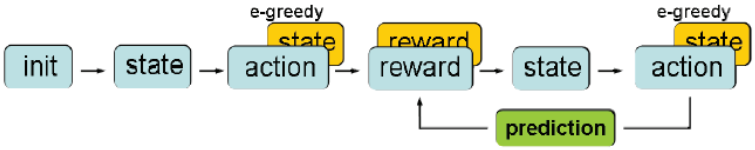


Fig 54. Block diagram of cycle with state-to-state prediction. After an initialization phase implemented in the init block, the (s, a, r, s*,n*) vector is built each iteration, model Q_s is updated and agent prediction is evaluated by the prediction block and darker reward block.

reward and γ is the discount factor while s* is the next step state. Considering that the largest part of the movement time of the real manipulator from one cell to another is not dependent on the distance, though white cells are not a physical constraint, the best path should avoid them. Therefore the problem can be easily split into two sub-problems: the agent should learn to avoid white squares and than learn the shortest path through black squares.
As can be easily understood, the former is not suitable for one-step prediction: too many iterations are needed to obtain low errors (i.e. big deal for real robot implementation).
A. Eligibility Traces In order to reduce the number of iterations despite the increase in computational cost, eligibility traces have been introduced (Sutton, 1998) (Riebeiro, 1999).
When a state-action pair (s, a) is visited, the corresponding value in eligibility function is updated as follows:

$$e(s,a) \quad \leftarrow \quad e(s,a) + 1. \qquad (39)$$

The Q-function update is modified as in the following equations:

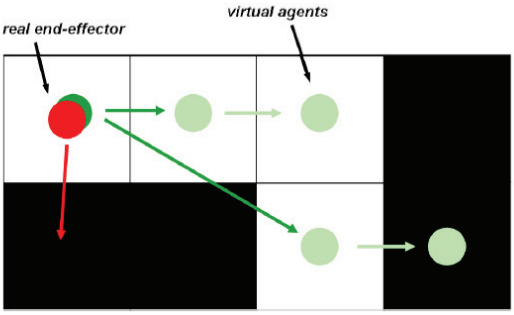


Fig 55. Parallel agent simulation in state predicted tree for a four by two state-space checkerboard. The light arrows and circles indicates possible transitions and states of the virtual agent.

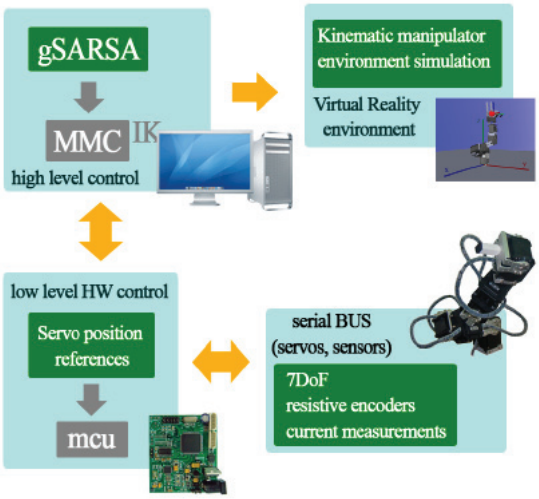


Fig 56. Hardware block diagram of the adopted experimental setup. High level algorithms run on a PC-based platform. Both tri-dimensional kinematic simulation and real hardware control have been implemented. Low level control of the custom manipulator (*i.e.* the Mini-ARM) is achieved, through a USB-to-serial converter, thanks to a 32 bit microcontroller-based custom designed board. Lowest level motor control and angular joint readings are decentralized in each servo.

$$\delta = r + \gamma Q(s^*, a^*) - Q(s, a), \qquad (40)$$

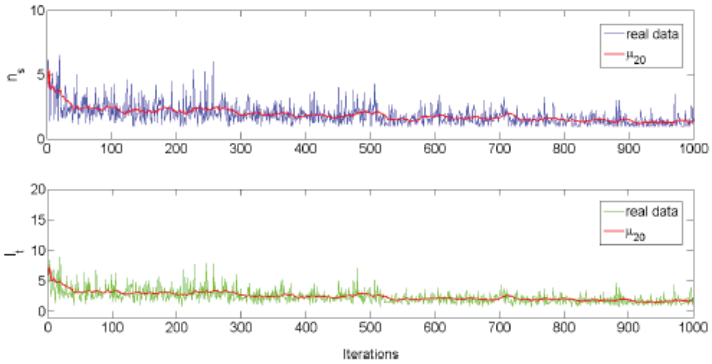


Fig 57. Analyzed indexes in an experimental test. The dashed line indicates the real data and the solid line μ_{20} is the moving average computed with the most recent twenty samples. Both n_s and l_t have been investigated in order to understand different learning capabilities on tasks (*i.e.* avoid white squares and choose shortest black squares path.

and then for all state-action pairs (s, a)

$$\begin{aligned} Q(s,a) &\leftarrow Q(s,a) + \alpha\delta e(s,a), \\ e(s,a) &\leftarrow \lambda e(s,a), \end{aligned} \qquad (41)$$

where λ is the eligibility discount factor and is again the step-size of the learning process. From a computational point of view the straightforward application of eligibility trace implies a single episode problem dimensionality to increase from O(*steps*) to O(*steps x ns*), where steps is the number of action performed in order to reach the goal.
Therefore in the real application a modified version of the algorithm has been implemented to hold information about the (s, a) pairs for which the following inequality is verified:

$$e(s,a) \geq \varepsilon \qquad with \quad \varepsilon \geq 0. \qquad (42)$$

This increases the complexity from O(*steps*) to O(*steps!*) (it must be noticed that typically n_s steps).

Reward cost function
The definition of the reward function is an important aspect of the

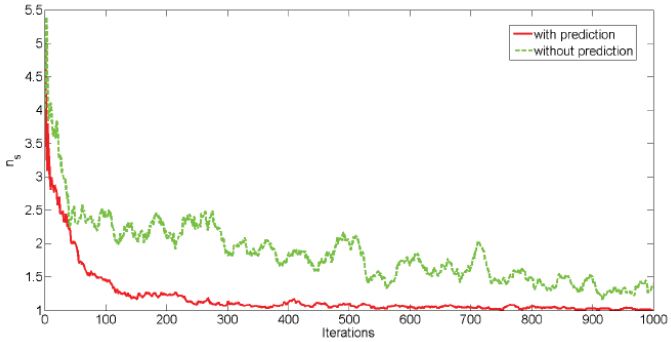


Fig 58. Normalized number of steps needed for task achievement. It must be remarked that when the agent is able to avoid white squares (first task achieved) $n_s = 1$. Dashed line exploits the performance of the first tested algorithm, without prediction. Solid line indicates the predictive model performances.

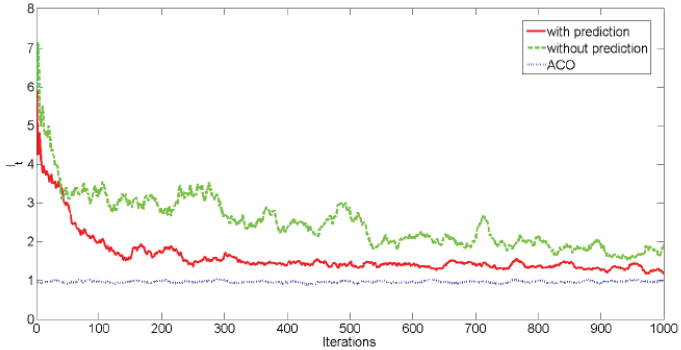


Fig 59. Summary of the experimental results on shortest path task-performance evaluation expressed through the index It that is a function of the over-length of the path traveled. Dashed line exploits the performance of the first tested algorithm, without prediction. Solid line depicts the predictive model performances, while the dotted line are the Ant-Colony Optimization performances.

algorithm in the proposed application. For shortest path solution, the coefficient γ is set to 1, as suggested in (Sutton, 1998), and a negative reward for all non-terminal states is given, while zero reward is assigned for goal achievement.

The cost function used as reward is state-dependent and evaluated through the following system:

$$r = \begin{cases} -(x - a_x)^2 - (y - a_y)^2 & if \quad s_v = 1, \\ -v_b \quad with \quad v_b \geq 0 & if \quad s_v = 0, \end{cases} \qquad (43)$$

where it is clear how all non-terminal states are negatively rewarded according to the squared step length, in order to progressively reduce overall traveled distance. Actions that lead to a white square (i.*e.* in which $s_v = 1$) are negatively biased to further increase convergence speed of the algorithm. An example of the cost function used as reward is shown Fig 53.

Predictive model

As already described, the model has been enhanced to improve the performances in terms of number of episodes needed for a complete learning, in order to meet real hardware timing necessities. The approach consists in reducing the number of real actions needed for learning by introducing a virtual agent that will simulate the outcomes of the environment based on what previously learnt by the real agent.

A state-to-state transition matrix Q_s is built incrementally through iterations based on actual state-action pair (s, a) next state s_n and respective reward r. Therefore the updated block diagram is shown in the diagram in (53), where the agent movement prediction is performed (by the prediction block) and evaluated at each iteration using the vector *(s, a, r, s*, n*)* updating the Q_s function.

$$Q_s(s, s_n) \leftarrow f(s, a, r, s_n), \qquad (44)$$

where f is a reinforcement-based state-to-state function. An example of parallel agent motion simulation that forms a state predicted tree is shown in Fig 55.

It must be noticed that even for a trivial case of M = 4 and N = 2, Qs is quite a huge matrix 2048 by 2048. Nevertheless, if we choose to initialize it as zeros matrix instead of random, this results sparse: the possible transitions from one state to any other are fewer, as computed in the equation:

$$n_a^* = M \cdot N \cdot n_{bk}, \qquad (45)$$

where n_{bk} is the number of black cells in the state space. Therefore the worst case is $n_a = (MxN)^2$, in our example is $n_a = 64$.

Experimental setup

The considered robot is the MiniARM (SPARK project website (shown in Fig 56), a custom built seven degrees of freedoms manipulator with revolute joints. The architecture for both simulation and hardware control is sketched in a diagram in Fig 56 Virtual Reality Modeling Language (VRML) models of simple environment and manipulator have been realized in order to have the same function interfaces as the real hardware control devices.
The highest level control is achieved with a PC, while the low level control is performed with a 32 bit mcu-based board. The PC-board communication is done with a USB-serial interface, while the robot is position-controlled through a RS485 serial bus.
Although positions are known and few, in the case study the end-effector positioning is realized with inverse kinematic algorithm for generality (an iterative novel strategy called Mean of Multiple Computation MMC Cruse, 1998 and Arena, 2009).

Experimental results and comparisons

A reinforcement learning algorithm, because of the high parameter and particular strategy dependency, is not easy to be numerically compared to other approaches. Under this point of view a cross-comparison between predictive and non predictive algorithm seemed a straightforward comparison.
Moreover the well known ACO meta-heuristic method (Dorigo, 2004) is used as gold-reference to give idea of best achievable performance without explicitly computing it.
Two different performance indexes have been chosen to compare the proposed algorithm with respect to other classical SARSA-based approaches. The first is the normalized number of steps needed for a single task achievement.

$$n_s = \frac{n_{steps}}{n_{bk}} \qquad (46)$$

where nsteps is the overall number of steps executed by the agent while nbk is the number of black squares in the checkerboard.
The second performance index is:

$$l_t = \frac{d - d_b^*}{n_{bk}}, \qquad (47)$$

where d is the total distance traveled by the agent and d_b^* is the minimum distance between the end-effector starting position and the black squares. It is clear how though the minimum value for l_t is $l_t^* = 1$, not all configurations admit this value as optimal sequence because of the distance between black squares in initial state. A typical dynamical evolution of the two indexes, chosen to evaluate system performance, through iteration is shown is Fig 57. As it is possible to see, both indexes show a fast decrease through iterations. Considering an average time per movement t_{avg} = 1s, the overall computational time requested per iteration is far less and therefore, as discussed above, the number of real agent actions can be kept low.
In order to compare the performances, a number of iteration of 1000 was chosen (*i.e.* number of real robot actions).
Each robot experiment is completed in about t_{max} = 20 minutes (*i.e.* all together with computational time).
As shown in Fig 58 and in Fig 59, the presence of the state transition model improves the learning time in terms of number of actions performed from the real robot: both chosen indexes show faster decreasing in presence of a predictive model. It must be remarked that the ACO algorithm has been implemented, for simplicity, to solve shortest path sub-task without considering the actions leading to white squares.
Moreover, by using the ACO strategy the problem is solved iteratively for a given static environment configuration and it is not possible to extract a general model from the data. Nevertheless, it is a quite good benchmark algorithm because it provides a fast numerical suboptimal solution for a particular problem. For each RL iteration,

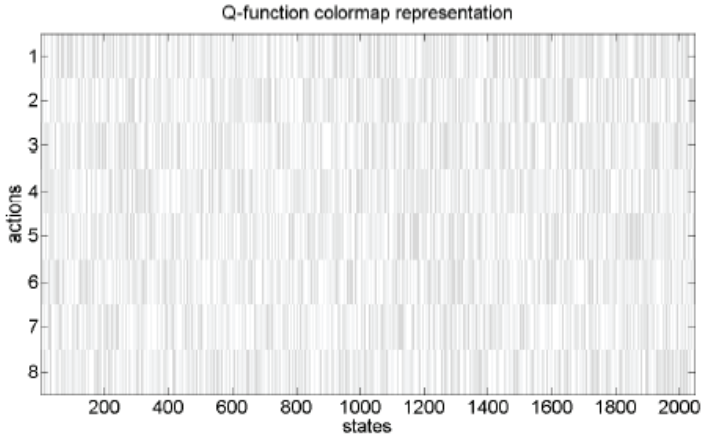


Fig 60. Example Q-function before learning. In abscissa there are all the possible 2048 states, in ordinate there are all the possible actions. As shown, random high value (lighter) initializations have been chosen.

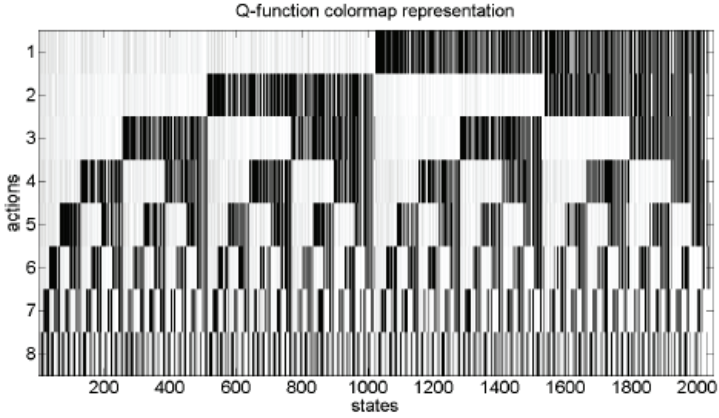


Fig 61. Example of Q-function after learning. In abscissa there are all the possible 2048 states, in ordinate there are all the possible actions. Due to the binary coding of the states, the trained Q-function looks like a binary tree, confirming the correct training. Nevertheless, high value stripes (lighter) in low value areas (darker) indicates non-explored state-action pairs.

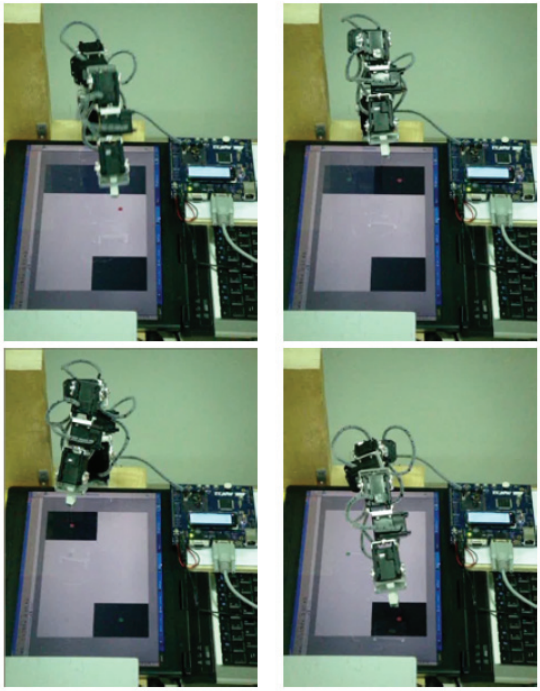


Fig 62. Example of sequence reproduction after learning process has been successfully performed. A laptop LCD was adopted to visualize the solution.

i_{ACO} = 500 have been performed for a computational time t_{ACO} = 0.12s per each solution. The color map depicted in Fig 60 shows the state-action weights matrix Q before learning. Iteration after iteration the weight matrix changes its values. The color map in Fig 61 shows the same matrix after learning: as it is possible to see, all non-rewarding actions values are decreased.
As previously introduced, all described learning results have been obtained using both simulations and the real robot.
Up to now no sensors have been applied for state transition check. A laptop LCD monitor has been used for solution visualization. Nevertheless, a touch sensitive panel is a straightforward upgrade that can be used in order to close the loop with the real hardware. Snapshots from the real manipulator learning are shown in Fig 62.

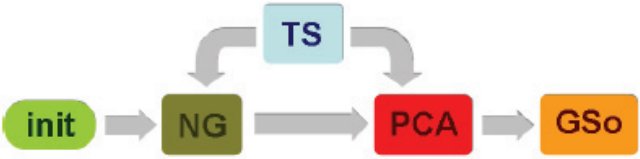


Fig 63. Classic, NGPCA algortim block diagram. The NG block performs center updating for a given training vector, extracted by Training Selector (TS), the PCA block executes one step of local principal components algorithm and GSo (Graham-Schmidt ortho-normalization) is able to normalize eigenvectors.

Fig 64. Algorithm block diagram for sequence learning with Neural Gas with local Principal Component Analysis (NGPCA). The Pattern Constructor block (PC) performs control modifying the input pattern with the oset Pd. The miniARM block is the serial redundant manipulator (controlled in the operating space).

In this section, an application to shortest path discrete problem in real hardware is presented. Both simulation and experimental results show the improvement achieved thanks to the state transition model. It must be remarked that a considerable improvement was obtained in the real application because the number of actions performed by the real agent (end-effector manipulator) was heavily reduced. In fact, the virtual agent performs a lot of simulated actions based on what previously learnt by the real agent.

Moreover it following paragraphs of this chapter, a generalization of the Neural Gas Algorithm for sequence learning and reproduction will be shown.

Sequences generation

One of the possible extensions of this vector reconstruction strategy is to introduce the actual state x_i and next state x_{i+1} of the considered system as part of the input vector

$$\mathbf{p}_i = (\mathbf{x}_i, \mathbf{x}_{i+1}), \qquad i = 1, 2, ..., N-1. \tag{48}$$

In this way it is possible to learn not only a static complex law but even a particular sequence.
Under this point of view it is possible to make both short and long term prediction and then analyze model prediction with real sensor information for updating the internal model. The so generalized model, shown in Fig 64, has been applied to control a custom built seven degrees of freedom redundant manipulator (SPARK II website) not only in forward and inverse kinematic problem solving (see Hoffmann, 2003) but also in operating space motion planning.

The introduced Pattern Constructor block (PC) processes inputs for the abstract network (NGPCA) in order to determine the constrained space for the current iteration, reading the joint variables , and therefore leading the system toward the given reference Pd (*e.g.* solving forward kinematics), as in

$$\mathbf{x_i} = f(\theta_\mathbf{i}, \mathbf{P_d}). \tag{49}$$

The miniARM block is the serial manipulator itself and has been implemented both in simulation and in real hardware setup. It take an input vector reference and give a feedback vector (*e.g.* it can be implemented in the operating space solving the inverse kinematics and reading joint angular position) (see [5],[6]).
The same overall model architecture could also be used to determine an iterative converging recall algorithm, not discussed in this chapter, modifying the one described in the previous section, in order to control the system with constrains toward a desired trajectory in the free-parameters subspace.

Experimental setup
As in other experiment, the overall control of the robot is made up of a high level sequence control in the computer and a low level hardware layer control (custom built) with a 32 bit microcontroller unit.

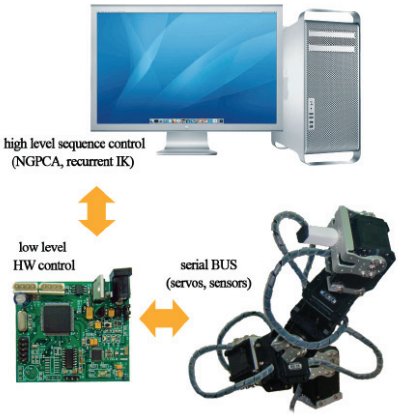


Fig 65. Functional diagram of the implemented control algorithm.

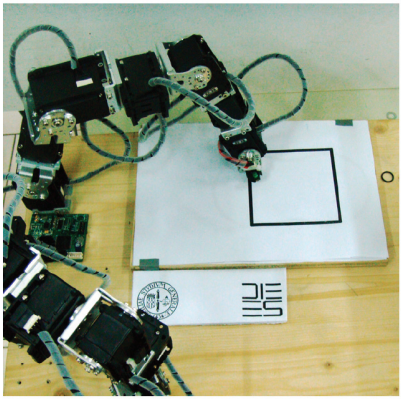


Fig 66. Picture of MiniARM first prototype realized in our laboratories. Experimental setup for square sequence learning in a plane.

Architecture functional diagram is shown in Fig 65.

Training set

The training set have been acquired from the real robot reading all the encoder positions through the direct kinematic of the manipulator. Each data set is made up of N=1000 three-dimensional points acquired every 0.08*s* inside the operating space during a user guided real-time trajectory following.

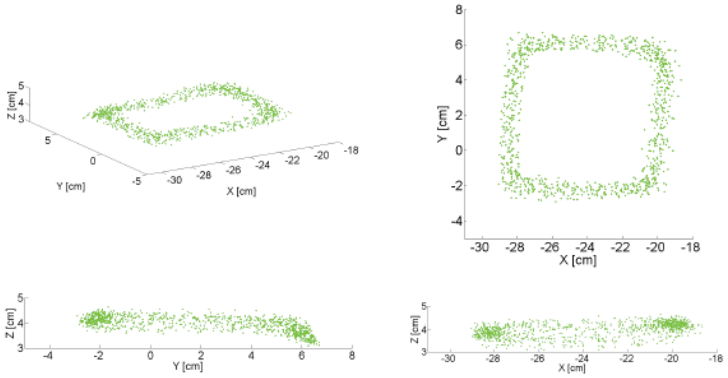


Fig 67. Three dimensional plot of an example data set acquired with n =1000 (up-left), XY plane plot (up-right), YZ (down-left), XZ (down-right).

$$\mathbf{x_i} \triangleq (x_i, y_i, z_i), \qquad (50)$$

and

$$\mathbf{p}_i = (x_i, y_i, z_i, x_{i+1}, y_{i+1}, z_{i+1}), \qquad i = 1, 2, ..., N-1.$$

Though desired trajectories are planar and repetitive, the acquisition method is very noisy and three dimensional by definition, as shown in Fig 67.
Learning patterns have been built up using two consecutive points from acquired data set.

Results
Performance of the algorithm have been tested under different operating conditions. All presented results have been obtained with the experimental setup shown in Fig 66.

Normal operations
As described in the following relationship, the algorithm provide just the next point of the sequence in the operating space given the actual point as part of the reconstructed pattern $\hat{z}^*$

$$\hat{\mathbf{z}}^* = \mathbf{M}\hat{\eta}_{\mathbf{j}^*} + \mathbf{p}, \qquad (51)$$

where the offset vector p is able to define the constrained subspace

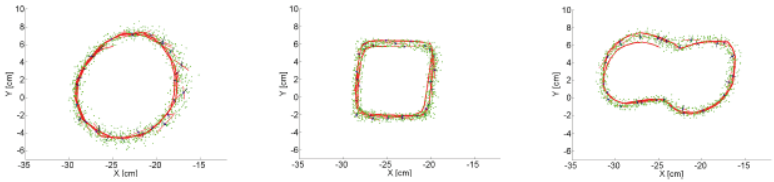


Fig 68. Examples of sequences learned with the NGPCA.

as in equation (52) iteration after iteration.

$$\mathbf{p}_i = (x_i, y_i, z_i, 0, 0, 0), \qquad i = 1, 2, ..., N; \tag{52}$$

The M, as usual (see previous chapters for details), align free and constrained subspaces in separate regions of the whole space.

$$\mathbf{M} = \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix}, \qquad \mathbf{0} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{53}$$

The joint space trajectories are generated solving iteratively the inverse kinematic problem (see Cruse, 1993 and Arena, 2009). Robot reproduced trajectories are shown in Fig 68.
Three different trajectories have been reproduced in order to analyze the generalization capabilities. All these have been generated with human-guided points acquisition method.

Numeric robustness
One of the most important features of the algorithm is that both learning phase and recall phase have very high numeric robustness. As shown in Fig 67, learning data set defines a real noisy trajectory. Same trajectories have been reproduced adding noise in feedback variable as in equation (54).

$$\theta = \widehat{\theta}(1 + e(rand - 0.5)), \tag{54}$$

where $\hat{\theta}$ is the measured feedback variable vector while $rand \subseteq [0, 1]$ with uniform probability density distribution and $e \subseteq [0, 1]$ is the maximum error amplitude.

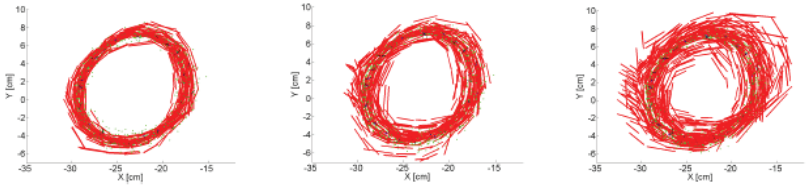


Fig 70. Examples of NGPCA sequences reproduction in presence of simulated feedback errors: p = 0:05 (5% of random component in) (on the left), p = 0:1 of random component (center) and p = 0:15 (right).

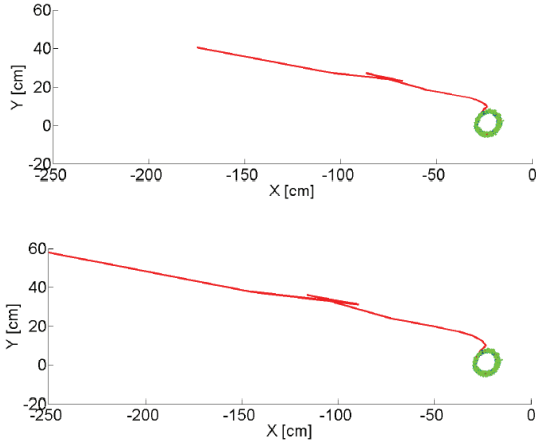


Fig 69. Trajectories reproduced from points far away from the trained-operating space. DDR=7 on the left and DDR=10 on the right.

The circle trajectory has been chosen in order to test numeric robustness of sequence reproduction.

Performance outside learned space

The performance outside the operating space have been tested using multiple distances from the centre of the trajectory and measuring the number of steps needed to go through the sequence (*i.e.* the Euclidean distance E_i under a chosen threshold $E_{th} = 0.1$ *cm* as in inequality (55)).

Though particular values strongly depend on the shape of the path

DDR	2	10	100	1000
n	13	18	24	36
n_{10}	16	20	24	36

TABLE VII Algorithm performance outside the learned space

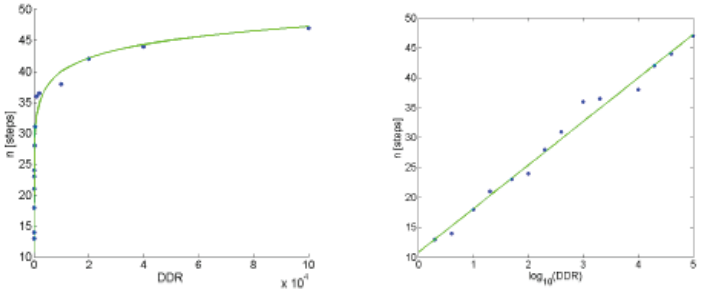


Fig 71. Number of steps needed to go into the sequence respect to the DDR on the left (dots are real data while line is a logarithmic interpolation), same plot with DDR on logarithmic scale (abscissa) on the right.

and on the learning phase, for a given trajectory and a dened training phase values can be compared through all dierent distances. In order to normalized these distances with the eective dimension of the path an adimensional Distance over Dimension Ratio (DDR) is defined as in equation (56).

$$E_i < E_{th}, \tag{55}$$

$$DDR = \frac{d}{m_d}, \tag{56}$$

where *d [m]* is the distance from the centre of the trajectory and m_d*[m]* is the maximum dimension of it. TABLE VII and Fig 71 summarize the algorithm performance: n is the number of steps noiseless and n_{10} is the same quantity when an additional 10% of random component is added in feedback variable .

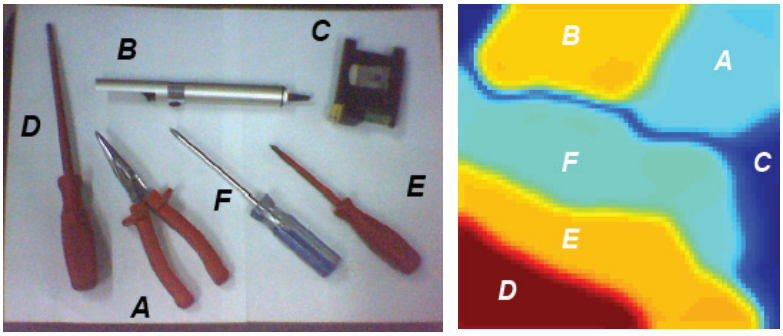


Fig 72. Example image used for sequence learning (on the left). 70x70 cells map for Self Organizing Map (SOM) implementation (on the right).

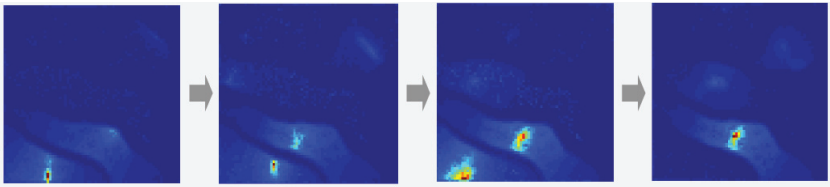


Fig 73. Sequence reproduced by a learned SOM. Extracted from longer sequence: ...F-A-B-C-**D-E F**-A...

Reversed operations

As in common RNNs, in the NGPCA the pattern reconstruction is possible careless on which part of the pattern is lacking. Therefore it is possible to use the same learning not only for one way sequences reproduction but also for reversed sequences.

$$\mathbf{p}_i = (0, 0, 0, x_i, y_i, z_i), \qquad i = 1, 2, ..., N, \qquad \mathbf{M} = \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} \quad (57)$$

In the last part of this chaper, the Neural Gas algorithm with local Principal Component Analysis implementation has been tested and extended for the control of motion sequences for a redundant serial manipulator.

As it is possible to imagine its modification for different robotic struc-

tures are minor and, in any case,straightforward.
Though the training phase needs a complete training set and a computational effort the recalling phase is very fast and possible to implement also in a common microcontroller-based platform. The one-to-many mappings, the prediction capability and the inputs/outputs role independence in the training phase shows the possibility of generalization over a wide range of control applications.
Several different implementation with similar structures can be done. In Fig 72 and in Fig 73 some results of a SOM-based implementation herewith used just for performances comparison are depicted. Nevertheless, within the paradigm of sub-problem hierarchical solving, all of these can directly be exchanged with those presented here.

chapter 5

a general framework for robot control and system integration: *from structure to complex algorithms*

Small universities and schools have several problems to cope with robot architecture definition.
Principal problems are introduced by limited founds and by the reduced hardware development knowledge and possibilities.
The main needs are for sure the short time-to-demonstration for untested algorithm (that needs powerful debug tools); low cost, programming simplicity (avoid low-level programming) through different operating systems (namely Windows, Linux and eventually MacOS), high customizability and opening to 3rd party devices and applications (indeed this is probably the most important feature because of the growing robotic community all over the world).

Though several existing modular architectures are present on the

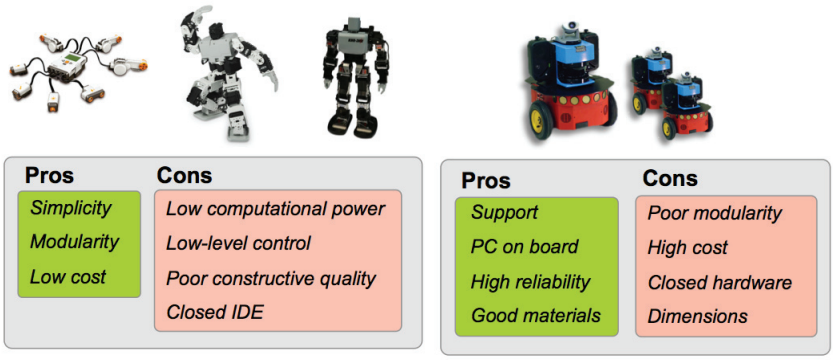


Fig 74. Commercial solution for actuated modular systems built up as network of several modules: sensors, control cores and actuator. Main solutions can be split by the overall cost. Low cost solution on the left while high cost solution on the right.

market, probably because of the the dimension of this small technological niche none of them is actually universally adopted.
Moreover, each of them has several features but it lacks of something else and none of them can completely replace all the others (Fig 74).

Its most important features are:
- very high computational power (up to 54 Gflops)
- high performance-per-watts
- x86 CPU architecture (windows, linux, macos, …)
- GPU parallel computation with CUDA™
- remote control and multi-robot cooperation trough 802.15.4/n
- iNemo™ and Teseo™ STMicroelectronics technologies
- decentralized low-level control (simple wiring through buses)
- very high hardware modularity

As it is possible to imagine, because of the nature of this project the two most important features can be considered the hardware modularity and the computational capability (indeed, strictly related to hardware modularity because, replicated parallel hardware is faster than any of the most powerful supercomputers. Moreover the modularity of this solution is crucial to guarantee the interface with multiple and flexible sensor interfaces and actuator systems.
In order to test the goodness of the proposed approach, several struc-

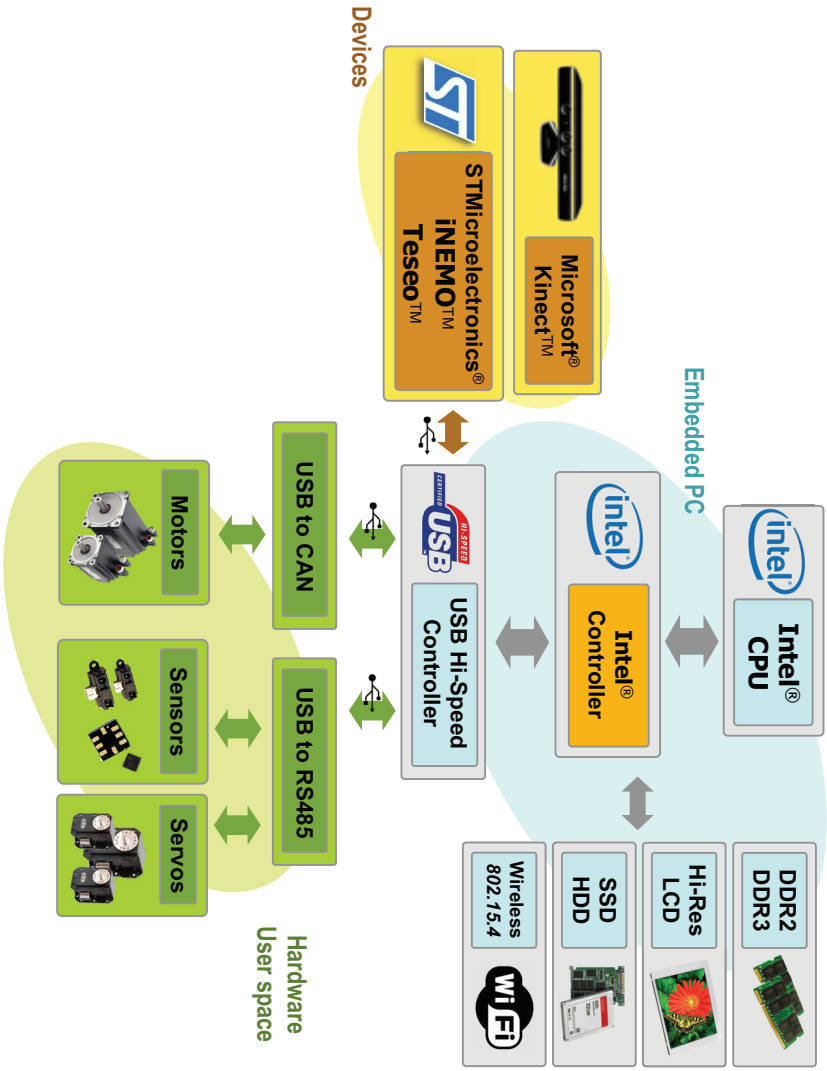


Fig 75. General block diagram for system network control. It could be (and indeed it have been) used for robot control (sensors and servo) and for sensor network solution.

tures have been realized and tested with simple and complex algorithms.

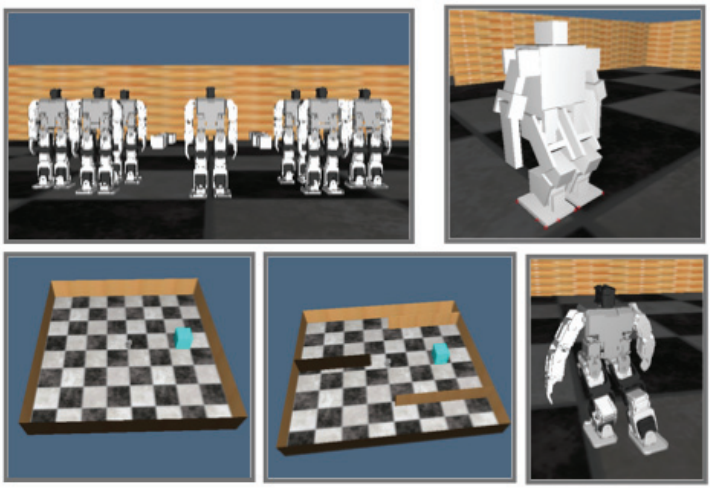


Fig 76. Snapshots from a physics simulation based on nVidia PhysX and C++ implementation. The Bioloid robot and the miniARM have been simulated in order to obtain a realistic interaction simulation (for both each-other and environment interaction).

The hierarchical sub-problem approach proposed as biologically driven solution of the motion control problem is mirrored in the presented solution: the level of abstraction of the control grows layer by layer. In this direction it is possible to modify a complete layer (algorithm, reflex or mechanical layer) without changing the other if the overall interfaces are kept. In this way the nature of the mechanical structure, as well as any other part of the architecture, can be changed with minor effort.

The analyzed platforms are various and very different. The most used platforms were a roving platform a serial manipulator and a legged robot (e.g. a humanoid structure). Also a simulator of all of these has been developed and interfaced to some part of the hardware.

Rovers

The roving platform used for navigation experiments, is a modified version of the dual drive Lynx Motion rover, called Rover I and II. It is a classic four wheeled drive rover controlled through a differential drive system. The robot dimensions are 35 x 35 *cm* about.

It is equipped with an on-board netbook based on Intel Atom 455, two sonar range (detection range 3 *cm* to 3 *m*), a low level target sensor to detect color spot on the ground, the Eye-RIS v1.3 visual system for panosferic application (*e.g.* insects behaviour experiments), a 640x480 webcam and the Eye-RIS v2.1 visual system mounted on a pan-tilt module for frontal view purpose (*e.g.* focused attention and feature detection in insects).
The Eye-RIS systems are bio-inspired vision devices that implement retina-like architecture which combines signal acquisition and embedded processing on the same physical structure. The core of the device is the Q-Eye chip, an evolution of the previously adopted Analogic Cellular Engines (ACE), the family of stand-alone chips developed in the last decade and capable of performing analogue and logic operations on the same architecture. The Q-Eye was devised to overcome the main drawbacks of ACE chips, such as lack of robustness and large power consumption.
Eye-RIS is a multiprocessor system since it employs two different processors: the Anafocus' Q-Eye Focal Plane Processor and the Altera's Nios II Digital Soft Core processors. The AnaFocus Q-Eye Focal Plane Processor (FPP) acts as an Image Coprocessor: it acquires and processes images, extracting the relevant information from the scene being analyzed, usually with no intervention of the Nios II processor. Its basic analog processing operations among pixels are linear convolutions with programmable masks. Size of the acquired and processed image is the Q-CIF (Quarter Common Intermediate Format) standard 176 x 144. Altera NIOS II digital processor is a FPGA-synthesizable digital microprocessor (32-bit RISC µP at 70 MHz- realized on a FPGA). It controls the execution flow and processes the information provided by the FPP.
Generally, this information is not an image, but image features processed by Q-Eye. Thus, no image transfer is usually needed in Eye-RIS, increasing in this way the frequency of operation.

As said before, the robot is equipped with two different versions of the Eye-Ris vision system. Practically, the 2.1 version has the same capabilities of the 1.3 one reducing the power consumption and the dimensions.

The robot is also equipped with the STEVAL-MKI062V2 board (from STMicroelectronics). It is the second generation of the iNEMO module family. As already described, it combines accelerometers, gyroscopes and magnetometers with pressure and temperature sensors to provide 3-axis sensing of linear, angular and magnetic motion, complemented with temperature and barometer/altitude readings.

In the robot, the inertial sensor board (*i.e.* the iNEMO) and the accelerometers are used to control low level (*e.g.* rotation and forward motion) movements and the magnetometers for robot orientation.

Finally the robot is completely autonomous from the power supply. One 14:8V, 5Ah Li Poly battery pack is used that guarantee autonomy of about 2.5 hours.

The complete control architecture, similar to the one reported in Fig 75, shows the distributed architecture used to control all the activities of the robot.
As already discussed, all the modules of the robot are independent (and interchangeable) and the communication is implemented thanks to a Master-Slave protocol on RS485 bus at 1 Mbit.
The adopted communication protocol allows to exchange information among different nodes and supports connections such as TTL, RS485, and XBee.

Thanks to the variety of the sensors mounted and the on board computer, the Rover permits to perform a different types of experiments in a wide variety of scenarios (*i.e.* the Robot is able to perform a different type of image processing thanks to the three different types of the image acquisition sensors mounted).

Industrial-like serial manipulator: the miniARM case
Serial manipulators are, nowadays, commonly used in industrial applications, for example in the automotive market.
Their manipulation capabilities and kinematic properties are already known and well studied in past decades. Nevertheless their use in

service robotics is almost completely limited in big static structure with high costs and a really closed development and control environment. Due to these characteristics they are often not well suited for academic research purposes. Exceptions are Schunk (PowerCube, used for MMC experiment and Light Weight Arm) and DLR modular robots, for high-end users, and Lynxmotion Robotic Arms, mostly for low-end users and hobbyists.

Inside the SPARK II project, a standard serial manipulator for research activity, called PowerCube, is used to apply the cognitive algorithms to different robotic platforms.
The PowerCube is a commercial platform provided by Schunk. In order to decrease a development time, an efficient small version of PowerCube, called MiniARM, has been designed and realized.
Its control has been implemeted as part of the proposed architecture and the modularity of the approach has been kept.
As introduced, for the same reason, its accurate kinematic and dynamic model have been developed. Their extensive use have been shown trough the chapters of this work for algorithm implementation and debug.

The MiniARM, shown in Fig 77, is a serial manipulator with rotational joints. It is completely custom built and developed in order to analyze and compare algorithm performances with low implementation effort: the underling idea for the design was the realization of a robust testbed for high-level control algorithms. In this way, the most important features were the control and simulation simplicity of a such complex structure.
The chosen actuators for the entire system are the Dynamixel smart servos from Robotis and an embedded microcontroller board with an AVR32 UC3A 32bit microcontroller architecture, from Atmel, is used to control all the servos and retrieve sensory information through a RS-485 four-wires bus. It must be noticed that each of this smart servo as an internal microcontroller to perform a reflexes control very similar to the multiple muscles control presented in (Latash, 2008) under the Equilibrium Point (EP) hypothesis.
A custom built sensor unit with a three-axial accelerometer a gyro-

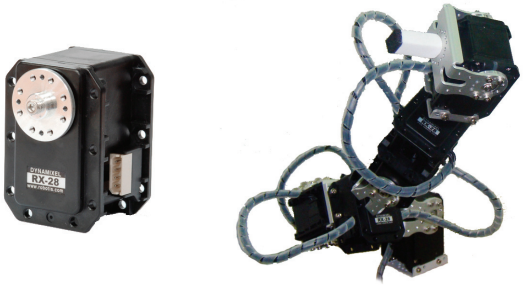


Fig 77. Smart servo used to implement a bio-inspired torque ramp (dynamixel from Robotis)

scope and a digital compass (more deeply discussed in the next chapter) is linkable to any of the present servo through the common serial bus.
As in the case of the rover platform, all the high level control algorithms can be developed directly in the PC (Windows, Linux and MacOS operating systems are supported) using a USB-to-Serial communication and thanks to a Matlab Graphical User Interface and a developed interface library.
An external USB camera directly connected to the PC is supported by the GUI to achieve image segmentation and feature detection.
The compatibility with all the most common simulation tools and control environments is given by a standard 3D model reconstruction, realized in a CAD drawing software (Solid Works) using IGES model distributed by Robotis (available on the web), together with a simplified low-count vertex VRML mesh and the control interface library.

The MiniARM serial kinematic chain is designed to be redundant in space, within its dextrous operating space, and the mechanical structure has been realized for high payload/weight ratios (1 : 2, 1 : 3).

The geometry of the manipulator and the assemble of the entire structure have been studied through a detailed CAD project,.
The final developed configuration shows the subsequence of seven rotational axis servos.
Almost all the structure interconnection frames are made with light

commercial aluminium parts. The gripper module is a custom design. A fully functional first prototype of the overall architecture has been realized. The estimated payload is of about 300*g* when fully stretched. A simple pointer as been realized as end-effector for inverse kinematic problem solving using multiple approaches.
The overall robot weight has been estimated to be about 0.7 Kg (slightly depends on last module) and it is 45 *cm* long, together with the end-effector module.
Preliminary power supply test shows a maximum power consumption of 3.5 A with full payload. Despite the complexity of the structure the wiring is extremely simple thanks to the communication bus presence. The mechanical structure has been assembled using aluminium brackets (2 mm thick) and hexagonal mounting screws (M2 and M2.5 and M3).

Multi-limb structures and complex algorithms
Robotic multi-limb structures are widely used in research as test-bed for biologically relevant models (Siciliano ed. 2008) (Cruse, 1998). The control of these kinds of structures is complex and often implies high computational capabilities or preprogrammed motion control. On the other hand, the biological paradigm of the Central Pattern Generator (CPG) can be considered to impose a certain sequence of states on the arms and legs of a robot. In some cases, a redundant configuration can be used, increasing the complexity needed to define a model of the structure and to develop a reliable control algorithm. The robustness against singularities, the capability to face angular constraints in the joint space or other constraints in the operating workspace (*e.g.* obstacle avoidance) are important aspects that have to be considered (Ahn, 2002) (Shimizu, 2008) (Tondu, 2006).
In embedded control platforms, an important performance index that can be, therefore, considered is the computational power. In fact, it constrains the control frequency and therefore the possibility to develop an algorithm on a low MIPS microcontroller unit (mcu) for real-time applications.
The proposed approach is based on the already developed, herewith presented in the previous chapters, Mean of Multiple Compu-

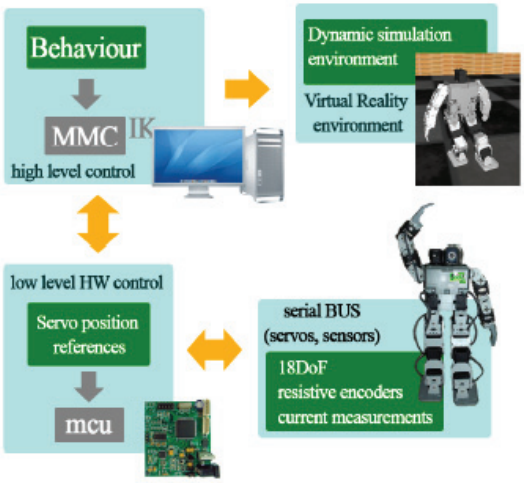


Fig 78. Hardware configuration of the experimental setup: low level microcontroller AVR32 board, actuators and sensors bus architecture- and high level control host computer.

tations (MMC) Recurrent Neural Network (RNN) algorithm which provides a fast, flexible and robust to configuration singularities sub-optimal solution to the discussed problems (Cruse, 1993) (Arena, 2009).
In the following section the MMC, a direct and inverse kinematic solver, is used to control the motion of a very complex structure. In particular the case of a humanoid robot (*i.e.* Bioloid from Robotis) with 18 Degrees of Freedom has been taken into account.

Limbs model and control strategies
In order to kinematically model both arms and legs of the humanoid, different structures have been considered.
The modelled geometrical structure for each arm (*i.e.* left and right) is a three links serial structure with three Degrees of Freedom (DoF) (from θ_1 to θ_3 for each arm).
The geometrical structure used for each leg is a four links serial structure with five (from θ_1 to $\theta_5$for each leg) DoF and therefore more than one DoF are computed with the same geometrical quantity (*e.g.* two relative joints are associated with the same link).

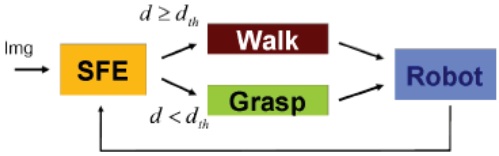


Fig 79. Simple behaviour selection loop implemented through visual-processing and robot control. Segmentation and Feature Extraction (SFE) block performs image processing algorithm to detect object distance d. Behaviour evaluation is achieved with a binary threshold d_{th}.

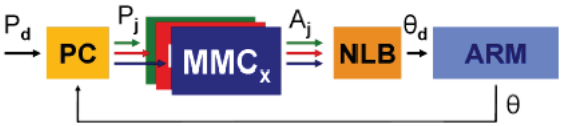


Fig 80. Control scheme for each arm for a given absolute reference input pattern, P_d. PC is the Pattern Constructor, MMC_x, MMC_y and MMC_z are the three-dimensional linear computational networks. NLB is the Non-Linear Block and ARM is the manipulator itself (simulator or real robot). P_j are the MMC input pattern while A_j are the outputs. θ_d are the desired angular values and are the read joints values.

Control strategies

As presented in the previous chapters and as thought in the proposed architecture, the robot control can be easily and conceptually divided into high level (*i.e.* behaviour) control task, and low level (*i.e.* gait and grasping strategies) control.

1) Robot behavior: The robot overall behavior is controlled through simple visual frame-based decision making through direction of walking and grasping. The behavior selection loop is shown in Fig 79. The Segmentation and Feature Extraction block (SFE) processes image from on-board camera and it uses distance d for discrimination.

2) Grasping: as introduced, two different position controlled MMCs with three links have been used to control recognized object grasp-

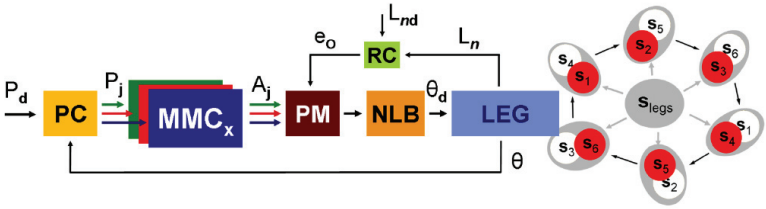


Fig 81. Control scheme for each arm for a given absolute reference-input pattern, P_d. PC is the Pattern Constructor, MMC_x, MMC_y and MMC_z are the three-dimensional linear computational networks. NLB is the Non-Linear Block and miniARM is the manipulator itself (simulator or real robot). P_j are the MMC input pattern while A_j are the outputs.θ_d are the desired angular values and are the read joints values.

ing. The simplest extension of the two-dimensional positioning of a planar redundant serial manipulator (shown in the previous chapter) (deeply described in Cruse, 1993 and Cruse, 1998) is the three-dimensional positioning of the end-effector for the redundant serial manipulator in space (*i.e.* R^3).
As usual, each vector component of the geometrical links is processed, iteration after iteration by a different linear MMC network and then given as input at the non linear block (NLB), as shown in Fig 80.

3) Walking gait: For the described legs control strategy, the walking gait is obtained using a state dependent reference (from S_1 to S_6) for each leg and a state transition model with transition conditions (as sketched in Fig 81).
Two position and orientation controlled MMCs have been used for convergence toward reference within the state. State dependent offset are introduced in θ_{leg_1} (off-sagittal plane hip angle) and θ_{leg_5} (off-sagittal plane ankle angle) for balance control.
As depicted in Fig 81, in the algorithm block diagram, the low level control strategy guides the system (*i.e.* the single leg) towards the desired orientation through a relative error feedback. Therefore the feedback for the orientation error (e_O) is analyzed.
The distance between two different leg states can be defined, in

the simplest case, as the Euclidian distance between the two end-effector points.

Furthermore, overall distance (and therefore the whole system state) can be computed as the mean distance of both legs).

Simulation and experimental results
The considered robot has 18 DoF, through revolute joints: three joints for each arm and six joints for each leg. The physical structure can be mapped, as requested for the MMC modeling, into the geometric theoretical model links. Due to the chosen real robot architecture, angular limits are not equal for each joint.
As described in the previous section, each presented control strategy has been tested both in simulation and on real robot.
As in the previous case of MMC implementation, the proposed algorithms have been implemented both in the PC and in a custom mcu-based board. Respectively a Dual Core Intel Centrino 2.2 GHz host computer with 2GB of RAM and an UC3A AVR32 mcu with 66MHz of maximum clock and only 64KB of RAM were used as comparison platforms (Fig 78).
As sketched in Fig 78, in our first implementation the overall robot control can be split in two levels: low level (hardware) control and high level control. The low level control is, in all cases, achieved thanks to a mcu-based board that is used both to acquire information from the distributed sensory system and to control the actuators. Moreover a serial bus is used for low level communication purposes. The high level control and the data logging are made through a host PC connected to the board via serial interface (with a USB-to-serial transceiver).
In the second (embedded) version of the control algorithm all the MMC-based calculations are directly executed locally in the mcu-based board while the PC is just used for data logging and for the Virtual Reality (VR) simulation environment.

Simulation and Experimental Results
In order to test performance of the designed control structure, trajectories followed in both walking and grasping behaviours have been

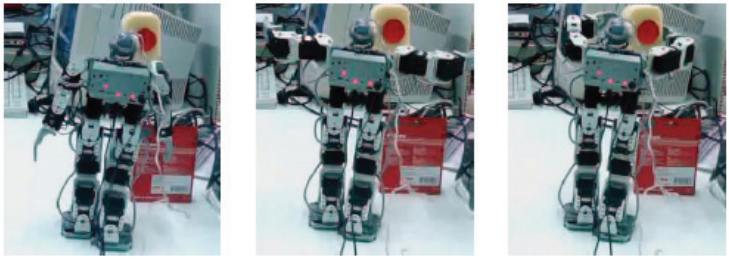


Fig 82. Multiple snapshots of grasping sequence. End of walking (i.e. d < dth) and start of grasping (on the left). MMC iterations are executed in parallel on both arms (center). Grasping sequence ends when both arm position errors go below error tolerance (on the right).

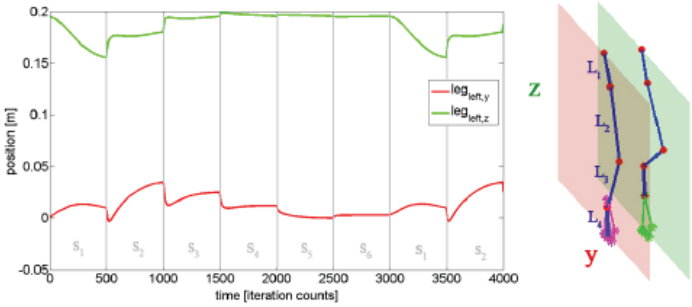


Fig 83. Left leg end-effector position through different states (onthe left). Relevant coordinates (robot sagittal plane) have been plotted with respect to time. Simple sketch of the geometrical structure inwhich legs are mapped (on the right)

analyzed. The center point trajectory of the foot-base (*i.e.* end-effector of leg serial structure) of one leg during sequence in the walking behavior is depicted in Fig 83. The distance from a known object is estimated based on simple segmented image feature (*e.g.* area in px^2 or maximum edge of minimum rectangle containing the object). After the object area (i.e. camera estimated distance under threshold dth) is reached the arms start to converge toward the selected centroid, as shown in Fig 82.

Although the iteration number needed to reach the desired reference strongly depends on the chosen parameters and on the particular given reference, in order to estimate the algorithm performance, multiple comparisons with common algorithms for kinematic inversion, such as pseudo-inverse Jacobian (J) and Jacobian transpose (J_t), have been considered. It must be taken into account that in the J* the joint limit constraints are introduced in the form of an additional task to be completed while in the J^t no joint limitations are considered.
Free parameters were chosen to maximize convergence speed keeping a non-overshoot condition on end-effector positioning.

A complete hardware and software architecture has been described and proposed. Modularity, short time-to-demonstration and hierarchical decomposition are the most important feature of this solution.

Several robotic structures and a complex algorithmic example have been presented.

The extension of the MMC approach to a multi-limb structure (*e.g.* a humanoid robot) is presented for walking and grasping behavior. The problem is addressed not only in presence of angle limits but also in a dynamically changing environment even in a mcu-based hardware.

Moreover, in the next chapter a first step to the development of an embedded solution for distributed sensing suitable for this hw/sw architecture will be presented.

chapter 6

advanced motion platform for inertial sensing: *beyond the pure motion control research*

Electronics devices, currently deployed in worldwide market, are using an increasing amount of MEMS technology for motion identification and reconstruction with multiple sensors. The greater part of these sensors is made up of accelerometers, gyroscopes and magnetometers (*i.e.* magnetic field sensors used as compasses) (Abbate, 2009). For instance, in the game industry, accelerometers and gyros are used as player user interfaces. In telecommunications, especially in smartphones, accelerometers are often used as inclinometers and gyros enable an improved gaming experience, and

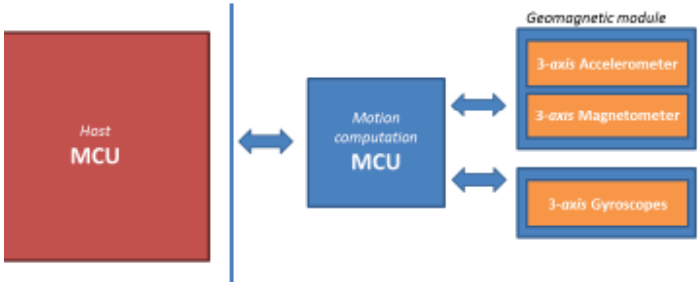


Fig 84. Inertial Motion Module (IMM) block diagram for host application integration: the module microcontroller takes data from the sensors (gyroscope and geomagnetic module), performs the computation needed for sensor fusion algorithm and orientation estimation and provides multiple communication interfaces to a host microcontroller (application level).

eventually a camera image stabilization (Cardani, 2006). Together with GPS and compass they are also investigated for augmented reality and Location Based Services (LBS) (Hide, 2005).
In sports and healthcare systems accelerometers are used for pedometer function (Foxlin, 2005). Adding gyros and, eventually, a compass allows for true 3-D motion tracking, which can be useful for evaluating trajectories and body movement (Roetemberg, 2007) (Roetemberg, 2006).
As it is possible to figure out, the computational capabilities of these applications are heterogeneous: todays high-end smartphones have powerful application processor (up to 1*GHz*) with Floating Point Unit (FPU) while consumer pedometer have a very low amount of computational power.

For what concerns smart system for movement analysis it must be noted that the overall complexity of this kind of systems is increasing over time. Under this point of view, while commercial platforms for reliable orientation estimation currently exist (Roetemberg, 2006), they are mostly standalone solutions not suitable for embedded system integration.
The possibility to have a small-packaged monolithic device that provides, effortless, the overall orientation “on demand”, integrating

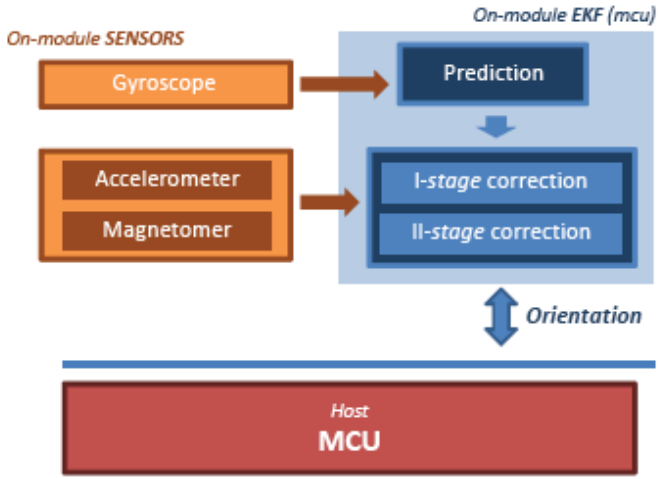


Fig 85. Block diagram of the sensor fusion algorithm running inside on-module processor (STM32) that uses gyroscope and geomagnetic module for orientation (i.e. quaternion vector) estimation.

Type	Sensor	Full-scale ranges
Accelerometer	LSM303DLH	±2*g*, ±4*g*, ±8*g*, ±16*g*
Magnetometer	LSM303DLH	±1.3, ±1.9, ±2.5, ±4.0 ±4.7, ±5.6, ±8.1 *gauss*
Gyroscope	L3G4200D	±250°/s, ±500°/s, ±2000°/s

TABLE IX Sensor specification in terms of full-scale range for accelerometer, gyroscope and digital compass (*i.e.* magnetometer).

accelerometer, gyroscope and digital compass (*i.e.* magnetometer) measures is still under development.

In order to investigate this solution, starting from the iNEMO™ evaluation board from STMicroelectronics (for further information see iNEMO page on ST website), whose dimensions are *4cm x 4cm*, a first prototype of a smart Inertial Motion Module (IMM) has been realized.

This module keeps the main functionalities of the iNEMO platform but with a very different form factor: *13mm x 13mm* (depicted in

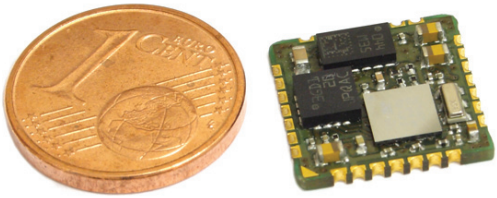


Fig 86. Realized prototype of iNEMO M1. As it is possible to observe it is very tiny. Its real dimensions are 13 x 13 x 2 mm^3. It embeds an accelerometer a gyroscope, a digital compass with a Cortex M3 core from STMicroelectronics. A Kalman filter internally performs the sensor fusion.

Fig 86).
From design issues to first results, this chapter introduces this module through different sections. In particular first, the hardware and conceptual architecture of the module have been addressed together with sensors and microcontroller specifications. Next, the algorithmic solution together with the firmware implementation will be analyzed with respect to the state of the art and then, some output results of the orientation estimation of the module filter are showed. Finally a couple of straightforward applications of this system are described.
Let's start from the module architecture.

Module architecture
The module takes data from the sensors (gyroscope and geomagnetic module), performs the computation needed for sensor fusion algorithm and orientation estimation and provides multiple communication interfaces to an host microcontroller (application level) (Fig 84).
Furthermore, the IMM design aims to reach the best possible integration level in user applications: for this reason particularly attention was paid to the form factor and to the pin out. The overall system can be easily seen as a Surface-Mount Device (SMD) that features 28 pins in a 169 mm^2 PCB. It means that in this, very small size, all the components are placed on the top layer and at its edges metal-

ized areas realize the module pin out. In this way the module can be directly soldered in the user system as a single component.

As briefly sketched, the designed architecture is made up of a 32-bit microcontroller unit (MCU), a geomagnetic module (3-axis accelerometer and a 3-axis magnetometer) and a 3-axis gyroscope (Fig 85). While the inter-module communication (between local MCU and sensors) is managed through the I^2C and SPI serial interfaces, the communication with the outside world (*e.g.* between the module itself and the host application) can be achieved through several interfaces (USART, SPI, I^2C, CAN).
The microcontroller is a STM32F103 from STMicroelectronics and it features an ARM Cortex M3 architecture (from STM32 reference manual). This MCU has a maximum core frequency of 72 MHz, a computational capability of 1.25 DMIPS/MHz, 64 Kbyte of RAM and 512 Kbyte of flash memory. To keep the overall system dimensions as small as possible a Wafer-Level Chip Scale Package (WLCSP) has been chosen for this component for a volume of 4.5x4.4x0.5 mm^3. The gyroscope is the L3G4200D (also from ST) (information available in L3G4200D datasheet), a 16-bit digital sensor with user-selectable full-scale and bandwidth: the former is between ±250°/s and ±2000°/s while the latter is between 100 *Hz* and 800 *Hz*. Its package form factor is 4x4x1.1 mm^3.
The geomagnetic module is the LSM303DLH (from ST) [9]. It unifies in a single tiny package (3x5x1 mm3) both the accelerometer and the magnetometer: the former full-scale, user modifiable, is between ±2g and ±16g; the latter, also user-selectable, full-scale are between ±1.3 gauss and ±8.1 gauss.

Powered by a single supply voltage between 2.16V and 3.6V, provides orientation data (*e.g.* rotation matrix, quaternions and/or Euler angles) with a maximum update rate of 100 Hz. The fully operating power consumption is around 210mW (@3V), nevertheless low power and stop modes can be forced by the external application processor through both hardware and software commands.
Let us consider now the adopted software solution.

Software solution
The idea to combine digital processing and calibrated sensor measurements to provide an absolute orientation data to the host application processor is achieved through an Extended Kalman Filter (EKF) (Bishop, 2001)), running on the embedded MCU, that manages the smart sensor fusion to obtain the benefit of each sensor in orientation estimation (Fig 85). Even if the EKF theory is beyond the scope of this chapter, it is useful to remark that its discrete time formulation, it tries to estimate the state of a generic non-linear system from a system model and multiple noisy sensor measurements. Its algorithmic procedure iterates through cycles and each of them can be easily divided in two main parts: prediction phase and correction phase.

As showed in (Bishop, 2001), after the prediction phase updates the state using he model of the process, the output is corrected through the measures and Kalman gain computation. The gyroscope measures are used for strap-down integration (Sabatini, 2005) (Sabatini, 2006) in the prediction phase while both accelerometer and magnetometer measures are used to compensate the angular-rate integration drift in the correction phase (Yun 2006). In particular the accelerometers provide an attitude reference using gravity acceleration projections while the magnetometers provide a heading reference using the earth's magnetic field vector (Yun, 2006).
The implemented filter is based on quaternion vector $\vec{q} = [\ q_0 \quad q_1 \quad q_2 \quad q_3\ ]$ estimation and therefore they are part of state vector.
The quaternions formulation ensures, at the same time, the lack of representation singularities typical of Roll, Pitch and Yaw (RPY) based filters and an efficient mathematical framework (Yun, 2006) (Vlasic, 2007).

Nevertheless both rotation matrices notation and RPY orientation representation are internally computed and available as outputs. It must be noticed that, without a filter structure and a sensor fusion algorithm, considering only separate independent measurements, as usually done in current application, the orientation data would easily be far from the true trajectory due to: noisy integration (typical of

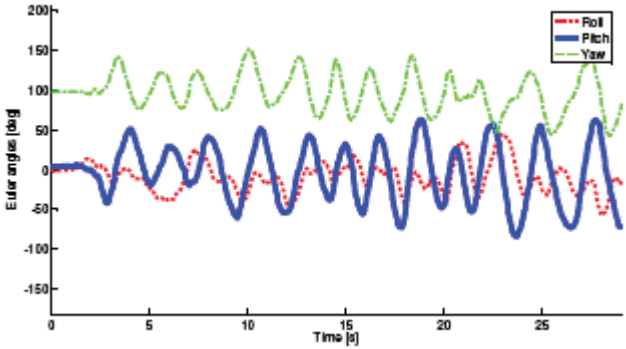


Fig 87. Euler angles time series from the proposed Inertial Motion-Module solution: Roll (red), Pitch (blue) and Yaw (green).

gyroscopes), low-pass filtering (accelerometer) and magnetic disturbances (compasses problems).
Moreover the Kalman filter correction step provides an on-the-fly calibration for the gyros by providing corrections to the attitude trajectory and a characterization of the gyro bias state.
Furthermore, in order to reduce disturbances due to external accelerations (*i.e.* different from gravity) and perturbing magnetic fields, respective sensor covariance has been considered variable in the filter (Roetemberg, 2006) (Roetemberg, 2007).
In this way the individual sensor error is estimated dynamically from measures.

Results and future development
As described in the previous sections the IMM provides orientation data in the form of quaternions, rotation matrix or RPY. Some examples of RPY angles estimation obtained in a preliminary motion tests are showed in Fig 87. Overall errors of the system have been estimated to be around ±0.5° in Pitch and Roll angles and ±2° in Yaw in quasi-static condition while ±1.5° in Pitch and Roll angles and ±4° in Yaw in free motion (within sensors operative range) (example in Fig. 6). It must be noticed that the adopted microcontroller, as often happens in low-cost units, does not have a FPU and therefore all the mathematical computations for sensor data filtering and fusion

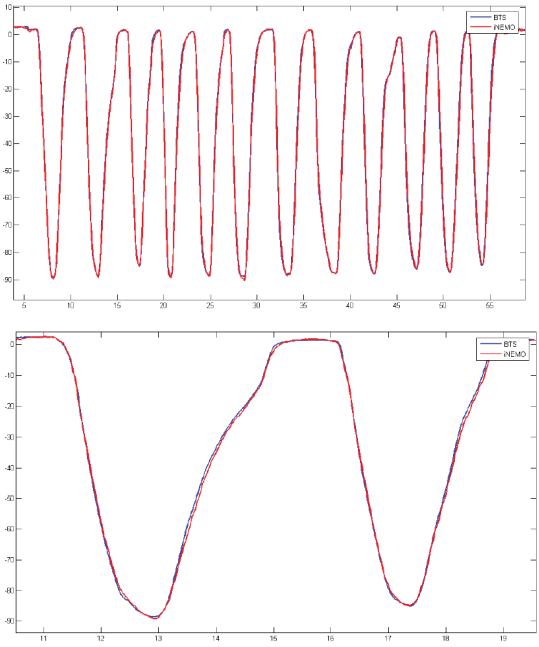


Fig 88. Data from inertial (iNEMO, solid red line) and optical systems (BTS, solid blue line) comparison

requires several CPU clock cycles and must be optimized. A preliminary timing and complexity analysis of the implemented algorithm on the considered processing core (STM32) lead to an iteration time of 4 x10^{-3} s (*i.e.* time needed for a complete prediction-correction cycle), for a theoretical maximum filter update rate of 250Hz.

Beyond all the typical applications in which the orientation of the single device is the only need (e.g. smartphone, Inertial Navigation Systems), an increasing interest in Inertial Measurement Units (IMUs) is coming from MOtion CAPture (MOCAP) (Vlasic, 2007) (Roetemberg, 2007) systems (especially form lowcost ones) from entertainment industry (*e.g.* game and movies), from medical rehabilitation studies and from sport performance analysts.

Two similar versions of the module have been realized and additional tests have been performed in order to evaluate the overall robust-

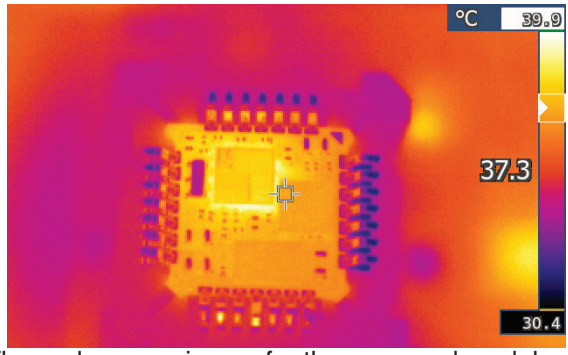


Fig 89. Thermal camera image for the proposed module with environment temperature around 25°C in fully operative condition. No forced convection nor any dissipative substrate have been considered in this test.

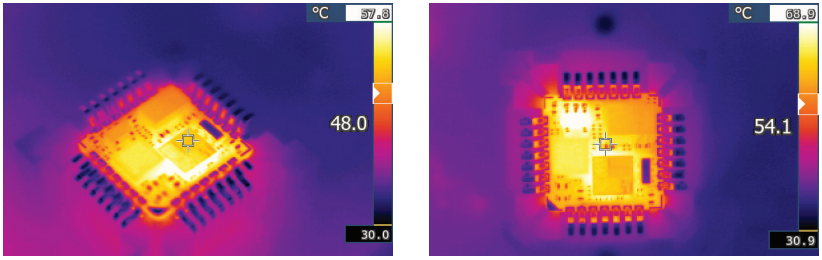


Fig 90. Thermal camera image for alternative version of the proposed module. A linear regulator is present in these images

ness of the both.
The first prototype has been thought as a component like *System-on-Board* and under this understanding it doesn't feature an internal voltage regulator.
The second prototype has some minor pinout revision and more importantly it features an internal LDO in order to be powered up to 5V.
Some pictures of thermal analysis of both of them are shown in Fig 89 and in Fig 90. Additionally some X-Ray analysis has been done on a malfunctioning unit.

Motion Capture solution
In order to validate these approaches and to perform a feasibility study for module-based realization, a first prototype of a MOCAP so-

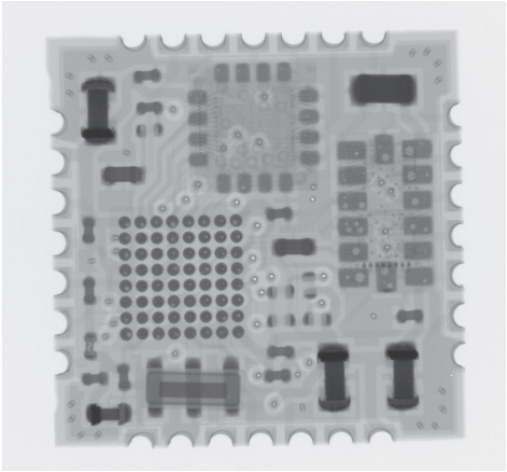


Fig 91. X-Ray vision of a bad functioning unit.

lution, based on the iNEMO evaluation board, has been developed. In particular this system has been realized by daisy-chained module network achieved through a serial bus. In order to extend this prototype to a module based implementation the module-bus interface needs to be managed, in our analysis by an adapter board whose only functions are to match voltages and physical layer bus requirements.

Let us consider this scenario as a straightforward application of the described Motion Processing Module and a RS485 bus interfaced with the USART port of the module (up to 4.5*Mbps*). Under these considerations this interface board has been designed with just two 4-poles connector, a low drop voltage regulator and a RS485 3.3V transceiver. In this way with a total amount of 10 components (including passives) in an overall volume of 20x17x5 mm^3 this platform is able to provide a daisy-chained sensor network suitable for all kinds of MOCAP applications.
Considering a maximum 100 Bytes (largely much more than necessary in typical applications) for sensor node orientation communication, through a bus protocol, to the master of the network (*e.g.* host computer or an embedded system working as a concentrator) and considering a 100 *Hz* update rate, the throughput on the physical

Fig 92. Motion sequence captured from a little girl learning physical interaction between her body and the environment.

layer of the bus should be able to cope with more than 35 devices.

In this chapter a tiny Surface-Mount Device (SMD) module to provide both the calibrated sensor measurements and an effortless orientation measure of the system to any kind of embedded applications has been presented. This device integrates accelerometer,

Fig 93. MOCAP interensting points in human body. In its simplest implementation it counts 10 sensors: two per each limb plus two through the spine. Additional sensors could be mounted on wrist, ankles and head.

gyroscope and magnetometer measures and is able to combine them with digital processing capabilities within an embedded sensor fusion algorithm. Some design considerations and benefits with respect to the state of the art have been considered together with a very brief description and scheme of some possible applications. Under this point of view it is important to highlight that, as exemplified in MOCAP application description, the possibility to have a monolithic, component-like, device ready to be soldered could simplify a lot users application design.

The developed application is still under development within the STMicroelectronics as technological demonstration of the capabilities of inertial sensors.

Moreover as additional investigation a combined approach between inertial Motion Capture and Simultaneous Localization And Mapping (SLAM) has been performed.
Under this point of view, it must be considered that, at the time of writing, SLAM algorithms are almost completely related to robotic research labs and so to robotics platform. In most cases this is

Fig 94. Snapshot for the realized graphical user interface for inertial Body Motion Reconstruction (iBMR) demonstration.

achieved through odometry information from the robotic platform (e.g. PIONEER from mobile robots) fused together with scanner laser information (examples from SICK, Hokuyo).

It must be noticed that a Microsoft kinect-based SLAM is reliable, low cost and quite simple but
- has some robustness issues;
- does not provide body tracking nor any body related information.
At the same time, the inertial full body tracking provides:
- body tracking;
- rough estimation in relative pose change for camera (or kinect);
- a step into the sensor network concept.

At the same time traditional computer vision approaches to depth estimation in image processing have been significantly enhanced. The hardware embedded in Microsoft Kinect solution is based on an Infrared (IR) projector and a standard CMOS sensor and it is capable, through the Prime Sense (PS1080) chip, to correctly estimate a depth image (640x480*@60fps*) in indoor environment. The accuracy

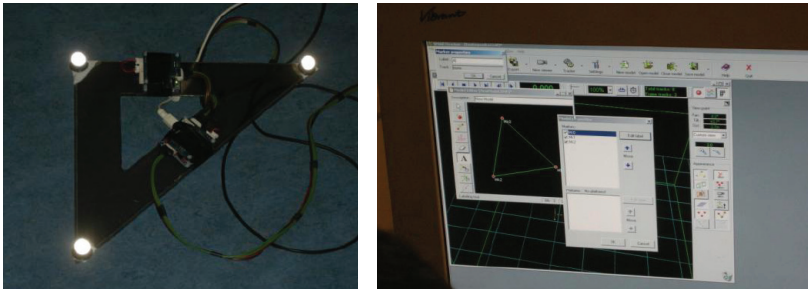


Fig 95. Experimental setup used for inertial Attitude and Heading Reference systems evaluation. In order to evaluate performance the system has been compared with an optical tracking system gold standard.

of this solution (±3cm in 5m max distance) is certainly lower than the laser--based solution (±1mm over 5m of range) and its horizontal Field Of View (60°) is also largely narrower (240°).
Nevertheless having RGB and Depth images (RGB-D), aligned in a true 3D coloured perspective, in a simple and low cost solution seems attractive.

The possibility to have the whole inertial data and RGB-D images available can be used in multiple ways: first of all, an animated virtual avatar of the agent could be integrated in the 3D rendered scene; second the MOCAP data could be used as additional information to initialize/enhance the standard ICP algorithm (eventually substituting the RGB image processing step); finally the created virtual world could serve to enhance inertial data information in terms of position estimation (the inertial data truly estimate the orientation and provide position trough environment hypothesis).
Dieter Fox et al. (Fox, 2011) at the University of Washington (in Seattle WA), in a joint work with Intel Labs proposed a RGB-D Mapping technique for dense 3D modelling in indoor environment (reference).

Their idea, reproduced in the following pages, is to start with a standard Iterative Closest Point (ICP), commonly used for SLAM, and to enhance it considering additional information obtained from aligned depth and colour images provided by the Prime Sense solu-

Fig 97. Hardware modification for combined SLAM and MOCAP approach. The iNEMO-based solution for inertial movement reconstruction has been linked to a Microsoft Kinect-based solution for environment reconstruction.

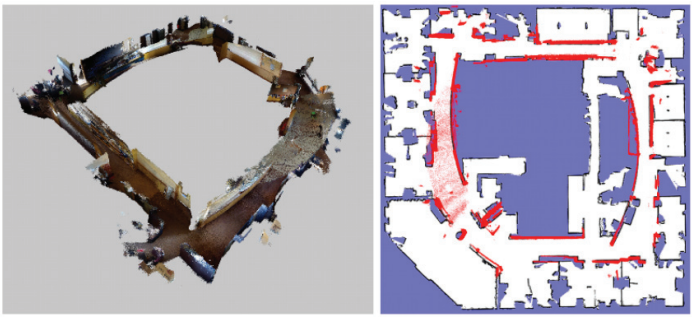


Fig 96. (On the left) 3D maps generated by RGB-D Mapping for large loops in the Intel. (on the right) Maps (red) overlaid on a 2D laser scan map of the Intel Lab. For clarity, most floor and ceiling points were removed from the 3D maps (extracted from D. Fox work)

tion. They executed a first level of image processing in order to have multiple information on the same feature from the two images and two minimize local minima problems in the ICP algorithm. Moreover they organize the globally aligned dense 3D cloud in small surfaces, called surfels for rendering simplicity and speed.
Some brief consideration on both inertial Kalman Filter and on ICP algorithm are reported below.

Extended Kalman Filter

The idea behind the sensor fusion is that, using several kinds of sensors, the characteristics of one type of sensor avoid overcoming the limitation of other sensors (Bishop 2001)(Sabatini, 2005).
In this way, the Kalman filter is useful for combining data from sev-

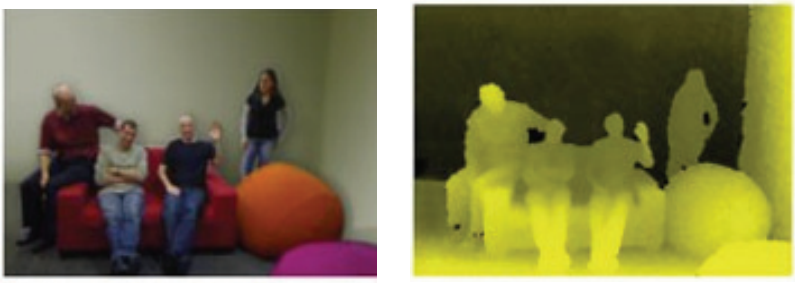


Fig 98. RGB-D camera images. The colour information is depicted on the left (RGB) while the depth information is on the right (monochrome) (pictures from prime sense website).

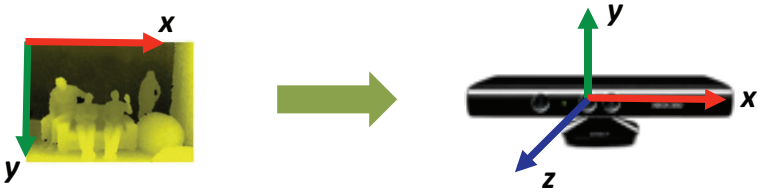


Fig 99. Frame of references from image coordinate system to real system coordinates (real world reference) .

eral different noisy measurements. In fact, gyroscopes measure orientation by integrating angular rates whereas the accelerometer and magnetometer provide a noisy and disturbed but drift-free measurement of orientation. The Kalman filter weights the three sources of information appropriately with knowledge about the signal characteristics based on their models, to make the best use of all sensor data.

In general, the Extended Kalman Filter algorithm addresses the problem of trying to estimate the state of a discrete-time process described by the equations below:

(78)

where x is the state vector; u is the input vector; A_k, B_k, H_k, are respectively state, input, and output matrices; furthermore w, v are state and measurement noise.

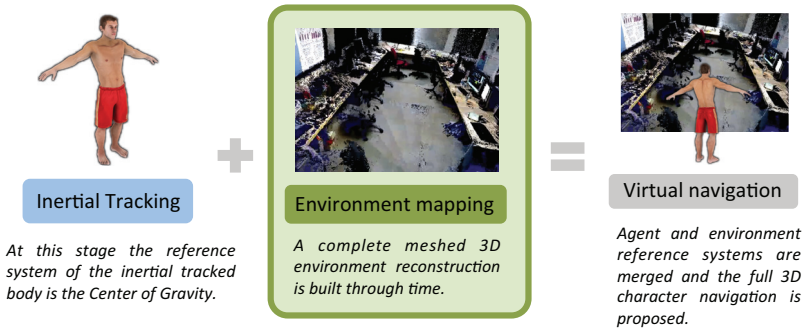


Fig 100. Overall structure of mixed inertial-camera environment reconstruction solution based on human movement motion capture system together with an RGB-D environment mapping.

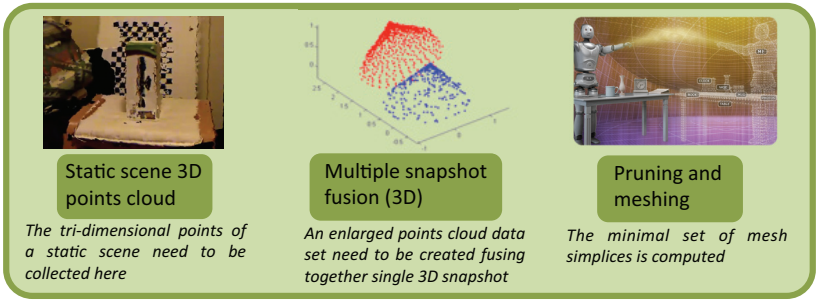


Fig 101. Major problems in environment mapping and reconstruction from multiple snapshot from a RGB-D camera like Microsoft kinect.

The state and measurement noise are Gaussian and white noise sources with covariance matrix Q and R respectively.
At each time step, the algorithm propagates both the state estimation and the error covariance matrix. The latter provides an indication of the uncertainty associated with the current state estimation. These are evaluated in the prediction equations.
The Kalman Gain is derived from minimizing the a posteriori error covariance, and it could be considered as a measurement of the level of confidence to give to the predict state.
If the the problem is an orientation problem, as in the considered case, one good choice is to use the unit quaternion-based orientation representation ($\| q \| = 1$) in order to avoid singularities.

In this case, one good choice (Sabatini, 2005) for the state vector is to made it up of orientation and gyroscopes biases.
Considering now the general Extended Kalman Filter equations:

$$\begin{aligned} \dot{x} &= f(x,\omega) + w \\ y &= h(x) + v \end{aligned} \qquad (79)$$

where:

$$x = \begin{bmatrix} q & b_w \end{bmatrix}^T \quad q = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^T$$

and

$$b = \begin{bmatrix} b_{w_x} & b_{w_y} & b_{w_z} \end{bmatrix}^T \qquad \vec{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T \qquad (80)$$

The measurement vector is $y = a$ where $a = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T$
In a strap-down inertial navigation system, the rigid body angular motion is described by the differential equation:

$$\dot{q} = \begin{bmatrix} [\vec{\omega}\times] & \vec{\omega} \\ -\vec{\omega}^T & 0 \end{bmatrix} q \qquad (81)$$

where:

$$[\vec{\omega}\times] = \begin{bmatrix} 0 & \omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \qquad (82)$$

If quasi-static condition, when the acceleration acting on the body is far less than the gravity acceleration the following is true:

$$\begin{cases} \vec{\omega} = G_g \omega_r + \vec{b}_\omega + \vec{v}_g \\ \vec{a} = G_a \left[C_n^b(q)(\vec{g}) \right] + \vec{v}_a \end{cases} \qquad (83)$$

While the previuos section was dedicated to Extended Kalman filter algorithm the next one will be devoted to Iterative Closest Point (ICP) algorithm and to possible applications to point clouds.

Iterative Closest Point and its application to mesh alignment
As briefly discussed the first big problem working with multiple point clouds is alignment.
Going more in deep, in our workflow (depicted in Fig 101) the snap-shots from the RGB-D camera are, sooner or later, converted in a cloud of points and considering a, even large, range of speeds for

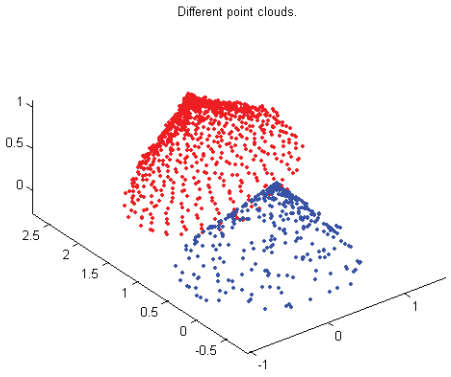


Fig 102. A point cloud (blue) and its transformed version (red) after a roto-translation matrix have been applied. Iterative Closest Point (ICP) algorithm inted to solve the inverse problem: given the two meshes

the motion of the human body the 30 frame per seconds of those cameras are usually enough to obtain successive overlapped frames.
The result is that usually the points in two clouds (respectively a_i and b_i) are mostly overlapped and then a rotation matrix between them can be computed as follows:

$$\sum_{i=1}^{n} ||Ra_i - t - b_i||^2 \qquad (84)$$

where

$$\bar{a} = \frac{1}{|A|}\sum_i a_i \qquad \bar{b} = \frac{1}{|B|}\sum_i b_i \qquad (85)$$

and

$$t = R\bar{a} - \bar{b} \qquad (86)$$

where

$$R = VU^T \qquad N = \sum_{i=1}^{n} a_i' b_i'^T \qquad (87)$$

and

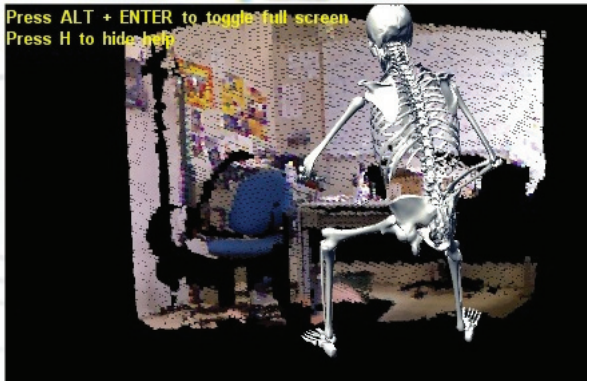


Fig 103. Snapshot from the working developed solution based on inertial full body tracking (iBMR) and Microsoft Kinect.

$$N = U\Sigma V^T \qquad (88)$$

The implemented ICP algorithm is made up of two major phases: a matching step and a transformation step.
At each step both have to be performed. In the matching step each point of cloud B (or A) is assigned to the *nearest* point in cloud A (or B). As in any other matching algorithm, metric could make the difference: distance could be computed just as simple point-to-point distance or point-to-plane or even more complex non-euclieadian distances but this is out of the scope of this work. At this time, let us consider the simplest point-to-point distance.

Though in its naive implementation the complexity of the matching step is $O(n^2)$ it is quite trivial to move toward a $O(n \log(n))$ implementation, using much more complex data structure (*e.g.* kdtree).

One of the major concerns about the ICP algorithm is that its convergence is often affected by wrong local minima drop.
For this reason the proposed idea is to use all the information from the Inertial Body Motion Recognition (iBMR with iNEMO) to estimate (even, roughly) the initial position and orientation of the camera (*e.g.* the kinect).
The iBMR is presented as a technology of recording movements of

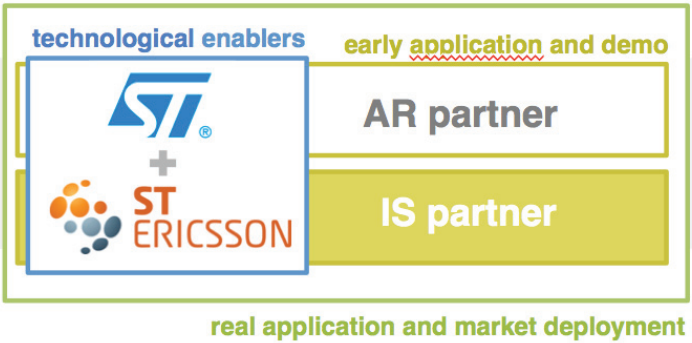


Fig 104. How to proceed toward a system integration for inertial body reconstruction and augmented reality. While, in line of principle, the technological needs are full filled by ST-Microelectronics and ST-Ericsson.

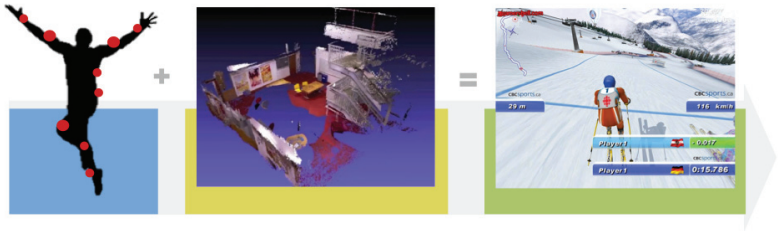


Fig 105. Major problems in environment mapping and reconstruction from multiple snapshot from a RGB-D camera like Microsoft kinect.

the human body and translating them to a digital model in real-time mode. Its applications include animation of characters in movies and video games; gait analysis (study of human motion) in clinical and sports medicine to help identify posture-related or movement-related problems in people with injuries and help athletes run more efficiently.

As described, each node functions as an Attitude and Heading Reference System (AHRS) with 9-axis MEMS sensing of linear, angular, and magnetic motion. It integrates a 32-bit microcontroller that computes the complex AHRS algorithm, using ST's proprietary filtering and predictive software for sensor data fusion. All nodes are sending

their data to the control unit – a PC – which applies the measurement to a graphical skeleton model and displays body motions in real time.

In the current version of the inertial motion capture solution (iBMR), there is a iNEMO *M1* system in each node: on each arm, forearm, thigh, calf, and two nodes on the back; however, the system is scalable up to 15 nodes, so additional nodes can, for example, be mounted on shoes or on the head.
The size of the node is 4 x 4 x 10 *mm*, its weight is 12 *g*, and it is encapsulated in a special housing.
Extensive tests with realistic, complex motions of the human body have been carried out, in which this solution showed outstanding precision and speed. Deviation in spatial accuracy is below 0.5 *cm* during the movement and negligible in motionless conditions. Meanwhile, the system is able to process the acquired data in less than 15 *ms* - the time elapsed from the acquisition of sensor inputs up to their applying to the skeleton model in the control unit.

Moreover it is important to highlight that as discussed in block diagrams depicted in Fig 104 and in Fig 105 the combined inertial motion capture and camera based environment mapping and reconstruction open multiple scenarios into the augmented reality applications.

Consider now just a limb (*e.g.* an arm) and start to develop its kinematic model. Then think a step beyond and link this analysis approach to a possible synthesis in a robotic perspective.
It must be remarked that this indicates absolute orientation, therefore if used in multi-link structure (*e.g.* human limbs) relative orientation have to be retrieved if relative rotation have to be performed for kinematic motion of the model.

Kinematic model of the human body
If we take into consideration that each sensor computes the absolute orientation, starting from the Centre of Gravity (CoG) of the body, geometrical relationships (Sciavicco ed., 2008) leads to following

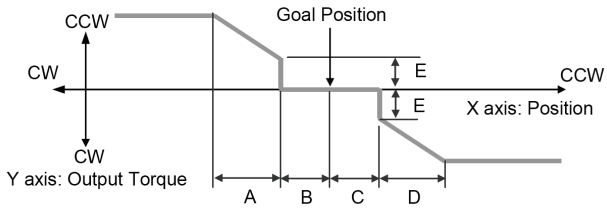


Fig 106. Torque ramp characteristic of the dynamixel servo. B and C are no laod zones, useless from a theoretical point of view, useful for real life application (and strictly related to E,F value, minimal torque value). A and D value determine the slope of the ramp.

equations:

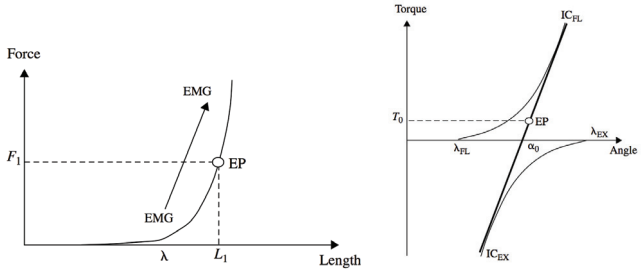


Fig 107. Equilibrium point hypothesis (Latash 2008) in single muscle reflex and agonist-antagonist couple for joint motion (picture from Latash, 2008).

$$p_{Elbow,0} = p_{Shoulder,0} + R^0_{Shoulder} \cdot p_{Elbow} \quad (89)$$

and to its similars (ankles, wrists, knees, etc.)
In which P_{limb}
with $limb = \{Cord_1, Cord_2, Hip, Shoulder, Knee, Ankle, Head\}$
have been derived from a kinematic body skeleton.
Moreover it must be noticed that quaternion needed to derive rotation matrices (Sciavicco ed., 2008) are computed locally from each node (through Kalman filter Bishop, 2001) while matrices are reconstructed by the host as follows:

$$\vec{q} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^T \quad (90)$$

$$R = a \begin{pmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1 q_2 + q_3 q_4) & 2(q_1 q_3 - q_2 q_4) \\ 2(q_1 q_2 - q_3 q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2 q_3 + q_1 q_4) \\ 2(q_1 q_3 + q_2 q_4) & 2(q_2 q_3 - q_1 q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{pmatrix}$$

where

$$a = \frac{1}{\sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}}$$

and q_4 is the scalar part of the quaternion.

Similarly to what happens with threshold position control in agonist-antagonist muscles couple (Latash, 2008), using these servos it is possible to separately control, and just with parameters change, both the equilibrium angular position of the joint and its stiffness. The resulting dynamical system can be therefore considered globally asintotically stable around a phase-state point described by the angle-torque couple. The effective dynamics of the system from outside the equilibrium point toward it depends both on the inner control loop and on the environmental condition.

For sake of simplicity the case of a single human upper limb has been selected and its movement monitored through two inertial platforms (*i.e. i*Nemo boards from STMicroelectronics): one for each major kinematic link, the arm and the forearm.
At each time step the rotations around the articulation of the shoulder and of the elbow have been identified.
Moreover the orientation of each limb is then used to rotate the model parts that are kinematically constrained to be linked to each other.

$$\vec{p}_{1,0} = R_1^0 \cdot \vec{p}_{1,1}$$
$$\vec{p}_{2,0} = R_1^0 R_2^1 \cdot \vec{p}_{2,2} + R_1^0 \cdot \vec{p}_{1,1} \qquad (91)$$

It must be remarked that the rotation matrix obtained for each link is absolute, therefore R_0^2 is directly computed as follow:

$$R_1^0 R_2^1 = R_2^0 \qquad (92)$$

Elementary, single axis, joint rotations have been selected for the given kinematic chain of the serial manipulator from the measured five DoF.
Under this point of view the rotation matrix can be computed through

multiple rotation matrices around rotated cartesian axis: x, y or z. In particular, the forearm rotations can be described as two rotations around elbow articulation in order to be comparable with joint actuation in the robotic structure.

$$R_2^1 = R_0^1 R_2^0 \text{ and } R_2^1 = R_z R_y \quad (93)$$

Comparing previous relations it is possible to write:

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} = \begin{pmatrix} \cos\phi\cos\theta & \sin\phi & 0 \\ -\sin\phi\cos\theta & \cos\phi & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \quad (94)$$

In which, for $r_{22} \neq 0$ and for $r_{33} \neq 0$, angles ϕ and θ can be determined as follows:

$$\phi = \tan^{-1}\frac{r_{12}}{r_{22}} \text{ and } \theta = -\tan^{-1}\frac{r_{31}}{r_{33}}$$

This is just to show that using all these relations, the developed distributed sensory system and the control hardware architecture it is possible to mimic the human body motion in a robotic structure. Furthermore it is possible to reproduce recorded movement into a topologically (i.e. kinematically) different structure and finally to outsource the action perception loop...

....but this is another story.

conclusion

To conclude, wrap-up and propose how to go beyond what it has been already done, let me briefly discuss about this work.
The challenging problem of human machine interaction has been addressed from an analysis-to-synthesis perspective in the field of motion control.
The overall problem has been divided into several sub-problems.
Major hardware concerns are related to a modular architecture able to cope with very different platforms: sensors, actuators, power and logic interfaces and mathematical algorithmic cores.
From a software perspective it is important to distinguish between low level programming and high level programming.

While both need to be developed in such type of solution, the former is mainly involved in distributed decentralized control (*e.g.* actuator feedback control under parametric high level control), the latter is devoted to complex algorithm implementation.
Under these points of view, each of the proposed architectures try to efficiently divide these two aspects: reflexes implementation and sensor fusion algorithms.
Several algorithms for reflex-like motor control have been first described, then implemented and finally tested on real platforms.
A systematic solution to problem of system integration has been proposed. In this field a sensor solution and a network architecture for embedded sensing has been presented.
Problems like embedded computing and reduced computational power on mobile platforms have been considered and addressed.
Advances beyond the state of the art are herewith proposed for both structures and algorithms.
Wherever it has been possible, multiple real life problems like movement disorders or inertial based machine interfaces have been considered

As a final remark, the path toward a satisfying solution to the problem of motion analysis and synthetic motion generation and control in a robotic (and hopefully, everyday life) perspective, it is still far and hard (very hard). However I think that all the presented solutions are in line with the already introduced idea that this work should serve just as first gatherer of technological and mathematical research.
I also hope that the reader has taken home some food for thoughts to further exploit each of these sub-problems from both my good solution and from my mistakes.

references

[1] B. Siciliano O. Khatib, editors, ”Handbook of robotics”, Springer 2008.

[2] D.N. Nenchev, ”Redundancy resolution through local optimization: review”, J. Robot. Syst., vol. 6,no. 6,pp. 769-798, 1989.

[3] D. N. Nenchev, Y. Tsumaki, ”Singularity-consinstent kinematic redundancy resolution for the S-R-S manipulator”, in Proc. 2004 IEEE/RSJ Int. Conf. Intell. Robots Syst., pp. 3607-3612, 2004.

[4] K. Ahn and W.K. Chung, ”Optimization with joint space reduction and extension induced by kinematic limits for redundant manipula-

tors", in Proc. IEEE Int. Conf. Robot Autom., Washington, DC, pp. 2412-2417, 2002.

[5] T.F.Chan and R. V. Dubey, "A weighted least-norm solution based scheme for avoiding joint limits for redundant joint manipulators", IEEE Trans. Robot. Autom., vol. 11, no. 2, pp. 286-292, Apr. 1995.

[6] M. Shimizu, H. Kayuka, "Analytical Inverse Kinematic Computation for 7-DOF Redundant Manipulators With Joint Limits and Its Application to Redundancy Resolution", IEEE transaction on robotics vol.24, No.5, October 2008.

[7] B. Tondu, "A closed-form inverse kinematic modelling of a 7R anthropomorphic upper limb based on a joint parametrization", in Proc. 6th IEEE-RAS Int. Conf. Hum. Robots, pp. 423-432, 2006.

[8] H. Moradi and S. Lee, "Joint limit analysis and elbow movement minimization for redundant manipulators using closed form method", in Advances in Intelligent Computingm, Berlin/Heidelberg: Springer, , Part 2, vol. 3645, pp. 423-432, 2005.

[9] S. Lee and A. K. Bejczy, "Redundant arm kinematic control based on parametrization", in Proc. IEEE Int. Conf. Robot. Autom., Sacramento, CA, pp. 458-465, 1991.

[10] H. Cruse and U. Steinkuhler, "Solution of the direct and inverse kinematics problems by a common algorithm based on the mean of multiple computations", Biol. Cybernetics 69, 345-351 2, 1993.

[11] P. Arena and L. Patan, "Spatio Temporal Patterns for Action Oriented Perception in Roving Robots", Springer, Series: Cognitive Systems Monographs, vol. 1, 2009.

[12] H. Cruse, U. Steinkuhler and Ch. Burkamp, "MMC - a recurrent neural network which can be used as manipulable body model", in From animals to animats 5, R. Pfeifer, B. Blumberg, J.-A. Meyer, S.W. Wilson (eds.) MIT Press, pp. 381-389, 1998.

[13] G. Bosco and R. E. Poppale, ”Proprioception from a spinocerebellar perspective”, Physiol. Rev. 81: 539-568, 2001.

[14] Kandel, Schwartz and Jessel, ”Principles of neural science”, McGraw- Hill, 2000.

[15] M. C. Park, A. Belhaj-Saf, M. Gordon, and P. D. Cheney, ”Consistent Features in the Forelimb Representation of Primary Motor Cortex in Rhesus Macaques”, The Journal of Neuroscience, April 15, 2001, 21(8):2784-2792.10

[16] C. Capaday, C. Ethier, L. Brizzi, A. Sik, C. Van Vreeswijk, D. Gingras, ”On the Nature of the Intrinsic Connectivity of the Cat Motor Cortex”, J. Neurophys. 102: 2131-2141, 2009.

[17] W. T. Miller, R. P. Hewes, F. H. Glanz and L. G. Kraft, ”Real-Time Dynamic Control of an Industrial. Manipulator Using a Neural-Network-Based. Learning Controller”, IEEE Trans. on Robotics and Automation vol. 6. no. I, February 1990.

[18] H. Ritter, T. Martinetz and K. Schulten, ”Neural Computation and selsorga izing maps”, Addison-Wesley, New York, 1992.

[19] U. Steinkuhler, H. Cruse, ”A holistic model for an internal representation to control the movement of a manipulator with redundant degrees of freedom”, Biol. Cybernetics 79, 457-466, 1998.

[20] H. Seraji and B. Bon, ”Real-Time Collision Avoidance for Position-Controlled Manipulators”, IEEE Trans. on Robotics and Automation, Vol.15, No. 4, pp. 670-677, August 1999.

[21] F. Cheng, Y. Lu, Y. Sun, ”Window-shaped obstacle avoidance for a redundant manipulator”, IEEE Trans. on Systems, Man and Cybernetics, Part B, Vol. 28, No. 6, pp. 806-815, Dec 1998.

[22] Reinhard Diestel, ”Graph Theory”, Springer-Verlag Heidelberg, New York 2005.

[23] S. Kheradpir, J.S. Thorp , ”Real-time control of robot manipulators in the presence of obstacles”, IEEE Trans. on Robotics and Automation, Vol. 4, No. 6, pp. 687-698, Dec 1988.

[24] T. Tsuji, S. Nakayama, K. Ito, ”Distributed feedback control for redundant manipulators based on virtual arms”, Prooceeding of Autonomous Decentralized Systems, pp. 143-149, 1993.

[25] EU Project SPARK II, website online at www.spark2.diees.unict.it/MiniArm.html

[26] Hoffmann, H., M¨oller, R. , ”Unsupervised learning of a kinematic armmodel”, Artificial Neural Networks and Neural Information Processing, vol. 2714, (ICANN/ICONIP), LNCS, Kaynak O, Alpaydin E, Oja E, Xu L,Springer, Berlin, pp. 463-470, 2003.

[27] Moller R. and Hoffmann, H. ,”An extension of neural gas to local PCA”, Neurocomputing, vol. 62, 305-326, 2004.

[28] Moller, R. ”Interlocking of learning and orthonormalization in RRLSA”.Neurocomputing, vol. 49, pp. 429-433.

[29] Ouyang, S., Bao, Z., Liao, G.S. , ”Robust recursive least squares learning algorithm for principal component analysis”, IEEE Transactions on Neural Networks, vol. 11(1), pp. 215-221, 2000.

[30] Tipping, M. E., Bishop, C. M. ,”Mixtures of probabilistic principal component analyzers”, Neural Computation, vol. 11, pp. 443-482, 1999.

[31] Hinton, G. E., Dayan, P., and Revow, M. ”Modeling the manifolds of images of handwritten digits”, IEEE Transactions on Neural Networks, vol. 8, pp. 65-74, 1997.

[32] Kohonen, T. , ”Self-Organizing Maps” Springer, Berlin, 1995.

[33] Martinetz, T. M., Berkovich, S. G., and Schulten, K. J., ”Neural-Gas network for vector quantization and its application to time-series prediction”, IEEE Transactions on Neural Networks, 4, pp. 558-569, 1993.

[34] OpenCV website homepage online at http://opencv.willowgarage.com

[35] Honglak Lee, Yirong Shen, Chih-Han Yu, Gurjeet Singh and Andrew Y. Ng, “Quadruped Robot Obstacle Negotiation via Reinforcement Learning”, ICRA, 2006.

[36] Arthur P. S. Braga, Aluizio F. R. Araujo, “A topological reinforcement learning agent for navigation” Neural Comput. and Applic., vol. 12, pp. 220-236, 2003.

[37] Bram Bakker, Viktor Zhumatiy, Gabriel Gruener and J¨urgen Schmidhuber, “Quasi-Online Reinforcement Learning for Robots”, Proc. IEEE International Conference on Robotics and Automation, 2006.

[38] Kolter, Abbeel and Ng, “Hierarchical Apprenticeship Learning with Application to Quadruped Locomotion”, NIPS 2008

[39] Andrew W. Moore and Christopher G. Atkeson, “Prioritized sweeping: Reinforcement learning with less data and less time”, Machine Learning, Springer, vol. 13, pp. 103-130, 1993.

[40] Marco Dorigo and Thomas Stutzle, “Ant Colony Optimization”, MITPress, 2004.

[41] Leslie P. Kaelbling, Michael L. Littman and Andrew W. Moore, “Reinforcement Learning: A Survey”, Journal of Artilcial IntelligenceR-

esearch, vol. 4, 1996.

[42] Andrew G. Barto and Sridhar Mahadevan “Recent advances in hierarchical reinforcement learning”, Discrete Event Dynamic Systems: Theory and Applications, vol. 13, pp. 41-77, 2003.

[43] Sutton, R.G, Barto, A.G, “Reinforcement Learning: An Introduction” 2 ed. (sl): Mit Press. pp. 51-185,1998.

[44] Ribeiro, C.H.C., “A Tutorial on Reinforcement Learning Techniques” in Proc. of International Conference on Neural Networks, INNS Press, Washington, DC, USA, pp. 1-23, 1999.

[45] H. Cruse, U. Steinkuhler and Ch. Burkamp, “MMC - A recurrent neural network which can be used as manipulable body model”, From animals to animats vol. 5, R. Pfeifer, B. Blumberg, J.-A. Meyer, S.W. Wilson (eds.) MIT Press, pp. 381-389, 1998.

[46] U. Steinkuhler, H. Cruse, “A holistic model for an internal representation to control the movement of a manipulator with redundant degrees of freedom”, Biol. Cybernetics vol. 79, pp. 457-466, 1998.

[47] Tipping ME, Bishop CM (1999) “Mixtures of probabilistic principal component analyzers”, Neural Computation vol. 11, pp: 443-482, 1999.

[48] N. Abbate, A. Basile, A. Brigante, C. Faulisi, “Development of a MEMS based wearable motion capture system,” Human System Interactions (HSI), 2009.

[49] E. Foxlin, “Pedestrian tracking with shoe-mounted inertial sensors,” IEEE Comput. Graph. Appl. , vol. 25 (6), pp. 38-46, 2005.

[50] D. Roetenberg, P. Slycke, and P. Veltink, “Ambulatory position and orientation tracking fusing magnetic and inertial sensing,” IEEE Transactions on Biomedical Engineering, vol. 54, pp. 883–890, 2007.

[51] D. Roetenberg, “Inertial and magnetic sensing of human motion,” Ph.D. thesis, University of Twente, 2006.

[52] D. Roetenberg, H. Luinge, and P. Slycke, “Xsens MVN: Full 6DOF Human Motion Tracking Using Miniature Inertial Sensors,” whitechapter. Available: www.xsens.com

[53] iNEMO data brief. Available: www.st.com/inemo

[54] STM32 datasheet and reference manual. Available: www.st.com

[55] L3G4200D datasheet. Available: www.st.com

[56] LSM303DLH datasheet. Available: www.st.com

[57] G. Bishop and G. Welch, “An Introduction to the Kalman Filter,” presented at the SIGGRAPH course notes, 2001.

[58] A. M. Sabatini, “Quaternion-based strap-down integration method for application of inertial sensing to gait analysis,” Medical and-Biological Engineering and Computing, vol. 43 (1), pp. 94-101, 2005.

[59] A. M. Sabatini, “Quaternion-based extended Kalman filter for determining orientation by inertial and magnetic sensing,” IEEE Trans. Biomed. Eng., vol. 53, pp. 1346-56, Jul 2006.

[60] X. Yun, “Design, Implementation, and Experimental Results of a Quaternion-Based Kalman Filter for Human Body Motion Tracking,” IEEE Transactions on Robotics, vol. 22, no. 6, Dec. 2006

[61] D. Vlasic et al., “Practical motion capture in everyday surroundings,” ACM Transactions on Graphics (TOG), vol. 26, 2007.

[62] B. Cardani, “Optical Image Stabilization for Digital Cameras,” IEEE Control System Magazine, vol.26 (2) pp. 21-22, 2006.

[63] C. Hide and T. Moore, “GPS and low cost INS integration for positioning in the urban environment,” Proceedings of ION GPS, Long Beach, CA, USA, pp.1007-1015, 2005.

[64] Bizzi, E. “Spinal Cord Modular Organization and Rhythm Generation: An NMDA Iontophoretic Study in the Frog.” 1-18., 1998

[65] Dean, J., T. Kindermann, et al. “Control of walking in the stick insect: from behavior and physiology to modeling.” Auton Robot 7(3): 271-288., 1999

[66] Feldman, A. G. “New insights into action–perception coupling.” Experimental Brain Research.,2009

[67] Ijspeert, A. J., J. Nakanishi, et al. “Learning attractor landscapes for learning motor primitives.” Advances in neural information processing systems: 1547-1554., 2003

[68] Latash, M. Synergy., 2008

[69] Peters, J. “Machine learning of motor skills for robotics.”, 2007

[70] Pfeifer, R. and J. Bongard. “How the body shapes the way we think: a new view of intelligence.”, 2006

[71] Pilon, J.-F., S. De Serres, et al. (2007). “Threshold position control of arm movement with anticipatory increase in grip force.” Experimental Brain Research 181(1): 49-67.

[72] Schaal, S. (2010). “Dynamic Movement Primitives–A Framework for Motor Control in Humans and Humanoid Robotics.” 1-10.

[73] Schaal, S., P. Mohajerian, et al. (2007). “Dynamics systems vs. optimal control--a unifying view.” Progress in brain research 165: 425.

[74] Schaal, S., D. Sternad, et al. .“Rhythmic arm movement is not discrete.” 1-8., 2004

[75] Siciliano, B., L. Sciavicco, et al. (2009). “Robotics: Modelling, Planning and Control.” 1-644.

[76] Tedrake, R. . “Stochastic Policy Gradient Reinforcement Learning on a Simple 3D Biped.” 1-6., 2005

[77] Bernstein N. The Coordination and Regulation of Movements. Pergamon Press. New York. OCLC 301528509, 1967