A. Turetta
G. Casalino
A. Sorbara
DIST – University of Genova
Via Opera Pia 13
16145 Genova, Italy
{turetta, casalino, andrea.sorbara}@dist.unige.it

Distributed Control Architecture for Self-reconfigurable Manipulators

Abstract

In recent years self-reconfigurable modular robots have gained increasing interest from part of the international robotic community. Although recent robots of this type are characterized by advanced electro-mechanical designs, the development of their supporting control techniques have only registered strong results in the field of locomotion problems, while the manipulation capabilities of existing systems still appear to be quite limited. Aiming to provide a contribution along this latter direction, in this paper we propose a computationally distributed technique for controlling the motion of any tree-structured chain resulting from reconfiguration in its operational space. The presented strategy, which could actually be adopted when dealing with any kind of chain-based modular robotic system, turns out to be particularly well suited to self-reconfigurable structures for three main reasons: (i) it is not based on any explicit role assignment; all of the modules can be added, removed or exchanged online as required, with no impact on the overall control architecture; (ii) each module has only a very limited set of local information that must be known a priori and can be totally unaware of the remaining part of the chain; (iii) no external centralized controller is necessary; basic local processing and communication units onboard every module and a simple man–machine interface providing high-level commands are enough. A global self-coordinating behavior is automatically exhibited by the proposed technique at power-on or immediately after any configuration change as the result of a number of repeated data exchanges, performed online along the chain at every sampling interval. Although achievable performances depend on the available communication bandwidth, the convergence towards a final position error of zero is, however, always guaranteed. Moreover, because the computational burden required by every module is extremely light, the proposed technique represents an effective control solution that can be easily implemented onboard many of the low-cost and small control platforms available on existing self-reconfigurable robots.

KEY WORDS—cellular and modular robots, cooperative manipulators, distributed robot systems

1. Introduction and Related Work

Self-reconfigurable robots constitute a particular class of modular robotic systems. As with any other modular robot, they are composed of basic robotic units (each with a low number of degrees of freedom) that can be connected together in several alternative ways. The distinctive feature of a self-reconfigurable robot is its capability to *autonomously* assume the most suitable configuration for the assigned task by simply re-arranging the modules (i.e. by varying their number and/or the way they are connected together). In this way the system may operate several shape transformations during the same mission, depending on the contingent needs: when it has to pass through a narrow hole or inside a pipe, it can become a snake; when it has to climb some stairs, it may prefer moving like a multi-legged system; or when it has to grasp and transport objects it may even become a manipulator.

The idea of a hyper-versatile autonomous robotic system, capable of changing its shape and size to meet the demands of the task, is clearly very appealing and motivates the increasing interest in the topic. Among the several potential application fields, attention so far has focused mostly on sectors requiring the execution of missions within unstructured or unknown environments. Relevant examples are space missions for planetary exploration or rescue activities in scenarios following calamitous events, applications in which the presence of unforeseen obstacles or the occurrence of unexpected events make *self-adaptability* a fundamental factor for success (much more than other traditional robot performance indicators, such as motion speed and accuracy).

In order to achieve an adequate level of self-adaptability, before starting to approach higher-level problems concerning

The International Journal of Robotics Research
Vol. 27, No. 3–4, March/April 2008, pp. 481–504
DOI: 10.1177/0278364907085565

perception, understanding, reasoning and reactive planning, the research efforts of the international robotic community have so far been concentrated mainly on: (a) studies of electro-mechanical nature, aiming to realize robotic structures with increasing levels of versatility and robustness; (b) investigations on control strategies, oriented to develop and improve the two basic capabilities for a self-reconfigurable robot, locomotion and manipulation.

From the electro-mechanical point of view strong progress has been made in recent years. Two alternative design approaches have been proposed, chain-based and lattice-based[1] structures, and several performing robots of both the categories have been developed, among which ATRON (Jørgensen et al. 2004), Polybot (Duff et al. 2001), CONRO (Shen et al. 2002), M-TRAN (Murata et al. 2002; Kurokawa et al. 2003; Murata et al. 2006) and Superbot (Salemi et al. 2006) are worthy of mention. With respect to the earliest versions, systems of the current generation present highly improved features from several points of view: compact design, robust multi-face connectors, reliable docking/undocking mechanisms, increased single-module payload–weight ratio, advanced embedded sensing devices, improved inter-module communication and better power management (Murata et al. 2006; Salemi et al. 2006).

As concerns control issues, solid results have so far been obtained mainly within the framework of locomotion. Different kinds of locomotion modes have been investigated: rolling, walking, slithering, climbing and crawling; a good overview on the matter can be found in Shen et al. (2006). Different control techniques have been consequently proposed for locomotion problems: gait tables (Yim et al. 2002a,b), role-based control (Støy et al. 2002), central pattern generators (Kurokawa et al. 2003) and hormone-inspired techniques (Shen et al. 2000; Hou and Shen 2006). Almost all strategies (although with some differences) are oriented to establish "resonant" cyclic sequences of basic motions according to the specifications of the command gait. More recently (Moll et al. 2006; Shen et al. 2006) the attention has also moved onto aspects concerning robustness to perturbations and fault-recovery capability by looking for solutions guaranteeing online adaptation to the varying environmental characteristics, such as the presence of terrain with changing slopes or roughness, the presence of steps to be climbed or descended, or the presence of unforeseen obstacles that must be over-passed.

While very nice results have been achieved in the field of locomotion, there is still a lack of effective solutions for self-reconfigurable robots within the framework of manipulation problems. Such a limitation is not too critical for lattice-based structures (whose manipulation potentialities are in any case quite limited[2]), but it certainly represents a crucial issue for chain-based robots, which otherwise could be effectively employed as hyper-flexible manipulators. Indeed, to the best of the authors' knowledge, only a few very basic manipulation tasks have been performed by chain-based modular structures, such as pushing objects while moving on a surface or digging (Yim et al. 2001). No other significant examples of operations involving external objects, such as grasping, transportation or assembly activities, have so far been executed by chain-based self-reconfigurable robots. Naturally enough the same consideration can be extended to also include the control solutions proposed for performing self-reconfiguration[3]. Even if some successful experimental results on autonomous docking have been obtained in different operative situations (Shen and Will 2001; Yim et al. 2001; Murata et al. 2006), the more general case of a three-dimensional (3D) chain-based structure capable of performing any desired change in its shape by connecting any pair of its composing modules has not yet been considered from a sufficiently general perspective.

One of the main motivations behind such limitations is strictly related to the difficulties encountered so far in precisely controlling the positioning, as well as the associated motions, of the terminal frames of a generic modular chain in its operational space. The control functionalities offered by the set of embedded controllers enabling locomotion are indeed not sufficient to allow the end-effectors of a generic structure to move in position and/or orientation towards a given 3D location (for example, to reach an object to be grasped or other modules to be docked for re-configuration purposes). More specifically, when the required task (as is the case for a locomotion task) consists of making every module execute a periodic motion (defined *a priori* and, at most, just tuned slightly online), basic joint-level control functionalities may be enough. When the given task (as is the case for a manipulation task) is instead expressed in Cartesian space coordinates, a preliminary problem has to be solved by taking into account the current kinematic structure, i.e. transforming the required end-effectors motions into a proper set of control actions in the joint domain.

Consolidated techniques for this kind of problem do exist for centralized controlled robots and have appeared in the manipulation literature since the late 1980s. Several alternative online control solutions have been proposed over the years, and a number of additional functionalities have been investigated and developed, such as strategies for singularity detection and management (Chiaverini 1997), algorithms for

1. Modules of the first category can form kinematic chains, which can move and behave like a snake or a manipulator. Modules of the second category are always located in points of a grid structure (*lattice*), like atoms in a crystal, and can move only to neighboring positions in the lattice by attaching to (and detaching from) adjacent modules.

2. A lattice-based robot might grasp and manipulate objects only through several successive docking–undocking phases. In addition, the position constraints imposed on modules by the lattice structure introduces a sort of discretization of the workspace, which further limits the manipulation capabilities.

3. Self-reconfiguration for chain-based robots can be considered as a particular kind of manipulation task where the system performs manipulation on its own modules rather than interacting with external objects.

dealing with redundancy (Chiacchio et al. 1991) and hyper-redundancy (Chirikjian and Burdik 1994), strategies enabling the priority-based simultaneous execution of multiple tasks along the same kinematic chain (Nakamura 1991) and methods for preserving path-following capabilities under joint limits (Antonelli et al. 2003). Despite their differences, all such solutions share a common element: the already mentioned fact that they are based on centralized control systems performing computations regarding the overall structure (typically based on kinematic or dynamic inversion). As a consequence, none of the above techniques is particularly well suited to be directly transferred to self-reconfigurable robots. Indeed a control solution based on centralized units suffers from a lack of scalability (when applied to complex chains), is characterized by low fault-tolerance (in the case of a failure of the unique controller) and reduces the flexibility of the system (because any module re-arrangement would require all of the structural parameters of the controller, such as Jacobian matrices, inertia tensors and so on, to be updated). For these reasons, it is almost unanimously agreed that any control strategy for self-reconfigurable robots may offer significant benefits if and only if it is obtained from a decentralized approach and if it can be implemented by just exploiting the processing and communication capabilities available onboard every robotic module.

Therefore by focusing attention on modular systems equipped with simple embedded "processing and communication units" (PCUs), the present work proposes a distributed control technique aiming to sensibly improve the capabilities of execution of both manipulation and self-reconfiguration tasks within the class of chain-based modular robots.

The main characteristics of the proposed method can be summarized as follows: (a) it is not based on any explicit role assignment; all of the modules are identical from the control point of view and can therefore be added, removed or exchanged as required online, with no impact on the overall control architecture; (b) each module has only a very limited set of local information that must be known *a priori* (such as its dimensions and the number and location of its degrees of freedom) and can be totally unaware of the remaining part of the chain and its topological (and time-varying) structure; (c) no external centralized controller is consequently required for continuously updating the overall robot geometry and kinematics; the proposed technique just requires basic local PCUs onboard every robotic module and a simple man–machine interface providing high-level commands and allowing their execution to be monitored.

A global self-organizing behavior is then automatically established and propagated along the chain at power-on and after any module re-arrangement, through repeated high-rate data exchanges. The efficient self-coordination of the modules is indeed obtained through a (sufficiently high) number of *iterations* performed at every sampling interval among all of the PCUs in the chain. Although the achievable performances in terms of trajectory tracking capabilities consequently depend on the available communication bandwidth, we show how the convergence of the final position error of the end-effectors to zero is, however, always guaranteed[4].

In addition, as concerns technological and implementation aspects, we also show how the amount of local computation performed in every iteration by a single PCU is actually very limited, as is the quantity of data to be exchanged by two adjacent units. As a consequence the overall required processing effort (including both computation and communication times) to guarantee satisfactory motion performances is actually comparable with what is required by any centralized control procedure. Furthermore, results from both simulation and experiments show that valuable improvements in performance can be obtained even with a restricted number of iterations.

Therefore, in light of all of the previous considerations, it should appear evident how the technique proposed here actually represents a very interesting control solution for any kind of self-reconfigurable chain-based robotic structure. Indeed the presented control strategy, in addition to enabling the execution of generic manipulation tasks, also provides the fundamental advantage of permitting the efficient execution of (possibly complex) reconfiguration maneuvers. As a matter of fact the possibility of precisely controlling the 3D motion of any kinematic chain enables its end-effectors to reach any module that has to be docked. In addition the topological changes consequent to any module re-configuration are automatically dealt with by the proposed architecture thanks to its immediate self-organizing property. Finally the very limited computational burden allows an easy implementation of the technique onboard many of the low-cost and low-size control platforms typically available on most existing self-reconfigurable robots.

The present paper is in the line of some other recent works of the authors on the subject. The earliest concerned the decentralized coordination of interconnected sub-chains (generally used for composing multi-arm mobile manipulators (Casalino and Turetta 2004)), each one assumed separately governed by its own centralized controller. Successively, in Casalino and Turetta (2005) the problem of self-coordination within modular atomic units was approached for the first time and a solution based on the same iterative technique presented here was proposed, even when referring to open linear kinematic chains only. In parallel with the above works, an alternative distributed control solution has been devised by applying traditional dynamic programming techniques to the kinematic inversion problem (Casalino et al. 2006). Such a technique, although almost always allowing the achievement of optimal control laws for both serial and branched structures, is obtained by paying a price in terms of increased processing and communication burdens. As a consequence, when applied within the framework of self-reconfigurable robots, it appears to be a worse solution than the iterative technique proposed here, which is instead

4. Provided that the goal frames are located inside the reachable workspace of the considered kinematic structure.

certainly well suited to implementation on architectures with limited processing and communication resources.

Moving on from this last consideration, the present paper aims to extend the preliminary results described in Casalino and Turetta (2005) by presenting interesting simulative and experimental developments and by providing a theoretical extension also allowing open tree-structured chains to be included in the algorithmic framework.

To this aim the work is organized as follows. As the proposed distributed control architecture has been obtained by "distributing" the computations required for controlling a conventional (i.e. non-modular) manipulator, the centralized control scheme typically employed for governing manipulators is first recalled in the next section. Then, in Section 3, we start the process of transformation of the centralized scheme into an equivalent distributable system by focusing attention on the kinematic inversion problem, because it represents the most crucial and innovative part of the work, thus deserving a dedicated deep analysis. Next, the associated distributed control architecture is presented in Section 4, when employed in controlling modular serial chains. Subsequently the obtained results are extended in Section 5 to the more general case of modular tree-structured chains. Section 6 then describes a set of simulative and experimental results and, finally, Section 7 draws some conclusions, together with some indications about current and future work activities.

2. Centralized Control of a Traditional Manipulator

As the computationally distributed control architecture proposed here has been devised by "distributing" the computations required for controlling a conventional (i.e. not modular) manipulator among different processing units, it is convenient to start by first briefly recalling the basic features of the kinematic-based centralized controller typically employed for governing the operational space motion of robotic manipulators.

To this aim consider a generic manipulator, for instance similar (without loss of generality) to the commonly used manipulator with seven degrees of freedom sketched in Figure 1, where the wrist is constituted by a rotational joint (typically of Euler or roll–pitch–yaw type) with three degrees of freedom. In Figure 1, frame $<g>$ represents the "goal frame", which has to be reached (in position and orientation) by the so-called "tool frame" $<t>$, rigidly attached to the "end-effector frame" $<e>$.

Positions (described through vectors $p \in \Re^3$) and attitudes (described through rotational matrices $R \in \Re^{3x3}$) of frames $<g>$ and $<t>$ with respect to an inertial "world frame" $<w>$ are respectively embedded within the transformation matrices

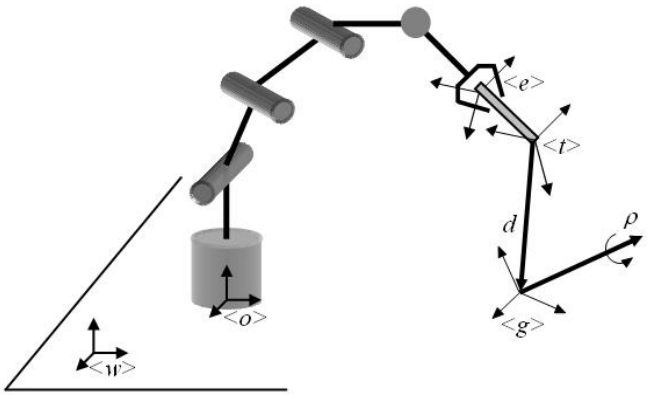


Fig. 1. Generic non-modular manipulator.

$$
{}^{w}_{g}T(t) = \begin{bmatrix} {}^{w}_{g}R(t) & p_g(t) \\ 0_{1x3} & 1 \end{bmatrix};
$$

$$
{}^{w}_{t}T(q) = \begin{bmatrix} {}^{w}_{t}R(q) & p_t(q) \\ 0_{1x3} & 1 \end{bmatrix}. \tag{1}
$$

The former is given as an external time-varying reference input, while the latter, depending on actual posture, is real-time computed as the result of the product

$$
{}^{w}_{t}T = {}^{w}_{0}T\,{}^{0}_{e}T\,(q)\,{}^{e}_{t}T, \tag{2}
$$

where ${}^{e}_{t}T$ and ${}^{w}_{o}T$ (the transformation matrices of $<t>$ with respect to $<e>$ and of $<0>$ with respect to $<w>$, respectively) are given as external configuration data, while ${}^{o}_{e}T\,(q)$ (the transformation matrix of $<e>$ with respect to $<0>$) is real-time evaluated by means of the knowledge of robot geometry and of the current joint position vector q.

From the knowledge of ${}^{w}_{g}T$ and ${}^{w}_{t}T$ it is easy to compute the position error vector (projected on world frame $<w>$):

$$
e \doteq \begin{bmatrix} \rho \\ d \end{bmatrix} = \begin{bmatrix} \rho \\ p_g - p_t \end{bmatrix}, \tag{3}
$$

where $\rho \in \Re^3$ is the misalignment error vector (expressed in "eigen-axis" notation), computed by application of the so-called "versor lemma" (Aicardi et al. (1995)), and $d \in \Re^3$ is the linear error vector of frame $<t>$ with respect to $<g>$. Then by letting (with a small abuse of notation owing to the use of the "$\cdot$" upper sign)

$$
\dot{x}_g \doteq \begin{bmatrix} \omega_g \\ v_g \end{bmatrix}; \quad \dot{x}_t \doteq \begin{bmatrix} \omega_t \\ v_t \end{bmatrix} \tag{4}
$$

be the collections of (projected on $<w>$) angular and linear velocities, of frame $<g>$ and $<t>$, respectively (with the former, if available, it also given as a real-time reference input),

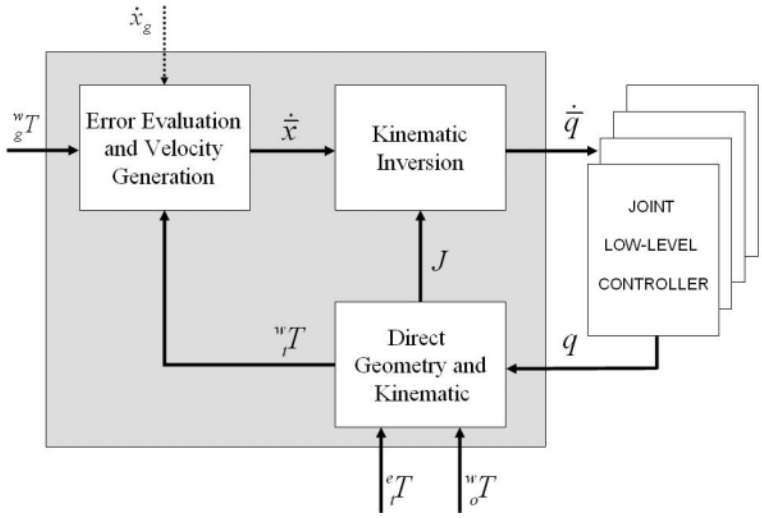


Fig. 2. Centralized control architecture.

we can immediately see, by considering the (tentative) Lyapunov function

$$V \doteq \frac{1}{2} e^T e \quad \Rightarrow \quad \dot{V} \doteq -e^T(\dot{x}_t - \dot{x}_g), \tag{5}$$

that an assignment $\dot{\bar{x}}$ to the absolute tool velocity $\dot{x}_t$ of the form

$$\dot{\bar{x}} \doteq \gamma\, e + \dot{x}_g; \quad \gamma > 0 \tag{6}$$

would actually drive $<t>$ towards $<g>$ asymptotically.

The desired tool frame velocity, $\dot{\bar{x}}$, then has to be transformed, via kinematic inversion, into one of the corresponding (if any) joint velocity reference vectors $\dot{\bar{q}}$ capable of maximally satisfying, at all times, the condition

$$\dot{\bar{x}} \doteq J(q)\dot{\bar{q}}, \tag{7}$$

where $J(q)$ is the Jacobian matrix of $<t>$ with respect to $<w>$ (with output velocities projected onto $<w>$).

The set of joint velocity references, solving (7), is finally provided to the set of joint low-level controllers which, in the scope of present paper, are assumed to be capable of working "ideally", thus guaranteeing the perfect fulfillment of any velocity reference signal.

The resulting centralized kinematic-based control scheme is reported in the gray rectangle of Figure 2, organized into three main functional blocks.

(i) The *error evaluation and velocity generation* module, which is in charge of computing position error vector e, through (3) and the versor lemma, as well as the tool frame velocity reference vector $\dot{\bar{x}}$ via (6).

(ii) The *direct geometry and kinematic* module, which is used for real-time evaluation of both the transformation matrix ${}^{w}_{t}T$ via (2) and the Jacobian matrix $J(q)$ via the product

$$J(q) \doteq S_{t/e} J_{e/0}(q), \tag{8}$$

where $J_{e/0}(q) \in \Re^{6xn}$ is the so-called "basic" Jacobian matrix of $<e>$ with respect to $<0>$ (with output velocities projected on $<0>$) and where $S_{t/e} \in \Re^{6x6}$ represents the instantaneous rigid body velocity transformation matrix from frame $<e>$ to frame $<t>$ (input velocities projected on $<0>$, output velocities projected on $<w>$), easily computable via transformation matrices ${}^{e}_{t}T$ and ${}^{o}_{e}T(q)$.

(iii) The *kinematic inversion* module, which is in charge of solving (7). At this level different techniques may be employed, depending on the performance criterion to be maximized. One of the most efficient and typically employed solutions is obtained via singular value decomposition (SVD), by computing a damped least-squares regularized pseudo-inversion of the Jacobian matrix $J(q)$ (Nakamura (1991)):

$$\dot{\bar{q}} \doteq J^{\#}\dot{\bar{x}}. \tag{9}$$

Before concluding this section it is worth noting how, among the three functional modules, only the last two need to perform distributed computations, since the *error evaluation and velocity generation* module is employed to generate Cartesian velocity reference for the tool frame and therefore its role is, and has to remain, centralized. The other two modules instead have to be translated into a distributable form and, from this perspective, the *direct geometry and kinematic* module is not critical, because the computation of the feedback matrix

${}^{o}_{e}T(q)$ can also be easily executed as the product of elementary transformation matrices, each one at most depending on a single joint variable; i.e. more precisely:

$$ {}^{0}_{e}T \doteq {}^{0}_{1}T(q_1)\,{}^{1}_{2}T(q_2)\,\ldots\,{}^{n-1}_{n}T(q_n)\,{}^{n}_{e}\bar{T}. \tag{10} $$

Similarly it is shown in later sections how the computation of Jacobian matrix (8) can also be easily executed in a distributed manner.

The only difficult part to be distributed is hence represented by the *kinematic inversion* module, as computations of an intrinsically centralized nature are required to perform a matrix pseudo-inversion as in (9). For this reason one of the main contributions of the present work is therefore represented by the definition of an alternative computationally distributable method for implementing a kinematic inversion procedure: this is the subject of the next section.

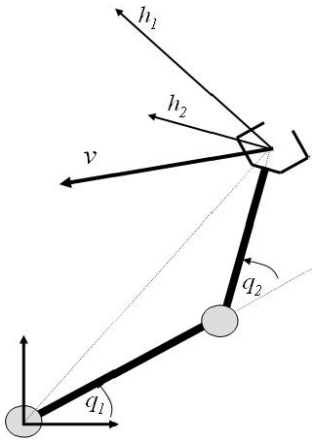


Fig. 3. A 2D planar manipulator.

3. An Iterative Algorithm for Kinematic Inversion

As mentioned above, a computationally distributable iterative technique is proposed in this section as an effective method for obtaining a solution to the kinematic inversion problem (7), an alternative to the centralized form (9). The very simplified case of a 2D linear velocity task performed by a planar chain with two degrees of freedom is proposed as a preliminarily introductory example, because it admits a nice graphical representation. The extension to the more general case of a 3D serial chain performing six-dimensional velocity tasks (thus, including linear and angular velocity component terms) is then given in Section 3.2, while a formal proof of convergence of the proposed algorithm is provided in Section 3.3.

In order to emphasize the distributable essence of the proposed algorithm from the beginning, the presence of n separate "virtual processing units" (VPUs) is assumed in the current section. Although all VPUs are running the same unique control platform (as independent software processes), each of them is assumed to be associated and devoted to only a single degree of freedom. Therefore, it only has to find the local control action for its corresponding joint. Furthermore, it is only allowed *a priori* knowledge of the geometric and kinematic parameters of the sole associated robotic module, and has to exchange data with other peers for coordination purposes.

3.1. 2D Manipulator Performing Two-component Linear Velocity Tasks

Consider, as an introductory example, the manipulator with two degrees of freedom sketched in Figure 3, whose tool (assumed coincident with the end-effector) is allowed to move just on the plane of the paper. In Figure 3, h_1 and h_2 represent the vectors along which the instantaneous velocity contributions to the tool provided by joint 1 and joint 2, respectively, are directed[5].

When the tool of the manipulator is commanded to move with a generic linear velocity $v \in \Re^2$, the proposed distributable procedure requires that, within every sampling interval, the available VPUs perform several pipelined processing phases (*iterations*) in succession. For this motivation, hereinafter all of the variables which are computed during iterations will be denoted with two lower-right indices: the first referring to the ordinal number of the joint in the chain and the second referring to the number of the current iteration.

Going into the details of algorithm, at the first iteration the overall velocity vector v is assigned as a reference input solely to one of the two VPUs, for example that associated to joint 1. As mentioned before, because every VPU only has local knowledge of the robotic structure, it is unaware of the presence of other VPUs and therefore it tries to accomplish the required task by itself, without expecting any contributions from other peers. As a consequence the problem that VPU 1 has to solve at the first iteration consists of finding out the scalar joint velocity term $\dot{\bar{q}}_{1,1}$ maximally satisfying pseudo-equality $v \approx h_1 \dot{\bar{q}}_{1,1}$. For such a of problem it is well known that the best solution is represented by a least-squares form

$$ \dot{\bar{q}}_{1,1} \doteq h_1^{\#} v, \tag{11} $$

where the pseudo-inverse row vector

$$ h_1^{\#} \doteq h_1^T / \left(h_1^T h_1\right) \tag{12} $$

in (11) plays the same role as the matrix $J^{\#}$ in (9).

5. In other words, vectors h_1 and h_2 constitute the columns of the 2×2 Jacobian matrix of the robot.

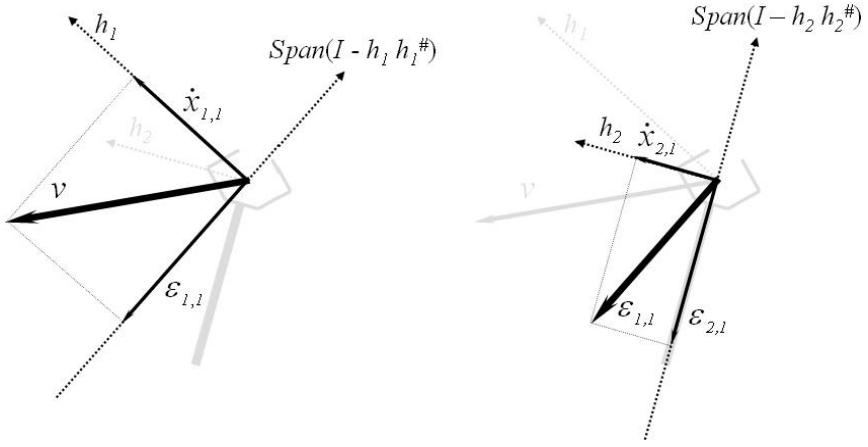


Fig. 4. Velocity contribution and velocity error vectors computed at the first iteration by (a) VPU 1 and (b) VPU 2.

Although solution (11) guarantees a minimum-norm error, such an error vector is not generally the null one; indeed the resulting end-effector velocity induced by $\dot{\tilde{q}}_{1,1}$; that is

$$\dot{x}_{1,1} \doteq h_1 \dot{\tilde{q}}_{1,1}, \tag{13}$$

is generally (as in the case at hand, see Figure 4(a)) different from the required velocity. Therefore, a velocity error vector (corresponding to the component part of the original task which is still unaccomplished) is originated:

$$\varepsilon_{1,1} \doteq v - \dot{x}_{1,1} = \left(I - h_1 h_1^{\#}\right) v \tag{14}$$

and can be easily evaluated by VPU 1 via definition (14).

Once computed, vector $\varepsilon_{1,1}$ is then sent, as a reference input, to the second VPU, invoking for its corrective contribution (see Figure 4(b)). Then, since VPU 2 acts in the same manner as VPU 1, it also finds out its corresponding scalar joint velocity term by solving a least-squares problem, thus obtaining

$$\dot{\tilde{q}}_{2,1} = h_2^{\#} \varepsilon_{1,1} \tag{15}$$

(where pseudo-inverse row vector $h_2^{\#}$ has been obtained by applying (12) to vector h_2). Although $\dot{\tilde{q}}_{2,1}$ induces a tool velocity term, $\dot{x}_{2,1}$, contributing to the execution of the original velocity task, part of v, corresponding to the error vector

$$\varepsilon_{2,1} \doteq \varepsilon_{1,1} - \dot{x}_{2,1} = \left(I - h_2 h_2^{\#}\right) \varepsilon_{1,1}, \tag{16}$$

still remains unaccomplished. Therefore, with the aim of fulfilling such a residual part, a second iteration, identical to the first that has just concluded, can be started by asking VPU 1 to provide a tool velocity contribution maximally similar to vector $\varepsilon_{2,1}$.

To this aim, VPU 1 (via the application of (11) to input $\varepsilon_{2,1}$) finds a second joint velocity term $\dot{\tilde{q}}_{1,2}$, which provides an additional tool velocity contribution, $\dot{x}_{1,2}$, and leads to an error vector $\varepsilon_{1,2}$ (see Figure 5(a)). This is therefore sent as a reference vector to VPU 2, which in turn finds $\dot{\tilde{q}}_{2,2}$, inducing $\dot{x}_{2,2}$ and leaving the unaccomplished part $\varepsilon_{2,2}$ (see Figure 5(b)). A number of other iterations (say K) can then be performed in the same manner, before the end of the overall external sampling interval is reached. At that point every VPU evaluates the total joint velocity reference by summing up the set of terms computed during all of the iterations; namely

$$\dot{\tilde{q}}_i = \sum_{k=1}^{K} \dot{\tilde{q}}_{i,k}; \quad i = 1, 2. \tag{17}$$

The resulting joint velocity reference $\dot{\tilde{q}}_i$ is then sent to the underlying low-level joint controller, and the overall procedure can be started again, working on a new tool velocity reference vector v.

Before concluding this section it is worth commenting on Figure 6, where all of the relevant vectors computed during iterations are sketched. More precisely: black solid arrows represent the tool velocity contributions provided by joint 1 (i.e. vectors $\dot{x}_{1,k}$); gray solid arrows (vector v apart) represent the tool velocity contributions provided by joint 2 (i.e. vectors $\dot{x}_{2,k}$); horizontal dotted arrows represents the error vectors computed by VPU 1 (i.e. vectors $\varepsilon_{1,k}$); and oblique light-grey dotted arrows represents the error vectors computed by VPU 2 (i.e. vectors $\varepsilon_{2,k}$).

By now arranging the error vectors according to the chronological order they are computed, and by considering the resulting sequence $\{\varepsilon_{1,1}, \varepsilon_{2,1}, \varepsilon_{1,2}, \varepsilon_{2,2}, \ldots, \varepsilon_{1,k}, \varepsilon_{2,k}, \varepsilon_{1,k+1}, \ldots\}$, from Figure 6 it should appear evident how any processing phase actually contributes to reduce the norm of the unaccomplished part of the original velocity task. This fact should not be too surprising as, by analyzing expressions (14) and (16), it is easy to see that every error vector actually represents the

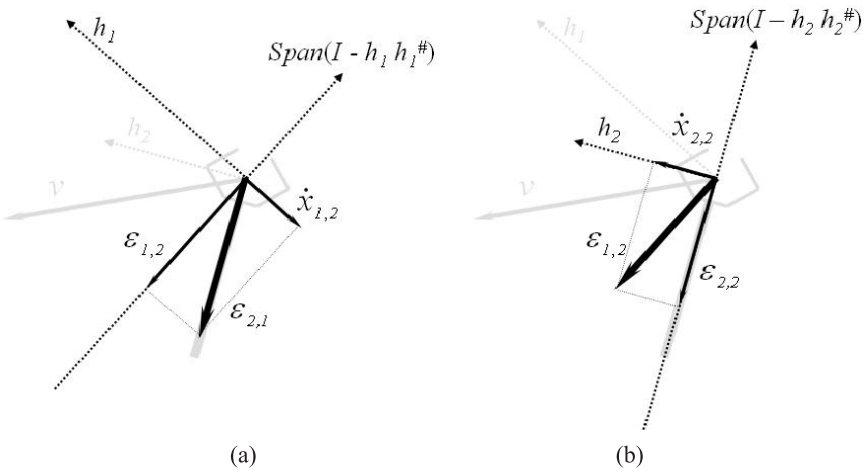


Fig. 5. Velocity contribution and velocity error vectors computed at the second iteration by (a) VPU 1 and (b) VPU 2.

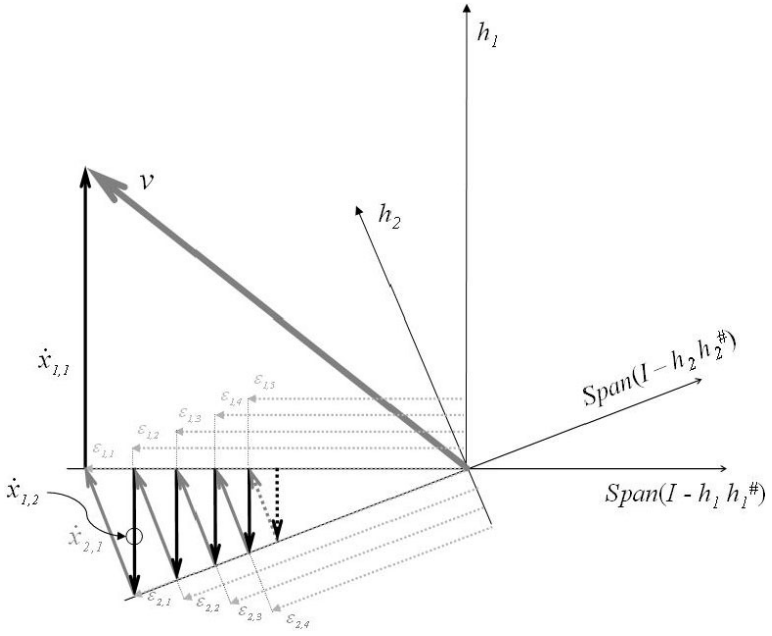


Fig. 6. Velocity contribution and velocity error vectors computed during several iterations.

orthogonal projection of the previous vector in the sequence onto a specific subspace of the task space. More specifically, for the case at hand, the task space is $\Re^2$ and the subspaces on which VPU 1 and VPU 2 project their input errors are those orthogonal to h_1 and h_2, respectively.

As a consequence, although the conditions guaranteeing the convergence of the proposed technique are analyzed more formally in Section 3.3, it can already be reasonably argued that the proposed algorithm always assures the non-increasing behavior of the norm of the error vectors.

3.2. 3D Manipulator Performing Six-component Velocity Tasks

With the above preliminary considerations in mind, we shift attention back to the original and more general case of a 3D manipulator (such as that depicted in Figure 1) whose tool frame is asked to accomplish a generic six-component velocity task $\dot{\bar{x}}$. This general case differs from the special case dealt with in the previous section in two main ways: the enlarged task dimension (six instead of two) and the increased number of

VPUs (equal to the number of degrees of freedom, now assumed to be greater than six). As will be pointed out in the following, despite such differences, the problem of finding a distributable kinematic inversion procedure can be stated and solved by adopting an approach that is very similar to that used previously. To this aim, first expand the Jacobian matrix J into its n columns:

$$J = \begin{bmatrix} h_1 & h_2 & \dots & h_n \end{bmatrix} \tag{18}$$

and consequently remember how each h_i vector, also in this case (although now belonging to $\Re^6$), actually represents the contribution to the instantaneous tool frame velocity given by the corresponding scalar joint velocity $\dot{q}_i$.

Then define the corresponding pseudo-inverse row vectors:

$$h_i^{\#} = h_i^T / \left(h_i^T h_i\right); \quad i = 1, \dots, n, \tag{19}$$

which are used by every ith VPU for computing at every kth iteration its corresponding partial joint velocity term:

$$\dot{\tilde{q}}_{i,k} = h_i^{\#} \dot{\tilde{x}}_{i,k} \tag{20}$$

(with vector $\dot{\tilde{x}}_{i,k}$ being the reference input provided to the ith VPU at the kth iteration).

Finally, since, through the application of (20), the resulting tool velocity induced by the ith joint at the kth iteration

$$\dot{x}_{i,k} \doteq h_i \dot{\tilde{q}}_{i,k} \tag{21}$$

is generally different from the required velocity, $\dot{\tilde{x}}_{i,k}$, a velocity error vector may be defined (as done before) as follows:

$$\varepsilon_{i,k} \doteq \dot{\tilde{x}}_{i,k} - \dot{x}_{i,k} = (I - h_i h_i^{\#}) \dot{\tilde{x}}_{i,k}. \tag{22}$$

As can be easily realized the internal function of every VPU is substantially the same as described in the previous section. The major difference is now represented by the presence of more than two VPUs, a fact that introduces an issue concerning the order in which the VPUs have to be involved during iterations. As indicated before (and more rigorously pointed out in the next section) the convergence property of the proposed iterative technique is the result of successive orthogonal projection operations, guaranteeing that the norm of error vectors is always decreased (or, at least, never increased) after every processing phase. As a consequence the order in which the projections are performed during the iterative phase is irrelevant to convergence. Therefore, several alternative choices can be made at this level: the order may be constant or time-varying; the iterations may be performed by all of the available VPUs or just by a proper subset of them (provided that this is recognized as sufficient to accomplish the required task).

Among the several possible choices, that adopted here corresponds to the most natural, consisting of starting with the first unit and then successively exploring all of the other

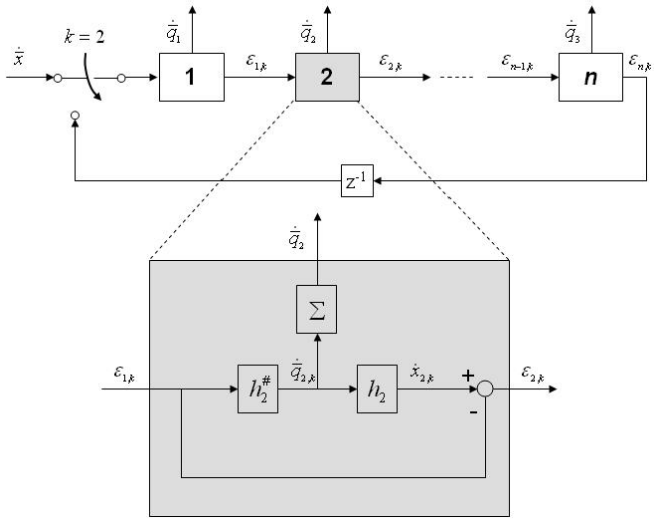


Fig. 7. Iterative centralized kinematic inversion module.

VPUs by following the topological order of their corresponding joints along the chain. Such a choice leads to the control scheme of Figure 7, where every box represents a single virtual processing unit.

By analyzing Figure 7, the overall iterative procedure should become clearer. At the beginning of every sampling interval (when k, which is the counter of iterations, is equal to one), the values of control actions computed in the previous interval (after having been assigned to low-level joint controllers) become zero:

$$\dot{\tilde{q}}_i = 0; \quad i = 1, 2, \dots, n \tag{23}$$

and the original task velocity reference vector is sent to VPU 1:

$$\dot{\tilde{x}}_{1,1} = \dot{\tilde{x}}. \tag{24}$$

Then, at every kth iteration, every ith VPU first computes its joint velocity term according to control law (20) and sums the obtained result to the previously computed terms

$$\dot{\tilde{q}}_i = \sum_{j=1}^{k} \dot{\tilde{q}}_{i,j}. \tag{25}$$

Then it computes the resulting velocity error vector via definition (22). In case $\varepsilon_{i,k}$ turns out to be the null vector, the overall procedure is terminated. Otherwise such an error is considered as a new reference vector for the next unit in the sequence. Therefore, in case $i < n$, it is sent to the adjacent VPU; this means

$$\dot{\tilde{x}}_{i+1,k} = \varepsilon_{i,k} \tag{26}$$

or it is sent to the first VPU, in case $i = n$; namely

$$\dot{\tilde{x}}_{1,k+1} = \varepsilon_{n,k} \tag{27}$$

thus starting a new iteration.

Before proceeding further with a more detailed convergence analysis, it is worth remarking how the proposed iterative procedure actually works under the assumption that a *certain* number of processing iterations is executed before the end of every sampling interval is reached. However, it should be noted how the computations executed by a single unit at every iteration are of very simple nature. The joint velocity term $\dot{\tilde{q}}_{i,k}$ and output error $\varepsilon_{i,k}$ are indeed obtained via three simple operations[6] performed on the input $\dot{\tilde{x}}_{i,k}$ and on vectors h_i, $h_i^{\#}$ (which remain constant during all of the iterations). Furthermore, since the amount of data exchanged at every iteration by two VPUs is very limited (consisting of just the six components of the error vector), the required single-iteration processing effort (including both computation and communication times) is quite low, thus allowing the current technology to easily support a *significant* number of high-rate iterations, as better shown in section 4.

3.3. Convergence Analysis

In order to demonstrate the effectiveness of the proposed technique, it is convenient to analyze the behavior of the sequence $\{\varepsilon_{n,k}\}$, whose generic term represents the overall task velocity error evaluated at the end of the kth iteration.

To this aim, once the orthogonal projection matrices

$$P_i \doteq h_i h_i^{\#}; \quad i = 1, 2, \ldots, n \tag{28}$$

are introduced, the previous relationship (22) can be conveniently rewritten in the following more compact form

$$\varepsilon_{i,k} = (I - P_i^{\#})\dot{\tilde{x}}_{i,k}; \quad i = 1, 2, \ldots, n. \tag{29}$$

Then, by also taking into account expressions (24), (26) and (27), it is not hard to extract the following recursive expression for the $\{\varepsilon_{n,k}\}$ sequence

$$\begin{cases} \varepsilon_{n,k} = \left[\prod_{i=1}^{n}(I - P_i)\right]\varepsilon_{n,k-1} \doteq A_n \varepsilon_{n,k-1}, \\ \varepsilon_{n,0} = \dot{\tilde{x}}, \end{cases} \tag{30}$$

which in turn allows us to express the dependency existing between the given velocity task and the resulting error vector $\varepsilon_{n,k}$ at the end of the kth iteration

$$\varepsilon_{n,k} = A_n^k \dot{\tilde{x}}. \tag{31}$$

6. The scalar product in (20), the vector–scalar multiplication in (21) and the vector difference in (22).

As it instead concerns a generic intermediate error vector $\varepsilon_{i,k}$, it is also an easy matter to verify how the previous relationship (31) actually generalizes as follows

$$\varepsilon_{i,k} = A_i \varepsilon_{i,k-1} = A_i^{k-1} B_i \dot{\tilde{x}} \tag{32}$$

with the generic matrix A_i directly obtainable from matrix A_n by properly commuting its factors and with the generic matrix B_i given by the expression

$$B_i \doteq \left[\prod_{j=1}^{i}(I - P_j)\right]. \tag{33}$$

Finally, by considering expressions (20) and (25), the ith joint velocity after k iterations takes on the following form:

$$\dot{\tilde{q}}_i \doteq \sum_{j=1}^{k} \dot{\tilde{q}}_{i,j} = h_i^{\#} \sum_{j=1}^{k} \varepsilon_{i-1,j} = h_i^{\#} \left(\sum_{j=1}^{k} A_{i-1}^{j-1}\right) B_{i-1} \dot{\tilde{x}}, \tag{34}$$

where, for notational consistency, it has been posed

$$A_0 \doteq A_n; \quad B_0 \doteq I; \quad \varepsilon_{0,j} \doteq \varepsilon_{n,j-1}. \tag{35}$$

At this point, by noting that all A_i matrices are given by the product of n (differently ordered) orthogonal projectors $(I - P_i)$ and by calling Π the intersection space of the subspaces spanned by $(I - P_i)$ matrices, namely

$$\Pi \doteq \bigcap_{i=1}^{n} \mathrm{Im}\{I - P_i\}, \tag{36}$$

it is possible to state that all A_i matrices correspond to n nondegenerate contraction mappings, provided that

$$\Pi = \phi. \tag{37}$$

Then, under fulfillment of condition (37) it immediately follows, from (31), that the sequence $\{\varepsilon_{n,k}\}$ will certainly converge to zero for an increasing number of iterations, thus guaranteeing, in the case of an infinite iterations, the total accomplishment of the assigned velocity task $\dot{\tilde{x}}$. Naturally enough, for a finite number of iterations, a sufficiently good approximation of $\dot{\tilde{x}}$ is obtained.

Instead, in case condition (37) is not fulfilled, it may happen that a component of $\dot{\tilde{x}}$ (say η) belonging to Π actually exists. In such a situation, despite η passing unaltered through (31) (as well as through all of its generalization (32)), it is however certainly blocked by the $h_i^{\#}$ terms appearing in (34), owing to the fact that every row vector $h_i^{\#}$ is orthogonal to Π and consequently to η. Moving on from this consideration, it can therefore be stated that, even in case of non-fulfillment of (36), any risk of joint velocity divergence is certainly avoided.

The non-fulfillment of condition (36) may occur in two cases: when a defective manipulator (i.e. with a number of

degrees of freedom lower than the dimension of the task) is asked to move along some unfeasible direction or in correspondence of the exact attainment of a singular posture. In this latter case, the absence of risk of divergences just described allows us to conclude that the proposed recursive algorithm for kinematic inversion is also characterized by an intrinsic *self-regularization* property, which is generally not *a priori* exhibited by any other centralized procedure.

This latter consideration may be extended to also include situations in which a singular posture is just approached by the robotic structure. As is well known, in such a case, the actual risk (typical of any kinematic inversion procedure) is that the achieved joint velocities will be too high, owing to the presence (induced by the nearby singular configuration) of a direction along which the tool frame exhibits motion capabilities that become progressively reduced as the arm gets closer to a singular posture.

However, also in such circumstances, the proposed technique behaves effectively. Indeed, even if the external velocity reference vector $\dot{\bar{x}}$ presents a component part (again say η) directed along such a "quasi-forbidden" direction, the effect induced by component η on every $\dot{\bar{q}}_{i,k}$ term is certainly always quite limited, since all of the $h_i^{\#}$ vectors in expression (36) are "quasi-orthogonal" to η. As a consequence, because a finite limited number of iterations is in any case performed (as unavoidably imposed by existing limitations in the channel bandwidth), any risk of joint velocities increasing indefinitely is certainly avoided.

In order to better understand the meaning of this last consideration, consider again the 2D manipulator of Figure 3, when it occasionally approaches a "quasi-orthogonal" posture (i.e. when it is almost totally stretched or folded). In such a case the vectors h_1 and h_2 become "quasi-collinear" (and the same obviously holds for their orthogonal directions spanned by matrices P_1 and P_2); consequently, every term $\dot{x}_{1,k}$ and $\dot{x}_{2,k}$ (successive to $\dot{x}_{1,1}$) becomes a very short vector, as can be easily realized by considering the diagram of Figure 6 corresponding to the scenario at hand. Therefore, all of the terms $\dot{\bar{q}}_{i,k}$ (successive to $\dot{\bar{q}}_{1,1}$) also become very small scalars and, hence, by collecting a finite number of them, any risk of obtaining joint velocity references that are too high is certainly avoided.

4. A Self-coordinating Distributed Architecture

By taking into account the previously devised iterative algorithm for kinematic inversion, we can now shift our attention back towards the original problem of controlling the motion of self-reconfigurable modular manipulators through a totally decentralized control architecture.

To this aim, the case of a. serial chain with n degrees of freedom composed of a set of atomic robotic modules with only one degree of freedom, as depicted in Figure 8, is considered here (without loss of generality) as a representative example. With reference to Figure 8, for every ith robotic unit

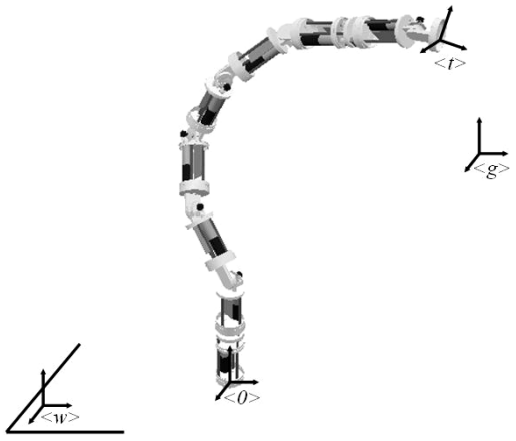


Fig. 8. Modular manipulator.

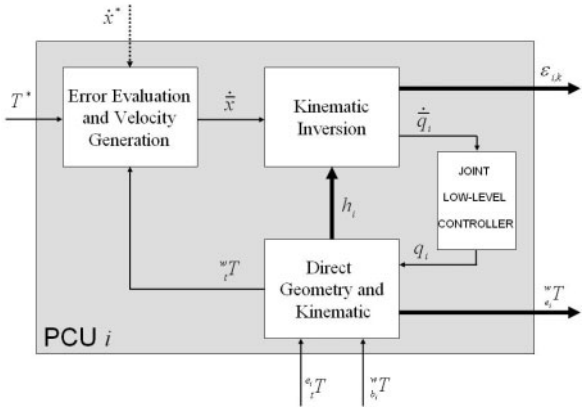


Fig. 9. Single PCU control scheme

an end-effector frame $<e_i>$ and a base frame $<b_i>$ may be defined (with an obvious meaning), whose relative position and attitude are described by the "internal" transformation matrix ${}^{b_i}_{e_i}T(q_i)$, depending on the sole joint variable q_i, as well as on the local geometry of the considered module. In addition, all of the above frames satisfy the following coincidence relationships:

$$\begin{aligned} <b_1> &\equiv <0> \\ <e_i> &\equiv <b_{i+1}> \;; \quad i = 1, 2, \ldots, n-1 \\ <e_n> &\equiv <t> \end{aligned} \tag{38}$$

reflecting the way the modules are assembled together.

The required motion task is defined in terms of making the overall tool frame $<t>$ asymptotically converge towards an assigned goal frame $<g>$. To accomplish such a task, as mentioned in Section 1, every module here is assumed to be equipped with an embedded dedicated local PCU, providing basic communication and joint-level control functionalities.

Moving on from such a premise, assume first that implemented onboard every PCU is the control scheme of Figure 9,

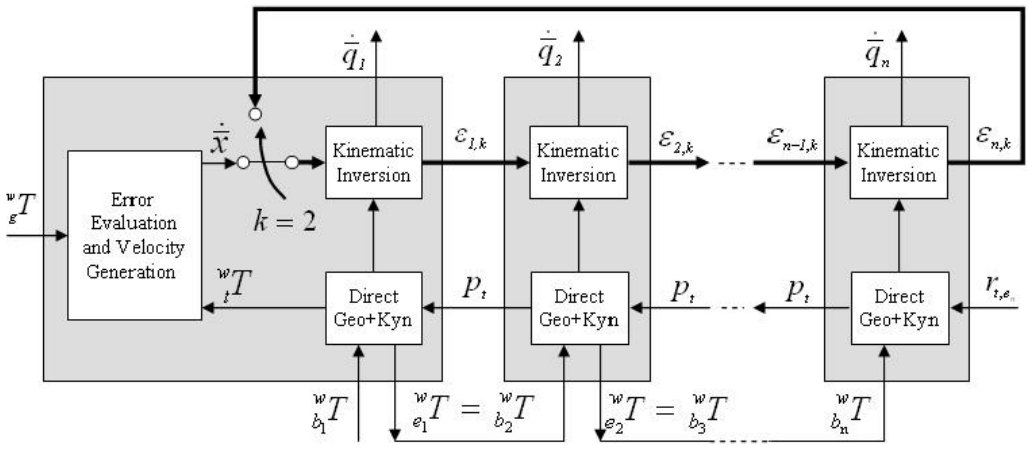


Fig. 10. Computationally distributed control architecture.

which is very similar to the centralized scheme of Figure 2 apart from the following three structural differences.

1. The nature of the *kinematic inversion* functional module, now acting exactly like one of the VPUs of Figure 7.
2. The presence of the additional output signal ${}^{w}_{e_i}T$ from the *direct geometry and kinematic* module (whose role will become evident below) which can be easily computed as follows:
$$ {}^{w}_{e_i}T = {}^{w}_{b_i}T \; {}^{b_i}_{e_i}T(q_i), \tag{39} $$
where ${}^{w}_{b}T$ matrix is provided as an input signal.
3. The evaluation inside the *direct geometry and kinematic* module of the sole vector h_i (in lieu of global J) which can be easily performed, despite the knowledge of the sole joint variable q_i, thanks to inputs ${}^{e_i}_{t}T$ (allowing the computation of the distance between the tool frame and the ith joint axis) and ${}^{w}_{b}T$ (enabling the successive projection onto the world frame).

Before proceeding further it should be noted how, with the adoption of such a control scheme, every PCU is in the condition of controlling the motion of its corresponding robotic module in its operational space, also in cases where it is not mechanically connected to any other robotic units (even if obviously, in such a case, the presence of a single degree of freedom allows only very limited control capabilities). When, instead, a modular manipulator is composed by plugging together n modules, data exchanged through the common communication channel allows the establishment of the announced self-coordinating behavior. The resulting final distributed control architecture is obtained by connecting the n PCUs as shown in Figure 10.

When analyzing Figure 10, first note how the *error evaluation and velocity generation* functional module of just the first PCU is enabled, because the computation of the overall velocity reference $\dot{\bar{x}}$ has to be performed just once. Every other PCU therefore disables such a functional block, maintaining however the possibility of re-enabling it, in case any future self-reconfiguration process lead its corresponding robotic module to the bottom of the chain.

Then note how the lines connecting modules can be subdivided into two clusters: the thin lines represent data exchanged by the set of PCUs for preliminarily transmitting geometric information just reflecting the considered modules arrangement, thus this data is not related to any specific motion task to be accomplished; the bold lines are instead used to perform the iterative kinematic inversion technique described in the previous section.

In more detail, at every sampling interval, first a forward recursion from the first to the last PCU is performed by applying (39). Then the last PCU computes the absolute tool position p_t and broadcasts it to all of the other units, provided that the available communication channel allows it, or backward propagates it along the chain otherwise. In both cases, at the end of this preliminary forward–backward data-exchange phase, every PCU is in the condition of evaluating its Jacobian vector h_i, as well as its corresponding pseudo-inverted vector $h_i^{\#}$. In addition, still in this preliminary phase, the overall task feedback ${}^{w}_{t}T$ is used by the *error evaluation and velocity generation* of the first module for computing, on the basis of reference matrix ${}^{w}_{g}T$, the reference velocity vector $\dot{\bar{x}}$, as described in Section 2. At this point, the kinematic inversion may be accomplished by performing a proper number of high-rate pipelined scattered iterations.

It is also worth noting how the computations performed during the preliminary geometric data-exchange phase are of a very simple nature. Therefore, because the overall resulting processing burden of every PCU module is actually quite light, the most significant limits in the applicability of the self-

coordinating procedure proposed here are just represented by the technological constraints in terms of achievable bandwidth. In this sense, a more precise analysis on the communication requirements can be performed as follows. By denoting the time interval necessary for transferring a six-dimensional vector from one PCU to another by Δ, it is possible to obtain the existing relationship between the total communication burden and the number of performed iterations.

Going into detail, the forward propagation of transformation matrices performed in the first phase of every sampling interval requires an overall duration of $2(n-1)\Delta$, since it provides the transmission of 12-component matrices (indeed just the first three rows of every transformation matrix contain relevant information). The successive backward propagation of the distance vector instead requires at most (in case broadcast communication is not available) $\frac{1}{2}(n-1)\Delta$, because only a three-element vector has to be transmitted along the chain. As concerns the iterative process, the time required for completing a single "high-rate" iteration is $n\Delta$ and, consequently, $pn\Delta$ is the time required for completing p iterations.

Therefore, by combining the expressions of the time necessary for performing all of the different processing phases, an upper bound for the admissible duration of the sub-interval Δ (given the external sampling interval T) is consequently established in terms of the condition

$$\Delta\left[\left(p+\frac{5}{2}\right)n-2\right]\leq T, \tag{40}$$

trivially emphasizing the role played by both the number of degrees of freedom and the number of iterations.

The above condition therefore leads us to conclude that in the cases where, for instance, a manipulator with seven degrees of freedom is considered, having to perform at least $p=10$ iterations within an overall sampling time T of 10 ms, a sub-interval Δ of at most 116 μs is necessary, which certainly represents a very affordable requirement, even if we decide to implement all of the PCUs on low-cost microcontrollers (as shown in Section 6.3).

As a final consideration, note that in cases where a very long kinematic chain is considered, because a large n value would have a negative impact on condition (40), it is always possible to maintain the requirement on Δ above a reasonable threshold, either by reducing the number of iterations executed along the overall chain or, equivalently, by simply employing a reduced number of modules to perform the task, thus letting the iterative procedure only recur along a subset of the available PCUs. The results obtained in both cases are generally comparable, because the achieved motion performances are sensibly determined by the overall number pn of processing phases; thus, an over-employment of a small set of modules is generally equivalent to a reduced employment of a large set of modules.

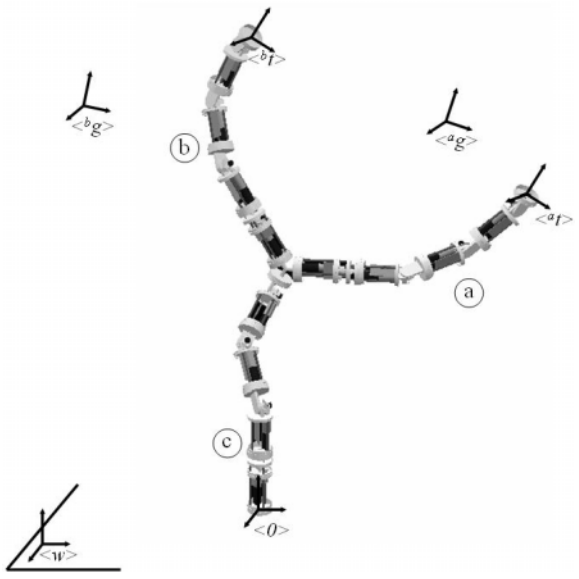


Fig. 11. Tree-structured modular manipulator.

5. Extensions to Tree-structured Kinematic Chains

In this section the extension of the previously devised control architecture and algorithms to the more general case of a tree-structured kinematic chain is provided. For ease of exposition, but without any loss of generality, reference will be made to the structure sketched in Figure 11, obtained by joining three serial chains, named "a", "b" and "c", analogous to the chain seen in the previous section.

As the composing robotic modules (identical to those developed at Genoa Robotic And Automation Laboratory (GRAAL) and described in the next section) are characterized by single-face docking connectors, a special Y-shaped module (hereinafter called a "Y-module") has been inserted at the junction point of the three chains in order to enable a branched connection to be established. While for all of the other modules a single end-effector frame $<e_i>$ may be still defined as done previously, for the two-headed Y-module, two end-effector frames, called $<y_a>$ and $<y_b>$, have to be considered instead.

The presence of a special module, heterogeneous with respect to the others, might seem to introduce significant limitations on the self-reconfigurable potentialities of the overall structure. Nevertheless, it has been necessary here because of the electro-mechanical characteristics of the considered robot. Within a different architecture, if multi-face docking connectors are available, the *joining* role of the Y-module may be indifferently played by any other module, whenever it is contemporaneously docked to three (or possibly more) peers.

As concerns its *functional* role, the Y-module (or any other module occasionally becoming the joining element of a branched chain) has to perform some additional computations (other than those required by the procedure described in previous sections) for dealing with the presence of the branching point. Nevertheless, it will be shown that even such additional processing burden is quite limited. Therefore, its implementation may be easily provided onboard all of the PCUs (because every module might at a certain time "become" a Y-module) without a significant effect on the architectural considerations drawn at the end of previous section.

To better describe the additional functionalities that must be provided by the Y-module, consider again the structure in Figure 11, whose task clearly consists of making tool frames $<^a t>$ and $<^b t>$ (associated with the upper sub-chains) asymptotically converge toward goal frames $<^a g>$ and $<^b g>$, respectively[7].

To accomplish such a task in a distributed fashion, analogously to what happens in the case of serial chains, at every sampling interval a preliminary data-exchange phase has to be performed among all of the PCUs transmit information of a geometric nature. In this case, however, the presence of the Y-module introduces two slight modifications on the procedure seen in previous sections: (i) during the forward phase the Y-module now has to evaluate two "internal" transformation matrices in order to compute matrices ${}^{w}_{y_a}T$ and ${}^{w}_{y_b}T$ which are then sent in parallel to the first PCUs of sub-chains "*a*" and "*b*", respectively; (ii) during the backward phase three parallel data-exchange processes are independently performed by the PCUs of the three sub-chains; more specifically modules of sub-chains "*a*" and "*b*" transmit the position of their corresponding tool frame $<^a t>$ and $<^b t>$, respectively; modules of sub-chain "*c*" instead exchange just the position of the tool frame of their sub-chain, $<^c t>$, which coincides with the base frame of Y-module.

Despite the above differences, at the end of this preliminary phase, every *i*th PCU in this case is also in the condition of computing its corresponding ${}^{(\cdot)}h_i$ vector, which represents the contribution provided by the scalar joint velocity ${}^{(\cdot)}\dot{q}_i$ to the instantaneous velocity of the corresponding tool frame, $<^a t>$, $<^b t>$, $<^c t>$. In addition, still in this phase, one PCU for every upper sub-chain (say, for example, the last one) evaluates the task error (by comparing the feedback matrix ${}^{w}_{(\cdot)_t}T$ with the task matrix ${}^{w}_{(\cdot)_g}T$) and then computes the corresponding desired tool velocity vector ${}^{(\cdot)}\dot{\tilde{x}}$.

As concerns the successive iterative procedure along the overall (now tree-structured) chain, more significant modifications (with respect to what was described in previous sections) have to be introduced and explained.

7. For notational convenience hereinafter all of the terms (scalars, vectors, matrices, frames) referred to a specific sub-chain will be indicated with a left-upper index ${}^{(\cdot)}$ containing the letter of the corresponding sub-chain.

To this aim first observe how, by deciding to initiate the procedure from the last PCU of every upper sub-chain, two independent iterations can be made running in parallel (in the way described in Section 3), proceeding backwards until they reach the Y-module. At this point, by grouping the residual velocity error vectors computed by the first PCU of the upper sub-chains in the more compact form

$$E_{1,1} \doteq \left[{}^a\varepsilon^T_{1,1},\ {}^b\varepsilon^T_{1,1}\right]^T \in \Re^{12}, \tag{41}$$

we can see how the lower supporting sub-chain should be therefore involved with the aim of providing the tool frames $<^a t>$ and $<^b t>$ with a common velocity contribution maximally compensating for the above 12-dimensional error vector.

In order to be in the condition of working in the so-augmented task space, every *i*th PCU of sub-chain "*c*" has to perform a preliminary rigid body transformation on its original Jacobian vector ${}^c h_i$, to obtain the augmented Jacobian vector ${}^c H_i \in \Re^{12}$ describing the contribution provided by its corresponding joint to the motion of tool frames $<^a t>$ and $<^b t>$. To this aim, by defining the matrix

$$S \doteq \left[{}^a S^T,\ {}^b S^T\right]^T \in \Re^{12x6} \tag{42}$$

as the vertical collection of rigid body transformation matrices from $<^c t>$ to $<^a t>$ and from $<^c t>$ to $<^b t>$, respectively, vector ${}^c H_i$ can be easily obtained as follows:

$${}^c H_{\mathrm{i}} = S\,{}^c h_{\mathrm{i}}. \tag{43}$$

To enable the execution of (43), it is necessary that the information on the distance vectors among the three tool frames (required for evaluating matrix S) be known to every PCU of the lower sub-chain. This last fact is not critical, because the Y-module owns this information and can therefore transmit it to all of the underlying PCUs.

By assuming therefore the availability of ${}^c H_i$ vectors, the iterative procedure may continue after the Y-module intervention, by proceeding in its augmented form along all of the PCUs of the lower sub-chain, until it reaches the base of the structure. At this point the residual 12-dimensional error vector outputted by the first PCU of the lower sub-chain is used (after its coherent partitioning) as a new reference input for the upper sub-chains, to initiate a new iteration; and so on for all of the successive iterations.

As can now be easily realized, the above-presented procedure is simply the extension (owing to the presence of two independent six-dimensional tasks) of the same procedure already seen for the linear chain case to the 12-dimensional case; thus, it necessarily exhibits the same convergence properties.

The obvious disadvantage of such a procedure is, however, represented by its increased dimensionality. Although it is irrelevant for the upper lying sub-chains (each one still running a six-dimensional process), it remains a crucial point for the

lower supporting sub-chain, where the 12-dimensional procedure cannot be decomposed into lower-dimensional parallel procedures. As a consequence of the considered procedure necessarily implies a doubling effort, especially in terms of required communication bandwidth. Such a criticality, despite still being considered as acceptable for the case of a 2-end-effector branched chain considered here, it obviously risks becoming unsustainable for increasing numbers of end-effectors.

Thus, in order to eliminate the above drawback (i.e. in order to force the underlying sub-chain process to also be six-dimensional) the alternative iterative procedure described hereafter can consequently be adopted in lieu of the previous procedure. Again start by considering two independent procedures running in parallel along the PCUs of upper sub-chains until they reach the Y-module, leading again to augmented error vector $E_{1,1}$. Then note how *any* non-null six-dimensional $<^c t>$ frame velocity reference vector, ${}^c\dot{\bar{x}}$, if totally accomplished by the lower sub-chain, would induce a 12-component velocity on frames $<^a t>$ and $<^b t>$, which is, in general, different from the required velocity. This is an obvious consequence of the fact that the supporting sub-chain may just provide contributions to the upper tool frames motions which lay on a six-dimensional subspace of $\Re^{12}$ (i.e. that spanned by the columns of the S matrix). Therefore, any component of vector $E_{1,1}$ not belonging to such a subspace is not obtainable via a rigid body motion of the upper sub-chains, and therefore cannot be produced by the sole motion of the lower modules.

In the light of the previous consideration, vector $E_{1,1}$ may always be decomposed as follows:

$$E_{1,1} \doteq S\,{}^c\dot{\bar{x}} + \Phi, \quad \Phi \in \Re^{12}, \tag{44}$$

where ${}^c\dot{\bar{x}}$, as before, is an arbitrary vector, which at this stage can be interpreted as a sort of "guesstimate" of the velocity that the underlying sub-chain modules could possibly provide at their tool frame $<^c t>$, and where Φ (simply obtained via vector difference) represents the "virtual" error at the tool frames $<^a t>$ and $<^b t>$, after the intervention of the lower sub-chain, in cases where it could totally accomplish the required velocity vector ${}^c\dot{\bar{x}}$.

Before commenting on the possible choices of ${}^c\dot{\bar{x}}$, assume for now that a certain choice has been made according to some policy. The resulting ${}^c\dot{\bar{x}}$ is then sent as a reference input to the last PCU of the lower sub-chain, thus making the iterative procedure continue backwards, in its nominal six-dimensional form, also along sub-chain "c" until it reaches the base of the structure. At this point the still residual six-dimensional error vector ${}^c\varepsilon_{1,1}$ is sent back to the Y-module where it is used to evaluate the now "true" resulting residual error at the tool frames $<^a t>$ and $<^b t>$, which is simply obtained as

$$\hat{E}_{1,1} \doteq E_{1,1} - S\left({}^c\dot{\bar{x}} - {}^c\varepsilon_{1,1}\right), \tag{45}$$

where the term in parentheses clearly represents the velocity of $<^c t>$ frame induced by lower module motions.

Note, however, that since vector $\hat{E}_{1,1}$ strictly depends on the true motion capabilities of the lower sub-chain modules, it might happen that its norm becomes greater than the norm of $E_{1,1}$ vector. The reasons that may lead to this situation are not reported here owing to space limitations[8]. For the scope of the present section it is sufficient to say that, when such an event occurs, it means that if lower modules were allowed to use the joint velocity contributions computed during the iteration just concluded, they would provide a worsening contribution to the overall task execution. Thus, in order to avoid such an undesirable occurrence, a suitable scalar-scaling factor β, multiplying all of the joint velocity contributions provided by lower modules, must be introduced. As a consequence, owing to linearity, the corresponding tool frame motion is also scaled by the same factor β and consequently the $\hat{E}_{1,1}$ vector assumes the following modified form, in lieu of (45):

$$\hat{E}_{1,1} \doteq E_{1,1} - \beta\,{}^c\dot{X}, \tag{46}$$

where

$${}^c\dot{X} \doteq S\left({}^c\dot{\bar{x}} - {}^c\varepsilon_{1,1}\right) \tag{47}$$

is the vector of tool frames velocity induced by lower modules (before the intervention of the scaling term β). In order to find out the most suitable β term, the Y-module has just to guarantee the fulfillment of the following condition:

$$\left\|\hat{E}_{1,1}\right\| \doteq \left\|E_{1,1} - \beta^c\dot{X}\right\| \leq \left\|E_{1,1}\right\|, \tag{48}$$

which is clearly optimally solved by the following form (easily computable with the Y-module):

$$\beta = \frac{{}^c\dot{X}^T}{({}^c\dot{X}^T\,{}^c\dot{X})}E_{1,1}. \tag{49}$$

Once the scaling factor β has been evaluated via (49), it is sufficient for the Y-module to send it to all of the lower modules letting them scale their previously evaluated joint velocity contributions accordingly. After the transmission of β, the Y-module has to finally send (upon coherent partition) vector $\hat{E}_{1,1}$, as defined in (46), to the last modules of the two upperlying chains, to start a new iteration. Then the same process can be repeated for all of the successive iterations.

Once the overall procedure has been defined, an open question still needs an answer, concerning the choice of the "guesstimate" ${}^c\dot{\bar{x}}$ vector. To this aim it should be noted how the presence of the scaling factor β allows such a choice to be made within a substantial arbitrariness. Indeed, any eventual worsening effect induced by a "wrong" selection is always compensated for by the corrective intervention of the β term. As a consequence ${}^c\dot{\bar{x}}$ may be kept constant during the entire iterative procedure, or may alternatively be changed at every iteration according to a certain policy. For the time being, an

8. A deeper analysis of the subject can be found in Turetta (2005).

"optimal policy" for choosing the best "guesstimate" corresponding to each iteration has not yet been devised (if it even exists). Thus, the choice is still made by assuming a choice of a "reasonable" nature, such as setting the "guesstimate" at every kth iteration ${}^{c}\dot{\tilde{x}}_k$ as the simple mean vector among upper sub-chains errors:

$$ {}^{c}\dot{\tilde{x}}_k \doteq \frac{1}{2}({}^{a}\varepsilon_{1,k} + {}^{b}\varepsilon_{1,k}) \in \Re^6. \tag{49} $$

Finally, as concerns the convergence properties exhibited by the above-suggested procedure, an analysis similar to that performed for the serial chains can be executed by now shifting attention to the sequence of 12-dimensional error vectors $\{\hat{E}_{1,k}\}$, whose generic term represents the collection of velocity errors of the pair of end-effectors evaluated at the end of the kth iteration.

To this aim, by taking into account expressions (30) and (48), the following recursive form for the $\{\hat{E}_{1,k}\}$ sequence can first be obtained through simple computations:

$$ \begin{cases} \hat{E}_{1,k} = V_k\, A \hat{E}_{1,k-1} \\ \hat{E}_{1,0} = \left[{}^{a}\dot{\tilde{x}}^T\ {}^{b}\dot{\tilde{x}}^T\right]^T \end{cases} \tag{50} $$

where the matrix

$$ A \doteq \begin{bmatrix} {}^{a}A_{a_n} & 0 \\ \hat{E} & {}^{b}A_{b_n} \end{bmatrix} \tag{51} $$

(with sub-matrices ${}^{a}A_{a_n}, {}^{b}A_{b_n} \in \Re^{6x6}$ defined according to (30)) is the constant-during-iterations diagonal block matrix taking into account the role of the upper sub-chains modules, and where V_k matrix is the linear operator satisfying, at every iteration,

$$ \hat{E}_{1k} = V_k E_{1k} \tag{52} $$

and therefore representing a varying-during-iteration matrix (as, at every iteration, it depends on the selected ${}^{c}\dot{\tilde{x}}_k$ vector) expressing the velocity contribution provided by the lower sub-chain modules.

By now considering expression (50), the convergence properties of the proposed procedure for tree-structured chains can be easily obtained by proceeding through the following line of reasoning.

First note how, in the case where (for some reason) all of the lower modules were forced to remain still by a set of null β_k terms, every V_k matrix at every iteration would clearly result to be coincident with the identity matrix, because the contribution to the velocity tasks provided by the motionless lower sub-chain would clearly be null. As a consequence, expression (50) would become absolutely equivalent to a pair of decoupled expressions (30) and all of the considerations about convergence drawn for serial chains would hold exactly here, once separately applied to every one of the upper sub-chains.

In the most interesting case in which the lower sub-chain modules are allowed to move by not-null β_k terms instead, by substituting the expression of β_k (resulting from a generalization of (49) at every iteration) into (48), it is easy to obtain the following relationships:

$$ \begin{aligned} \hat{E}_{11} &= \left[E_{11} - \left(\frac{\dot{X}_k^T E_{11}}{\dot{X}_k^T \dot{X}_k}\right) {}^{c}\dot{X}_k\right] \\ &= \left[I_{12} - \frac{{}^{c}\dot{X}_k \dot{X}_k^T}{\dot{X}_k^T \dot{X}_k}\right] E_{11}, \end{aligned} \tag{53} $$

leading to the following explicit expression for V_k matrix:

$$ V_k = \left[I_{12} - \frac{{}^{c}\dot{X}_k \dot{X}_k^T}{\dot{X}_k^T \dot{X}_k}\right]. \tag{54} $$

By looking at form (54), it appears evident how all of the V_k matrices are actually orthogonal projectors, thus they also perform a norm reduction of input vectors. As a consequence it can be stated that the convergence properties of the upper sub-chains are not affected at all by the motion of the lower sub-chain (or even have a positive effect thanks to the reducing role played by projectors V_k), also contributing (whenever $\beta \neq 0$) to the accomplishment of the velocity tasks.

6. Simulative Results and Experimental Activities

The effectiveness of the proposed distributed control architecture has been validated through several simulative campaigns performed on the kinematic models of different modular chains, of both serial and tree-structured nature. In addition, parallel experimental research activities have been oriented to develop two robotic prototypes; a 3D modular manipulator with five degrees of freedom and a 2D tree-structured chain with nine degrees of freedom. Details on both the simulative results and the experimental activities are provided in this section.

6.1. Simulative Results on Serial Chains

As concerns the tests performed on serial chains, the present work shows the results of three different simulations performed on the model of a 3D modular manipulator with eight degrees of freedom similar to the one depicted in Figure 8.

As a preliminary task (see Extension 1), an end-effector point-to-point movement, from the initial position of [0.4, –0.6, 1.3] m with respect to the base of the robot towards the point [–0.4, –0.8, 0.7] m, while maintaining the initial end-effector orientation, has been executed. Different trials have

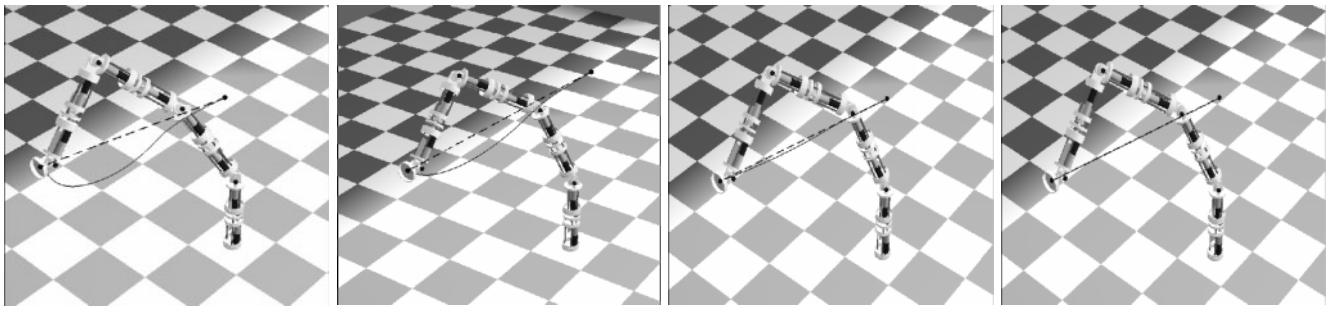


Fig. 12. Trajectory followed.

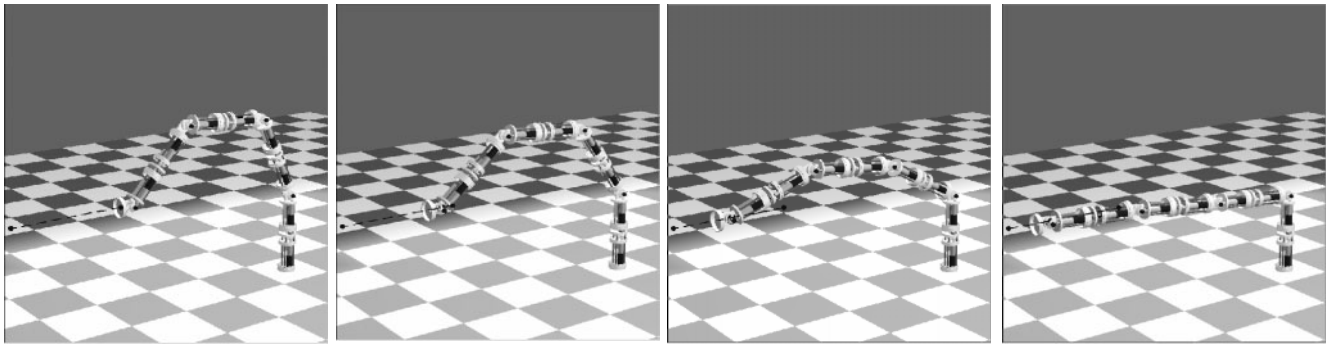


Fig. 13. Motion in the neighborhood of singularities.

been performed, by varying the number of iterations computed within an overall sampling interval T of 5 ms.

In Figure 12 the 3D trajectories actually followed by the end-effector during the different trials are reported (solid lines) and compared with the ideal straight trajectory (dashed). As was obviously expected, although the final convergence to zero of the position error is in any case guaranteed by the presence of the external position control loop, the proposed distributed algorithm for kinematic inversion exhibits improved performance for increasing numbers of iterations.[9]

As a second simulation, the performance of the system in correspondence with singularities is reported here. In this case the manipulator has been asked to move towards a position (at [0.0, –1.6, 0.5] m) located outside its reachable workspace, in such a way that the resulting motion leads to the singular posture corresponding to the arm totally stretched as shown in Figure 13. The test has been executed by performing quite a high number of iterations, 50, within the same sampling interval of 5 ms.

As it can be appreciated from Figure 14, the reported time behavior of joint velocities does not exhibit the presence of any component parts growing up excessively, thus confirming the presence of the self-regularizing behavior described above.

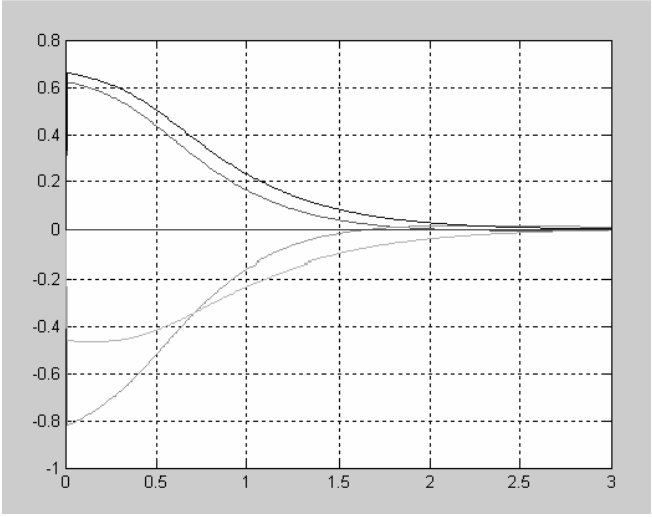
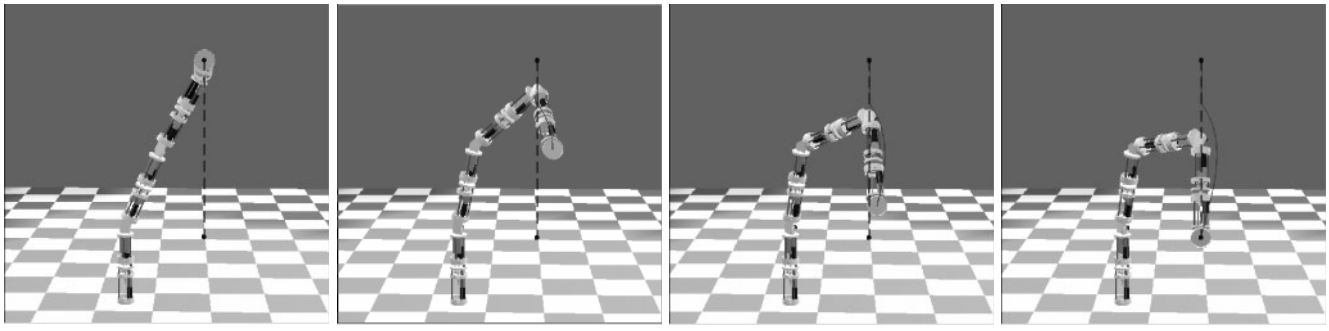


Fig. 14. Motion in the neighborhood of singularities.

The final simulation here reported (see Extension 2) has been included to show the behavior of the proposed control

9. Naturally enough by reducing the gain of the outer loop (thus slowing the required reaching rate), quasi-rectilinear trajectories can be achieved with a reduced number of iterations for the sampling interval.



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

Fig. 15. Motion in the presence of a joint limit: (a) initial position; (b) sixth module at joint limit; (c) sixth module outside the joint limit and (d) final position.

technique under joint limits constraints. Actually, in such an occurrence (i.e. when a joint occasionally reaches its mechanical limit), the distributed kinematic inversion technique allows the implementation of a very simple and effective strategy for managing such a case. At every iteration, first the velocity term $\dot{\bar{q}}_{i,k}$ is always computed as seen before. Then, if it is "acceptable" (i.e. if it induced a motion in the opposite direction of the reached limit) it is summed to the previously computed terms and the process continues as described in previous sections. In the opposite case, it is disregarded and no error velocity reduction is performed by that PCU at that iteration (i.e. the output error is coincident with the reference input). To demonstrate the effectiveness of this strategy, a Cartesian motion unavoidably causing a joint (the sixth joint) to reach its physical limit has been commanded to the manipulator.

In Figure 15 different phases of the movement are shown, while the time behavior of joint 6 is reported in Figure 16. The task is again a point-to-point motion from position [0.4, –0.6, 1.3] m reported in Figure 15(a) to position [0.4, –0.6, 0.5] m (Figure 15(d)). To emphasize the presence of the joint limit, the test has been executed by again performing 50 iterations per sampling interval, thus guaranteeing an almost linear trajectory when the manipulator is far from the joint limits; moreover, the task has been selected in such a way that the attainment of the joint limit temporarily prevents the exact achievement of the commanded velocity, i.e. the other modules, despite the structural redundancy of the manipulator, are not able to compensate for the absence of joint 6 in this case. In this way the trajectory followed by the end-effector becomes non-linear. Otherwise, if the posture allowed the other modules to produce a contribution compensating for the temporary inoperativeness of joint 6, the resulting trajectory would be sufficiently linear (as compatible with 50 iterations).

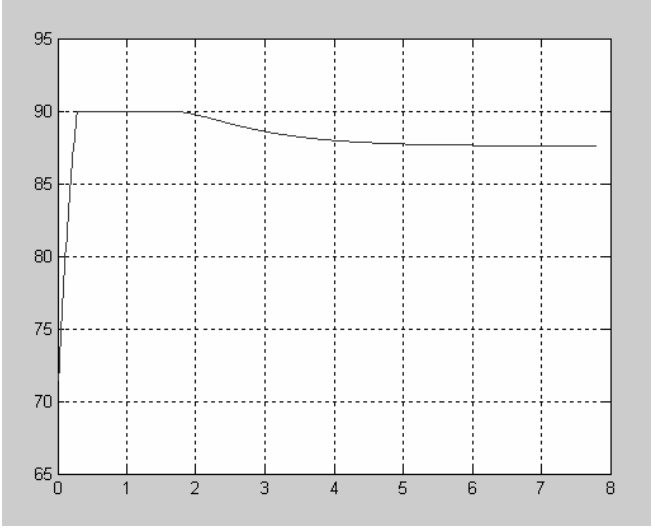


Fig. 16. Time behavior (degrees) of joint 6 (physical limit 90°).

6.2. Simulative Results on Tree-structured Chains

As concerns tree-structured chains, many simulations have been executed by considering different models of modular manipulators similar to that depicted in Figure 11. In particular, hereafter the results obtained on a model of a structure with 20 degrees of freedom are reported. More specifically, the considered robot is characterized by two upper sub-chains with eight degrees of freedom mounted on a common lower sub-chain with four degrees of freedom. In order to underline the behavior of the proposed technique once applied to tree-structured chains, a sufficiently high number of iterations, namely 100, have been used.

In the first simulation (see Extension 3) the pair of end-effectors is asked to move from points [0.0, –2.0, 1.2] m and [0.0, 0.5, 2.0] m toward points [0.6, 1.0, 2.0] m and [–0.5, –0.8,

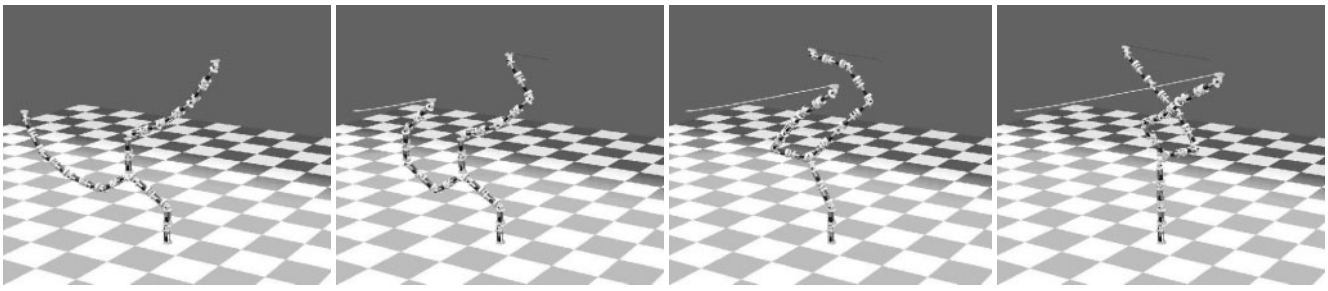
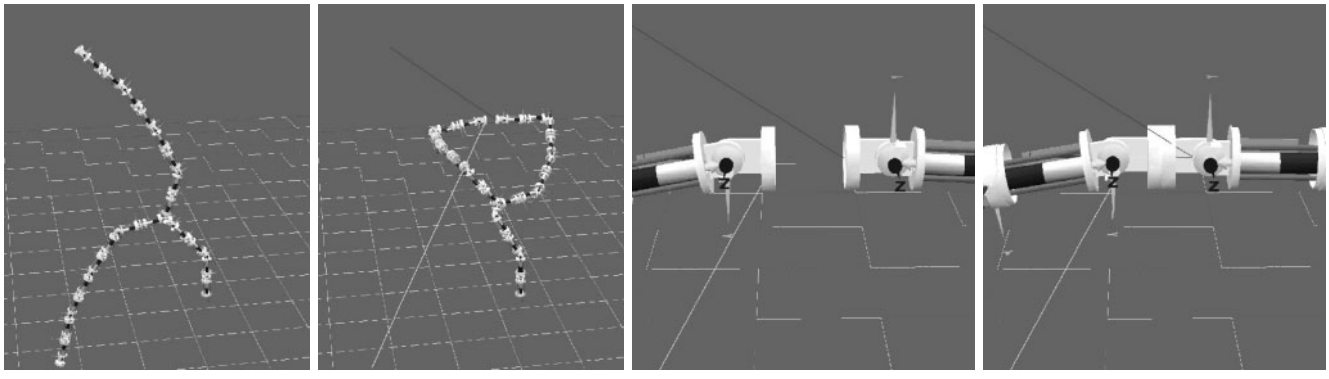


Fig. 17. Different motion phases.



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

Fig. 18. Approaching and docking maneuver: (a), (b) initial and final positions of the approaching phase; (c), (d) initial and final positions of the docking phase.

2.0] m, respectively. In addition, both of the arms are required to maintain the initial end-effector orientations.

In Figure 17 some snapshots from the simulation are reported in order to show how the proposed control and coordination strategy is actually very effective when applied to tree-structured chains, allowing both of the end-effectors to reach their respective goal frames with null final position errors.

In the second simulation (see Extension 4) the end-effectors start from points [0.0, –1.4, –0.6] m and [0.0, –1.1, 2.3] m, respectively (see Figure 18(a)) and are first commanded to approach and then to touch each other at point [–0.95, –0.1, 1.4] m. Although the modules considered cannot actually dock upon contact, a docking phase is "mimicked" in the simulation to show how the proposed control technique actually enables self-reconfiguration. Indeed, any modular robot realized for supporting self-reconfiguration (from the electromechanical point of view), by adopting the proposed self-organizing technique, is in a condition to easily operate any module re-arrangement for autonomously varying its shape.

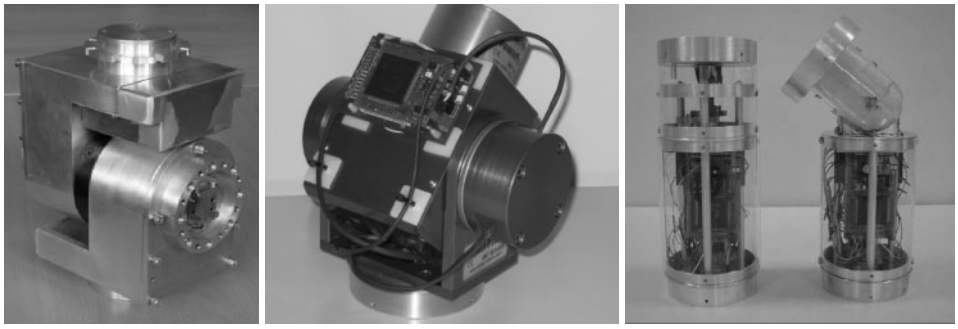
6.3. Experimental Activities

A recently concluded research project led to the development of the second generation of P^{2}MR (Plug & Play Modular Robot)[10], a prototype of a modular robot exhibiting the previously described self-coordinating behavior.

In its current version it is made up of four heterogeneous modules (see Figure 19), which can be easily assembled together in several alternative ways (see Extension 6) to obtain a class of manipulators with up to five degrees of freedom characterized by different workspaces (see Figure 20).

Although the composing modules present highly diversified mechanical characteristics and mount different servos, they share a common element: all of them are equipped with the same PCU, based on a 24 MHz 16-bit microcontroller of the C167 family from Infineon (see Figure 21). Every PCU is interfaced with the corresponding low-level joint controller via an RS-232 serial data link and can communicate with the

10. For the first-generation modules see Extension 5.



(a) (b) (c)

Fig. 19. Atomic modules of P^{2}MR: (a) a module with a single degree of freedom developed during the project; (b) a commercial wrist with two degrees of freedom acquired from Shunk, DE; (c) a pair of modules with a single degree of freedom from the first generation of P^{2}MR.

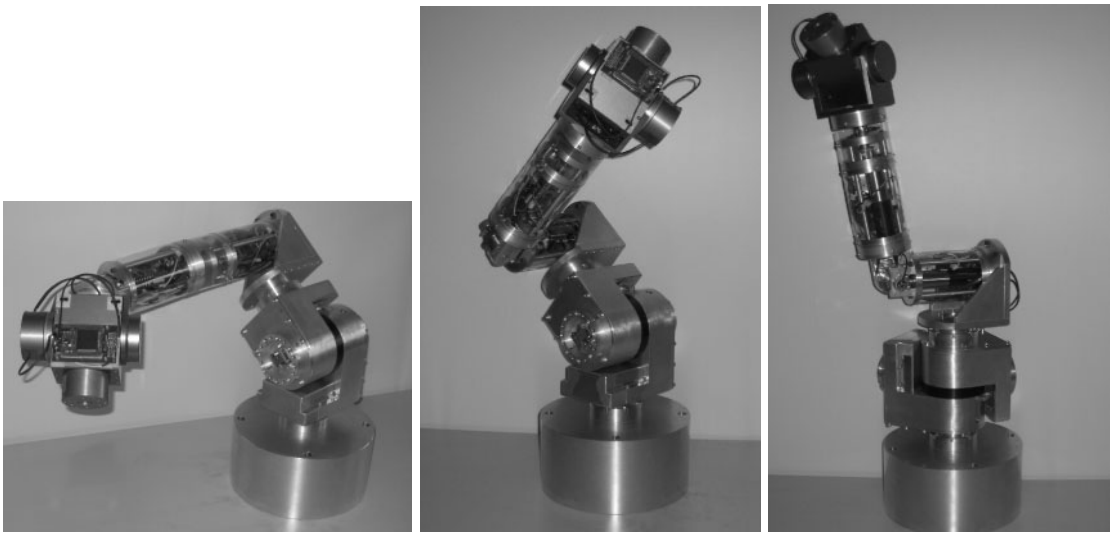


Fig. 20. Some of the different module configurations of P^{2}MR.

other peers by means of a common synchronous serial channel (SSC) allowing high-speed transmission at up to 4 MBaud s^{-1}.

With the available bandwidth the time interval Δ (defined in Section 4) necessary for transferring a 16-bit six-dimensional vector from one PCU to another becomes equal to 24 ms. As a consequence, by applying relationship (40) to the case at hand, it turns out that within a sampling interval T of 10 ms (which is reasonable for guaranteeing the stability of the system), up to 55 iterations may be performed along the chain. However, note that this value has to be considered as an upper limit. Indeed, as mentioned before, it does not take into account the time required for all of the processing phases, which is negligible with respect to the communication but certainly not null. In addition, the implemented communication protocol may introduce some overhead, by requiring that addi-

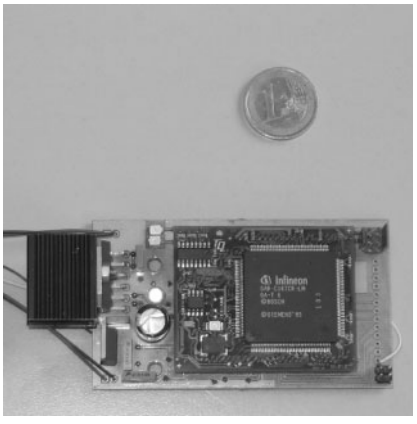


Fig. 21. The PCU in P^{2}MR.

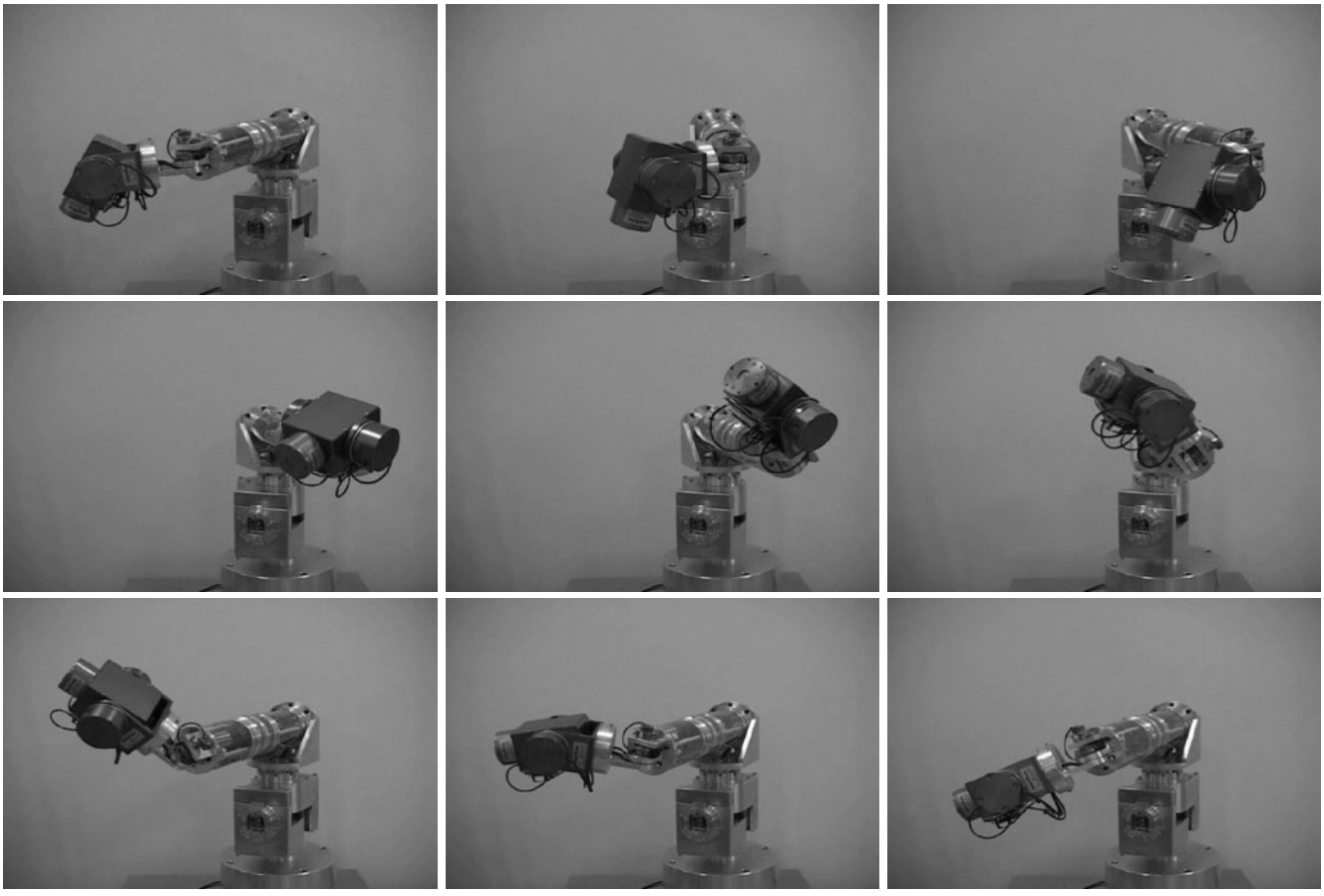


Fig. 22. Sequence of movements of the P^2MR experimental setup.

tional information be transmitted together with relevant data, such as, for example, the destination of the message and/or the sender.

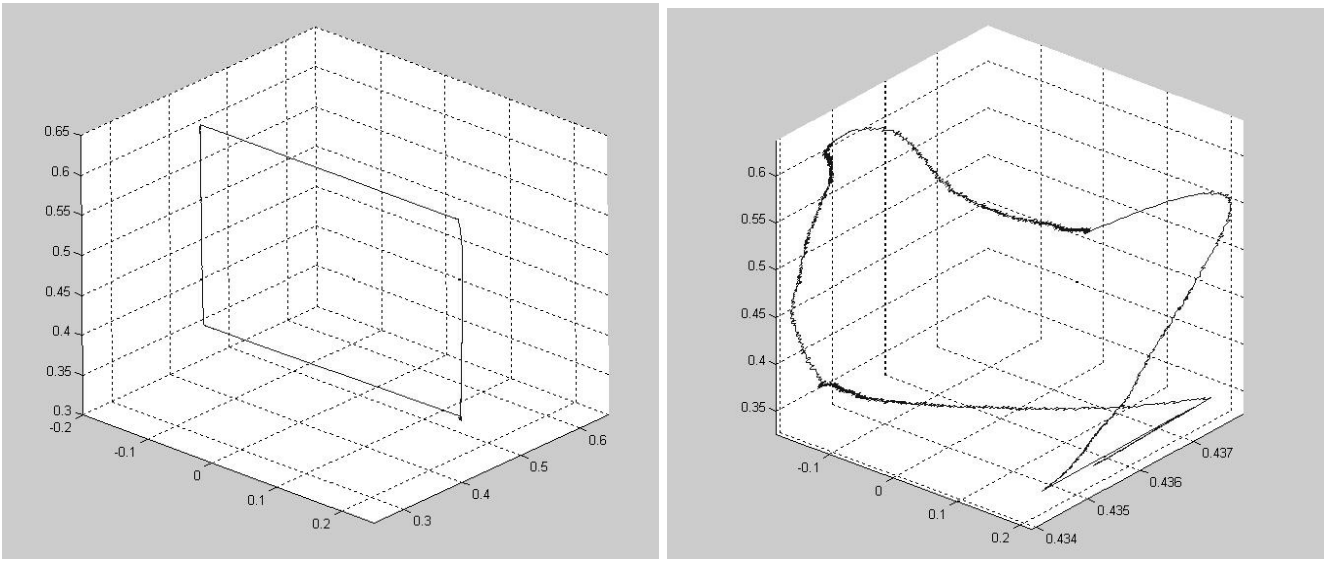
Indeed, data collected during the experimental phases allows to state that at most 29 iterations may actually be performed with the considered architecture within a sampling interval T of 10 ms.

Nevertheless, as foreseen by simulations, experimental motion tests on the physical robot confirmed that even with a lower number of iterations, say 10, good motion performances are achievable. In the preliminary test, the end-effector of the robot was asked to move along the four edges of a rectangle. Figure 22 shows some snapshots of the actual movements of the robot.

At the present stage of the system development, it is not yet possible to exhibit a nice graph on the actual robot movements based on sensors measurements collected from the field. Indeed, sensor data cannot be registered during the experiment (owing to memory limitations) no transmitted online to the man–machine interface at a sufficiently high rate to allow a significant plot (owing to bandwidth limitations on the implemented serial link).

Although the whole experiment is documented in Extension 7, a realistic idea of the trajectory followed by the end-effector of the robot can be obtained from the Figure 23, representing the results of a simulation performed by using the same code running on the real PCUs (thus employing just fixed-point 16-bit numbers) and by explicitly taking into account all of the characteristics of the real robot relevant to PCUs (such as sensors resolution, quantization errors and so on). As can be appreciated in particular from Figure 23(b) (which differs from Fig. 23(a) only in the scale of the y-axis) the position errors during the motion (despite the limited number of iterations performed per sampling interval, 10) always remain lower than 4 mm, which certainly represents a very nice result.

Before concluding this section we feel it is worth describing the second prototype currently under construction at GRAAL. It is a robotic system that has been conceived with the intent of testing the behavior of the developed control framework within more complex structures. As a consequence, in order

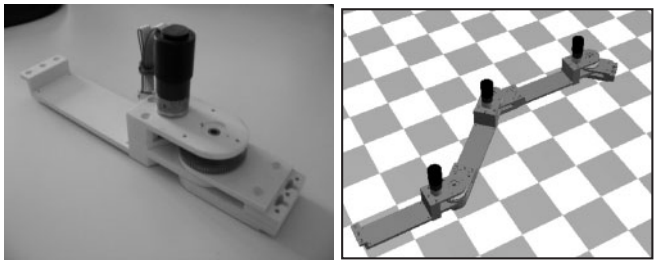


(a) (b)

Fig. 23. 3D end-effector trajectory during the simulated experiment plotted at different scales.

to avoid any problems induced by payload/weight ratio constraints (while maintaining the development costs), this second prototype will be consequently realized as a very simple planar structure. It will be made of TUFSET, a plastic material with mechanical characteristics similar to aluminum but with reduced cost and weight. The control board will be realized on an FPGA-based faster processing architecture, which is also at an advanced stage of development.

At this time, a preliminary unit has been finished (see Figure 24). Another eight are already planned and will soon be constructed, in order to reach an adequate number of modules to allow the formation of tree-structured and even closed kinematic chains.



(a) (b)

Fig. 24. Planar modular robot under development: (a) first prototype; (b) simulative environment.

7. Conclusions

An effective, computationally distributed, algorithm for controlling self-reconfigurable modular tree-structured chains has been presented. Such a technique allows the motion of the end-effectors to be controlled in the operational space by solely exploiting the control capabilities of the processing units embedded inside the modules. No external centralized control hardware is required. A global *self-organizing* behavior among the modules is indeed established automatically and propagated along the chain. Coordination emerges as a result of the application of an iterative procedure based on repeated data exchanges among the processing units. Although the required communication bandwidth introduces technological constraints in terms of achievable data-transfer rates, the amount of local computation performed by a single unit at every iteration is actually very limited, as is the quantity of data to be exchanged by two adjacent units. As a consequence even current low-cost technology can suitably support the proposed self-organizing technique, which consequently appears as a promising approach to the problem of motion control, possibly becoming a future less-expensive solution than traditional centralized control hardware.

Further work will be primarily oriented to extend the developed theory to closed kinematic chains, because the possibility of managing structures with loops represents a fundamental capability for self-reconfigurable robots. Moreover, the case of simultaneous execution of multiple tasks with priorities along the same chain will also be investigated, with the obvious intent of identifying suitable strategies for exploiting redundancy.

Acknowledgment

The present work has been supported by PSTL: "Parco Scientifico e Tecnologico della Liguria" (Science and Technology Park of Liguria) within a special project on modular robotics.

Appendix: Index to Multimedia Extensions

The multimedia extension page is found at http://www.ijrr.or

Table of Multimedia Extensions

Extension	Type	Description
1	Video	Simulation of a modular manipulator with eight degrees of freedom executing a point-to-point motion task while performing a varying number of iterations
2	Video	Simulation of a modular manipulator with eight degrees of freedom moving in the presence of a joint limit
3	Video	Simulation of a tree-structured modular manipulator with 20 degrees of freedom executing a point-to-point movement
4	Video	Simulation of a tree-structured modular manipulator with 20 degrees of freedom performing a docking maneuver
5	Video	P^2MR: first-generation modules
6	Video	P^2MR: different configurations obtainable by second-generation modules
7	Video	P^2MR: movement along the edges of a rectangle

References

Aicardi, M., Cannata, G., and Casalino, G. (1995). Stability and robustness analysis of a two layered hierarchical architecture for the control of robots in the operational space. *Proceedings of the International Conference on Robotics and Automation*, pp. 2771–2778.

Antonelli, G., Chiaverini, S. and Fusco, G. (2003). A new online algorithm for inverse kinematics of robot manipulators ensuring path tracking capability under joint limits. *IEEE Transactions on Robotics and Automation*, **19**(1): 162–167.

Casalino, G. and Turetta, A. (2004). Coordination and control of multiarm nonholonomic mobile manipulators. *MISTRAL: Methodologies and Integration of Subsystems and Technologies for Robotic Architectures and Locomotion*, Siciliano, B., Casalino, G., De Luca, A. and Melchiorri, C. (eds), *Springer Tracts in Advanced Robotics*. Berlin, Springer.

Casalino, G. and Turetta, A. (2005). A computationally distributed self-organizing algorithm for the control of manipulators in the operational space. *Proceedings of the IEEE International Conference on Robotics and Automation*, Barcelona, Spain, April.

Casalino, G., Turetta, A. and Sorbara, A. (2006). Dynamic programming based computationally distributed kinematic inversion technique. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, October.

Chiacchio, P., Chiaverini, S., Sciavicco, L. and Siciliano, B. (1991). Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy. *International Journal of Robotics Research*, **10**(4): 410–425.

Chiaverini, S. (1997). Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Transactions on Robotics and Automation*, **13**(3): 398–410.

Chirikjian, G. S. and Burdick, J. W. (1994). A modal approach to hyper-redundant manipulator kinematics. *IEEE Transactions on Robotics and Automation*, **10**(3): 343–354.

Duff, D. G., Yim, M. and Roufas, K. (2001). Evolution of PolyBot: a modular reconfigurable robot. *Proceedings of the Harmonic Drive International Symposium*, Nagano, Japan, November.

Hou F. and Shen W. M. (2006). Mathematical foundation for hormone-inspired control for self-reconfigurable robotic systems. *Proceedings of the IEEE International Conference on Robotics and Automation*, Orlando, FL, May, pp. 1477–1482.

Jørgensen, M. W., Østergaard, E. H. and Lund, H. H. (2004). Modular ATRON: modules for a self-reconfigurable robot. *Proceedings of the* IEEE/RSJ *International Conference* on Intelligent Robots and Systems, Sendai, Japan, October.

Kamimura, A., Kurokawa, H., Yoshida, E., Tomita, K., Kokaji, S. and Murata, S. (2004). Distributed adaptive locomotion by a modular robotic system, M-TRAN II. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, October.

Kamimura, A., Kurokawa, H., Yoshida, E., Murata, S., Tomita, K. and Kokaji, S. (2005). Automatic locomotion design and experiments for a modular robotic system. *IEEE/ASME Transactions on Mechatronics*, **10**(3).

Kurokawa, H., Kamimura, A., Yoshida, E., Tomita, K., Kokaji, S. and Murata, S. (2003). M-TRAN II: metamorphosis from a four-legged walker to a caterpillar. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, NV, October.

Moll, M., Will, P., Krivokon, M. and Shen W. M. (2006). distributed control of the center of mass of a modular robot. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, October.

Murata, S., Yoshida, E., Kamimura, A., Kurokawa, H., Tomita, K. and Kokaji, S. (2002). M-TRAN: self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, **7**(4).

Murata, S., Kakomura, K. and Kurokawa, H. (2006). Docking experiments of a modular robot by visual feedback. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, October.

Nakamura, Y. (1991). *Advanced Robotics: Redundancy and Optimization*. Reading, MA, Addison-Wesley.

Salemi, B., Moll, M. and Shen, W. M. (2006). SUPERBOT: a deployable, multi-functional, and modular self-reconfigurable robotic system. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robot and Systems*, Beijing, China, October.

Shen, W. M., Krivokon, M., Chiu, H., Everist, J., Rubenstein, M. and Venkatesh, J. (2006). Multimode locomotion for reconfigurable robots. *Autonomous Robots*, **20**(2): 165–177.

Shen, W. M., Salemi, B. and Will, P. (2000). Hormones for self-reconfigurable robots. *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS-6)*, Venice, Italy, pp. 918–925.

Shen, W. M., Salemi, B. and Will, P. (2002). Hormone-inspired adaptive communication and distributed control for CONRO self-reconfigurable robots. *IEEE Transactions on Robotics and Automation*, **18**(5): 700–712.

Shen, W. M. and Will, P. (2001). Docking in self-reconfigurable robots. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1049–1054.

Støy, K., Shen, W.-M. and Will, P. (2002). Using role-based control to produce locomotion in chain-type self-reconfigurable robots. *IEEE/ASME Transactions on Mechatronics*, **7**(4): 410–417.

Turetta, A. (2005). Self-coordinating distributed control algorithms for multi-robot systems. *Ph.D. Thesis*, DIST, University of Genova.

Yim, M., Zhang, Y., Roufas, K., Duff, D. and Eldershaw, C. (2002a). Connecting and disconnecting for chain self-reconfiguration with PolyBot. *IEEE/ASME Transactions on Mechatronics*, **7**(4).

Yim, M. H., Zhang, Y. and Duff, D. G. (2002b). Modular robots. *IEEE Spectrum*, **39**(2): 30–34.

Yim, M. H., Roufas, K., Duff, D., Zhang, Y, Eldershaw, C. and Homans, S. (2001). Modular reconfigurable robots in space applications. *Proceedings of the International Conference on Advanced Robotics*, Budapest, Hungary, August.