Post-print version (generated on 17.12.2021)
This and other publications are available at:
http://www.acin.tuwien.ac.at/publikationen/ams/

Proc. of the IEEE Int. Conf. on Industrial Technology, pages 848–853, 2016

A Framework for Modular and Distributable Control of Reconfigurable Robotic Systems

Michael Steinegger*, Nikolaus Plaschka*, Martin Melik-Merkumians, and Georg Schitter

Automation and Control Institute, Vienna University of Technology
Gußhausstraße 27-29, AT-1040 Vienna, Austria
Email: {steinegger,plaschka,melik-merkumians,schitter}@acin.tuwien.ac.at

***Abstract*—In this paper, a distributable control approach for reconfigurable robotic systems is presented. The approach is based on the dynamic programming technique to solve the inverse kinematics problem in a distributed way. This algorithm is extended by an additional control input to reduce the orthogonal error to the desired Cartesian trajectory and to ensure smooth joint angles and velocities. The required torques to track the desired Cartesian trajectory are then computed by application of the well-known recursive Newton-Euler algorithm. Furthermore, the entire algorithm is implemented according to the international standard IEC 61499 and encapsulated into several reusable function blocks for controlling a single joint. This reduces the effort for implementing the control algorithm as well as the computational time, since they can be transferred to the corresponding joint control device. The evaluation of the algorithm is carried out on a simulation of a manipulator with four degrees of freedom.**

I. Introduction

The increasing demand for individual products leads to smaller lot sizes and away from conventional mass production [1]. This paradigm shift requires fast *reconfigurable manufacturing systems* (RMS) in order to increase the flexibility of the manufacturing system. Fast and easy reconfiguration of the underlying subsystems of a manufacturing system would therefore enable the ability to efficiently produce customized products while limiting the costs for the manufacturers. Here, modularization is considered as the basis to enable highly flexible RMS [2]. The aim of modularization is to divide the rigid structure of current manufacturing systems into smaller independent subsystems which can then be combined in order to assemble the manufacturing system for producing the desired product.

Robotic cells are usually considered as one monolithic subsystem for flexible manufacturing systems. However, such cells provide only a limited amount of manufacturing capabilities, since they are mostly designed and also programmed for specific tasks. One possibility to decrease the reconfiguration time of present robotic cells is to modularize the trajectory planning process. Thus, joint-specific control policies are defined, also denoted as dynamic movement primitives [3], which enable a very fast online (re-)planning of the manipulator trajectory while still supporting complex and also optimal end-effector trajectories [4]. However, the fixed structure of the manipulator itself still limits the manufacturing capabilities of the robotic cell. Therefore, the modularization of the entire robotic system for industrial manufacturing became a major research topic [5].

The goal of modular design in robotics is to break the barrier of fixed kinematics by assembling desired handling systems out of mass-produced modules as well as replacing proprietary control hardware and vendor-specific programming languages. This can enable the faster (re-)configuration on changing task requirements and could lower the costs for the robotic system [6]. The mechatronic design of such modules should therefore consider properties like mechanical and computational compatibility of the individual modules to guarantee interchangeability and a fast assembly and reconfiguration of the system [7].

In order to further increase the flexibility of modular handling systems, several approaches have been developed to additionally modularize the control algorithms. Therefore, classical software engineering paradigms like object-orientation or component-based engineering [8] are applied to control software design for robotic systems. For instance, Stewart and Khosla [9] describe the development of different software components in order to compute the forward and inverse kinematics. However, the underlying robotic system is still considered as a fixed kinematic chain executing several predefined tasks.

A totally distributed approach, where the robotic system is composed out of several modules that are equipped with their own controller and local control program, additionally requires the distributed computation of the kinematics. Here, the *cyclic coordinate descent* (CCD) method [10] represents one possible solution. The main idea is to vary only one single joint variable per optimization cycle to minimize the corresponding objective function, which is in this case the error between the actual and desired pose. Due to the heuristic nature of the CCD algorithm, however, a smooth movement of the robot is not guaranteed.

This paper presents a modular approach for computing the inverse kinematics and the required joint torques in a distributed way. The proposed approach extends the algorithm based on *Dynamic Programming* (DP) to solve the inverse kinematic on different control devices as described in [11]. Smooth joint angle trajectories are ensured by additionally considering the orthogonal error vector from the actual manipulator pose to the desired Cartesian trajectory in the reference input signal, which enables the exact tracking even of linear Cartesian trajectories. The solution of the inverse kinematic problem is then fed to an implementation of the Newton-Euler algorithm to compute the desired joint torques. The combination of both algorithms enables the possibility to distribute the entire control software for a robotic system over different devices and requires only the trajectory planning to be executed in a centralized way.

The remainder of the paper is organized as follows. In Sec. II, the modular kinematic control concept proposed by Casalino and Turetta [11] and its extension are described. The computation of the desired torques based on the extended kinematic control is described in Sec. III. Section IV presents the implementation of the entire modular control approach according to the international standard IEC 61499 [12]. Simulation results of the proposed method applied to a SCARA (*Selective Compliance Assembly Robot Arm*) manipulator are shown and discussed in Sec. V, and Sec. VI concludes the paper.

*These authors contributed equally to this work

Post-print version of the article: Michael Steinegger, Nikolaus Plaschka, Martin Melik-Merkumians, and Georg Schitter, "Vision-based probabilistic absolute position sensor," *2016 IEEE International Conference on Industrial Technology (ICIT)*, 2016. DOI: 10.1109/ICIT.2016.7474862
© 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Post-print version (generated on 17.12.2021)
This and other publications are available at:
http://www.acin.tuwien.ac.at/publikationen/ams/

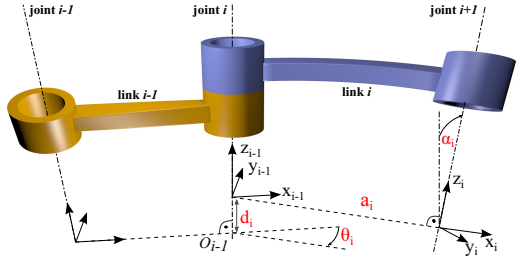


Fig. 1. Description of a kinematic chain based on the Denavit-Hartenberg convention with translational (d_i, a_i) and rotational parameters (θ_i, α_i).

II. Modular Kinematic Control

In order to increase the flexibility of present handling systems in industrial manufacturing plants, the modularization into small mechatronic units that are equipped with their own control software has attracted a lot of attention in research and engineering. Splitting the kinematic chain, as depicted in Fig. 1, into several joint / link pairs (e.g, i^{th} joint and i^{th} link) enables the description of the transformation ${}^0\boldsymbol{T}_n$ between the frame $\mathcal{F}_0$ attached to the manipulator base and the last frame $\mathcal{F}_n$ of the kinematic chain as a series of matrix multiplications

$$ {}^0\boldsymbol{T}_n = {}^0\boldsymbol{T}_1 {}^1\boldsymbol{T}_2 \dots {}^{n-1}\boldsymbol{T}_n \tag{1} $$

with individual homogeneous transformation matrices ${}^{i-1}\boldsymbol{T}_i$ and n representing the number of manipulator joints [13]. A single transformation matrix can be parametrized with the *Denavit-Hartenberg* (DH) parameters [14] which only requires four parameters to describe a transformation in Cartesian space. Applying the short notation $\sin(\cdot) = \mathrm{s}_{(\cdot)}$ and $\cos(\cdot) = \mathrm{c}_{(\cdot)}$, one transformation from frame $\mathcal{F}_{i-1}$ to $\mathcal{F}_i$ can be written as

$$ {}^{i-1}\boldsymbol{T}_i = \begin{bmatrix} {}^{i-1}\boldsymbol{R}_i & \boldsymbol{\gamma}_i \\ \boldsymbol{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathrm{c}_{\theta_i} & -\mathrm{s}_{\theta_i}\mathrm{c}_{\alpha_i} & \mathrm{s}_{\theta_i}\mathrm{s}_{\alpha_i} & a\mathrm{c}_{\theta_i} \\ \mathrm{s}_{\theta_i} & \mathrm{c}_{\theta_i}\mathrm{c}_{\alpha_i} & -\mathrm{c}_{\theta_i}\mathrm{s}_{\alpha_i} & a\mathrm{s}_{\theta_i} \\ 0 & \mathrm{s}_{\alpha_i} & \mathrm{c}_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2} $$

where $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix, $\boldsymbol{\gamma} \in \mathbb{R}^{3\times1}$ is the translation vector, $\boldsymbol{0} \in \mathbb{R}^{1\times3}$ is the null vector, and θ, d, a, α are the DH parameters. Depending on the joint type, either the angular parameter θ (for revolute joints) or the displacement d (for prismatic joints) is variable. These variables are combined in the generalized joint variable $q_i \in (\theta_i, d_i)$ for the i^{th} joint.

Since trajectories for robots and handling systems in general are often planned in Cartesian space, the corresponding joint variable series has to be computed for each joint in order to follow the planned trajectory precisely. This problem is known as inverse kinematics and is highly non-linear, which can be directly seen by considering the forward kinematics based on homogeneous transformations as in Eq. (1). Therefore, the inverse kinematics problem is often solved differentially in terms of Jacobian methods, linking the joint velocities to the Cartesian velocities by a locally linear equation [13]. The computation of the desired joint velocities can be written as

$$ \dot{\boldsymbol{q}}^{\text{des}} = \boldsymbol{J}^{+}(\boldsymbol{q})\, \dot{\boldsymbol{x}}^{\text{des}} \ , \tag{3} $$

where $\boldsymbol{J}^{+}(\cdot)$ is the pseudo-inverse of the Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{6\times n}$ and $\dot{\boldsymbol{q}}^{\text{des}} \in \mathbb{R}^{n\times1}$ the computed joint velocity vector to reach the desired pose $\boldsymbol{x}^{\text{des}} \in \mathbb{R}^{6\times1}$ of the end-effector with a given velocity $\dot{\boldsymbol{x}}^{\text{des}} \in \mathbb{R}^{6\times1}$ in Cartesian space.

A modular and distributed approach to compute the inverse kinematics based on Jacobian methods is described by Casalino and Turetta [11]. The idea is to split the computation of the inverse kinematics into n sub-problems. Therefore, each pair (joint / link) is equipped with its own controller that computes the possible participation of the corresponding joint to the desired entire manipulator movement. The algorithm and the proposed extensions are described in the following subsection, since it enables the modularization of the inverse kinematics into software components and represents the basis for the entire distribution approach described in this paper.

A. Modular Inverse Kinematics Approach

The algorithm presented in [11] applies a dual-step DP approach to solve the inverse kinematics problem iteratively and in a distributed way. First, the Jacobian matrix $\boldsymbol{J}$ is split into its column vectors $\boldsymbol{h}_i \in \mathbb{R}^{6\times1}$, $i \in \{1, \dots, n\}$, which describe the contribution of the i^{th} joint to the movement of the end-effector [11]. The forward kinematic relation for the Cartesian velocities given by $\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}}$ can then be written as

$$ \dot{\boldsymbol{x}}_i = \dot{\boldsymbol{x}}_{i-1} + \boldsymbol{h}_i \dot{q}_i \ , \tag{4} $$

with $\dot{\boldsymbol{x}}_0 = \boldsymbol{0}^{\mathrm{T}}$. By introducing the Cartesian velocity error vector $\dot{\boldsymbol{\varepsilon}}_i = \dot{\boldsymbol{x}}^{\text{ref}} - \dot{\boldsymbol{x}}_i$, the recursive form (4) can be written as

$$ \dot{\boldsymbol{\varepsilon}}_i = \dot{\boldsymbol{\varepsilon}}_{i-1} - \boldsymbol{h}_i \dot{q}_i \ , \tag{5} $$

with $\dot{\boldsymbol{\varepsilon}}_0 = \dot{\boldsymbol{x}}^{\text{ref}}$ initially. It can be shown, that the optimization problem assigned to (5) can be solved with a dual-step DP approach (cf. [11] for details). Here, the goal of the optimization is to minimize the error $\dot{\boldsymbol{\varepsilon}} = \dot{\boldsymbol{x}}^{\text{des}} - \dot{\boldsymbol{x}}^{\text{act}}$ between the actual Cartesian velocity $\dot{\boldsymbol{x}}^{\text{act}}$ and the desired velocity $\dot{\boldsymbol{x}}^{\text{des}}$ by computing the contribution of each joint / link pair locally and then propagating the remaining error to the next joint controller. Thus, an optimal solution is guaranteed as long as the desired position is reachable. The applied DP algorithm consists of a backward and a forward phase as depicted in Fig. 2.

1) Backward Phase: In the backward phase, the i^{th} joint controller computes the actual transformation matrix ${}^{i-1}\boldsymbol{T}_i$ according to (2) with the locally stored DH parameters and multiplies it with ${}^{i}\boldsymbol{T}_n$. The resulting homogeneous transformation matrix is then propagated to the previous link controller. Thus, the controller of the first joint can compute the actual Cartesian position $\boldsymbol{x}^{\text{act}}$ of the n^{th} link (or the end-effector).

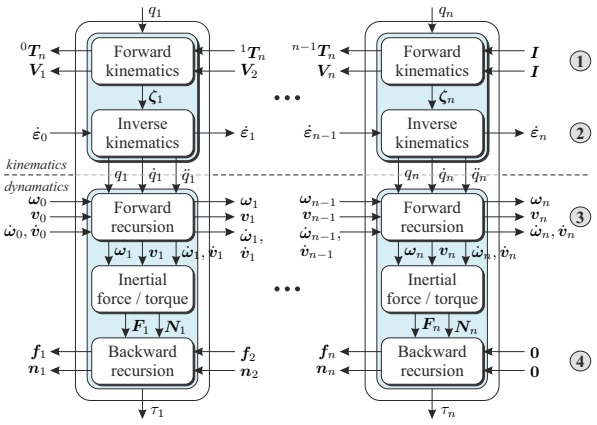


Fig. 2. Schematic overview of the modular torque computation concept.

Post-print version of the article: Michael Steinegger, Nikolaus Plaschka, Martin Melik-Merkumians, and Georg Schitter, "Vision-based probabilistic absolute position sensor," *2016 IEEE International Conference on Industrial Technology (ICIT)*, 2016. DOI: 10.1109/ICIT.2016.7474862
© 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Post-print version (generated on 17.12.2021)
This and other publications are available at:
http://www.acin.tuwien.ac.at/publikationen/ams/

The second step of the backward phase for the i^{th} controller consists of computing the matrix $\boldsymbol{V}_i \in \mathbb{R}^{6\times6}$ according to

$$\boldsymbol{V}_i = \boldsymbol{V}_{i+1}\left(\boldsymbol{I} - \boldsymbol{h}_i\boldsymbol{\zeta}_i\right) \text{ with } \boldsymbol{V}_{n+1} = \boldsymbol{I}\ , \tag{6}$$

where $\boldsymbol{I} \in \mathbb{R}^{6\times6}$ is the identity matrix. Here, the symmetric and positive semi-definite matrix $\boldsymbol{V}_i$ can be considered as orthogonal projection matrix. The corresponding projection vector $\boldsymbol{\zeta}_i \in \mathbb{R}^{1\times6}$ for the i^{th} column vector of the Jacobian matrix can then be computed as

$$\boldsymbol{\zeta}_i = \left(p_i + \boldsymbol{h}_i^{\mathrm{T}}\boldsymbol{V}_{i+1}^{\mathrm{T}}\boldsymbol{V}_{i+1}\boldsymbol{h}_i\right)^{-1}\boldsymbol{h}_i^{\mathrm{T}}\boldsymbol{V}_{i+1}^{\mathrm{T}}\boldsymbol{V}_{i+1}\ . \tag{7}$$

Therein, p_i represents a damping function which can be defined individually for each joint. This bell-shaped damping function is required in order to limit the joint velocities since they would increase unacceptable high near singular configurations of the manipulator. The choice of p_i significantly influences the accuracy of the trajectory tracking since the damping results in an offset to the desired Cartesian trajectory (see also Siciliano and Khatib [15, page 253]). Thus, the damping function should be defined in such a way that it only restricts the joint velocities in the direct neighborhood of singular configurations of the manipulator in order to avoid the mentioned tracking errors in the other regions of the manipulator workspace.

2) Forward Phase: During the forward phase, marked as ② in Fig. 2, the error according to (5) is computed by each joint controller. In the initial iteration, the controller of the first joint takes the reference input $\dot{\boldsymbol{x}}^{\text{ref}}$ as the error. It has been shown in [11], that the reference velocity input computed by

$$\dot{\boldsymbol{x}}^{\text{ref}} = \dot{\boldsymbol{x}}^{\text{des}} + \psi\boldsymbol{\varepsilon} \tag{8}$$

drives the manipulator controlled by the modular DP algorithm to the desired final position $\boldsymbol{x}^{\text{des}}$ in a quasi-optimal way. Here, $\boldsymbol{\varepsilon} \in \mathbb{R}^{6\times1}$ is the Cartesian pose (position and orientation) error and ψ is the weighting factor for the pose error. The possible contribution of the first joint / link pair to the desired movement in Cartesian space is computed and the remaining error is propagated to the next controller. Since the last error $\dot{\boldsymbol{\varepsilon}}_n$ typically is not equal to zero after the first iteration, additional iteration cycles are required. After the initial iteration, the first controller receives $\dot{\boldsymbol{\varepsilon}}_n$ and starts another iteration cycle.

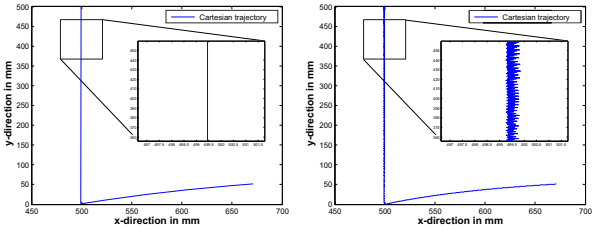
The inverse kinematics algorithm terminates when the remaining error computed by the n^{th} controller is equal to zero or a fixed number of iterations is reached. However, if specific sampling intervals have to be considered, the definition of a fixed number of iterations is absolutely necessary.

Finally, the joint velocity setpoints sent to the corresponding joint controller have to be computed. By setting $\boldsymbol{V}_{i+1} = \boldsymbol{I}$ in Eq. (7), it can be seen that $\boldsymbol{\zeta}_i$ represents the pseudo-inverse column vector for the i^{th} joint of the Jacobian $\boldsymbol{J}$. Thus, together with (5), the joint velocities can be computed as

$$\dot{q}_i = \sum_{k=1}^{M}\boldsymbol{\zeta}_i\dot{\boldsymbol{\varepsilon}}_{i-1,k} \text{ with } \dot{\boldsymbol{\varepsilon}}_{0,k} = \begin{cases}\dot{\boldsymbol{x}}^{\text{ref}} & \text{if } k=1\\ \dot{\boldsymbol{\varepsilon}}_{n,k-1} & \text{if } k>1\end{cases}\ , \tag{9}$$

where M represents the number of iterations.

Due to the overall modular structure of the described algorithm, each joint controller can compute its own contribution to the entire movement of the manipulator locally. Only the kinematic parameters have to be communicated between the individual controllers. Thus, the computational time to iteratively solve the inverse kinematics problem can be reduced since it is computed on different processors (joint controllers).



(a) additional control ON **(b)** additional control OFF

Fig. 3. End-effector movement when tracking a linear trajectory in Cartesian space with (a) and without (b) taking the orthogonal error $\boldsymbol{\epsilon}^{\perp}$ into account.

B. Extension of the Algorithm for Tracking Linear Trajectories

If the manipulator is required to track a continuous path with N sampling points, the reference input $\dot{\boldsymbol{x}}_j^{\text{ref}}, j \in \{1,\ldots,N\}$ at the j^{th} sampling point is computed according to (8). The series of reference inputs $\dot{\boldsymbol{x}}^{\text{ref}}$ can, for example, be calculated by trajectory planning algorithms based on motion primitives to further reduce the computational effort [4]. However, if the task requires exact tracking of a linear trajectory in Cartesian space, the sole application of the reference input (8) would lead to non-smooth motions of the end-effector which can also be seen in e.g. [16, Figure 23]. To guarantee a smooth motion, the orthogonal error $\boldsymbol{\epsilon}_{j,k}^{\perp}$ from the actual computed Cartesian pose and velocity to the desired values is computed in each iteration cycle k and added to the error input for the first joint controller. The resulting input $\dot{\boldsymbol{\varepsilon}}_{i,j,k}^{\perp}$ is then defined as

$$\dot{\boldsymbol{\varepsilon}}_{0,j,k}^{\perp} = \dot{\boldsymbol{\varepsilon}}_{0,j,k} + \nu\boldsymbol{\epsilon}_{j,k}^{\perp} \text{ with } \dot{\boldsymbol{\varepsilon}}_{0,j,1} = \dot{\boldsymbol{x}}_j^{\text{ref}}\ , \tag{10}$$

where ν represents the weighting factor for the orthogonal error. An exemplary Cartesian end-effector movement of a SCARA manipulator with and without considering the orthogonal error is depicted in Fig. 3. It can be seen, that with $\nu > 0$ the manipulator is forced to follow the desired linear trajectory precisely. Thus, the resulting joint angle velocities, as computed via Eq. (9), can be directly applied to compute the required torques in the next step.

III. Modular Torque Control

Precise control of modular handling systems requires to additionally consider the dynamics of the entire system based on torque control methods. It has been shown in the previous section that the modularization of the kinematic control can provide sufficiently smooth joint velocity trajectories, which can be applied as input for modular torque control based on the *recursive Newton-Euler* (RNE) algorithm (cf. Featherstone and Orin [17]). The algorithm is applied to compute the joint torques $\boldsymbol{\tau} = \{\tau_1,\ldots,\tau_n\}$ that are based on the joint angles $\boldsymbol{q}$ and their time derivatives (velocity $\dot{\boldsymbol{q}}$ and acceleration $\ddot{\boldsymbol{q}}$) in an iterative way. According to Vuskovic et al. [18], the distributable version of the RNE algorithm consists of a backward and a forward recursion.

A. Recursive Newton-Euler Algorithm: Forward Recursion

In the forward recursion, the velocities of the link corresponding to a revolute joint i are computed as

$$\boldsymbol{\omega}_i = {}^{i-1}\boldsymbol{R}_i\left(\boldsymbol{\omega}_{i-1} + \dot{\theta}_i\boldsymbol{e}_z\right)\ , \tag{11}$$

$$\boldsymbol{v}_i = \boldsymbol{\omega}_i \times \boldsymbol{\gamma}_i + {}^{i-1}\boldsymbol{R}_i\boldsymbol{v}_{i-1}\ , \tag{12}$$

Post-print version of the article: Michael Steinegger, Nikolaus Plaschka, Martin Melik-Merkumians, and Georg Schitter, "Vision-based probabilistic absolute position sensor," *2016 IEEE International Conference on Industrial Technology (ICIT)*, 2016. DOI: 10.1109/ICIT.2016.7474862
© 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Post-print version (generated on 17.12.2021)
This and other publications are available at:
http://www.acin.tuwien.ac.at/publikationen/ams/

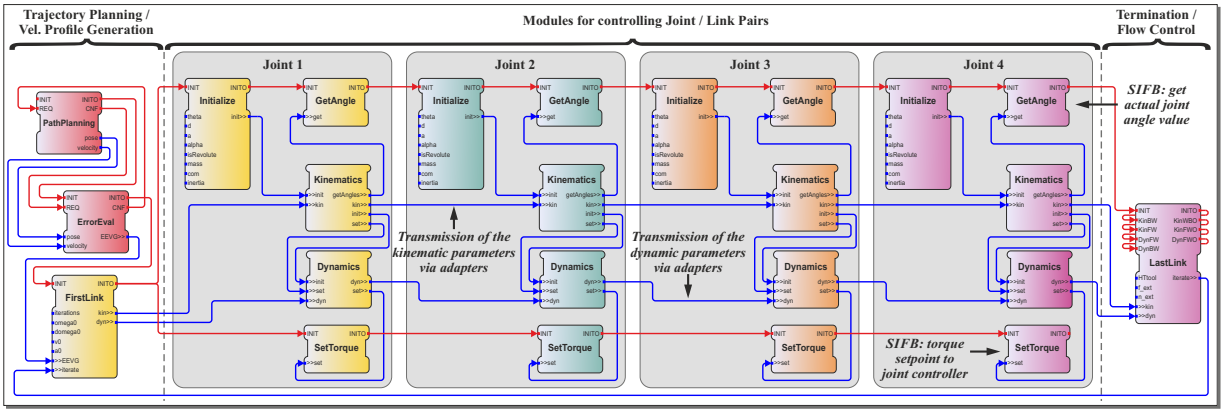


Fig. 4. Implementation example of the modular and distributed control of the SCARA robot with four degrees of freedom as described in Sec. V.

where $\boldsymbol{\omega}_i \in \mathbb{R}^{3\times1}$ is the angular velocity, $\boldsymbol{v}_i \in \mathbb{R}^{3\times1}$ represents the linear velocity, and $\boldsymbol{e}_z = [0\ 0\ 1]^{\mathrm{T}}$ is the unit vector in z-direction. For prismatic joints, the following equations apply

$$\boldsymbol{\omega}_i = {}^{i-1}\boldsymbol{R}_i \boldsymbol{\omega}_{i-1} \ , \tag{13}$$

$$\boldsymbol{v}_i = {}^{i-1}\boldsymbol{R}_i \left(\dot{d}_i \boldsymbol{e}_z + \boldsymbol{v}_{i-1} \right) + \boldsymbol{\omega}_i \times \boldsymbol{\gamma}_i \ . \tag{14}$$

The difference of these equations compared to those proposed in [18] is that, for example, in Eq. (11) the rotation axis of the considered i^{th} joint is always z_{i-1} according to the applied DH convention (see also frames in Fig. 1). Thus, the angular velocity $\boldsymbol{\omega}_{i-1}$ of the previous link is added to the contribution of the actual considered link given by $\Theta_i \boldsymbol{e}_z$. The sum of both is then projected into the frame $\mathcal{F}_i$ of the following link by multiplying it with the rotation matrix ${}^{i-1}\boldsymbol{R}_i$.

Based on (11)–(14), the accelerations for revolute joints

$$\dot{\boldsymbol{\omega}}_i = {}^{i-1}\boldsymbol{R}_i \left(\dot{\boldsymbol{\omega}}_{i-1} + \boldsymbol{\omega}_{i-1} \times \dot{\theta}_i \boldsymbol{e}_z + \ddot{\theta}_i \boldsymbol{e}_z \right) \ , \tag{15}$$

$$\dot{\boldsymbol{v}}_i = {}^{i-i}\boldsymbol{R}_i \dot{\boldsymbol{v}}_{i-1} + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{\gamma}_i + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{\gamma}_i) \ , \tag{16}$$

and for prismatic joints

$$\dot{\boldsymbol{\omega}}_i = {}^{i-1}\boldsymbol{R}_i \dot{\boldsymbol{\omega}}_{i-1} \ , \tag{17}$$

$$\begin{aligned} \dot{\boldsymbol{v}}_i = {}& {}^{i-1}\boldsymbol{R}_i \left(\ddot{d}_i \boldsymbol{e}_z + \dot{\boldsymbol{v}}_{i-1} \right) + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{\gamma}_i \\ & + 2\boldsymbol{\omega}_i \times \left({}^{i-1}\boldsymbol{R}_i \dot{d}_i \boldsymbol{e}_z \right) + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{\gamma}_i^*) \ , \end{aligned} \tag{18}$$

can be computed, where $\boldsymbol{\gamma}_i^*$ is the translation vector of the inverted homogeneous transformation matrix ${}^{i-1}\boldsymbol{T}_i$. The acceleration at the center of mass of the i^{th} link is given by

$$\dot{\boldsymbol{v}}_i^{\text{com}} = \dot{\boldsymbol{v}}_i + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{r}_i^{\text{com}} + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_i^{\text{com}}) \ , \tag{19}$$

where $\boldsymbol{r}_i^{\text{com}}$ is the Cartesian position vector of the center of mass, and is applied to compute the inertial force

$$\boldsymbol{F}_i = m_i \dot{\boldsymbol{v}}_i^{\text{com}} \ , \tag{20}$$

with the mass m_i of the i^{th} link. The last step of the forward recursion is the computation of the inertial torques given as

$$\boldsymbol{N}_i = \boldsymbol{\Xi}_i \dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times (\boldsymbol{\Xi}_i \boldsymbol{\omega}_i) \ , \tag{21}$$

where $\boldsymbol{\Xi}_i$ is the inertia matrix of the i^{th} link.

The forward recursion is marked as ③ in Fig. 2. It can be seen that only the results of Eq. (11)–(19) have to be transmitted between the individual joint controllers. The computation of the forces (20) and torques (21) can be computed locally and are then used as input for the subsequent backward recursion.

B. Backward Recursion

In Fig. 2, the backward recursion is marked with ④ and represents the last step of the entire modular and distributable control algorithm. During the backward recursion the interlinked forces given as

$$\boldsymbol{f}_i = {}^{i+1}\boldsymbol{R}_i \boldsymbol{f}_{i+1} + \boldsymbol{F}_i \tag{22}$$

and the interlinked torques

$$\boldsymbol{n}_i = \boldsymbol{N}_i + {}^{i+1}\boldsymbol{R}_i \boldsymbol{n}_{i+1} + \boldsymbol{r}_i^{\text{com}} \times \boldsymbol{F}_i + \boldsymbol{\gamma}_i^* \times \left({}^{i+1}\boldsymbol{R}_i \boldsymbol{f}_{i+1} \right) \tag{23}$$

are computed. Finally, the joint torques can be computed as

$$\tau_i = \boldsymbol{n}_i^{\mathrm{T}} \boldsymbol{e}_z \tag{24}$$

which can be directly transmitted as setpoint to the controller corresponding to the i^{th} joint / link pair.

IV. Implementation

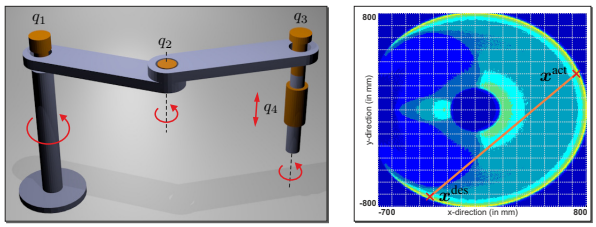
The distributable torque control algorithm described in the previous sections is implemented according to the international standard IEC 61499 in the 4DIAC (*Framework for Distributed Industrial Automation and Control*) integrated development environment (IDE) [19]. Since the standard IEC 61499 utilizes the *function block* (FB) concept, which is well-suited for modularization, and enables the distribution of individual subparts of an entire IEC 61499 program, the 4DIAC framework is the preferred implementation environment.

A. Modularization

Both, the kinematic and the dynamic control algorithm, are implemented as *composite function blocks* (CFBs) and represent the core building blocks of a single joint control (cf. Fig. 4). These CFBs represent logically grouped and encapsulated FB sub-networks with individual FBs for computing, for example, the Jacobian matrix or the transformation matrices according to (1). Here, all mathematical computation methods contained in the *Kinematics* and *Dynamics* CFBs are implemented in C++ based on the linear algebra library Armadillo [20].

Post-print version of the article: Michael Steinegger, Nikolaus Plaschka, Martin Melik-Merkumians, and Georg Schitter, "Vision-based probabilistic absolute position sensor," *2016 IEEE International Conference on Industrial Technology (ICIT)*, 2016. DOI: 10.1109/ICIT.2016.7474862
© 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Post-print version (generated on 17.12.2021)
This and other publications are available at:
http://www.acin.tuwien.ac.at/publikationen/ams/



(a) (b)

Fig. 5. Simulation of the SCARA robot (a) with density plot corresponding to the damping factors p_i in the workspace and Cartesian trajectory (b).

In Fig. 4, the implementation of the modular and distributed control for a SCARA manipulator is depicted, which is applied for evaluation as described in Sec. V. It can be seen that each joint control network contains one *Initialize* FB which requires the DH parameters $\Theta_i, d_i, a_i, \alpha_i$ of the individual joint / link pair as well as the dynamic parameters $(m_i, \boldsymbol{r}_i^{\text{com}}, \boldsymbol{\Xi}_i)$ as input. Furthermore, the data input *isRevolute* specifies which equations (11)–(18) are applied for computing the velocities $\boldsymbol{\omega}_i, \boldsymbol{v}_i$ and the accelerations depending on the type of joint (prismatic or revolute). The specific parameters are then communicated to the *Kinematics* and *Dynamics* FBs via bidirectional adapter connections. Adapters are used to reduce the amount of connections in a FB network since they combine different event and data connections into a single common interface between FBs. This significantly increases the readability of control programs implemented according to the standard IEC 61499.

As shown in Fig. 4, the first and last joint control networks (joint 1 and joint 4) additionally require delimiter FBs for the coordination of the individual phases of the kinematic and dynamic control as described in the previous sections. These FBs are denoted as *FirstLink* and *LastLink* in Fig. 4. The *FirstLink* FB receives the pose and velocities error which is computed by the *ErrorVal* FB as the error between the actual Cartesian pose and velocity and the path way-points (pose and velocity) transmitted from the superior path planning FB.

The *LastLink* incorporates the control of the program flow and triggers an iteration when events are received at the input *KinBW*. This starts the backward propagation of the solutions of (1) and (6) to the *Kinematics* CFBs. The signal is then reflected at the *FirstLink* block and the forward propagation of (5) is started. After a predefined number of iterations, specified at the input *iterations* of the *FirstLink* FB, the joint angle velocities are computed according to (9) and propagated to the *Dynamics* CFBs. Then, the forward and the backward recursion for computing the desired joint torques is started by the *LastLink* FB and the torque setpoints are finally transmitted to the underlying joint controller via the *SetTorque* FBs. After setting the torques, the computation process is triggered with the new way-point received from the trajectory planning FB.

B. Distribution

The implemented FBs can be mapped to different resources (also on different devices like the joint controllers) as it can be seen in Fig. 4 (different coloring of the FBs). Therefore, the direct connections between FBs which should be mapped to different joint controllers has to be replaced by standard communication blocks. This can be achieved with publish/subscribe or client/server FBs which are neglected in Fig. 4 for readability reasons. These communication FBs can also be directly implemented in the resource model such that they do not show up in the application model (for a detailed overview on the IEC 61499 models it is referred to [21]). Furthermore, the joint-specific control modules were implemented with the 4DIAC-IDE since the underlying IEC 61499 distribution model enables the distribution of a control application across different joint controllers running the 4DIAC runtime environment (FORTE).

V. Simulation Results

In order to evaluate the proposed modular and distributable torque control method it is applied to a simulation of a SCARA manipulator which is depicted in Fig. 5.

A. Description of the Setup

The SCARA robot consists in total of four joints, where three are revolute (q_1, q_2, q_3) and one is a prismatic joint (q_4). For evaluation, the simulation environment V-REP (*Virtual Robot Experimentation Platform*) of the vendor Coppelia Robotics is used and interfaced by implemented *service interface function blocks* (SIFBs) in 4DIAC. These SIFBs enable the transmission between the implementation and the simulation environment of all required sensor values and the torque setpoints computed by the proposed modular control algorithm. Therefore, the dynamics engine *BULLET* of the V-REP simulator is applied and switched to torque/force input mode in order to control the SCARA robot with torque setpoints. Kinematic and dynamic parameters of the manipulator are specified as standardized COLLADA (*COLLAborative Design Activity*) model and directly imported by the simulation environment.

The SIFBs representing the simulator interface are named with *GetAngle* and *SetTorque* in Fig. 4. This blocks have to be replaced with SIFBs interfacing the underlying system I/Os when applying the control algorithm to a real robotic system. An additional simulator control SIFB enables the synchronization between 4DIAC and the V-REP simulator in order to ensure the desired execution and sampling time.

B. Results and Discussion

For evaluation, the manipulator is requested to follow a linear trajectory in Cartesian space as depicted in Fig. 5. The Cartesian positions were defined as $\boldsymbol{x}^{\text{act}} = (738, 307, 160)\,\text{mm}$ and $\boldsymbol{x}^{\text{des}} = (-338, -728, 160)\,\text{mm}$. The number of iterations for computing the joint velocities according to (9) is set to $M = 20$. The simulation is carried out on an Intel Core i5-4300M processor with 2.6 GHz and 8 GB RAM.

The resulting joint angle velocities are depicted in Fig. 6. It can be seen that the Cartesian trajectory of the manipulator end-effector starts near a singular configuration and the final pose

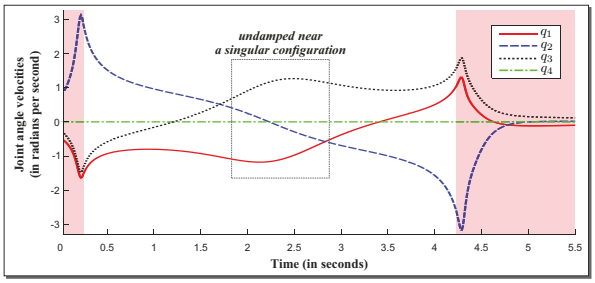


Fig. 6. Joint velocities in undamped and damped regions (red regions) while tracking a straight line in Cartesian space with the SCARA manipulator.

Post-print version of the article: Michael Steinegger, Nikolaus Plaschka, Martin Melik-Merkumians, and Georg Schitter, "Vision-based probabilistic absolute position sensor," *2016 IEEE International Conference on Industrial Technology (ICIT)*, 2016. DOI: 10.1109/ICIT.2016.7474862
© 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Post-print version (generated on 17.12.2021)
This and other publications are available at:
http://www.acin.tuwien.ac.at/publikationen/ams/

also represents a singular configuration. The initial position was assumed as an intermediate position where the desired velocities are unequal to zero. Near singular configurations, the damping function limits the joint angle velocities as, for example, seen for q_2 between $t = 0\,\mathrm{s}$ to $0.2\,\mathrm{s}$. When approaching the final singular pose, the joint angle velocities are also damped and the joint angles converge towards the desired values. However, the convergence rate decreases drastically the more the manipulator reaches the singular configuration which is caused by the damping function and can be directly observed in Fig. 6 after $t > 0.75\,\mathrm{s}$. If, however, the manipulator does not approach a singular configuration but reaches one of the joint angle limits, the controllers of the remaining joints would try to compensate for this restriction. In general, the limitations of joint angles can be compensated as long as the remaining number of joints is equal or greater than the number of degrees of freedom in Cartesian space (kinematic redundancy). When the remaining joints cannot compensate the limitations, the joints are controlled by the proposed algorithm such that the end-effector follows the desired trajectory as close as possible.

The computation of the desired torques for a single point-to-point movement takes in average $5\,\mathrm{ms}$ for $M = 100$ iterations. With the mentioned number of iterations, the desired Cartesian pose is reached with an accuracy in the sub-millimeter range in undamped regions. In damped regions, the achievable accuracy strongly depends on the distance to the singular configuration and the choice of the damping function. However, the presented framework shows good results and represents an important step towards an entirely modularized robotic system.

VI. Conclusion and Outlook

This paper presents a modular and distributable approach for kinematic and dynamic control of serial handling systems. The approach proposed by Casalino and Turetta [11] is extended in terms of an additional control input $\epsilon^{\perp}$ to guarantee a smooth joint angle and velocity series even for linear Cartesian trajectory tracking. Based on the extended modular kinematic control the dynamic control is achieved with the well-known recursive Newton-Euler algorithm. The entire control algorithm for computing the torques to precisely follow a desired trajectory in Cartesian space is implemented according to the international standard IEC 61499. The kinematic and dynamic control of a single joint has been implemented in two different FBs which can be mapped to the according joint controller. This enables the distribution of the entire torque control algorithm over different devices and could reduce the computational time.

For each joint / link pair the corresponding FBs are instantiated and parametrized with the Denavit-Hartenberg parameters and all parameters for dynamic control like mass, center of gravity etc. Therefore, the proposed modular and distributable approach enables the fast and easy reconfiguration of the underlying handling system in terms of the control software.

The future work is concerned with automatically generating the entire control algorithm based on a formalized model of the applied handling system. Since most of the robotic system vendors provide the description of their systems as COLLADA files, such formal models can be used as input for a knowledge-based code generation approach as presented by Steinegger and Zoitl [22] in order to automatically instantiate, configure, parametrize, and also interconnect the different required FBs for controlling the entire modular robotic system.

Acknowlegment

Financial support of the Austrian Research Promotion Agency (FFG) under grant no. 848623 is gratefully acknowledged.

References

[1] R. Duray, P. T. Ward, G. W. Milligan, and W. L. Berry, "Approaches to mass customization: Configurations and empirical validation," *Journal of Operations Management*, vol. 18, pp. 605–625, 2000.

[2] G. G. Rogers and L. Bottaci, "Modular production systems: A new manufacturing paradigm," *Journal of Intelligent Manufacturing*, vol. 8, no. 2, pp. 147–156, 1997.

[3] S. Schaal, "Dynamic movement primitives – A framework for motor control in humans and humanoid robots," in *The Int. Symposium in adaptive Motion of Animals and Machines*, 2003.

[4] M. Steinegger, B. Passenberg, M. Leibold, and M. Buss, "Trajectory planning for manipulators based on the optimal concatenation of LQ control primitives," in *Proc. of the 50th IEEE Conf. on Decision and Control and European Control Conference*, 2011, pp. 2837–2842.

[5] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems: Challenges and oportunities for the future," *IEEE Robotics & Automation Magazine*, pp. 43–52, 2007.

[6] A. G. O. Mutambara and H. F. Durrant-Whyte, "Estimation and control for a modular wheeled mobile robot," *IEEE Trans. on Control Systems Technology*, vol. 8, no. 1, pp. 35–46, 2000.

[7] C. Ünsal and P. K. Khosla, "Mechantronic design of a modular self-reconfiguring robotic system," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2000, pp. 1742–1747.

[8] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2002.

[9] D. B. Stewart and P. K. Khosla, "Rapid development of robotic applications using component-based real-time software," in *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems*, 1995, pp. 465–470.

[10] L.-C. T. Wang and C. C. Chen, "A combined optimization method for solving the inverse kinematics problem of mechanical manipulators," *IEEE Trans. on Robotics and Automation*, vol. 7, pp. 489–499, 1991.

[11] G. Casalino and A. Turetta, "Dynamic programming based, computationally distributed control of modular manipulators in the operational space," in *Proc. of the IEEE Int. Conf. on Mechantronics and Automation*, 2005, pp. 1460–1467.

[12] Int. Electrotechnical Commision, "IEC 61499 – Function blocks, Part 1: Architecture," January 2005.

[13] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, 1st ed. CRC Press, March 1994.

[14] J. Denavit and R. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *Trans. of the ASME / Journal of Applied Mechanics*, vol. 22, pp. 215–221, 1955.

[15] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. Springer Verlag, 2008.

[16] A. Turetta, G. Casalino, and A. Sorbara, "Distributed control architecture for self-reconfigurable manipulators," *Int. Journal of Robotics Research*, vol. 27, pp. 481–504, 2008.

[17] R. Featherstone and D. Orin, "Robot dynamics: Equations and algorithms," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, vol. 1, 2000, pp. 826–834.

[18] M. Vuskovic, T. Liang, and K. Anantha, "Decoupled parallel recursive newton-euler algorithm for inverse dynamics," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1990, pp. 832—838.

[19] T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sünder, A. Valentini, and A. Martel, "Framework for distributed industrial automation and control (4DIAC)," in *IEEE Int. Conf. on Industrial Informatics*, 2008.

[20] C. Sanderson, "Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments," National Information Communications Technology Australia Research Center (NICTA), Tech. Rep., 2008.

[21] R. Lewis, *Modelling Control Systems Using IEC 61499: Applying Function Blocks to Distributed Systems*, ser. IEE Control Engineering. The Institution of Electrical Engineers, 2001, no. 59.

[22] M. Steinegger and A. Zoitl, "Automated code generation for programmable logic controllers based on knowledge acquisition from engineering artifacts: Concept and case study," in *IEEE Int. Conf. on Emerging Technologies & Factory Automation*, 2012, pp. 1–8.

Post-print version of the article: Michael Steinegger, Nikolaus Plaschka, Martin Melik-Merkumians, and Georg Schitter, "Vision-based probabilistic absolute position sensor," *2016 IEEE International Conference on Industrial Technology (ICIT)*, 2016. DOI: 10.1109/ICIT.2016.7474862
© 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.