PDEEC@FEUP: Special Topics
2020/2021: 2nd Semester

Industrial Robotics: Multivariable Control

Teaching Assistance (TA) at the curricular unit Industrial Robotics (EEC0093) of MIEEC

Student: Ricardo B. Sousa[1,2]
Supervisor: António P. Moreira[1,2]

[1] Faculty of Engineering of the University of Porto
(email: `up201503004@edu.fe.up.pt`, `amoreira@fe.up.pt`)
[2] INESC TEC – Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência
(email: `ricardo.b.sousa@inesctec.pt`)

Abstract. The course unit Special Topics of the ECE Ph.D. program at FEUP proposed two activities. The one chosen and reported in this work was being a teaching assistant in the curricular unit Industrial Robotics. The work plan was to introduce a new topic into the unit's curricular plan: inverse dynamics with multivariable control on robot manipulators. Multivariable control allows the consideration of effects such as the gravity, inertia, and coupling effect between the joints in the closed-loop control system. One difference from independent joint control is not treating the effects as disturbances. The dynamics of an n-link robot manipulator were defined, and then the multivariable control scheme was formulated. The students had the opportunity of observing the improved tracking performance of multivariable control over independent control when the robot was subject to effects such as gravity. A practical example on a 3-link SCARA manipulator was given to the students to consolidate the topic, with the inverse dynamics computed using symbolic math from MATLAB.

1 Introduction

Industrial Robotics (RIND) [1] is a curricular unit that belongs to the specialization in Robotics of the Master's degree in Electrical and Computers Engineering (ECE) at the Faculty of Engineering of the University of Porto (FEUP). The current curricular plan of RIND formulates the robot manipulators' control with control laws for each joint based on a Single-Input-Single-Output (SISO) model - the so-called independent joint control. This approach usually implements proportional-derivatives controllers for positioning control of each joint and inverse dynamics with feed-forward controllers for eliminating the offset in steady-state [2].

However, coupled effects are regarded as disturbances to the individual systems. Indeed, effects such as the gravity, centripetal and/or centrifugal forces, or inertia are not considered in independent joint control. Also, if the parameters of the system are not known exactly or unmodelled dynamics are not considered such as joint flexibility, the SISO model's ideal performance cannot longer be guaranteed [2].

Multivariable control focuses on implementing robust and adaptive control to maintain performance in terms of stability, tracking error despite nonlinearity, dynamic model changes, and external disturbances, among others. Indeed, multivariable control provides a more rigorous analysis of the performance of control systems, while designing robust and adaptive nonlinear control laws that guarantee stability and tracking of arbitrary trajectories [2]. In the literature, multivariable control usually considers the dynamics of robot manipulators in the control laws' formulation. Furthermore, multivariable control is implemented with inverse dynamics [3,4,5,6], PD controllers (in the scope of multivariable control) [7,8], Euler-Lagrange models [9], Newton-Euler formulations [10,11,12], fuzzy control [13,14,15], adaptive control [16,17] and adaptive fuzzy control [18,19], predictive control [20], and neural networks [17,21,22,23].

The goal of this work is the introduction of multivariable control into the Industrial Robotics's curricular plan. The basis for studying multivariable control was the book *Robot modeling and control* [2] that is part of the curricular unit's mandatory literature. Given my interest in teaching high education alongside research, I will introduce this module in a theoretical followed by laboratory work to implement inverse dynamics in a SimTwo-based simulation environment [24], as a teaching assistant. First, the formulation of the inverse dynamic for a Selective Compliance

Assembly Robot Arm (SCARA) robot manipulator was tested in SimTwo [24]. Then, a presentation and a guide for the laboratory work were elaborated for the students.

The rest of this report is organized as follows. Section 2 presents the related work. Section 3 explains the concept of dynamics applied to robot manipulators. Section 4 formulates the concept of inverse dynamics with multivariable control. Section 5 presents an example of applying inverse dynamics on a manipulator (specifically, a SCARA). Section 6 explains the elaboration of the presentation for the theoretical class. Section 7 explains the laboratory work proposed for the students. Lastly, Section 8 presents the conclusions of this work.

2 Related Work

Different control techniques have been proposed in the literature for either parallel or serial manipulators control. In the case of parallel manipulators, their closed-loop structure increases the complexity of analyzing the dynamic model. However, several works proposed methods for solving the inverse dynamics problem. Tsai (1999) [3] proposed a methodology for deriving the dynamic motion equations of a Stewart-Gough (6 translation Degrees of Freedom – DoF) manipulator based on the principle of virtual work. With a similar approach to [3], Zhao and Gao (2009) [6] focused on a 6 DoF seismic simulator parallel manipulator. Staicu *et al.* (2007) [5] proposed recursive matrix relations for the kinematics and dynamics of a HALF (2 translation and 1 rotational DoF) parallel manipulator. Zhang and Song (1993) [10] used the Newton-Euler formulation of inverse dynamics for parallel manipulators focusing on the method's computational efficiency. [10] claimed similar computational effort relative to serial manipulators. Also, Dasgupta and Mruthyunjaya (1998) [11] and Li *et al.* (2003) [12] formulated the inverse dynamics using the Newton-Euler approach for the Stewart platform and a HALF parallel manipulator, respectively. Based on the principle that a Stewart platform has a relatively smaller workspace compared to serial manipulators, Lee *et al.* (2003) [4] simplified the inverse dynamics problem approximating nonlinear coefficient matrices to constant ones.

As for serial manipulators, Valle *et al.* (2002) [9] formulated a multivariable predictive control using the Euler-Lagrange formulation of inverse dynamics. Cuvillon *et al.* (2012) [20] also used a generalized predictive control with H_∞ control for a 2 DoF planar manipulator accounting for all the dynamics of the system. Saab and Ghanem (2018) [7] and Saab and Jaafar (2021) [8] formulated multivariable stochastic controllers considering noisy position and velocity measurements. The difference is that Saab and Jaafar [8] only uses position measurements, unlike Saab and Ghanem [7] that requires position and velocity data of the joints. The works of Santibanez *et al.* (2000)[13] and Mahmoodabadi and Ziaei (2019) [15] used fuzzy Proportional-Derivative (PD) controllers for multivariable control of the serial manipulators' dynamics. Shaocheng *et al.* (2000) [14] developed a robust fuzzy adaptive control scheme with the H_∞ control technique to account for the uncertainties relative to the model's parameters. Chiou and Huang (2005) [18] proposed a model-free adaptive fuzzy sliding mode controller to hit a user-defined sliding surface, and then slide along it to approach a reference model. The initialization of the parameters for fuzzy control could be initialized at zero because an online parameter tuning algorithm was implemented derived from the Lyapunov stability theory. Bobaşu and Popescu (2006) [16] and Lei and Wu (2006) [17] used multivariable adaptive controllers for the tracking control of robot manipulators. The former used the gradient algorithm and the exact feedback input-output linearization model to design the adaptive nonlinear control. The latter also focused on the problem of unknown dynamics using the weighted least-squares estimation method and a compensation neural network law to reduce the influence of the coefficients estimation error on the control performance. Also, Chen *et al.* (2014) [19] proposed 2 types of robust adaptive inverse dynamics control schemes for the trajectory tracking control of manipulators with uncertain dynamics: an adaptive fuzzy control algorithm (to approximate the structured uncertainties) and a nonlinear H_∞ controller (to synthesize controllers to achieve stabilization). Lastly, the works of Moradi and Malekizade (2013) [21], Yu *et al.* (2014) [22] and Polydoros *et al.* (2015) [23] used neural networks for the trajectory tracking problem of robot manipulators.

In summary, the analysis of the literature showed six main trends for multivariable control of robot manipulators: multivariable control with PD controllers, predictive controllers, fuzzy control, adaptive and robust fuzzy control, sliding mode control, and neural networks. Nevertheless, the works that implemented multivariable control considering the manipulator's dynamics usually required formulating the equations of motions.

3 Dynamics

The dynamics of robot manipulators are introduced through Euler-Lagrange equations in this section. These equations describe the evolution of a mechanical system subject to holonomic constraints from the principle of virtual work in the general case. First, the Lagrangian of the system must be defined, i.e., the difference between kinetic and potential energy. Then, the dynamic equations of robot manipulators can be derived. The analysis presented in this section and in section 4 is based on the chapters 6 and 8 of the book [2], respectively.

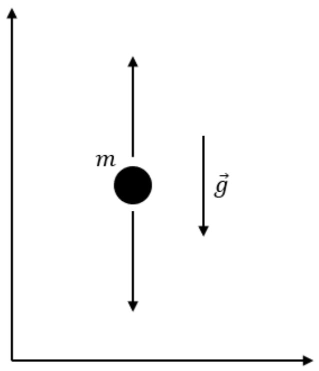


Fig. 1. One Dimensional System

3.1 One dimensional system

Let us first analyze a one-dimensional system composing of a particle of mass m, constrained to move only in the y-direction, and subject to the gravitational force mg and an external force f. This system is illustrated in figure 1. The equation of motion of the particle can be derived from Newton's Second Law – the net force on a particle is equal to its mass m times its acceleration $\ddot{y}$ –, as illustrated in equation 1.

$$m\ddot{y} = f - mg \tag{1}$$

The kinetic energy $\mathcal{K}$ of the particle due to its motion can be defined as in equation 2. Equation 3 defines the potential energy $\mathcal{P}$ of the particle due to the gravity g. Then, the Lagrangian ($\mathcal{L}$) of the system can be defined by the difference of the kinetic ($\mathcal{K}$) and potential ($\mathcal{P}$) energy, as formulated in equation 4. The Lagrangian defines a mechanical system depending on a set of independent generalized coordinates that characterize the possible motion of the variable. The use of the Lagrangian eliminates the requirement of defining the constraint force to enter into the resultant system of equations.

$$\mathcal{K} = \frac{1}{2}m\dot{y}^2 \tag{2}$$

$$\mathcal{P} = mgy \tag{3}$$

$$\mathcal{L} = \mathcal{K} - \mathcal{P} \tag{4}$$

Finally, we can defined the motion of the particle through the Euler-Lagrange formulation presented in equation 5. This formulation defines the dynamic equations of motion equivalent to the Newton's Second Law.

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{y}} - \frac{\partial \mathcal{L}}{\partial y} = m\ddot{y} + mg = f \tag{5}$$

3.2 Single link manipulator

In the case of a single link manipulator (illustrated in figure 2), the angles of the link (θ_l) and the motor shaft (θ_m) are related by the gear reduction ratio (r): $\theta_m = r\theta_l$. This relation means that the system has only one DoF, similar to the previous example. The kinetic energy ($\mathcal{K}$) of this system is defined in equation 6. Unlike the previous example (kinetic energy due to linear motion), the single link manipulator with a revolute joint only has kinetic energy due to the rotational motion of the link. The kinetic energy depends on the angular velocity of the motor shaft ($\dot{\theta}_m$) and link ($\dot{\theta}_l$), and also on the rotational inertias of the motor (J_m) and the link (J_l).

$$\mathcal{K} = \frac{1}{2}J_m\dot{\theta}_m^2 + \frac{1}{2}J_l\dot{\theta}_l^2 = \frac{1}{2}\left(r^2 J_m + J_l\right)\dot{\theta}_l^2 = \frac{1}{2}J\dot{\theta}_l^2 \tag{6}$$

The potential energy ($\mathcal{P}$) of the manipulator due to gravity is given by equation 7. This equation considers that the link's total mass (M) is concentrated at the link's center of mass.

$$\mathcal{P} = Mgl_c\sin\left(\theta_l\right) \tag{7}$$

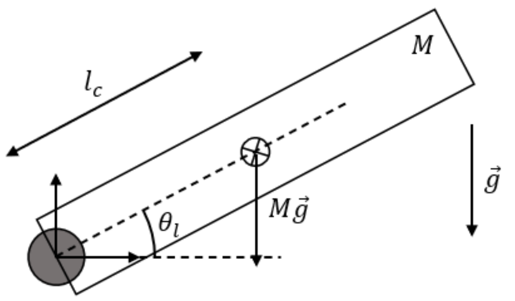


Fig. 2. One Dimensional Single Link Manipulator

Then, it is possible to define the Lagrangian ($\mathcal{L}$) of the system and the Euler-Lagrange motion equation by the equations 8 and 9, respectively. Considering the torque of the motor (u) and the damping motor (B_m) and link (B_l) torques (where $B = rB_m + B_l$), the generalized force τ_l represents all forces and torques applied to the link.

$$\mathcal{L} = \mathcal{K} - \mathcal{P} = \frac{1}{2} J\dot{\theta}_l^2 - Mgl_c \sin(\theta_l) \tag{8}$$

$$J\ddot{\theta}_l + Mgl_c \cos(\theta_l) = u - B\dot{\theta}_l = \tau_l \tag{9}$$

3.3 General case

In general, the Euler-Lagrange equations lead to n second-order nonlinear ordinary differential equations, as illustrated in equation 10. The number n is determined by the number of generalized coordinates that are required to describe the evolution of the system. One possible formulation of generalized coordinates for an n-link robot manipulator is using the n Denavit-Hartenberg joint variables.

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i, \text{ where } i = 1, ..., n \tag{10}$$

Note that the advantage of using the Euler-Lagrange formulation of dynamic equations of motion is not requiring the definition of constraint forces (forces required to make the constraints of an n-link robot manipulator hold). However, this method only works with holonomic constraints (e.g., two particles connected by a massless rigid wire). Also, it is necessary to define generic expressions for the kinetic ($\mathcal{K}_i$) and potential ($\mathcal{P}_i$) energy of a generic n-link manipulator for computing the Lagrangians ($\mathcal{L}_i$) and generalized forces (τ_i) of the system, in terms of a set of generalized coordinates.

3.3.1 General expressions for kinetic energy

First, the kinetic energy ($\mathcal{K}$) of a rigid object is the sum of two terms: translational energy ($\mathcal{K}_{trans.}$) concentrating the entire object's mass at its center of mass and rotational energy ($\mathcal{K}_{rot.}$) around the object's center of mass. Equation 11 defines the sum of translational and rotational kinetic energies, where m is the total mass of the object, v and ω are the linear and angular velocity vectors, respectively, and $\mathcal{I}$ is a symmetric 3×3 matrix called the Inertia Tensor (is defined later in the report). The latter must be defined relative to the inertial coordinate frame, i.e., coordinate frame where the direction of the gravity vector is defined and independent of the generalized coordinates.

$$\mathcal{K} = \mathcal{K}_{trans.} + \mathcal{K}_{rot.} = \frac{1}{2} mv^T v + \frac{1}{2}\omega^T \mathcal{I} \omega \tag{11}$$

In the case of an n-link robot manipulator, the linear and angular velocities of any point of any link can be expressed in terms of a Jacobian matrix and the derivative of the joint variables ($\dot{q}$). Considering the joint variables (q) as the generalized coordinates (DH convention), it is possible to define the linear (v_{c_i}) and angular (ω_{c_i}) velocities at each link's center of mass relative to the inertial coordinate frame in terms of these generalized coordinates and the Jacobian matrices $J_{v_{c_i}}$ and $J_{\omega_{c_i}}$. Note that $J_{v_{c_i}}$ and $J_{\omega_{c_i}}$ represent the linear and angular part, respectively, of the Jacobian matrices relative to the center of mass of link i. The linear (v_{c_i}) and angular (ω_{c_i}) velocities are formulated in equations 12 and 13, respectively.

$$v_{c_i} = J_{v_{c_i}}(q)\,\dot{q} \tag{12}$$

$$\omega_{c_i} = J_{\omega_{c_i}}(q)\,\dot{q} \tag{13}$$

Next, the overall kinetic energy ($\mathcal{K}$) of the manipulator is formulated as in equation 14. The entire mass (m_i) of each link is concentrated at the link's center of mass, and the link i's inertia tensor (I_i) is evaluated around a coordinate frame parallel to frame i (defined by DH convention) but whose origin is at the link's center of mass. Equation 15 defines the kinetic energy of the n-link manipulator in matrix form. $D(q)$ is the so-called inertia matrix, and it is a symmetric positive definite matrix. In general, this matrix is configuration-dependent, i.e., dependent on the kinematic arrangement of the manipulator.

$$\mathcal{K} = \frac{1}{2}\dot{q}^T\left(\sum_{i=1}^{n} m_i J_{v_{c_i}}(q)^T J_{v_{c_i}}(q) + J_{\omega_{c_i}}(q)^T R_{c_i}(q)\, I_i R_{c_i}(q)^T J_{\omega_{c_i}}(q)\right)\dot{q} \tag{14}$$

$$\mathcal{K} = \frac{1}{2}\dot{q}^T D(q)\,\dot{q} \tag{15}$$

Inertia tensor In equation 11, we defined the kinetic energy ($\mathcal{K}$) of an object of mass m as dependent on its Inertia Tensor ($\mathcal{I}$). This tensor represents the moment of inertia relative to the inertial frame. However, we can define the inertia tensor relative to a coordinate frame attached to the object's center of mass (I). In the scope of robot manipulators and generalized coordinates, this coordinate frame is parallel to frame i, as already stated. The relation between the inertia tensors relative to different frames is given by equation 16, where R_c represents the orientation matrix of the frame attached to the link's center of mass relative to the inertial frame.

$$\mathcal{I} = R_c I R_c^T \tag{16}$$

The advantage of defining the inertia tensor expressed in the link's attached frame (I) is that I is a constant matrix (defined in equation 17) independent of the motion of the link.

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \tag{17}$$

The diagonal terms I_{xx},I_{yy},I_{zz} are the principal moments of inertia around the x,y,z axes, respectively, and the other terms are the cross-products of inertia. Each term of the matrix I can be defined as follows:

- $I_{xx} = \int\int\int\left(y^2+z^2\right)\rho(x,y,z)\,dxdydz$
- $I_{yy} = \int\int\int\left(x^2+z^2\right)\rho(x,y,z)\,dxdydz$
- $I_{zz} = \int\int\int\left(x^2+y^2\right)\rho(x,y,z)\,dxdydz$
- $I_{xy} = I_{yx} = -\int\int\int xy\rho(x,y,z)\,dxdydz$
- $I_{xz} = I_{zx} = -\int\int\int xz\rho(x,y,z)\,dxdydz$
- $I_{yz} = I_{zy} = -\int\int\int yz\rho(x,y,z)\,dxdydz$

Figure 3 illustrates the uniform rectangular solid, i.e., its mass density ($\rho = m/V$) is constant. Given that it is an uniform object and considering the frame attached to the object's center of mass concentric with its geometric center, then by symmetry, the cross products of inertia are 0. The terms of the inertia tensor I of the uniform rectangular solid are defined as follows:

- $I_{xx} = \int_{-l_z/2}^{l_z/2}\int_{-l_y/2}^{l_y/2}\int_{-l_x/2}^{l_x/2}\left(y^2+z^2\right)\rho(x,y,z)\,dxdydz = \rho\frac{l_xl_yl_z}{12}\left(l_y^2+l_z^2\right)$
- $I_{yy} = \rho\frac{l_xl_yl_z}{12}\left(l_x^2+l_z^2\right)$
- $I_{zz} = \rho\frac{l_xl_yl_z}{12}\left(l_x^2+l_y^2\right)$
- Cross-products of inertia (I_{xy}, I_{yx}, I_{xz}, etc.) are 0

3.3.2 General expressions for potential energy

The overall potential energy ($\mathcal{P}$) of an n-link robot manipulator is computed as in equation 18. This energy is the sum of each link's potential energy ($\mathcal{P}_i$) due to gravity (vector g is the direction of gravity in the inertial frame) assuming

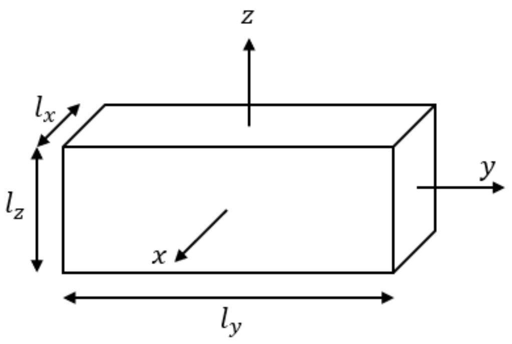


Fig. 3. Uniform Rectangular Solid

that the entire mass of the link (m_i) is concentrated at its center of mass (vector o_{c_i} gives the coordinates of the link's center of mass relative to the inertial frame).

$$\mathcal{P} = \sum_{i=1}^{n} \mathcal{P}_i = \sum_{i=1}^{n} -g^T o_{c_i} m_i \tag{18}$$

If the robot contains elasticity (e.g., flexible joints), the overall potential energy will include terms that represent the energy stored in the elastic elements. Also, note that the potential energy only depends on the robot's configuration and the joints' position (q) but not on the joints' velocity $(\dot{q})$.

3.4 Equations of motion

Now, we can define the Euler-Lagrange equations based on the system's Lagrangian $(\mathcal{L})$ for an n-link robot manipulator. This formulation is illustrated in equation 19, where d_{kj} is the k,j-th element of the inertia matrix $(D(q))$, c_{ijk} are the Christoffel symbols of the first kind, and $\partial P/\partial q_k$ is the gravity effect relative to each joint.

$$\begin{gathered}
\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_k} - \frac{\partial \mathcal{L}}{\partial q_k} = \tau_k \Leftrightarrow \\
\sum_j d_{kj}\ddot{q}_j + \sum_{i,j}\left(\frac{\partial d_{kj}}{\partial q_i} - \frac{1}{2}\frac{\partial d_{ij}}{\partial q_k}\right)\dot{q}_i\dot{q}_j + \frac{\partial P}{\partial q_k} = \tau_k \Leftrightarrow \\
\sum_j d_{kj}\ddot{q}_j + \sum_{i,j} c_{ijk}\dot{q}_i\dot{q}_j + \frac{\partial P}{\partial q_k} = \tau_k
\end{gathered} \tag{19}$$

The Christoffel symbols (c_{ijk}) are defined in equation 20. These symbols reduce in half the computation effort due to $c_{ijk}(q) = c_{jik}(q)$ for a given k. Analyzing the terms $\sum_{i,j} c_{ijk}\dot{q}_i\dot{q}_j$, there are two types: terms involving a product of the type $\dot{q}_i^2$, and terms of the type $\dot{q}_i\dot{q}_j$ $(i \neq j)$. The former terms are called centrifugal, and the latter ones are called Coriolis/centripetal terms.

$$c_{ijk}(q) = \frac{\partial d_{kj}}{\partial q_i} - \frac{1}{2}\frac{\partial d_{ij}}{\partial q_k} = \frac{1}{2}\left(\frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} - \frac{\partial d_{ij}}{\partial q_k}\right) \tag{20}$$

The matrix form of the Euler-Lagrange equations for an n-link robot manipulator is shown in equation 21, where the matrix $D(q)$ represents the inertia matrix, $C(q,\dot{q})$ depends on the Christoffel symbols c_{ijk} and joints' velocities $\dot{q}$ (the k,j-th element of the matrix C is defined in equation 22), $\phi(q)$ is the gravity influence vector $(\partial P/\partial q_k)$, and τ is the generalized force vector.

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + \phi(q) = \tau \tag{21}$$

$$c_{kj} = \sum_{i=1}^{n} c_{ijk}(q)\dot{q}_i \tag{22}$$

A special case is when the inertia matrix $(D(q))$ is diagonal and independent of the joint variables (q). In this case, the Christoffel symbols (c_{ijk}) are zero because each element of the inertia matrix $(D(q))$ is constant.

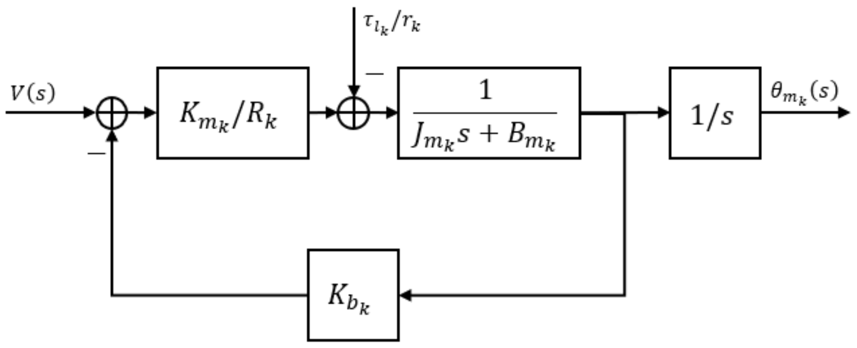


Fig. 4. Approximated Model of a Revolute Joint

4 Multivariable Control

Given the dynamics formulated in the previous section, we can now formulate a multivariable control scheme for an n-link robot manipulator. This section formulates the inverse dynamics of a manipulator considering the influence of gravity, coupling effects among joints, the inertia of the robot manipulator, and the joints' motors, among other effects, in the context of multivariable control.

4.1 Model of a revolute joint

First, let us recall the model of a revolute joint k illustrated in figure 4 considering the electric time constant much smaller than the mechanical one. The parameters of the model (voltage V applied to the motor and external torque τ_l as inputs, and the output is the joint's angular position θ_{m_k}) are the following ones:

- r_k: gear reduction ratio ($[r_k : 1]$)
- K_{m_k}: torque constant (N.m.A^{-1})
- K_{b_k}: speed constant (V.s.rad^{-1}); note that $K_{m_k,S.I.} = K_{b_k,S.I.}$
- J_{m_k}: inertia of the motor (Kg.m^2)
- B_{m_k}: motor viscous constant (N.m.s)
- R_k: internal resistance of the motor (ohm)

Considering that $B_k = B_{m_k} + K_{b_k}K_{m_k}/R_k$, the model of a revolute joint illustrated in figure 4 is equivalent to the differential equation formulated in equation 23. Note that $\dot{\theta}_{m_k} = r_k\dot{q}_k$.

$$\begin{aligned} J_{m_k}\ddot{\theta}_{m_k} + B_k\dot{\theta}_{m_k} &= \frac{K_{m_k}V_k}{R_k} - \frac{\tau_k}{r_k} \Leftrightarrow \\ r_k^2 J_{m_k}\ddot{q}_k + r_k^2 B_k\dot{q}_k &= r_k\frac{K_{m_k}V_k}{R_k} - \tau_{l_k} \end{aligned} \tag{23}$$

If it was a prismatic joint (driven by an electric motor), the only thing that would change in the joint's model is the relation between the motor's angular speed ($\dot{\theta}_{m_k}$) and the joint's velocity ($\dot{q}_k$): $\dot{\theta}_{m_k} = r_k\dot{\theta}_{l_k} = (r_k/\mathcal{R}_k)\cdot\dot{q}_k$, where $\mathcal{R}_k$ would be the radius of the circular gear (*pinion*) that transforms circular into linear motion on a linear gear (*rack*).

4.2 Equations of motion

Next, the motion equations of an n-link robot manipulator are retrieved from equations 19 and 23. If we equal the external torque τ_{l_k} to the generalized force τ_k, it is possible to formulate the manipulator's motion equations considering its dynamics and the joint's model, as illustrated in equation 24. In the matrix form, $M(q) = D(q) + J$ (where J is a diagonal matrix with elements $r_k^2 J_{m_k}$) and B is a diagonal matrix with elements $B_k = B_{m_k} + K_{b_k}K_{m_k}/R_k$.

$$\begin{gathered} \sum_j d_{kj}\ddot{q}_j + r_k^2 J_{m_k}\ddot{q}_k + \sum_{i,j} c_{ijk}\dot{q}_i\dot{q}_j + r_k^2 B_k\dot{q}_k + \frac{\partial P}{\partial q_k} = u_k \Leftrightarrow \\ M(q)\ddot{q} + C(q,\dot{q})\dot{q} + B\dot{q} + \phi(q) = u \end{gathered} \tag{24}$$

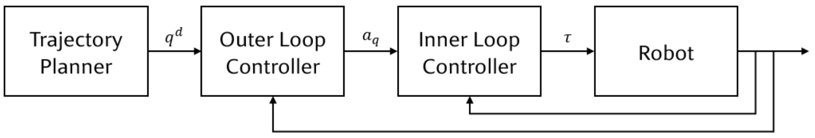


Fig. 5. Control Architecture of Inverse Dynamics

4.3 Inverse dynamics

The main goal of inverse dynamics is to formulate the nonlinear feedback control law defined in equation 25. When considering the motion equations defined in equation 24, the result is a linear closed-loop system that implements the inverse dynamics control scheme.

$$u = f\left(q, \dot{q}, t\right) \tag{25}$$

If we choose the control u (torque control) accordingly to equation 24 with $a_q = \ddot{q}$, given that the matrix M is invertible (symmetric positive definite matrix), the combined system is the equation $\ddot{q} = a_q$ itself. The term a_q represents a new input to the system that is yet to be chosen. Assuming that a_q is a function-only of q_k and its derivates, it only affect the joints' positions q_k independently of the motion of the other links.

So, two control loops are defined for the dynamic control scheme: outer and inner control loops. These two control loops are explained in the following sections.

4.3.1 Outer loop

Given a desired trajectory (defined as $t \to \left(q^d\left(t\right), \dot{q}^d\left(t\right)\right)$), a_q (output of the outer control loop) can be controlled as defined in equation 26. K_0 and K_1 are diagonal matrices with diagonal elements consisting of position and velocity gains, respectively. The reference signal r is defined by equation 27.

$$a_q = -K_0 q - K_1 \dot{q} + r \tag{26}$$

$$r = \ddot{q}^d + K_0 q^d + K_1 \dot{q}^d \tag{27}$$

Equation 28 shows that K_0 and K_1 gains influence the position $(q^d - q)$ and velocity $(\dot{q}^d - \dot{q})$ errors relative to the desired trajectory. A possible choice for these gains is $K_0 = \text{diag}\{\omega_1^2, ..., \omega_n^2\}$ and $K_1 = \text{diag}\{2\omega_1, ..., 2\omega_n\}$, which results in a closed loop sytem with each joint response equal to a critically damped linear second order system with natural frequency ω_i.

$$a_q = \ddot{q}^d + K_0\left(q^d - q\right) + K_1\left(\dot{q}^d - \dot{q}\right) \tag{28}$$

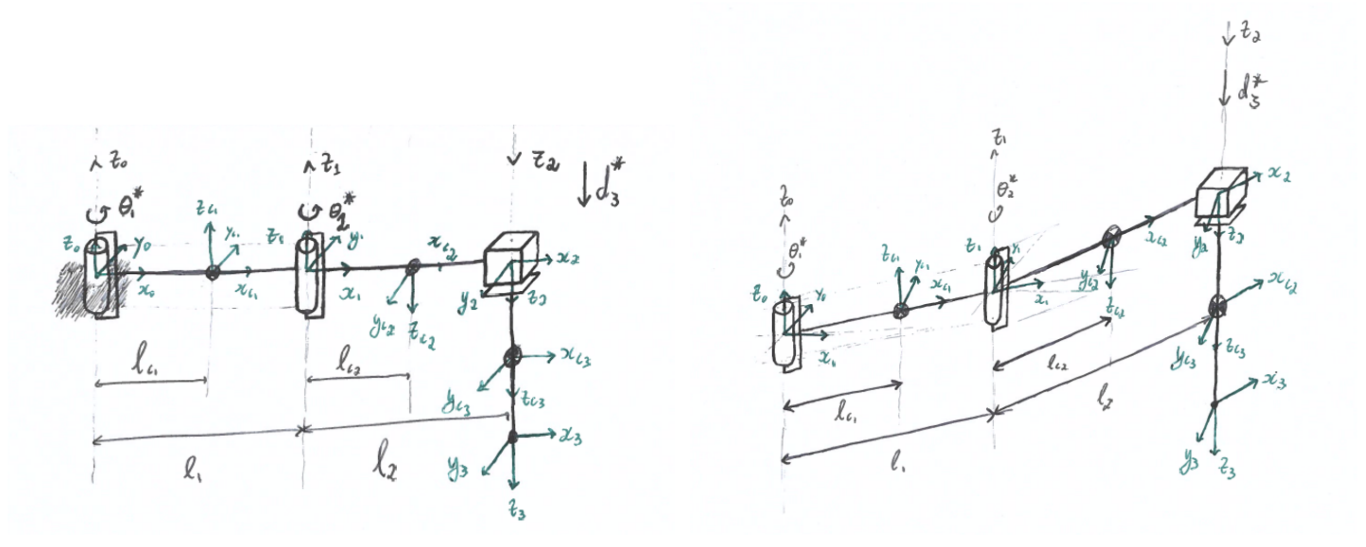
4.3.2 Inner loop

Then, the inner control loop takes as input the acceleration a_q to actuate in torque (u) the manipulator's joint, as illustrated in equation 29. Figure 5 shows the control architecture of inverse dynamics with the two outer and inner control loops. If the inverse dynamics control scheme is implemented for the set-point use case, the desired velocities (q^d) and accelerations $(\ddot{q}^d)$ of each joint must be 0. This requirement is because the goal of set-point is setting a desired point in space and the manipulator remaining at that point in steady-state.

$$u = M\left(q\right) a_q + C\left(q, \dot{q}\right) \dot{q} + B\dot{q} + \phi\left(q\right) \tag{29}$$

5 SCARA Manipulator: Practical Application of Inverse Dynamics

This section presents a practical example of implementing inverse dynamics on a 3-link SCARA manipulator. After presenting the theoretical base of inverse dynamics, the main purpose of this example is to demonstrate how to compute the matrices required to implement this control scheme. Also, this same example is the one given in the laboratory work proposed to the students.



(a) (b)

Fig. 6. Coordinate Frames of a SCARA Manipulator Following the Denavit-Hartenberg (DH) Convention

Table 1. DH Parameters of a 3-Link SCARA Manipulator

Link i	θ_i	d_i	a_i	α_i
1	q_1	0m	l_1	0º
2	q_2	0m	l_2	180º
3	0º	q_3	0m	0º

5.1 Forward kinematics (DH convention)

First, it is necessary to define the forward kinematics of the SCARA manipulator. The method used to define the SCARA's forward kinematics was the DH convention due to inverse dynamics requiring the use of generalized coordinates.

The coordinate frames of the manipulator defined by the DH convention are presented in figure 6. The DH convention defines the frames $o_i x_i y_i z_i$ for $i = 1, 2, 3$. In addition to those frames, three other ones $(o_{c_i} x_{c_i} y_{c_i} z_{c_i})$ are defined relative to each link's center of mass. Note that the coordinate frames $o_{c_i} x_{c_i} y_{c_i} z_{c_i}$ are parallel with $o_i x_i y_i z_i$ for each $i = 1, 2, 3$. This alignment is more perceptible in figure 6b.

Next, it is possible to determine the DH parameters of the 3-link SCARA manipulator. These parameters are defined in table 1. The DH parameters allow us to determine the homogeneous transformation matrices that relate the coordinate frames to each other. Equation 30 formulates the transformation of a point defined on a coordinate frame j into a point relative to a coordinate frame i, where H_j^i is the transformation matrix between these two frames. Given the DH parameters of the SCARA manipulator, we can formulate the transformation matrices H_i^0 from each coordinate frame $o_i x_i y_i z_i$ into $o_0 x_0 y_0 z_0$ for $i = 1, 2, 3$, as illustrated in equations 31, 32, and 33, respectively. The matrix H_3^0 formulates the forward kinematics of the SCARA manipulator relative to the robot's end-effector.

$$X^i = H_j^i\,(q)\,X^j \tag{30}$$

$$H_1^0 = \begin{bmatrix} \mathrm{c}_{q_1} & -\mathrm{s}_{q_1} & 0 & l_1\mathrm{c}_{q_1} \\ \mathrm{s}_{q_1} & \mathrm{c}_{q_1} & 0 & l_1\mathrm{s}_{q_1} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{31}$$

$$H_2^0 = \begin{bmatrix} \mathrm{c}_{q_1+q_2} & \mathrm{s}_{q_1+q_2} & 0 & l_1\mathrm{c}_{q_1} + l_2\mathrm{c}_{q_1+q_2} \\ \mathrm{s}_{q_1+q_2} & -\mathrm{c}_{q_1+q_2} & 0 & l_1\mathrm{s}_{q_1} + l_2\mathrm{s}_{q_1+q_2} \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{32}$$

$$H_3^0 = \begin{bmatrix} c_{q_1+q_2} & s_{q_1+q_2} & 0 & l_1 c_{q_1} + l_2 c_{q_1+q_2} \\ s_{q_1+q_2} & -c_{q_1+q_2} & 0 & l_1 s_{q_1} + l_2 s_{q_1+q_2} \\ 0 & 0 & -1 & -q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{33}$$

5.2 Inverse kinematics

As for the inverse dynamics of SCARA, where c_2 is defined by equation 34, each of the joints variables q_2, q_1, q_3 are formulated by equations 35, 36, and 37, respectively. The inverse kinematics are required for computing the desired trajectory for the robot's end-effector in the joint space.

$$c_2 = \frac{x^2 + y^2 - {l_1}^2 - {l_2}^2}{2 l_1 l_2} \tag{34}$$

$$q_2 = \operatorname{atan2}\left(-\sqrt{1 - {c_2}^2}, c_2\right) \tag{35}$$

$$q_1 = \operatorname{atan2}(y, x) - \operatorname{atan2}(l_2 s_{q_2}, l_1 + l_2 c_{q_2}) \tag{36}$$

$$q_3 = -z \tag{37}$$

5.3 Velocity kinematics

The matrix J_3^0 (defined in equation 38) defines the velocity kinematics of the end-effect relative to the coordinate frame $o_0 x_0 y_0 z_0$. This matrix relates the joints velocities $\dot{q}^T = [\dot{q}_1, \dot{q}_2, \dot{q}_3]$ to the velocities $\dot{X}^T = [\dot{x}, \dot{y}, \dot{z}, \dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z]$ in the coordinate frame 0. Note that $\dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z$ are the angular velocities around the x, y, z axes of the coordinate frame 0.

$$J_3^0 = \begin{bmatrix} -l_1 s_{q_1} - l_2 s_{q_1+q_2} & -l_2 s_{q_1+q_2} & 0 \\ l_1 c_{q_1} + l_2 c_{q_1+q_2} & l_2 c_{q_1+q_2} & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \tag{38}$$

Furthermore, we can define the velocity kinematics relative to any point of the robot manipulator's links. Given that the inverse dynamics require the known velocities of each link's center of mass, let us defined the forward kinematics of each center of mass o_{c_i} relative to the coordinate frame 0. These kinematics are represented by the matrices $H_{c_1}^0, H_{c_2}^0, H_{c_3}^0$ that are defined by equations 39, 40, and 41, respectively. The distances l_{c_i} are the distance from the joint i to the link i's center of mass, as illustrated in figure 6.

$$H_{c_1}^0 = \begin{bmatrix} c_{q_1} & -s_{q_1} & 0 & l_{c_1} c_{q_1} \\ s_{q_1} & c_{q_1} & 0 & l_{c_1} s_{q_1} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{39}$$

$$H_{c_2}^0 = \begin{bmatrix} c_{q_1+q_2} & s_{q_1+q_2} & 0 & l_1 c_{q_1} + l_{c_2} c_{q_1+q_2} \\ s_{q_1+q_2} & -c_{q_1+q_2} & 0 & l_1 s_{q_1} + l_{c_2} s_{q_1+q_2} \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{40}$$

$$H_{c_3}^0 = \begin{bmatrix} c_{q_1+q_2} & s_{q_1+q_2} & 0 & l_1 c_{q_1} + l_2 c_{q_1+q_2} \\ s_{q_1+q_2} & -c_{q_1+q_2} & 0 & l_1 s_{q_1} + l_2 s_{q_1+q_2} \\ 0 & 0 & -1 & -q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{41}$$

Then, the velocity kinematics are defined based on the center of masses' forward kinematics. The matrices $J_{v_{c_i}}$ and $J_{\omega_{c_i}}$ for $i = 1, 2, 3$ represent the linear velocity and angular velocity components of the velocity kinematics, respectively. These matrices are defined in equations 42, 43, and 44. Note that for link 1's center of mass, only the first joint contributes to link 1's velocity (2nd and 3rd columns are null of the matrices $J_{v_{c_1}}$ and $J_{\omega_{c_1}}$). For link 2, only the first and second joints contribute to link 2's velocity. These observations make sense because the SCARA manipulator studied in this section is a serial configuration.

$$J_{v_{c_1}} = \begin{bmatrix} -l_{c_1}\mathrm{s}_{q_1} & 0 & 0 \\ l_{c_1}\mathrm{c}_{q_1} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \; ; \; J_{\omega_{c_1}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{42}$$

$$J_{v_{c_2}} = \begin{bmatrix} -l_1\mathrm{s}_{q_1} - l_{c_2}\mathrm{s}_{q_1+q_2} & -l_{c_2}\mathrm{s}_{q_1+q_2} & 0 \\ l_1\mathrm{c}_{q_1} + l_{c_2}\mathrm{c}_{q_1+q_2} & l_{c_2}\mathrm{c}_{q_1+q_2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \; ; \; J_{\omega_{c_2}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \tag{43}$$

$$J_{v_{c_3}} = \begin{bmatrix} -l_1\mathrm{s}_{q_1} - l_2\mathrm{s}_{q_1+q_2} & -l_2\mathrm{s}_{q_1+q_2} & 0 \\ l_1\mathrm{c}_{q_1} + l_2\mathrm{c}_{q_1+q_2} & l_2\mathrm{c}_{q_1+q_2} & 0 \\ 0 & 0 & -1 \end{bmatrix} \; ; \; J_{\omega_{c_3}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \tag{44}$$

5.4 Inverse dynamics

Now, we can define the inverse dynamics of the SCARA manipulators. Given the extent of the expressions for the matrices $D(q)$, J, $C(q,\dot{q})$, $B\dot{q}$, and $\phi(q)$ (see sections 3 and 4 for further details), all the matrices required for inverse dynamics were computed using symbolic math equations in MATLAB. The scripts used for this effect are presented in appendix A and were make available for the students in SIGARRA.

5.5 Next steps: Task space

Lastly, it is possible to change the outer loop controller to define the desired pose in the task space instead of in the joint space. Let us consider the end-effector pose X considering any minimal representation $SO(3)$. Then, it is possible to define the velocity $(\dot{X})$ and acceleration $(\ddot{X})$ of the end-effector in the task space, as follows:

- $X = [x, y, z, \phi, \theta, \psi]^T \in \mathbb{R}^6$
- $\dot{X} = J_a(q)\,\dot{q}$
- $\ddot{X} = J_a(q)\,\ddot{q} + \dot{J}_a(q)\,\dot{q}$

The J_a is the analytical jacobian than considers the relation between the joints' velocity $(\dot{q})$ and the velocity in the task space $(\dot{X})$, where $\dot{x}, \dot{y}, \dot{z}$ are the linear velocity components and $\dot{\phi}, \dot{\theta}, \dot{\psi}$ angular components around a vector of Euler angles $([\phi, \theta, \psi]^T)$.

Now, we can choose the double integrator system $\ddot{X} = a_X$ in task space coordinate as in equation 45. This modification achieves the same linear and decoupled properties of the initial outer loop designed for inverse dynamics but with the advantage of being defined directly in the task space. Indeed, this approach does not require the computation of inverse kinematics.

$$a_X = \ddot{X}^d + K_P\left(X^d - X\right) + K_D\left(\dot{X}^d - \dot{X}\right) \tag{45}$$

Then, the output of the outer loop can be computed by equation 46. If the matrix J_a is not a square one, it should be used the Monroe-Penrose pseudoinverse instead of the Jacobian's inverse.

$$a_q = \ddot{q} = J_a^{-1}(q) \cdot \left(a_X - \dot{J}_a(q)\,\dot{q}\right) \tag{46}$$

However, this section was only presented to the students with the sole goal of giving hints for the ones interested in inverse dynamics for, e.g., the Master's thesis.

6 Theoretical Class

Until now, it was analyzed the theory behind inverse dynamics. This analysis was the basis of the presentation elaborated for the theoretical class of the MIEEC@FEUP's course unit. The class was lectured in English at a Zoom conference on May 21, from 8:30 am until 10:30 am. As for the presentation, it is appended to the present report (appendix B).

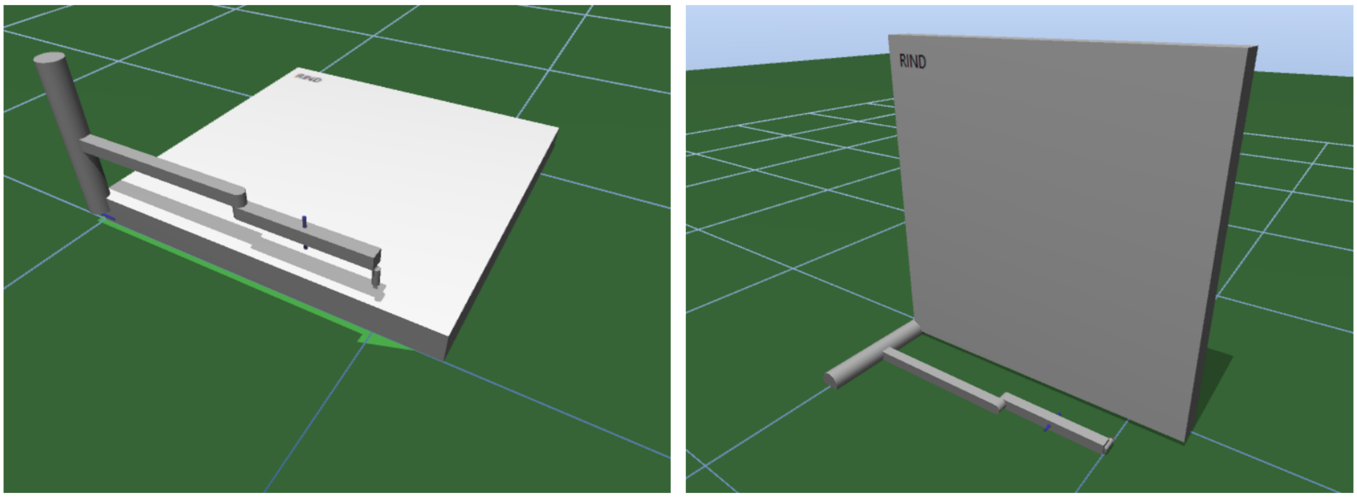


Fig. 7. SCARA Manipulator Used for the Laboratory Work: (a) Horizontal Orientation; (b) Vertical Orientation

In terms of the presentation's template, it was designed to be simple and have space to put equations, explanations, among other content or media, that could be helpful for the students to comprehend the topic. The identification of FEUP is provided by its logo. As for the choice of colors, the red color is the same chromaticism that represents FEUP with the following RGB code: #8C2D19[1]. The background color was not white (specifically, a slightly grey color with the RGB code #E6E6E6) to not "hurt" the eyes when watching the class or reading the document.

As for the presentation's structure, it is composed of three topics: dynamics, multivariable control, and the analysis of inverse dynamics for a specific practical application (SCARA manipulator). Note that these are the same sections 3, 4, and 5 presented in this report, respectively. This structure was organized in this way for, first, explaining the dynamics of a generic n-link manipulator, next, formulating the multivariable control scheme with inverse dynamics, and then, present a practical implementation of the inverse dynamics' control scheme.

In addition to the presentation, it was presented two videos[2][3]. The videos did not contain sound because their explanation was given in the class itself. The purpose of these was to show the implementation of inverse dynamics in SimTwo [24] that was also the student's goal for the laboratory work. The videos were recorded with OBS Studio using the implementation in SimTwo [24] of inverse dynamics presented in appendix D.1.

7 Laboratory Work

The main goal set for the students relative to the laboratory work "Inverse Dynamics with Multivariable Control" was implementing the two control loop characteristics of inverse dynamics (explained in section 4) on the SCARA manipulator illustrated in figure 7. This laboratory work did not focus on how to compute the matrices required for applying inverse dynamics. Indeed, the matrices' computation is already explained in the presentation (see appendix B), the MATLAB code available in SIGARRA (see appendix A), and also implemented in the base code provided to the students for the laboratory work (see appendix D.2).

Appendix C presents the guide provided to the students for the laboratory work "Inverse Dynamics with Multivariable Control". This guide consisted of four steps: analyze the code, implement inverse dynamics for horizontal and vertical orientations of the robot, and evaluate the performance of inverse dynamics compensating only the gravity in both horizontal and vertical orientations. Even though the presentation for the theoretical class already explains what it is necessary to do in the laboratory work, the guide presents the essential equations and SimTwo [24] functions required for the work.

Finally, the practical classes lecture were on May 24, 25, 26, and 27 of the classes 4MIEEC_A4_RS, 4MIEEC_A5_RS, 4MIEEC_RIND1, and 4MIEEC_RIND2, respectively. The class began by introducing the laboratory work and the code provided to the students and then assisting them in implementing the laboratory work.

[1] `https://sigarra.up.pt/feup/pt/web_base.gera_pagina?p_pagina=*normas%20-%20log%c3%b3tipo/s%c3%admbolo`

[2] `https://inesctecpt-my.sharepoint.com/:v:/g/personal/ricardo_b_sousa_office365_inesctec_pt/EeCXIxyGtt1HvtHdSki4WQwBdhgTcIOyKqDVnB-4vMW2Ng?e=bT22DG`

[3] `https://inesctecpt-my.sharepoint.com/:v:/g/personal/ricardo_b_sousa_office365_inesctec_pt/ERqZAACEIhBCqK8pGuEXysoBThdQk6awdYfPHxx_LizAtA?e=Vc5Ll2`

8 Conclusions

The course unit Special Topics from the ECE Ph.D. program at FEUP consists of two types of alternative activities: a planned individual study – study a scientific topic that could be relevant for the student's thesis – or teaching assistance – in courses offered in the department of ECE of FEUP. I have chosen the activity of teaching assistance in the curricular unit Industrial Robotics (RIND) of the Master in ECE at FEUP. The work plan was to study a new topic to be inserted in the course. This topic was inverse dynamics with multivariable control for robot manipulators.

Unlike independent joint control, multivariable control considers effects such as gravity, centripetal and/or centrifugal forces, the inertia of the manipulator itself, and coupling effects between joints. Existent works applied multivariable control with approaches such as fuzzy control and neural networks. However, the basis of most approaches is the inverse dynamics model of a robot manipulator.

This work focused on studying and teaching the topic of inverse dynamics to the students enrolled in RIND. First, the dynamics of an n-link robot manipulator were formulated. Then, the inverse dynamics with a multivariable control scheme were defined in the joints space. A practical example of applying inverse dynamics on a 3-link SCARA manipulator was explained to the students to consolidate the topic. In the classes, it was shown the clear difference between independent control and inverse dynamics: the latter achieves an improved tracking performance when the robot was subjected to different effects such as the gravity itself. The SimTwo [24] environment used in the practical classes is available in GitHub[4].

Overall, the teaching experience was very interesting and rich for me. Even though I had some experience in teaching (music classes and monitoring Project FEUP), being a teaching assistant in Industrial Robotics was the one that taught me the more, given my interest in teaching high education. I had to study a new topic, implementing it, and teach it to the students. The difficulties felt throughout the study and implementation led me to conclude that I needed more experience in the topic both in simulation and real environments. I concluded that a professor must be a true specialist with several field experiences to be capable of giving more knowledge to the students. Even so, it was a great experience and I hope in the future to come back to teaching high education in my research areas.

Acknowledgments

I would like to thank professors António Paulo Moreira and Paulo G. Costa for all the help given to me throughout the teaching assistance in Industrial Robotics. Their help and collaboration were crucial for the successful implementation of inverse dynamics and the topic's introduction into the curricular unit.

References

1. A.P. Moreira. Industrial Robotics (2021). URL `https://sigarra.up.pt/feup/en/UCURR_GERAL.FICHA_UC_VIEW?pv_ocorrencia_id=461550`. Accessed on June 02, 2021
2. M. Spong, S. Hutchinson, M. Vidyasagar, *Robot Modeling and Control*, 1st edn. (John Wiley & Sons, 2005)
3. L.W. Tsai, *Solving the inverse dynamics of a Stewart-Gough manipulator by the principle of virtual work*, Journal of Mechanical Design **122**(1) 3–9 (1999). DOI `10.1115/1.533540`
4. S.H. Lee, J.B. Song, W.C. Choi, D. Hong, *Position control of a Stewart platform using inverse dynamics control with approximate dynamics*, Mechatronics **13**(6) 605–619 (2003). DOI `10.1016/S0957-4158(02)00033-8`
5. S. Staicu, X.J. Liu, J. Wang, *Inverse dynamics of the HALF parallel manipulator with revolute actuators*, Nonlinear Dynamics **50** 1–12 (2007). DOI `10.1007/s11071-006-9138-5`
6. Y. Zhao, F. Gao, *Inverse dynamics of the 6-dof out-parallel manipulator by means of the principle of virtual work*, Robotica **27**(2) 259–268 (2009). DOI `10.1017/S0263574708004657`
7. S. Saab, P. Ghanem, *A multivariable stochastic tracking controller for robot manipulators without joint velocities*, IEEE Transactions on Automatic Control **63**(8) 2481–2495 (2018). DOI `10.1109/TAC.2017.2771154`
8. S. Saab, R. Jaafar, *A proportional-derivative-double derivative controller for robot manipulators*, International Journal of Control **94**(5) 1273–1285 (2021). DOI `10.1080/00207179.2019.1642518`
9. F. Valle, F. Tadeo, T. Alvarez, *Predictive control of robotic manipulators*, in *Proceedings of the International Conference on Control Applications*, vol. 5 (2002), pp. 203–208. DOI `10.1109/CCA.2002.1040186`
10. C.D. Zhang, S.M. Song, *An efficient method for inverse dynamics of manipulators based on the virtual work principle*, Journal of Robotic Systems **10**(5) 605–627 (1993). DOI `10.1002/rob.4620100505`
11. B. Dasgupta, T. Mruthyunjaya, *A Newton-Euler formulation for the inverse dynamics of the Stewart platform manipulator*, Mechanism and Machine Theory **33**(8) 1135–1152 (1998). DOI `10.1016/S0094-114X(97)00118-3`
12. Y.W. Li, J.S. Wang, L.P. Wang, X.J. Liu, *Inverse dynamics and simulation of a 3-DOF spatial parallel manipulator*, in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 3 (2003), pp. 4092–4097. DOI `10.1109/ROBOT.2003.1242226`

[4] `https://github.com/sousarbarb/pdeec-st`

13. V. Santibanez, R. Kelly, M. Llama, *Fuzzy PD+ control for robot manipulators*, in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 3 (2000), pp. 2112–2117. DOI 10.1109/ROBOT.2000.846341
14. T. Shaocheng, T. Jiantao, W. Tao, *Fuzzy adaptive control of multivariable nonlinear systems*, Fuzzy Sets and Systems **111**(2) 153–167 (2000). DOI https://doi.org/10.1016/S0165-0114(98)00052-9
15. M. Mahmoodabadi, A. Ziaei, *Inverse dynamics based optimal fuzzy controller for a robot manipulator via particle swarm optimization*, Journal of Robotics **2019** (2019). DOI 10.1155/2019/5052185
16. E. Bobaşu, D. Popescu, *On modeling and multivariable adaptive control of robotic manipulators*, WSEAS Transactions on Systems **5**(7) (2006). URL https://www.researchgate.net/publication/266999327_On_modeling_and_multivariable_adaptive_control_of_robotic_manipulators
17. Y. Lei, H. Wu, *Tracking control of robotic manipulators based on the all-coefficient adaptive control method*, International Journal of Control, Automation and Systems **4**(2) (2006). URL https://www.researchgate.net/publication/254382510_Tracking_Control_of_Robotic_Manipulators_based_on_the_All-Coefficient_Adaptive_Control_Method
18. K.C. Chiou, S.J. Huang, *An adaptive fuzzy controller for robot manipulators*, Mechatronics **15**(2) 151–177 (2005). DOI 10.1016/j.mechatronics.2004.07.005
19. Y. Chen, G. Mei, G. Ma, S. Lin, J. Gao, *Robust adaptive inverse dynamics control for uncertain robot manipulator*, International Journal of Innovative Computing, Information and Control **10**(2) 575–587 (2014)
20. L. Cuvillon, E. Laroche, J. Gangloff, M. Mathelin, *A mutivariable methodology for fast visual servoing of flexible manipulators moving in a restricted workspace*, Advanced Robotics **26**(15) 1771–1797 (2012). DOI 10.1080/01691864.2012.685230
21. M. Moradi, H. Malekizade, *Neural network identification based multivariable feedback linearization robust control for a two-link manipulator*, Journal of Intelligent & Robotic Systems **72** 167–178 (2013). DOI 10.1007/s10846-013-9827-5
22. L. Yu, S. Fei, L. Sun, J. Huang, *An adaptive neural network switching control approach of robotic manipulators for trajectory tracking*, International Journal of Computer Mathematics **91**(5) 983–995 (2014). DOI 10.1080/00207160.2013.813021
23. A.S. Polydoros, L. Nalpantidis, V. Krüger, *Real-time deep learning of robotic manipulator inverse dynamics*, in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015), pp. 3442–3448. DOI 10.1109/IROS.2015.7353857
24. P. Costa, J. Gonçalves, J. Lima, P. Malheiros, *SimTwo realistic simulator: a tool for the development and validation of robot software*, International Journal of Theory and Applications of Mathematics & Computer Science 11–16 (2011). URL http://hdl.handle.net/10198/4117

A MATLAB Scripts

A.1 DH convention

Listing 1. Homogeneous Matrix From DH Parameters

```
function A = DH(theta,d,a,alpha)
%DH

  A = [
    cos(theta) , -sin(theta)*cos(alpha) ,  sin(theta)*sin(alpha) , a*cos(theta) ;
    sin(theta) ,  cos(theta)*cos(alpha) , -cos(theta)*sin(alpha) , a*sin(theta) ;
             0 ,             sin(alpha) ,             cos(alpha) ,            d ;
             0 ,                      0 ,                      0 ,            1
  ];
end
```

A.2 Rotation matrices

Listing 2. Orientation Matrix Relative to a Rotation Around a X Axis

```
function R = Rx(thx)
%RX

  R = [
    1 ,        0 ,         0 ;
    0 , cos(thx) , -sin(thx) ;
    0 , sin(thx) ,  cos(thx)
  ];
end
```

Listing 3. Orientation Matrix Relative to a Rotation Around a Y Axis

```
function R = Ry(thy)
%RY

  R = [
     cos(thy) , 0 , sin(thy) ;
            0 , 1 ,        0 ;
    -sin(thy) , 0 , cos(thy)
  ];
end
```

Listing 4. Orientation Matrix Relative to a Rotation Around a Z Axis

```
function R = Rz(thz)
%RX

  R = [
    cos(thz) , -sin(thz) , 0 ;
    sin(thz) ,  cos(thz) , 0 ;
           0 ,         0 , 1
  ];
end
```

A.3 Transpose matrices

Listing 5. Transpose Matrices Considering Real Numbers-Only

```
function [Aout] = Transpose(Ain)
%TRANSPOSE

  [m,n] = size(Ain);
  %Aout = Ain;
  Aout = vpa(zeros(n,m));
  for i=1:m
    for j=1:n
      Aout(j,i) = Ain(i,j);
    end
  end
end
```

A.4 Christoffel symbols

Listing 6. Christoffel Symbols From The Inertia Matrix D and Joint Variables Q

```
function [cijk] = ChristoffelSymbols(D,q,i,j,k)
%CHRISTOFFELSYMBOLS

  cijk = ( diff(D(k,j),q(i)) + diff(D(k,i),q(j)) - diff(D(i,j),q(k)) ) / 2;
end
```

A.5 Inverse dynamics of a SCARA manipulator

Listing 7. Forward Kinematics of a SCARA Manipulator

```
close all
clear all
clc

%% INITIALIZATION
% Joints
syms q1 q2 q3
% Lengths
syms l1 l2

%% FORWARD KINEMATICS
A1 = DH(q1, 0,l1, 0)
A2 = DH(q2, 0,l2,pi)
A3 = DH( 0,q3, 0, 0)

H = A1*A2*A3
```

Listing 8. Intertia Matrix of a SCARA Manipulator

```
close all
clear all
clc

%% INITIALISATION
```

```
syms q1 q2 q3
syms m1 m2 m3
syms lc1 lc2
syms l1 l2
syms rho1 rho2 rho3 l1x l1y l1z l2x l2y l2z l3x l3y l3z

%% "PARTIAL" JACOBIANS
Jvc1 = [
  -lc1*sin(q1)  , 0 , 0 ;
   lc1*cos(q1)  , 0 , 0 ;
   0            , 0 , 0
];
Jwc1 = [
  0 , 0 , 0 ;
  0 , 0 , 0 ;
  1 , 0 , 0
];
Jvc2 = [
  -l1*sin(q1) - lc2*sin(q1+q2)  , -lc2*sin(q1+q2)  , 0 ;
   l1*cos(q1) + lc2*cos(q1+q2)  ,  lc2*cos(q1+q2)  , 0 ;
   0                            ,  0               , 0
];
Jwc2 = [
  0 , 0 , 0 ;
  0 , 0 , 0 ;
  1 , 1 , 0
];
Jvc3 = [
  -l1*sin(q1) - l2*sin(q1+q2)  , -l2*sin(q1+q2)  ,  0 ;
   l1*cos(q1) + l2*cos(q1+q2)  ,  l2*cos(q1+q2)  ,  0 ;
   0                           ,  0              , -1
];
Jwc3 = Jwc2;

%% INERTIA TENSORS
Ic1 = diag([
  rho1*l1x*l1y*l1z*(l1y*l1y+l1z*l1z)/12  ,
  rho1*l1x*l1y*l1z*(l1x*l1x+l1z*l1z)/12  ,
  rho1*l1x*l1y*l1z*(l1x*l1x+l1y*l1y)/12
]);
Ic2 = diag([
  rho2*l2x*l2y*l2z*(l2y*l2y+l2z*l2z)/12  ,
  rho2*l2x*l2y*l2z*(l2x*l2x+l2z*l2z)/12  ,
  rho2*l2x*l2y*l2z*(l2x*l2x+l2y*l2y)/12
]);
Ic3 = diag([
  rho3*l3x*l3y*l3z*(l3y*l3y+l3z*l3z)/12  ,
  rho3*l3x*l3y*l3z*(l3x*l3x+l3z*l3z)/12  ,
  rho3*l3x*l3y*l3z*(l3x*l3x+l3y*l3y)/12
]);

%% ROTATION INERTIA MATRICES
Rc1 = [
  cos(q1)  , -sin(q1)  , 0 ;
  sin(q1)  ,  cos(q1)  , 0 ;
  0        ,  0        , 1
];
Rc2 = [
  cos(q1+q2)  ,   sin(q1+q2)  ,   0 ;
  sin(q1+q2)  , -cos(q1+q2)  ,   0 ;
  0           ,  0           ,  -1
];
Rc3 = Rc2;

%% INERTIA MATRIX
D = ...
  m1*Transpose(Jvc1)*Jvc1 + Transpose(Jwc1)*Rc1*Ic1*Transpose(Rc1)*Jwc1 + ...
  m2*Transpose(Jvc2)*Jvc2 + Transpose(Jwc2)*Rc2*Ic2*Transpose(Rc2)*Jwc2 + ...
  m3*Transpose(Jvc3)*Jvc3 + Transpose(Jwc3)*Rc3*Ic3*Transpose(Rc3)*Jwc3
```

Listing 9. Christoffel Symbols of a SCARA Manipulator

```
%% INITIALISATION
scara_inertia_matrix
q = [q1 q2 q3];

%% CHRISTOFFEL SYMBOLS
% k=1
c111 = ChristoffelSymbols(D,q,1,1,1)
c121 = ChristoffelSymbols(D,q,1,2,1)
c131 = ChristoffelSymbols(D,q,1,3,1)
c211 = ChristoffelSymbols(D,q,2,1,1)
c221 = ChristoffelSymbols(D,q,2,2,1)
c231 = ChristoffelSymbols(D,q,2,3,1)
c311 = ChristoffelSymbols(D,q,3,1,1)
c321 = ChristoffelSymbols(D,q,3,2,1)
c331 = ChristoffelSymbols(D,q,3,3,1)

% k=2
c112 = ChristoffelSymbols(D,q,1,1,2)
c122 = ChristoffelSymbols(D,q,1,2,2)
c132 = ChristoffelSymbols(D,q,1,3,2)
c212 = ChristoffelSymbols(D,q,2,1,2)
c222 = ChristoffelSymbols(D,q,2,2,2)
c232 = ChristoffelSymbols(D,q,2,3,2)
c312 = ChristoffelSymbols(D,q,3,1,2)
c322 = ChristoffelSymbols(D,q,3,2,2)
c332 = ChristoffelSymbols(D,q,3,3,2)

% k=3
c113 = ChristoffelSymbols(D,q,1,1,3)
c123 = ChristoffelSymbols(D,q,1,2,3)
c133 = ChristoffelSymbols(D,q,1,3,3)
c213 = ChristoffelSymbols(D,q,2,1,3)
c223 = ChristoffelSymbols(D,q,2,2,3)
c233 = ChristoffelSymbols(D,q,2,3,3)
c313 = ChristoffelSymbols(D,q,3,1,3)
c323 = ChristoffelSymbols(D,q,3,2,3)
c333 = ChristoffelSymbols(D,q,3,3,3)
```

Listing 10. Vector Phi (Gravity Influence) of a SCARA Manipulator with Vertical Orientation

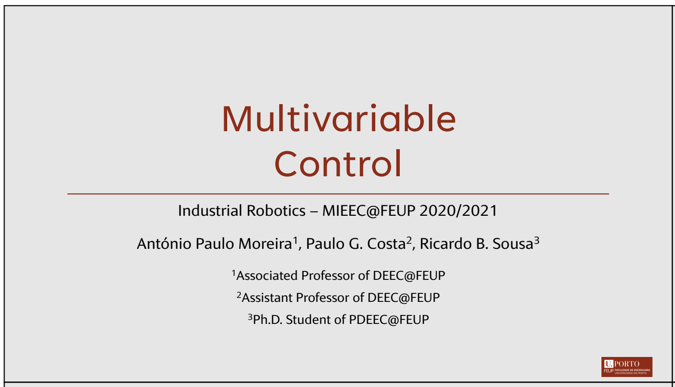
```
close all
clear all
clc

%% INITIALISATION
syms q1 q2 q3
syms m1 m2 m3
syms lc1 lc2
syms l1 l2
syms rho1 rho2 rho3 l1x l1y l1z l2x l2y l2z l3x l3y l3z
syms g

%% POTENTIAL ENERGY
P1 = -g*lc1*sin(q1)*m1;
P2 = -g*l1 *sin(q1)*m2 - g*lc2*sin(q1+q2)*m2;
P3 = -g*l1 *sin(q1)*m3 - g*l2 *sin(q1+q2)*m3;
P  = P1 + P2 + P3;

%% PHI = d P / d qi
Phi1 = diff(P,q1)
Phi2 = diff(P,q2)
Phi3 = diff(P,q3)
```

B Presentation for the Theoretical Class

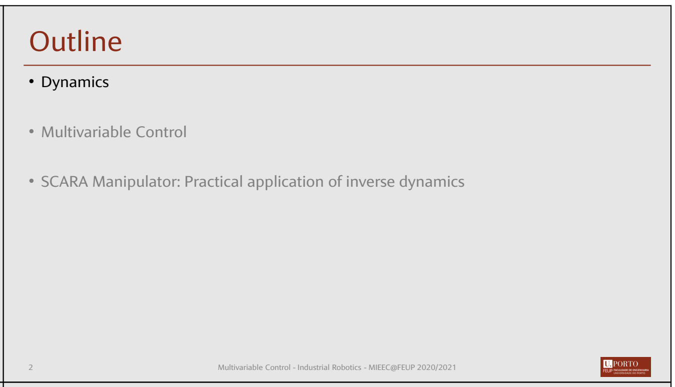


Multivariable Control

Industrial Robotics – MIEEC@FEUP 2020/2021

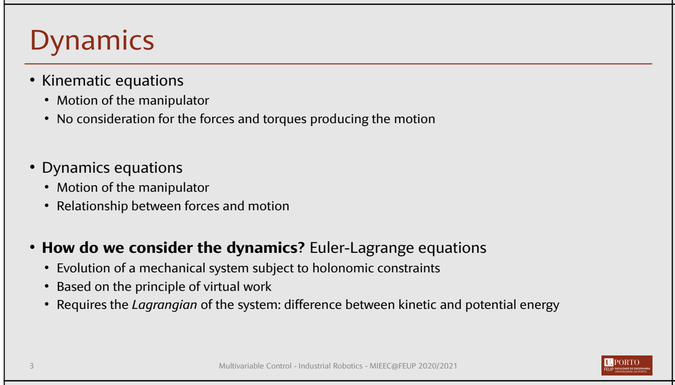
António Paulo Moreira[1], Paulo G. Costa[2], Ricardo B. Sousa[3]

[1]Associated Professor of DEEC@FEUP
[2]Assistant Professor of DEEC@FEUP
[3]Ph.D. Student of PDEEC@FEUP



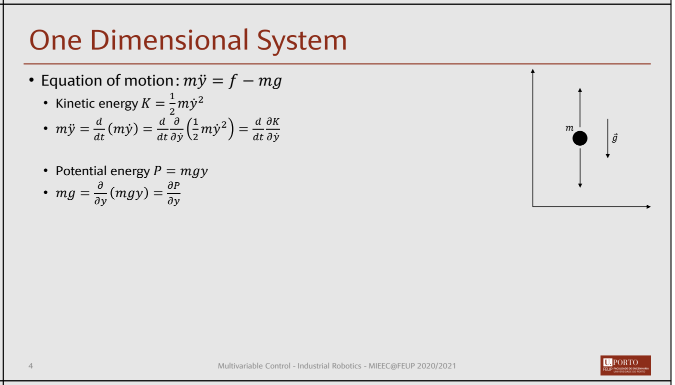
Outline

- Dynamics
- Multivariable Control
- SCARA Manipulator: Practical application of inverse dynamics



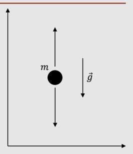
Dynamics

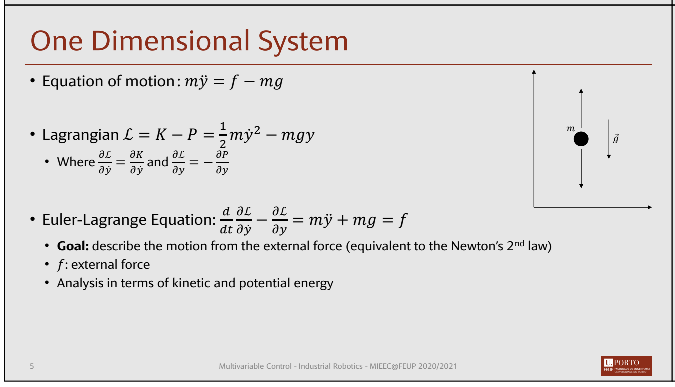
- Kinematic equations
  - Motion of the manipulator
  - No consideration for the forces and torques producing the motion
- Dynamics equations
  - Motion of the manipulator
  - Relationship between forces and motion
- **How do we consider the dynamics?** Euler-Lagrange equations
  - Evolution of a mechanical system subject to holonomic constraints
  - Based on the principle of virtual work
  - Requires the *Lagrangian* of the system: difference between kinetic and potential energy



One Dimensional System

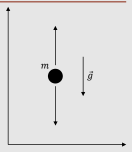
- Equation of motion: $m\ddot{y} = f - mg$
  - Kinetic energy $K = \frac{1}{2}m\dot{y}^2$
  - $m\ddot{y} = \frac{d}{dt}(m\dot{y}) = \frac{d}{dt}\frac{\partial}{\partial \dot{y}}\left(\frac{1}{2}m\dot{y}^2\right) = \frac{d}{dt}\frac{\partial K}{\partial \dot{y}}$
  - Potential energy $P = mgy$
  - $mg = \frac{\partial}{\partial y}(mgy) = \frac{\partial P}{\partial y}$

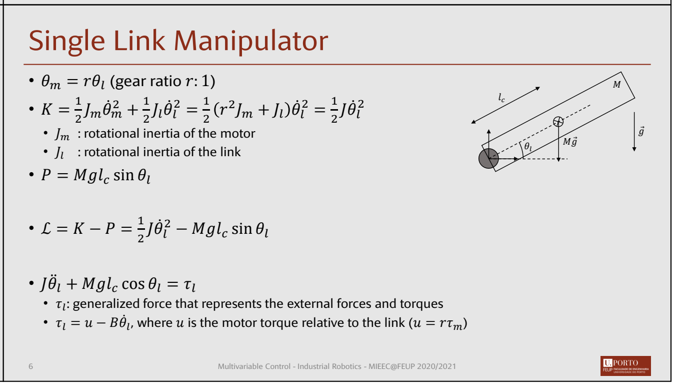




One Dimensional System

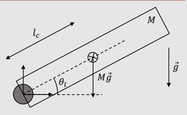
- Equation of motion: $m\ddot{y} = f - mg$
- Lagrangian $\mathcal{L} = K - P = \frac{1}{2}m\dot{y}^2 - mgy$
  - Where $\frac{\partial \mathcal{L}}{\partial \dot{y}} = \frac{\partial K}{\partial \dot{y}}$ and $\frac{\partial \mathcal{L}}{\partial y} = -\frac{\partial P}{\partial y}$
- Euler-Lagrange Equation: $\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{y}} - \frac{\partial \mathcal{L}}{\partial y} = m\ddot{y} + mg = f$
  - **Goal:** describe the motion from the external force (equivalent to the Newton’s 2nd law)
  - f: external force
  - Analysis in terms of kinetic and potential energy

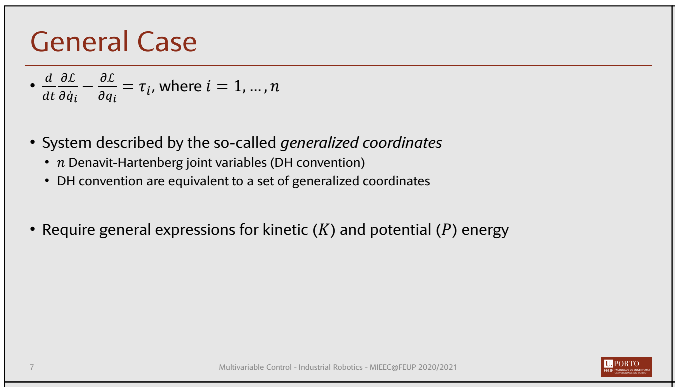




Single Link Manipulator

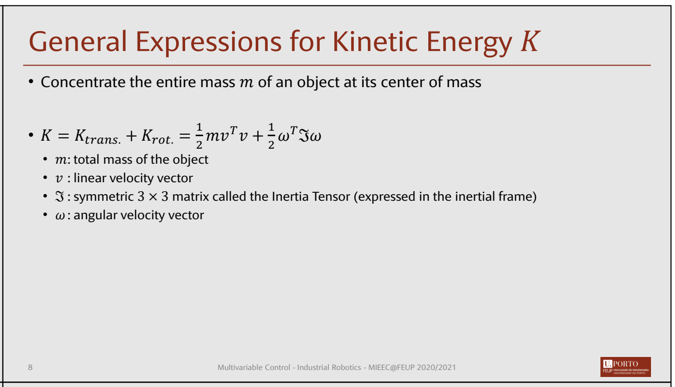
- $\theta_m = r\theta_l$ (gear ratio r: 1)
- $K = \frac{1}{2}J_m\dot{\theta}_m^2 + \frac{1}{2}J_l\dot{\theta}_l^2 = \frac{1}{2}(r^2J_m + J_l)\dot{\theta}_l^2 = \frac{1}{2}J\dot{\theta}_l^2$
  - J_m : rotational inertia of the motor
  - J_l : rotational inertia of the link
- $P = Mgl_c \sin\theta_l$
- $\mathcal{L} = K - P = \frac{1}{2}J\dot{\theta}_l^2 - Mgl_c \sin\theta_l$
- $J\ddot{\theta}_l + Mgl_c \cos\theta_l = \tau_l$
  - τ_l: generalized force that represents the external forces and torques
  - $\tau_l = u - B\dot{\theta}_l$, where u is the motor torque relative to the link ($u = r\tau_m$)





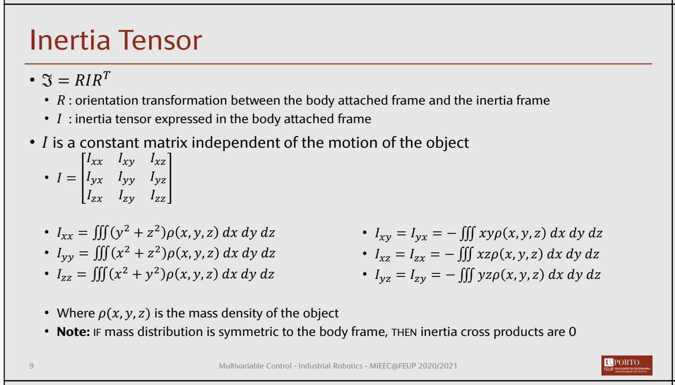
General Case

- $\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i$, where $i = 1, \dots, n$
- System described by the so-called *generalized coordinates*
  - n Denavit-Hartenberg joint variables (DH convention)
  - DH convention are equivalent to a set of generalized coordinates
- Require general expressions for kinetic (K) and potential (P) energy



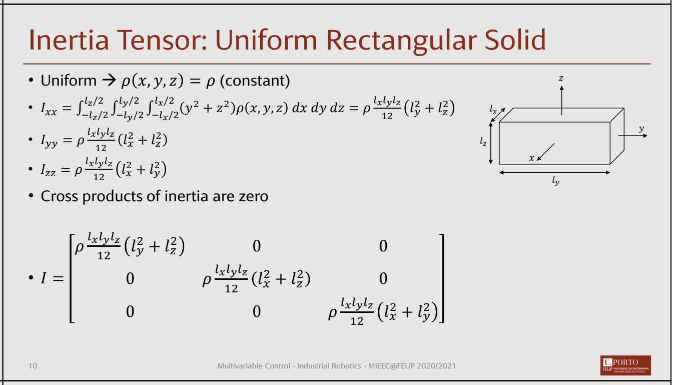
General Expressions for Kinetic Energy K

- Concentrate the entire mass m of an object at its center of mass
- $K = K_{trans.} + K_{rot.} = \frac{1}{2}mv^Tv + \frac{1}{2}\omega^T\mathfrak{I}\omega$
  - m: total mass of the object
  - v : linear velocity vector
  - $\mathfrak{I}$: symmetric 3×3 matrix called the Inertia Tensor (expressed in the inertial frame)
  - ω : angular velocity vector



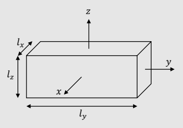
Inertia Tensor

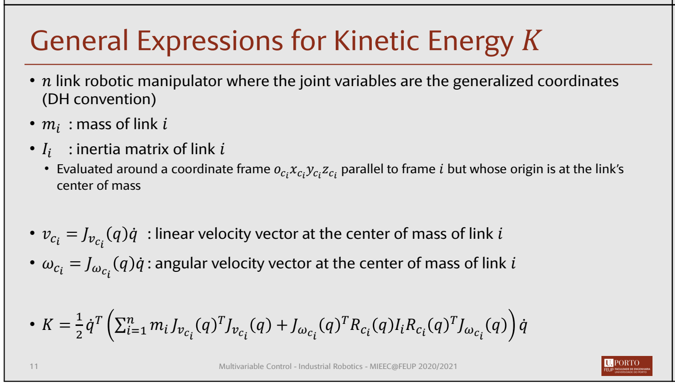
- $\mathfrak{I} = RIR^T$
  - R : orientation transformation between the body attached frame and the inertia frame
  - I : inertia tensor expressed in the body attached frame
- I is a constant matrix independent of the motion of the object
  - $I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$
  - $I_{xx} = \iiint (y^2+z^2)\rho(x,y,z)\,dx\,dy\,dz$
  - $I_{yy} = \iiint (x^2+z^2)\rho(x,y,z)\,dx\,dy\,dz$
  - $I_{zz} = \iiint (x^2+y^2)\rho(x,y,z)\,dx\,dy\,dz$
  - $I_{xy} = I_{yx} = -\iiint xy\rho(x,y,z)\,dx\,dy\,dz$
  - $I_{xz} = I_{zx} = -\iiint xz\rho(x,y,z)\,dx\,dy\,dz$
  - $I_{yz} = I_{zy} = -\iiint yz\rho(x,y,z)\,dx\,dy\,dz$
  - Where $\rho(x,y,z)$ is the mass density of the object
  - **Note:** IF mass distribution is symmetric to the body frame, THEN inertia cross products are 0



Inertia Tensor: Uniform Rectangular Solid

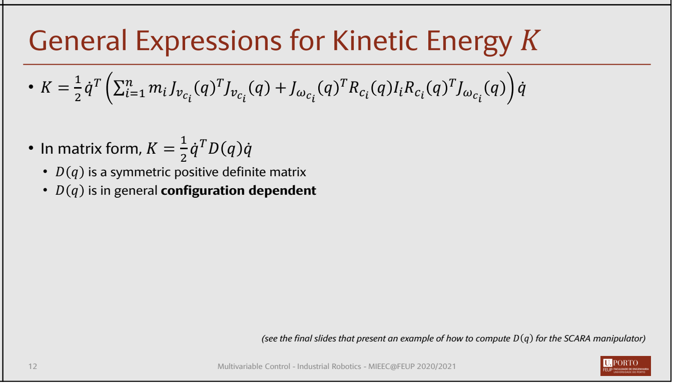
- Uniform → $\rho(x,y,z) = \rho$ (constant)
- $I_{xx} = \int_{-l_z/2}^{l_z/2}\int_{-l_y/2}^{l_y/2}\int_{-l_x/2}^{l_x/2}(y^2+z^2)\rho(x,y,z)\,dx\,dy\,dz = \rho\frac{l_xl_yl_z}{12}(l_y^2+l_z^2)$
- $I_{yy} = \rho\frac{l_xl_yl_z}{12}(l_x^2+l_z^2)$
- $I_{zz} = \rho\frac{l_xl_yl_z}{12}(l_x^2+l_y^2)$
- Cross products of inertia are zero
- $I = \begin{bmatrix} \rho\frac{l_xl_yl_z}{12}(l_y^2+l_z^2) & 0 & 0 \\ 0 & \rho\frac{l_xl_yl_z}{12}(l_x^2+l_z^2) & 0 \\ 0 & 0 & \rho\frac{l_xl_yl_z}{12}(l_x^2+l_y^2) \end{bmatrix}$





General Expressions for Kinetic Energy K

- n link robotic manipulator where the joint variables are the generalized coordinates (DH convention)
- m_i : mass of link i
- I_i : inertia matrix of link i
  - Evaluated around a coordinate frame $o_{c_i}x_{c_i}y_{c_i}z_{c_i}$ parallel to frame i but whose origin is at the link's center of mass
- $v_{c_i} = J_{v_{c_i}}(q)\dot{q}$: linear velocity vector at the center of mass of link i
- $\omega_{c_i} = J_{\omega_{c_i}}(q)\dot{q}$: angular velocity vector at the center of mass of link i
- $K = \frac{1}{2}\dot{q}^T\left(\sum_{i=1}^n m_iJ_{v_{c_i}}(q)^TJ_{v_{c_i}}(q) + J_{\omega_{c_i}}(q)^TR_{c_i}(q)I_iR_{c_i}(q)^TJ_{\omega_{c_i}}(q)\right)\dot{q}$



General Expressions for Kinetic Energy K

- $K = \frac{1}{2}\dot{q}^T\left(\sum_{i=1}^n m_iJ_{v_{c_i}}(q)^TJ_{v_{c_i}}(q) + J_{\omega_{c_i}}(q)^TR_{c_i}(q)I_iR_{c_i}(q)^TJ_{\omega_{c_i}}(q)\right)\dot{q}$
- In matrix form, $K = \frac{1}{2}\dot{q}^TD(q)\dot{q}$
  - $D(q)$ is a symmetric positive definite matrix
  - $D(q)$ is in general **configuration dependent**

(see the final slides that present an example of how to compute $D(q)$ for the SCARA manipulator)

General Expressions for Potential Energy P

- Concentrate the entire mass m of an object at its center of mass
- $P = \sum_{i=1}^{n} P_i = \sum_{i=1}^{n} -g^T o_{c_i} m_i$
  - g : gravitational vector in the inertial frame
  - o_{c_i} : coordinates of the center of mass of link i in the inertial frame
  - **Note:** the minus sign is required for the potential energy to be correctly computed
- IF robot contains elasticity (e.g., flexible joints), THEN potential energy P will include terms containing the energy stored in the elastic elements
- Potential energy P **depends on the configuration** of the robot **but not on its velocity**

(see the final slides that present an example of how to compute the potential energy P for the SCARA manipulator)

Equations of Motion

- Since $\mathcal{L} = K - P = \frac{1}{2}\sum_{i,j} d_{ij}(q)\dot{q}_i\dot{q}_j - P(q)$:
  - $\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_k} = \sum_j d_{kj}\ddot{q}_j + \sum_{i,j}\frac{\partial d_{kj}}{\partial q_i}\dot{q}_i\dot{q}_j$
  - $\frac{\partial \mathcal{L}}{\partial q_k} = \frac{1}{2}\sum_{i,j}\frac{\partial d_{ij}}{\partial q_k}\dot{q}_i\dot{q}_j - \frac{\partial P}{\partial q_k}$
- Euler-Lagrange equation: $\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_k} - \frac{\partial \mathcal{L}}{\partial q_k} = \tau_k$
  - $\sum_j d_{kj}\ddot{q}_j + \sum_{i,j}\left\{\frac{\partial d_{kj}}{\partial q_i} - \frac{1}{2}\frac{\partial d_{ij}}{\partial q_k}\right\}\dot{q}_i\dot{q}_j + \frac{\partial P}{\partial q_k} = \sum_j d_{kj}\ddot{q}_j + \sum_{i,j} c_{ijk}\dot{q}_i\dot{q}_j + \frac{\partial P}{\partial q_k} = \tau_k$
  - Christoffel symbols of the 1st kind: $c_{ijk} = \left\{\frac{\partial d_{kj}}{\partial q_i} - \frac{1}{2}\frac{\partial d_{ij}}{\partial q_k}\right\} = \frac{1}{2}\left\{\frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} - \frac{1}{2}\frac{\partial d_{ij}}{\partial q_k}\right\}$
    - Reduces the computation effort because $c_{ijk} = c_{jik}$ for a given k

Equations of Motion

- In matrix form, $D(q)\ddot{q} + C(q,\dot{q})\dot{q} + \phi(q) = \tau$
  - Where the (k,j)-th element of the matrix $C(q,\dot{q})$ is c_{kj}
  - $c_{kj} = \sum_{i=1}^{n} c_{ijk}(q)\dot{q}_i$
    $= \sum_{i=1}^{n}\frac{1}{2}\left\{\frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} - \frac{1}{2}\frac{\partial d_{ij}}{\partial q_k}\right\}\dot{q}_i$
  - $\phi(q) = \frac{\partial P}{\partial q_k}$ is the influence of the gravity

Outline

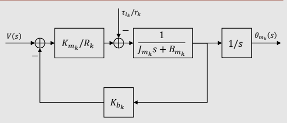
- Dynamics
- Multivariable Control
- SCARA Manipulator: Practical application of inverse dynamics

Multivariable Control

- **Previously** (independent joint control)**:**
  - Single-Input / Single-Output model
  - Coupling effects → treated as disturbances
- **Reality:** robot manipulator is a complex, non-linear, multivariable system!
- Multivariable Control
  - A more rigorous analysis of the performance of control systems
  - Design a robust and adaptive nonlinear control laws

Model of a Revolute Joint

- r_k : gear reduction ratio ($[r_k : 1]$)
- K_{m_k} : torque constant (N.m.A^{-1})
- K_{b_k} : speed constant (V.s.rad^{-1})
  - **Note:** $K_{m_k,S.I.} = K_{b_k,S.I.}$
- J_{m_k} : inertia of the motor (Kg.m^2)
- B_{m_k} : motor viscous constant (N.m.s)



- $J_{m_k}\ddot{\theta}_{m_k} + B_k\dot{\theta}_{m_k} = \frac{K_{m_k}V_k}{R_k} - \frac{\tau_{l_k}}{r_k}$, where $B_k = B_{m_k} + \frac{K_{b_k}K_{m_k}}{R_k}$ and $\theta_{m_k} = r_k q_k$
- $r_k^2 J_{m_k}\ddot{q}_k + r_k^2 B_k\dot{q}_k + \tau_k = \frac{r_k K_{m_k} V_k}{R_k} = u_k$

Equations of Motion

- Motor model:
  - $r_k^2 J_{m_k} \ddot{q}_k + r_k^2 B_k \dot{q}_k + \tau_k = \frac{r_k K_{m_k} V_k}{R_k} = u_k$
- Euler-Lagrange equation:
  - $\sum_j d_{kj}\ddot{q}_j + \sum_{i,j}\left\{\frac{\partial d_{kj}}{\partial q_i} - \frac{1}{2}\frac{\partial d_{ij}}{\partial q_k}\right\}\dot{q}_i\dot{q}_j + \frac{\partial P}{\partial q_k} = \sum_j d_{kj}\ddot{q}_j + \sum_{i,j} c_{ijk}\dot{q}_i\dot{q}_j + \frac{\partial P}{\partial q_k} = \tau_k$

- $r_k^2 J_{m_k}\ddot{q}_k + \sum_j d_{kj}\ddot{q}_j + \sum_{i,j} c_{ijk}\dot{q}_i\dot{q}_j + r_k^2 B_k \dot{q}_k + \phi_k = u$
- $M(q)\ddot{q} + C(q,\dot{q})\dot{q} + B\dot{q} + \phi(q) = u$
  - $M(q) = D(q) + J$, where J is a diagonal matrix with elements $r_k^2 J_{m_k}$
  - B is a diagonal matrix with elements $B_{m_k} + \frac{K_{b_k} K_{m_k}}{R_k}$

Inverse Dynamics

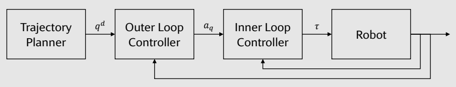
- **Goal:** nonlinear feedback control ($u = f(q,\dot{q},t)$) with a linear closed loop system

- Outer loop: $\ddot{q} = a_q$
  - Output of the loop : $a_q = -K_0 q - K_1\dot{q} + r$
  - Reference input : $r = \ddot{q}^d + K_0 q^d + K_1 \dot{q}^d$
  - Gain matrices K_0, K_1: (possible values)
    - $K_0 = \text{diag}\{\omega_1^2, \dots, \omega_n^2\}$
    - $K_1 = \text{diag}\{2\omega_1, \dots, 2\omega_n\}$

- Inner loop: $u = M(q)a_q + C(q,\dot{q})\dot{q} + B\dot{q} + \phi(q)$

(see the final slides that present a practical application of inverse dynamics with the SCARA manipulator)

Inverse Dynamics



Outline

- Dynamics
- Multivariable Control
- SCARA Manipulator: Practical application of inverse dynamics

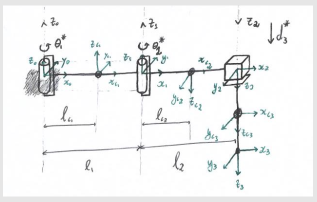
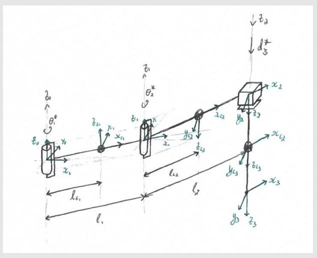
SCARA Manipulator

- SCARA: Selective Compliant Articulated Robot for Assembly
- Popular manipulator tailored for assembly operations
- RRP configuration

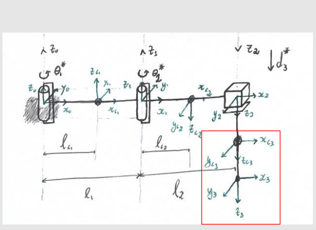
Epson SCARA E2L653S

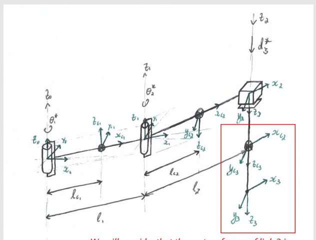
ABB IRB 910SC

Forward Kinematics (DH convention)

Forward Kinematics (DH convention)





We will consider that the center of mass of link 3 is align with the end-effector's coordinate frame

Forward Kinematics (DH convention)

- $H_1^0 = \begin{bmatrix} c_{q_1} & -s_{q_1} & 0 & l_1 c_{q_1} \\ s_{q_1} & c_{q_1} & 0 & l_1 s_{q_1} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Link i	$\boldsymbol{\theta_i}$	$\boldsymbol{d_i}$	$\boldsymbol{a_i}$	$\boldsymbol{\alpha_i}$
1	q_1	0 m	l_1	0º
2	q_2	0 m	l_2	180º
3	0º	q_3	0 m	0º

- $H_2^0 = \begin{bmatrix} c_{q_1+q_2} & s_{q_1+q_2} & 0 & l_1 c_{q_1} + l_2 c_{q_1+q_2} \\ s_{q_1+q_2} & -c_{q_1+q_2} & 0 & l_1 s_{q_1} + l_2 s_{q_1+q_2} \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- $H_3^0 = \begin{bmatrix} c_{q_1+q_2} & s_{q_1+q_2} & 0 & l_1 c_{q_1} + l_2 c_{q_1+q_2} \\ s_{q_1+q_2} & -c_{q_1+q_2} & 0 & l_1 s_{q_1} + l_2 s_{q_1+q_2} \\ 0 & 0 & -1 & -q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Forward Kinematics (DH convention)

- $H_1^0 = \begin{bmatrix} c_{q_1} & -s_{q_1} & 0 & l_1 c_{q_1} \\ s_{q_1} & c_{q_1} & 0 & l_1 s_{q_1} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Link i	$\boldsymbol{\theta_i}$	$\boldsymbol{d_i}$	$\boldsymbol{a_i}$	$\boldsymbol{\alpha_i}$
1	q_1	0 m	l_1	0º
2	q_2	0 m	l_2	180º
3	0º	q_3	0 m	0º

...

- $H_2^0 = \begin{array}{c} \begin{array}{cccc} x_2^0 & y_2^0 & z_2^0 & o_2^0 \end{array} \\ \begin{bmatrix} \boxed{\begin{matrix} c_{q_1+q_2} \\ s_{q_1+q_2} \\ 0 \end{matrix}} & \boxed{\begin{matrix} s_{q_1+q_2} \\ -c_{q_1+q_2} \\ 0 \end{matrix}} & \boxed{\begin{matrix} 0 \\ 0 \\ -1 \end{matrix}} & \boxed{\begin{matrix} l_1 c_{q_1} + l_2 c_{q_1+q_2} \\ l_1 s_{q_1} + l_2 s_{q_1+q_2} \\ 0 \end{matrix}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$

...

- $H_3^0 = \begin{bmatrix} c_{q_1+q_2} & s_{q_1+q_2} & 0 & l_1 c_{q_1} + l_2 c_{q_1+q_2} \\ s_{q_1+q_2} & -c_{q_1+q_2} & 0 & l_1 s_{q_1} + l_2 s_{q_1+q_2} \\ 0 & 0 & -1 & -q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Inverse Kinematics

- $c_2 = \frac{x^2+y^2-l_1^2-l_2^2}{2 l_1 l_2}$

- $q_2 = \operatorname{atan2}\left(-\sqrt{1-c_2^2}, c_2\right)$
- $q_1 = \operatorname{atan2}(y,x) - \operatorname{atan2}\left(l_2 s_{q_2}, l_1 + l_2 c_{q_2}\right)$
- $q_3 = -z$

- **Note:** the inverse kinematics are still require for the inverse dynamics (compute q^d)

Velocity Kinematics

- $J_3^0 = \begin{bmatrix} -l_1 s_{q_1} - l_2 s_{q_1+q_2} & -l_2 s_{q_1+q_2} & 0 \\ l_1 c_{q_1} + l_2 c_{q_1+q_2} & l_2 c_{q_1+q_2} & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$

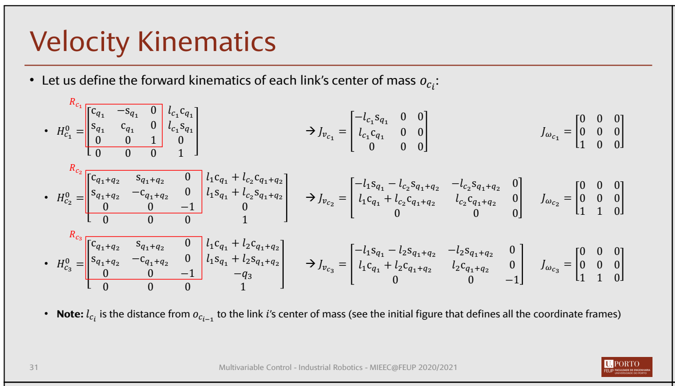
- Matrix J_3^0 defines the velocity kinematics of the end-effector relative to the coordinate frame 0
- However, we can define the velocity kinematics relative to any point of the robot manipulator

- **Note:** the inverse kinematics require the definition of the velocity kinematics of the links' centers of mass

Velocity Kinematics

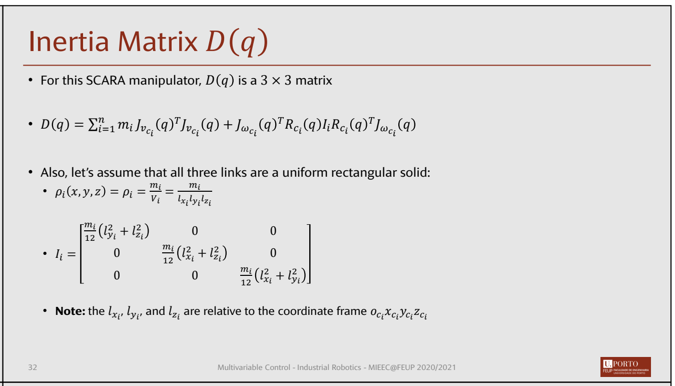
- Let us define the forward kinematics of each link's center of mass o_{c_i}:
  - $H_{c_1}^0 = \begin{bmatrix} c_{q_1} & -s_{q_1} & 0 & l_{c_1} c_{q_1} \\ s_{q_1} & c_{q_1} & 0 & l_{c_1} s_{q_1} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $\rightarrow J_{v_{c_1}} = \begin{bmatrix} -l_{c_1} s_{q_1} & 0 & 0 \\ l_{c_1} c_{q_1} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $J_{\omega_{c_1}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$
  - $H_{c_2}^0 = \begin{bmatrix} c_{q_1+q_2} & s_{q_1+q_2} & 0 & l_1 c_{q_1} + l_{c_2} c_{q_1+q_2} \\ s_{q_1+q_2} & -c_{q_1+q_2} & 0 & l_1 s_{q_1} + l_{c_2} s_{q_1+q_2} \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $\rightarrow J_{v_{c_2}} = \begin{bmatrix} -l_1 s_{q_1} - l_{c_2} s_{q_1+q_2} & -l_{c_2} s_{q_1+q_2} & 0 \\ l_1 c_{q_1} + l_{c_2} c_{q_1+q_2} & l_{c_2} c_{q_1+q_2} & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $J_{\omega_{c_2}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$
  - $H_{c_3}^0 = \begin{bmatrix} c_{q_1+q_2} & s_{q_1+q_2} & 0 & l_1 c_{q_1} + l_2 c_{q_1+q_2} \\ s_{q_1+q_2} & -c_{q_1+q_2} & 0 & l_1 s_{q_1} + l_2 s_{q_1+q_2} \\ 0 & 0 & -1 & -q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $\rightarrow J_{v_{c_3}} = \begin{bmatrix} -l_1 s_{q_1} - l_2 s_{q_1+q_2} & -l_2 s_{q_1+q_2} & 0 \\ l_1 c_{q_1} + l_2 c_{q_1+q_2} & l_2 c_{q_1+q_2} & 0 \\ 0 & 0 & -1 \end{bmatrix}$ $J_{\omega_{c_3}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$

  - **Note:** l_{c_i} is the distance from $o_{c_{i-1}}$ to the link i's center of mass (see the initial figure that defines all the coordinate frames)



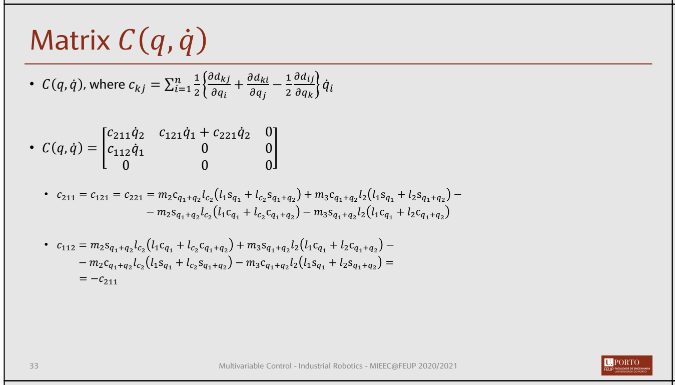
Velocity Kinematics

- Let us define the forward kinematics of each link's center of mass o_{c_i}:
  - R_{c_1} $$H_{c_1}^0 = \begin{bmatrix} \boxed{\begin{matrix} c_{q_1} & -s_{q_1} & 0 \\ s_{q_1} & c_{q_1} & 0 \\ 0 & 0 & 1 \end{matrix}} & \begin{matrix} l_{c_1} c_{q_1} \\ l_{c_1} s_{q_1} \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \rightarrow J_{v_{c_1}} = \begin{bmatrix} -l_{c_1} s_{q_1} & 0 & 0 \\ l_{c_1} c_{q_1} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad J_{\omega_{c_1}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$
  - R_{c_2} $$H_{c_2}^0 = \begin{bmatrix} \boxed{\begin{matrix} c_{q_1+q_2} & s_{q_1+q_2} & 0 \\ s_{q_1+q_2} & -c_{q_1+q_2} & 0 \\ 0 & 0 & -1 \end{matrix}} & \begin{matrix} l_1 c_{q_1} + l_{c_2} c_{q_1+q_2} \\ l_1 s_{q_1} + l_{c_2} s_{q_1+q_2} \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \rightarrow J_{v_{c_2}} = \begin{bmatrix} -l_1 s_{q_1} - l_{c_2} s_{q_1+q_2} & -l_{c_2} s_{q_1+q_2} & 0 \\ l_1 c_{q_1} + l_{c_2} c_{q_1+q_2} & l_{c_2} c_{q_1+q_2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad J_{\omega_{c_2}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$
  - R_{c_3} $$H_{c_3}^0 = \begin{bmatrix} \boxed{\begin{matrix} c_{q_1+q_2} & s_{q_1+q_2} & 0 \\ s_{q_1+q_2} & -c_{q_1+q_2} & 0 \\ 0 & 0 & -1 \end{matrix}} & \begin{matrix} l_1 c_{q_1} + l_2 c_{q_1+q_2} \\ l_1 s_{q_1} + l_2 s_{q_1+q_2} \\ -q_3 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \rightarrow J_{v_{c_3}} = \begin{bmatrix} -l_1 s_{q_1} - l_2 s_{q_1+q_2} & -l_2 s_{q_1+q_2} & 0 \\ l_1 c_{q_1} + l_2 c_{q_1+q_2} & l_2 c_{q_1+q_2} & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad J_{\omega_{c_3}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$
  - **Note:** l_{c_i} is the distance from $o_{c_{i-1}}$ to the link i's center of mass (see the initial figure that defines all the coordinate frames)



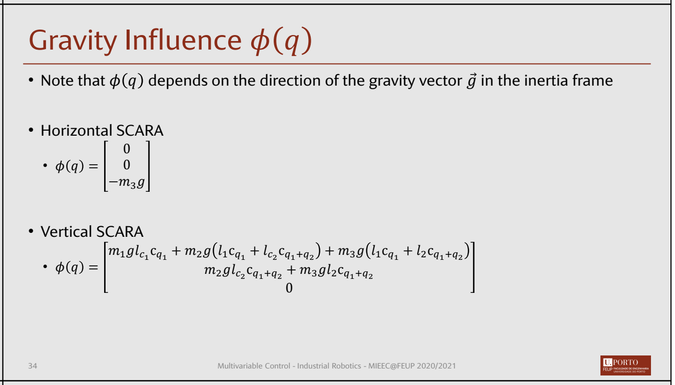
Inertia Matrix $D(q)$

- For this SCARA manipulator, $D(q)$ is a 3×3 matrix
- $D(q) = \sum_{i=1}^{n} m_i J_{v_{c_i}}(q)^T J_{v_{c_i}}(q) + J_{\omega_{c_i}}(q)^T R_{c_i}(q) I_i R_{c_i}(q)^T J_{\omega_{c_i}}(q)$
- Also, let's assume that all three links are a uniform rectangular solid:
  - $\rho_i(x,y,z) = \rho_i = \frac{m_i}{V_i} = \frac{m_i}{l_{x_i} l_{y_i} l_{z_i}}$
  - $$I_i = \begin{bmatrix} \frac{m_i}{12}\left(l_{y_i}^2 + l_{z_i}^2\right) & 0 & 0 \\ 0 & \frac{m_i}{12}\left(l_{x_i}^2 + l_{z_i}^2\right) & 0 \\ 0 & 0 & \frac{m_i}{12}\left(l_{x_i}^2 + l_{y_i}^2\right) \end{bmatrix}$$
  - **Note:** the l_{x_i}, l_{y_i}, and l_{z_i} are relative to the coordinate frame $o_{c_i} x_{c_i} y_{c_i} z_{c_i}$



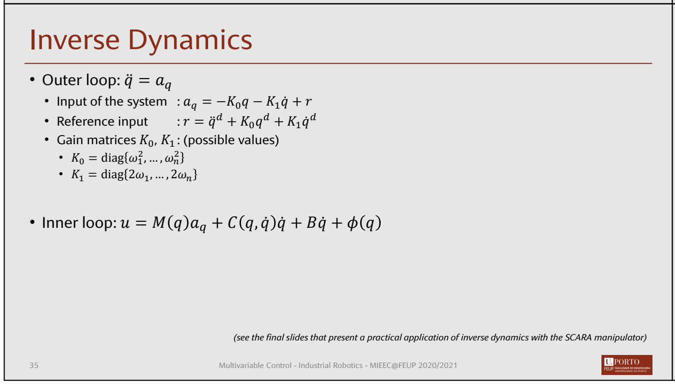
Matrix $C(q,\dot{q})$

- $C(q,\dot{q})$, where $c_{kj} = \sum_{i=1}^{n} \frac{1}{2}\left\{\frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} - \frac{1}{2}\frac{\partial d_{ij}}{\partial q_k}\right\}\dot{q}_i$
- $$C(q,\dot{q}) = \begin{bmatrix} c_{211}\dot{q}_2 & c_{121}\dot{q}_1 + c_{221}\dot{q}_2 & 0 \\ c_{112}\dot{q}_1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
  - $$\begin{aligned} c_{211} = c_{121} = c_{221} &= m_2 c_{q_1+q_2} l_{c_2}\left(l_1 s_{q_1} + l_{c_2} s_{q_1+q_2}\right) + m_3 c_{q_1+q_2} l_2\left(l_1 s_{q_1} + l_2 s_{q_1+q_2}\right) - \\ &- m_2 s_{q_1+q_2} l_{c_2}\left(l_1 c_{q_1} + l_{c_2} c_{q_1+q_2}\right) - m_3 s_{q_1+q_2} l_2\left(l_1 c_{q_1} + l_2 c_{q_1+q_2}\right) \end{aligned}$$
  - $$\begin{aligned} c_{112} &= m_2 s_{q_1+q_2} l_{c_2}\left(l_1 c_{q_1} + l_{c_2} c_{q_1+q_2}\right) + m_3 s_{q_1+q_2} l_2\left(l_1 c_{q_1} + l_2 c_{q_1+q_2}\right) - \\ &- m_2 c_{q_1+q_2} l_{c_2}\left(l_1 s_{q_1} + l_{c_2} s_{q_1+q_2}\right) - m_3 c_{q_1+q_2} l_2\left(l_1 s_{q_1} + l_2 s_{q_1+q_2}\right) = \\ &= -c_{211} \end{aligned}$$



Gravity Influence $\phi(q)$

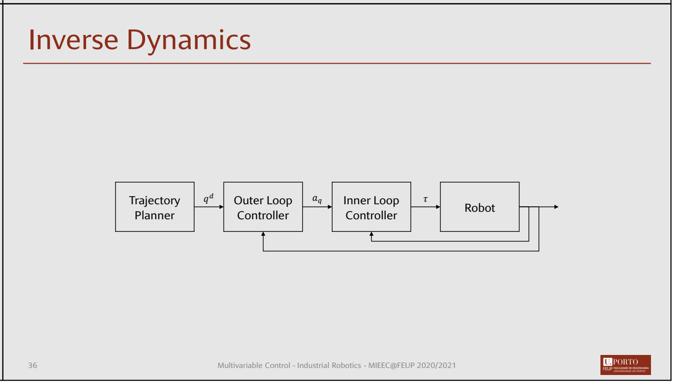
- Note that $\phi(q)$ depends on the direction of the gravity vector $\vec{g}$ in the inertia frame
- Horizontal SCARA
  - $$\phi(q) = \begin{bmatrix} 0 \\ 0 \\ -m_3 g \end{bmatrix}$$
- Vertical SCARA
  - $$\phi(q) = \begin{bmatrix} m_1 g l_{c_1} c_{q_1} + m_2 g\left(l_1 c_{q_1} + l_{c_2} c_{q_1+q_2}\right) + m_3 g\left(l_1 c_{q_1} + l_2 c_{q_1+q_2}\right) \\ m_2 g l_{c_2} c_{q_1+q_2} + m_3 g l_2 c_{q_1+q_2} \\ 0 \end{bmatrix}$$



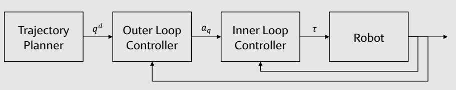
Inverse Dynamics

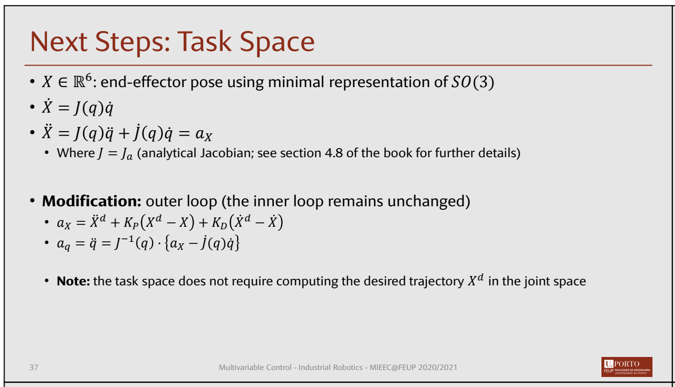
- Outer loop: $\ddot{q} = a_q$
  - Input of the system : $a_q = -K_0 q - K_1 \dot{q} + r$
  - Reference input : $r = \ddot{q}^d + K_0 q^d + K_1 \dot{q}^d$
  - Gain matrices K_0, K_1 : (possible values)
    - $K_0 = \text{diag}\{\omega_1^2, \dots, \omega_n^2\}$
    - $K_1 = \text{diag}\{2\omega_1, \dots, 2\omega_n\}$
- Inner loop: $u = M(q)a_q + C(q,\dot{q})\dot{q} + B\dot{q} + \phi(q)$

(see the final slides that present a practical application of inverse dynamics with the SCARA manipulator)



Inverse Dynamics



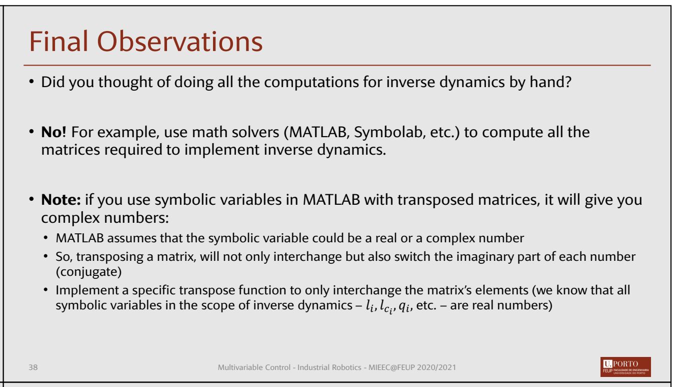


Next Steps: Task Space

- $X \in \mathbb{R}^6$: end-effector pose using minimal representation of $SO(3)$
- $\dot{X} = J(q)\dot{q}$
- $\ddot{X} = J(q)\ddot{q} + \dot{J}(q)\dot{q} = a_X$
  - Where $J = J_a$ (analytical Jacobian; see section 4.8 of the book for further details)

- **Modification:** outer loop (the inner loop remains unchanged)
  - $a_X = \ddot{X}^d + K_P(X^d - X) + K_D(\dot{X}^d - \dot{X})$
  - $a_q = \ddot{q} = J^{-1}(q) \cdot \{a_X - \dot{J}(q)\dot{q}\}$

  - **Note:** the task space does not require computing the desired trajectory X^d in the joint space

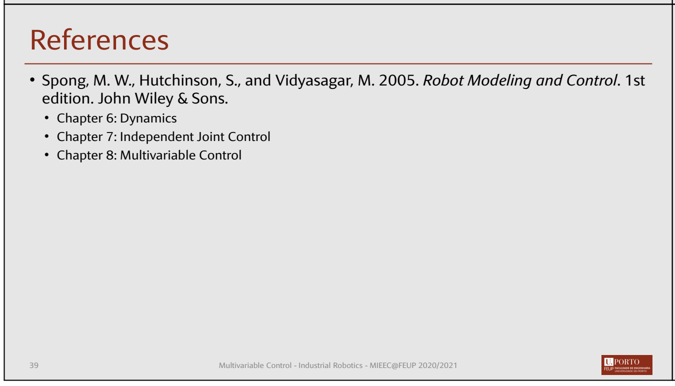


Final Observations

- Did you thought of doing all the computations for inverse dynamics by hand?

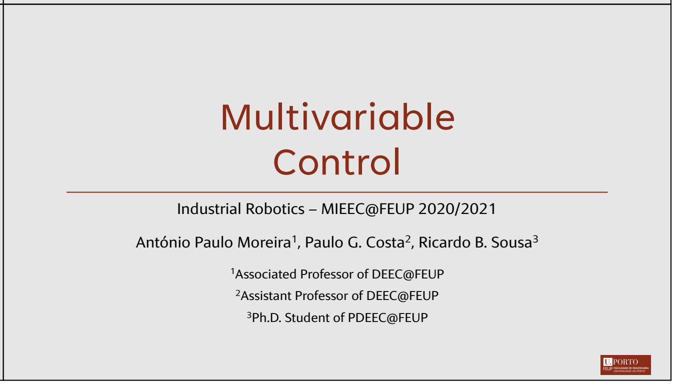
- **No!** For example, use math solvers (MATLAB, Symbolab, etc.) to compute all the matrices required to implement inverse dynamics.

- **Note:** if you use symbolic variables in MATLAB with transposed matrices, it will give you complex numbers:
  - MATLAB assumes that the symbolic variable could be a real or a complex number
  - So, transposing a matrix, will not only interchange but also switch the imaginary part of each number (conjugate)
  - Implement a specific transpose function to only interchange the matrix's elements (we know that all symbolic variables in the scope of inverse dynamics – l_i, l_{c_i}, q_i, etc. – are real numbers)



References

- Spong, M. W., Hutchinson, S., and Vidyasagar, M. 2005. *Robot Modeling and Control*. 1st edition. John Wiley & Sons.
  - Chapter 6: Dynamics
  - Chapter 7: Independent Joint Control
  - Chapter 8: Multivariable Control



Multivariable Control

Industrial Robotics – MIEEC@FEUP 2020/2021

António Paulo Moreira[1], Paulo G. Costa[2], Ricardo B. Sousa[3]

[1]Associated Professor of DEEC@FEUP

[2]Assistant Professor of DEEC@FEUP

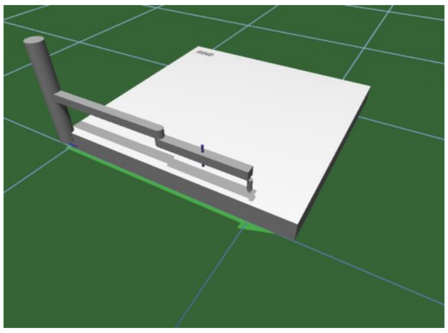
[3]Ph.D. Student of PDEEC@FEUP

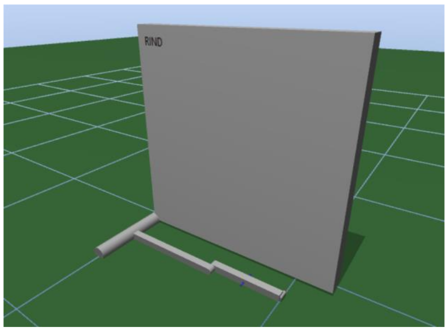
C Guide for the Laboratory Work

Laboratorial Work

Inverse Dynamics with Multivariable Control

The robot manipulators illustrated in the following two figures are equivalent to a SCARA configuration (RRP). This manipulator has three joints: the first two are revolute, and the other one is prismatic. As you can see in the two images, the only difference is the orientation of the manipulator relative to the inertial coordinate frame.





The main objective of this laboratory work is to implement the inverse dynamics control loop on the SCARA manipulators shown in the previous two figures. The manipulators are simulated in the SimTwo simulation environment.

Note that a practical application of inverse dynamics for the SCARA configuration is already exemplified in the slides presented in the theoretical class (and available in SIGARRA). Please review these slides before elaborating the laboratory work.

1. Analyze the code already implemented in SimTwo. There are functions available to compute the matrices and vectors require to implement the characteristic control loops of inverse dynamics.

2. Implement the inverse dynamics for the Horizontal SCARA (illustrated in the left image).
   a. Check if the orientation of the manipulator relative to the inertial frame is correct:
      i. Config > Control > Axis > RotUp: change the value in the right text box to 0 > SetRef.
      ii. Sheet: change the value in the cell right next to *Vertical* to 0.
   b. Implement the inverse dynamics in the Control procedure (check the notes present in the code).
   c. Test the inverse dynamics:
      i. Sheet: Click button ID > Change Ref, Wn > Set
   d. Change the value of wn and observe the differences in terms of tracking error and transient response.

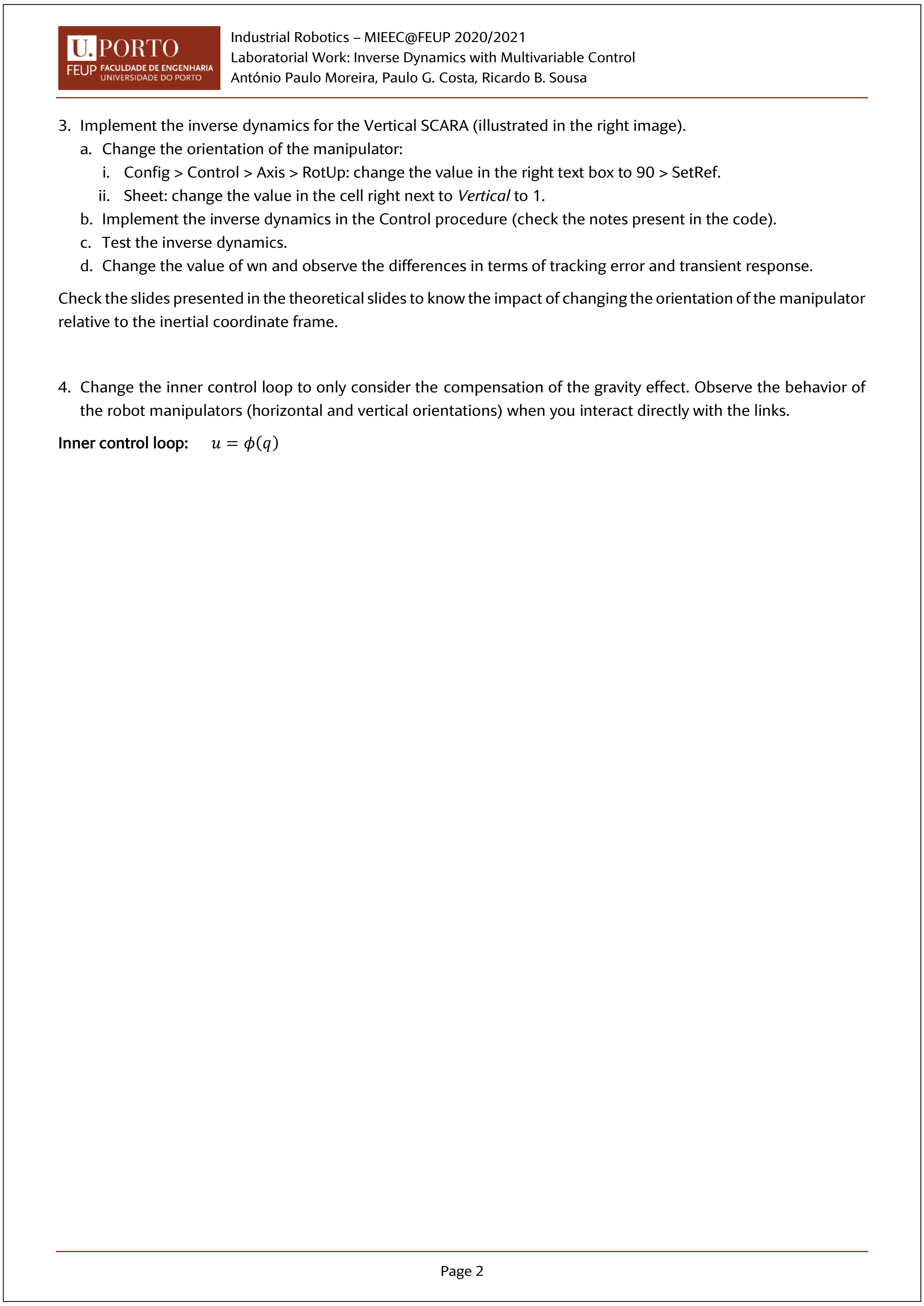
Outer control loop: $a_q = -K_0 q - K_1 \dot{q} + r$, where $r = \ddot{q}^d + K_0 q^d + K_1 \dot{q}^d$

Note: we will set $\ddot{q}^d$ and $\dot{q}^d$ to 0 because we are only evaluating the inverse dynamics for set-point

Inner control loop: $u = M(q)a_q + C(q,\dot{q})\dot{q} + B\dot{q} + \phi(q)$

Useful SimTwo functions:

- Add two matrices	$A + B$:	`MAdd( A: Matrix, B: Matrix): Matrix`
- Subtract one matrix to another	$A - B$:	`MSub( A: Matrix, B: Matrix): Matrix`
- Multiply two matrices	$A \cdot B$:	`MMult(A: Matrix, B: Matrix): Matrix`



3. Implement the inverse dynamics for the Vertical SCARA (illustrated in the right image).
   a. Change the orientation of the manipulator:
      i. Config > Control > Axis > RotUp: change the value in the right text box to 90 > SetRef.
      ii. Sheet: change the value in the cell right next to *Vertical* to 1.
   b. Implement the inverse dynamics in the Control procedure (check the notes present in the code).
   c. Test the inverse dynamics.
   d. Change the value of wn and observe the differences in terms of tracking error and transient response.

Check the slides presented in the theoretical slides to know the impact of changing the orientation of the manipulator relative to the inertial coordinate frame.

4. Change the inner control loop to only consider the compensation of the gravity effect. Observe the behavior of the robot manipulators (horizontal and vertical orientations) when you interact directly with the links.

Inner control loop: $u = \phi(q)$

D SimTwo

D.1 Implementation of inverse dynamics on a SCARA manipulator

Listing 11. Implementation in SimTwo of Inverse Dynamics on a SCARA Manipulator

```
// Constants
const
  NumJoints = 3;
  // Links
  l_1  = 0.6;   l_2  = 0.4;                 // length
  m_1  = 0.2;   m_2  = 0.2;   m_3  = 0.1;   // mass
  lc_1 = l_1/2; lc_2 = l_2/2;               // length to center of mass
  lx_1 = l_1;   lx_2 = l_2;   lx_3 = 0.02;  // dimensions
  ly_1 = 0.04;  ly_2 = 0.04;  ly_3 = 0.02;  //   (relative to frame oci_xci_yci)
  lz_1 = 0.04;  lz_2 = 0.04;  lz_3 = 0.05;
  rho_1 = m_1/(lx_1*ly_1*lz_1);             // mass density
  rho_2 = m_2/(lx_2*ly_2*lz_2);
  rho_3 = m_3/(lx_3*ly_3*lz_3);
  // Motors
  n_1  = 1;     n_2  = 1;     n_3  = 1;     // gear reduction ratio
  km_1 = 0.56;  km_2 = 0.56;  km_3 = 0.56;  // torque constant
  jm_1 = 0;     jm_2 = 0;     jm_3 = 0;     // inertia
  bm_1 = 0.10;  bm_2 = 0.10;  bm_3 = 0.10;  // viscous constant
  ri_1 = 0.12;  ri_2 = 0.12;  ri_3 = 0.12;  // internal resistance
  // Gravity
  //gacc = 9.80665;  // gravitational acceleration (m/s^2)
  gacc = 9.8;

// Global Variables
var iZAxis, iJ1Axis, iJ2Axis, iPen: integer;
    Q_meas, Qd1_meas, W_mat, K0_mat, K1_mat: Matrix;
    Q_desir, Qd1_desir, Qd2_desir, XYZ_desir: Matrix;
    InvDynON: boolean;



// Utilitary functions
// - set reference of  the joints (J1,J2: revolute; J3: prismatic)
procedure SetValuesJoints(Values: matrix);
begin
  // Controller mode: position control
  SetMotorControllerMode(0, iJ1Axis, 'pidposition');
  SetMotorControllerMode(0, iJ2Axis, 'pidposition');
  SetMotorControllerMode(0, iZAxis , 'pidposition');
  // Controller parameters:          ki  kp  kp  kf
  SetMotorControllerPars(0, iJ1Axis,  0, 25,  3,  0);
  SetMotorControllerPars(0, iJ2Axis,  0, 50,  4,  0);
  SetMotorControllerPars(0, iZAxis ,  0,  1,  5,  0);
  // Set position reference
  SetAxisPosRef(0, iJ1Axis, Mgetv(Values, 0, 0));
  SetAxisPosRef(0, iJ2Axis, Mgetv(Values, 1, 0));
  SetAxisPosRef(0, iZAxis , Mgetv(Values, 2, 0));
end;

// - compute Denavit-Hartenberg (DH) matrix
function DHMat(theta, d, a, alpha: double): Matrix;
var ct, st, ca, sa: double;
    R: Matrix;
begin
```

```
  ct := cos(theta);
  st := sin(theta);
  ca := cos(alpha);
  sa := sin(alpha);

  R := Meye(4);
  MSetV(R,0,0,ct); MSetV(R,0,1,-st*ca); MSetV(R,0,2, st*sa); MSetV(R,0,3,a*ct);
  MSetV(R,1,0,st); MSetV(R,1,1, ct*ca); MSetV(R,1,2,-ct*sa); MSetV(R,1,3,a*st);
  MSetV(R,2,0, 0); MSetV(R,2,1, sa    ); MSetV(R,2,2, ca    ); MSetV(R,2,3,d    );
  result := R;
end;

// - rotation matrix (z axis)
function RotZ(theta: double): Matrix;
var ct, st: double;
    R: Matrix;
begin
  ct := cos(theta);
  st := sin(theta);
  R := Mzeros(3,3);

  MSetV(R,0,0,ct); MSetV(R,0,1,-st);
  MSetV(R,1,0,st); MSetV(R,1,1, ct);
                                    MSetV(R,2,2,1);

  result := R;
end;

// - torque mode (controllers deactivated)
procedure SetJointsControllerState(activate: boolean);
begin
  SetMotorControllerState(0, iJ1Axis, activate);
  SetMotorControllerState(0, iJ2Axis, activate);
  SetMotorControllerState(0, iZAxis , activate);
end;

// - set torque value for the joints
procedure SetTorque(Torque: matrix);
begin
  SetAxisTorqueRef(0, iJ1Axis, Mgetv(Torque, 0, 0));
  SetAxisTorqueRef(0, iJ2Axis, Mgetv(Torque, 1, 0));
  SetAxisTorqueRef(0, iZAxis , Mgetv(Torque, 2, 0));
end;



// Inverse Dynamics (ID)
// - set inverse dynamics parameters
procedure SetIDParameters;
begin
  // wn
  W_mat := RangeToMatrix(12,10,3,1);
  // K0 = wn ^ 2
  K0_mat := Mzeros(3,3);
  MSetV(K0_mat,0,0,MGetV(W_mat,0,0)*MGetV(W_mat,0,0));
  MSetV(K0_mat,1,1,MGetV(W_mat,1,0)*MGetV(W_mat,1,0));
  MSetV(K0_mat,2,2,MGetV(W_mat,2,0)*MGetV(W_mat,2,0));
  // K1 = 2 * wn
  K1_mat := Mzeros(3,3);
  MSetV(K1_mat,0,0,2*MGetV(W_mat,0,0));
  MSetV(K1_mat,1,1,2*MGetV(W_mat,1,0));
  MSetV(K1_mat,2,2,2*MGetV(W_mat,2,0));
end;

// - inertia matrix (D)
function IDDMat(_Q: Matrix): Matrix;
var q1_, q2_, q3_: double;
    cq1, cq2, cq1pq2, sq1, sq2, sq1pq2: double;
    Jvc_1, Jvc_2, Jvc_3: Matrix;
    Jwc_1, Jwc_2, Jwc_3: Matrix;
    Rc_1, Rc_2, Rc_3: Matrix;
    I_1, I_2, I_3: Matrix;
begin
  q1_ := MGetV(_Q, 0, 0);
  q2_ := MGetV(_Q, 1, 0);
  q3_ := MGetV(_Q, 2, 0);
  cq1 := cos(q1_);
  sq1 := sin(q1_);
  cq2 := cos(q2_);
  sq2 := sin(q2_);
  cq1pq2 := cos(q1_+q2_);
  sq1pq2 := sin(q1_+q2_);

  // Matrices Jvc_i (linear speed jacobian relative to oci_xci_yci_zci)
  Jvc_1 := Mzeros(3,3);
  Msetv(Jvc_1,0,0,-lc_1*sq1);
  Msetv(Jvc_1,1,0, lc_1*cq1);

  Jvc_2 := Mzeros(3,3);
  Msetv(Jvc_2,0,0,-l_1*sq1-lc_2*sq1pq2); Msetv(Jvc_2,0,1,-lc_2*sq1pq2);
  Msetv(Jvc_2,1,0, l_1*cq1+lc_2*cq1pq2); Msetv(Jvc_2,1,1, lc_2*cq1pq2);

  Jvc_3 := Mzeros(3,3);
  Msetv(Jvc_3,0,0,-l_1*sq1-l_2*sq1pq2); Msetv(Jvc_3,0,1,-l_2*sq1pq2);
  Msetv(Jvc_3,1,0, l_1*cq1+l_2*cq1pq2); Msetv(Jvc_3,1,1, l_2*cq1pq2);
                                                                Msetv(Jvc_3
                                                                   ,2,2,-1);

  // Matrices Jwc_i (angular speed jacobian relative to oci_xci_yci_zci)
  Jwc_1 := Mzeros(3,3);
  Msetv(Jwc_1,2,0,1);

  Jwc_2 := Mzeros(3,3);
  Msetv(Jwc_2,2,0,1); Msetv(Jwc_2,2,1,1);

  Jwc_3 := Jwc_2;

  // Matrices Rc_i (orientation matrices relative to oci_xci_yci_zci)
  Rc_1 := RotZ(q1_);

  Rc_2 := Mzeros(3,3);
  MSetV(Rc_2,0,0,cq1pq2); MSetV(Rc_2,0,1, sq1pq2);
  MSetV(Rc_2,1,0,sq1pq2); MSetV(Rc_2,1,1,-cq1pq2);
                                              MSetV(Rc_2,2,2,-1);

  Rc_3 := Rc_2;

  // Matrices I_i (inertia tensors relative to oci_xci_yci_zci)
  I_1 := Mzeros(3,3);
  MSetV(I_1,0,0,rho_1*lx_1*ly_1*lz_1*(ly_1*ly_1+lz_1*lz_1)/12);
  MSetV(I_1,1,1,rho_1*lx_1*ly_1*lz_1*(lx_1*lx_1+lz_1*lz_1)/12);
  MSetV(I_1,2,2,rho_1*lx_1*ly_1*lz_1*(ly_1*ly_1+lx_1*lx_1)/12);

  I_2 := Mzeros(3,3);
  MSetV(I_2,0,0,rho_2*lx_2*ly_2*lz_2*(ly_2*ly_2+lz_2*lz_2)/12);
  MSetV(I_2,1,1,rho_2*lx_2*ly_2*lz_2*(lx_2*lx_2+lz_2*lz_2)/12);
  MSetV(I_2,2,2,rho_2*lx_2*ly_2*lz_2*(ly_2*ly_2+lx_2*lx_2)/12);

  I_3 := Mzeros(3,3);
  MSetV(I_3,0,0,rho_3*lx_3*ly_3*lz_3*(ly_3*ly_3+lz_3*lz_3)/12);
  MSetV(I_3,1,1,rho_3*lx_3*ly_3*lz_3*(lx_3*lx_3+lz_3*lz_3)/12);
  MSetV(I_3,2,2,rho_3*lx_3*ly_3*lz_3*(ly_3*ly_3+lx_3*lx_3)/12);

  // Inertia Matrix D(Q) (Q = [q1 q2 q3])
  result := MMultReal( MMult(MTran(Jvc_1),Jvc_1) , m_1 );
  result := MAdd( result , MMultReal( MMult(MTran(Jvc_2),Jvc_2) , m_2 ) );
  result := MAdd( result , MMultReal( MMult(MTran(Jvc_3),Jvc_3) , m_3 ) );
  result := MAdd( result , MMult(MMult(MMult(MTran(Jwc_1),Rc_1),I_1),MMult(MTran(Rc_1
      ),Jwc_1)) );
  result := MAdd( result , MMult(MMult(MMult(MTran(Jwc_2),Rc_2),I_2),MMult(MTran(Rc_2
      ),Jwc_2)) );
  result := MAdd( result , MMult(MMult(MMult(MTran(Jwc_3),Rc_3),I_3),MMult(MTran(Rc_3
      ),Jwc_3)) );
end;

// - Diagonal matrix r^2 * Jm
function IDJmMat: Matrix;
begin
  result := Mzeros(3,3);
  MSetV(result,0,0,n_1*n_1*jm_1);
  MSetV(result,1,1,n_2*n_2*jm_2);
  MSetV(result,2,2,n_3*n_3*jm_3);
end;

// - M matrix (D + J)
function IDMMat(_Q: Matrix): Matrix;
begin
  result := MAdd( IDDMat(_Q) , IDJmMat );
end;

// - C matrix (C)
function IDCMat(_Q,_Qd1: Matrix): Matrix;
var q1_, q2_, q3_: double;
    q1d1_, q2d1_, q3d1_: double;
    cq1, cq2, cq1pq2, sq1, sq2, sq1pq2: double;
    c211, c112, c121, c221: double;
begin
  q1_ := MGetV(_Q, 0, 0);
  q2_ := MGetV(_Q, 1, 0);
  q3_ := MGetV(_Q, 2, 0);
  q1d1_ := MGetV(_Qd1, 0, 0);
  q2d1_ := MGetV(_Qd1, 1, 0);
  q3d1_ := MGetV(_Qd1, 2, 0);
  cq1 := cos(q1_);
  sq1 := sin(q1_);
  cq2 := cos(q2_);
  sq2 := sin(q2_);
  cq1pq2 := cos(q1_+q2_);
  sq1pq2 := sin(q1_+q2_);

  // Christoffel symbols
  c211 := m_3 * cq1pq2 * l_2  * ( l_1 * sq1 + l_2  * sq1pq2 ) -
          m_2 * sq1pq2 * lc_2 * ( l_1 * cq1 + lc_2 * cq1pq2 ) -
          m_3 * sq1pq2 * l_2  * ( l_1 * cq1 + l_2  * cq1pq2 ) +
          m_2 * cq1pq2 * lc_2 * ( l_1 * sq1 + lc_2 * sq1pq2 );
  c121 := c211;
  c221 := c211;
  c112 := m_3 * sq1pq2 * l_2  * ( l_1 * cq1 + l_2  * cq1pq2 ) +
          m_2 * sq1pq2 * lc_2 * ( l_1 * cq1 + lc_2 * cq1pq2 ) -
          m_3 * cq1pq2 * l_2  * ( l_1 * sq1 + l_2  * sq1pq2 ) -
          m_2 * cq1pq2 * lc_2 * ( l_1 * sq1 + lc_2 * sq1pq2 );

  // Matrix C
  result := Mzeros(3,3);
  MSetV(result,0,0,c211*q2d1_); MSetV(result,0,1,c121*q1d1_+c221*q2d1_);
  MSetV(result,1,0,c112*q1d1_);
end;

// - Diagonal matrix B
function IDBMat: Matrix;
begin
  result := Mzeros(3,3);
  MSetV(result,0,0,bm_1+km_1*km_1/ri_1);
  MSetV(result,1,1,bm_2+km_2*km_2/ri_2);
  MSetV(result,2,2,bm_3+km_3*km_3/ri_3);
end;

// - gravity influence (Phi matrix)
function IDPhiMatHorizontalSCARA(_Q: Matrix): Matrix;
begin
  result := Mzeros(3,1);
  MSetV(result,2,0,-gacc*m_3);
end;

function IDPhiMatVerticalSCARA(_Q: Matrix): Matrix;
var q1_, q2_, q3_: double;
    cq1, cq2, cq1pq2, sq1, sq2, sq1pq2: double;
begin
  q1_ := MGetV(_Q, 0, 0);
  q2_ := MGetV(_Q, 1, 0);
  q3_ := MGetV(_Q, 2, 0);
  cq1 := cos(q1_);
  cq1pq2 := cos(q1_+q2_);

  result := Mzeros(3,1);
  MSetV(result,0,0,gacc*lc_1*cq1*m_1 +
                   gacc*l_1 *cq1*m_2 + gacc*lc_2*cq1pq2*m_2+
                   gacc*l_1 *cq1*m_3 + gacc*l_2 *cq1pq2*m_3);
  MSetV(result,1,0,gacc*lc_2*cq1pq2*m_2 +
                   gacc*l_2 *cq1pq2*m_3);
  MSetV(result,2,0,0);
end;



// Forward Kinematics
```

```
function DK3(_Q: matrix): matrix;
var A1, A2, A3: Matrix;
    P: Matrix;
    q1_, q2_, q3_: double;
begin
  q1_ := MGetV(_Q, 0, 0);
  q2_ := MGetV(_Q, 1, 0);
  q3_ := MGetV(_Q, 2, 0);

  A1 := DHMat(q1_,0   ,l_1 ,0        );
  A2 := DHMat(q2_,0   ,l_2 ,rad(180));
  A3 := DHMat(0   ,q3_,0   ,0        );

  P := MMult(A1,A2);
  P := MMult(P,A3);

  result := P;
end;

// Inverse Kinematics
function IK3(XYZ: matrix): matrix;
var
  xc, yc, zc, c2: double;
  q1_, q2_, q3_: double;
begin
  xc := Mgetv(XYZ, 0, 0);
  yc := Mgetv(XYZ, 1, 0);
  zc := Mgetv(XYZ, 2, 0);

  c2 := (Power(xc,2) + power(yc,2) - power(l_1,2) - power(l_2,2))/(2*l_1*l_2);

  q2_ := ATan2(-sqrt(1-power(c2,2)),c2);
  q1_ := ATan2(yc,xc) - ATan2(l_2*sin(q2_), l_1 + l_2*cos(q2_));
  q3_ := -zc;

  result := Mzeros(3, 1);
  MSetV(result, 0, 0, q1_);
  MSetV(result, 1, 0, q2_);
  MSetV(result, 2, 0, q3_);
end;



// Main Control Cycle: this procedure is called periodicaly
// (default: 40 ms; can change it in Config > Control > Global > ScriptPeriod)
procedure Control;
var
  t: tcanvas;
  HTrans, XYZ, ORIENTATION, AuxJoints, ReqJoints, ReqValuesDK: matrix;
  U1, U2, U3, T1, T2, T3: double;
  PosPen: TPoint3D;
  M_mat, C_mat, B_mat, Phi_mat: Matrix;
  Aq_mat, R_mat, U_mat: Matrix;
  Err_mat: Matrix;

begin
  // Initialization
  // - joint position
  MSetV(Q_meas,0,0, GetAxisPos(0, iJ1Axis) );
  MSetV(Q_meas,1,0, GetAxisPos(0, iJ2Axis) );
  MSetV(Q_meas,2,0, GetAxisPos(0, iZAxis ) );
  // - joint speed
  MSetV(Qd1_meas,0,0, GetAxisSpeed(0, iJ1Axis) );
  MSetV(Qd1_meas,1,0, GetAxisSpeed(0, iJ2Axis) );
  MSetV(Qd1_meas,2,0, GetAxisSpeed(0, iZAxis ) );
  // - pen position
  PosPen := GetSolidPos(0, iPen);
  // - canvas
  t := GetSolidCanvas(0,0);
  t.brush.color := clwhite;
  t.textout(10,10, GetRCText(8, 2));



  // Set color of the pen (RGB)
  if RCButtonPressed(1, 2) then
    SetSensorColor(0, 0, round(GetRCValue(1, 3)), round(GetRCValue(1, 4)),
        round(GetRCValue(1, 5)));

  // Set position of the pen
  // - up
  if RCButtonPressed(3, 2) then begin
    SetJointsControllerState(true);
    SetAxisPosRef(0, iZAxis, GetRCValue(3, 3));
  end;
  // - down
  if RCButtonPressed(4, 2) then begin
    SetJointsControllerState(true);
    SetAxisPosRef(0, iZAxis, GetRCValue(4, 3));
  end;

  // Reset angle of joints J1,J2
  if RCButtonPressed(6, 2) then begin
    SetJointsControllerState(true);
    SetValuesJoints(MZeros(3,1));
  end;

  // Set voltage of motor
  // (default controller should be disabled:
  //    Scene > Tag articulations > Tag controller > active='0' > Rebuild Scene)
  // - joint 1
  if RCButtonPressed(12, 2) then begin
    U1 := GetRCValue(12, 3);
    SetJointsControllerState(false);
    SetAxisVoltageRef(0, iJ1Axis, U1);
  end;
  // - joint 2
  if RCButtonPressed(13, 2) then begin
    U2 := GetRCValue(13, 3);
    SetJointsControllerState(false);
    SetAxisVoltageRef(0, iJ2Axis, U2);
  end;
  // - joint 3
  if RCButtonPressed(14, 2) then begin
    U3 := GetRCValue(14, 3);
    SetJointsControllerState(false);
    SetAxisVoltageRef(0, iZAxis , U3);
  end;
```

```
// - reset voltages
if RCButtonPressed(12, 4) then begin
  U1 := 0;
  U2 := 0;
  U3 := 0;
  SetJointsControllerState(false);
  SetAxisVoltageRef(0, iJ1Axis, U1);
  SetAxisVoltageRef(0, iJ2Axis, U2);
  SetAxisVoltageRef(0, iZAxis , U3);
end;

// Set torque of motor
// (default controller should be disabled:
//    Scene > Tag articulations > Tag controller > active='0' > Rebuild Scene)
// - joint 1
if RCButtonPressed(16, 2) then begin
  T1 := GetRCValue(16, 3);
  SetJointsControllerState(false);
  SetAxisTorqueRef(0, iJ1Axis, T1);
end;
// - joint 2
if RCButtonPressed(17, 2) then begin
  T2 := GetRCValue(17, 3);
  SetJointsControllerState(false);
  SetAxisTorqueRef(0, iJ2Axis, T2);
end;
// - joint 3
if RCButtonPressed(18, 2) then begin
  T3 := GetRCValue(18, 3);
  SetJointsControllerState(false);
  SetAxisTorqueRef(0, iZAxis , T3);
end;
// - reset voltages
if RCButtonPressed(16, 4) then begin
  T1 := 0;
  T2 := 0;
  T3 := 0;
  SetJointsControllerState(false);
  SetAxisTorqueRef(0, iJ1Axis, T1);
  SetAxisTorqueRef(0, iJ2Axis, T2);
  SetAxisTorqueRef(0, iZAxis , T3);
end;



// Forward Kinematics
if RCButtonPressed(1, 9) then begin
  SetJointsControllerState(true);

  ReqValuesDK := Mzeros(3, 1);
  Msetv(ReqValuesDK, 0, 0, rad(GetRCValue(2, 9)));
  Msetv(ReqValuesDK, 1, 0, rad(GetRCValue(3, 9)));
  Msetv(ReqValuesDK, 2, 0, GetRCValue(4, 9));

  SetValuesJoints(ReqValuesDK);
  HTrans := DK3(ReqValuesDK);
  XYZ := Mzeros(3,1);

  MSetV(XYZ, 0, 0, Mgetv(HTrans, 0, 3));
```

```
    MSetV(XYZ, 1, 0, Mgetv(HTrans, 1, 3));
    MSetV(XYZ, 2, 0, Mgetv(HTrans, 2, 3));

    ORIENTATION := Meye(3);

    MSetV(ORIENTATION,0,0,Mgetv(HTrans,0,0)); MSetV(ORIENTATION,0,1,Mgetv(HTrans,0,1)
        ); MSetV(ORIENTATION,0,2,Mgetv(HTrans,0,2));
    MSetV(ORIENTATION,1,0,Mgetv(HTrans,1,0)); MSetV(ORIENTATION,1,1,Mgetv(HTrans,1,1)
        ); MSetV(ORIENTATION,1,2,Mgetv(HTrans,1,2));
    MSetV(ORIENTATION,2,0,Mgetv(HTrans,2,0)); MSetV(ORIENTATION,2,1,Mgetv(HTrans,2,1)
        ); MSetV(ORIENTATION,2,2,Mgetv(HTrans,2,2));

    MatrixToRangeF(2, 10, XYZ, '%.3f');
    MatrixToRangeF(6, 7, ORIENTATION, '%.3f');
  end;

  // Inverse Kinematics
  if RCButtonPressed(1,14) then begin
    SetJointsControllerState(true);

    XYZ := Mzeros(3,1);

    Msetv(XYZ, 0, 0, GetRCValue(2, 14));
    Msetv(XYZ, 1, 0, GetRCValue(3, 14));
    Msetv(XYZ, 2, 0, GetRCValue(4, 14));

    ReqJoints := IK3(XYZ);
    SetValuesJoints(ReqJoints);
    AuxJoints := Mzeros(3,1);

    MSetV(AuxJoints, 0, 0, Deg(Mgetv(ReqJoints, 0, 0)));
    MSetV(AuxJoints, 1, 0, Deg(Mgetv(ReqJoints, 1, 0)));
    MSetV(AuxJoints, 2, 0, Mgetv(ReqJoints, 2, 0));

    MatrixToRangeF(2, 15, AuxJoints, '%.3f');
  end;



  // Inverse Dynamics
  if RCButtonPressed(10,7) then begin
    if InvDynON then begin
      InvDynON := false;
      SetJointsControllerState(true);
    end else begin
      InvDynON := true;
      SetJointsControllerState(false);
      SetIDParameters;
      // Requested set point
      XYZ_desir := RangeToMatrix(12,9,3,1);
      Q_desir := IK3(XYZ_desir);
      Qd1_desir := Mzeros(3,1);
      Qd2_desir := Mzeros(3,1);
    end;
  end;
  if InvDynON AND RCButtonPressed(10,9) then begin
      SetIDParameters;
      // Requested set point
      XYZ_desir := RangeToMatrix(12,9,3,1);
      Q_desir := IK3(XYZ_desir);
      Qd1_desir := Mzeros(3,1);
      Qd2_desir := Mzeros(3,1);
  end;
  if InvDynON then begin
    M_mat := IDMMat(Q_meas);
    C_mat := IDCMat(Q_meas,Qd1_meas);
    B_mat := IDBMat;
    if GetRCValue(10, 2) = 0 then begin
      Phi_mat := IDPhiMatHorizontalSCARA(Q_meas);
    end else begin
      Phi_mat := IDPhiMatVerticalSCARA(Q_meas);
    end;
    MatrixToRangeF(24, 7, M_mat, '%.3f');
    MatrixToRangeF(28, 7, C_mat, '%.3f');
    MatrixToRangeF(32, 7, B_mat, '%.3f');
    MatrixToRangeF(24,10, Phi_mat, '%.3f');

    // Outter loop
    // - reference
    R_mat := Qd2_desir;
    R_mat := MAdd(R_mat,MMult(K1_mat,Qd1_desir));
    R_mat := MAdd(R_mat,MMult(K0_mat,Q_desir));
    // - output
    Aq_mat := R_mat;
    Aq_mat := MSub(Aq_mat,MMult(K0_mat,Q_meas));
    Aq_mat := MSub(Aq_mat,MMult(K1_mat,Qd1_meas));
    MatrixToRangeF(16, 8, R_mat, '%.3f');
    MatrixToRangeF(16, 9, Aq_mat, '%.3f');

    // Inner loop - torque computation
    U_mat := MZeros(3,1);
    U_mat := MAdd(U_mat,MMult(M_mat,Aq_mat   ));
    U_mat := MAdd(U_mat,MMult(C_mat,Qd1_meas));
    U_mat := MAdd(U_mat,MMult(B_mat,Qd1_meas));
    U_mat := MAdd(U_mat,Phi_mat);
    MatrixToRangeF(16,10, U_mat, '%.3f');

    SetTorque(U_mat);

    // Error computation
    Err_mat := MSub(Q_meas,Q_desir);
    SetRCValue(20, 8, format('%.3g',[Deg(MGetV(Err_mat,0,0))]));
    SetRCValue(21, 8, format('%.3g',[Deg(MGetV(Err_mat,1,0))]));
    SetRCValue(22, 8, format('%.3g',[MGetV(Err_mat,2,0)]));
  end;



  // SimTwo Sheet
  // - joint value (forward kinematics)
  SetRCValue(2, 8, format('%.3g',[Deg(GetAxisPos(0, iJ1Axis))]));
  SetRCValue(3, 8, format('%.3g',[Deg(GetAxisPos(0, iJ2Axis))]));
  SetRCValue(4, 8, format('%.3g',[GetAxisPos(0, iZAxis)]));

  // - pen position (horizontal/vertical SCARA)
  SetRCValue(2, 13, format('%.3g',[PosPen.x]));
  SetRCValue(12, 8, format('%.3g',[PosPen.x]));
  if GetRCValue(10, 2) = 0 then begin
```

```
    SetRCValue(3, 13, format('%.3g',[PosPen.y]));
    SetRCValue(4, 13, format('%.3g',[PosPen.z - 0.25]));
    SetRCValue(13, 8, format('%.3g',[PosPen.y]));
    SetRCValue(14, 8, format('%.3g',[PosPen.z - 0.25]));
  end else begin
    SetRCValue(3, 13, format('%.3g',[PosPen.z]));
    SetRCValue(4, 13, format('%.3g',[-PosPen.y - 0.25]));
    SetRCValue(13, 8, format('%.3g',[PosPen.z]));
    SetRCValue(14, 8, format('%.3g',[-PosPen.y - 0.25]));
  end;

  // - inverse dynamics activated
  if InvDynON then begin
    SetRCValue(10, 8, 'ON');
  end else begin
    SetRCValue(10, 8, 'OFF');
  end;
end;

// Initialization procedure: is called once when the script is started
procedure Initialize;
begin
  iJ1Axis := GetAxisIndex(0, 'Joint1', 0);
  iJ2Axis := GetAxisIndex(0, 'Joint2', 0);
  iZAxis  := GetAxisIndex(0, 'SlideZ', 0);
  iPen := GetSolidIndex(0, 'Pen');

  Q_meas := Mzeros(3,1);
  Qd1_meas := Mzeros(3,1);
  Q_desir := Mzeros(3,1);
  Qd1_desir := Mzeros(3,1);
  Qd2_desir := Mzeros(3,1);
  W_mat := Mzeros(3,1);
  K0_mat := Mzeros(3,3);
  K1_mat := Mzeros(3,3);
end;
```

D.2 Code given to the students

Listing 12. Code Provided to the Students for the Laboratory Work

```
// Constants
const
  NumJoints = 3;
  // Links
  l_1  = 0.6;   l_2  = 0.4;                  // length
  m_1  = 0.2;   m_2  = 0.2;   m_3  = 0.1;    // mass
  lc_1 = l_1/2; lc_2 = l_2/2;                // length to center of mass
  lx_1 = l_1;   lx_2 = l_2;   lx_3 = 0.02;   // dimensions
  ly_1 = 0.04;  ly_2 = 0.04;  ly_3 = 0.02;   //   (relative to frame oci_xci_yci)
  lz_1 = 0.04;  lz_2 = 0.04;  lz_3 = 0.05;
  rho_1 = m_1/(lx_1*ly_1*lz_1);              // mass density
  rho_2 = m_2/(lx_2*ly_2*lz_2);
  rho_3 = m_3/(lx_3*ly_3*lz_3);
  // Motors
  n_1  = 1;     n_2  = 1;     n_3  = 1;      // gear reduction ratio
  km_1 = 0.56;  km_2 = 0.56;  km_3 = 0.56;   // torque constant
  jm_1 = 0;     jm_2 = 0;     jm_3 = 0;      // inertia
  bm_1 = 0.10;  bm_2 = 0.10;  bm_3 = 0.10;  // viscous constant
  ri_1 = 0.12;  ri_2 = 0.12;  ri_3 = 0.12;  // internal resistance
  // Gravity
  //gacc = 9.80665;  // gravitational acceleration (m/s^2)
  gacc = 9.8;

// Global Variables
var iZAxis, iJ1Axis, iJ2Axis, iPen: integer;
    Q_meas, Qd1_meas, W_mat, K0_mat, K1_mat: Matrix;
    Q_desir, Qd1_desir, Qd2_desir, XYZ_desir: Matrix;
    InvDynON: boolean;



// Utilitary functions
// - set reference of the joints (J1,J2: revolute; J3: prismatic)
procedure SetValuesJoints(Values: matrix);
begin
  // Controller mode: position control
  SetMotorControllerMode(0, iJ1Axis, 'pidposition');
  SetMotorControllerMode(0, iJ2Axis, 'pidposition');
  SetMotorControllerMode(0, iZAxis , 'pidposition');
  // Controller parameters:          ki  kp  kp  kf
  SetMotorControllerPars(0, iJ1Axis,  0, 25,  3,  0);
  SetMotorControllerPars(0, iJ2Axis,  0, 50,  4,  0);
  SetMotorControllerPars(0, iZAxis ,  0,  1,  5,  0);
  // Set position reference
  SetAxisPosRef(0, iJ1Axis, Mgetv(Values, 0, 0));
  SetAxisPosRef(0, iJ2Axis, Mgetv(Values, 1, 0));
  SetAxisPosRef(0, iZAxis , Mgetv(Values, 2, 0));
end;

// - compute Denavit-Hartenberg (DH) matrix
function DHMat(theta, d, a, alpha: double): Matrix;
var ct, st, ca, sa: double;
    R: Matrix;
begin
  ct := cos(theta);
  st := sin(theta);
  ca := cos(alpha);
  sa := sin(alpha);

  R := Meye(4);
  MSetV(R,0,0,ct); MSetV(R,0,1,-st*ca); MSetV(R,0,2, st*sa); MSetV(R,0,3,a*ct);
  MSetV(R,1,0,st); MSetV(R,1,1, ct*ca); MSetV(R,1,2,-ct*sa); MSetV(R,1,3,a*st);
  MSetV(R,2,0, 0); MSetV(R,2,1, sa   ); MSetV(R,2,2, ca   ); MSetV(R,2,3,d   );
  result := R;
end;

// - rotation matrix (z axis)
function RotZ(theta: double): Matrix;
var ct, st: double;
    R: Matrix;
begin
  ct := cos(theta);
  st := sin(theta);
  R := Mzeros(3,3);

  MSetV(R,0,0,ct); MSetV(R,0,1,-st);
  MSetV(R,1,0,st); MSetV(R,1,1, ct);
                                               MSetV(R,2,2,1);

  result := R;
end;

// - torque mode (controllers deactivated)
procedure SetJointsControllerState(activate: boolean);
begin
  SetMotorControllerState(0, iJ1Axis, activate);
  SetMotorControllerState(0, iJ2Axis, activate);
  SetMotorControllerState(0, iZAxis , activate);
end;

// - set torque value for the joints
procedure SetTorque(Torque: matrix);
begin
  SetAxisTorqueRef(0, iJ1Axis, Mgetv(Torque, 0, 0));
  SetAxisTorqueRef(0, iJ2Axis, Mgetv(Torque, 1, 0));
  SetAxisTorqueRef(0, iZAxis , Mgetv(Torque, 2, 0));
end;



// Inverse Dynamics (ID)
// - set inverse dynamics parameters
procedure SetIDParameters;
begin
  // wn
  W_mat := RangeToMatrix(12,10,3,1);
  // K0 = wn ^ 2
  K0_mat := Mzeros(3,3);
  MSetV(K0_mat,0,0,MGetV(W_mat,0,0)*MGetV(W_mat,0,0));
  MSetV(K0_mat,1,1,MGetV(W_mat,1,0)*MGetV(W_mat,1,0));
  MSetV(K0_mat,2,2,MGetV(W_mat,2,0)*MGetV(W_mat,2,0));
  // K1 = 2 * wn
  K1_mat := Mzeros(3,3);
  MSetV(K1_mat,0,0,2*MGetV(W_mat,0,0));
  MSetV(K1_mat,1,1,2*MGetV(W_mat,1,0));
  MSetV(K1_mat,2,2,2*MGetV(W_mat,2,0));
end;

// - inertia matrix (D)
function IDDMat(_Q: Matrix): Matrix;
var q1_, q2_, q3_: double;
    cq1, cq2, cq1pq2, sq1, sq2, sq1pq2: double;
    Jvc_1, Jvc_2, Jvc_3: Matrix;
    Jwc_1, Jwc_2, Jwc_3: Matrix;
    Rc_1, Rc_2, Rc_3: Matrix;
    I_1, I_2, I_3: Matrix;
begin
  q1_ := MGetV(_Q, 0, 0);
  q2_ := MGetV(_Q, 1, 0);
  q3_ := MGetV(_Q, 2, 0);
  cq1 := cos(q1_);
  sq1 := sin(q1_);
  cq2 := cos(q2_);
  sq2 := sin(q2_);
  cq1pq2 := cos(q1_+q2_);
```

```
sq1pq2 := sin(q1_+q2_);

// Matrices Jvc_i (linear speed jacobian relative to oci_xci_yci_zci)
Jvc_1 := Mzeros(3,3);
Msetv(Jvc_1,0,0,-lc_1*sq1);
Msetv(Jvc_1,1,0, lc_1*cq1);

Jvc_2 := Mzeros(3,3);
Msetv(Jvc_2,0,0,-l_1*sq1-lc_2*sq1pq2); Msetv(Jvc_2,0,1,-lc_2*sq1pq2);
Msetv(Jvc_2,1,0, l_1*cq1+lc_2*cq1pq2); Msetv(Jvc_2,1,1, lc_2*cq1pq2);

Jvc_3 := Mzeros(3,3);
Msetv(Jvc_3,0,0,-l_1*sq1-l_2*sq1pq2); Msetv(Jvc_3,0,1,-l_2*sq1pq2);
Msetv(Jvc_3,1,0, l_1*cq1+l_2*cq1pq2); Msetv(Jvc_3,1,1, l_2*cq1pq2);
                                                                  Msetv(Jvc_3
                                                                     ,2,2,-1);

// Matrices Jwc_i (angular speed jacobian relative to oci_xci_yci_zci)
Jwc_1 := Mzeros(3,3);
Msetv(Jwc_1,2,0,1);

Jwc_2 := Mzeros(3,3);
Msetv(Jwc_2,2,0,1); Msetv(Jwc_2,2,1,1);

Jwc_3 := Jwc_2;

// Matrices Rc_i (orientation matrices relative to oci_xci_yci_zci)
Rc_1 := RotZ(q1_);

Rc_2 := Mzeros(3,3);
MSetV(Rc_2,0,0,cq1pq2); MSetV(Rc_2,0,1, sq1pq2);
MSetV(Rc_2,1,0,sq1pq2); MSetV(Rc_2,1,1,-cq1pq2);
                                                 MSetV(Rc_2,2,2,-1);

Rc_3 := Rc_2;

// Matrices I_i (inertia tensors relative to oci_xci_yci_zci)
I_1 := Mzeros(3,3);
MSetV(I_1,0,0,rho_1*lx_1*ly_1*lz_1*(ly_1*ly_1+lz_1*lz_1)/12);
MSetV(I_1,1,1,rho_1*lx_1*ly_1*lz_1*(lx_1*lx_1+lz_1*lz_1)/12);
MSetV(I_1,2,2,rho_1*lx_1*ly_1*lz_1*(ly_1*ly_1+lx_1*lx_1)/12);

I_2 := Mzeros(3,3);
MSetV(I_2,0,0,rho_2*lx_2*ly_2*lz_2*(ly_2*ly_2+lz_2*lz_2)/12);
MSetV(I_2,1,1,rho_2*lx_2*ly_2*lz_2*(lx_2*lx_2+lz_2*lz_2)/12);
MSetV(I_2,2,2,rho_2*lx_2*ly_2*lz_2*(ly_2*ly_2+lx_2*lx_2)/12);

I_3 := Mzeros(3,3);
MSetV(I_3,0,0,rho_3*lx_3*ly_3*lz_3*(ly_3*ly_3+lz_3*lz_3)/12);
MSetV(I_3,1,1,rho_3*lx_3*ly_3*lz_3*(lx_3*lx_3+lz_3*lz_3)/12);
MSetV(I_3,2,2,rho_3*lx_3*ly_3*lz_3*(ly_3*ly_3+lx_3*lx_3)/12);

// Inertia Matrix D(Q) (Q = [q1 q2 q3])
result := MMultReal( MMult(MTran(Jvc_1),Jvc_1) , m_1 );
result := MAdd( result , MMultReal( MMult(MTran(Jvc_2),Jvc_2) , m_2 ) );
result := MAdd( result , MMultReal( MMult(MTran(Jvc_3),Jvc_3) , m_3 ) );
result := MAdd( result , MMult(MMult(MMult(MTran(Jwc_1),Rc_1),I_1),MMult(MTran(Rc_1
    ),Jwc_1)) );
```

```
  result := MAdd( result , MMult(MMult(MMult(MTran(Jwc_2),Rc_2),I_2),MMult(MTran(Rc_2
      ),Jwc_2)) );
  result := MAdd( result , MMult(MMult(MMult(MTran(Jwc_3),Rc_3),I_3),MMult(MTran(Rc_3
      ),Jwc_3)) );
end;

// − Diagonal matrix r^2 * Jm
function IDJmMat: Matrix;
begin
  result := Mzeros(3,3);
  MSetV(result,0,0,n_1*n_1*jm_1);
  MSetV(result,1,1,n_2*n_2*jm_2);
  MSetV(result,2,2,n_3*n_3*jm_3);
end;

// − M matrix (D + J)
function IDMMat(_Q: Matrix): Matrix;
begin
  result := MAdd( IDDMat(_Q) , IDJmMat );
end;

// − C matrix (C)
function IDCMat(_Q,_Qd1: Matrix): Matrix;
var q1_, q2_, q3_: double;
    q1d1_, q2d1_, q3d1_: double;
    cq1, cq2, cq1pq2, sq1, sq2, sq1pq2: double;
    c211, c112, c121, c221: double;
begin
  q1_ := MGetV(_Q, 0, 0);
  q2_ := MGetV(_Q, 1, 0);
  q3_ := MGetV(_Q, 2, 0);
  q1d1_ := MGetV(_Qd1, 0, 0);
  q2d1_ := MGetV(_Qd1, 1, 0);
  q3d1_ := MGetV(_Qd1, 2, 0);
  cq1 := cos(q1_);
  sq1 := sin(q1_);
  cq2 := cos(q2_);
  sq2 := sin(q2_);
  cq1pq2 := cos(q1_+q2_);
  sq1pq2 := sin(q1_+q2_);

  // Christoffel symbols
  c211 := m_3 * cq1pq2 * l_2  * ( l_1 * sq1 + l_2  * sq1pq2 ) −
          m_2 * sq1pq2 * lc_2 * ( l_1 * cq1 + lc_2 * cq1pq2 ) −
          m_3 * sq1pq2 * l_2  * ( l_1 * cq1 + l_2  * cq1pq2 ) +
          m_2 * cq1pq2 * lc_2 * ( l_1 * sq1 + lc_2 * sq1pq2 );
  c121 := c211;
  c221 := c211;
  c112 := m_3 * sq1pq2 * l_2  * ( l_1 * cq1 + l_2  * cq1pq2 ) +
          m_2 * sq1pq2 * lc_2 * ( l_1 * cq1 + lc_2 * cq1pq2 ) −
          m_3 * cq1pq2 * l_2  * ( l_1 * sq1 + l_2  * sq1pq2 ) −
          m_2 * cq1pq2 * lc_2 * ( l_1 * sq1 + lc_2 * sq1pq2 );

  // Matrix C
  result := Mzeros(3,3);
  MSetV(result,0,0,c211*q2d1_); MSetV(result,0,1,c121*q1d1_+c221*q2d1_);
  MSetV(result,1,0,c112*q1d1_);
end;
```

```
// - Diagonal matrix B
function IDBMat: Matrix;
begin
  result := Mzeros(3,3);
  MSetV(result,0,0,bm_1+km_1*km_1/ri_1);
  MSetV(result,1,1,bm_2+km_2*km_2/ri_2);
  MSetV(result,2,2,bm_3+km_3*km_3/ri_3);
end;

// - gravity influence (Phi matrix)
function IDPhiMatHorizontalSCARA(_Q: Matrix): Matrix;
begin
  result := Mzeros(3,1);
  MSetV(result,2,0,-gacc*m_3);
end;

function IDPhiMatVerticalSCARA(_Q: Matrix): Matrix;
var q1_, q2_, q3_: double;
    cq1, cq2, cq1pq2, sq1, sq2, sq1pq2: double;
begin
  q1_ := MGetV(_Q, 0, 0);
  q2_ := MGetV(_Q, 1, 0);
  q3_ := MGetV(_Q, 2, 0);
  cq1 := cos(q1_);
  cq1pq2 := cos(q1_+q2_);

  result := Mzeros(3,1);
  MSetV(result,0,0,gacc*lc_1*cq1*m_1 +
                   gacc*l_1 *cq1*m_2 + gacc*lc_2*cq1pq2*m_2+
                   gacc*l_1 *cq1*m_3 + gacc*l_2 *cq1pq2*m_3);
  MSetV(result,1,0,gacc*lc_2*cq1pq2*m_2 +
                   gacc*l_2 *cq1pq2*m_3);
  MSetV(result,2,0,0);
end;



// Forward Kinematics
function DK3(_Q: matrix): matrix;
var A1, A2, A3: Matrix;
    P: Matrix;
    q1_, q2_, q3_: double;
begin
  q1_ := MGetV(_Q, 0, 0);
  q2_ := MGetV(_Q, 1, 0);
  q3_ := MGetV(_Q, 2, 0);

  A1 := DHMat(q1_,0   ,l_1,0       );
  A2 := DHMat(q2_,0   ,l_2,rad(180));
  A3 := DHMat(0  ,q3_,0   ,0       );

  P := MMult(A1,A2);
  P := MMult(P,A3);

  result := P;
end;

// Inverse Kinematics
function IK3(XYZ: matrix): matrix;
```

```
var
  xc, yc, zc, c2: double;
  q1_, q2_, q3_: double;
begin
  xc := Mgetv(XYZ, 0, 0);
  yc := Mgetv(XYZ, 1, 0);
  zc := Mgetv(XYZ, 2, 0);

  c2 := (Power(xc,2) + power(yc,2) - power(l_1,2) - power(l_2,2))/(2*l_1*l_2);

  q2_ := ATan2(-sqrt(1-power(c2,2)),c2);
  q1_ := ATan2(yc,xc) - ATan2(l_2*sin(q2_), l_1 + l_2*cos(q2_));
  q3_ := -zc;

  result := Mzeros(3, 1);
  MSetV(result, 0, 0, q1_);
  MSetV(result, 1, 0, q2_);
  MSetV(result, 2, 0, q3_);
end;



// Main Control Cycle: this procedure is called periodicaly
// (default: 40 ms; can change it in Config > Control > Global > ScriptPeriod)
procedure Control;
var
  t: tcanvas;
  HTrans, XYZ, ORIENTATION, AuxJoints, ReqJoints, ReqValuesDK: matrix;
  U1, U2, U3, T1, T2, T3: double;
  PosPen: TPoint3D;
  M_mat, C_mat, B_mat, Phi_mat: Matrix;
  Aq_mat, R_mat, U_mat: Matrix;
  Err_mat: Matrix;

begin
  // Initialization
  // - joint position
  MSetV(Q_meas,0,0, GetAxisPos(0, iJ1Axis) );
  MSetV(Q_meas,1,0, GetAxisPos(0, iJ2Axis) );
  MSetV(Q_meas,2,0, GetAxisPos(0, iZAxis ) );
  // - joint speed
  MSetV(Qd1_meas,0,0, GetAxisSpeed(0, iJ1Axis) );
  MSetV(Qd1_meas,1,0, GetAxisSpeed(0, iJ2Axis) );
  MSetV(Qd1_meas,2,0, GetAxisSpeed(0, iZAxis ) );
  // - pen position
  PosPen := GetSolidPos(0, iPen);
  // - canvas
  t := GetSolidCanvas(0,0);
  t.brush.color := clwhite;
  t.textout(10,10, GetRCText(8, 2));



  // Set color of the pen (RGB)
  if RCButtonPressed(1, 2) then
    SetSensorColor(0, 0, round(GetRCValue(1, 3)), round(GetRCValue(1, 4)),
        round(GetRCValue(1, 5)));

  // Set position of the pen
  // - up
  if RCButtonPressed(3, 2) then begin
    SetJointsControllerState(true);
    SetAxisPosRef(0, iZAxis, GetRCValue(3, 3));
  end;
  // - down
  if RCButtonPressed(4, 2) then begin
    SetJointsControllerState(true);
    SetAxisPosRef(0, iZAxis, GetRCValue(4, 3));
  end;

  // Reset angle of joints J1,J2
  if RCButtonPressed(6, 2) then begin
    SetJointsControllerState(true);
    SetValuesJoints(MZeros(3,1));
  end;

  // Set voltage of motor
  // (default controller should be disabled:
  //    Scene > Tag articulations > Tag controller > active='0' > Rebuild Scene)
  // - joint 1
  if RCButtonPressed(12, 2) then begin
    U1 := GetRCValue(12, 3);
    SetJointsControllerState(false);
    SetAxisVoltageRef(0, iJ1Axis, U1);
  end;
  // - joint 2
  if RCButtonPressed(13, 2) then begin
    U2 := GetRCValue(13, 3);
    SetJointsControllerState(false);
    SetAxisVoltageRef(0, iJ2Axis, U2);
  end;
  // - joint 3
  if RCButtonPressed(14, 2) then begin
    U3 := GetRCValue(14, 3);
    SetJointsControllerState(false);
    SetAxisVoltageRef(0, iZAxis , U3);
  end;
  // - reset voltages
  if RCButtonPressed(12, 4) then begin
    U1 := 0;
    U2 := 0;
    U3 := 0;
    SetJointsControllerState(false);
    SetAxisVoltageRef(0, iJ1Axis, U1);
    SetAxisVoltageRef(0, iJ2Axis, U2);
    SetAxisVoltageRef(0, iZAxis , U3);
  end;

  // Set torque of motor
  // (default controller should be disabled:
  //    Scene > Tag articulations > Tag controller > active='0' > Rebuild Scene)
  // - joint 1
  if RCButtonPressed(16, 2) then begin
    T1 := GetRCValue(16, 3);
    SetJointsControllerState(false);
    SetAxisTorqueRef(0, iJ1Axis, T1);
  end;
  // - joint 2
```

```
  if RCButtonPressed(17, 2) then begin
    T2 := GetRCValue(17, 3);
    SetJointsControllerState(false);
    SetAxisTorqueRef(0, iJ2Axis, T2);
  end;
  // - joint 3
  if RCButtonPressed(18, 2) then begin
    T3 := GetRCValue(18, 3);
    SetJointsControllerState(false);
    SetAxisTorqueRef(0, iZAxis , T3);
  end;
  // - reset voltages
  if RCButtonPressed(16, 4) then begin
    T1 := 0;
    T2 := 0;
    T3 := 0;
    SetJointsControllerState(false);
    SetAxisTorqueRef(0, iJ1Axis, T1);
    SetAxisTorqueRef(0, iJ2Axis, T2);
    SetAxisTorqueRef(0, iZAxis , T3);
  end;



  // Forward Kinematics
  if RCButtonPressed(1, 9) then begin
    SetJointsControllerState(true);

    ReqValuesDK := Mzeros(3, 1);
    Msetv(ReqValuesDK, 0, 0, rad(GetRCValue(2, 9)));
    Msetv(ReqValuesDK, 1, 0, rad(GetRCValue(3, 9)));
    Msetv(ReqValuesDK, 2, 0, GetRCValue(4, 9));

    SetValuesJoints(ReqValuesDK);
    HTrans := DK3(ReqValuesDK);
    XYZ := Mzeros(3,1);

    MSetV(XYZ, 0, 0, Mgetv(HTrans, 0, 3));
    MSetV(XYZ, 1, 0, Mgetv(HTrans, 1, 3));
    MSetV(XYZ, 2, 0, Mgetv(HTrans, 2, 3));

    ORIENTATION := Meye(3);

    MSetV(ORIENTATION,0,0,Mgetv(HTrans,0,0)); MSetV(ORIENTATION,0,1,Mgetv(HTrans,0,1)
        ); MSetV(ORIENTATION,0,2,Mgetv(HTrans,0,2));
    MSetV(ORIENTATION,1,0,Mgetv(HTrans,1,0)); MSetV(ORIENTATION,1,1,Mgetv(HTrans,1,1)
        ); MSetV(ORIENTATION,1,2,Mgetv(HTrans,1,2));
    MSetV(ORIENTATION,2,0,Mgetv(HTrans,2,0)); MSetV(ORIENTATION,2,1,Mgetv(HTrans,2,1)
        ); MSetV(ORIENTATION,2,2,Mgetv(HTrans,2,2));

    MatrixToRangeF(2, 10, XYZ, '%.3f');
    MatrixToRangeF(6, 7, ORIENTATION, '%.3f');
  end;

  // Inverse Kinematics
  if RCButtonPressed(1,14) then begin
    SetJointsControllerState(true);

    XYZ := Mzeros(3,1);
    Msetv(XYZ, 0, 0, GetRCValue(2, 14));
    Msetv(XYZ, 1, 0, GetRCValue(3, 14));
    Msetv(XYZ, 2, 0, GetRCValue(4, 14));

    ReqJoints := IK3(XYZ);
    SetValuesJoints(ReqJoints);
    AuxJoints := Mzeros(3,1);

    MSetV(AuxJoints, 0, 0, Deg(Mgetv(ReqJoints, 0, 0)));
    MSetV(AuxJoints, 1, 0, Deg(Mgetv(ReqJoints, 1, 0)));
    MSetV(AuxJoints, 2, 0, Mgetv(ReqJoints, 2, 0));

    MatrixToRangeF(2, 15, AuxJoints, '%.3f');
  end;



  // Inverse Dynamics
  if RCButtonPressed(10,7) then begin
    if InvDynON then begin
      InvDynON := false;
      SetJointsControllerState(true);
    end else begin
      InvDynON := true;
      SetJointsControllerState(false);
      SetIDParameters;
      // Requested set point
      XYZ_desir := RangeToMatrix(12,9,3,1);
      Q_desir := IK3(XYZ_desir);
      Qd1_desir := Mzeros(3,1);
      Qd2_desir := Mzeros(3,1);
    end;
  end;
  if InvDynON AND RCButtonPressed(10,9) then begin
      SetIDParameters;
      // Requested set point
      XYZ_desir := RangeToMatrix(12,9,3,1);
      Q_desir := IK3(XYZ_desir);
      Qd1_desir := Mzeros(3,1);
      Qd2_desir := Mzeros(3,1);
  end;
  if InvDynON then begin
    M_mat := IDMMat(Q_meas);
    C_mat := IDCMat(Q_meas,Qd1_meas);
    B_mat := IDBMat;
    if GetRCValue(10, 2) = 0 then begin
      Phi_mat := IDPhiMatHorizontalSCARA(Q_meas);
    end else begin
      Phi_mat := IDPhiMatVerticalSCARA(Q_meas);
    end;
    MatrixToRangeF(24, 7, M_mat, '%.3f');
    MatrixToRangeF(28, 7, C_mat, '%.3f');
    MatrixToRangeF(32, 7, B_mat, '%.3f');
    MatrixToRangeF(24,10, Phi_mat, '%.3f');

    // Outter loop
    // - reference
    R_mat := Qd2_desir;
    R_mat := MAdd(R_mat,MMult(K1_mat,Qd1_desir));
    R_mat := MAdd(R_mat,MMult(K0_mat,Q_desir));
    // - output
    Aq_mat := R_mat;
    Aq_mat := MSub(Aq_mat,MMult(K0_mat,Q_meas));
    Aq_mat := MSub(Aq_mat,MMult(K1_mat,Qd1_meas));
    MatrixToRangeF(16, 8, R_mat, '%.3f');
    MatrixToRangeF(16, 9, Aq_mat, '%.3f');

    // Inner loop - torque computation
    U_mat := MZeros(3,1);
    U_mat := MAdd(U_mat,MMult(M_mat,Aq_mat   ));
    U_mat := MAdd(U_mat,MMult(C_mat,Qd1_meas));
    U_mat := MAdd(U_mat,MMult(B_mat,Qd1_meas));
    U_mat := MAdd(U_mat,Phi_mat);
    MatrixToRangeF(16,10, U_mat, '%.3f');

    SetTorque(U_mat);

    // Error computation
    Err_mat := MSub(Q_meas,Q_desir);
    SetRCValue(20, 8, format('%.3g',[Deg(MGetV(Err_mat,0,0))]));
    SetRCValue(21, 8, format('%.3g',[Deg(MGetV(Err_mat,1,0))]));
    SetRCValue(22, 8, format('%.3g',[MGetV(Err_mat,2,0)]));
  end;



  // SimTwo Sheet
  // - joint value (forward kinematics)
  SetRCValue(2, 8, format('%.3g',[Deg(GetAxisPos(0, iJ1Axis))]));
  SetRCValue(3, 8, format('%.3g',[Deg(GetAxisPos(0, iJ2Axis))]));
  SetRCValue(4, 8, format('%.3g',[GetAxisPos(0, iZAxis)]));

  // - pen position (horizontal/vertical SCARA)
  SetRCValue(2, 13, format('%.3g',[PosPen.x]));
  SetRCValue(12, 8, format('%.3g',[PosPen.x]));
  if GetRCValue(10, 2) = 0 then begin
    SetRCValue(3, 13, format('%.3g',[PosPen.y]));
    SetRCValue(4, 13, format('%.3g',[PosPen.z - 0.25]));
    SetRCValue(13, 8, format('%.3g',[PosPen.y]));
    SetRCValue(14, 8, format('%.3g',[PosPen.z - 0.25]));
  end else begin
    SetRCValue(3, 13, format('%.3g',[PosPen.z]));
    SetRCValue(4, 13, format('%.3g',[-PosPen.y - 0.25]));
    SetRCValue(13, 8, format('%.3g',[PosPen.z]));
    SetRCValue(14, 8, format('%.3g',[-PosPen.y - 0.25]));
  end;

  // - inverse dynamics activated
  if InvDynON then begin
    SetRCValue(10, 8, 'ON');
  end else begin
    SetRCValue(10, 8, 'OFF');
  end;
end;

// Initialization procedure: is called once when the script is started
procedure Initialize;
```

```
begin
  iJ1Axis := GetAxisIndex(0, 'Joint1', 0);
  iJ2Axis := GetAxisIndex(0, 'Joint2', 0);
  iZAxis  := GetAxisIndex(0, 'SlideZ', 0);
  iPen := GetSolidIndex(0, 'Pen');

  Q_meas := Mzeros(3,1);
  Qd1_meas := Mzeros(3,1);
  Q_desir := Mzeros(3,1);
  Qd1_desir := Mzeros(3,1);
  Qd2_desir := Mzeros(3,1);
  W_mat := Mzeros(3,1);
  K0_mat := Mzeros(3,3);
  K1_mat := Mzeros(3,3);
end;
```