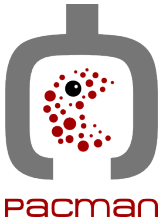


FP7-IST-60918
1 March 2013 (36months)

Milestone 4.2: Reactive haptic exploration control strategies ready for integration with planning

Manuel Bonilla, Carlos Rosales, Federico Spinelli, Marco Gabiccini

Centro di Ricerca "E. Piaggio", Università di Pisa
⟨manuel.bonilla@centropiaggio.unipi.t⟩

Due date of deliverable:	2015-02-28
Actual submission date:	2014-02-28
Lead partner:	Università di Pisa
Revision:	draft
Dissemination level:	PU

Here comes the abstract

1 Introduction

In recent years, there has been growing interest in the development of a standardized framework for robotic implementations, where the robots interdisciplinary code and software could be easy to write and accessible to everyone. This is the philosophy of the *Robot Operating System* (**ROS**), a flexible framework which provides all the instruments to develop and build robust and robot-independent applications, allowing researchers from different fields of robotics to share each others works with the great result of spreading the know-how.

1.1 Project Goals

The aim of this project focused on the development of a ROS-enabled software environment for controlling the manipulator KUKA LWR IV. After a first phase of studying and analyzing some of the novel approaches of manipulators control, the default control strategies provided by the manufacturer has been extended, implementing a new set of controllers. The development of the controllers, including tuning and debugging, has been realized with the support of *Gazebo*, a high performance realistic simulator which, interacting with ROS, gives the opportunity to test the efficiency of user-implemented control algorithms. Finally, the last phase consisted in the experimentation of the above mentioned control strategies on the physical robot, arising a variety of practical issues which don't come out in simulation, that are explained in details in the related section.

2 The Robot

2.1 KUKA LWR IV

The robot used for the project experiments, the *KUKA Lightweight Robot IV* [**?**], is a 7-axis industrial manipulator which finds a wide employment in the field of the robotics research due to its flexibility and modularity, including a payload capacity of 7 Kg. In addition, the seventh axis makes the robot redundant, which mean that, for a given position and orientation of the end effector, there exist multiple configurations in the joints space. Each joint is equipped with a position and a torque sensor, allowing the robot to be operated with position, velocity and torque control.

Figure 1: KUKA Lightweight Robot IV arm

The communication between the robot and an external computer is handled by the *Fast Research Interface* (FRI), a real-time proprietary software interface able to read and write system variables of the robot and execute user-programs.

2.2 Built-in Control Strategies

The robot provides three different control strategies as default setting, which are described below.

Position Controller

The position controller takes as command the desired joints set-point position, ensuring regulation through the use of an internal *PID*.

Cartesian Stiffness Controller

The Cartesian stiffness controller is characterized by the following control law:

$$\tau_{cmd} = J^T(k_c(x_{FRI} - x_{msr}) + F_{FRI}) + D(d_c) + f_{dynamics}(q, \dot{q}, \ddot{q}) \qquad (1)$$

where:

- x_{FRI}: cartesian commanded set-point position;
- k_c: cartesian stiffness;
- $D(d_c)$: damping term;
- $f_{dynamics}(q, \dot{q}, \ddot{q})$: the dynamic model;
- F_{FRI}: the cartesian force.

Axis-specific Stiffness Controller

The axis-specific stiffness controller is similar to the cartesian stiffness controller, but it takes commands in joints space:

$$\tau_{cmd} = (k_j(q_{FRI} - q_{msr}) + \tau_{FRI}) + D(d_j) + f_{dynamics}(q, \dot{q}, \ddot{q}) \qquad (2)$$

3 Software Environment

3.1 Robot Software Development

The middleware software used to implement the robot software is *Robot Operating System* (mostly known as ROS) [**?**]. Actually, ROS is not just an operating system, it's a complete ecosystem providing libraries, hardware abstraction, device drivers, visualizers, message-passing and package management.

3.1.1 How ROS Works

Basically, the ROS system is based on nodes, topics, messages and services, allowing the developer to work on a simple and modular code structure.

Nodes *Nodes* are processes that perform a specific task. They can use *topics* to communicate between each other and use *services* to handle external operations, e.g. a remote procedure call.

Topics *Topics* are the transport system used to delivery *messages* between *nodes*. They are based on the *publish/subscribe* semantics, so that a node sends out a message by publishing it on a specific topic. On the other hand, a node will subscribe to the appropriate topic to which it is interested.

Messages *Messages* are simply structures of data of typed fields. Just like *C*, they support both standard and user-defined types, as well as nested and complex structures.

Services The *publish/subscribe* model, even being a flexible and strong communication system, can't afford the *request/reply* interactions, which is managed by the *services*. An example of usage could be a situation in which a node has executed a task and needs a feedback or an acknowledge for it.

3.2 Simulation Environment

During the process of building robot applications, it's clearly not convenient to perform code debug and tests on the real robot. Therefore, the developer needs a simulated environment in which the behaviour of the real robot can be emulated via software in an as precise as possible way, replicating model dynamics, joints and links properties, loads handling and so on. The simulation software used for this purpose is *Gazebo* [**?**], which offers a strict interaction with ROS, providing a realistic rendering of the robot along with its real time ROS-controlled behavior.

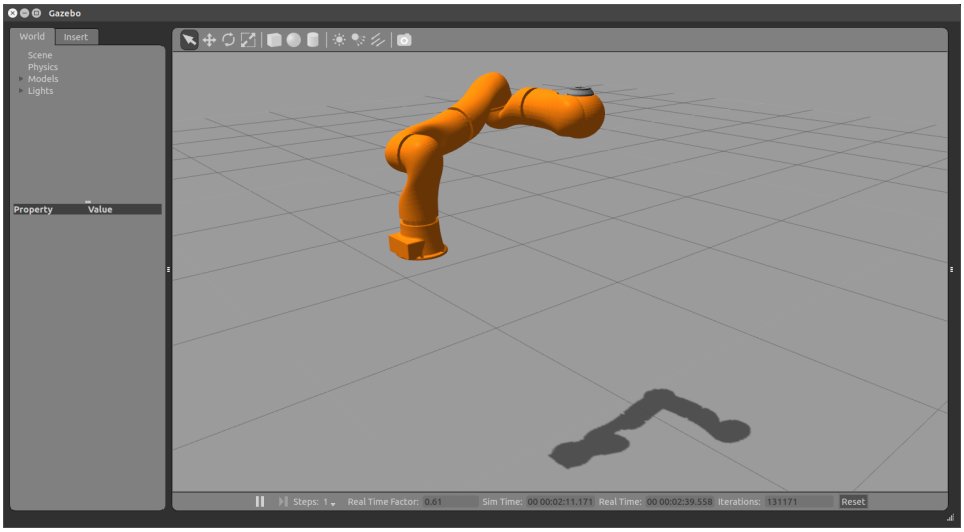


Figure 2: Gazebo Simulator with KUKA LWR IV robot

3.2.1 How Gazebo Works

To render the robot, the developer has to build a XML file describing all elements of the robot following a tree structure. This file is easily built using the robot focused DSL (*Domain Specific Language*) named URDF (*Unified Robot Description Format*). Gazebo has the task of parsing the URDF model and, through a publish/subscribe system with the ROS nodes and topics, it can apply the user-defined algorithms and control laws to the simulated robot.

3.3 Additional Tools

3.3.1 RViz

Another software to mention, used mainly in simulation but also during real experiments, is *RViz*, which is a 3D visualization tool included in ROS. It has been useful not only to monitor the current robot configuration (joint states, pose, orientation) but it's also capable of plotting executed trajectories.

3.3.2 Kinematics and Dynamics Library (KDL)

A great and valid support in coding has been given by the *Kinematics and Dynamics Library* (KDL), a software framework providing class libraries for robot modeling, as well as computation of forward and inverse chain kinematics and dynamics, motion specification and so on [**?**].

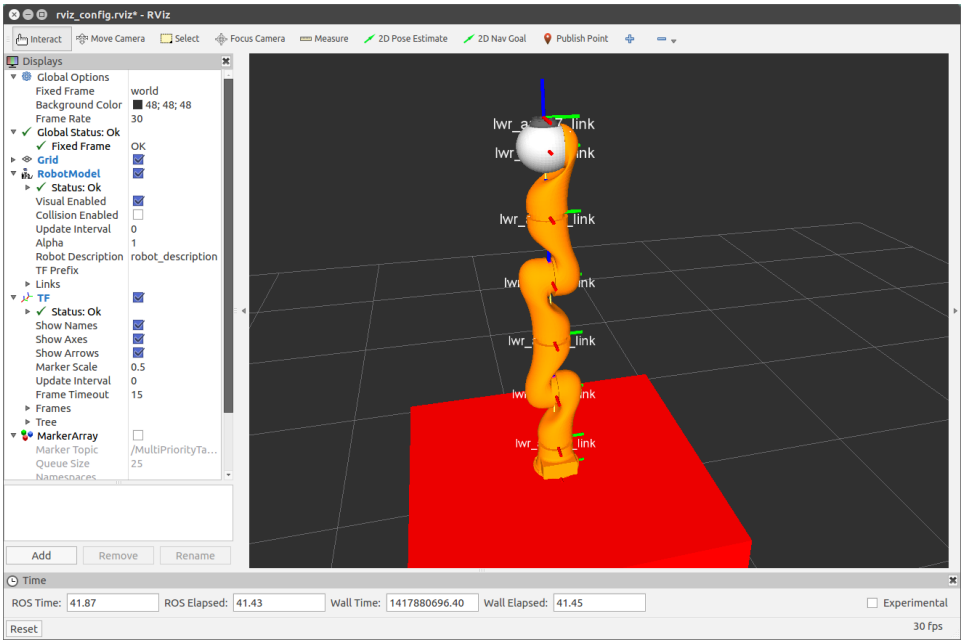


Figure 3: Visualization of the KUKA LWR IV robot in RViz

4 Implemented Controllers

Due to the limited number of control strategies provided by the manufacturer for the KUKA LWR IV robot, a new set of ROS-enabled controllers has been developed in *C++* language[1], following some interesting paradigms available in the robotics literature.

4.1 Robot Control Problem Definition

Consider the expression of the dynamics of a n-links robot manipulator

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau, \tag{3}$$

where $M \in \mathbb{R}^{n\times n}$ is the manipulator inertia matrix, $C \in \mathbb{R}^{n\times n}$ the Coriolis and centripetal forces matrix, $g \in \mathbb{R}^n$ is the gravity forces vector, $q \in \mathbb{R}^n$ is the set of configuration variables for the robot and $\tau \in \mathbb{R}^n$ denotes the torques applied at the joints. A compact version of (3) could be written in the form

$$M(q)\ddot{q} + n(q,\dot{q}) = \tau \tag{4}$$

where $n(q,\dot{q}) = C(q,\dot{q})\dot{q} + g(q)$. The problem of the control can be formalized as the choice of the torques τ that will stimulate the robot to behave according to some desired dynamics, such as reaching a set-point or tracking a trajectory.

4.1.1 Joint Space VS Operational Space Formulation

A classical approach for manipulators control is to define a task in the joint space, i.e. providing a motion law for specific joints to be followed. An alternative way to define the motion law in task space is the *Operational Space Formulation* [**?**], in which the desired behavior of the robot can be described through the dynamics of its end-effector. This approach allows to formulate tasks in a more human-like way, where the importance is put more on the manipulation task rather than on the internal behaviour of the robot.

4.2 Computed Torque Control

Given desired joint positions, velocities and accelerations, the required joint torques can be computed using (3), resulting in a *feedforward* (open-loop) control law

$$\tau(q,\dot{q},e,\dot{e})|_{q\equiv q_d} = M(q_d)\ddot{q_d} + C(q_d,\dot{q_d})\dot{q_d} + g(q_d). \tag{5}$$

But the limit of this control law is that, the tracking is assured only if the initial condition errors are zero ($q(0) = q_d(0), \dot{q}(0) = \dot{q_d}(0)$). To cope with

[1]Source code available at `https://github.com/enricocorvaglia/kuka-lwr`

these model uncertainties, a *feedback* action must be added to the controller, implemented as a simple PD. The result is the computed torque control law (control scheme in Figure 4)

$$\tau(q, \dot{q}, e, \dot{e}) = \underbrace{M(q)\ddot{q}_d + C(q, \dot{q})\dot{q} + g(q)}_{\tau_{feedforward}} + \underbrace{K'_p e + K'_v \dot{e}}_{\tau_{feedback}}, \tag{6}$$

where K'_p and K'_v are positive-definite matrices and $e = q_d - q$ is the tracking error. Furthermore, imposing $K'_p = M(q)K_p$ and $K'_v = M(q)K_v$ leads to

$$\tau(q, \dot{q}, e, \dot{e}) = M(q)(\ddot{q}_d + K_p e + K_v \dot{e}) + C(q, \dot{q})\dot{q} + g(q). \tag{7}$$

It is important to note that, if matrices K_p and K_v are diagonal, it leads to n decoupled second-order systems, where n is the number of the robot joints. In this way, the asymptotically stability of the equilibrium $e = \dot{e} = 0$ is guaranteed, according to the closed-loop poles placement rules.

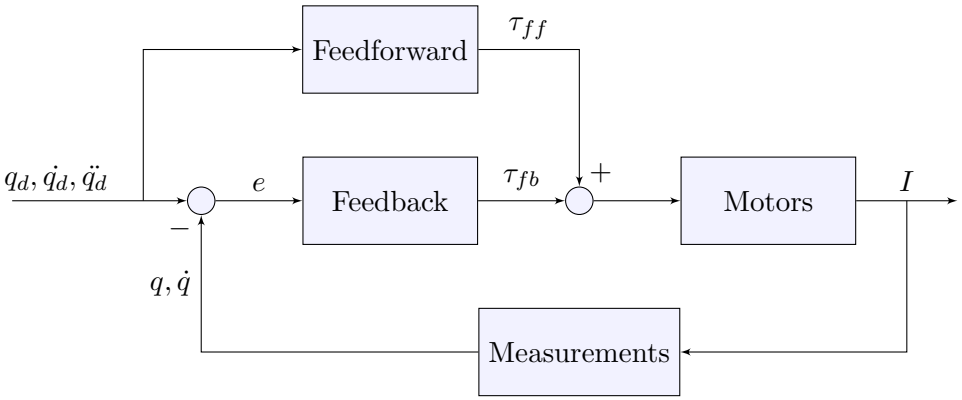


Figure 4: Control scheme for Computed Torque controller. I is the vector of output current measured from joint actuators which is then transduced to joint positions and velocity.

4.3 Inverse Kinematics-based Control

Given an end-effector position and orientation, the problem of inverse kinematics consists in determining the corresponding joint configurations. Since the system to solve is in general nonlinear, it's not possible to compute a solution in closed-form. This leads to different scenarios, i.e. no admissible solutions in presence of structure singularities, multiple or infinite solutions if the robot is redundant (which is the case of KUKA LWR IV, see Section 2). As an extension of the forward and inverse kinematics problem, it is also possible to define a relationship between the end-effector linear and angular velocity and the joint velocities as follows

$$\dot{x} = J\dot{q} \qquad q \in \mathbb{R}^n,\ x \in \mathbb{R}^m \tag{8}$$

representing the *Differential Kinematics* equation. The mapping is described by the *Jacobian* matrix $J \in \mathbb{R}^{m \times n}$ which depends on the robot configuration. Inverting (8), leads to the *Inverse Differential Kinematics* problem

$$\dot{q} = J^{+}\dot{x} \tag{9}$$

where J^{+} is a generalized inverse of the Jacobian. For the project's purpose, it has been implemented in ROS as the Moore-Penrose damped pseudo-inverse through the SVD method, denoted in the sequel as $J^{\dagger}$.
In view of the operational space formulation, given a desired end-effector pose x_d and velocity $\dot{x}_d$, a task can be defined as

$$e = x_d - x(t) \tag{10}$$

where $x(t)$ is the end-effector pose at time t. The related reference motion in the task space is

$$\dot{e}^{*} = \dot{x}_d + K(x_d - x) = \dot{x}_d + Ke \tag{11}$$

where K is a positive definite gain matrix (usually diagonal).

4.3.1 Hierarchical Inverse Kinematics

It is possible to exploit the redundancy of the manipulator, i.e. when the rank of J is smaller than n, to assign a secondary task by projecting an arbitrary vector on the null-space of the Jacobian, thus not affecting the higher priority task. So, considering two tasks e_1 and e_2, the following control law will perform exactly $\dot{e}_1^{*}$ and, if possible, $\dot{e}_2^{*}$

$$\dot{q} = J_1^{\dagger}\dot{e}_1^{*} + (J_2P_1)^{\dagger}(\dot{e}_2^{*} - J_2J_1^{\dagger}\dot{e}_1^{*}) \tag{12}$$

This method is called *Stack of Tasks* and can be generalized for k tasks, ensuring a correct hierarchy between them [**?**],[**?**]

$$\begin{cases} \dot{q}_0 = 0 \\ P_0 = I_n \\ \dot{q}_i = \dot{q}_{i-1} + (J_iP_{i-1})^{\dagger}(\dot{e}_i^{*} - J_i\dot{q}_{i-1}) \\ P_i = P_{i-1} + (J_iP_{i-1})^{\dagger}(J_iP_{i-1}) \end{cases} \tag{13}$$

where P_i is the projection operator onto the null-space of J_i and the robot joint velocity realizing all the tasks is $\dot{q} = \dot{q}_k$. This way, all the tasks are decoupled from each other and if a task won't be feasible it will be executed only at its best without disturbing the higher priority tasks.

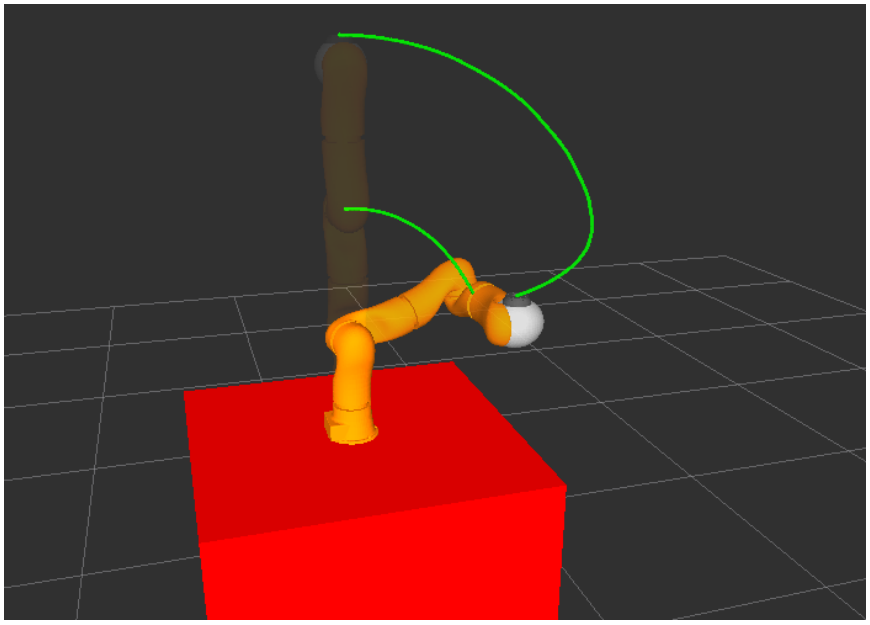


Figure 5: Simulated robot performing two tasks at different priority levels using Hierarchical Inverse Kinematics.

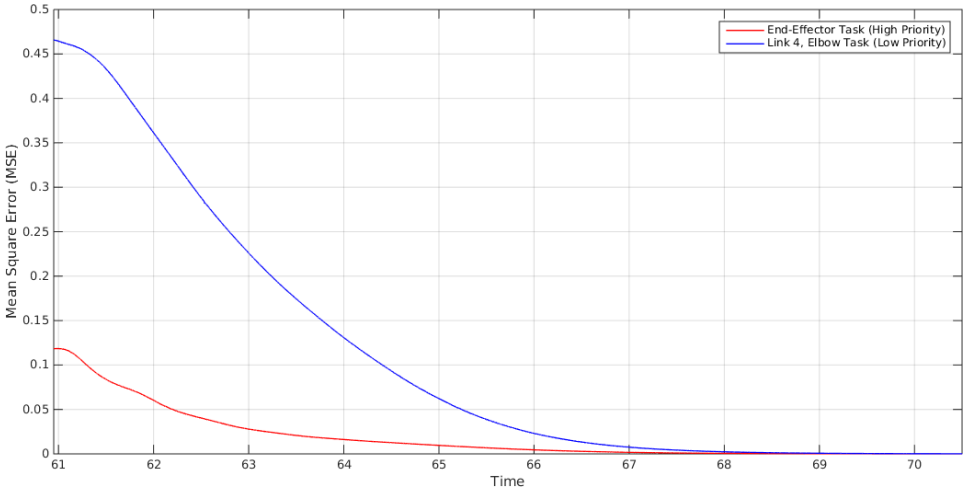


Figure 6: Evolution of the Mean Square Error (MSE) of two tasks controlled through Hierarchical Inverse Kinematics.

4.4 Inverse Dynamics-based Control

The problem of inverse dynamics in the operational space consists in the determination of the joint torques τ that realizes a specific motion $\ddot{e}^*$ assigned in terms of $\ddot{x}_d, \dot{x}_d, x_d$, the end-effector desired acceleration,velocity

and position

$$\ddot{e}^* = \ddot{x}_d + K_d(\dot{x}_d - \dot{x}) + K_p(x_d - x) = \ddot{x}_d + K_d\dot{e} + K_p e \tag{14}$$

where K_p and K_d are positive definite diagonal matrices. Recalling the manipulator dynamic model in the compact form (4), the choice of the *inverse dynamics control*

$$\tau = M(q)y + n(q, \dot{q}) \tag{15}$$

leads to a double integrators system

$$\ddot{q} = y$$

where the control y is designed to track the desired motion $\ddot{e}^*$ through the following second-order relationship

$$\ddot{e}^* = J(q)\ddot{q} + \dot{J}(q, \dot{q})\dot{q} \tag{16}$$

It is then possible to write, omitting dependencies in q and its derivatives

$$\ddot{e}^* + b = JM^{-1}\tau \tag{17}$$

where $b = JM^{-1}n - \dot{J}(q, \dot{q})\dot{q}$. Defining $\Omega = JM^{-1}J^T$ and $\Lambda = \Omega^{-1}$ and multiplying (17) by Λ leads to

$$\Lambda\ddot{e}^* + \Lambda b = \bar{J}^T\tau \tag{18}$$

where $\bar{J} = M^{-1}J^T\Lambda$ is the generalized inverse of J weighted by M^{-1}. Finally, multiplying (18) by J^T it is possible to define the control law that realizes the reference motion $\ddot{e}^*$

$$\tau = J^T\Lambda(\ddot{e}^* + b) \tag{19}$$

4.4.1 Hierarchical Inverse Dynamics

Equation (19) implements the inverse dynamics control for a single task e. As for inverse kinematics, the manipulator redundancy can be exploited also at dynamics level to give the possibility to execute other lower priority tasks, using the *Stack of Tasks* framework (applied to k tasks)

$$\begin{cases} \tau_0 = 0 \\ {N_0}^T = I_n \\ \tau_i = \tau_{i-1} + {J_i}^T\Lambda_{i|i-1}(\ddot{e}_i^* + b_i - J_iM^{-1}\tau_{i-1}) \\ {N_i}^T = {N_{i-1}}^T - {J_i}^T\Lambda_{i|i-1}J_iM^{-1} \end{cases} \tag{20}$$

where $\Lambda_{i|i-1} = {\Omega_{i|i-1}}^{-1}$ and $\Omega_{i|i-1} = {J_i}^TM^{-1}{N_{i-1}}^TJ_i$. The projection operator is ${N_i}^T$ and the control τ applied to the robot is τ_k.

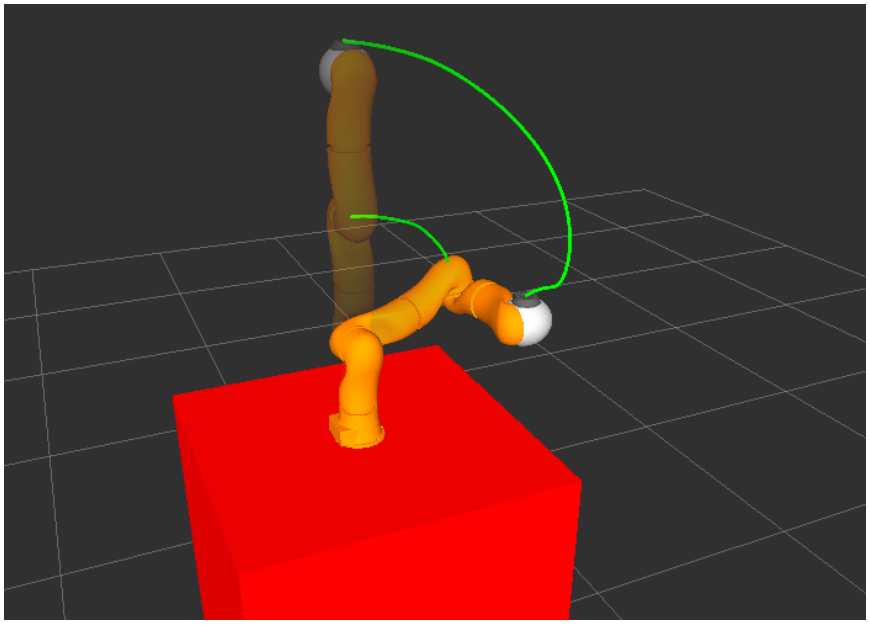


Figure 7: Simulated robot performing two tasks at different priority levels using Hierarchical Inverse Dynamics.

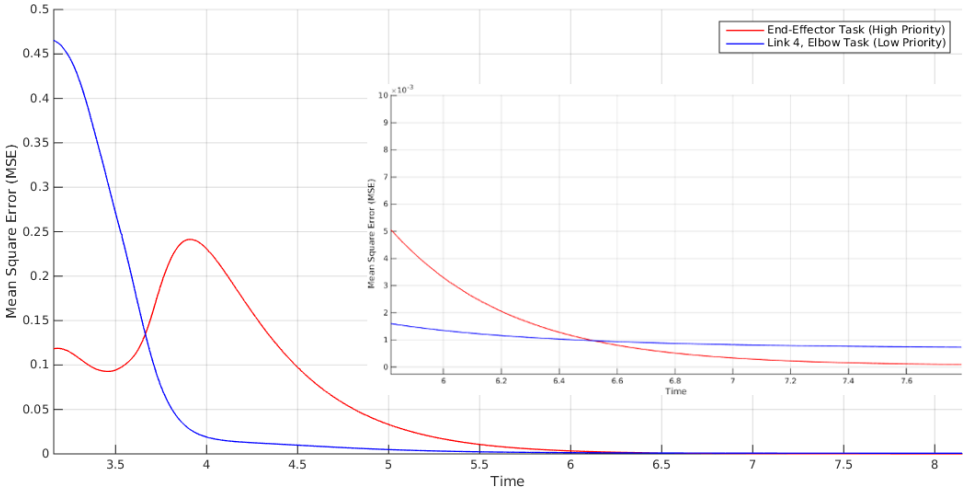


Figure 8: Evolution of the Mean Square Error (MSE) of two tasks controlled through Hierarchical Inverse Dynamics. Notice in the zoomed plot how the higher priority task converges to zero faster.

4.4.2 Minimum Effort Control

Another way to take advantage of the redundant DOFs, other than performing additional motion tasks, is to use the Jacobian null-space projection

method to maximize an objective function in the joint space as follows

$$\tau = J^T \Lambda(\ddot{e}^* + b) + N^T \tau_{null} \tag{21}$$

where in general $\tau_{null} = \nabla_q H(q)$ and $H(q)$ is a differentiable objective funcion to be maximized, implementing a specific criterion with the goal of moving the manipulator in its gradient direction. Taking inspiration from human manipulation, it is possible to reproduce those natural motions in an efficient way, setting the objective function $H(q) = g(q)^T W g(q)$, where $g(q)$ is the gravitational force vector as in (3) and W is a constant diagonal weighting matrix [**?**]. This way, control law (21) will minimize joint masses and payload gravitational forces while guaranteeing asymptotic stability of the system and not affecting the primary motion task referenced by $\ddot{e}^*$.

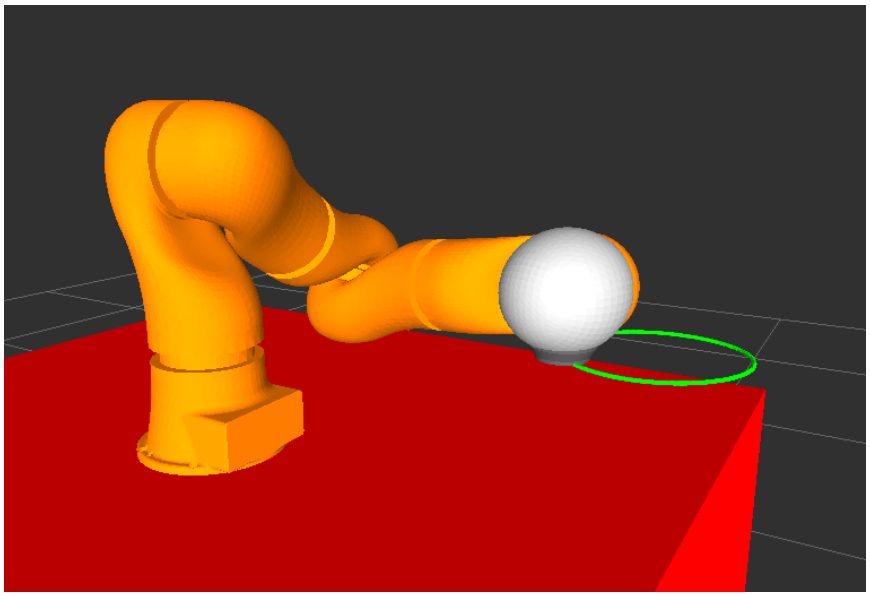
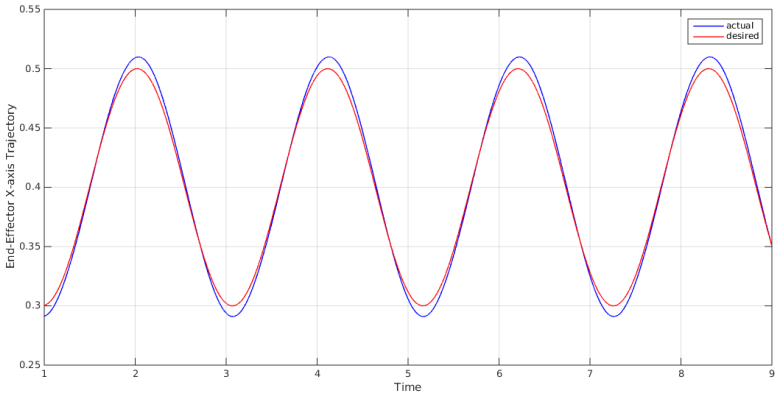
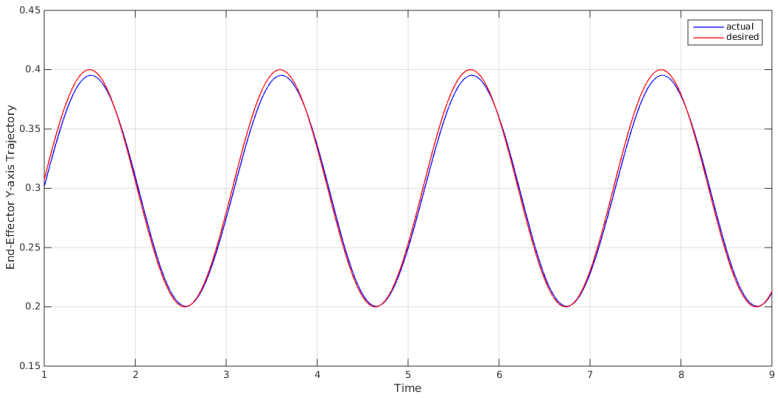


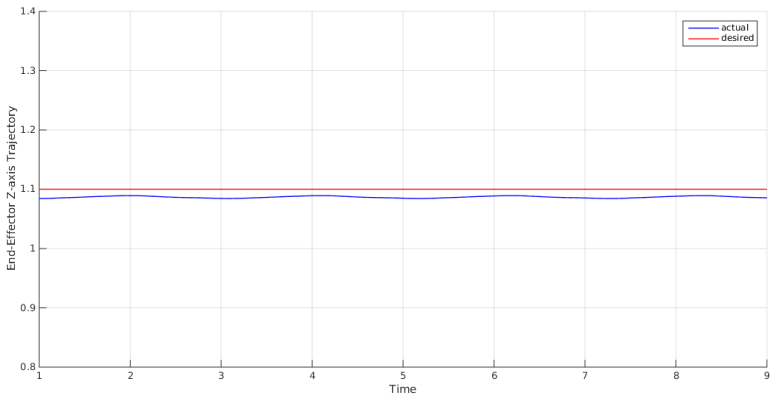
Figure 9: Simulated robot tracking a circle figure using Minimum Effort Control.



(a) X-Axis



(b) Y-Axis



(c) Z-Axis

Figure 10: Axis-specific end-effector trajectories tracking a circle figure using Minimum Effort Control.

4.5 Backstepping Control

Backstepping is a recursive methodology for designing both feedback control laws and associated Lyapunov functions in a systematic way, with the goal of obtaining a good flexibility over the nonlinearities. Consider the following nonlinear system in which the origin is an equilibrium

$$\begin{cases} \dot{\xi}_1 = f(\xi_1) + g(\xi_1)\xi_2 \\ \dot{\xi}_2 = h(\xi_1, \xi_2) + \ell(\xi_1, \xi_2)u \end{cases} \tag{22}$$

with $\xi_1 \in \mathbb{R}^{n_1}, \xi_2 \in \mathbb{R}^{n_2}, u \in \mathbb{R}^{n_2}$ and $\ell(\xi_1, \xi_2) \in \mathbb{R}^{n_2 \times n_2}$ is invertible for all ξ_1, ξ_2. Assume that a stabilizing feedback law $u' = \Gamma(\xi_1)$ exists for the subsystem

$$\dot{\xi}_1 = f(\xi_1) + g(\xi_1)u' \tag{23}$$

and that a Lyapunov function $V(\xi_1)$ is postulated such that

$$\dot{V}(\xi_1) = V_{\xi_1}(f + g\Gamma) \tag{24}$$

is at least negative semi-definite, with $V_{\xi_1} = \frac{\partial V}{\partial \xi_1}$. So, the control of the whole system (22) is obtained making ξ_2 to track the value of $u' = \Gamma(\xi_1)$.

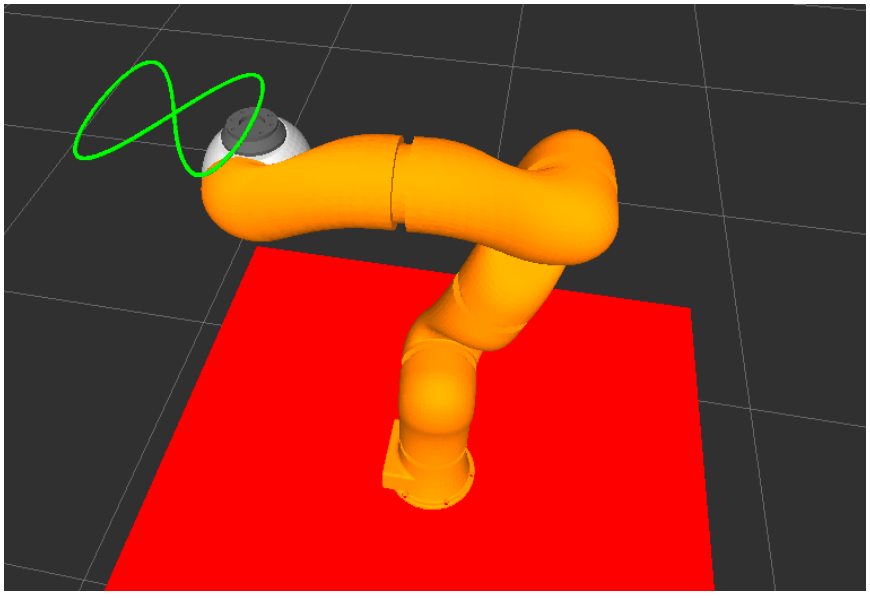


Figure 11: Simulated robot tracking a 8 figure using Backstepping Control.

The backstepping control method can be applied to a robotic manipulator described by its dynamic model (3), designing the joint torques τ that track the trajectory $q_d(t)$. Defining the tracking error $e = q - q_d$ and its

derivative $\dot{e} = w$, it is possible to consider the following system

$$\begin{cases} \dot{e} = w \\ \dot{w} = \ddot{q} - \ddot{q}_d = -\ddot{q}_d - M^{-1}\left(C\dot{q} + g(q)\right) + M^{-1}\tau \end{cases} \tag{25}$$

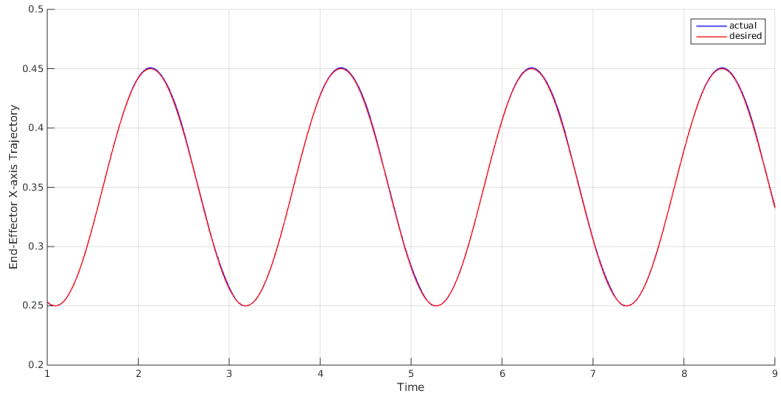
which can be interpreted in the form of (22), imposing $e = \xi_1, w = \xi_2, f(e) = 0, g(e) = 1, \ell(e,t) = M(e+q_d)^{-1}$ and $h(e,w,t) = -\ddot{q}_d(t) - M^{-1}\left(C(\dot{e} + \dot{q}_d) + g\right)$. After performing a stability study with Lyapunov control functions, the backstepping control law for a fully actuated manipulator can be defined as

$$\tau = M\ddot{q}_r + C\dot{q}_r + g(q) - K_d s - e \tag{26}$$

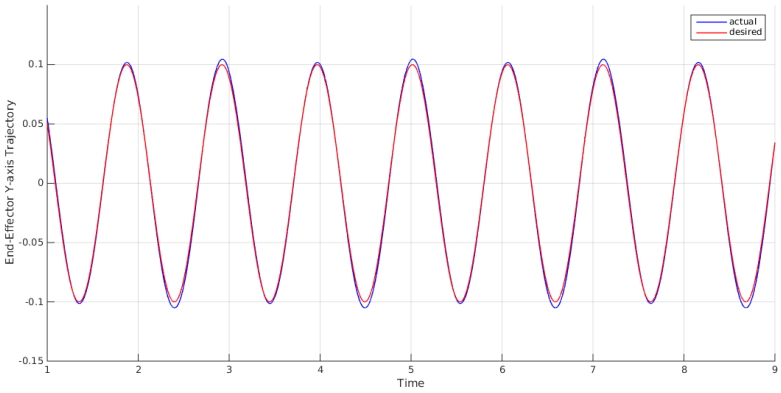
where

$$\begin{aligned} s &= \dot{e} + \Lambda e \\ &= \dot{q} - (\dot{q}_d - \Lambda e) = \dot{q} - \dot{q}_r \end{aligned}$$

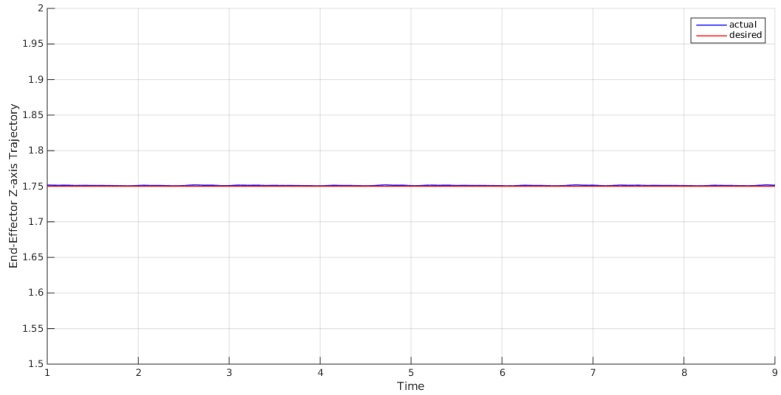
and $\dot{q}_r$ is the so-called reference velocity.



(a) X-Axis



(b) Y-Axis



(c) Z-Axis

Figure 12: Axis-specific end-effector trajectories tracking a 8 figure using Backstepping Control.

4.6 Dynamic Sliding PID Control

Regarding simple tasks such as set-point regulation, a proportional-integral-derivative (PID) controller would be enough, but the limitation of this class of controllers is that they can't achieve asymptotic stability for trajectory tracking. However, it is possible to enhance PID controllers by applying to them the sliding mode theory, as explained in the sequel [**?**]. So, given a desired motion specified by $q_d, \dot{q}_d, \ddot{q}_d \in \mathbb{R}^n$, it is analyzed the problem of designing the controller τ which guarantees the tracking of $q_d(t)$ without any knowledge of the system dynamic model and without introducing high-frequency components, by considering the following steps

$$\dot{q}_r = \dot{q}_d - \alpha\Delta q + S_d - \gamma\sigma \tag{27}$$
$$\dot{\sigma} = sgn(S_q) \tag{28}$$

where Δq is the tracking error $q - q_d$, $\alpha, \gamma \in \mathbb{R}^{n\times n}$ are diagonal positive definite gain matrices and $sgn(S_q)$ represents the input-wise discontinuous signum function of $S_q \in \mathbb{R}^n$, the so-called sliding surface, being defined by

$$S_q = S - S_d \tag{29}$$
$$S = \Delta\dot{q} + \alpha\Delta q \tag{30}$$
$$S_d = S(t_0)e^{-\kappa(t-t_0)} \tag{31}$$

with $\kappa > 0$. Considering a new error coordinates S_r defined as

$$S_r = \dot{q} - \dot{q}_r \tag{32}$$

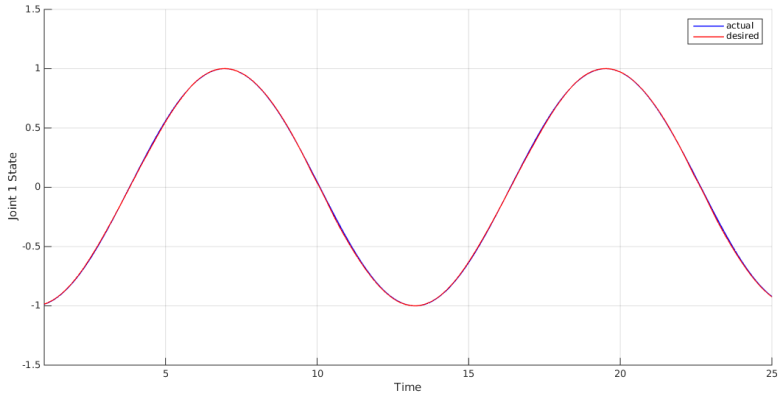
and substituting $\dot{q}_r$ as in (27), it leads to

$$S_r = S_q + \gamma\sigma \tag{33}$$

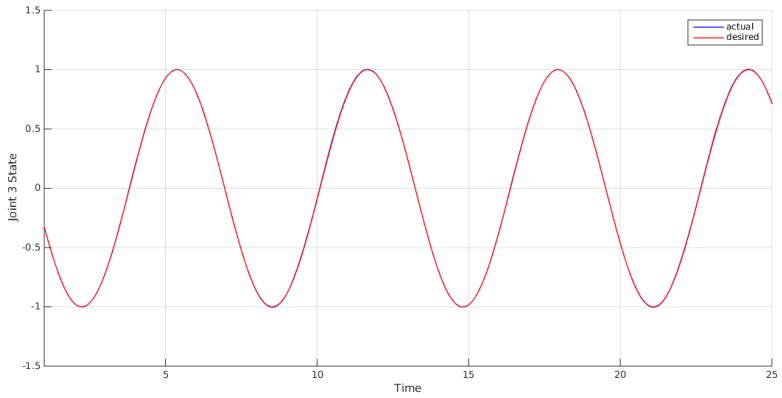
Finally, the controller τ which realizes the tracking of $q_d(t)$ in closed loop is

$$\tau = -K_d S_r \tag{34}$$

where $K_d \in \mathbb{R}^{n\times n}$ is a diagonal symmetric positive definite matrix. According to the controller structure, the semi-global exponential tracking is assured and the passivity of the system is preserved, as long as γ in (27) and K_d are large enough, for small initial conditions.



(a) Joint 1



(b) Joint 3

Figure 13: Joint-specific sine waves trajectories tracking using Dynamic Sliding PID Control.

5 Projecting Issues

In this section are described some of the major implementation problems encountered during the realization of the project and that unfortunately remained in part unresolved. When designing a controller, certainly one requirement that has to be met is to reach an efficient complexity in terms of computational resources. In the sequel it's analyzed a category of control strategies which requires, at least in this case study, the exploitation of a big amount of resources that unfortunately weren't available on the machines used for experiments. Another flaw encountered regards the process of switching from the simulation environment to the real robot interface. As expected, even if the robot has been modeled and rendered almost perfectly for the simulation purpose, it lacks some important aspects of its dynamics.

5.1 Adaptive Control

In the previous sections it has been assumed that the dynamic model of the robot is perfectly known at any instant of time, but, in practice, this can't be always true. In fact there exist some model uncertainties depending on the robot configuration, mainly mechanical parameters like inertia matrices and masses for the single links or those related to the robot payload. To overcome this intrinsic problem, it is always possible to write the dynamic model of the robot (3) through a linear parametrization

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = Y(q,\dot{q},\ddot{q})\pi = \tau \tag{35}$$

where the vector $\pi \in \mathbb{R}^r$ contains the combinations of the aforementioned physical parameters and $Y(q,\dot{q},\ddot{q}) \in \mathbb{R}^{n\times r}$ is called the *regressor* matrix. In brief, this sums up the aim of the *Adaptive Control*, which, as the name suggests, makes the control law to adapt to the variations of the unknown robot model parameters. Recently, an analytic closed formula for the regressor has been derived from the Lagrange equations, in terms of the Denavit-Hartenberg parameters and the joints configurations $q, \dot{q}, \ddot{q}$, and implemented in *Mathematica* environment [**?**]. For a 7-links manipulator like KUKA LWR, the resulting regressor matrix has dimensions 7x70 in its unsimplified form, making literally impossible to handle an adaptive control with the owned instruments.

5.2 From Simulation to Real Robot

The purpose of the development of robotics applications in a simulated environment is exactly to test and debug the software produced before loading it on the physical robot. Ideally, this passage would occur seamlessly (see Figure 14), but for several reasons which are now explained it doesn't happen in reality.

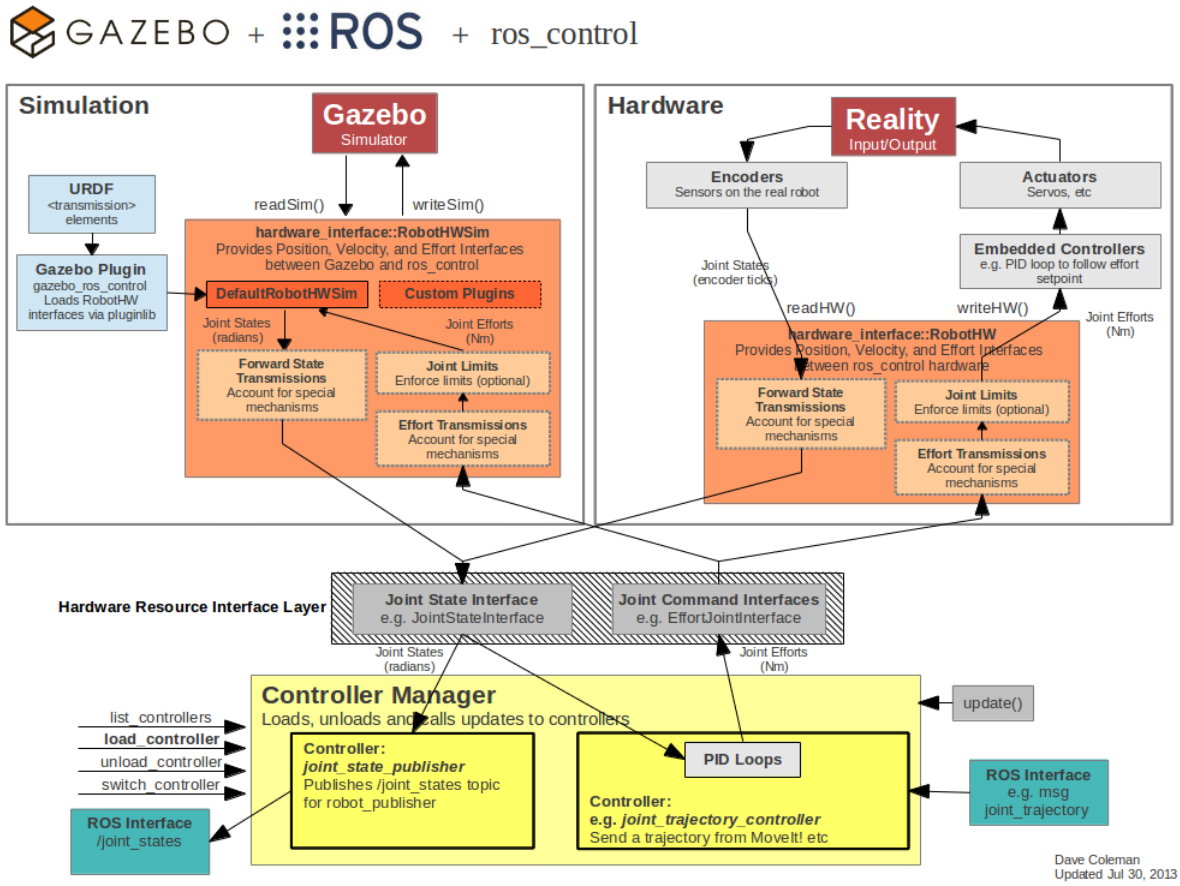


Figure 14: Scheme of the interaction between ROS and Gazebo, with particulars on simulated and real robot.

5.2.1 Sensors Measurements

The first discrepancy regards the presence of sensors for joint velocities measurements. In fact, while Gazebo provides them accurately through simulated sensors, the KUKA LWR arm is only equipped with position and torque sensors. This, indeed, arises the need to derive the velocities from the measured positions through a process of online differentiation, with the following methods being analyzed

Euler Method This is one of the most basic techniques to compute numerical differentiation and its formulation is

$$\dot{q}_n = \frac{q_n - q_{n-1}}{h} \tag{36}$$

where h is the sampling time. Since the output of this differentiator is usually a bit noisy, it's convenient to pass it through an exponential smoothing filter, which is basically a weighted average between the last computed velocity value and the previous one

$$\tilde{\dot{q}}_n = \alpha \dot{q}_n + (1 - \alpha)\tilde{\dot{q}}_{n-1} \tag{37}$$

where $\tilde{\dot{q}}$ denotes the smoothed value and $\alpha \in (0, 1)$ is the smoothing factor.

Levant Robust Differentiator This method is based on the sliding mode theory and gives asymptotically exact derivatives [**?**]. In comparison with Euler, Levant differentiator performs significantly better in terms of noise rejection and derivation of signals, but the major flaw which led this method to be discarded is that introduces a relevant phase shift.

So, according to the tests conducted and evaluating a trade-off between exact but out-of-phase signals and noisy but coherent signals, the best method resulted the Euler one.

5.2.2 Gravity Compensation

Recalling the built-in controllers (1),(2) of the KUKA LWR arm, the term $f_{dynamics}(q, \dot{q}, \ddot{q})$ must be analyzed. It refers to the gravity compensation performed internally by the robot, in a way that the commanded torque isn't indeed affected by the gravitational field. This, unfortunately, generates a conflict between the internal compensation and the whole control law, which at dynamic level requires the knowledge of the gravitational force to compute the input torques. Since such term can't be forced to zero, one workaround could be that of not considering the gravity ($g(q) = 0$) in all model-based controller, introducing a really negligible error.

6 Conclusions and Future Works

The present work had the aim of implementing an extended set of different classes of controllers for the robotic arm KUKA LWR IV, exploiting a flexible system which is rapidly growing in the field of robotics, including research and whole industrial processes. In fact, all the code produced could not only be accessed and used from users operating the same robot, but ideally from everyone who works with ROS, at least performing some minor edits to fit their needs. Thinking about future works starting from this project, it could be interesting the extension to a bimanual system in a human-like arms configuration. For example, taking advantage of the Stack of Tasks framework could be also provided obstacle and collision avoidance, making easier the interaction between the robot and humans or the handling of more complex operations.