A Component-Based Software Architecture for Control and Simulation of Robotic Manipulators

Federica Ferraguti, Nicola Golinelli, Cristian Secchi
Department of Science and Methods for Engineering
University of Modena and Reggio Emilia, Italy
{federica.ferraguti}@unimore.it
{cristian.secchi}@unimore.it
{nicola.golinelli}@gmail.com

Nicola Preda, Marcello Bonfè
Engineering Department
University of Ferrara, Italy
{nicola.preda}@unife.it
{marcello.bonfe}@unife.it

Abstract

The paper describes a software architecture for control and simulation of a generic robotic manipulator. The algorithmic part of the system is implemented using the Orocos component-based framework and its related library for robotic applications, while the graphical animation of the robot is developed with Blender. The proposed control and simulation framework is modular, reconfigurable and computationally efficient. Moreover, it can be seamlessly integrated into a more complex control architecture for a complete intelligent robotic system.

1. Introduction

Simulation of system components plays a crucial role in the verification of technological solutions which are conceptually developed in the Model-Based System Engineering (MBSE) [1]. In the context of MBSE, indeed, models and simulations allow systems engineers to investigate and predict the behavior of system alternatives without the need for physical prototyping. A physical model for simulation can be used even to test control algorithms and mathematical models before physical validation on the real robot or at the very early stage of the system development. Furthermore, a robotic setup like the one proposed in [2] may not be fully operational all the time, for several reasons (e.g. temporary hardware failures, laboratory space occupied for different tests, etc.).

Several commercial and open-source robotic simulators are available: some tools provide pure computational solutions (e.g. the Robotics Toolbox for Matlab [3]), while others, like Gazebo [4] and Morse [5], include advanced 3D graphics. In this paper we present a software architecture for control and simulation of a generic robotic manipulator that has the advantage, compared with other solutions, of being modular, reusable, completely reconfigurable and computationally efficient. Indeed, the dynamics simulation and the control architecture are implemented using the component-based design tool Orocos [6] and its related library for robotic applications (KDL), while the graphical animation of the robot is developed with Blender [7].

Blender could be used for both the dynamics simulation and the graphical rendering, as it is in one of the existing simulators previously cited [5]. In this way, the model of the robot would be directly interfaced in closed-loop with the control algorithms. However, this solution would increase the computational time because of the significant load introduced by the Blender physics simulation engine. The control and simulation framework proposed in this paper, instead, embeds a closed-form mathematical model of robot dynamics, so that physics is simulated with a reduced computational effort. To demonstrate the efficiency of the proposed architecture, we compared it with the Robotics Toolbox for Matlab [3], since the latter include similar closed-form dynamical models.

Finally, interfacing existing simulators with more complex architectures can be cumbersome. Since the proposed architecture uses Orocos components for both control and simulation, it allows one to easily connect the controllers used on the real robot or an existing trajectory planning to the developed simulator.

2. System Architecture

The request for an efficient simulator of a robotic arm, possibly interacting with a virtual environment and allowing a realistic animation of specific motion tasks, was raised during the development project of an intelligent surgical robotic system, as described in [2]. During this project, a distributed control and supervision software architecture has been developed, using the Orocos framework as a unifying implementation technology.

The main requirement on the development of a robotic simulator, to be included in the existing control and super-
vision architecture, was the possibility to use it as pure replacement of a physical robot or, in other words, to seamlessly “include” it in a closed-loop with the very same controllers used in real situations. These features have been obtained simulating the robot dynamics by specifically developed Orocos components, whose outputs are the joint positions, and creating a graphical model of the robot with Blender, a tool for 3D graphics and animation. Clearly, the advantages of using Orocos components for both control and simulation are related to the unification of software development tools and technologies, the computational efficiency and the total absence of “glueing” software modules, that would be instead required to connect the controllers used on the real robot and simulators developed in dedicated environments (e.g. [5]).

As a result, the overall component-based software architecture implementing the simulation method is described by Figure 1. The main Orocos components (Robot Dynamics, Runge-Kutta Integrator and Orocos Bridge) used for simulation are connected to the Orocos component developed for control purposes, using the same ports that would be connected to the Orocos components interfering with the robot hardware. It is important to remark that all of these software components are totally generic and parameterized by the data stored in an auxiliary URDF (Unified Robot Description Format, [8]) file.

![Figure 1. Simulator software architecture](image)

The graphical rendering of the robot is executed in Blender (Robot Animation), simultaneously with the execution of the simulation. Of course, this part of the architecture is strictly robot-dependent, because it relies on the availability of an accurate CAD model of the manipulator. In particular, this paper focus on the Barrett WAM 7–DOF manipulator, since it is physically available in our laboratory and much detailed information is available on its physical structure.

3. Robot Simulator in Orocos

As said in the previous section, the robot dynamics has been simulated using the Orocos framework and in particular the Kinematics and Dynamics Library (KDL), a sub-project of Orocos that develops an application independent framework for modelling and computation of kinematic chains.

The core part of the dynamic simulation, developed using Orocos KDL, calculates the corresponding joint positions, by means of three independent components:

1) “Controller” component (see naming on Figure 1): it computes the joint torques to be applied and write them on the specific port of the Orocos component (torquesOutPort). Two types of controllers have been implemented:

- **Proportional-Integral-Derivative controller (PID):** calculates an error value as the difference between the measured joint positions (read on the jointPosInPort) and a desired setpoint of joint positions. The controller attempts to minimize the error using the PID algorithm, implemented in the so-called discrete time “velocity form”.
- **Impedance controller:** this implementation strictly follows the approach described in [9]. Using Orocos KDL, the full implementation of the impedance controller exploits various solver classes, for the calculation of the forward kinematics, the Jacobian matrix (though KDL provides the geometric Jacobian, from which the analytic Jacobian is then computed), the inertia matrix and Coriolis/gravity terms.

The implementation of different controllers highlights modularity and reusability of the whole Orocos-based architecture: nothing else but the control law implemented in the Controller component has to be changed in the system to make the manipulator simulate a different behavior.

2) “Robot Dynamics” component: it contains an implementation of the dynamics of a serial robot, whose mechanical structure is loaded from a URDF file, so that the component is automatically configured and the kinematic (i.e. Denavit-Hartenberg parameters) and physical parameters of the robot (i.e. for each link: mass, center of gravity and inertia tensor) are stored independently from the compiled code.

As said before, Orocos KDL provides useful features for the implementation of the “Robot Dynamics” component. More precisely, such component calculates the acceleration of the robot joints, given the current joint positions, velocities and applied torque, using an instance of the KDL::ChainIdSolver_RNE class. Such class embeds the Recursive Newton-Euler formulation of robot dynamics and provides methods to compute the inertia matrix of the robot, gravity and Coriolis torque contributions acting on the robot joints. The external force acting on the end-effector of the robot may also be provided as an additional input of the “Robot Dynamics” component.

3) “Runge-Kutta Integrator” component: it is a generic integrator based on a Runge-Kutta 4th order solver; this
component has been developed in order to be completely configurable and to allow the user to select the number of state variables to be integrated and the maximum order of integration. For the WAM case, the state variables are 7 (one for each joint) and the order of integration is 2 (i.e. position and velocity).

The integrator component expects that another component loaded by the Orocos deployer provides a service called “derivative_callback”, that will be used by the Runge-Kutta algorithm to evaluate the derivative step given an initial state and a time instant. Thanks to this approach the integrator may be connected to any other component (i.e. not necessarily the “Robot Dynamics” one) calculating the input derivative.

As shown in Figure 1, another Orocos component, called “Orocos Bridge”, has been implemented. Even though in this paper the described usage of this component is to allow the interaction between the Orocos-based simulator and Blender, the very same component is also used for interfacing the controller with the Barrett WAM control hardware, by means of TCP (Transmission Control Protocol). The Orocos Bridge component will be described in Section 5.

4. Robotics Animation in Blender

Blender is a free and open-source 3D computer graphics software used for creating animated films, visual effects, interactive 3D applications or video games.

To simulate a robotic system in Blender it is required to import the solid model of the robot. In the specific, we imported the CAD models of links of the WAM. Figure 2(a) shows the reconstruction of the WAM structure in Solidworks, while Figure 2(b) shows the CAD models imported in Blender and the robot located in the zero position.

![Figure 2. CAD model of the Barrett WAM](image)

The robot links simply imported as CAD models are not related each other. To create a harmonious object with a relation between links it is required to build the skeleton inside the structure of the WAM. In Blender, skeletons are known as armature objects.

The Blender armature object is a container for sub-objects, called bones. When a bone is added it is added to the whole armature. Thus, moving the armature will move all its bones, while moving a bone will move only the selected bone and the other bones connected. Figure 3 shows the armature of the WAM, which is composed of ten bones. Bones 0-1-2-4-6-7-8 are used for controlling each joint of the robot while bones 3-5 are used for connecting joints and cannot rotate. Even the bone “Base” cannot rotate because it is associated with the fixed base of the WAM.

![Figure 3. Armature of the WAM](image)

Each link of the WAM represents a mesh, i.e. a common object type used in a 3D scene composed of a set of connected vertices. Each mesh has been associated with a bone of the armature, as shown in the table in Figure 3. In this way, as a bone moves, it deforms or moves the vertices associated with it, i.e. the link of the WAM. Thus, a mesh that is associated with one of the links can be rotated applying the rotation to the corresponding bone. The advantage in using armatures is that with a relatively small number of objects, the bones, it is possible to control more vertices of a mesh, which are in effect grouped and associated with a bone.

To handle the position values received from the Orocos simulator, we used the Blender Game Engine. The Blender Game Engine is a powerful high-level programming tool that allows to create interactive 3D applications. The core of the BGEs structure are Logic Bricks. The goal of Logic Bricks is to offer an easy to use visual interface for designing interactive applications without any programming language knowledge. There are three types of Logic Bricks:

- **Sensors**: grab the occurrence of a particular event.
- **Controllers**: collect data sent by the sensors and trigger the proper response. In particular, we used the Python controller, which is a controller that checks the input using a user-programmed Python script or any other file containing Python code.
- **Actuators**: perform actions, such as move, create objects, play a sound.
The Game Engine has its own Python API, separate from the rest of Blender, which can be used to write scripts to control the application. This is done by creating a Python Controller and linking it to a Python script. Therefore, each link of the WAM model is controlled by a Python script in a Python Controller.

5. Interaction between Blender and Orocos

The output joint positions calculated by the Orocos-based simulator have to be transmitted to Blender, which animates according to the 3D model of the robot.

The interaction between the Orocos-based simulator and Blender is allowed by an Orocos bridge component (Orocos Bridge in Figure 1) that establishes a TCP socket connection on the server side of the communication channel. To allow the communication between the two processes, a socket connection has to be established even on the client side, i.e., in the Blender simulator.

The bridge component receives the joint positions from the Runge-Kutta Integrator and transmits these positions to the Blender simulator through the established socket. Then, the joints of the WAM in the Blender simulator will rotate accordingly to the positions received.

More precisely, the Orocos Bridge component establishes the connection on the server side of the communication channel. After that, the Orocos Bridge listens on the socket for connections. This process, typically blocks the process until a connection from the Blender side has been successfully established. Once the connection has been established, the Orocos Bridge reads on the jointPosInPort the target joint positions computed by the Orocos simulator. The positions are then sent to Blender through the socket, using the frequency given by the Orocos simulator period.

As said at the end of Section 4, each link of the WAM model in Blender is controlled by a Python script. This script is called by the controller of each link and contains the code to establish the socket connection on the client side and the commands to rotate the joints of the WAM according to the positions received from the Orocos simulator. Once the connection to the Orocos Bridge has been established, the joint positions are read from the socket and the Python script makes the corresponding bones of the WAM armature moving according to the command received.

6. Simulation Results

The efficiency and accuracy of the proposed simulation framework has been tested in a comparison with an equivalent simulation setup implemented in Matlab/Simulink, using the Robotics Toolbox [3]. Corke’s toolbox is currently the open-source library for Matlab/Simulink with the most complete set of algorithms for robotics problems.

Clearly, the main difference between the proposed Orocos-based simulation architecture and the Matlab/Simulink solution is the fact that, in the latter environment, a simulation model is not compiled as a standalone executable and its interface with external programs (e.g., Blender) is possible but cumbersome. The advantage of Matlab/Simulink is related to the possibility to adopt different numerical integration algorithms, while in the Orocos simulation described in this paper only the Runge-Kutta 4th order has been implemented. On the other hand, simulation results shown in this section prove that this algorithm is sufficient for all the tested case studies. Moreover, the C++ compiled code of our component-based architecture outperforms, in terms of execution time, the equivalent Matlab/Simulink simulator by several orders of magnitude.

In particular, the Orocos-based and the Matlab/Simulink simulators have been tested using both the PID controller and the Impedance controller on the same test motion, with the parameters given in the following table.

<table>
<thead>
<tr>
<th>Initial end-effector position</th>
<th>Desired end-effector position</th>
<th>Gains of the joint-level PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0 = [0.59 0.17 0 \pi/2]$</td>
<td>$x_d = [0.69 0.17 0 \pi/2]$</td>
<td>Proportional 300 10</td>
</tr>
<tr>
<td>$q_0 = [0.06 0.78 3.00] - 1.57$</td>
<td>$q_d = [0.04 0.91 3.00] - 1.21$</td>
<td>Derivative 10 1</td>
</tr>
<tr>
<td>$0.04$</td>
<td>$0.05$</td>
<td>Integral 50 1</td>
</tr>
<tr>
<td>$0.07$</td>
<td>$0.13$</td>
<td></td>
</tr>
<tr>
<td>$5\pi$</td>
<td>$7\pi$</td>
<td></td>
</tr>
</tbody>
</table>

Cartesian parameters of the impedance controller

<table>
<thead>
<tr>
<th>Stiffness</th>
<th>Damping</th>
<th>Integration step of Runge-Kutta solver: 5 $\mu$s</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000 N/m</td>
<td>100 Nm/rad</td>
<td>5 $\mu$s</td>
</tr>
<tr>
<td>100 Ns/m</td>
<td>35 Nm/s/rad</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4 shows the comparison, focused on the behavior of Joint 2, between the Orocos-based and the Matlab/Simulink simulators while performing the simulation of the WAM robot with PID control and Impedance control. As can be seen the accuracy of the Orocos-based simulation is the same as the Matlab/Simulink implementation. The small difference can be related to the fact that the URDF description of the WAM parameters does not allow one to specify the numbers with more than 10 decimal digits, which may be a (generally negligible) limitation when constants like $\pi$ are involved.

Figure 5 shows the norm of the Cartesian space error for the robot under PID control and Impedance control, simulated in the Orocos framework. As can be seen, the closed-loop system is stable using both controllers, though the zero-error steady-state cannot be reached within one second of simulation when using the PID control.

The computation times required to obtain the four simulations presented in this section, using an eight core In-
In future works we expect to introduce a model of the interaction between the robot and a compliant environment, possibly with a variable stiffness. In this way, we aim to implement and test an enhanced impedance control method, recently proposed in [10].

References


