Prototyping advanced real-time robotic controllers on Linux RTAI systems with automatic code generation

Gabriella Alotto, Basilio Bona, Tommaso Calvelli

Dipartimento di Automatica e Informatica
Politecnico di Torino – Corso Duca degli Abruzzi 24
10129 Torino, Italy
basilio.bona@polito.it

Abstract

The use of industrial robots in research and teaching laboratories proves difficult since the closed controller software and the proprietary hardware architecture prevent the execution of experiments testing advanced control algorithms, or the integration of new sensors not present on the original robot.

The integration of external sensors, such as force/torque sensors or cameras for vision-in-the-loop control, is the first step toward a new class of algorithms allowing interaction with the environment at low costs and within industrial standards. The real-time nature of the control task, that sets tight temporal constraints on the algorithm step completion, forces the user to devote a large part of the design time to topics remote from the main issues of the controller design, chiefly when he/she is not interested in software implementation issues.

This paper presents a low-cost robot controller prototyping architecture, based on Simulink-Stateflow design environment. One of its main characteristics is the automatic generation of the real-time execution code for the target processor, that runs under Linux RTAI operating system.

To show the applicability of the proposed solution, rapid prototyping of force control algorithms is experimentally tested on a COMAU Smart S2 6DoF articulated industrial manipulator, endowed with a force-torque sensor mounted on its tip.

1 Introduction

The extensive use of industrial robots in experimental research laboratories is made difficult by the closed software architecture of the controller and the proprietary hardware architecture, both preventing the execution of experiments involving advanced control algorithms or the integration in the control loop of new sensors not present on the original robot. Only in a limited number of cases an open architecture is available to interact with the native controller using a standard Personal Computer.

On the other hand, in the last decade, several Real Time Operating Systems (RTOS) have been developed allowing a remarkable increase in the performances of the low-cost PC-based control architectures. An example of these RTOS is RTAI-Linux [4], a real-time extension of the well known Open-Source operating system Linux.

The aim of this work is to create a simple architecture for Rapid Control Prototyping (RCP) where the supervision architecture and the controller of an industrial robot are rapidly designed, then downloaded and executed on a PC running the RTAI-Linux operating system. Even though other results have been presented along the same line (see for example [8]), the main contribution of this paper is to implement automatic code generation using Matlab and Real Time
Workshop (RTW) toolbox for Simulink-Stateflow. In this way, creating a real-time application for the supervision and control of a robot does not require to deal with details and intricacies of real-time programming. Moreover, since the low level code required to exchange data with the robot is enclosed in a set of basic blocks, the user must not worry about the way this exchange is performed.

In synthesis, the principal steps to be executed for the creation of a real-time Linux application are the following:

- Creation of a Simulink-Stateflow model using proper block-sets;
- Automatic generation via RTW of the source code for Linux-RTAI;
- Code compilation and execution on the Linux-RTAI computer;
- Execution of the graphical user interface connected with the real-time modules.

The Graphical User Interface used in the experiments is a development tool of RTAI, called RTAI-Lab.

Further details about RTW, RTAI and RTAI-Lab will be introduced in Section 2. A description of the hardware used for the experiments is given in Section 3. Section 4 presents a number of original Simulink blocks and Section 5 describe some experimental test-cases.

## 2 Automatic code generation for Linux-RTAI

Real-Time Workshop is a powerful Matlab tool which allows automatic generation of portable and optimized C code from a Simulink/Sateflow model, as described in [1] and [2].

Using Simulink, it is easy to create abstract high-level block diagrams that represent complex dynamic systems, simulate their behavior, display results and analyze them. Stateflow allows the creation of finite state machines, supervising the different functions of the robot, as the starting and stopping phases, normal operation, fault diagnosis and failure recovery, emergency situations, etc.

On invoking the RTW build procedure, an executable code version of the block diagram and of the finite state machine can be obtained and downloaded on the target PC. This gives the possibility to test the performances and the capabilities of the control architecture on the real system. It should again be noted that the entire architecture, including discrete events state logic, is translated into real-time compliant code, allowing to produce, apart from new control algorithms, a general supervision environment, with alarms, calibration phases, power-on and power-off behaviors, etc.

User can focus his/her attention on modelling and simulation issues, taking advantage of the design environment to analyze problems and provide possible solutions; furthermore, no deep knowledge of C programming is needed, since creation of executable code from the model is straightforward and real-time hard constraints are automatically satisfied.

Figure 1 presents the flow of operation followed by RTW build procedure. The model is first processed to obtain source code, then, through the right makefile, an executable version of the model is created: in this phase the generated code is linked with the harness program, and, if necessary, with the user custom code.

In details, this procedure consists of three main phases:

1. RTW analyzes and translates the Simulink-Stateflow block model: first, an intermediate description of the model is created in order to obtain, for example, the execution order of the blocks; then the model is truly translated into C source code using a Target Language Compiler (TLC) program;

2. RTW creates a make-file that specifies any required harness program or library, as well as the source code provided by the user.
3. Finally, executable code is obtained through a make utility, that is driven by the instructions contained in the make-file created in the previous phase.

The PC on which the code is generated is usually called Host, while system on which application runs is called Target: the generated applications can run on a wide variety of target environments, such as DSPs or real-time operating systems, and a wide variety of CPUs. In fact, using the capabilities of TLC, that is a fully configurable code generator, it is possible to describe both how to generate code for a chosen target and how to translate every Simulink block.

During a simulation, it is possible to interact with the model by tuning parameters or monitoring signals: the same capabilities are provided in the generated code, through data structures created to fit these requirements. In this way user can modify parameters values and observe any effects directly on the real system, and examine or log signals for further analysis, using a proper graphical interface.

RTW produces a real-time executable version of a Simulink-Stateflow model, so an appropriate platform should be selected to guarantee a correct application’s execution: the chosen target must
be able to properly schedule processes under its control, and to strictly respect the given timing constraints; in other words, it must offer the right real-time services.

A good example of such an architecture is Linux-RTAI, which also possess many tools to create and interact with applications belonging to Matlab/Simulink. In fact, RTAI makes available the main program which has to be linked with the generated source codes, in order to obtain an executable that can correctly run under this system. A Graphical User Interface, called RTAI-Lab, is also available, that allows real-time interaction with running processes.

RTAI [4] is an extension of the Linux kernel, developed by DIAPM at Politecnico di Milano, which transforms an operating system with no real-time capabilities into a real-time operating system. Using RTAI API, the programmer can easily manage tasks, interrupts and inter-processes communication mechanisms as fifos and remote procedure calls. Another important feature, unique to RTAI, is the possibility of develop real-time processes in User Space, thanks to services provided by a module called LXRT.

Normally real-time application should be developed in Kernel Space, in which the required services are directly available: with the capabilities of LXRT, instead, user can have access to them, almost without differences, from the User Space where the operating system is able to perform many safety controls; this possibility speeds up the development phase, allowing to find and remove errors very fast, since all debugging tools are available.

At this point applications generated by Matlab can run completely in User Space, as a normal application, although they are real-time tasks running at a higher priority than that given to Linux, and as such they can have access to critical resources as the hardware interrupts.

3 Robot and hardware description

The robot used to test the prototyping architecture is a COMAU Smart S2 6-DoF articulated robot, shown in Figure 3, present in the Robotics Laboratory of the authors’ Department.

![Figure 2: The 6-DoF industrial robot Comau Smart 3-S2.](image)

The original robot architecture, is available only with a proprietary controller, called C3G; an additional feature allows to add a link between the C3G controller and a PC, through the proprietary BIT3 board, whose function is to exchange data with the electronic boards in the cabinet through a high speed cable.
In the original configuration, this solution strictly requires and is limited to the MS-DOS operating system. In our case this limitation and the given PCC3Link library for the data exchange has been completely removed. As said previously, the operating system running on the PC is RTAI-Linux; the communications to and from the C3G controller is performed using a custom C library, or an S-Function library that will be described in the next Section.

The PC is an Intel-Pentium III running at 500 MHz. Its only constraint is to have onboard a free ISA slot for the BIT3 card used for the communication with the robot controller.

From the C3G point of view, several different modes are available for the communication with the PC and each one of them has a fixed operating frequency and a different entry point in the control loop of the robot. The synchronization between the PC and the C3G is achieved by an interrupt signal generated by the controller. In our experiment only the so-called proprietary “Mode 4” has been used. The main characteristics of this mode are briefly summarized:

- data exchange is performed at the fixed frequency of 1 kHz: the input to the C3G is given by the six motor current set-points, while the output from the C3G is given by the six joint angular positions;
- the trajectory planner, the control algorithm and all the safety functions of the C3G are bypassed and transferred on the external Linux-based PC;
- the power stage of the C3G is kept in place and used to drive the six joint motors.

As a consequence, within every sampling time $T_s = 1\text{ms}$, the PC must perform the following actions:

- read the six joint positions from the C3G;
- compute the new current set-points according to a proper control law;
- send the current set-points to the C3G controller.

3.1 Force-torque sensor

The robot is equipped with an ATI Gamma SI-130-10 force/torque sensor mounted at the robot tip. It is connected to an ISA bus transducer on the PC and makes possible to measure the forces and torques at the manipulator end-effector, at a maximum frequency rate of 7.8 kHz; measurements are available in N and Nm.

4 The block-set library developed for Simulink

The development of a Simulink-Stateflow abstract model requires a block-set library in order to define and implement all the communication processes between the PC and the C3G board. Besides, any external sensor must correspond to a Simulink block, as in the case of the force/torque sensor. Other required blocks are those implementing the control algorithm, the reference planning algorithm and any other custom algorithm necessary for the application. Finally, since a complete interaction with the robot implies several working states, such as power stage status, emergency situations and pre-positioning/homing stages, a Stateflow diagram is used to supervise the correct behavior of the model.

In this Section a description of the main blocks, divided into two categories, and of the state diagram is given. The first category includes all the blocks allowing data exchange with the robot (see Figure 3).  Four types of blocks belong to this category:

- The Interrupt handler block guarantees the synchronization with the robot, by requesting, installing and enabling the interrupts generated by robot controller. Every time an interrupt occurs, this block enables a triggered subsystem which will execute all the operations
Figure 3: Blocks that perform communication between the PC and the robot

(read actual positions, compute the new set-point, execute the control algorithm) as soon as possible and strictly within a sampling time. Thanks to the services offered by LXRT, managing an interrupt from User Space becomes simpler using an appropriate set of RTAI API that can perform this task.

- As explained in the previous section, every 1 ms a new vector of six drive currents has to be sent to the C3G inside the robot cabinet, and a new vector of six positions has to be read from the C3G. The Write torques block performs the write task: it receives the control torques from the model, translates them into currents expressed in D/A converter unit and writes the result to the C3G shared memory. The Read positions block is dual: it reads from the C3G shared memory the joint positions, translates them from resolver unit to radians and send them to the model.

- The Drive block simply enables or disables the power stage of the robot, by writing the proper command to the C3G shared memory. The driving signal of this block is controlled by the Stateflow diagram, according to user commands and/or emergency situations.

- The Read forces block performs the acquisition of forces and torques measured by the robot tip sensor. Data are obtained by reading from a specific memory location of the ISA transducer, which is initialized when the application is launched. This block has a second output that indicates if the sensor is calibrated or not: the calibration process is performed by changing the value of the related parameter.

Figure 4: PID controller and reference planner

The second group of blocks (see Figure 4) include a Control Algorithm, implemented in this case as a simple PID, and a reference planner. The PID control can be replaced by any other control algorithm that the user wants to test; the block receives actual and desired positions as
inputs, and gives the torque command as output. In Figure 5 a graphical implementation of the PID controller in continuous time is given. The real algorithm differs slightly from this one in order to insure a numerically efficient implementation.

\[
\tau = K_i \int \frac{d}{dt} K_p (q_r - q) + K_d \frac{d}{dt} (q_r - q)
\]

Figure 5: PID controller

The second block is the Reference Planner (RP); it is used to guarantee a coordinate motion while constraining the joints velocity to a trapezoidal profile. Its output is a run-time generated reference signal which satisfies these limits. In this way we do not need to know the trajectory before the movement starts, since it is computed on-line at each sampling interval. Maximum allowed velocity and acceleration can be set and changed interactively by the user during task executions: when the related parameters are modified and the current working state allows this, the Stateflow diagram redirects the new values on the appropriate input port of the RP.

Planning starts only if on-line planning state is enabled and the motor power stage is on: this is managed by the state diagram depending on the user commands and on the working state of the robot. The desired final position can be specified directly by the user (for example when a pre-positioning is needed), set by the state diagram when the homing procedure is in progress, or computed by another block according to a certain strategy.

The Stateflow Diagram represents the logic supervisor of the model: it receives data from the model, commands and data from the user, and - according to the current state - sends a set of signals to the model. These can be enabling signals, used to activate the proper part of the model, or data signals, containing information used by the activated blocks.

The diagram shown in Figure 6 is organized on different levels in order to simplify changes and capability improvements. The first level is composed by two states, corresponding to the status of the power stage. A proper user command allows to switch from one state to the other, although, if an emergency condition is detected, the transition to the Power stage off state is forced. Indeed the Power stage on state is a superstate, in which two tasks are performed: the Fault detection state recognizes any emergency condition, while the superstate Modality enabler manages the working modality, by generating the right enabling signals for the rest of the model.

In this superstate each modality is represented by a simple state, and transitions among them occur only when the user sends a specified command and all additional conditions are satisfied. For example, the user can start the force control only if the force/torque sensor has been properly calibrated. Many working modality are available, since the user can:

- Pre-position the robot by manually specifying the position to be reached or return automatically to the home position.
- Move the robot according to a trajectory previously saved in a file.
- Move the robot using a specific control strategy, such as the force control.
5 The experimental test–bed

To illustrate the characteristics of the proposed architecture, an experimental test-bed has been set up at the Robotics Laboratory of Politecnico di Torino. Our goal is to use the force/torque sensor to generate a proper trajectory in the cartesian space. To do so, considering the available Simulink blocks, we need to define the algorithm for generating positions from the force readings; in other words to approach the problem as a position-based force control, which is a very simple algorithm in the class of impedance control techniques [5].

An appropriate S-Function has been written performing the action shown in Figure 7.

![Figure 6: Structure of the Stateflow diagram](image)

![Figure 7: Action to be performed by the S-Function](image)

Although both forces and torques could be used for the generation of the trajectory (see also [6] and [7]), in this experiment only the forces are used for simplicity: therefore, as explained below, the control will only generate translations in the cartesian work-space.

5.1 Set-point generation

The generation of the set-points follows from the particular choice of the desired robot behavior.

In our experiments the manipulator has the force/torque sensor mounted on the wrist flange; the operator will randomly push or pull the tip, applying an external force to the physical reference frame mounted on top of the force/torque sensor (see Figure 8).

The controller must act to counter-balance the instantaneous force applied at the end-effector, while torques are filtered out; in other words, if there is no force, the manipulator shall not move; if it is pushed or pulled, the manipulator shall move in order to reduce to zero this force, receding or advancing according to the sign and magnitude of the force. The algorithm used to generate the set-points is detailed in the following.
Let $q(k)$ be the vector of the six actual joint positions at time $kT_s$, where $T_s = 1$ ms is the sampling period, and $p(k)$ is the vector of cartesian positions and orientations. Let $\mathcal{R}_b$ and $\mathcal{R}_e$ be the base and the end-effector reference frames respectively, with $\mathbf{R}_b^e$ being the rotation matrix from $\mathcal{R}_b$ to $\mathcal{R}_e$ and $\mathbf{d}_b^e$ being the translation vector between the two frame origins.

Using the robot forward kinematic function $p(k) = h_{\text{kin}}(q(k))$ we can compute the homogeneous roto-translation matrix $\mathbf{T}_b^e(k)$:

$$
\mathbf{T}_b^e(k) = \begin{pmatrix}
\mathbf{R}_b^e(k) & \mathbf{d}_b^e(k) \\
0 & 1
\end{pmatrix}
$$

representing the end-effector frame in the base frame.

The generalized force vector $\mathbf{f}_e(k)$, measured by the force/torque sensor, is expressed in the end-effector frame $\mathcal{R}_e$.

To represent this vector in the base frame $\mathcal{R}_b$, the following equation holds:

$$
\mathbf{f}_b(k) = \mathbf{R}_b^e(k) \mathbf{f}_e(k)
$$

From $\mathbf{f}_b(k)$ we compute the new position $\mathbf{d}_b^e(k + 1)$ of the tool in the base frame:

$$
\mathbf{d}_b^e(k + 1) = \mathbf{d}_b^e(k) + K_f \mathbf{f}_b(k)
$$

where $K_f$ is a diagonal matrix of gains $k_{fi}$, expressed as the inverse of elastic constant, $m \cdot N^{-1}$.

The orientation of the tool, as previously said, is kept constant at its initial value, that is:

$$
\mathbf{R}_b^e(k + 1) = \mathbf{R}_b^e(k)
$$

From the updated homogeneous matrix $\mathbf{T}_b^e(k + 1)$ we can compute the new joint set-point using the inverse kinematic function $h_{\text{kin}}^{-1}(\mathbf{R}, \mathbf{d})$:

$$
q(k + 1) = h_{\text{kin}}^{-1}(\mathbf{R}_b^e(k + 1), \mathbf{d}_b^e(k + 1))
$$

5.2 Experimental results

The set-point generation algorithm described in the previous Section has been implemented as a particular My S-Function called Force control. Together with the block-set library, a Simulink-Stateflow model has been created to perform an experiment of trajectory generation using the
force/torque sensor. The general structure of the model is composed by subsystems that enclose different functional part, as displayed in Figure 9.

![Diagram of Simulink model](image)

**Figure 9: General structure of the Simulink model used to generate code**

In details, the Stateflow diagram contains all the logic required to manage the model, the **Input drivers** and the **Output drivers** subsystems contain the blocks that read/write data from/to the robot, such as the block **Read positions** described in Section 4; the PID controller is situated inside the **Control algorithm** subsystem. The **Reference generator** subsystem contains, among others, the **Reference planner** block and the **Force control** block, as shown in Figure 10.

The inputs to the custom block described in Section 5 and shown in Figure 7 are the actual joint positions and the force-torque readings. Its output is the desired joint position.

Executing the RTW command **generate code**, a list of C files is created. These files are compiled on the RTAI-Linux PC and executed as a real-time process. It is now possible to launch the RTAI-Lab graphical interface to control the robot.

The numerical values of the control constants used in the experiments are listed in Table 1. For the set-point generation $K_f = 0.01 I_{3 	imes 3}$.

A screenshot of the graphical interface, taken while the application was running and the robot was moving, is shown in Figure 11. On the left side, the panel used to interact and change the parameters of the control process or the output scopes can be seen. In particular, the parameter
panel shows the maximal velocities and accelerations for the trajectory planner: \( v_{\text{max}} = 100 \text{ deg/s} \) and \( a_{\text{max}} = 150 \text{ deg/s}^2 \) for all joints. Those values could be changed at any moment and for each joint.

On the right side of Figure 11 two scopes are shown, representing, respectively, the \( y \)-axis force reading (down) and the \( y \)-axis cartesian position of the robot tool (up). In particular, the bottom scope shows the run-time force acquisition during a contact with the environment and, at the same time, the upper scope shows the planned trajectory that the robot is following to reach the desired position.

The scopes are automatically provided by the interface according to the RTAI scope-block inserted in the Simulink model.

### 6 Conclusions and future developments

This paper presents a rapid control and supervision prototyping architecture based on the Simulink-Stateflow environment. It automatically generates specific real-time C code for a RTAI-Linux based robotic systems, effectively replacing the original closed controller of an industrial 6DoF manipulator.

The goal is attained using a specifically built Simulink-Stateflow library, whose blocks have been created as S-Functions.

The C-code is automatically generated for the host PC target running under RTAI-Linux. The PC is connected to the robot controller using a simple card available from the manufacturer. The great advantage of such a solution is the increased capability to build a real-time application on an industrial robot without dealing with the intricacies of real-time programming.

An experiment is described: a force-torque sensor is used to generate on-line a cartesian tra-
jectory for the manipulator end-effector. All the controller parameters can be changed while the application is running from the RTAI-Lab graphical user interface, and relevant signals could be observed and/or acquired during the motion, for further analysis of the controller behavior.

At present several further improvements are under study, starting from the improvement of the GUI interface to the addition of vision sensors, both single camera or stereoscopic systems, and its integration with the force/torques sensor to produce enhanced trajectory generation capabilities and interaction with the environment.

Acknowledgments

This project took place under the research project MATRICS, funded by the Italian Ministry of Education, University and Research (MIUR).

References


