Essential Real-Time and Modeling tools for Robot Rapid Prototyping
Jason Michel Lambert, Brian Moore and Mojtaba Ahmadi
Opal-RT Technologies Inc.
1751 Richardson, suite 2525, Montreal, QC, Canada H3K 1G6
phone:514-935-2323   email: info@Opal-RT.com   http://www.Opal-RT.com

Keywords
Real-time simulation, hardware-in-the-loop simulation, robot, mechanical structure, modeling, simulation.

Abstract
Rapid prototyping is essential in robotics. The Waterfall method is usually used to design complex systems, but this method has some drawbacks. Rapid Prototyping is a more recent approach, which is more suitable for real-time systems design. We propose a set of three requirements in order to have a flexible real-time systems design environment. SYMOFROS, a symbolic modeling and simulation software developed by Canadian Space Agency is presented. A real-time simulation environment using parallel processors called RT-Lab is next described. Combined together, these tools provide suitable environment to do rapid prototyping, starting with a description of a mechanical structure down to the real-time simulation with a complex controller architecture.

1. Introduction
The ways and methods of doing robotics research and development have always been influenced by the tools used. This is especially true when one considers the profound impact of recent computer technologies on robotics. Not more than 15 years ago, computing cost was still a significant factor to consider when deriving algorithms and new modeling techniques [1,2]. The advent of workstations and personal computers gave a tremendous boost to robotics, bringing them ever closer to implementing them and using them in real-life situations. Complex simulations of robots and control systems became possible, and people started to think in terms of real-time implementation even at early stages of the project. Nowadays, distributed computing, network technology enabling close and efficient cooperation between members of a project group and the computing power developed by commercial equipment open new avenues for doing systems design and implementation. This paper presents an approach to robot nonlinear controller design and implementation based on fast prototyping of the system and short iterative loops. From the description of the approach, essential characteristics of a real-time fast prototyping environment are defined. We next present SYMOFROS and RT-LAB, which together constitute an ideal environment for doing rapid robot prototyping. The setup for a nonlinear controller design, optimization and real-time validation using SYMOFROS and RT_LAB is finally presented to illustrate the points brought about in this paper.

2. Real-Time systems design methods
When dealing with large and costly projects involving real-time implementation, engineers would generally apply what is known as the Waterfall design methodology. This methodology focuses on breaking down the overall process into clearly defined sub-modules arranged in a dependent, sequential way. A group of highly specialized experts would typically be assigned to each stage of the design so that once a stage is complete, it is final; minimum feedback was planned in the process, so that all steps cascade into the final, implemented system, hence the term “Waterfall”. The Waterfall method applied to design of a real-time controller can be represented by figure 1.
Figure 1  The Waterfall Approach

The Waterfall design process is an approach which has successfully been used over several years. However, it holds some disadvantages, which mainly come from its lack of flexibility. Firstly, there is a strong dependency over numerous specialists. A problem in the control system may happen at any of the above-mentioned stages and only the people involved in this stage can identify the problem, let alone fix it. Another disadvantage resides in the lack of feedback loops in the cycle. As mentioned above, once a stage is complete, going back to that stage for re-design can be a lengthy and costly task. However, due to a complex implementation method, control systems designers were restricted to using tools which did not fully reproduce a real-time environment. The actual implementation process inevitably modifies the original system design. One must try to incorporate as many factors present in the real-time system in designing the controller. Also one should be able to easily go back to the original model at any stage in the design process, make modifications and see the effect of these modifications in the real-time environment. Finally, documentation and maintenance of the system developed will also be dependent on many different sources that may not be available at the time. An upgrade of some modules may require complete re-coding.

Figure 2  The rapid prototyping approach

As can be seen from the drawing, automation of the coding and implementation of the originally designed model gives more freedom to the robotics system engineer to address issues which may arise further down the design loop. Such an approach removes the software specialists from the critical stages of the cycle. They assume a more supporting role, giving assistance when required. Another beneficial side effect of rapid prototyping is that, by automating the steps needed to implement a control system model in a real-time environment, the original model becomes a better representation of reality. The model becomes a plan, describing its structure as well as its mathematical behavior. This greatly eases the necessary burden of documentation and maintenance; a well-documented model can be passed on from one engineer to the other in a comprehensive form.

2.1 Rapid Prototyping applied to robot controller design

The problem of designing a robot controller is one that involves several iterations on the system, and this at all implementation levels. The steps involved in designing and implementing a robotic control systems can be represented by the following diagram, adapted from Pahl-Beitz[3].

Figure 3  The systems approach applied to controller design
2.2 System studies/goal programme

The first stage is the problem definition. At this level, the task to be accomplished, the type of environment and the type of manipulator employed are assessed. In general, the robot controller design problem can be stated as: “Given a task to execute, find the control systems characteristics, in terms of architecture and parameters which will guarantee adequate performance expressed as a function of those parameters”. The first two stages are concerned with defining “task” and “performance” so that some form of optimization scheme can be formulated.

2.3 System synthesis and analysis

The next two stages deal with describing and analyzing the system under study. Models of the plant, the environment and of one or a series of controllers are produced at this stage, using more or less dedicated modeling tools. It is important that the simulation scenario be representative of reality from the control system point of view. This implies designing key trajectories and a model of the environment which will excite the controller so as to maximize its robustness while still giving a good overall performance. Referring to the design steps in the previous section, simulation either takes place offline, online, online with hardware, or a combination of the three in one integrated environment[4, 5].

2.4 System evaluation, decision and implementations plan

Once appropriate models have been derived and implemented into a simulation environment, the control system’s parameters have to be optimized against the performance criteria. Depending on the type of controller used (PID, model-based, Cartesian…) different techniques may be employed. However, in general, nonlinear controllers do not have design techniques like their linear counterpart. One technique which has been used effectively aims at optimizing simulation results which are based on previously-defined performance criteria [6, 7]. It is essentially an optimization problem, where the parameters are optimized using simulation instead of a function.

A decision is then made on the best solution that was found. This decision is taken based on reports and results of the analysis and evaluation stages. Final touches to the controller are then made and plans for next implementation stage are made.

This last point brings us to a consideration of the robot controller design scheme as it is realized either using the Waterfall approach or the Rapid Prototyping approach. In the Waterfall approach, the control design scheme is repeated at each stage, but with different concerns. The system must be analyzed and evaluated based on its real-time performance. Although this scheme has the advantage of simplifying the overall problem into sub-problems, the issues of specialist dependency and reduced inter-stage communication mentioned in the previous section puts the decisions undertaken at each stage at risk. This may result in changes affecting the original system design, or methods of implementation which limits the flexibility of the system.

When realized using the Rapid Prototyping approach, the controller design scheme is distributed across the whole cycle. The problem is defined once; in its definition are included all issues related to real-time implementation and testing. As the project goes along, the controller design steps are performed right down to real-time prototype validation. For example, Referring to the steps proposed above, system evaluation should be performed both on the offline and the real-time. Most importantly, however, the user should have control over the whole implementation process and be able to iterate back and forth during the process (notice the many feedback loops inherent to the systems approach). Those loops are fundamental to the method proposed in this paper. The offline/real-time simulation environment should reflect this important consideration.

3. Requirements for a Robotic real-time system simulation environment

Many commercially-available tools exist which will allow us to perform each of the steps in our controller design process. However, these tools will be aimed at fulfilling only parts of the task explained above. Few of these tools address the iterative design process as a whole, especially when dealing with robotic systems. For example, a change in the environment may necessitate a modification of the simulation scenario and a re-evaluation of the controller gains. System identification may also require refinement of the simulation parameters and a re-evaluation of the gains. Consequently, there must be a notion of integrated offline and real-time systems design environment in which the user can go through all the steps described above. However, “no simulator is an island”, so self-containment and homogeneity are not the only requirements for such an environment. Some key features of such an environment are discussed below.
3.1 Ease of use

In the 90s, software has become more and more easy to use. This was and this is still one of the major issue in software development for several reasons:

- More contractor, worker are nomad, they change often from job to job ⇒ Should learn fast, cannot invest in people that will change job in a few months.
- Have to do everything faster, delay are short, tight schedule
- Frustration when it's long to learn

Most of the general-purpose software have reach this goal. However, specialized software are often more difficult to use and required a good knowledge.

This is especially true in the context of Rapid Prototyping. Many steps are involved which require a certain degree of automation. The Rapid prototyping cycle presented above typically holds three simulation levels: offline, real-time and real-time with HIL. In traditional approaches, these three stages are treated independently, with different methodologies. Offline simulation, for example could be done with a dynamic simulation package such as Simulink, while real-time testing would be done using manual coding, e.g. C. Finally, implementation could take place on an entirely different platform. This results in the final controller being much different from the original model, thereby making the gains thus calculated less relevant. It is important, while doing fast prototyping that the offline, online and HIL simulation engines yield the same results. Of course, the offline and real-time simulations should represent the physical system, and not vice-versa. The offline designer should then be aware of the changes brought about by implementation in a real-time environment (I/O delays, time constraints, fixed time-step…) and the design environment should provide means to address such issues.

Rapid Prototyping may appear simple in principle, it holds many sub-stages, which are accessed bi-directionally, at anytime in the design process, and by different participants in the project. Results from all simulation stages should be cross-verifiable throughout the process. The engineer has to understand the inner workings to some extent, in order to configure the system and investigate problems. The complexity of a real-time system is such that great care must be taken in making the environment user-friendly. Concepts such as transparency, proper initialization procedures, ease of access to the model at key points in the implementation stage must be present in the design environment.

3.2 Reconfigurability

Reconfigurability and openness are features already recognized by many as essential in real-time systems [4,8]. Although several robot manufacturers are moving towards black-box designs with modules such as “smart” actuators and embedded, dedicated control systems, the needs of high-end robotic research and academia are different. Not only is it important to have easy access to the system at all levels (e.g. from high-level supervisory control all the way down to fast servo loops at the actuator’s electrical level), real-time control over the simulation’s parameters is paramount. Reconfigurability should also be reflected when more fundamental changes to the controller architecture are required, in the necessity of quickly being able to make modifications in the original design and verify the effect of these modifications on the prototype. Automatic real-time code generation and compilation, real-time control over the model parameters and automatic I/O driver integration are all features which contribute to quickly being able to “navigate” between different implementations. Also, from our problem definition above, the user should be able to quickly modify the architecture of the controller without having to alter the simulation system itself.

On a more managerial level, even though the user needs an environment into which he can smoothly go from modeling to Hardware-in-The Loop simulation, he may be required to interact with teams from other departments and be compatible with other tools. Designing and implementing a robot controller is an elaborate task which typically involves more than one robotic system engineer at work on the system in a parallel fashion. Hence, the environment must facilitate access to the models, controllers, simulation etc. by the members of the design team. Furthermore, to ensure consistency at the time of implementation, the same simulation engine should be used by all participants in the project. This suggests that simulation resources should be shared between the designers, in a server-client configuration, for example. The same situation arises when the system is monitored by more that one person. Control can be passed from one to the other, and displays distributed over several stations.

3.3 Hard-real-time capabilities

Properly designed and controlled robots are the most precise mechanical systems to this day. In addition to good kinematic and dynamic designs, the robot’s controller must be implemented as a system which can meets critical time constraints, both from software and hardware components. Nonlinear controllers are
generally complex and require a lot of computing power at a fast rate. Force-based controllers can be run at rates of up to 10 000kHz. Distributed systems can dramatically improve performance [9]. Controllers may also be run at multiple rates, which requires additional scheduling. Such factors require software capable of managing considerable memory space, running in an operating system where memory protection and low context-switching time are the main concerns.

On the hardware side, real-time communications between the computing nodes and the equipment must answer to the same time constraints mentioned above. Additional communication networks are also required for user interaction during real-time operation. Finally, fault-tolerance is a factor which needs to be considered right from the start of the design. The main motivation behind a fault-tolerant system is quite self-evident. Rapid control prototyping, as its name implies, is aimed at generating controller prototype. The prototype aspect brings a certain level of risk when performing Hardware-In-the Loop simulations. A hardware real-time robotic system is deceitfully complex, and precise and costly equipment can be damaged due to anything from violation of real-time constraints due to overruns to bad GUI design. Fault-tolerance, a concept first used in aerospace, guarantees critical system state detection both at the controller implementation level (actuator saturation, singularity detection, watchdogs…) and the hardware level (redundant processes and communication, additional hardware in the loop backup), procedures to override them and means to provide real-time support should a problem occur.

4. SYMOFROS

SYMOFROS is a modeling and simulation software for mechanical structure developed by the Canadian Space Agency for several projects, including STVF [4] and BORIS. Modeling and simulation of rigid, flexible and closed-loop mechanical structures are possible. SYMOFROS uses MathWorks product (MATLAB, Simulink, RTWorkshop) and Maple 6. SYMOFROS can be divided in five modules as shown below.

4.1 Model editor

First, a model is created using a graphical interface representing the mechanical structure:

The representation is flexible and a lot of options are given to the user. Using rigid and flexible body, the user describes the topology of the mechanical structure and orientation and position of each body relative to the previous body. Then, characteristics for each body are given from the kinematic description like the length of the body, to the inertial parameters description like the inertia, to the external forces. Joints are also define in each body to define the degree of freedom. Other settings are finally set like modeling options. On a simple click on the Export option, topology files describing the structure are created. These files are ASCII file accessible and modifiable by the user.

4.2 Symbolic Model Generator

The Symbolic Model Generator module uses the ASCII files to generate a .c file with 58 SYMOFROS functions. SYMOFROS functions are .c numerical procedure used to obtain basic information on the model, like the inertia matrix, Jacobian matrix, position of the end effector.

The symbolic Model Generator is divided in two sub-modules. The first module uses the description of the topology and other information to generate purely symbolic SYMOFROS functions. Then, numerical values are replaced in the equations and optimization for real-time simulation is applied on the equations to group SYMOFROS functions in groups of functions. For a group of SYMOFROS functions, common expressions to at least two functions in the group will be computed only once by time step. These groups can be customized by the user.

4.3 Non Real-time Shell Environment

Using the .c file or .dll file obtained by compiling the model .c file, the user can obtain values for the SYMOFROS functions depending on inputs given by the user. The inputs of the function usually are the parameters, states, accelerations, motion, internal force and external forces, depending on the SYMOFROS function used. In SYMOFROS, an environment is given in MATLAB to query the model, but the user could create another environment (like a .c environment) or use other existing tools.

4.4 Non Real-time/Real-Time Simulation Environments

In SYMOFROS, Simulink is used to do simulation. In Simulink, The SYMOFROS model functions are implemented as Simulink blocks which constitute a library. Moreover, higher-level blocks are also included in the toolbox which allow for complex functions to be readily implemented in the simulation. Functions like the
inverse kinematics, direct dynamics, controller components (PIDs, feedback linearisation...) and other. New blocks can be created using basic SYMOFROS functions block in the library, so a user could create its own library of controllers for example. SYMOFROS is also fully compatible with rtw and RT-Lab, thereby allowing SYMOFROS models to be run in a real-time environment.

Although SYMOFROS is a really complex software, with huge capabilities, it is really easy to use. SYMOFROS is also really easy to learn because there are only a few interactions with the user.

5. RT-LAB

RT-LAB is a distributed real-time systems design environment. Developed by Opal-RT technologies, its main purpose is to supply a COTS-based environment for fast prototyping of control systems which can directly interact with hardware equipment. The performance of RT-LAB is such that it can handle extremely tight real-time constraints, of the order of 25?s for single-node simulations and 100?s for distributed simulations. Its main feature is the complete automation of the integration process from block-diagram graphical representation down to distributed hard real-time execution.

5.1 Model Definition

RT-LAB uses either of the popular MATLAB/Simulink or MATRIX?/SystemBuild tools packages as a front-end for model definition and viewing. The defined models become the source from which code can be generated, manipulated and downloaded onto target processors for real-time or distributed simulation. From the designer’s point of view, everything between the graphical model and its real-time counterpart is transparent. The original block diagram serving both as a tool and its documentation can be the main focus of the designers since the rest below is generated automatically.

5.2 Automatic integration in a distributed, hard real-time environment

RT-LAB code “understands” distributed, real-time simulation, and automatically takes care of setting up all the communication, including real-time data exchanges between processors and interfacing with the real-time operating system. This leaves the engineer free to work on the already difficult task of creating a proper mathematical model. Since RT-LAB supports distributed computing, complex simulations can be subdivided for execution on separate processors. The user only needs to graphically subdivide the models and add the inter-processor connection blocks supplied with RT-LAB. The model is then automatically separated and its code generated with the appropriate communication and I/O drivers. The code is transferred in the real-time environment and compiled. QNX and Neutrino are the two currently supported operating systems for hard real-time implementation.

![Figure 4](image-url) A typical RT-LAB hardware configuration
The real-time simulation executes on a distributed system of standard PCs. No custom DSP, or proprietary boards are utilized.

Inter-PC communication options range from low-cost UDP/IP on Ethernet at 100Mb/s, through FireWire and OHCI universal drivers at 400Mb/s to high-performance cLAN at 1.2GB/s on Windows NT and 2000. Extra processor nodes can be added to the existing configuration and the model can be redistributed at the graphic level.

5.3 I/O integration

I/O interfaces are configured through custom blocks, supplied with RT-LAB. The blocks are added to the graphic model and input/output signals are sent from the model to I/O devices that can be ISA-, PCI-, or VME-compatible. These in turn are connected to the actuator drives for output and to the encoder, thereby closing the system loop.

5.4 Execution and real-time parameter control

RT-LAB supports three execution modes: as-fast-as-possible, soft and hard real time. In the two latter cases, execution time of each simulation step is slaved to a timer from the CPU clock or from a hardware timer board. Within both of these methods RT-LAB provides the interface to start, stop and pause the execution, as well as stepping through the execution.

It is not unusual for HIL simulations to require sub-millisecond hard real-time step sizes, and RT-LAB as been used in hard real-time applications with step sizes as small as 22 microseconds. The user has access to the model signals and parameters via a TCP/IP link. These signals can be displayed on the command station at refresh rates of up to 200Hz. Synchronized input commands to the model can also be sent for human-in-the-loop interaction. Real-time recording on the cluster of signals can also be triggered by events from the model. The entire model’s parameters can be monitored from the command station and execution of Monte-Carlo simulation sets can be automated by a sequencer. A snapshot feature allows for recording the system’s state at a given moment and restoring that state later on.

6. Case Study: Design of a crs controller for position-control applications

The approach and design environment presented above will now be demonstrated in the context of designing a controller for interaction-free operation of a CRS-A465 robot. It is a fairly straightforward and known problem, so the steps are fairly well-established. Only the models and system setup are presented here; results and performance evaluation will be part of a second paper.

6.1 Controller Model Library

Four controllers will be tested and their performance compared. The controllers are implemented in the SIMULINK graphical programming software for offline testing and automatic implementation in RT-LAB. They make extensive use of blocks from the SYMOFROS blockset, which makes the controller models reusable with any other SYMOFROS-generated model. Being generic in this sense, the controllers are regrouped in a SIMULINK library which eventually can itself become a toolbox.

Figure 5 Controller Library

Two joint-space controllers and two cartesian-space controllers are being tested.

- Joint-space PID-Control
- Computed Torque Control
- Cartesian PD Control
- Cartesian Impedance Controller

Implementation of the Cartesian PD Controller is shown on figure 6 below. Extensive use of the SYMOFROS functions have been used in creating the model, both for the plant and the controller. Test trajectories are also generated and implemented as a custom-made Simulink library.
7. Conclusions

Tools for the Rapid Prototyping of robot control systems were presented in this paper. The first section introduced the Rapid Prototyping approach, and how it can solve problems inherent to the more classic waterfall approach. Next, the problem of creating a robot controller was defined, and the steps discussed in the context of the systems design approach. Finally, this control systems design approach is analyzed as it is either applied using the Waterfall or Rapid Prototyping design schemes. The next section derives essential characteristics for an environment in which Rapid robot Prototyping takes place. In the light of these characteristics and requirements, SYMOFROS and RT-LAB are presented as tools for doing robot modeling and real-time simulation. Their respective description shows that, when combined together, SYMOFROS and RT-LAB constitute an ideal environment for real-time systems design. Finally, the system configuration for a case study of robot controller design is described. Further work will include creating additional control components for real-time and HIL simulation and defining the optimization problem. Results of the optimization will be presented in upcoming publications.

Their respective description shows that, when combined together, SYMOFROS and RT-LAB constitute an ideal environment for real-time systems design. The graphical programming environment provided by SIMULINK and the reusability of the SYMOFROS function blocks sped up the setup of the controller models by an order of magnitude compared to using manual coding. More important, though, the transparency of the model generation and real-time integration process leaves more thought space to the engineer creating the system. By automating the major integration steps, extra-time can be spent on the design itself, leaving more room for imagination and innovation.

References


