VFAST Transactions on Software Engineering
http://vfast.org/journals/index.php/VTSE@ 2015 ISSN(e): 2309-3978;ISSN(p): 2411-6246
Volume 3, Number 1, January-December, 2015

pp. 01-12

LINUX-XENOMAI TARGET: A REAL-TIME HARDWARE-IN-THE-LOOP SIMULATION FRAMEWORK BASED ON SIMULINK

EGEMEN CUMHUR KALELI[1], ERKAN ZERGEROĞLU[1]
[1]Department of Computer Engineering, Gebze Institute of Technology
[ekaleli,ezerger]@bilmuh.gyte.edu.tr

ABSTRACT. *Hardware-In-The-Loop (HIL) simulation has an important role on rapid prototyping and developing control software for various application areas. Having deterministic time response and high computational power, a personal computer (PC) with a real-time operating system is an efficient platform solution for HIL simulation needs. Fast development time and cost efficiency of simulation platform are other important issues for reduction of total production lead time and overall cost. In this paper; taking into consideration all these aforementioned aspects, we propose a HIL simulation software framework for PC target. Building a MATLAB/Simulink diagram and using available device drivers, an ordinary user can develop and execute HIL simulation of his/her controller. The proposed framework is developed for PC platforms that have Linux Xenomai real-time operating system infrastructure. The framework contains a Simulink Xenomai target control module, a Graphical User Interface (GUI) and a Simulink device driver library. Also, to illustrate the performance and feasibility of the proposed framework, a Xenomai real-time driver model (RTDM) based driver for Quanser Q8 data acquisition board has been implemented. Experiments performed on servo control of a two degree of freedom (DOF) robot illustrated that proposed framework meets hard real-time requirements*
Keywords: Control; HIL Simulation; Hard Real-Time System.

1. **Introduction.** The use of a hardware-in-the-loop (HIL) simulation software environment in the testing of design and development of a control algorithms for various systems is a common approach among controller designers. HIL simulation of a dedicated control algorithm for a specific system is useful as it enables the rapid prototyping realization providing fast development time and cost reduction, and also testing a control algorithm before the end-product stage to minimize malfunctioning risks. Unfortunately, to satisfy all those requirements a general purpose HIL simulation software running on a PC with a deterministic time response and hard-real time control accuracy is not sufficient. The infrastructure should also support at least one data acquisition hardware that is preferably wide-spread and well-known for control applications, should be capable of on-line tuning of plant control parameters and monitoring plant outputs, and should use processor power efficiently. On top of these it should be easy to learn, well-documented and user friendly so that development time of a test application is decreased, should be able to be updated and should be low-cost. HIL simulation software package and the platform on which the software package runs should ideally own following components to satisfy aforementioned qualifications: a real-time operating system (RTOS) with application and device driver development framework, a set of complete controller algorithm implementation tools and a graphical user interface (GUI) software tool that is functional and user-friendly.

In literature solutions for a real-time application environments, both commercial and free, are available. QNX [1], and VxWorks [2] are some examples of commercial RTOSes. Commonly used operating systems like Windows and Linux are not designed to meet a RTOS specifications. However via proper kernel patches or add-ons both can have, up to a point, real-time capabilities. Linux, being our choice due to the vast free applications and friendly open source community, via proper RTOS emulation frame work kernel patches can become a real-time environment. Xenomai [3] and RTAI [4] are the most common RTOS emulation frameworks for Linux. These frameworks aim at providing a consistent environment that helps implementing real-time interfaces and debugging real-time software on Linux. On the other hand, these frameworks were designed with the goal to help application designers using traditional and proprietary real-time operating systems to move as smoothly as possible to a Linux based execution environment.

MATLAB/Simulink with Simulink Coder environment is perhaps the most-widely used commercial simulation tool providing nearly complete algorithm implementation tools and an internal GUI. It provides support for some proprietary real-time operating systems and hardware so that a user can have complete HIL simulation software package. Some of HIL simulation hardware manufacturers such as dSpace [5] supports Simulink/Simulink Coder tools to implement control algorithms and to generate executable code for their specific target hardware. 20-sim 4C [6] is another commercial HIL simulation software package that resembles Simulink/Simulink Coder. It generates code for proprietary hardware that runs Linux-RTAI, however the algorithm implementation tools and GUI are used under Windows environment. Quarc [7], supplied by Quanser Consulting, is another HIL simulation software package totally integrated with MATLAB/Simulink providing device driver support and various toolboxes to control proprietary hardware. RTAI-Lab [8] is one of the real-time simulation software package that needs an algorithm implementation and a code generation tool such as Simulink/Simulink Coder or Scilab/Scisos [9]. Scisos is a Simulink/Simulink Coder like simulation program that has capability of generating C code from Scicos models and building hard-real time control executable with RTAI.

Several HIL software tools have been proposed in literature. In [10], authors introduce a control software package that is totally integrated with MATLAB/Simulink and works under RT-Linux, a proprietary RTOS environment. Providing graphical components and environment to design controller, Simulink like packages reduces development time of implementation process. Eventual product of this process is a model diagram that needs to be transformed into C/C++ code. Simulink Coder and Scicos generates this code and a corresponding executable. Another approach is using a programming language (C/C++) as an implementation tool so that requirement for a code generation tool is eliminated. In [11] such an approach is used. In [12] introduces another real-time control system based on Simulink and RTAI for a target embedded industrial computer where the communication between host computer and target is provided by using TCP/IP sockets. Host computer is a Windows system with MATLAB/Simulink and an implemented GUI. The GUI has components to monitor model signals coming from the target and to tune model parameters. This approach resembles dSpace's in that dSpace uses its own GUI software: ControlDesk. [13] introduces a software package named Xenomai-Lab that emulates the RTAI-Lab/SciCos package for Xenomai.

In this paper we propose a framework that has been developed for PC platforms running on Linux-Xenomai real-time operating system infrastructure. The framework contains a Simulink Xenomai target control software module, a GUI module and a device driver library dedicated for Simulink. The framework utilizes MATLAB/Simulink/Simulink Coder as algorithm implementation/code generation tool and Linux-Xenomai RTOS as real-time execution environment. Also we implemented our GUI to monitor signals of Simulink model and to tune Simulink model parameters on-line. We have chosen Simulink due to its common usage in industry and in academy and it is better documented compared to its free alternatives. On the other hand, Linux-Xenomai is well documented, better structured than RTAI and valid alternative for VxWorks [17]. We aimed at combining usability of a proprietary software, advantage of using a free RTOS and flexibility of a custom GUI.

The rest of the paper is organized as follows: in Section 2 we describe software architecture and design considerations; Section 3 describes device driver library; Section 4 describes performed servo control

experiment on a two-link planar robot; in Section 5 we discuss experimental results and conclusions from the points of both servo control performance and real-time execution performance.

2. **Software Architecture**. The proposed software architecture uses source files generated for Linux Xenomai target. The term *target* means an operating system or a processored device on which the generated and built code runs. Simulink Coder has no built-in support for the Xenomai target. We implemented required system target file and template make file so that Simulink Coder can generate files for the target needs and make utility of MATLAB/Simulink can build a Xenomai target-specific executable.

The architecture is based on client-server mechanism schematically shown in Figure 1. The *Linux Xenomai Target Control Module* part is the server. It listens to requests of the client and manages execution flow of Simulink model in order to respond to the client's requests. The client side, *GUI Module* part, sends runtime commands and model parameter inputs to the server. The detailed description of each part is given in the following sections.

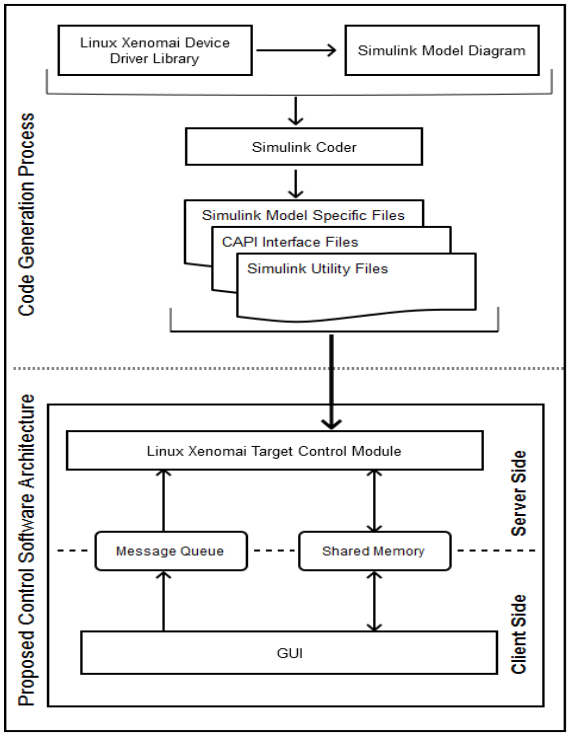

Figure 1. Software Architecture and Code Generation Process

2.1. **Xenomai Target Control Module**. The control module consists of tasks that handle basic Simulink model execution, communication with the client, logging signals of the model and updating of parameters of the model. Figure 2 shows these tasks and their functions in the module architecture. *Main task* is a soft real-time thread of execution which is responsible for creation of shared memory using native Xenomai memory heap interface, partial initialization of model data structures, creating and starting user interface task (*UI Task*) and handling signals of Linux operating system. *UI Task* is a real-time thread which handles GUI commands, sets model parameters in runtime via C API, which is a Simulink Coder interface and will be described in Section 2.1.2 in detail, and properly terminates allocated memory for the model. *Base task* is another real-time thread that is the core of the model code execution. In runtime, it updates model data in C API structures and logs model signals, whom it reads from C API, into the shared memory. Each task have been affined to a

separate CPU using native Xenomai task management interface to make sure that no task intervenes each other and consumes common CPU power. Following this method, we aimed at obtaining decrease in execution duration of *the base task* and other tasks as much as possible. The detailed description of each task is given in the following sections.

2.1.1 **Main Task.** *Main task* is the entry point of the control module for initializations. Some of model initialization functions are called in this function. It creates shared memory blocks for the model parameters, signals and control variables such as model step size, simulation duration, signal logging frequency. Memory of control variables are initialized with read values from the model data structure. It can allocate memory locations up to 256 model parameters and 64 model signals. Each model signal can be logged up to 1 million data points as default. The parameters and signals can be organized in scalar or vector data structure. One of the most important operations of this task is to create and start *UI task*. Starting the *UI task*, *main task* goes into waiting mode and stay in this mode until other tasks finish their execution or "CTRL+C" program termination signal. *Main task* deallocates all shared memory locations, deletes Xenomai heap structures and the model memory just before the program is terminated.

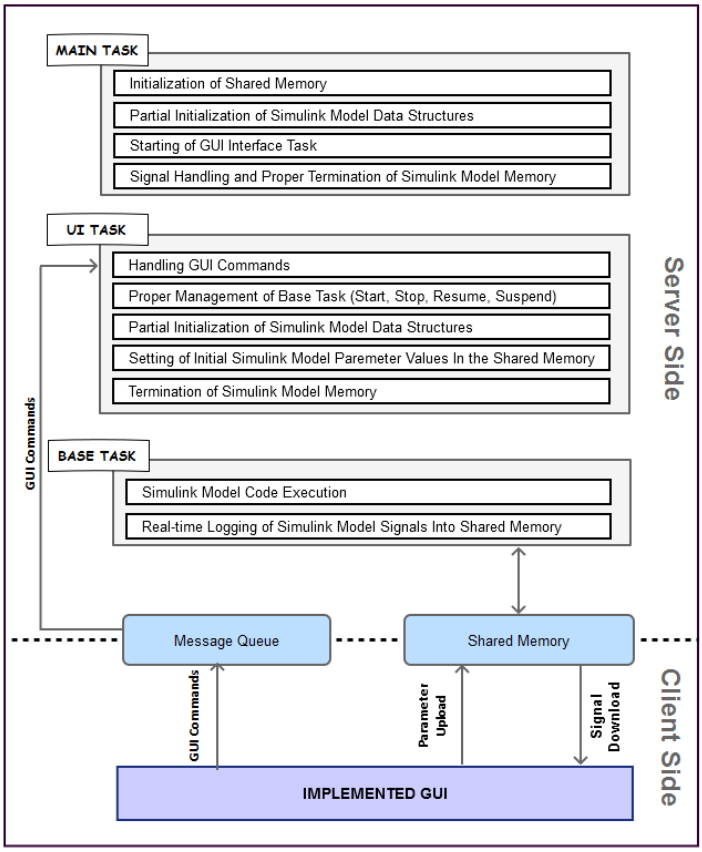

Figure 2. Tasks of Linux Xenomai Target Control Module

2.1.2 **UI Task.** *UI task* is started by the *main task*. Initially, *UI task* tries to connect to message queue that is expected to be created by the client GUI program. If the connection is successful, other initialization functions are called so that initialization of Simulink model data structures is completed. Then it sets shared memory of model parameters with initial values read from C API(C Application Program Interface). C API is a collection of source files in C language which provides data structures and macros to access these structures. The C API provides programmatic access to root-level inputs and outputs. This allows a user to log and monitor the root-

level inputs and outputs of a model while the user runs the code which is generated for the model. After all those initializations are accomplished, *UI task* goes into a periodic loop in which Simulink model parameters are updated with values read from the shared memory. Also client commands are listened and handled in the periodic loop. A user can send four types of command to the *UI task: start, stop, pause and terminate.*

If the command is *start, UI task* either starts or resumes the *base task* depending on status of the *base task*. In the case of that command is *stop, UI task* stops the *base task* then exits from periodic loop. Subsequently, the *main task* restarts the *UI task* or finishes overall model code execution according to received user command. If the *UI task* receives pause command, it simply suspends the base task. In the case of the user command is *terminate,* which means "terminate the program", instead of restarting the *UI task,* it accomplishes operations mentioned in the last part of Section 2.1.1.

2.1.3 **Base Task**. *Base task* contains a real-time thread of execution. This thread accommodates a periodic loop that calls a generated model function calculating and setting model outputs. Period and duration of the loop is set to "step size" and "stop time" of the model respectively before the loop begins to run. The loop also logs signal values of the model into shared memory and checks whether overrun happens at each loop cycle. If an overrun happens the task exits from the loop printing a warning message on a Linux terminal. Execution of the task can also be ended by the *UI task*. However, when "simulation stop time" is elapsed, the *base task* exits from the loop ending the execution.

2.2 **GUI Module.** GUI module consists of source files that implement *handler task, rtgui task* and various graphical control components: *main console, file browser, model manager window, parameter window, scope* and *seven-segment display.* The module has been implemented using FLTK 2.0 cross-platform C++ toolkit [14] and openGL graphic library. When a user runs the GUI program, *main console* is constructed and becomes visible on the screen. The console has various graphical control buttons that allows a user to run *file browser, model manager window* and to terminate *Linux Xenomai Target Control* (server) program which executes Simulink model code. *The Main console* also initializes Xenomai heap structures, tries to connect to the shared memory which is expected to be available when the server has started to run. Interacting with a graphical control component labelled as "Terminate", a user can terminate all of running GUI and server tasks, can close all of open windows related to Simulink model. Using the *file browser*, a user can navigate file system on PC and run server program. *The Model manager window* implicitly creates the message queue to which the server connects. The manager has graphical control buttons which allow a user to create *parameter window, scope window* and *seven-segment display*. It also has other buttons which allow a user to start, pause and stop the model simulation. The manager window utilizes the message queue to deliver these commands to the server. A user can adjust *down sampling rate* from 1 to 100. This rate determines logging frequency of signals of a Simulink model.

Parameter window is another GUI component that consists of numeric inputs by which a user can tune model parameters. Editing these inputs, a user updates related shared memory area so that the *UI task* of the server sets Simulink model parameters with new values through C API.

Scope window and *seven-segment display* are used to plot and monitor model signal values. *T*he *Scope window* reads values of model signals and renders pixels corresponding to these signal values. This component uses FLTK and openGL tools to redraw plotted graphics on the PC screen. Default redrawing period is 50 milliseconds. Besides, *scope window* allows a user to export logged signal data to a file that can be opened by MATLAB.

*Handler tas*k is a soft real-time thread that handles generated GUI events and redraws GUI components. Those events happens when a user interacts with graphical controls. Event handling process involves actions

such as applying model parameter changes into shared memory, constructing GUI components, starting *rtgui task* and plotting 2D graphic values of model signals with respect to time.

Rtgui task is a real-time task of which main job is sending stop command to the server when model simulation time is elapsed. Other roles of the task are updating seven-segment display and progress bar component which graphically shows elapsed simulation time. The task accomplishes these jobs in a periodic loop. Default period has been set as 1/30 second. Execution management of rtgui task is handled by handler task according to user-generated events

3. **Device Driver Library**. The designed device driver library consists of a number of Simulink blocks: *Analog/Digital Converter (ADC), Digital/Analog Converter (DAC), Encoder, PWM* and *Counter, Digital Output*, and *Digital Input*. The Figure 3 shows the library. Each block has been implemented through a C source file and a TLC (Target Language Compiler) file pair. C source file defines I/O structure and passes device configuration parameters to the TLC file. TLC is a tool which is used to generate code from user-defined Simulink blocks. Users can replace their own device driver function which is implemented in C language into corresponding TLC file we implemented.

Users can create the Xenomai target Simulink model by using the same methodology as that used in constructing a simulation. At the MATLAB start menu or Simulink library menu, the user accesses the driver library block set by clicking "Linux-Xenomai Target" option. Through the use of pop-up menu, this command allows the user to drag the desired block into the Simulink model and then make the necessary connections. After the user has placed the I/O blocks in the Simulink model, the user can edit the I/O block properties by double clicking on the block. For example, the user can set or edit the input channel that the data is read from a hardware device.

ADC block provides the data available at the specified *ADC* channel to the Simulink model. Analog to digital converter property sheet allows the user to select input channels. The channel specified should be available on the used I/O board. If an invalid channel is specified, then channel 0 will be used (this convention applies to all of the I/O blocks). *DAC* block writes data from the Simulink model to the specified *DAC* channel. Similar to the *ADC* block, digital to analog converter property sheet allows the user to select conversion range: 0...5V, -5...5V, 0...10V, -10V...+10 and the output channel(s) on the board that corresponds to the location of the output data. *Digital input* and *digital output* blocks write and read data, respectively from the Simulink model to the specified digital channel. The property sheet of the *digital input* and *digital output* blocks are structured in a way that is similar to *DAC* and *ADC* blocks.

Encoder block provides the data available at the specified encoder channel to the Simulink model. A user can adjust the I/O board encoder parameters such as quadrature/1X mod, binary/BCD, e.g. However, a separate *Reset Encoder Block* we implemented must be used for resetting the relevant encoder channel.

PWM generator block enables the user to get square wave output at specified frequency and duty cycle. However, counter output block provides square wave output at a constant frequency. We implemented a real-time device driver for Quanser Q8 data acquisition board using Xenomai RTDM [16] interface. This interface allows the driver developer to use a structured methodology during implementation process. The driver consists of a kernel module and a user application program interface (API). We placed API functions into the designed Simulink device driver library so that Simulink models using this library can access Quanser Q8 hardware in real-time.

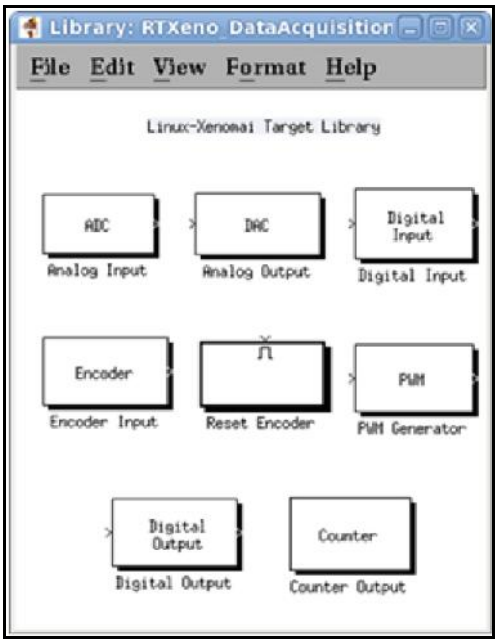

Figure 3. Linux Xenomai Target Device Driver Library

4. **Experimental Results.** The real-time simulation framework that is discussed in this paper has been applied to HIL simulation of a planar robot with two joint and two link. The robot is illustrated in Figure 4. It has direct current (DC) motors on its each joint. Each motor is driven by one linear power amplifier circuitry. The host computer we used is a personal computer with Intel Core(TM) Quad Q9400 with 2.66GHz. CPU. Quanser Q8 data acquisition board is connected to PCI expansion bus of our computer. Real-time software package is composed of Fedora14 Linux, of which kernel version is 2.6.38.8, Xenomai 2.6.0 and MATLAB 2011b.

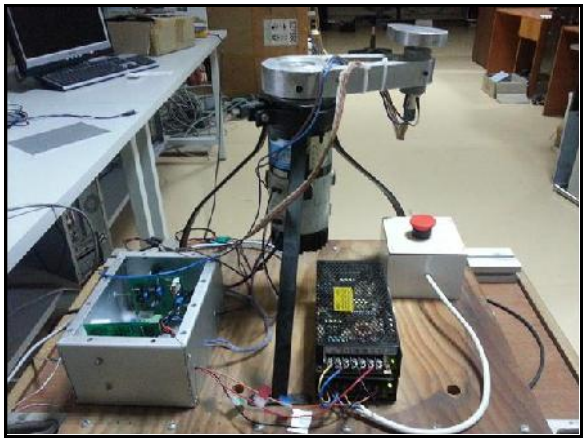

Figure 4. Two-Link Planar Robot Used In the Experiment

Our plant is shown in Figure 5. It is consisted of PC, Q8 board 2 DOF planar robot, power supply and power amplifier circuitry that drives two dc motors which actuates robot motors. Each motor has an encoder on its shaft. Encoder-1 produces 4096 pulses per revolution and encoder-2 produces 1024 pulses per revolution. We used *DAC* outputs of the Q8 board to generate control signals for each power amplifier circuitry of the link

motors. DAC0 and DAC1 has been connected to power drivers of joint-1 and that of joint-2 respectively. We used Encoder0 and Encoder1 channels of the Q8 card to track actual positions of the joints. Configuring the quadrature (4X) mode on encoder, we increased tracking resolution.

Figure 5. The Plant on Which the Discussed Framework Applied

Controller algorithm which was applied to the robot is an observer based desired compensation adaptation law controller (DCAL) that is an extension of the controller proposed in [15]. The algorithm had been implemented as a C MEX S-function in Simulink environment. We built a Simulink model using aforementioned algorithm block, the built-in Simulink library and our driver library to control the robot. The Simulink model we used is illustrated in Figure 6. We set Simulation duration and step size 300 seconds and 0.25 milliseconds respectively.

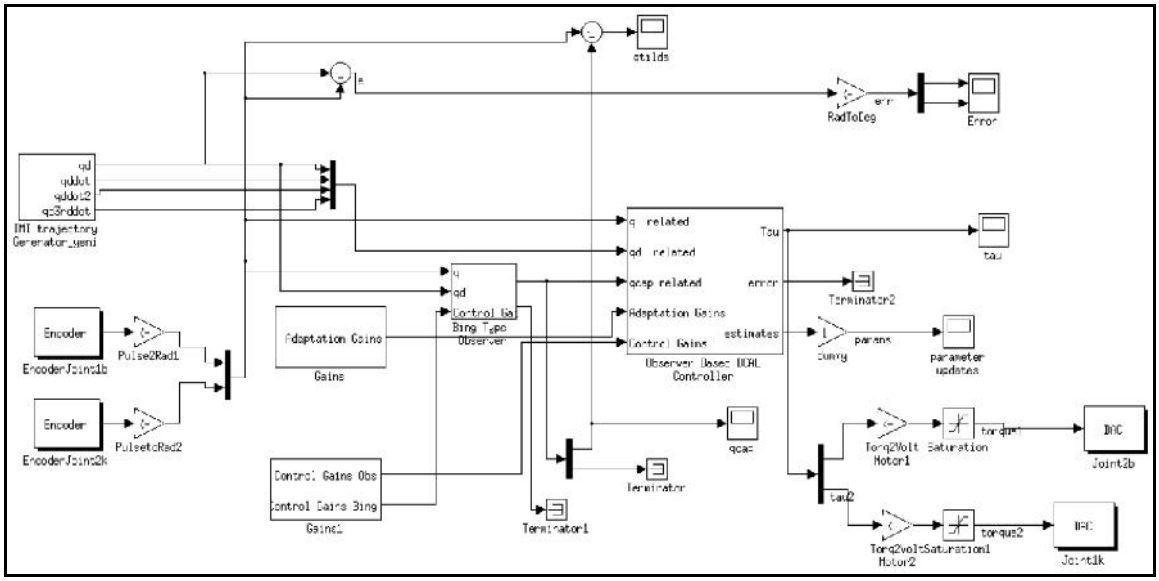

Figure 6. Simulink Model Used In the Experiment

After building the model for Xenomai target, we had an executable that can be run by our GUI mentioned in Section 2.2. Figure 7 illustrates the GUI just after the HIL simulation stop. The GUI has scope windows which plots graphics of joint control voltages, position errors, and parameter adaptations with respect to the

time. GUI also has seven-segment displays, the parameter window, the main console and the model manager window displaying the simulation stop time and elapsed time.

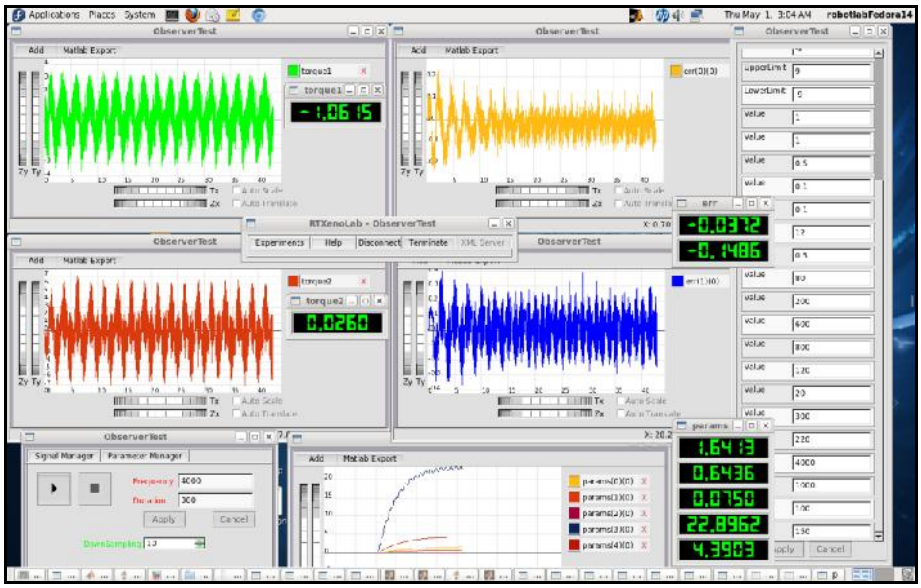

Figure 7. Screen Shot of the GUI Components of the Framework during the Experiment

The parameter tuning process was comparatively easier and was finished less than a few hours. As it can be seen in the following figures that joint-1 position error has converged to approximately 0.15 degree in average from 30 degrees and remained under 0.20 degree in steady state. Also it can be observed that joint-2 position error has converged to 0.30 degree in average and remained under 0.4 degree in steady state. Related joint position errors which were plotted by our GUI module are illustrated in the following figures.

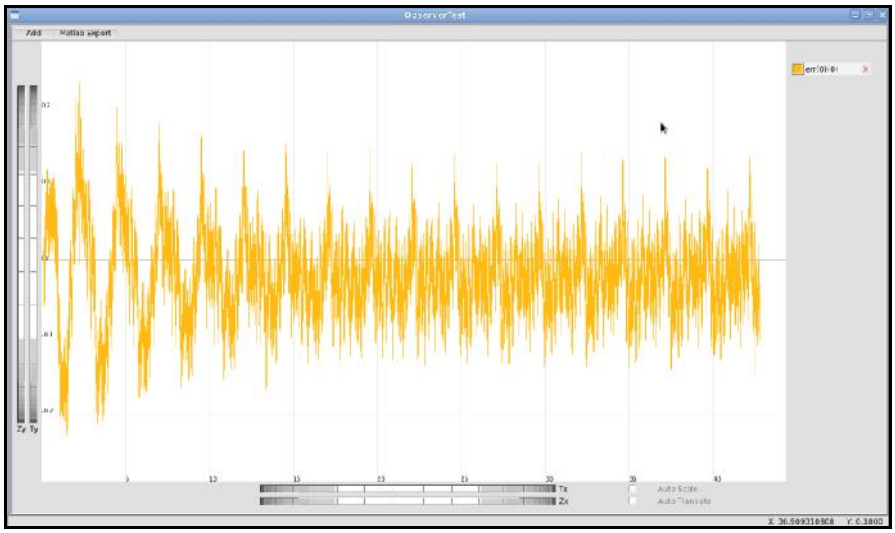

Figure 8. *Scope Window* Plotting Joint-1 Position Error in Degrees

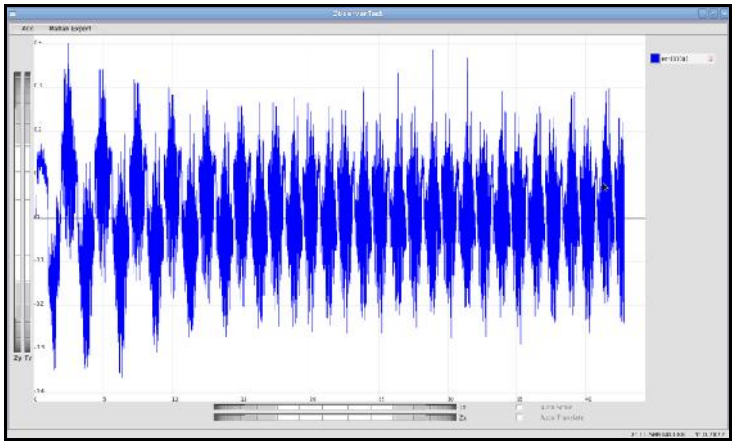

Figure 9. *Scope Window* Plotting Joint-2 Position Error in Degrees

Applied control voltage values to the power amplifiers of the joint motors are illustrated in the following figures. These values are same as applied voltage to the joint motors.

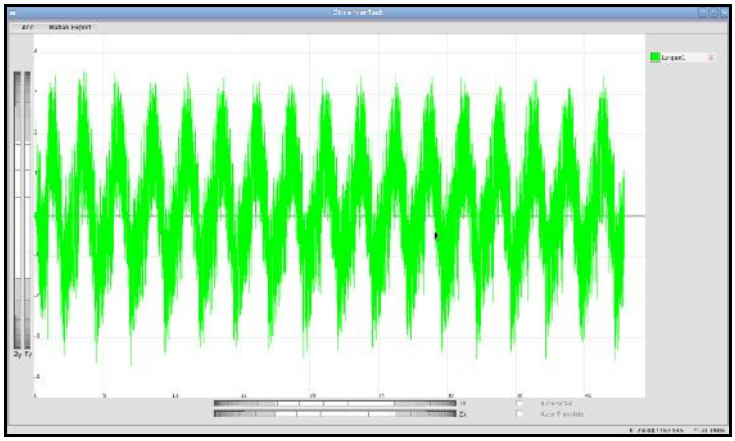

Figure 10. *Scope Window* Plotting Applied Voltage to the Power Amplifier of the Joint-1 Motor

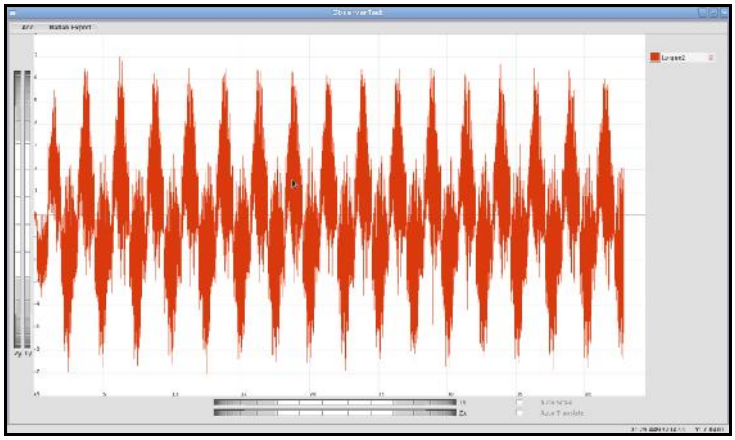

Figure 11. *Scope Window* Plotting Applied Voltage to the Power Amplifier of the Joint-2 Motor

5. **Conclusion**. In this paper we proposed a new hard-real time HIL simulation framework for a PC target. The real-time performance depends on computational power of PC and complexity of used Simulink model.

Performed servo control experiments showed that, the proposed framework have sufficient real-time control performance in handling complex nonlinear control algorithms such as an observer based DCAL controller. It was observed that the framework succeeded in handling the controller used in the experiment up to control frequency of 10 kHz. After this frequency level, hard-real time control of the robot could not be achieved. In basic data acquisition experiments, it was seen that the framework keeps real-time performance up to control frequency (execution repetition of base task loop in one second) of 20 kHz. The framework has own human user interface so that MATLAB does not have to run concurrently with a real-time task and a user can use more functional GUI components.

Compared to Real-Time Windows Target (RTWT) [18], a user does not need to use a sink type GUI component, such as a scope or a display, per each signal component desired to view. Because with *scope window, of Linux Xenomai Target (LXT)* a user can view desired number of model signal. In addition, *LXT* reaches same frequency ranges of RTWT.

As a conclusion, we have experienced that *Linux Xenomai Target* can be a good alternative for commercial HIL simulation solutions. Our future work will focus on extending the framework to embedded distributed targets and combining the approach proposed in [11] with the overall framework architecture.

REFERENCES

[1] QNX Software Systems, 1001 Farrar Road, Ottawa, ON K2K 0B3, Canada, web page: www.qnx.com (Accessed May 2014).

[2] Behnam, M., Nolte, T., Shin, I., Åsberg, M., & Bril, R. (2008). Towards hierarchical scheduling in VxWorks. In OSPERT 2008, Proceedings of the Fourth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (pp. 63-72).

[3] Gerum, P. (2004). Xenomai-Implementing a RTOS emulation framework on GNU/Linux. White Paper, Xenomai.

[4] Mantegazza, P., Dozio, E. L., & Papacharalambous, S. (2000). RTAI: Real time application interface. Linux Journal, 2000(72es), 10.

[5] dSpace GmbH, Paderborn, Germany, web page: http://www.dspace.com (Accessed May 2014)

[6] Broenink, J. F. (1999). 20-sim software for hierarchical bond-graph/block-diagram models. Simulation Practice and Theory, 7(5), 481-492.

[7] Sadeghzadeh, I., Mehta, A., Zhang, Y., & Rabbath, C. A. (2011, September). Fault-tolerant trajectory tracking control of a quadrotor helicopter using gain-scheduled PID and model reference adaptive control. In Annual Conference of the Prognostics and Health Management Society (Vol. 2).

[8] Bucher, R., Mannori, S., & Netter, T. (2008). RTAI-Lab tutorial: Scilab, Comedi, and real-time control. Recuperado el, 26.

[9] Campbell, S. L., Chancelier, J. P., and Nikoukhah, R. (2006). Modeling and Simulation in SCILAB (pp. 73-105). Springer New York.

[10] Yao, Z., Costescu, N. P., Nagarkatti, S. P., and Dawson, D. M. (2000). Real-time linux target: A MATLAB-based graphical control environment. In Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on (pp. 173-178). IEEE.

[11] Loffler, M. S., Costescu, N. P., and Dawson, D. M. (2002). QMotor 3.0 and the QMotor robotic toolkit: a PC-based control platform. Control Systems, IEEE, 22(3), 12-26.

[12] An, B., Liu, G., & Senchun, C. (2012, September). Design and implementation of real-time control system using RTAI and Matlab/RTW. In Control (CONTROL), 2012 UKACC International Conference on (pp. 834-839). IEEE.

[13] Azevedo, J. M. C. A. D. (2011). Xenomai Lab: a platform for digital real-time control.

[14] Spitzak, B. (1998). Fast light toolkit (fltk). FTLK: Fast light toolkit. Available: http://www. fltk. org/.(Accessed: March 2009).

[15] Zergeroglu, E., and Tatlicioglu, E. (2010, September). Observer based output feedback tracking control of robot manipulators. In Control Applications (CCA), 2010 IEEE International Conference on (pp. 602-607). IEEE.
[16] Kiszka, J. (2005, November). The real-time driver model and first applications. In 7th Real-Time Linux Workshop, Lille, France.
[17] Barbalace, A., Luchetta, A., Manduchi, G., Moro, M., Soppelsa, A., & Taliercio, C. (2007, April). Performance comparison of VxWorks, Linux, RTAI and Xenomai in a hard real-time application. In Real-Time Conference, 2007 15th IEEE-NPSS (pp. 1-5). IEEE.
[18] Mathworks Inc., Massacuhetts, USA, web page: http://www.mathworks.com/products/rtwt (Accessed June 2014).