Int. J. Engng Ed. Vol. 26, No. 5, pp. 1249–1265, 2010
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Design and Implementation of a Linear Axis Rapid Development System for Education*

MICHAEL A. FLEMING AND ROBERT G. LANDERS
Missouri University of Science and Technology, Department of Mechanical and Aerospace Engineering, 400 West 13th Street, Rolla, Missouri 65409–0050. E-mail: maf7d7@mst.edu and landersr@mst.edu

It is difficult to experimentally implement a controller in a typical semester-long course after the theoretical work is complete due to the time required to interface the controller with the physical hardware. A concept for a Rapid Development System (RDS) is introduced in this paper to automatically generate an interface between the student controller and physical hardware based on controller specifications input by the student. This provides the student with a tool to explore aspects of control design and implementation within the time constraints of a typical semester–long course. The RDS concept is applied to a linear axis system to create a Linear Axis RDS, which is utilized by a group of sixteen manufacturing automation and mechatronics students. Little specific knowledge about the interface is required to successfully implement a controller. The time required to simulate and implement a controller using the Linear Axis RDS is significantly shorter compared with the traditional simulation and implementation process. The controller performance results show that students were able to spend the majority of their time in the controller tuning process, as opposed to spending their time configuring hardware when it is not part of the course material. The students were successful in implementing their controllers in a typical semester-long course utilizing the Linear Axis RDS.

Keywords: control simulation; emulation and implementation; rapid development learning environment

1. INTRODUCTION

A CRITICAL PART of controller design is the implementation of the controller in a physical hardware system. Typically, students design a theoretical controller algorithm using a simple model of a physical system. The controller performance is analyzed theoretically and the closed–loop system is simulated. Simulating the closed–loop response requires time to encode the controller and system model and does not expose students to the limitations involved when implementing a controller on physical hardware. The difficulty in students implementing a controller on a physical system during a semester long course lies in the time required to implement the controller. Traditionally, after spending time to simulate the closed–loop system, students must design an interface to connect the controller to the physical hardware. For many students this process is tedious and time consuming because they must learn specific interface details. While a valuable exercise, this leaves less time for controller design and detailed analyses. A software program which greatly reduces the amount of effort required by the student to simulate and implement a controller is proposed in this paper. This software program is referred to as a Rapid Development System (RDS). There are three controller operation modes used to analyze the controller performance: simulation, emulation, and implementation. The RDS provides the interface needed to quickly operate the controller in these three modes and allows the students to focus on controller design and analysis, as opposed to hardware configuration when it is not part of the course material.

The development of assistive tools used to simulate, emulate and implement controllers has stemmed from efforts in industry to simulate manufacturing processes. Examples include a variety of CNC tool path simulators, each are developed as an aid in machining and manufacturing process planning [1–3]. These simulators are primarily concerned with tool path planning and prediction, and are not designed for rapid insertion of controllers. Further work in simulating specific machining processes, such as milling, has been performed; however, these specific process simulators do not provide users with the option of simulating or implementing their own controller [4]. A method for rapidly reconfiguring a controller in a real–time system is introduced by McDuffie [5]. A sliding mode controller is encoded in Simulink (a user friendly graphical syntax), converted using Beacon into MetaH syntax, and integrated into an embedded hardware platform for real–time flight control testing. This method reduces the amount of coding required to integrate a controller

* Accepted 15 October 2009.

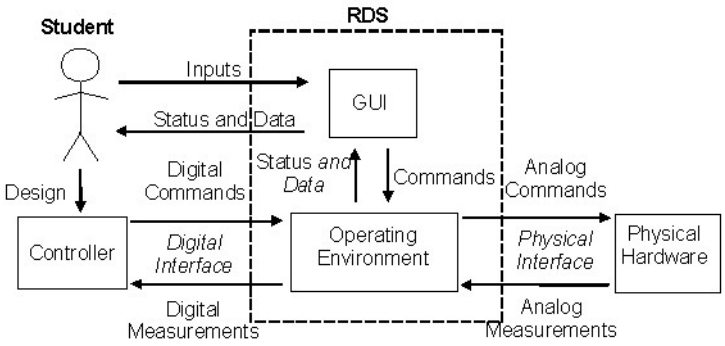

Fig. 1. General RDS structure.

but does not focus on the application of instructional use. Tools for controls instruction such as Simulink, used to encode and simulate closed–loop systems, have been utilized to allow students to design and simulate their own controllers [6]. These tools require the student to manually insert their controller in the encoded closed–loop system, which can be tedious and requires extra effort to implement the controller on a physical system. A network application, Second Best to Being There (SBBT), has been developed to enable students to remotely implement a controller in a controls engineering laboratory [7]. The SBBT is used to implement a student designed controller in real–time on a robotic arm. The motion and sounds of the robotic arm are sent to the student with the goal of projecting the experience of implementing their controller in the laboratory. The controller implementation process does not include simulation or emulation of the controller. A comprehensive two–axis turning simulator has been developed for instructional purposes [8]. The turning simulator allows students to design the dynamic models and controllers of the linear axes, spindle, and machining force process. The turning process is animated on a visual display after which the students can access the controller performance results for analysis. No emulation or implementation of the student controller is performed. A Hardware in the Loop environment utilizing microcontrollers and MATLAB xPC Target was developed in [9] where a virtual dynamic system was controlled. A Direct Current motor was controlled using MATLAB Simulink in [10]; however, embedded simulation and emulation capabilities were not provided. The RDS aims to provide students with the ability to simulate their dynamic model of a physical system and controller, emulate their controller on the target processor, and implement their controller on a physical system, all with little effort in a relatively short amount of time.

2. GENERAL RDS

A RDS is designed to connect the physical hardware, controller, and student, as shown in Figure 1. The RDS uses an operating environment to send commands from the controller and receive measurements from the physical hardware. The RDS uses a graphic user interface (GUI) to receive controller information and commands from the student, and provide the student with status and performance data from the programmable operating environment.

The connection between the controller and physical hardware is established using the operating environment as shown in Figure 1. The student uses a specific syntax which is recognizable by the RDS to encode the controller. Specific information about the controller, such as the controller operation mode, is used to program the controller interface algorithm in the operating environment. The RDS inserts the student controller into the controller interface algorithm, thereby connecting the controller to the operating environment. The controller calculates digital controller commands in the operating environment. The operating environment sends corresponding analog controller commands to the physical hardware using a physical hardware interface. The physical hardware interface is also used to convert measurements into digital data, which are manipulated by the student controller in the operating environment.

The connection between the student, controller, and physical hardware is established through the operating environment using the GUI as shown in Figure 1. Specific information about the controller is input by the student using the GUI. Based on the student input, the RDS automatically creates the controller interface algorithm of the operating environment. When the programming is complete, the student can input operation commands, such as starting and stopping the controller operation, into the GUI. The student can also manually manipulate the physical hardware using the GUI, which is often necessary before implementing the controller. An animation of the physical hardware displays the corresponding physical action of the hardware through the GUI. This is useful when the student cannot see the physical hardware. The status of the controller insertion process, the controller tracking performance, and the status

of the physical hardware are sent to the student through the GUI.

3. APPLICATION TO A LINEAR AXIS SYSTEM

The general RDS concept is applied to the x–axis of the mini–CNC machine system shown in Figure 2. The physical hardware is the x–axis, and is referred to as the linear axis system. A RDS is designed in two major components, the operating system and the GUI, specifically for the linear axis system and is called the Linear Axis RDS.

3.1 Operating Environment

The connection between the controller and the physical hardware is made using a host–target operating environment. This operating environment consists of software and hardware components. The two computers shown in Figure 3, called the host computer and the target computer, comprise the operating environment hardware.

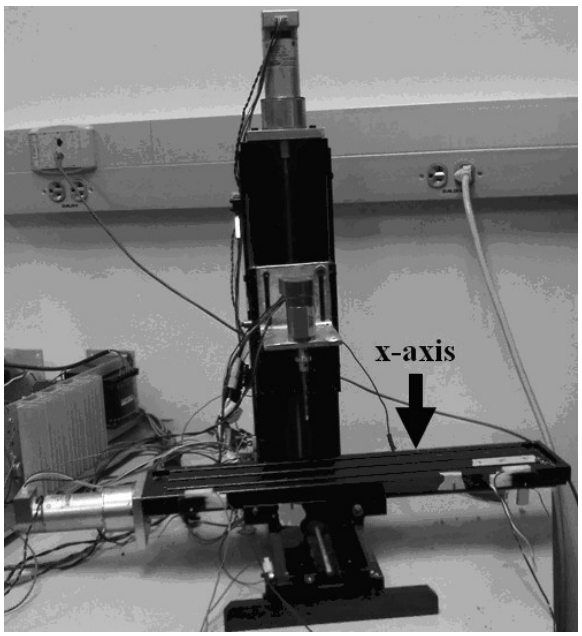

Fig. 2. Mini–CNC system containing the linear x–axis.

The host computer is used to operate the GUI and build the controller interface algorithm, which governs the communication between the controller and the linear axis system. The target computer is used to operate the controller interface algorithm sent from the host computer via a TCP/IP connection. Unlike the host computer, the target computer operates in real–time. Digital command signals from the controller, computed on the target computer, are converted to analog signals by holding the digital value constant for each sample period. This conversion is preformed by a National Instruments (NI) 6711 digital to analog (D/A) output board installed in the target computer. The D/A board allows the target computer to send continuous control signals to the linear axis. Conversely, measurements sent from the linear axis hardware to the target computer are converted into digital signals by an NI 6602 counter–timer (C/T) board. These digital signals, as well as the status of the real time operation, are sent from the target computer to the host computer and displayed on the GUI. The software used in the host–target operating environment includes MATLAB Simulink, used on the host computer, and MATLAB xPC Real–Time environment, used on the target computer. The MATLAB Simulink syntax is displayed to the student through the GUI on the host computer in a visual diagram, called a Simulink model, containing blocks and connectors used to encode the student controller. The controller interface algorithm used to connect the controller to the operating environment is also encoded in a Simulink model. The student controller is inserted into the controller interface algorithm Simulink model and converted into machine code syntax by the MATLAB compiler. This machine code syntax is then sent to the target computer, where it is executed. The MATLAB xPC Real–Time environment operates the target computer with a finite sample period, i.e., in real–time, using machine code to operate the physical hardware.

The contents of the controller interface algorithm depend upon the controller operation mode. Each operation mode is used for the purpose of iteratively evaluating the performance of the controller. The controller is not recoded for each operation mode, but simply adjusted until the

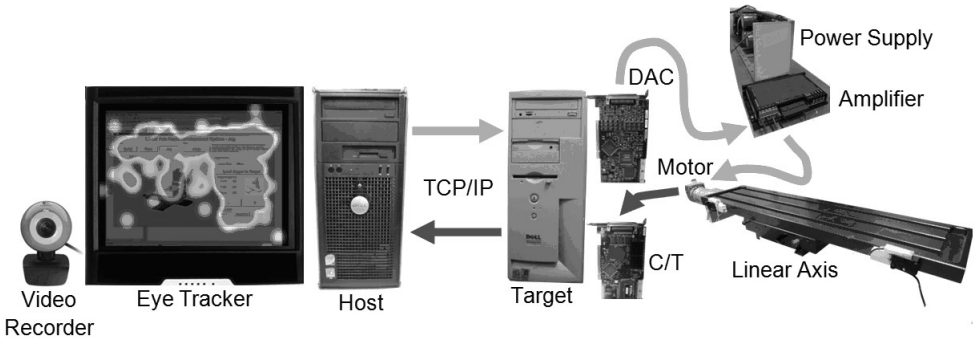

Fig. 3. Experimental setup with eye tracker, host–target environment, and linear axis.

desired level of performance is achieved. The first controller operation mode (i.e., simulation mode) allows the student to execute the Simulink model on the host computer with a model simulating the linear axis response. Physical hardware interface limitations from the D/A and C/T conversions are included in the simulation. These limitations include saturation, which is the finite analog output range, and quantization, which are the finite analog output and measurement resolutions. The student controller does not need to be connected to the target computer and physical hardware to perform simulations. This allows the student to quickly implement changes to their controller. The Simulink model for simulation mode, shown in Figure 4, consists of four basic blocks called subsystems. The first subsystem contains the Reference Generator which calculates the linear axis reference position (i.e., desired position). The reference position and the measured position of the simulated physical hardware are input into the second subsystem containing the Controller. This subsystem is specified by the student in Simulink syntax. The controller calculates the control voltage given the reference positions, measured positions, and previous control signals. The control voltage is sent to the third subsystem, called the Computer–System Interface, shown in Figure 5. The Computer–System Interface subsystem contains the simulated physical hardware interface effects, i.e., saturation and quantization, for both the D/A and C/T conversions. The control command is converted from a digital signal into a simulation of a saturated and quantized analog signal, which is sent to the fourth subsystem, called the Linear Axis Model. The Linear Axis Model contains a dynamic model of the physical hardware and calculates the simulated linear axis position response. The simulated position is sent to the Computer–System Interface where it is converted from a simulated analog position into a quantized digital measured position. This measured position is then sent to the Controller, thereby closing the interface system controller loop.

The second controller operation mode (i.e., emulation mode) allows the student to execute their controller on the target computer in real–time without moving the physical hardware. Therefore, no action is commanded to the linear axis hardware and the physical measurements are disregarded. Instead, a model of the linear axis response, including the physical hardware interface effects, is simulated on the target computer. This allows the student to determine if the target computer is able to perform all calculations and execute send and receive communication tasks to the linear axis hardware within the requested sample period. The Simulink model for emulation mode contains the same four subsystems found in the simulation Simulink model. The contents of the Reference Generator, Controller, and Linear Axis Model subsystems are the same. The difference between the emulation and simulation Simulink models lies in the Computer–System Interface subsystem. In the emulation Simulink model shown in Figure 6, the Computer–System Interface subsystem contains not only the simulated saturation and quantization effects, it contains the physical hardware interface communication blocks. The communication blocks enable the emulation Simulink model, converted to machine code and down-

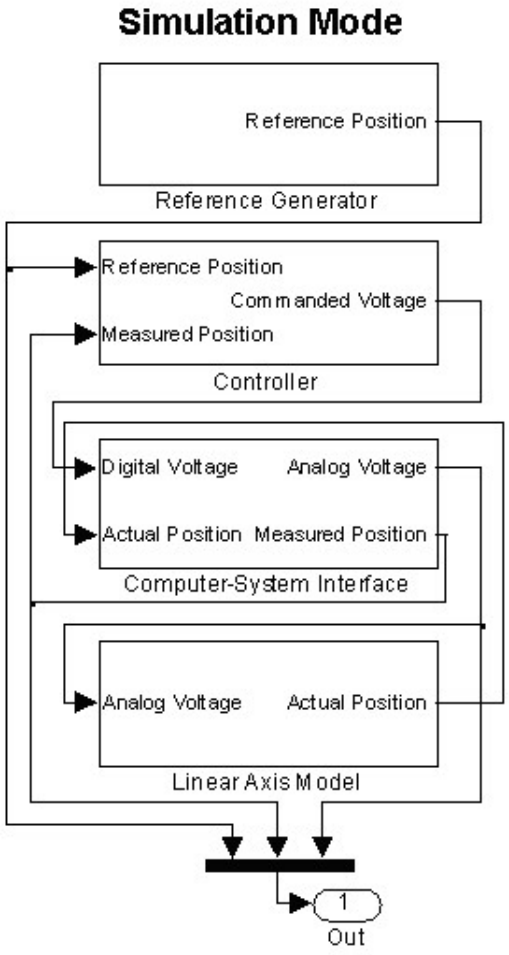

Fig. 4. Simulation mode Simulink model.

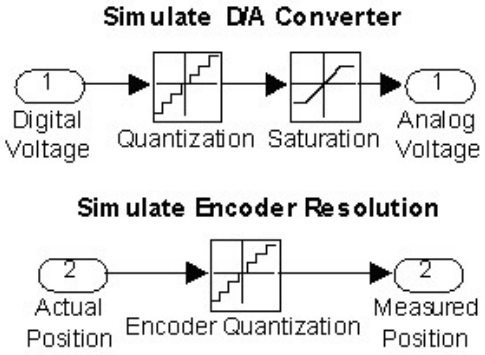

Fig. 5. Contents of Computer–System Interface subsystem for Simulation mode.

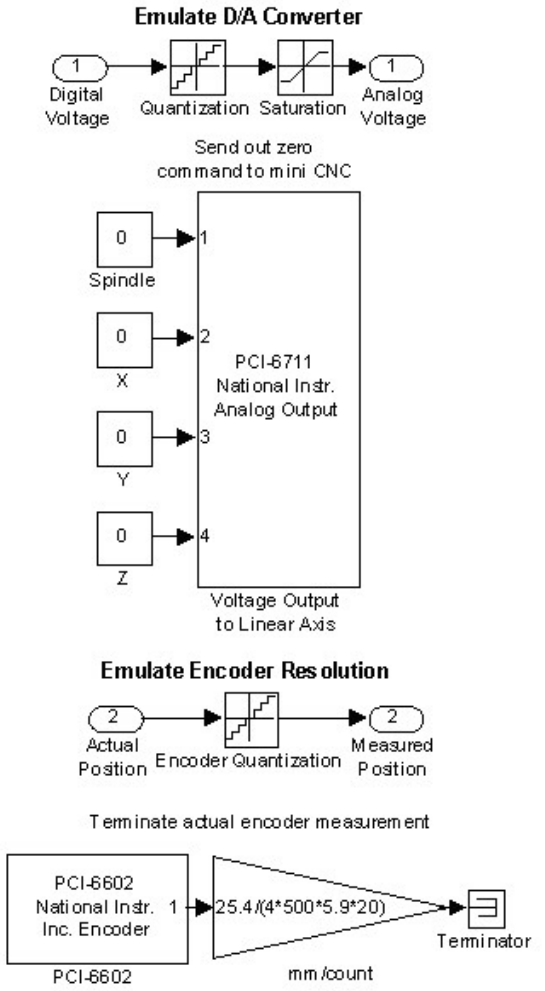

Fig. 6. Contents of Computer–System Interface subsystem for Emulation mode.

loaded to the target computer, to communicate with the linear axis hardware. No action, i.e., zero voltage, is sent to the linear axis and other components of the mini CNC to ensure the physical system does not move. The physical measurement received from the linear axis hardware is terminated, i.e., the value is not used.

The final controller operation mode (i.e., implementation mode) allows the student to operate the physical hardware using their controller. The implementation Simulink model is converted to machine code and downloaded to the target computer. There is no simulation of the linear axis dynamics or physical hardware interface effects because the linear axis and physical interface hardware, i.e., D/A and C/T boards, are utilized. The Simulink model for implementation mode, shown in Figure 7, contains three subsystems: Reference Generator, Controller, and Computer–System Interface. The contents of the Reference Generator and Controller subsystems are the same as in emulation and simulation modes. The difference lies in the Computer–System Interface subsystem shown in Figure 8, which contains only the communication blocks

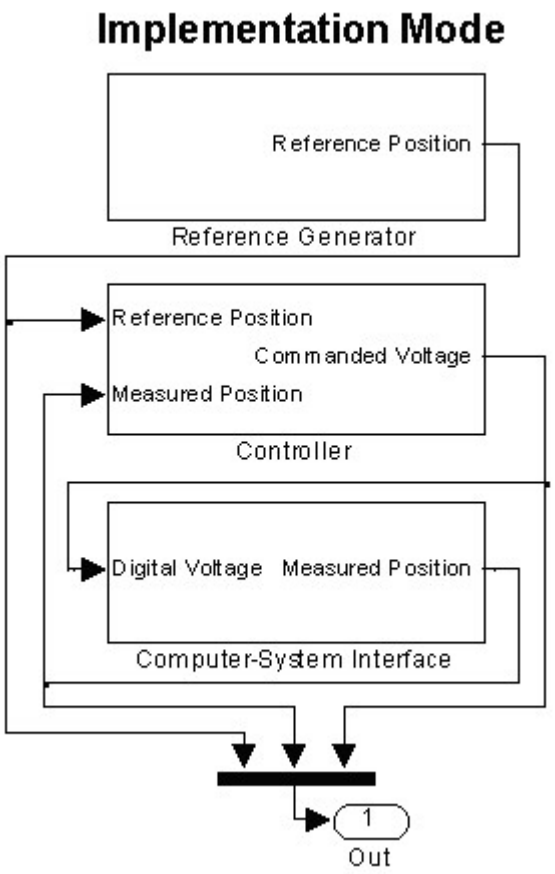

Fig. 7. Implementation mode Simulink model.

found in the emulation Simulink model. The analog control signal is sent through the D/A board to the linear axis amplifier and the physical encoder measurements are sent through the C/T board to the controller. No action, i.e., zero voltage, is sent to the other components of the mini CNC to ensure only the linear axis is activated.

3.2 Graphical User Interfaces

The student is connected to the host–target operating environment through a series of Graphical User Interfaces (GUIs), created using the MATLAB program GUIDE. The GUI programs operate in MATLAB on the host computer. The student uses three interlinked GUIs called Build, Operate, and Jog to build the controller interface algorithm in a Simulink model, operate the Simulink model containing the controller interface algorithm, and manually manipulate the linear axis hardware, respectively. A fourth GUI, Help, is interlinked to the other three GUIs and provides the student with step by step directions for completing each action. Each GUI consists of two components: the visual display and the callback code. The GUIDE program provides basic buttons and drop down lists which are positioned on the visual display as selectable options. The order in which the options appear, the location of

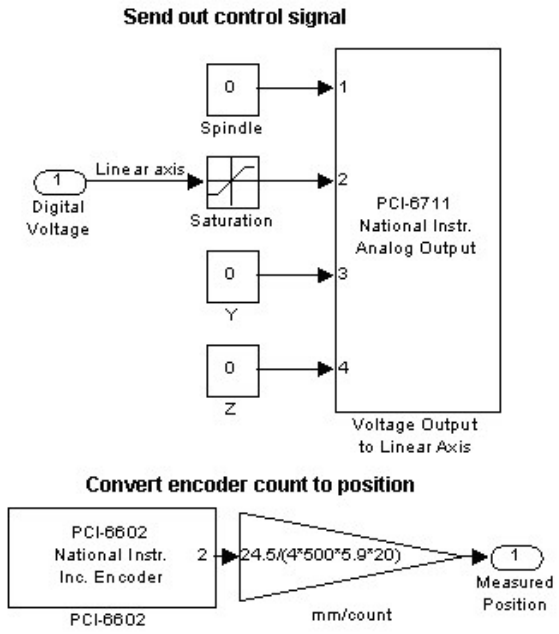

Fig. 8. Contents of Computer–System Interface for Implementation mode.

the options on the host computer screen, and the content of the options are designed in a way to allow the student to easily make their selections. An animation of the linear axis and the operation status of the controller interface algorithm are located on the visual display. The visual display is linked to a callback code generated by the GUIDE program in MATLAB syntax. The callback code is modified to perform various selections, such as starting or stopping the controller operation, and to provide visual confirmation of any completed or ongoing tasks, such as changing the position of the animation corresponding to the linear axis motion. Each GUI is now described in detail.

Build. The Build GUI is used to create the Simulink model containing the controller interface algorithm. The student selects the specific details of the controller interface algorithm, such as the controller operation mode, in the Build GUI visual display. These selections are used by the Build GUI callback to generate the corresponding Simulink model containing the controller interface algorithm. The Build GUI visual display is shown in Figure 9. The Operate and Jog toggle buttons located on the top left are pushed to navigate from the Build GUI to the Operate and Jog GUIs, respectively. The Help pushbutton on the top center is used to open the Help GUI. Tips are displayed on the top right, giving a summary of how to use the Build GUI. The information required to create the controller interface algorithm is arranged in five selections. First, the controller domain type is specified, using radio buttons, as either discrete or continuous. The discrete domain is selected for a controller designed in the Z–domain and encoded using discrete domain Simulink blocks. Continuous time domain is selected for a controller designed in the Laplace domain and encoded using continuous domain Simulink blocks. Second, the controller operation mode is specified. A popup menu containing the three mode options: simulation, emulation, and implementation, is positioned below the domain type option. Next, the linear axis model is specified. This model is used to simulate the linear axis dynamics for simulation and emulation modes and is encoded using discrete or continuous blocks in Simulink, depending upon the domain type. Students may encode their own model in a Simulink subsystem or select a pre–existing model. This selection is unnecessary for implementation mode and is disregarded if a selection is made. A popup menu containing the default model option or the student defined model

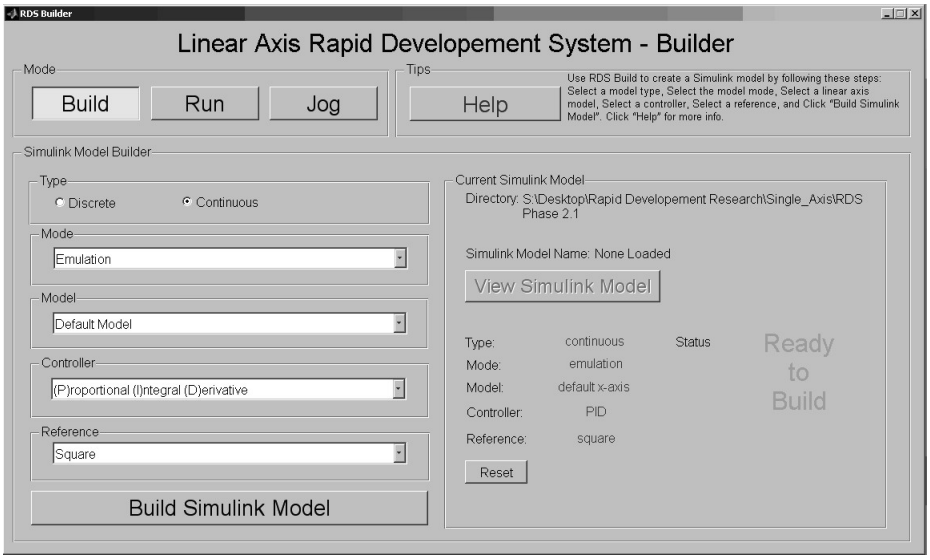

Fig. 9. Build GUI.

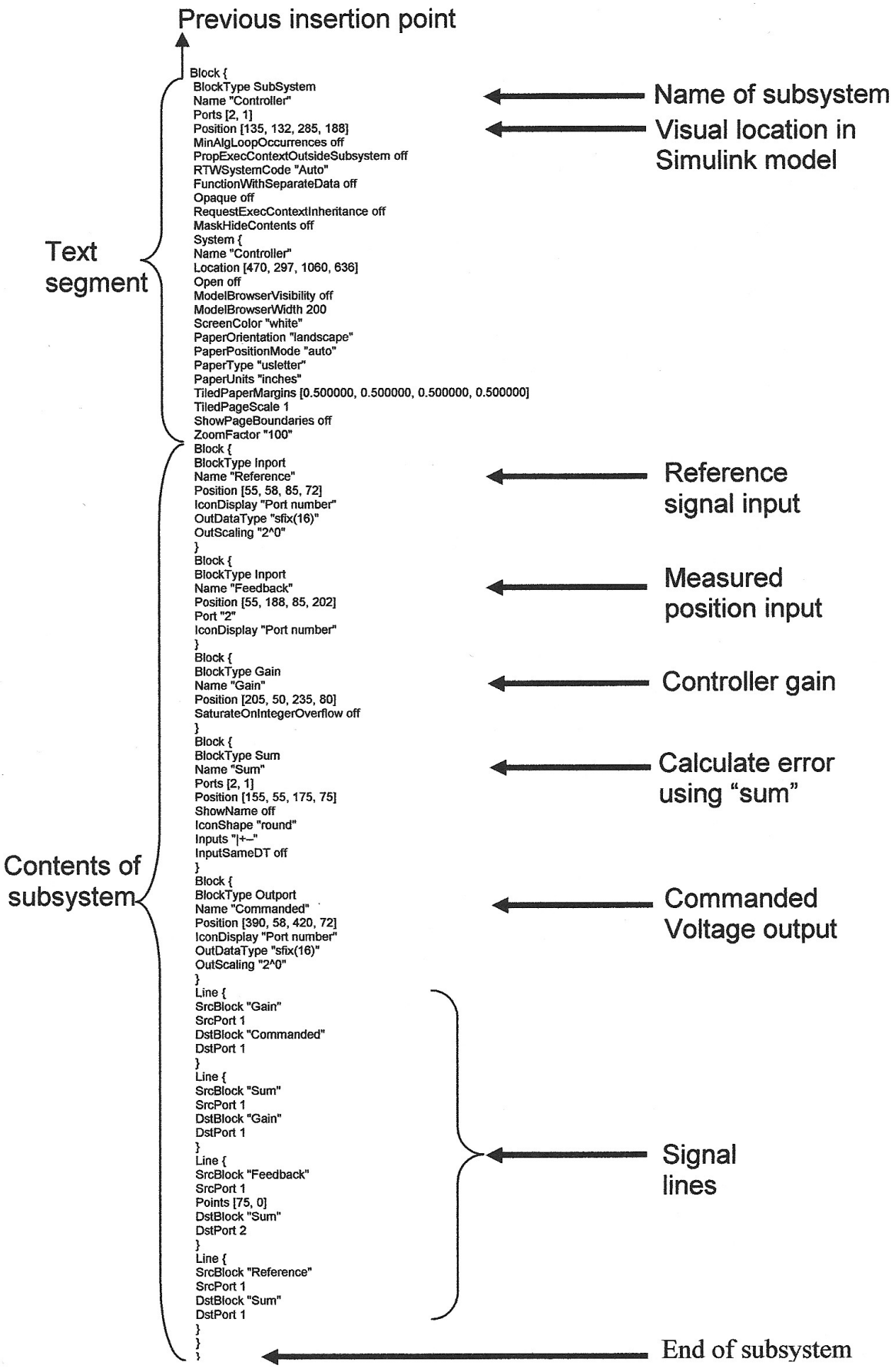

Fig. 10. Controller subsystem text based syntax.

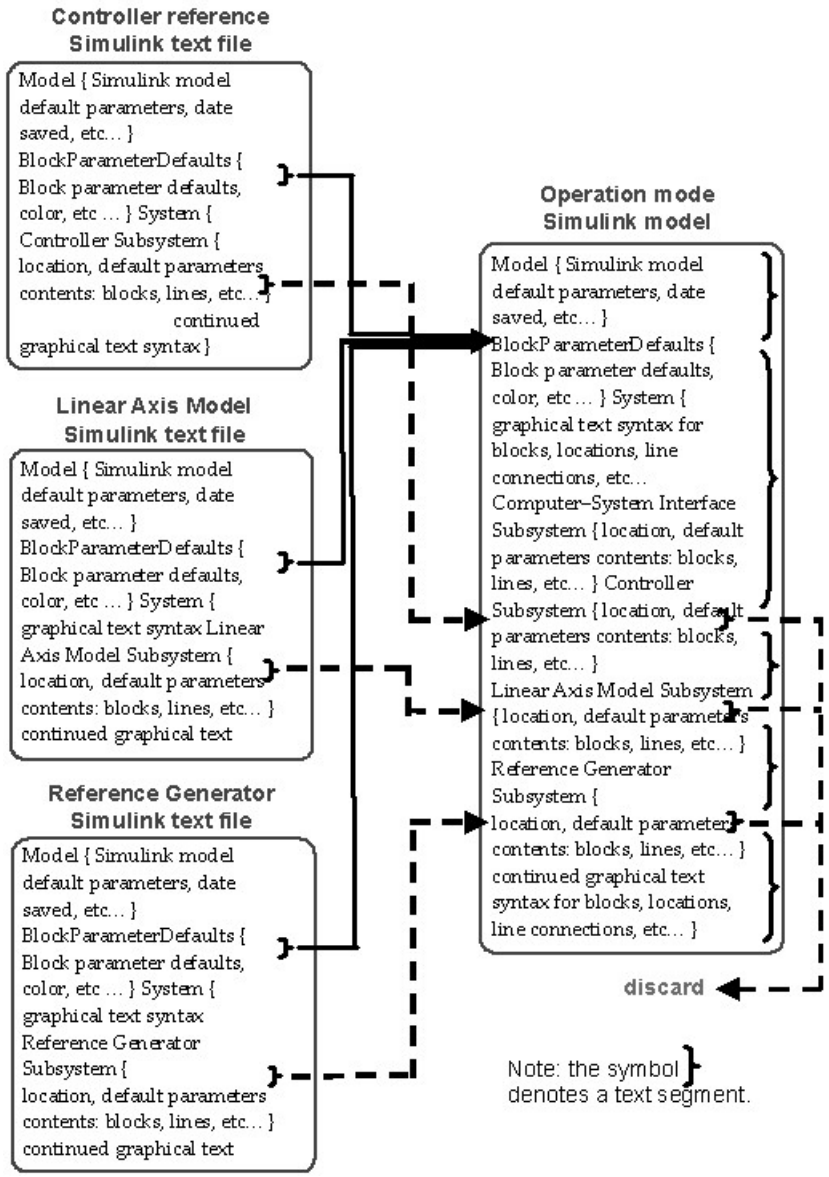

Fig. 11. Process for building new Simulink model.

option is positioned below the mode popup menu. If the student defined model option is selected, the location of the Simulink model file is specified using a browse box. Next, the controller is selected. Students may encode their own controller in a Simulink subsystem or select a pre–existing controller. A popup menu containing the option to insert a student controller or select a pre–existing controller is positioned below the model popup menu. Finally, the reference signal is specified. Three reference signals, sinusoidal, triangle, and square, are used to generate the desired reference position of the linear axis and to analyze the controller tracking performance. A popup menu containing the three reference signals is positioned below the controller popup menu. A text field displaying each option the student specified is positioned to the right of the field of five popup menus. A status text displays "Ready to Build" indicating to the student when all selections have been specified. The five selections are displayed in the selection text field to the left of the status text and can be reset to "not specified" by pushing the Reset pushbutton located under the selection text field. The student initiates the generation of the Simulink model containing the controller interface algorithm by pushing the Build pushbutton positioned below the reference popup menu. The status text indicates when the Simulink model has been generated and the View Simulink Model pushbutton is enabled. The student may open the Simulink model visual display by pushing the View Simulink Model pushbutton.

When the Simulink model is created graphically, a corresponding text file is also created containing the graphical information. Each block and signal contained in the Simulink model is represented by text based syntax. For example, the text based

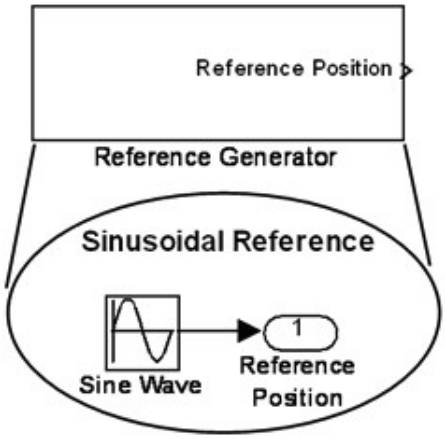

Fig. 12. Reference Generator subsystem and contents.

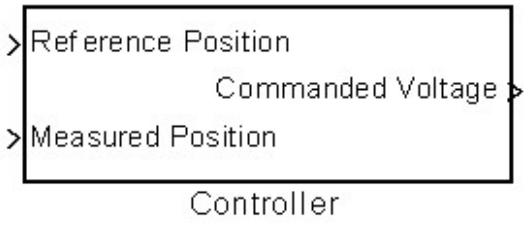

Fig. 13. Standard student Controller subsystem.

syntax used to represent the Controller subsystem in the three operation mode Simulink models is shown in Figure 10. The name, location and default parameters of the Controller subsystem appear first, followed by the graphical components of the controller, such as the gain block. A diagram of the text based syntax structure for the simulation mode Simulink model is shown in Figure 11. The structure is similar for the other Simulink models. The general Simulink model parameters appear first. Next are the block default parameters contained in the Simulink model, followed by the content of the four subsystems listed in alphabetical order, as well as the connections between them.

The Build GUI callback generates the Simulink model by piecing together text based syntax segments from pre–existing source Simulink models. A source Simulink model is created for each available selection from the Build GUI visual display and contains a subsystem with the selectable contents. An example of a source Simulink model is the Reference Generator subsystem shown in Figure 12 with the selectable sinusoidal wave content. Each source Simulink model is saved as a text file and stored on the host computer. The Build GUI callback reads the desired content from each selected source Simulink text file and combines the content into a new Simulink text file. In order to locate the selected subsystem content within the syntax of the source Simulink text files, standard subsystem names and connection labels are used. This standard also applies to the Simulink model containing the student controller and student linear axis model. The student encodes the controller or physical hardware model in a subsystem named "Controller" or "Model," respectively. An example of a student Controller subsystem is shown in Figure 13. The Build GUI callback combines the selected source Simulink text file content, first, by collecting the selected contents of the Reference Generator, Controller, and the Physical System Model subsystems. This is done by finding the location of the subsystem in the source Simulink text file by searching for the standard name of the specific subsystem, such as Controller. From this location the start and end of the subsystem content is determined. The Build GUI callback reads the content and stores it as a text segment. The same procedure is used to obtain the text of the default block parameters for each subsystem. Next, when the selected content text segments, i.e., selected subsystem content and default block parameters, are stored the Build GUI callback reads the text syntax of the selected operation mode Simulink model as shown in Figure 11. Each selected subsystem has a corresponding insertion point in the operation mode Simulink model. The default block parameters also have an insertion point. The Build GUI callback reads the text syntax of the operation mode Simulink model before and after the selected content insertion points and stores the syntax as text segments. The first text segment starts at the beginning of the operation mode Simulink model and ends where the default block parameters of the desired subsystems are inserted. The next text segment starts after the default block parameters insertion point and ends where the first selected subsystem, i.e., the Controller subsystem, is inserted. The insertion point of the Controller subsystem is below the location information of the corresponding operation mode Controller subsystem. This allows the operation mode Simulink model to maintain the original location of each subsystem. The default content in the operation mode Controller subsystem is replaced by the selected subsystem content. Additional text segments are read between the remaining subsystem insertion points in the same manner. A final text segment starts at the last subsystem insertion point and ends at the end of the operation mode Simulink model. The text segments are combined in consecutive order and saved as a Simulink file corresponding to the new Simulink model containing the desired controller interface algorithm.

Operate. The Operate GUI is used to operate the Simulink model containing the controller interface algorithm and to analyze the controller performance. Operation commands from the Operate GUI callback are sent to the Simulink model on the host computer for simulation mode and to the target computer for emulation and implementation modes. The visual display of the Operate GUI is shown in Figure 14. The Build and Jog toggle

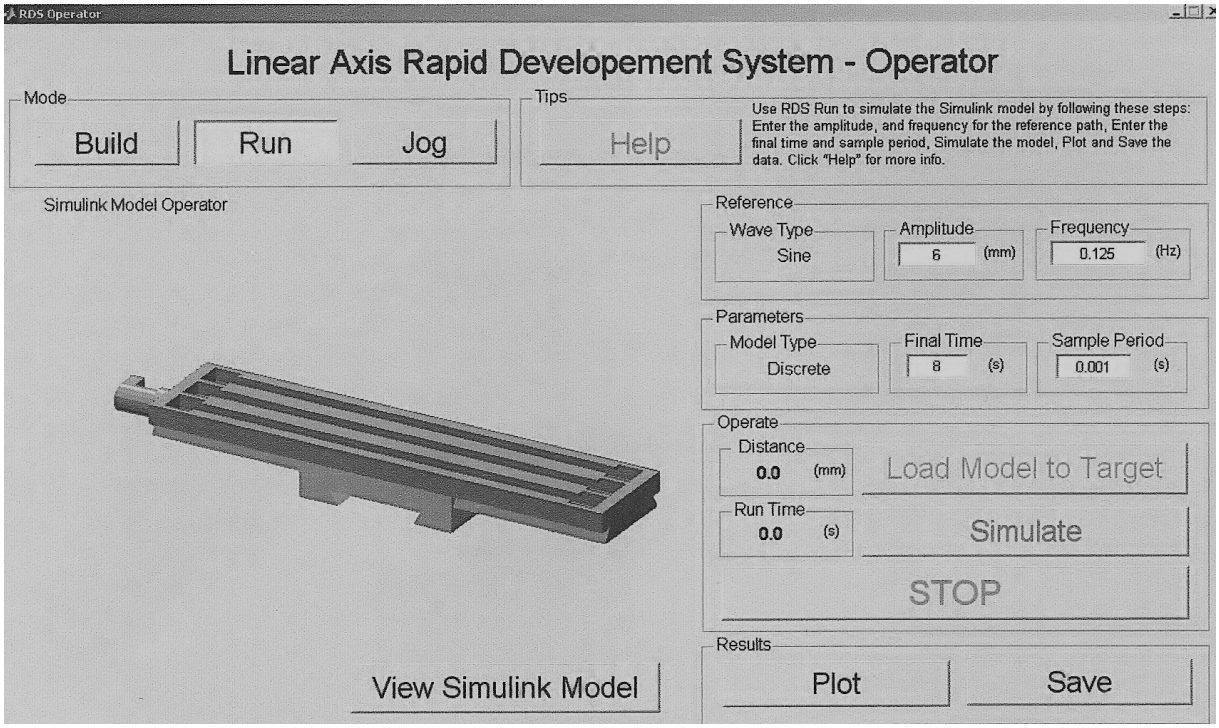

Fig. 14. Operate GUI.

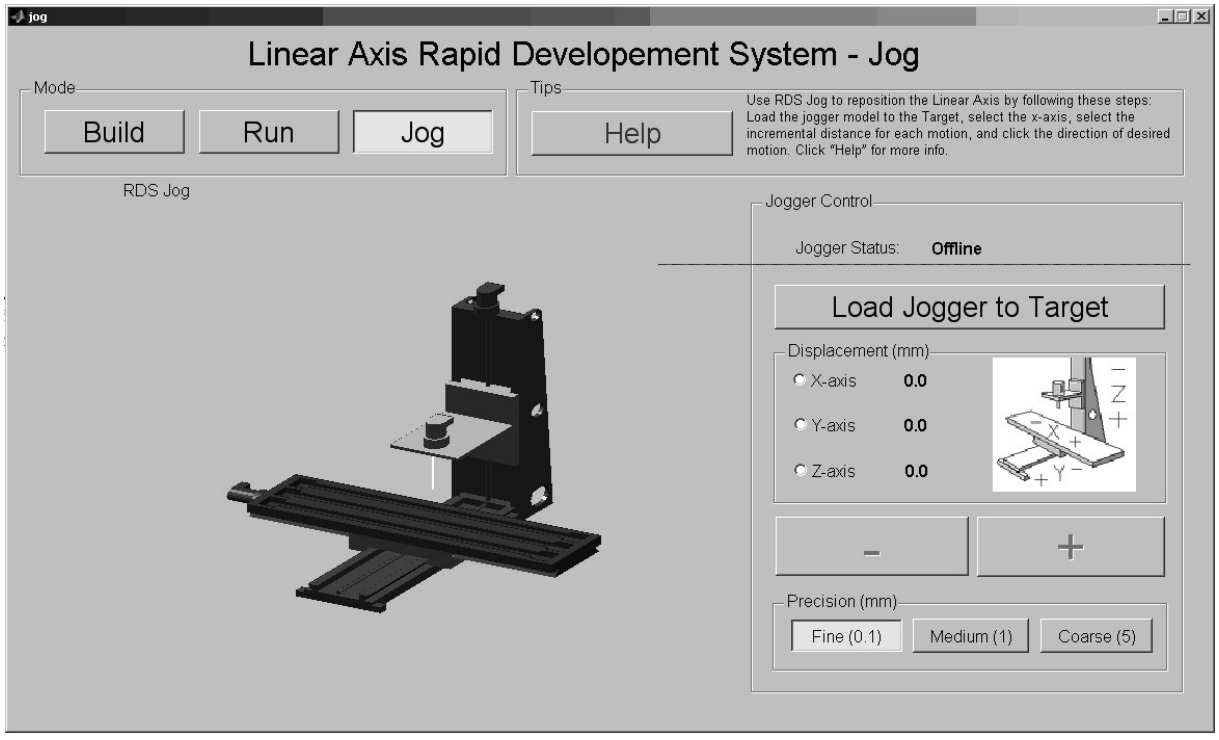

Fig. 15. Jog GUI.

buttons located on the top left are pushed to navigate from the Operate GUI to the Build and Jog GUIs, respectively. The Help pushbutton on the top center is used to open the Help GUI. Tips are displayed on the top right, providing a summary of how to use the Operate GUI. The first input is the reference signal parameters. The reference signal amplitude (mm) and frequency (Hz) are entered in the text boxes. These values are stored on the MATLAB workspace and are input into the Reference Generator subsystem to create the reference signal. The second input is the Simulink model operation parameters. The duration of the controller operation and the sample period are

entered in the text boxes below the reference parameters. These values are stored and used by the Simulink model solver during operation. The student initiates the operation by pushing the start pushbutton labeled "Simulate," "Emulate," or "Implement," corresponding to the respective operation mode. If the operation mode is emulation or implementation, then the Simulink model is converted into machine code and downloaded from the host computer to the target computer. This process is initiated by pushing the "Load Model to Target" pushbutton positioned below the Simulink model operation parameters and above the start pushbutton. If the host computer is not connected to the target computer, the Operate GUI will prompt the student to connect by specifying the TCP/IP connection between the host and target computer using the MATLAB connection program xPC Real–Time Explorer. A three–dimensional animation of the linear axis, located to the left of the Operate GUI visual display, illustrates the linear axis motions. During the operation, the measured position from the linear axis hardware in implementation mode, or linear axis model in simulation or emulation modes, is used to update the position of the linear axis animation. The operation is terminated when the operation is complete or by pushing the Stop pushbutton. The recorded data is displayed on a graph or saved in a text file using the "Plot" or "Save" pushbuttons, respectively, positioned below the "Stop" pushbutton.

Jog. The Jog GUI is used to manually position the linear axis. Jog commands are sent from the Jog GUI callback to a Simulink model on the target computer containing a controller interface algorithm designed specifically to receive jog commands. The jog Simulink model includes a pre–existing controller and communication blocks similar to the communication blocks in the emulation and implementation Simulink models. The Jog GUI visual display is shown in Figure 15. The Build and Operate toggle buttons located on the top left are pushed to navigate from the Jog GUI to the Build and Operate GUIs, respectively. The Help pushbutton on the top center is used to open the Help GUI. Tips are displayed on the top right, providing a summary of how to use the Jog GUI. The student initiates the jog action by pushing the "Load Jogger to Target" pushbutton. The same connection procedure in the Operate GUI is used for the Jog GUI. The jog Simulink model is compiled in machine code and loaded to the target computer. The status of the jog Simulink model is displayed above the "Load Jogger to Target" pushbutton. When the Jog GUI is ready for commands the status displays "Online." Next, the linear axis is selected by pushing the x–axis radio button located under the "Load Jogger to Target" pushbutton. The linear axis is incrementally repositioned, i.e., jogged, in the positive or negative direction by pushing the "+" or "–" pushbuttons, respectively. Three incremental motion magnitudes can be selected: course (5 mm), medium (1 mm), and fine (0.1 mm). A three–dimensional animation of the mini–CNC system, located to the right of the Jog GUI visual display, illustrates the motion of the components of the mini–CNC system. The linear axis position measurement from the jog Simulink model on the target computer is used to update the position of the linear axis in the mini–CNC animation.

Help. The Help GUI provides information that guides the student through the use of the Build, Operate, and Jog GUIs. This information is available for students who misunderstand the use of the three GUIs. The Help GUI visual display is shown in Figure 16. Step by step directions for performing different actions, such as connecting the host and target computers, are displayed in a scrollable text box on the right of the Help GUI. The topics are arranged in a vertical field of toggle buttons to the left of the scrollable text box and are labeled according to the corresponding topic. The Help GUI is dynamic, meaning that when the student pushes the Help pushbutton in any of the three GUIs the relevant information is automatically displayed in the scrollable text box when the Help window appears. Figures correlating to the information provided in the scrollable text box are displayed using pushbuttons on the bottom or by double–clicking the text referring to the figures in the scrollable text box. An example is shown in Figure 16 where the xPC Explorer connect figure is displayed by pushing the "xPC Connect" pushbutton or double–clicking the corresponding "<Double Click>" text.

3.3 Linear Axis Hardware

The linear axis hardware, shown in Figure 3, is connected to the host–target environment through the physical hardware interface consisting of D/A and C/T boards. The target computer operates the downloaded Simulink model in real–time, sending a digital controller voltage command to the D/A output board where it is converted into an analog voltage command. The D/A output voltage range is ±10 V and is amplified before reaching the linear axis DC motor by a factor of 2.4. The motor has a gear ratio of 5.9 from the motor shaft to the output shaft. The output shaft is connected to a lead screw with a pitch of 0.787 rev/mm, which translates the linear axis in the positive or negative direction, depending on the motor rotational direction. The motor encoder counts the rotations of the motor shaft with a resolution of 500 counts/rev. Quadrature encoding is utilized for an effective resolution of 2000 counts/rev. The C/T board converts the encoder counts into a corresponding digital signal sent to the controller interface algorithm on the target computer where it is converted into a linear displacement measurement.

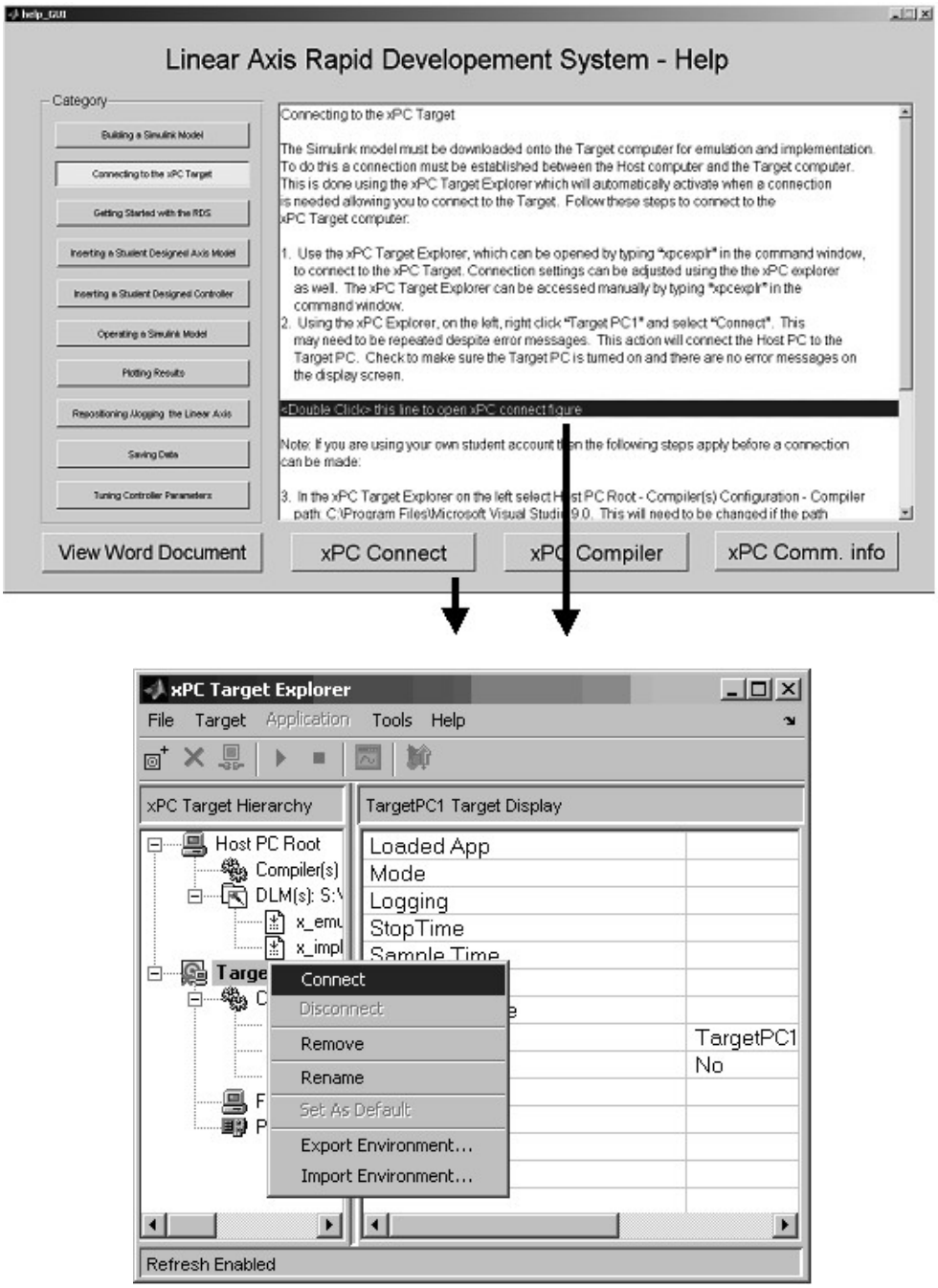

Fig. 16. Help GUI with open figure.

3.4 Linear Axis Model and Controllers

Source Simulink text files used to construct the Simulink models are created specifically for the linear axis system. The linear axis dynamics are modeled in the discrete and continuous domains and saved as separate files. The linear axis is subjected to a series of step voltage inputs and Recursive Least Squares estimation is used to determine the dynamic model parameters. The continuous domain model used to describe the linear axis is a nonlinear second order differential equation

$$\tau_x \ddot{x}(t) + \dot{x}(t) = K_x V_C(t) - K_x f_C \operatorname{sgn}(\dot{x}) \qquad (1)$$

where t_x = 8.00 ms is the system time constant, K_x = 1.72 (mm/s)/V is the steady–state gain, and f_C = 0.10 V is the Coulomb friction voltage offset. The parameter f_C is determined by finding the minimum voltage required to move the linear axis. The linear axis model is used in simulation and emulation modes. The discrete model, encoded in Simulink syntax, is shown in Figure 17. The physical hardware interfaces are modeled in simulation and emulation modes with a D/A quantization of 4.88 mV and an encoder quantization of 0.108 μm. Four Controller subsystems are added to the Simulink text files. These include proportional, proportional plus integral plus derivative (PID),

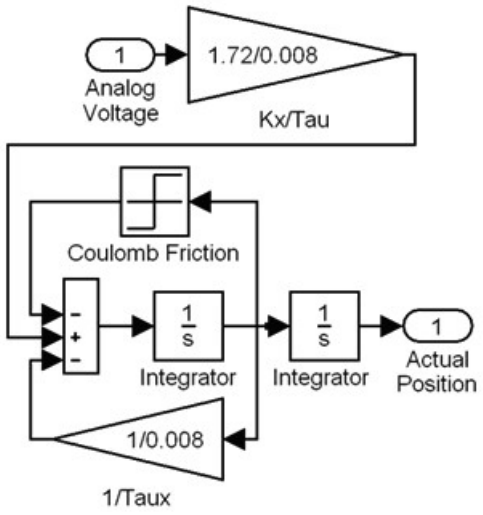

Fig. 17. Linear axis model contained in Linear Axis Model subsystem.

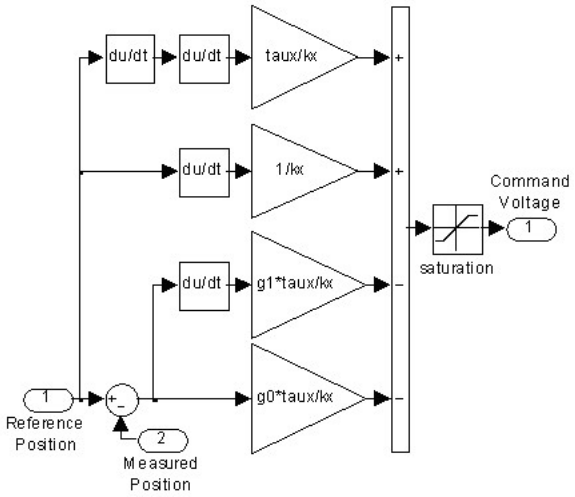

Fig. 18. Student controller encoded in Simulink syntax.

modified PID, and general tracking controllers. The student selects one of the four controllers or their own controller, to regulate the linear axis position.

4. RESULTS

A group of sixteen manufacturing automation and mechatronics students learning the basics of controlling manufacture processes and interfacing mechanical, electrical, and computer systems is given the assignment to implement a general tracking controller on the linear axis system using the Linear Axis RDS. The students are given the general tracking controller design [11], the linear axis model in equation (1), and the standard names of the subsystem and connections used to encode their controllers. The students design and encode their controllers in Simulink before the experiment. They are given four tasks requiring them to use the Jog, Build, and Operate GUIs. The tasks are: jog the linear axis such that it is centered, simulate your controller, emulate your controller, and implement a pre–existing controller. An eye tracker is used to record the student's eye movements on the host computer screen during the jog task and the pre–existing controller implementation task to analyze the design of three GUI visual displays. The Think Aloud Protocol is used during controller simulation and emulation tasks to evaluate the students' spoken thoughts and actions. The eye tracking and Think Aloud Protocol results are combined to analyze the usability of the Linear Axis RDS [12]. The controller tracking results for the simulation and emulation tasks are recorded. At the end of the four tasks the students are given an additional task to implement their controller. The controller tracking results for the implementation task are also recorded. Post experiment interviews are conducted to obtain student evaluations of the Linear Axis RDS.

The controller implementation process used by one of the sixteen students is given as an example. The student designed a general tracking controller for the linear axis prior to using the Linear Axis RDS. The equation the student used to calculate the commanded control voltage is

$$u(t) = \frac{\tau_x}{K_x}\left[\ddot{r}(t) + \frac{1}{\tau_x}\dot{r}(t) - g_1\dot{e}(t) - g_0 e(t)\right] \quad (2)$$

where $u(t)$ is the control voltage, $r(t)$ is the reference position, g_1 and g_0 are the controller gains, and $e(t)$ is the position error. The student controller gains, g_1 and g_0, are determined by selecting the desired closed loop time constants, t_1 and t_2, and are given, respectively, by

$$g_0 = -\frac{1}{\tau_1\tau_2} \quad (3)$$

$$g_1 = -\frac{1}{\tau_x} - \frac{1}{\tau_1} - \frac{1}{\tau_2} \quad (4)$$

The student controller is encoded in Simulink and shown in Figure 18. The controller is simulated, emulated, and implemented on the linear axis using the Linear Axis RDS. A triangle reference signal with a magnitude of 5 mm and a frequency of 0.125 Hz is used for all of the controller performance tests. The controller performance test results are shown in Table 1.

The simulated controller tracking results from Test 1, given in Table 1 and Figure 19, show the initial controller performance test had poor tracking results, as indicated by the comparatively large value of the mean and standard deviation of the absolute error. The desired closed–loop time constants are reduced for Test 2, dramatically improving the simulated controller tracking results as shown in Table 1 and Figure 20. The student

Table 1. Student Controller Tracking Results

Test #	Operation Mode	t_1 (s)	t_2 (s)	g_0 (1/s^2)	g_1 (1/s)	$\|e\|_{ave}$ (μm)
1	Simulation	0.1	0.5	–20	$1.1 \cdot 10^2$	63
2	Simulation	$1 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$-2.0 \cdot 10^3$	5.0	0.70
3	Simulation	$2.5 \cdot 10^{-3}$	$7.5 \cdot 10^{-3}$	$-5.3 \cdot 10^4$	$-4.1 \cdot 10^2$	0.72
4	Emulation	$2.5 \cdot 10^{-3}$	$7.5 \cdot 10^{-3}$	$-5.3 \cdot 10^4$	$-4.1 \cdot 10^2$	0.72
5	Implementation	$2.5 \cdot 10^{-3}$	$7.5 \cdot 10^{-3}$	$-5.3 \cdot 10^4$	$-4.1 \cdot 10^2$	0.94
6	Implementation	$2.5 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	$-8.0 \cdot 10^2$	$-4.8 \cdot 10^2$	0.63

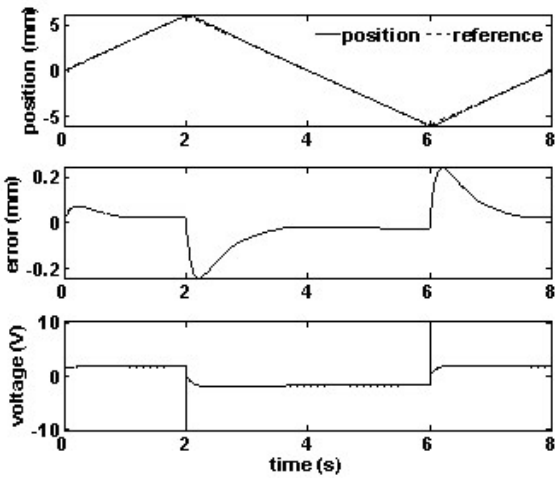

Fig. 19. Test 1 simulation controller tracking results with e_{max} = 243 μm and $|e_{ss}|_{ave}$ = 25.8 μm.

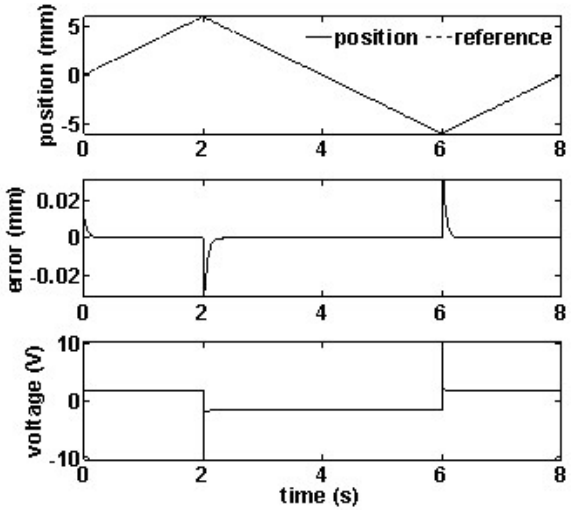

Fig. 20. Test 2 simulation controller tracking results with e_{max} = 31.2 μm and $|e_{ss}|_{ave} = 7 \cdot 10^{-3}$ μm.

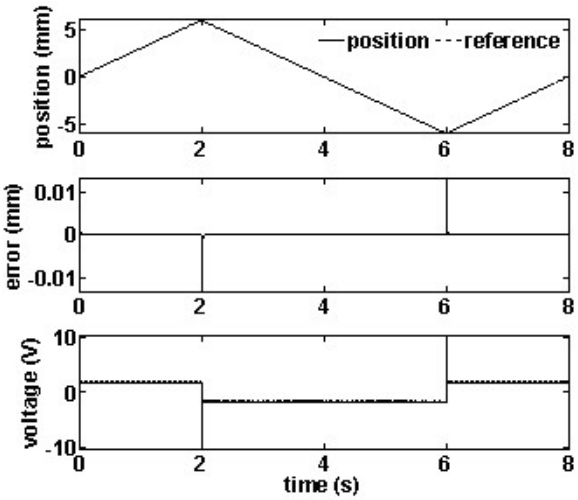

Fig. 21. Test 3 simulation controller tracking results with e_{max} = 13.5 μm and $|e_{ss}|_{ave} = 7 \cdot 10^{-3}$ μm.

again reduced the closed loop time constants in Test 3. This improved the simulated controller tracking results, shown in Table 1 and Figure 21, yielding a bounded steady–state error of $\pm 7 \cdot 10^{-3}$ μm and a maximum absolute error of 14 μm. The emulated controller tracking results from Test 4 are similar to the simulation results from Test 3 and are shown in Table 1 and Figure 22. The method used by the Simulink model solver for emulation mode is different from the method used by the Simulink model solver for simulation mode, causing minor differences between the results. The implementation controller tracking results from Test 5, shown in Table 1 and Figure 23, have a steady state error of ±1.6 μm and a maximum absolute error of 19 μm. In Test 6, the student made a final reduction to the desired closed–loop time constants yielding results, shown in Table 1 and Figure 24, with a steady state error of ±1.1 μm and a maximum absolute error of 19 μm. The difference between the linear axis model and the physical linear axis is evident when the implementation results are compared to the simulation and emulation results. The student controller is limited by the inaccuracy of the model used in the design. Friction was included in the linear axis model, but no friction compensation was included in the student controller design, resulting in a constant offset in the steady–state errors from zero. The maximum errors in the controller tracking results are caused by the discontinuity at the peaks in the triangular reference signal. These errors could be reduced by designing the reference signal to slow the linear axis velocity near the triangular peaks.

The primary mode the student used for controller tuning was the simulation mode. This is expected because simulation mode is the mode used for initial tuning, where several adjustments are made to the controller. Emulation mode was used solely as a check to verify the ability to operate the controller in real–time. Students used implementation mode to fine tune their controller, i.e. small adjustments to the closed–loop time constants. This is also expected because of the differences between the simulated and implemented closed–loop responses.

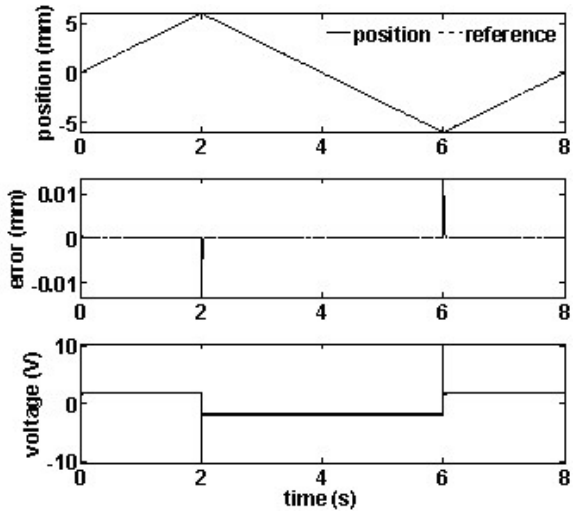

Fig. 22. Test 4 emulation controller tracking results with e_{max} = 13.5 μm and $|e_{ss}|_{ave} = 7 \cdot 10^{-3}$ μm.

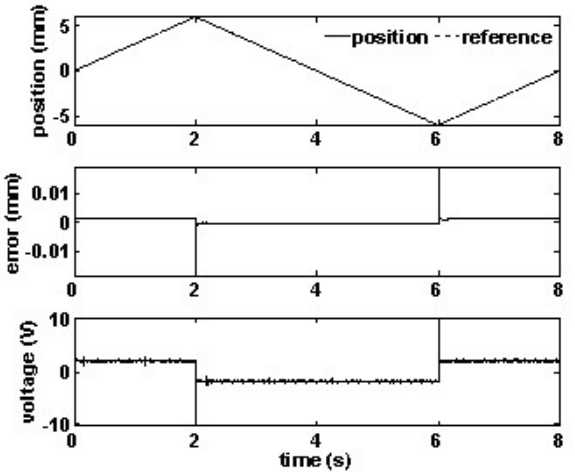

Fig. 23. Test 5 implementation controller tracking results with e_{max} = 18.7 μm and $|e_{ss}|_{ave}$ = 1.60 μm.

The usability results of the Linear Axis RDS are qualitative and reflect a good design. The average time for a student to complete the four tasks was 35 min, with a standard deviation of 21 min, a maximum time of 91 min, and a minimum time of 10 min. The post experiment survey revealed that the Linear Axis RDS experiment was the first time several of the students implemented a controller on a physical system. The students confirmed the Linear Axis RDS was a valuable tool in the controller implementation process. An evaluation of the Linear Axis RDS was performed and recommendations for improvement were provided [12]. All of the students were able to build the Simulink model containing the controller interface algorithm using the Build GUI. Multiple attempts to build the Simulink model were required by 9 of the 16 students due to confusion regarding the labeling of the popup menu options. A recommendation to improve the Build GUI is to change the label "model" on the model popup menu to refer to the linear axis model, which was confused with

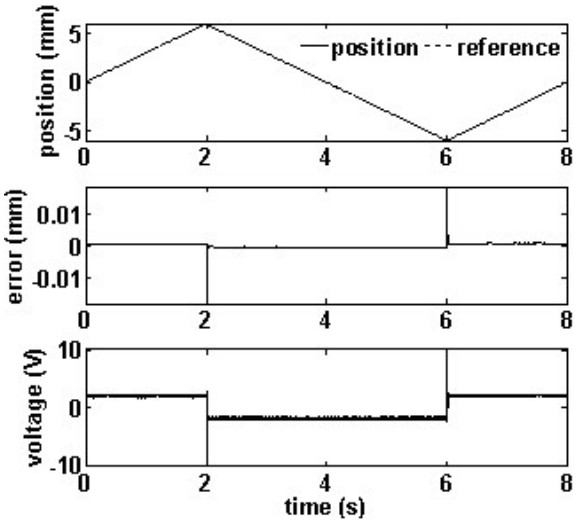

Fig. 24. Test 6 implementation controller tracking results with e_{max} = 19.1 μm and $|e_{ss}|_{ave}$ = 1.09 μm.

the Simulink model containing the student controller. Another recommendation is to eliminate the selected option text on the right of the Build GUI, shown in Figure 9, to reduce redundancy and increase room for more tips. All of the students were able to operate the Simulink model using the Operate GUI. Several students made multiple attempts to start and stop the Simulink model due to confusion regarding the operating status of the Simulink model. A recommendation to improve the Operate GUI is to provide more active visual confirmation, such as flashing the status text, when the operation of the Simulink model is complete. Of the 16 students, 13 were able to jog the linear axis using the Jog GUI. Recommendations to improve the Jog GUI include increasing the size of the jog status text and including a visual status update, such as a progress bar, while students are waiting for the jog Simulink model to convert into machine code and download to the target computer. The tips box and the Help GUI were beneficial in providing information to guide the students through the use of the Linear Axis RDS. Pre and post questionnaires revealed that the Linear Axis RDS helped the students to learn the course material and their interest in science and engineering increased after using the Linear Axis RDS.

5. SUMMARY AND CONCLUSIONS

A Rapid Development System provides a connection between a student, their controller, and the physical hardware, and is implemented on a linear axis system. MATLAB GUIDE, Simulink, and xPC Real–Time Environment are used to create the Linear Axis RDS. The Linear Axis RDS consists of three GUIs; Build, Operate, and Jog, which allow the student to automatically create a

connection between a controller and linear axis, operate the controller and linear axis, and jog the linear axis, respectively. A group of sixteen manufacturing automation and mechatronics students created controllers and used the Linear Axis RDS to implement their controllers on the linear axis. A usability study of the Linear Axis RDS was included in the test. A student example of the controller implementation process illustrated the steps the student used to tune the controller.

The Linear Axis RDS is successful in automatically creating a connection between a student controller and the linear axis system. The time required for a student to simulate, emulate, and implement a controller on the linear axis system using the Linear Axis RDS is, on average, 35 min, with a minimum time of 10 min. Little specific knowledge regarding the controller interface or linear axis physical interface is required to successfully implement a controller. The students were able to spend the majority of the time on the controller tuning process, as opposed to spending their time configuring hardware when it is not part of the course material. No friction compensation was included in the student controller design, resulting in an offset in the steady–state errors of the example student controller tracking results. The students were able to achieve satisfactory controller performance results, with the best having an average absolute error of 0.63 μm. From observations of the 16 student tests it was determined that the simulation mode was the primary mode used in the controller tuning process because it was the mode most used to make the initial adjusts to the controller. The emulation mode was used solely to check the controller real–time performance. Implementation mode was used to fine tune the controller due to the differences in the simulated and implemented closed–loop responses. The three GUIs are well designed with minor recommendations for improvement provided by the usability study. The reduced time to simulate, emulate, and implement a controller allows students to focus on controller design and experience the controller implementation process in a typical semester–long course. Pre and post questionnaires revealed that the Linear Axis RDS helped the students to learn the course material and their interest in science and engineering increased after using the Linear Axis RDS.

Acknowledgements—This material is based upon work supported by the National Science Foundation under Grant Number DUE–0736731. The results from the Linear Axis RDS usability study are the work of Vedant Jain, Dr. Hong Sheng, and Dr. Richard H. Hall.

REFERENCES

1. Y. Huang and J. H. Oliver, Integrated Simulation, Error Assessment, and Tool Path Correction for Five–Axis NC Milling, *Journal of Manufacturing Systems*, **14**(5), 1995, pp. 331–344.
2. R. Uptal, 3–D Object Decomposition with Extended Octree Model and its Application in Geometric Simulation of NC Machining, *Robotics and Computer Integrated Manufacturing*, **14**(4), 1998, pp. 317–327.
3. A. D. Spence and Y. Altintas, A Solid Modeler Based Milling Process Simulation and Planning System, *ASME Journal of Engineering for Industry*, **116**, 1994, pp. 61–69.
4. Machine Tool Agile Manufacturing Research Institute (MTAMRI), http://mtamri.me.uiuc.edu/, accessed 2009.
5. J. H. McDuffie, Using the Architecture Description Language Metah for Designing and Prototyping an Embedded Reconfigurable Sliding Mode Flight Controller, *AIAA/IEEE Digital Avionics Systems Conference*, Irvine, California, October 2002, pp. 8B11–8B17.
6. C. D. Vournas, E. G. Potamianakis, C. Moors and T. Van Cutsem, An Educational Simulation Tool for Power System Control and Stability, *IEEE Transactions on Power Systems*, **19**(1), 2004, pp. 48–55.
7. A. Burcin, C. A. Bohus, L. A. Crowl and M. H. Shor, Distance Learning Applied to Control Engineering Laboratories, *IEEE Transactions on Education*, **39**(3), 1996, pp. 320–326.
8. J. Liu and R. G. Landers, Integrated Modular Machine Tool Simulation for Education in Manufacturing Automation, *International Journal of Engineering Education*, **20**(4), 2004, pp. 594–611.
9. A. A. Wahyudi and M. J. E. Salami, Development of a Microcontroller-Based Control System with a Hardware in–the–Loop (HIL) Method for Control Education using MATLAB/Simulink/xPC Target, *International Journal of Engineering Education*, **21**(5), 2005, pp. 846–854.
10. A. Tornambe, Analysis and Synthesis in the Simulink Environment of Control Laws for DC Motors: A Real–time Implementation with the Microcontroller BASIC Stamp II as a Simple Tool for an Education Control Laboratory, *International Journal of Engineering Education*, **21**(5), 2005, pp. 814–837.
11. X. Zhao, R. G. Landers and M. C. Leu, Adaptive Control of Freeze–form Extrusion Fabrication Processes, *ASME Dynamic Systems and Controls Conference*, Ann Arbor, Michigan, October 20–22, 2008.
12. V. Jain, H. Sheng, R. H. Hall and M. Higlers, Introduction of an Iterative Approach to Evaluate Computer–Mediated Learning Technology, *Proceedings of the Interactional Conference on Information Systems*, Phoenix, Arizona, December 15–18, 2009:

Michael A. Fleming is a former graduate student from the Missouri University of Science and Technology (Missouri S&T) where he received his M.S. in mechanical engineering in 2009. He received his B.S in mechanical engineering from the University of Missouri—Rolla in 2007. He is currently employed as a manufacturing engineer for Alliant Techsystems, an aerospace and defense company. His research interests primarily focus on control systems, specifically dealing with control implementation in machining processes. he is a member of the International Mechanical Engineering Honor Society, Pi Tau Sigma. He was a recipient of the Chancellor's fellowship at Missouri S&T.

Robert G. Landers (landersr@mst.edu) is currently an Associate Professor of Mechanical Engineering at the Missouri University of Science and Technology (Missouri S&T). He received his B.S. degree from the University of Oklahoma in 1990, M.E. degree from Carnegie Mellon University in 1992, and Ph.D. degree from the University of Michigan in 1997, all in Mechanical Engineering. His research and teaching interests are in the areas of modeling, analysis, monitoring, and control of manufacturing processes (namely, metal cutting, friction stir welding, laser metal deposition, and freeze extrusion fabrication), control of alternative energy systems, and electro–hydraulic systems, integrated design and control, and digital control applications. He has over ninety technical refereed publications, including four book chapters, and has received funding from the National Science Foundation, US Department of Energy, Air Force Research Laboratory, US Department of Education, Society of Manufacturing Engineers, Missouri Research Board, and various companies. He received the Society of Manufacturing Engineers' M. Eugene Merchant Outstanding Young Manufacturing Engineer Award in 2004, has received nine faculty research and teaching excellence awards from Missouri S&T, and is a member of ASEE, ASME, IEEE, and SME. He is currently an associate editor for the *IEEE Transactions on Control System Technology* and the *ASME Journal of Dynamic Systems, Measurement, and Control*.