A Unified Approach for Classroom and Laboratory Control Systems Education

Zaher M. Kassas [*] **and Ricardo Dunia** [**]

[*] *Electrical and Computer Engineering Department, The University of Texas, Austin, TX (e-mail:* `zkassas@ieee.org`*).*
[**] *Chemical Engineering Department, The University of Texas, Austin, TX (e-mail:* `rdunia@che.utexas.edu`*)*

Abstract: In this paper, a unified approach for control systems education will be presented. This approach is a friendly software and hardware platform that provides students with the necessary tools for gathering data for system identification, designing controllers, simulating the closed-loop system, and implementing the controller on real-time hardware. The proposed approach has been used in introductory control systems courses and can be easily extended to advanced and research-based courses. Software-based classroom assignments and projects have been presented based on this platform. Moreover, these classroom assignments and projects were smoothly extended to hardware-based labs. The feedback conveyed by students due to the incorporation of this platform into control engineering curricula has been positive. Also, professors have been endorsing the use of this platform as part of teaching control systems engineering.

Keywords: Education, Software Tools, CAD, Simulation, Real-Time Implementation, Laboratory, Distributed Control

1. INTRODUCTION

Control systems engineers in the industry have been using computer-aided control systems design (CACSD) for design, simulation, and implementation well before software packages made their way to control systems engineering curricula. As these tools are becoming indispensable for teaching control systems theory and applications, we must ask: what are the important attributes of CACSD tools for the academia? Some of the desired attributes would entail such tools to have a steep learning curve, provide a rich library of functions, and be interactive in the sense that students may easily visualize the effects of adjusting different parameters of a system on the overall performance. Moreover, it would be ideal if such tools are not only utilized in relevant industries, but also could be straightforwardly "tailored" for classroom education. It would be desirable if these tools adopt an open source code methodology. Open source software makes a programming language viewable, understandable, and adjustable to the needs of its users.

The control system design process goes through a cycle of modeling and system identification, controller design, closed-loop simulation, and implementation (see Fig. 1) . In this paper a CACSD framework for such cycle will be presented. The framework is based on graphical programming using LabVIEW. LabVIEW stands for **Lab**oratory **V**irtual **I**nstrumentation **E**ngineering **W**orkbench, and the LabVIEW code is referred to as a Virtual Instrument (VI). A typical LabVIEW code consists of several blocks (VIs), which are "wired together". These wires carry the data flow and convey the information among the different blocks. The LabVIEW environment consists of two programming layers: a front panel and a block diagram. The front panel is built with controls and indicators, which are the interactive input and output terminals of the VI, respectively. These terminals are used to communicate data to and from the block diagram of the VI. The block diagram contains the source code of the VI that is compiled as the code is being developed.

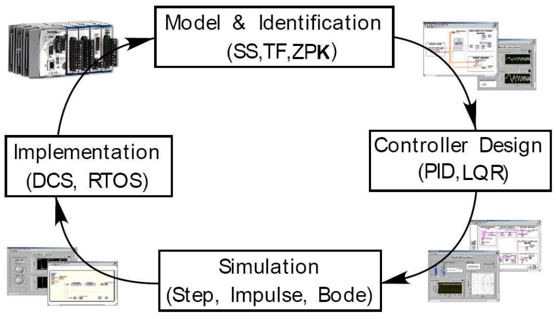


Fig. 1. Control System Design Cycle

LabVIEW has numerous built-in functions for conventional programming, file and instrument input/output (I/O), data communication, state charts, mathematics, signal processing, system identification, control systems design, and dynamic systems simulation, to name a few. LabVIEW's graphical nature makes it very attractive for control systems education, in which block diagrams are an indispensable tool to convey control systems theory Keller [2003], Bebyo et al. [2003], Bishop [2006], Aradi and Lipovski [2003]. LabVIEW also provides textual math environments that could be embedded inside the graphical programming interface as textual nodes. These nodes represents a textual editor inside the graphical block diagram, in which equations can be defined, and an interface permits the exchange of variables and results with the rest of the graphical programming code. The “Formula Node” accepts “C” syntax, whereas the “MathScript Node” accepts the standard syntax that is used in many control engineering textbooks.

This paper is organized as follows. Section 2 introduces some of the essential tools that are encountered in modeling and system identification, control design, and simulation of the control system. Section 3 presents the role of real-time (RT) implementation and distributed control systems. Section 4 outlines the application of the proposed framework on a three degrees-of-freedom (3DOF) helicopter case study. Concluding remarks are discussed in Section 5.

2. SYSTEM IDENTIFICATION, CONTROLLER DESIGN, AND SIMULATION

This section will outline some of the essential tools that are commonly encountered in the process of modeling and system identification, controller design, and control system simulation. These tools are available as libraries of VIs in the System Identification and Control Design Toolkits and Simulation Module in LabVIEW. The VIs in these libraries are functionally organized into subpalettes as illustrated in Fig. 2, 3, and 4. Students may use these VIs as “building blocks” for modeling, system identification, controller design, and simulation of the system under study in open- and closed-loop configuration. Moreover, students may view the underlying algorithms for such VIs and customize them as necessary, while utilizing the rich interactive programming environment of LabVIEW.

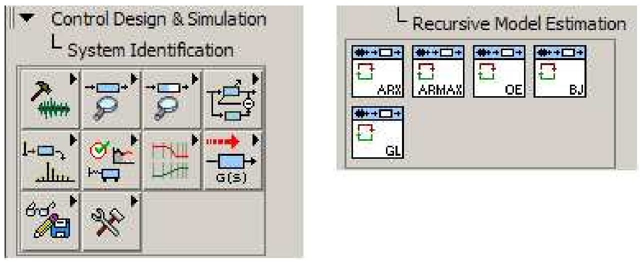


Fig. 2. System Identification Library Palette. Shown (expanded) is the Recursive Model Estimation Subpalette.

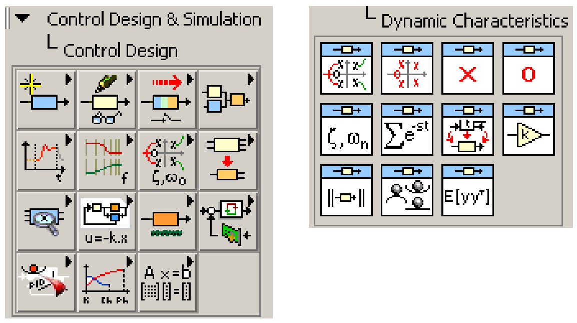


Fig. 3. Control Design Library Palette. Shown (expanded) is the Dynamic Characteristics Subpalette.

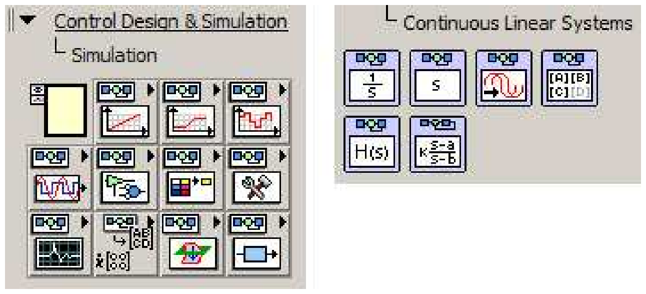


Fig. 4. Simulation Library Palette. Shown (expanded) is the Continuous Linear Systems Subpalette.

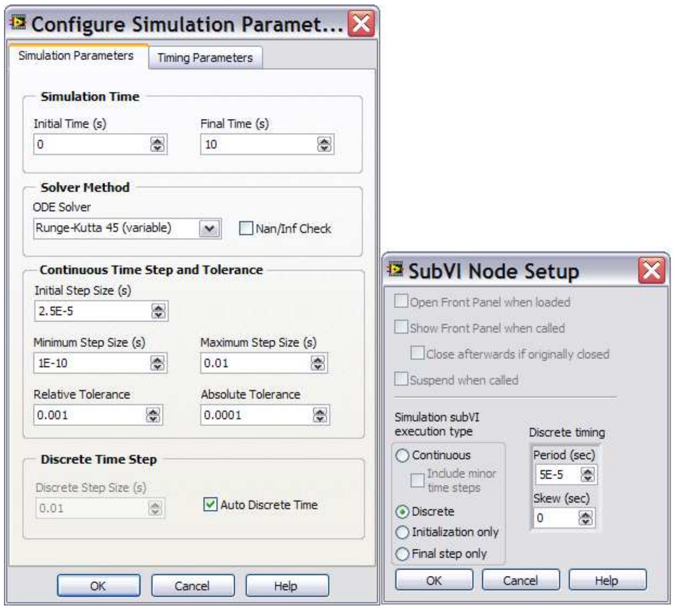
First, the VIs in the System Identification Toolkit are grouped into 10 categories that are functionally organized into subpalettes, NI [August, 2006]. Some of the tasks that can be accomplished with this toolkit include data preprocessing, parametric model estimation, partially known (grey-box) model estimation, recursive model estimation, non-parametric model estimation, model validation, model analysis, model conversion, model management, and various utilities.

Second, the VIs in the Control Design Toolkit are grouped into 12 categories that are functionally organized into subpalettes, NI [August, 2007a]. Some of the tasks that can be accomplished with this toolkit include model construction; gathering model information; model conversion; model interconnection; time response; frequency response; dynamic characteristics; model reduction; state-space model analysis; state-feedback design; stochastic systems; controller, observer, and Kalman filter implementation; proportional-integral-derivative (PID) design; model-predictive control (MPC); and solvers for commonly encountered equations (e.g. Lyapunov, Riccati, integrals involving matrix exponentials, etc).

Third, the VIs in the Simulation Module are grouped into 12 categories that are functionally organized into subpalettes, NI [August, 2007c]. The left uppermost entry in this palette is the Simulation Loop. All VIs that are placed inside a Simulation Loop will be executed recursively based on prescribed simulation parameters. These parameters dictate how the loop is executed by solving an ordinary differential equation (ODE) using a specified continuous time solver or by solving a difference equation at specified discrete-time steps. Moreover, individual VIs inside the Simulation Loop maybe configured to run at discrete or continuous time-steps or to run at the initial or final time-steps only (see Fig. 5).

Moreover, the timing behavior of the Simulation Loop can be set into software or hardware timing source. If a software timing source is specified, the time steps of the simulation execute as fast as the computer hardware permits. If a hardware timing source is specified, the major time steps execute at the periodic rate that timing source specifies. Moreover, we may specify whether the VIs inside the Simulation Loop will run either as continuous-time blocks, as discrete-time blocks, at the initialization step only, or at the final step only.

Some of the tasks that can be accomplished with this module include implementation of continuous linear systems, implementation of nonlinear systems, implementation of


Configure Simulation Paramet...

Simulation Parameters | Timing Parameters

Simulation Time
Initial Time (s): 0
Final Time (s): 10

Solver Method
ODE Solver: Runge-Kutta 45 (variable)
☐ Nan/Inf Check

Continuous Time Step and Tolerance
Initial Step Size (s): 2.5E-5
Minimum Step Size (s): 1E-10
Maximum Step Size (s): 0.01
Relative Tolerance: 0.001
Absolute Tolerance: 0.0001

Discrete Time Step
Discrete Step Size (s): 0.01
☑ Auto Discrete Time

OK | Cancel | Help

SubVI Node Setup
☐ Open Front Panel when loaded
☐ Show Front Panel when called
☐ Close afterwards if originally closed
☐ Suspend when called

Simulation subVI execution type
○ Continuous
☐ Include minor time steps
● Discrete
○ Initialization only
○ Final step only

Discrete timing
Period (sec): 5E-5
Skew (sec): 0

OK | Cancel | Help

Fig. 5. Simulation Loop Parameters

discrete linear systems, signal generation, signal arithmetic, lookup tables, graph utilities, trim and linearize, optimal design, and RT implementation of state-feedback controllers, observers, and Kalman filters.

3. REAL-TIME IMPLEMENTATION AND DISTRIBUTED CONTROL

A real-time (RT) control application repeatedly performs user-defined tasks with a specified elapsed time between the operations. There are different ways to describe an RT control application, such as *control loop cycle time*, *determinism*, and *jitter*. Determinism measures the consistency of the specified time interval between the user-defined tasks. Jitter measures the amount of time (error) that the loop cycle time varies from the desired time. Jitter can be measured as the maximum difference between any individual *actual* time delay and the *specified* time delay of a system. The RT operating system (RTOS) guarantees operation within a time-bounded amount of jitter. If the jitter is not time-bound, the control system may suffer from instability as the control algorithm is typically calculated assuming a predetermined fixed time interval. Such a difference between the fixed time interval and the actual time interval leads to performance degradation.

In a number of control systems applications, the control system is usually distributed across multiple computational platforms communicating over a closed network. Distributed control systems (DCSs) are found in many control systems application including those applications where the control system is used in manufacturing industries, such as industrial motion control and precision machine control; and those applications where the control system is embedded into a larger system, such as automotive or flight systems. The computational and network communication delays can degrade the performance of the control system and should be taken into account when implementing DCSs. Consequently, it is crucial that control systems students have exposure to simple DCS cases in laboratory experiments to demonstrate the fact that "classical" control systems designed by "pencil-and-paper" behave differently when implemented in a DCS fashion.

Many discrepancies between simulated and implemented control systems are due to *modeling errors* (structure or parameter uncertainty), *truncation errors* (in the computational platform where the controller is deployed), and *timing errors* (added/reduced delays and jitter due to RTOS or the lack of).

A simple DCS experimental platform is depicted in Fig. 6. In this platform, the plant, sensors, estimator, and controller may be distributed in different configurations to emphasize different system behavior. Such configurations could be

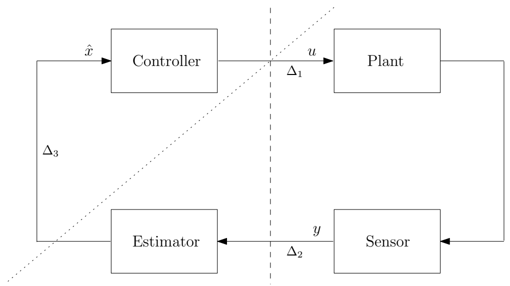


Fig. 6. Distributed Control Illustration

- Plant, sensors, estimator, and controller all on one computational platform. This is the most straightforward "classical" configuration used in most control labs. Here, the communication delays Δ_1, Δ_2, and Δ_3 maybe ignored.
- Plant and sensors on one computational platform and controller and estimator on another, i.e. the DCS is partitioned across the dashed line. Here, the I/O delays Δ_1 and Δ_2 effects may be significant and worth studying.
- Plant, sensors, and estimator on one computational platform and controller on another, i.e. the DCS is partitioned across the dotted line. Here, the delays Δ_1 and Δ_3 may be significant. In such configuration, students may exploit the predictive observer or the Kalman filter concepts to account for such delays.
- Plant and sensors on one computational platform, estimator on another, and controller on a third one. Here, delays Δ_1, Δ_2, and Δ_3 are all present and maybe significant. Their effects can be studied with this framework.

On the computational platform, it is imperative that all the operations it performs are deterministic or can be preempted by higher priority operations. With DCSs, there needs to be a communication mechanism to distribute data between each node in the network. Most communication methods such as TCP/UDP, serial, etc. are intrinsically not deterministic. As such, many DCSs will turn to a hardware solution like distributed shared memory (reflective memory) boards for a solution. In LabVIEW, there is a software mechanism (the shared variable) that can be used to guarantee deterministic communication between nodes in the DCS, NI [August, 2007b]. The LabVIEW shared variable can be easily configured in two different modes when applying a DCS. First, we can use a network-published shared variable with an RT first-in-first-out

(FIFO) to share data between a VI running on an RT target and a host computer that is acting as a human machine interface (HMI) without affecting the determinism of the VI. Second, we can use time-triggered shared variables to transfer data across a network deterministically using a network card. When using time-triggered shared variables to share data across a time-triggered network, current NI hardware and software can close deterministically a network loop at rates approaching 5 kHz.

4. CASE STUDY

This section will illustrate the use of the proposed framework for system and parameter identification, control system design, simulation, and controller implementation on a Quanser [1] 3DOF helicopter. Fig. 7 illustrates the hardware used in the lab, where encoders and two voltage amplifiers are connected to the helicopter to measure the outputs and apply the control action to the system under study. The RT execution is accomplished on a PXI RT controller, and a laptop is used for control design and offline simulation of the system as well as serving as the host machine (HMI) in the RT system.

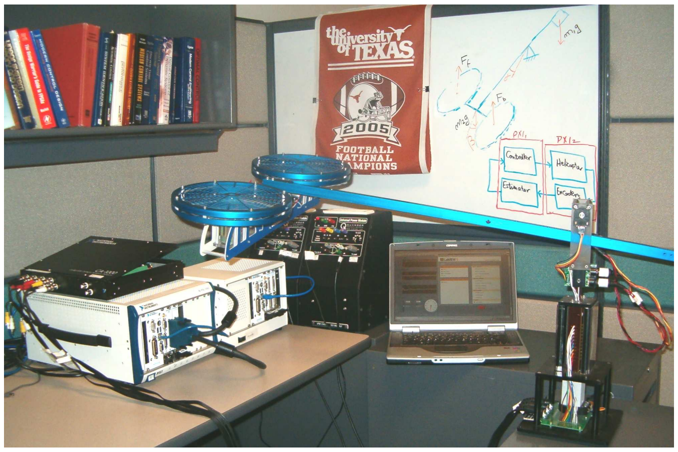


Fig. 7. 3DOF Helicopter System Lab

The helicopter system is modeled as a nonlinear dynamical system with two inputs, represented by the voltages applied to the front and back motors; three outputs that define the elevation, travel, and pitch of the helicopter; and six states: the elevation angle (α), the pitch angle (β), the travel angle (γ), and their respective first derivatives. The friction parameters are denoted by f_i, where i represents a state or degree of freedom for the system. The set of nonlinear differential equations governing the dynamics of the helicopter are given by

$$\begin{aligned} J_e\ddot{\alpha} &= \left[(F_f + F_b)cos(\beta) - m_2 g\right] l_2 cos(\alpha) \\ &\quad + m_1 g \left[l_0 cos(\psi) + l_1 cos(\alpha)\right] - f_\alpha(\dot{\alpha}) \quad (1) \\ J_p\ddot{\beta} &= (F_b - F_f)cos(\beta)l - f_\beta(\dot{\beta}) \quad (2) \\ J_e\ddot{\gamma} &= (F_f + F_b)sin(\beta)l_2 cos(\alpha) - f_\gamma(\dot{\gamma}). \quad (3) \end{aligned}$$

The table above provides the physical parameters of the system. The nominal operating values for all the output

Parameter/State	Description
J_e	Inertia at elevation-travel axis
J_p	Inertia at pitch axis
F_f, F_b	Front and back motor forces
V_f, V_b	Front and back motor voltages
m_1	Counter weight mass
m_2	Helicopter mass
ψ	Counter weight deflection
l_0, l_1	Counter weight distances to pivot
l_2	Helicopter distance to pivot
l	Motor distance to pitch
α	Elevation angle
β	Pitch angle
γ	Travel angle
g	Gravity constant
$f_x(\dot{x})$	friction term for state x

angles is zero degrees. An Input of 0.45 Volts is required on the back (V_b) and front (V_f) motors to keep the helicopter flying at zero elevation. The voltage difference, $V_b - V_f$, provides a pitch angle that affects the travel of the helicopter.

All the numerical values for the parameters in Table 1 are known, except for the friction terms. In order to identify these parameters, grey-box parametric identification is required. In this respect, the structure of the nonlinear model of the helicopter is known. The students determine the optimal estimates of the friction parameters utilizing I/O data. Since the helicopter system is open-loop unstable, a basic PID controller was implemented and the PID gains were tuned to stabilize the system and gather the data for grey-box parametric identification. The I/O data representing the stimulus/response signals were saved into files. In this stage of the experiment the students are exposed to non-model-based control systems techniques. These techniques are embraced by a good number of industries to achieve feedback control. Fig. 8 illustrates the block diagram for the grey-box parametric identification. Here, the "Estimate Parameters" VI takes the following inputs: the stimulus/response signals from the saved files, the initial estimates of the friction terms along with their lower and upper limits, and a path to a file that contains the model of the system over which we apply grey-box parametric identification. The model in this case consists of the closed-loop system, which contains the helicopter system with the PID controller. The helicopter model file is illustrated in Fig. 9. Here, the nonlinear differential equations (1)-(3) governing the dynamics of the helicopter were included into the "Formula Node". Input and output terminals were created around this node in order to pass and retrieve results from this textual math environment.

Once the numerical values of the friction parameters have been identified, the students linearized the system described in (1)-(3) using the linearize tool in the Simulation Module. A linear representation based on a numerical approximation of the open-loop system around nominal operating conditions is obtained by simply selecting the file that contains the nonlinear dynamical expressions. The tool that calculates the linearized system also allows to preset the nominal conditions around which the nonlinear subsystem should be linearized. It also permits to fix some inputs, outputs, or states while the linear system is being estimated.

[1] www.Quanser.com

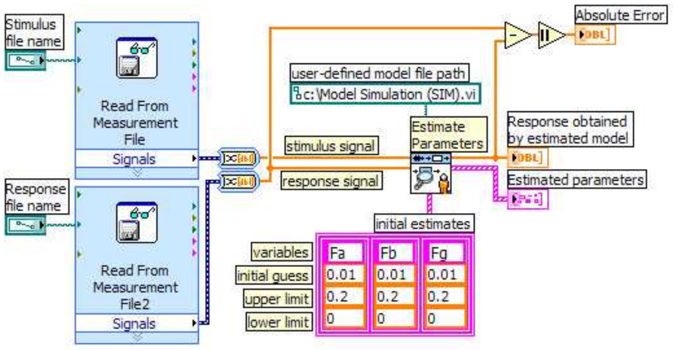
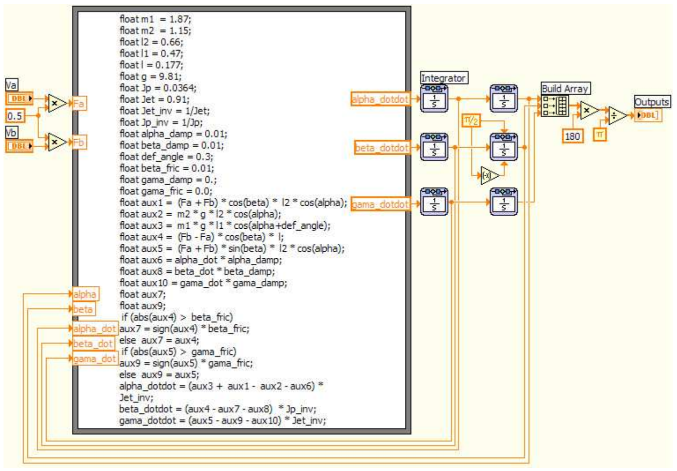


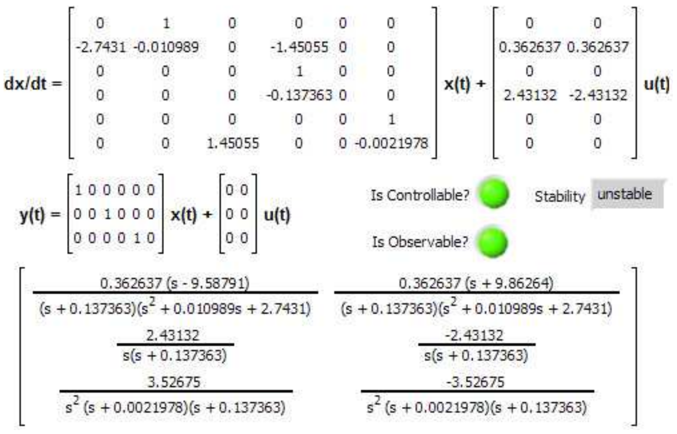
Fig. 8. Block Diagram for Grey-Box Parametric Identification



```
float m1  = 1.87;
float m2  = 1.15;
float l2 = 0.66;
float l1 = 0.47;
float l = 0.177;
float g = 9.81;
float Jp = 0.0364;
float Jet = 0.91;
float Jet_inv = 1/Jet;
float Jp_inv = 1/Jp;
float alpha_damp = 0.01;
float beta_damp = 0.01;
float def_angle = 0.3;
float beta_fric = 0.01;
float gama_damp = 0.;
float gama_fric = 0.0;
float aux1 =  (Fa + Fb) * cos(beta) *  l2 * cos(alpha);
float aux2 =  m2 * g * l2 * cos(alpha);
float aux3 =  m1 * g * l1 * cos(alpha+def_angle);
float aux4 =  (Fb - Fa) * cos(beta) *  l;
float aux5 =  (Fa + Fb) * sin(beta) *  l2 * cos(alpha);
float aux6 = alpha_dot * alpha_damp;
float aux8 = beta_dot * beta_damp;
float aux10 = gama_dot * gama_damp;
float aux7;
float aux9;
 if (abs(aux4) >  beta_fric)
aux7 = sign(aux4) * beta_fric;
else  aux7 = aux4;
 if (abs(aux5) >  gama_fric)
aux9 = sign(aux5) * gama_fric;
else  aux9 = aux5;
alpha_dotdot = (aux3 +  aux1 -  aux2 - aux6) *
Jet_inv;
beta_dotdot = (aux4 - aux7 - aux8)  * Jp_inv;
gama_dotdot = (aux5 - aux9 - aux10) * Jet_inv;
```

Fig. 9. Nonlinear Model for 3DOF Helicopter

Once the linearized model was obtained, students were able to analyze open-loop stability, controllability, and observability of the system. The state-space (SS) linear system was converted to a transfer function (TF), and both were visually rendered. Fig. 10 shows the front panel for this model analysis stage.



$$dx/dt = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -2.7431 & -0.010989 & 0 & -1.45055 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -0.137363 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1.45055 & 0 & 0 & -0.0021978 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 \\ 0.362637 & 0.362637 \\ 0 & 0 \\ 2.43132 & -2.43132 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} u(t)$$

Is Controllable? ●

Stability unstable

Is Observable? ●

$$\begin{bmatrix} \dfrac{0.362637\,(s - 9.58791)}{(s + 0.137363)(s^2 + 0.010989s + 2.7431)} & \dfrac{0.362637\,(s + 9.86264)}{(s + 0.137363)(s^2 + 0.010989s + 2.7431)} \\ \dfrac{2.43132}{s(s + 0.137363)} & \dfrac{-2.43132}{s(s + 0.137363)} \\ \dfrac{3.52675}{s^2\,(s + 0.0021978)(s + 0.137363)} & \dfrac{-3.52675}{s^2\,(s + 0.0021978)(s + 0.137363)} \end{bmatrix}$$

Fig. 10. Front Panel for Linearized System Analysis

Control design methods can be made interactive in LabVIEW, i.e. step-response plots and closed-loop specifications can be calculated as the controller parameters are adjusted without stopping the program execution. In this illustration, it was desired to have a step-response with overshoot $\leq 5\%$ and settling time ≤ 5 seconds. The root-locus method was first used to design a controller to meet the desired closed-loop response specifications. These specifications were met by employing a PID controller. Fig. 11 illustrates the block diagram for model analysis and controller design using the root-locus and PID methods. Fig. 12 shows the interactive front panel. Is is important to mention that these tools are interactive in the sense that students can visualize the effect of each controller parameter in the time response plot.

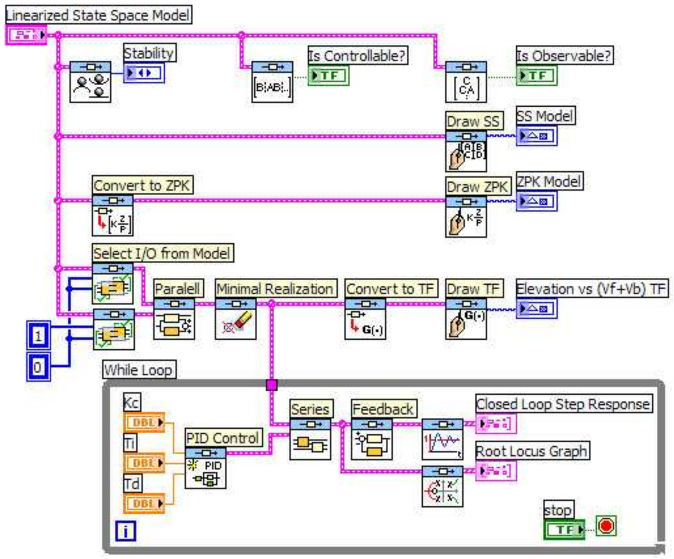


Fig. 11. Block Diagram of Linearized System Analysis and Controller Design

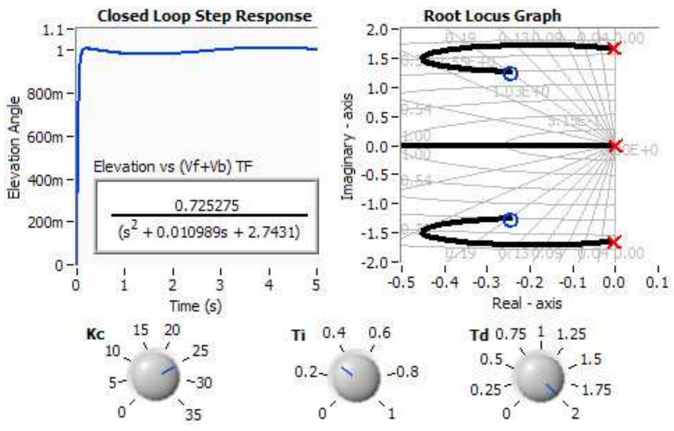


Fig. 12. Front Panel of Interactive Root-Locus and PID Tuning

Next, a more advanced controller was designed in the lab based on optimal control theory. In this respect, the continuous-time helicopter model was discretized, based on a desired sampling time, and a Linear Quadratic Regulator (LQR) controller was designed. The helicopter system was simulated off-line inside the Simulation Loop along with the controller to “mimic” the closed-loop system behavior. Since not all the system states were measurable, an observer was used to estimate the states for state-feedback control. In order to have a more realistic “feel” of the controlled helicopter simulation, the Picture

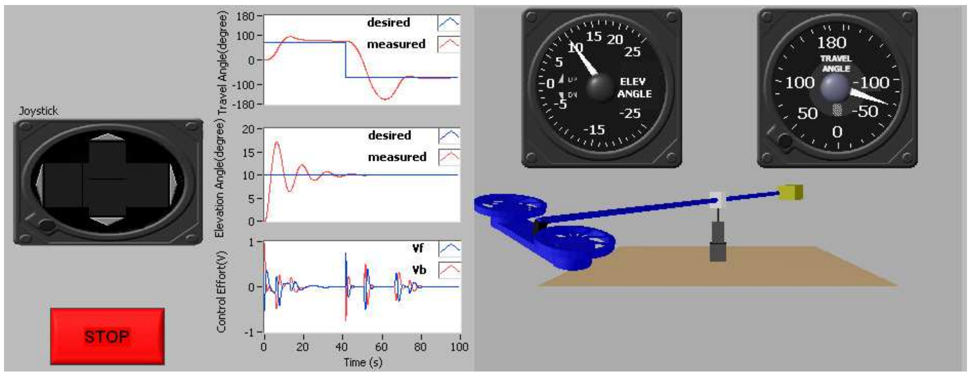


Fig. 13. Front Panel of Simulation of 3DOF Helicopter System with LQR Control and Predictive Observer

3D Toolkit was used. The Picture 3D Toolkit consists of a set of functions that allow the modeling and rendering of three-dimensional (3D) scenes for advanced visualization in LabVIEW. This toolkit has a library to build and animate 3D scenes, as well as import 3D objects from ASE, STL, and WRL file formats. Fig. 13 illustrates the front panel for the simulation and Fig. 14 illustrates the block diagram for the observer design, LQR controller design, and closed-loop system simulation.

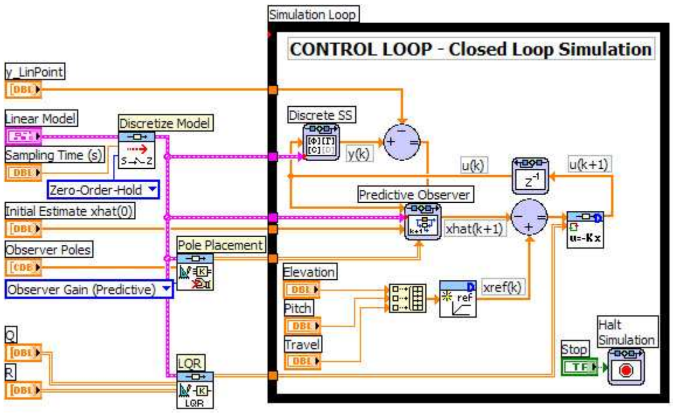


Fig. 14. Block Diagram of 3DOF Helicopter System simulation with LQR Control and Predictive Observer

Finally, the students implemented the controller in RT using the actual helicopter hardware. In this respect, DAQmx was used for data acquisition and control action applications. The sampling time used in model discretization has been chosen according to the DAQmx sampling rate and execution rate of the Simulation Loop. The simplicity and clearness with which we transition from off-line simulation of the helicopter system to RT implementation can be visualized in Fig. 15. In such case, the state-space subsystem that was *simulating* the plant behavior has been disconnected and replaced by I/O read/write DAQmx VIs.

5. CONCLUSION

This paper presented a framework for teaching control systems students how to apply the control design cycle presented in Fig. 1 in a methodological sequence. The framework provided simple tools to perform all the tasks associated with such a design cycle. Furthermore, the framework was friendly, interactive, and easy to configure.

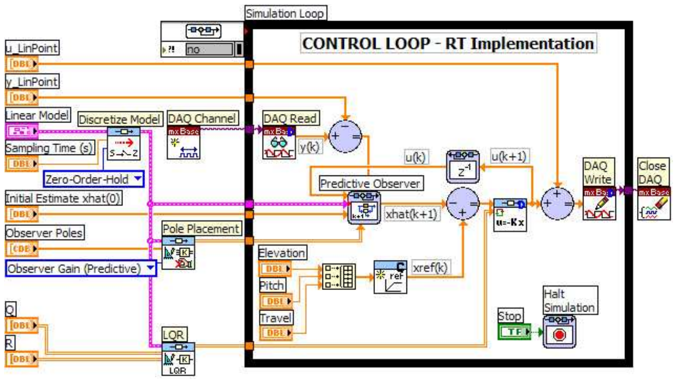


Fig. 15. Block Diagram of Real-Time Implementation of LQR Controller with Predictive Observer

The application of this framework to a 3DOF helicopter system for system identification, analysis, control design, simulation, data acquisition, and RT implementation has shown the advantages of using this platform for the control design cycle. This platform allowed the students to focus on the academic control concepts, while reducing the typical overhead spent configuring software and hardware. These software tools can be readily extended to teaching more advanced topics such as nonlinear and adaptive control systems, Kalman filtering, and distributed control systems.

REFERENCES

P. Aradi and G. Lipovski. Labview as a teaching aid for control engineering. *IFAC Advances in Control Education*, pages 321–326, 2003.

I. Bebyo, G. Lipovski, and J. Kovacs. Advanced control: Simulation tools in labview environment. *IFAC Advances in Control Education*, pages 237–241, 2003.

R. H. Bishop. *LabVIEW 8 Student Edition*. Prentice Hall, 2006.

J. Keller. Interactive control system design. *IFAC Advances in Control Education*, pages 333–328, 2003.

NI. *LabVIEW System Identification Toolkit User Manual*. National Instruments, August, 2006.

NI. *LabVIEW Control Design Toolkit User Manual*. National Instruments, August, 2007a.

NI. *LabVIEW Help, Real-Time Module*. National Instruments, August, 2007b.

NI. *LabVIEW Simulation Module User Manual*. National Instruments, August, 2007c.