# PROGRAMMING FROM THE GROUND UP IN CONTROLS LABORATORIES USING GRAPHICAL PROGRAMMING

**Todd D. Murphey** [*,1] **Jeannie Sullivan Falcon** [**]

[*] *Electrical and Computer Engineering*
*University of Colorado*
*Boulder, CO, USA*
[**] *National Instruments*
*Austin, TX, USA*

Abstract: This paper describes a recent, innovative educational approach taken at the University of Colorado for teaching a senior-level undergraduate controls course. By using software written in a graphical programming language (in this case LabVIEW$^{TM}$) to control the hardware, the students were able to do all the programming themselves. The advantages of this approach over "canned" lab approaches are many. The students feel more responsible for the final product and they are able to apply control techniques learned in class in a more fundamentally creative way. We present some anecdotal evidence from the class taught in Autumn term, 2005, that suggests that students will often use tools from beyond the scope of the class to solve problems in creative and occasionally unexpected ways.

Keywords: Education, Control Education, Laboratory Education

## 1. INTRODUCTION

This paper describes an innovative approach to teaching a controls laboratory aimed at upper division undergraduate students. We use a less structured approach to the laboratory than is typically used. In particular, the students are in charge of all programming, the labs are more open-ended than typical controls labs, and the students work in rotating groups (which forces them to incorporate "legacy" code from previous groups). This course structure gives the students a sense of the difficulties and ambiguities often associated with solving real problems. The basic thesis of this paper is that if one uses a graphical programming environment, a course can be designed to give students much more control by allowing them to do all the programming and giving them more open-ended assignments. The consequence of this is a deeper understanding of the application of controls to real problems, which we illustrate anecdotally.

Traditional control courses often involve pre-written, or "canned," software (often written in an intermediate-level programming language like C or C++). Moreover, partially because of the constraints imposed by the pre-written software, labs are highly regimented. This choice is made largely because it is not feasible for the students to write all the code–there simply is not enough time during a one-term course for students to cover as much material as we would like as well as do all the programming themselves in an intermediate-level language.

[1] This work was supported by National Instruments, the National Instruments Foundation, and the University of Colorado.

By using a graphical programming environment, however, it is possible to create a course where students do nearly all of the programming themselves. This prepares them well for jobs where they will be expected to know how to integrate software and hardware, and it additionally helps them learn controls concepts more deeply by allowing them more flexibility in exploring control designs. Moreover, this course structure has other advantages as well. In particular, it is possible to give students more open-ended labs (that do not include step-by-step instructions), which is typically more engaging for the students. This also leaves more room for creative application of concepts the students are learning.

The decision to let students write all of the code was inspired by noticing that graphical programming environments (such as LabVIEW, Matlab/Simulink$^{TM}$, and others) are faster environments for programming. We used LabVIEW for all laboratory development and students used LabVIEW for analysis, simulation, and experimental implementation. (For a useful, if not comprehensive, introduction to LabVIEW, see (Bishop, 2003).)

Over the course of the term, the students learned to write all of their own code, both for simulation and for hardware experiments. They quite literally started with blank code, or would incorporate student-created code from previous labs. They used LabVIEW in a manner similar to that reported in (Benyo *et al.*, 2003). However, instead of the instructor and teaching assistant writing the code for the students, the students were given very short tutorials on graphical programming and wrote the code themselves.

This paper is organized as follows. Section 2 discusses the hardware setup, the lab organization, the tasks the students were expected to complete, and the general level of background students had coming into the course. Section 3 has anecdotal evidence that this structuring of the class led to more internalization of controls techniques as well as more creative approaches to open-ended problems. Section 4 discusses the necessary trade-offs made when teaching a course using the described approach. Section 5 discusses the various advantages and disadvantages to the instructor using this method and Section 6 describes student reactions to the course, including course evaluation results. We end with conclusions in Section 7.

## 2. LAB ORGANIZATION

The Electrical and Computer Engineering Department at the University of Colorado offers two senior elective control courses: Control Systems Analysis (ECEN 4138) and Control Systems Laboratory (ECEN 4638). Both courses typically have 20-30 students. This last autumn was the first time in the last several years the laboratory was offered using hardware, made possible through funds provided by the university and National Instruments. Typically students take both courses concurrently, although they may take the lab if they have taken the lecture course previously.

The goal of the class was two-fold. First and foremost, the class aimed to reinforce the students' understanding of introductory controls concepts–Modeling, PID, Root Locus, Bode, Nyquist, and state-space techniques. A secondary goal of the class was to reinforce the fact that the models we write down do not *actually* reflect the true dynamics of a system; they only approximate them. We wanted to see students discover control designs that work in principle but fail in practice. The more students are in charge of discoveries of this sort, the more we expect they will retain key concepts past the class.

All the students worked in randomly selected, rotating groups of three people each. These groups changed five times over the course of the term. This structure forced the students to combine and use legacy code from each member's previous group. One of the byproducts of this was that by the end of term, all the students had very good, stable code for running experiments. Moreover, they typically understood the code and could replicate it as needed.

The course included the following topics.

(1) Modeling
(2) System Identification
(3) PID tuning
(4) Lead/Lag Controllers
(5) Root Locus Analysis
(6) Bode/Nyquist Analysis

Moreover, the students did a final project. [2]

### *2.1 Student Background Prior To This Laboratory*

Students were predominantly seniors with a standard electrical engineering background. All but one student was taking the lecture course (ECEN 4138) concurrently (the one student had taken it a year previously). Most had never seen any more controls than a PID controller.

Almost none of the students had any graphical programming experience, but several had seen Matlab/Simulink and LabVIEW in demonstrations in other classes. Hence, students view these languages more as interfaces than computer languages. Because of this, the students had some-

[2] Although this last term we did not have time for a state-space lab, this was largely due to the course being in its infancy, and we anticipate that we will include state-space design in Autumn term, 2006.

what of a predisposition against using the software, but this was overcome by pointing out that it would be nearly impossible to have them write all the code in C or C++.

*2.2 Hardware*

The experimental set-up used in the laboratory was comprised of a torsional disk system from Education Control Products, seen in Figure 1. These experiments are relatively robust, which was particularly important given the level of control we gave the students. The software used was LabVIEW 7.1 (as it has been used in other classrooms, see (Benyo *et al.*, 2003)), the Simulation Module 1.0 (Haugen, 2005*b*) and Control Design Toolkit 2.0 (Haugen, 2005*a*) that both run as part of LabVIEW. Our input/output capabilities were all provided by a National Instruments FPGA (NI FPGA 7831R) (Falcon, 2006; Limroth *et al.*, 2005). The students modeled the system, performed system identification, applied the basic techniques of PID tuning, Lead/Lag, Root Locus, Bode, and Nyquist analysis. The course ended with a final project and competition.

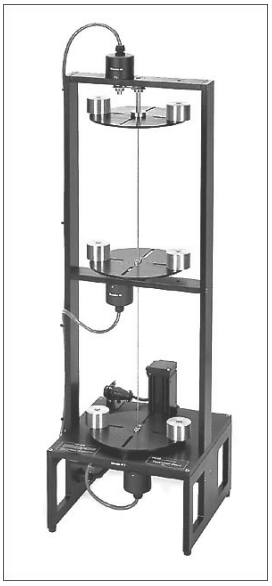

Fig. 1. The ECP Model 205a Torsional Plant (photograph taken from the manufacturer's website *http://www.ecpsystems.com*)

This system has a 2 $N \cdot m$ DC motor at the bottom that drives the first disk. Two more disks are connected via a torsional spring that allow for torsional displacement between disks with an associated torsional spring constant. Thus, the dynamics of this system are sixth-order, making this a nontrivial control system. Moreover, weights (also seen in Fig. 1) are included that allow the user to change the inertia of each disk. This system is basically linear, making it a good choice for a linear controls laboratory. The biggest downside to this experiment is that it is neutrally stable, unlike other classic control systems such as an inverted pendulum. (For a description of using these plants with LabVIEW, see (Bishop and Lin, 2005).)

*2.3 Programming From the Ground Up*

As previously mentioned, the students wrote all of the code for the course. Hence, for simulation and analysis, the students wrote their own simulation code from scratch and integrated analysis tools (such as root locus, bode, and Nyquist plots) into their simulation. With the exception of some basic hardware safety (such as turning the system off if the torsion between two disks became too high), the students also wrote all of the experimental software. The communication and hardware safety was given to them in the form of a template, an example of which is seen in Fig.2.

Labs were generally open-ended (with the exception of the system identification laboratory). They focused on problem solving rather than following instructions. In particular, all labs were divided into high-level tasks (known in the education community as "authentic" tasks), most of which were not mathematically defined for the student, described next.

*2.4 Examples of Assigned Tasks*

In designing the tasks for this lab, we were trying to avoid the more traditional approach of giving students a long series of steps to follow. Instead, we tried to give them tasks that were more similar to verbal tasks they might be asked to accomplish in a job, but that nevertheless required the technical detail and insight generally learned in the lecture course.

Examples of assigned tasks included the following:

(1) Model the ECP system as a single-input-single-output system first using the bottom disk as an output and then using the top disk as an output. Simulate this using both the symbolic transfer function and symbolic state-space model representations in LabVIEW. (Note to reader: The sample code given to students regarding how to create state space models in LabVIEW consisted of what is contained in Fig.3–more than enough to solve the problem, but in no way a "canned" solution for them to use.)
(2) Design a controller that is stable for all possible locations of two weights on disk 3. Is it possible to do this and guarantee a rise time of 0.5 second?
(3) Run the experiment and your simulation simultaneously. What are the differences? If there are differences, where are they coming from and can you fix them?
(4) Consider the system with the bottom disk as an output and with the top disk as an

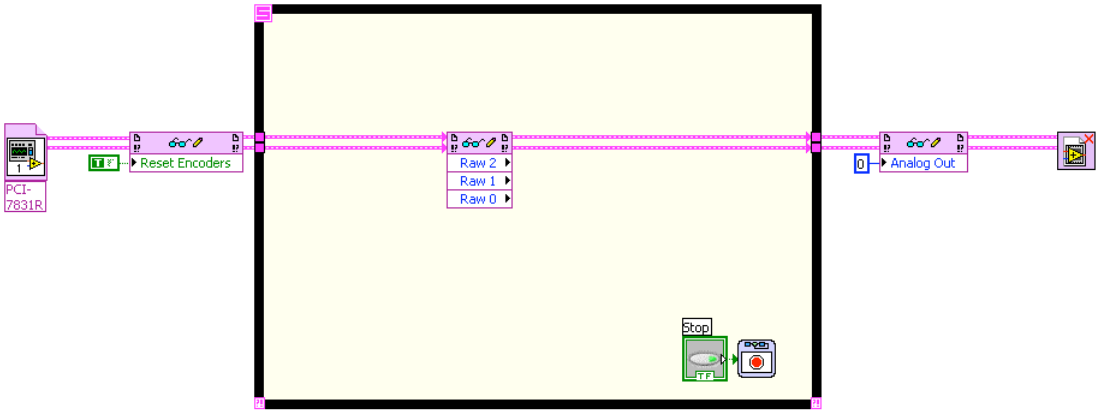


Fig. 2. One of the templates the students used for communicating with the hardware. It provided the signals going to the DC motor and coming from the encoders, as well as a halt feature to gracefully reset the FPGA if needed. This was all the students were provided with–everything else needed for the laboratory they created themselves.

output. Which is "more" stable using a PID or Lead/Lag controller? Why? You may approach this problem in whatever way you find most convenient–just detail it in your writeup.

(5) Plot the root locus for the parameter you think introduces the largest amount of uncertainty. You may wish to "lump" some uncertainties together. Does a parameter variation necessarily give a valid root locus? [3]

(6) We know that using a PID controller with the top disk as the output requires a very low gain on the controller. Design a higher performance controller that allows you to push the overall system gain up higher while not destabilizing.

The labs generally became shorter and shorter as the term progressed and the students required less and less guidance in the labs. This, it seems, is a good indication that the students were becoming more competent. By the time the end of term drew near, and it was time to do the final project, it was possible to give the final project using the following four bullets:

(1) make the bottom disk go from 0 to $\pi$ and stabilize back to 0 as quickly as possible,
(2) accomplish the same task for the top disk,
(3) do so without breaking the ECP unit,
(4) and do so knowing that the instructor will move the weights up to three centimeters away from their nominal position on the ECP machine. [4]

## 3. CLASSROOM SUCCESSES

There were several different elements of this endeavor that we consider substantial successes. These include the increasing quality of computer code the students were writing, the improved understanding of block diagram representations (that was a clear consequence of having to create block diagrams repeatedly while programming in the graphical environment), and the creativity shown in the final projects. By the end of the term, the students were treating control problems as *actual* problems to be solved, and were using an appropriate balance of physical intuition and analysis skills from the control analysis lecture course.

Often, because of the lack of structure in the teaching of this laboratory, students would come up with controller designs that worked in principle but not in practice. This, too, is a valuable aspect of the open-ended nature of the way the laboratory was designed.

### *3.1 Final Project*

This section describes anecdotal evidence, based on this one class, that the "programming from the ground up" teaching philosophy does indeed have positive consequences. As noted above, the final project was very simple. Students were to make the bottom disk go from 0 to $\pi$ and stabilize back to 0 (within 2 degrees or 0.035 radians) as quickly as possible and then were to do a separate design with the same goal for the top disk. The students then competed on the final day of class for who had the best performance for the bottom disk and who had the best performance for the top disk. In addition, the students knew that on the day of the competition the instructor would move the weights to introduce some plant uncertainty (although not of a particularly malicious type).

Using the bottom disk as the output, of course, did not present much difficulty. All the students knew that they should go with some version of a PID or Lead/Lag controller. However, using the top

[3] The answer to this is "no" because the root locus may have higher order dependencies, depending on how the parameter enters the transfer function.
[4] Hence, introducing uncertainty into the system.

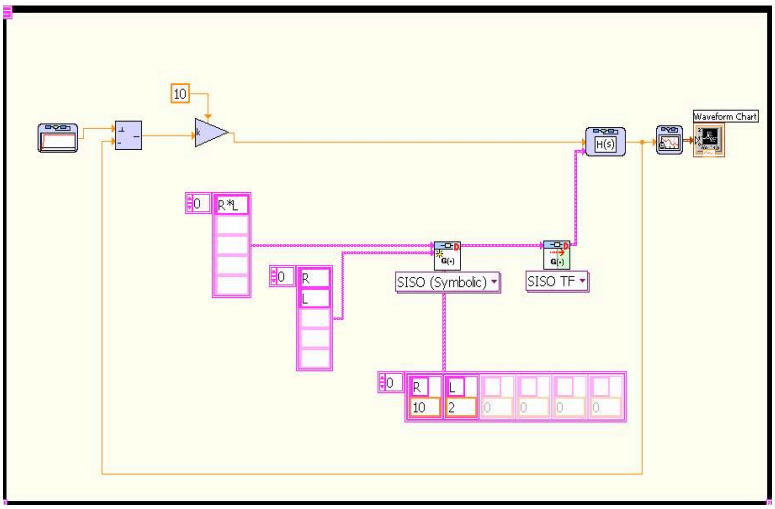


Fig. 3. Sample code given to the students to help them in creating state space models in LabVIEW.

disk as the output generated surprisingly creative approaches. This is because the top disk is the "noncollocated" problem (where the actuation is not located at the same place as the sensing), which is substantially more challenging than the collocated problem. Moreover, we believe that some of the approaches students used would have been impossible to expect in a class that gave the students "canned" software for control design.

A few notable designs (using the top disk as the output) by students:

(1) The noncollocated problem has a transfer function with six poles and no zeros. One student tried a controller that had five poles and no zeros. This controller worked in simulation (with terrific performance and stability) but did not work on the experiment. He discovered that the controller was requiring $10^4$ $N \cdot m$ while the DC motor we had could only provide 2 $N \cdot m$. Fortunately, he introduced a saturation into his simulation after this.

(2) Some groups decided that despite the fact that *we had not covered state-space control in the laboratory*, they would use a state-space design for the final project. They learned LabVIEW's tools for doing so and used both an LQR and pole placement technique for their designs.

(3) Another group also chose to use a state-space design. However, they noticed that in fact the ECP unit has all three outputs available. This, they reasoned (correctly), implied that they could get a much better estimate of the system using all three outputs rather than getting an estimate only using one output and using a sixth order estimator. Their performance ended up being nearly twice as fast as any other group. This is an excellent example of the groups choosing a perfectly valid control strategy that creatively applies what they have learned in class and in lab, but nevertheless goes beyond the scope of what they have explicitly learned. Moreover, this strategy would have impossible to implement, much less conceive of, in a more structured lab where all the code was provided a priori.

(4) The second best performance came from a group that realized that the top disk was passively stable with respect to the bottom disk. Hence, they chose to use a PID controller on the bottom disk and allowed the natural dynamics of the system to stabilize the top disk. Hence, they were using a form of passivity control (certainly without ever even having heard the phrase). There is no question that this solution was somewhat outside the *intended* approach, but this does not mean that it does not reflect real insight and learning on the part of the students.

## 4. WHAT IS LOST IN THIS APPROACH

As with all choices in teaching, some things were lost because of having the students write their own code in the graphical environment. One of the most significant of these losses is that the students never had to concern themselves with digital control–LabVIEW took care of all the translation of continuous time design into digital control on the FPGA.

Moreover, the students did not learn anything about the embedded systems (in particular, the FPGA) they were using. This is because LabVIEW automatically compiled all the code they were writing to the FPGA. Whether or not this is actually a negative or a positive aspect of the class is up for debate, particularly since the course is not aimed at embedded systems. Such a course would, in all likelihood, not allow students to actually implement much more than a PID controller by the end of a one term course.

Another downside, which we intend to address in Autumn term, 2006, is the fact that there was only one experiment. The motivation for this was that the students had to derive all the equations of motion themselves, and this was reasonably time-consuming. However, there were several times during the course when the laboratory was substantially ahead of the lecture course. Therefore, what is needed are substantially simpler (e.g., lower-order) systems that are reasonably easy to model but that have different properties from the ECP unit. In particular, we would like to use a plant that is unstable, such as an inverted pendulum.

## 5. PROS/CONS OF TEACHING THIS WAY

This method of teaching is easier for the instructor prior to a course beginning and harder once the course has started. The instructor *must* be competent at programming in the chosen graphical programming language. This is because the students are actually learning not just how to program in the graphical environment, but they are simultaneously improving their more generic programming skills.

Another aspect of a course like this is that students get to see the instructor solving problems in class. Although in principle this can undermine the instructor's authority, the first author's experience teaching this class is that the students generally appreciated the spontaneous nature of the interaction, even when it occasionally produced enigmatic results that were not immediately resolvable.

## 6. STUDENT REACTIONS

The students reacted positively to nearly all portions of the course. Our university administers mandatory course evaluations at the end of every course, with the most important metrics being "Course Rating" and "Instructor Rating." The students gave the course a B+ and the instructor an A, both of which are high for a laboratory (particularly given that it was the first time this laboratory was offered in its current form). Students specifically mentioned the fact that programming was a valuable experience (partially because many of them were asked in job interviews whether they were familiar with graphical programming techniques). Moreover, the majority of students felt that they had learned control techniques that they would feel comfortable applying in realistic settings. (One student (out of 22) complained that the labs were not "industrial" enough.) Roughly two-thirds of the students mentioned that the open-ended labs were more interesting and fun than previous lab courses they had taken.

The only part of the course that the students did not like was the rotating, assigned groups. This is because of the fact that they had to incorporate legacy code into their projects at the beginning of each new rotation. Although they viewed this negatively, the authors feel that the uniform improvement in the quality of code over the course of the term is an indication that this was a useful part of the course. Therefore, despite the fact that students disliked the practice, we will continue to have rotating, assigned groups in future versions of the course.

## 7. CONCLUSION

This paper describes an approach to teaching a hardware-based controls laboratory that allows students to be in charge of all the programming required for the course. This was made possible through the choice of a graphical programming environment, specifically LabVIEW 7.1, though any graphical programming language would accomplish the same goal. We found that students were more involved in the course and offered more intrinsically creative solutions to problems than they would in a course where the software is pre-written for them. Additionally, the laboratory used open-ended labs, which was facilitated by the fact that the students wrote the software themselves. We are going to revise the course again to involve more experiments, to cover state-space methods, and to make the assignments more open-ended. Moreover, we are going to engage in more formal assessment of the course, partially supported by the U.S. National Science Foundation.

## REFERENCES


Benyo, Imre, Gyorgy Lipovszki and Jeno Kovacs (2003). Advanced control simulation tools in LabVIEW environment. In: *Proc. 6th IFAC Symposium on Advances in Control Education*.

Bishop, Robert (2003). *Learning with LabVIEW 7 Express*. Pearson Prentice Hall.

Bishop, Robert and Nick Lin (2005). Control lab setup: Using LabVIEW to control ECP plants. Technical report. Connexions.

Falcon, Jeanne Sullivan (2006). Graphical programming for field programmable gate arrays. In: *Proc. American Control Conference*. Tutorial Session.

Haugen, Finn (2005*a*). *Introduction to LabVIEW Control Design Toolkit*. TechTeach. www.techteach.no/publications/labview_control_design/.

Haugen, Finn (2005*b*). *Introduction to LabVIEW Simulation Module*. TechTeach. www.techteach.no/publications/labview_simulation/.

Limroth, John, Jeanne Sullivan Falcon, Dafna Leonard and Jenifer Loy (2005). Labview real-time for networked/embedded control.. In: *Handbook of Networked and Embedded Control Systems*. pp. 447–470. Birkhauser.