Coding with Applications to Data Acquisition and Graphical Interfaces in a Freshman Design Course

Timothy Johnson
Engineering Lab Assistant
Geneva College
Beaver Falls, Pennsylvania 15010
Email: timothy.johnson@geneva.edu

Michael Parkinson
Engineering Lab Assistant
Geneva College
Beaver Falls, Pennsylvania 15010
Email: michael.parkinson@geneva.edu

Course Background

EGR 101 is an Introduction to Engineering course at Geneva College for first-year engineering students. It consists of 1 hour of lecture and 3 hours of lab work each week and is intended to teach freshmen the skills they need to work well on an engineering project alongside other team members. The lecture portion of the course teaches the engineering design process proposed by the late Professor Gerard Voland of Indiana University using his book *Engineering by Design*. In the lab portion of the course, groups of 3-5 students spend 15 weeks of the semester designing and building an electromechanical project. The project helps the students practice teamwork within a multidisciplinary group as well as apply the design principles they learn in lecture. At the end of the semester, each team demonstrates the functionality of their project and presents how the design process guided their decisions.

The engineering design process advocated by Voland is an iterative one that begins with needs assessment to establish why a solution is necessary and problem formulation to define fundamental goals that the solution must achieve. Next comes abstraction and synthesis; abstraction is the formulation of ideas to solve subproblems, and synthesis is the combination of these ideas into various complete solutions. In the analysis stage, all of the potential solutions are compared and the best design is chosen. Finally, implementation is the process of developing and distributing the chosen solution, at which point the cycle may start over again to reassess the goals and improve the design.[1] The primary desired educational outcome of this course is for the engineering students to have a strong understanding of and practice with this design process.

To integrate its electrical and mechanical components, the semester design project required LabVIEW, with which most students had no prior experience and needed instruction. LabVIEW is a virtual programming language designed by National Instrument. LabVIEW programs can be created without writing any code, since all the necessary logic can be constructed by dragging virtual components into a workspace and connecting them with wires. The only LabVIEW instruction was a 3-hour long crash course that covered logic, graphics, and electrical inputs and outputs. After this, students were expected to figure out how to program the project on their own without any resources to help them apart from being able to ask teachers questions. To provide programming assistance, the Engineering Department at Geneva College hired two lab assistants in 2016.

Identification of Problems with Course

In Spring of 2017, the Engineering Department formulated a goal to make the Introduction to Engineering course accessible to other institutions. This involved a transition to new technology that was more universal and affordable. In addition to LabVIEW, the class was using the USB-1208LS data acquisition board developed by Measurement Computing so that students could connect electronic components to a computer through a USB cable. LabVIEW licenses for schools and data acquisition devices such as the USB-1208LS are expensive and not viable for smaller schools. Additionally, these technologies are not commonly used in educational settings, especially small or versatile ones, so it is unlikely that teachers would have previous knowledge to support the students. Basic documentation does exist for LabVIEW and the USB-1208LS, but examples are either limited or too large of a scope for an introductory course. The course would also need to be maintainable, which involves documentation of the requirements of the course itself as well as electrical protection for physical devices. To further investigate and meet the needs of the Engineering Department and the students, the lab assistants began to use the Voland design process to evaluate the course's educational methods.

An optional survey of students who had taken the course any time in the previous four semesters was performed to identify their opinions of the current system and suggestions for a potential transition. Figure 1 shows a summary of questions and consensuses from the survey, which received responses from a sample size of 18 out of a population of 206 (9%). The students with previous programming experience rated those other programs, on average, 63% higher than LabVIEW. From the question responses and general comments in the survey, it was determined that if more time could be set aside to teaching programming, the use of a different language with good graphics support would provide better means to complete the design project.

Figure 1: Survey Results (n=18)

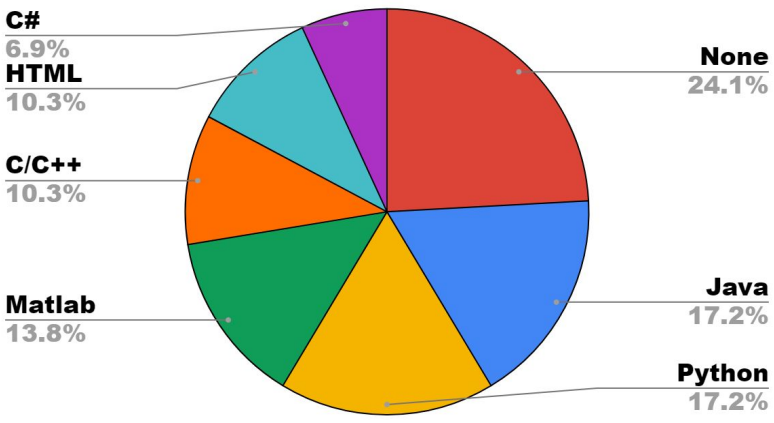
	LabVIEW	Other Languages
Average rating	4.0 / 10	6.5 / 10
What do you like?	Graphical interface is easy to use	Balanced versatility and simplicity
What do you dislike?	Oversimplifies programming	More time required to learn

With the needs of the Engineering Department and the students assessed, a more generalized set of problems was identified. This is a first-year course for all concentrations of engineering, and as such it should require minimal previous knowledge, avoid excessive busywork, and focus on concepts and skills useful to all engineering students. It should also provide enough foundation for successful completion of the semester design project and the students' future careers. The principle of an electromechanical device for the semester design project provides the best balance of various concentrations (e.g. mechanical, civil, computer, electrical) while being basic enough for all required knowledge and skills to be taught in parallel with development. However, there were some incompatibilities between these goals and the current system for completing that design project. LabVIEW is significantly different from most programming languages, so students who already knew how to program before taking the course or learned programming in more depth after taking it were not easily able to apply their previous knowledge. In addition, installation of LabVIEW and its necessary accessories on their personal computers was a long and tedious process. Finally, the time allotted for LabVIEW instruction and the minimal amount of resources provided were inadequate to give the students the foundation they needed.

Selection and Implementation of New Data Acquisition Technology

One of the most important decisions to be made was what data acquisition device and programming language to use. The options for a data acquisition device were quickly narrowed down to Arduino or Raspberry Pi because of their extensive documentation and their focus on education. Both devices are good candidates that have been considered for freshman engineering courses by other colleges as well.[2-4] The Raspberry Pi is a full microcomputer with compatibility for multiple programming languages, and high compatibility with

Figure 2: Programming Language Popularity (n=18)



specifically Python. Using an Arduino would require using the C programming language, and would be a simpler but also more limited system. A question in the student survey asked what programming languages the students had previous experience using; the results are summarized in Figure 2, limited to responses with more than one result. Notable results include Python and Java tied for most common, and C/C++ with somewhat lower popularity.

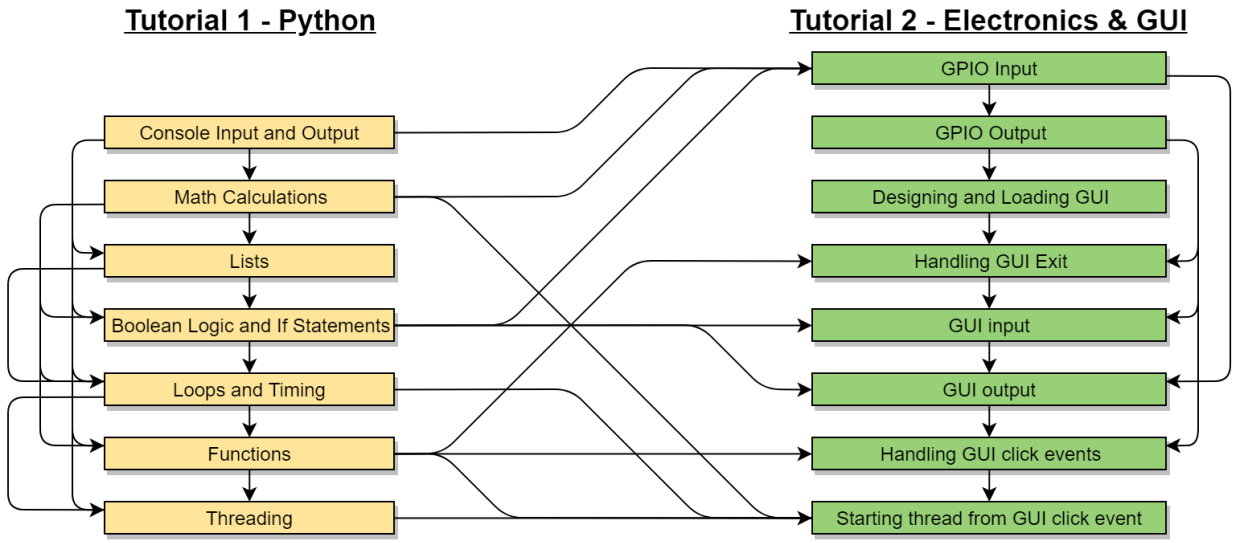
The combination of Raspberry Pi 3 and Python 3 was chosen for many reasons. The Raspberry Pi 3 is a good platform because it includes 40 GPIO (General Purpose Input/Output) pins for digital data acquisition and control, supports a graphical interface (an ability deemed important by the students), stores code and development tools on a microSD card, and has Bluetooth and Wi-Fi built-in. The Python programming language is simple and easy to learn, powerful, similar to other languages, and popular both within the surveyed student population as well as the workforce.

With a combination of data acquisition device and programming languages chosen, the next step was to provide all of the functionalities that the old system provided. This included electrical considerations such as analog voltage measurement, high-power reversible motor control, and easy wire connections, which were solved with an MCP3002 Analog to Digital Converter, L293D motor control chip, and RasPiO Pro Hat, respectively. The RasPiO Pro Hat also provides new functionalities such as electrical protection, a small breadboard, and labeled pins. Another important consideration included peripherals for the Raspberry Pi; rather than requiring a mouse, keyboard, and monitor, the small application RealVNC (and Bonjour on Windows) can be used on any laptop to control the Raspberry Pi over Ethernet or Wi-Fi. The graphical interface framework was chosen to be Qt, which supports diverse and relatively simple integration with Python. Programming and GUI development software was chosen to be Thonny and Qt Designer respectively, both of which are easy for beginners to use. A proof of concept using these components was successfully designed and built by the lab assistants in parallel with the students' design project that semester, including all of the same functionalities.

Creation of Tutorials for Raspberry Pi and Python

Two custom tutorials were written to cover all basic concepts required for completion of the project; the first covers basic Python logic and the second covers Raspberry Pi usage, GUI creation and integration with Python, and electronics integration with Python. The tutorial handouts were designed to be self-explanatory so that students could refer back to them independently. Two full class periods are used to teach these tutorials, in contrast to the insufficient one period for LabVIEW. The connection within and between the tutorials is illustrated in Figure 3; frequent references to early concepts further strengthens them.

Figure 3: Tutorial Interconnection Flowchart



The first tutorial uses a web browser for Python development because the Raspberry Pi is not introduced until the next class period. It consists of basic concepts and examples (console input and output, math calculations, lists, Boolean logic and decision making, looping, timing, functions, and threading) as well as a larger example which incorporates all concepts into a single program as they are introduced. This allows the students to get a feel for how a larger scale script works, which they never got from the LabVIEW tutorial. At the end of the first programming period, all students have a fully functioning program which they have written on their own and can reference later.

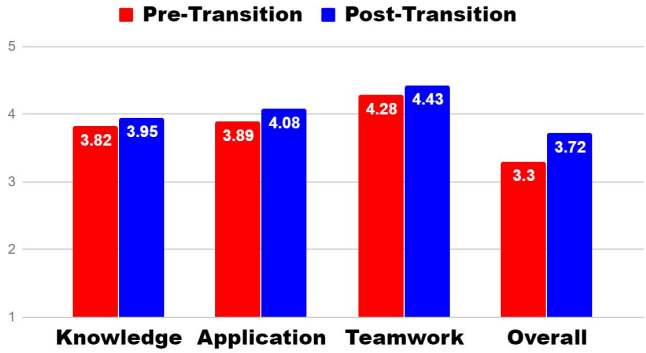
The second tutorial instructs how to connect to the Raspberry Pi and use the Raspbian operating system. It then details GUI creation with buttons, data inputs, labels, indicators, and events, and shows how to control and read electrical signals in Python. The first version of this tutorial applied these concepts to a continuation of the full example program from the previous Python tutorial.

Evaluation of Transition to Raspberry Pi and Python

After the transition to Raspberry Pi and Python development, significant improvement has been noticed. Nearly all students are able to understand the programming aspect of the design project and communicate with their teammates about it. They have also shown a greater interest for the subjects of electronics and programming inside and outside of class, as evidenced by increased attendance and participation in the Computer and Electrical Engineering club on campus. Analysis was performed on IDEA evaluations of 4 pre-transition labs (60 students) and 4 post-transition labs (66 students), containing an equal distribution of two teachers. A selection of

pertinent questions (1-5 scale) were chosen including progress on "gaining factual knowledge," "learning to apply course material," and "acquiring skills in working with others as a member of a team," as well as agreement with the statement "Overall, I rate this course as excellent," and the average results before and after the transition were compared (Figure 4). All of these questions received higher ratings after the transition, and the overall course rating closed the gap to a maximum 5.0 by 25%.

Figure 4: IDEA Evaluations Comparison (n=126)



The needs of the students were reassessed and the strengths and weaknesses of the new system were observed. Even with an extra day dedicated to programming, the class period usually ended before the final sections of the Python tutorial could be reached. It was not possible to cover all the material in the tutorials adequately, and it was concluded that some of their content was too advanced and not suitable for students who are new to programming. The second tutorial was too compartmentalized, which lengthened instruction time and made it more difficult for the students to understand how all the sections relate to each other. Finally, the list of project requirements given to students was not sufficiently clear, as some crucial information was missing and some parts present were ambiguous.

Improvements to Requirements, Tutorials, and Given Code

The most significant part of the tutorials that was considered for simplification was the process of creating and integrating a graphical user interface (GUI). In the first revision of the Raspberry Pi tutorial, students were provided with advanced Python code to run a simple Qt app around which they needed to write their own code. This code required an understanding of object-oriented programming and inhibited their understanding of the program as a whole. Switching to a different GUI framework was considered, but other toolkits available for Python seemed less user friendly because they do not contain a GUI builder as easy to use as the Qt Designer. Another option was providing students with a premade GUI, but still requiring students to construct their own seemed desirable since the graphical aspect of LabVIEW was one of their favorite things about it. The solution chosen was to create a lightweight Python module that makes GUI creation easier by handling the obscure parts of the code behind the scenes, while still granting students the freedom to creatively design their own interface. It is now possible to launch a GUI created in Qt Designer from a Python program with 3 lines of code, which is much fewer than before and uses only concepts that the students can readily understand.

Other methods for addressing the lack of sufficient time to teach the tutorials were also considered. Adding more weeks of programming lectures could be beneficial since fitting all the required programming knowledge for a semester long project into two 3-hour sessions is difficult, but this would limit the time for project development. Simplifying the goal of the design project would reduce how much knowledge is required to accomplish it but remove the benefits of a challenge. It was determined that after simplifying GUI implementation, the tutorials could be shortened further by refactoring their material and removing a number of advanced concepts. These changes are anticipated to encourage teamwork because each of the students in a lab group can contribute to the coding when they do not feel overwhelmed by it.

Many difficulties that students had while working on their semester projects were traced back to the second tutorial being unclear or omitting important concepts. GUI and electronics examples were isolated from each other in the tutorial which prevented a clear understanding of how they relate to each other. Additionally, the GUI example was repetitive and had an incorrect focus, and the electronics examples were impractical and unengaging. This was fixed by creating an entirely new example program which combines both concepts into a single example and grows consistently with frequent stepping stones. Similar to the first tutorial, this more completely shows how each concept relates to the others, and the example chosen also is subjectively much more fun and engaging. Additionally, the properties and usage of GUI widgets were unclear, so a detailed table describing their functionality was appended to the end of the tutorial.

Making it easier for students to understand what is expected of them was also recognized as an important problem to address. The list of project requirements was made more thorough and less ambiguous, and a strategy for defining the program structure was chosen. Since many students are new to programming, it is difficult for them to plan the program structure needed to implement the project. Students could work more independently and be less reliant on help from lab assistants if the code for the project was provided in parts that they could assemble to produce a working program. However, this would not grant them the significant learning experience of writing most of the code on their own and understanding it. Instead it was decided to only provide students with a brief skeleton of the code required for the design project to help them get started. The skeleton consists of function signatures to show the inputs and outputs used by each section of the program, and comments that concisely explain the purpose of each function. This still requires that students write most of the code themselves, but they should find the skeleton very helpful because it shows the functionality they need to implement. If students have prior programming experience, they are free to structure their code differently from the provided skeleton.

Conclusions

The tutorials that were created have been made available online (https://github.com/geneva-egr101/coding-resources) with the hope that they will be useful for anyone who wants to teach basics of Python programming and its application to data acquisition and graphical interfaces on the Raspberry Pi. They are available on GitHub in both DOCX and PDF format, and files containing code from each step of the tutorials are also included.

Giving students the tools and resources they need is the first part to a successful learning experience, which was accomplished in the first stage of this transition. Providing them with structure to allow them to flourish with those tools and resources is just as important, and the plans to improve that structure for this course are nearing completion. Yet even when that is completed, problems will still exist with potential to be minimized or eliminated. Balancing difficulty with accessibility, instruction with application, and structure with creativity is a never-ending cycle. According to Voland, "Engineering design does not end with an optimal solution. There is no such thing as a 'perfect' solution to an engineering problem because of the compromises that one usually must make in order to resolve conflicts among the design goals."[1] These ideas are not limited to first-year laboratory courses for designing electromechanical devices; they can, and should, be applied to almost any course.

Bibliography

1. Voland, Gerald. *Engineering by Design* (2nd ed.). Upper Saddle River, NJ: Pearson, 2004.
2. Steinmeyer, Joseph Daly. "Project-Based Learning with Single-Board Computer." Paper presented at 2015 ASEE Conference & Exposition, Seattle, WA.
3. Frederickson, Carl. "Introducing Code in Freshman Physics Labs using Arduinos." Paper presented at 2017 ASEE Conference & Exposition, Columbus, OH.
4. Husseini, Naji S and Ian Kaszubski. "Incorporating the Raspberry Pi into lab experiments in an introductory MATLAB course." Paper presented at 2017 ASEE Conference & Exposition, Columbus, OH.