MEEM 4990/5990: Getting Data in the Lab

Final Project (Draft)

Due: to be determined but pretty much at the end of the semester.

Summary

Design, implement, and test software that drives the experiment or process studied in your interim project. (Note, deviations for your interim hardware design are acceptable, so long as the software and system still meet the final project requirements.) Please note that while the requirements list is long and initially daunting, it can be summed up as "write a clean, easy to use LabVIEW program to acquire data from your system, display it, and save it to a file." Many of the requirements would be difficult to *not* do.

Preparatory Work (this will be due in advance of the final due date)

- ☐ **Requirements List:** Develop and document a list of software requirements for your project.
  - The main goal here is to think through how the user should interact with the software and how the software should interact with the hardware to perform the experiment or process.
  - These requirements should list the channels to be read, any analysis performed, the data to be displayed, the user interactions to support, and how the data will be saved.
  - Given the diverse nature of the projects, it is impossible for me to write a list of software requirements that are applicable to every individual project. However, the minimum requirements in "Software Requirements" must be incorporated into your requirements (feel free to copy word-for-word, if you like).
  - Relate the software requirements to your system and hardware requirements, as described in your Interim Project.
  - If any of the following requirements are unreasonable or inapplicable to your project, please contact me and we will develop an alternative plan.
- ☐ **Flow/State Chart:** Develop a flow chart or state chart to that describes your program execution
  - To reduce the time-burden of this step, hand-drawn figures are acceptable, but please be sure that your text is legible.

System Hardware Requirements (copied from Interim Project)

- ☐ The system to be tested (measured, studied, etc.) must utilize at least three channels total of input and output.
- ☐ At least two of the channels must be of different major types (analog input, analog output, digital I/O, counter-timers, instrument control).
- ☐ At least one channel must utilize hardware-timed acquisition

Software Requirements

- ☐ **GUI:** The software shall present a graphical user interface
    - The interface shall allow for configuration of the experiment, e.g. selection of channels; sensor types; processing algorithms; data acquisition speed, duration, or number of samples; or anything else that makes sense for your project. Consider carefully each parameter and whether or not it should be user-configurable or hard-coded (see also Code Documentation requirements below).
    - Additionally, the user-interface must provide for information about the experiment/process to be stored with the data files (see Output, below).
    - The user interface shall be responsive to changes in configuration and execution state: i.e. the software shall have an event loop.
    - The interface shall display the acquired data to the user in a prompt and meaningful way. "Prompt and meaningful" will allow the user to determine if the experiment is running as expected as soon as possible.
    - The interface shall allow the user to select data file output
    - The interface shall be clean and easy to understand and use (well-aligned, non-overlapping controls and indicators; control and indicators grouped by functional similarity; labels or other front-panel documentation as appropriate, etc.)
- ☐ **DAQ:** The software shall acquire data from (or generate signals on) each of the channels.
    - Show appropriate use of the Mantra: Open, Configure, Start, Read/Write, Stop, Close
    - Handle hardware-timed data (either continuous or single-shot "Multiple samples") appropriately using either the waveform data type or an array.
    - The acquired/generated data must be appropriately synchronized i.e. he program must "know" the time each sample was acquired/generated. Time may be relative (e.g. to the start of the acquisition) or absolute (e.g. May 14, at 2:05:12.012 PM) as appropriate.
- ☐ **Analysis:** The software shall perform analysis on the input data
    - Minimal analysis will include converting the data from voltage to physical units. The use of DAQmx scales, either directly or via NI-MAX channels is permissible, but must be documented in the code.
    - More complex analysis such as performing math on multiple channels, digital filtering, averaging, etc., is encouraged, as necessary or helpful to the user, but is not required.
- ☐ **Output:** The software shall save the acquired data in an appropriate format. (Alternatively, if generation rather than acquisition is the primary goal of the software, the software shall read data.)
    - The data format must either be a self-documenting format such as HDF5 or TDMS
    - **OR**, the data must be in a delimited text file which stores attributes and column names in an appropriate header.
    - At a minimum attributes for the date and time of the acquisition and a brief description of the data must be stored in the file. The user must be able to populate the description field from the user interface.
    - The user must be able to choose the location of the file output (or input).

- ☐ **LabVIEW Coding Practices:**
  - SubVIs shall be used collect repeated or similar functionality and modularize code, as necessary or prudent. (Note that it is *possible* that your project will not require subVIs beyond those provided by NI. I am looking for readable, modular code, not subVIs for the sake of subVIs.)
  - Diagram shall be clean and easy to understand: few wire bends, no wires or nodes hidden underneath other objects, sequential function groups (like The Mantra functions and file functions) kept in a single row when possible.
  - Appropriate error handling shall be used. All subVI error outputs shall be handled unless sufficient justification is provided in code documentation.
- ☐ **Code Documentation**
  - Use labels to explain any non-trivial hard-coded values (constants). (E.g., justify a hard-coded sample rate, but you don't need to justify an array index constant unless you deem it difficult to understand why that constant was chosen).
  - Use block diagram labels on any portions of the diagram you believe are difficult to understand without such comments.
  - Document all subVIs using the VI Description (VI Properties->Documentation)
- ☐ **User Documentation**
  - Document all top-level VI (those VIs with which the user interacts) controls and indicators via Tip Strips and/or Description (Right-click->Description and Tip)
  - Write a brief user manual for your software. The manual shall walk a user through the typical software use case. Use screen-shots (search for "Snipping Tool" in Windows) to aid the user.

Do it

- ☐ Implement your software according to your requirements
- ☐ Assemble your hardware system at least to a point sufficient to test and demonstrate your software
- ☐ Demonstrate that your software meets your requirements by running it on your assembled hardware system or a reasonable facsimile thereof
- ☐ Deliver your hardware, software and documentation to your adviser/project sponsor

Suggestions

- Start now
- Start your VI or copy heavily from already built example file. Use the Example Finder (Help-> Find Examples…). If you browse by task, particularly useful examples may be found in Hardware Input and Output -> DAQmx -> Analog Input.
- Use the Template "User Interface Event Handler"

Warnings

- As should be obvious, you may not know everything necessary for your project when this assignment is first handed out. Start anyway. I expect that most knowledge gaps will be filled by lecture but some may require independent research.
- Also, anticipate that your system will have problems outside the scope of the assignment (equipment failure, transducers that do not work properly, etc.). Given the limited time during the semester, try not to spend too much time making the *system* work if possible. Instead, find ways around the misbehaviors that enable you to still develop and test the software.

Grading

To be determined.