ECE 490: Deep Dive

Daylen Mackey

1 Background

Milky Way Solutions has been tasked with developing the first prototype of the X-Calibrator: A device capable of calibrating Dose Calibrators in clinics across the world, while being sturdy enough to withstand the rough contact that comes with international shipping. Dose Calibrators are large, stationary instruments used to measure the concentration of a radioactive sample [1]. This prototype needs a touchscreen for user interaction, a single mains power, and a tough exterior casing. Samples will be put in the portable ion chamber, and small resulting currents are measured using the Dose View Electrometer (DV). A Raspberry Pi will be responsible for coordinating all actions, gathering data from the Temperature/Pressure probe, and outputting the relevant data.

Due to the complexity of our project, I have broken down my area of contribution as follows:

1) **Interface the Raspberry Pi with the touchscreen**
2) **Create a User Interface for the touchscreen**
3) **Build out a Back End on the Raspberry Pi that:**
   a) Allows touchscreen inputs to send out commands to the DV
   b) Calculates Current, Temperature/Pressure correction factors, calculates the amount of a given radioactive sample
   c) Stores measurements and calculations internally with an option to write to an external USB.

All these will be addressed below.

2 Nomenclature:

R-Pi: Raspberry Pi
UI: User Interface
DV: Dose View Electrometer:

3 Deep Dive

3.1 Interface the Raspberry Pi with the touchscreen:

This is the least challenging of the tasks. To validate initial ideas, we found other students with R-Pi compatible touchscreens, so we could begin testing. These smaller and cheaper screens (3.2 inch and 4inch Waveshare) required installation of a special driver on the pi, HDMI connections, and 8 pin connections [3] . After discussing parts with our client, we've decided on the 7 inch WVGA Multitouch [4].

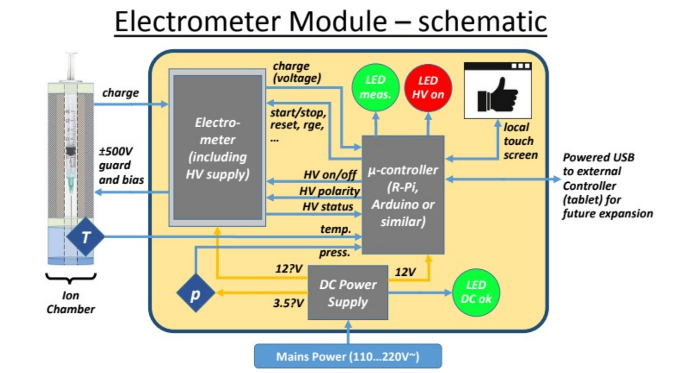
Electrometer Module – schematic
charge
±500V guard and bias
Ion Chamber
T
p
Electro-meter (including HV supply)
charge (voltage)
start/stop, reset, rge, ...
HV on/off
HV polarity
HV status
temp.
press.
LED meas.
LED HV on
local touch screen
µ-controller (R-Pi, Arduino or similar)
Powered USB to external Controller (tablet) for future expansion
12?V
3.5?V
12V
DC Power Supply
LED DC ok
Mains Power (110...220V~)

Fig. 1. High Level Schematic of the X-Calibrator [2]

This touchscreen is larger, has a greater resolution of 800 x 480 and should be faster (doesn't need another driver). Another reason for selecting this screen is the large number of open source projects using the screen. [5], [6] . While this isn't exactly a proof of concept, it gives me much more confidence that we can successfully interface the screen as others have already done so. It could also provide us with guidance should we encounter any issues.

3.2 Create a User Interface for the touchscreen

Much discussion went into the design of the user interface. Our client provided us with a basic layout of what he wanted, but the complexity came in choosing the frameworks to use. Because I was confident I would be able to build a simple UI quickly, I began prototyping with HTML, CSS, and JavaScript. As of writing this, the GUI is approximately 75% completed. All that is left to add is a " popup" number pad so that users can enter values using the touchscreen (you currently need a keyboard). The GUI does not look very good right now (it is simplistic and static), but I built it for 2 reasons:

1) It did not require any special hardware to build, so I could begin assembling it during my free time

2) Even if the current version is not the one that's used throughout the project, it gives us something to test once our hardware does arrive.

Further integration of Node JS and Electron JS would allow for a 'sleeker' look, as well as increased functionality and communication with the back end. These two frameworks are commonly used, and could facilitate future expansion efforts.

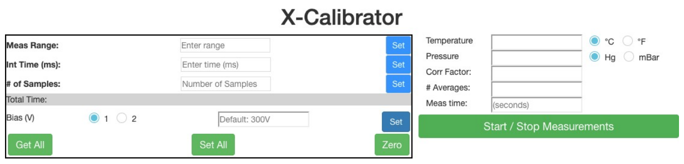

Fig. 2. User Interface Prototype Version 1

Over Christmas break, I intend to build a similar GUI system using Python Tkinter frameworks as well [7] . This may make connecting the front end to the back end systems simpler. If we were on a tighter timeline, I would not go forward with both options. However, since these tasks are not too time-consuming, and we don't have other hardware to work with, I figured I would do both.

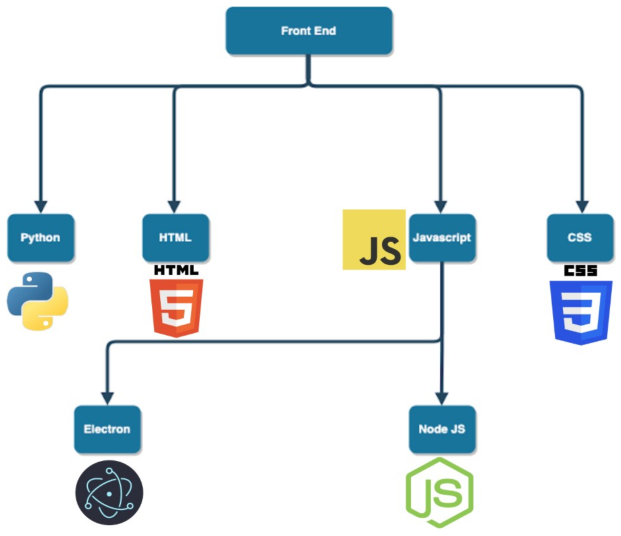

Fig. 3. Overview of frameworks used to build the Front End

3.3 Build out a back end on the Raspberry Pi

3.3.1 *Allowing touchscreen inputs to send out commands to the Dose View Electrometer*

This task will likely be the most challenging. To complete this task, we need to be able to take the user inputs from the front end, interpret them in the back end, and send off the appropriate commands to the DV. This is challenging because the DV's code base is written in MATLAB, and MATLAB can't be run on a R-Pi. We initially wanted to use Python as our primary language for the back end (for reasons of simplicity and familiarity), but because of the large number of frameworks in use, we need to be flexible. We have considered several approaches for this:

1) Convert all the MATLAB code to Python

This consideration is inefficient and unappealing. Our client spent a lot of time developing the existing code base because the DV's communication protocol is so complex. While adjustments will be necessary, we hope to use what he has already given.

2) Use Octave

Octave is a MATLAB-like program that can be run on an R-Pi. It is compatible with most simple MATLAB scripts as long as they don't use some of the newer, more complex MATLAB libraries [8] . The idea behind this approach is that we will use import user inputs into our back end language (Python), then pass those values to our octave program. We have already tested our MATLAB code base with Octave, and the two are compatible. That being said, we haven't tried using Octave directly with the DV yet. We plan to begin testing with Octave on a PC, then try through the R-Pi. The reason for this is that the DV requires a special driver that's currently only available for Windows and Mac. By testing on the PC first, we can identify if Octave is the problem, before implementing the new driver. We have found several resources that successfully modified the driver, so we believe we'll be able to implement this successfully [9] .

Another upside of using Octave, is access to the Pythonic library. [10] This library allows you to call and create Python objects, within Octave. The plan has always been to only use Octave when communicating with the DV, but this opens new possibilities. The majority of the back end could potentially be built in Octave, and we could use Pythonic when needed. While we don't plan on taking this approach, it serves as a promising backup.

3.3.2 *Calculations (Current, Temperature/Pressure correction factors, and concentration)*

This is one of the simpler tasks. Our client has provided us with all the formulas necessary, we just need to implement them in the code. With python libraries like Math, Pandas, and NumPy, implementing these formulas (even with larger scale data sets) will not be challenging.

Because we want the prototype to work for many different radioisotopes, we need to have attributes of each isotope stored on the system. We initially thought this may require the creation of an SQL database, but we should be able to store the data in a Pandas dataframe.

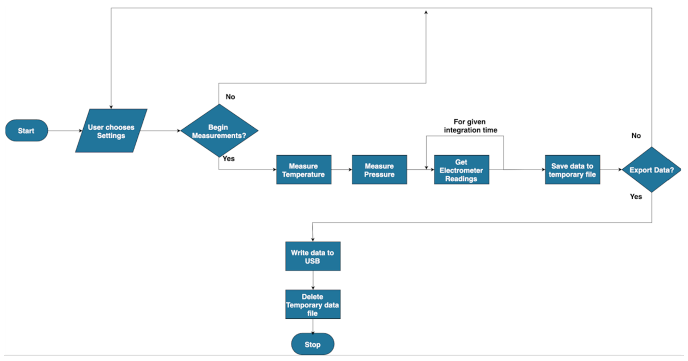

Fig. 4. Flow Chart of Back End Procedures

3.3.3 *Storing data internally with an option to write to an external USB*

This has been a fairly last-minute addition to the project. The initial plan only included saving the information internally, and maybe displaying it as an onscreen popup. Even for a minimum viable product though, this solution is unimpressive. In the last few weeks I came up with the idea of adding USB writing capability. The results will be logged internally initially, but once the user is ready to export all the data, the data will be exported as a csv (commonly used for excel) or txt file on to the USB. Upon exporting, the data should be cleared from internal memory. Writing to an internal file or USB will not require additional frameworks, just the file path of the system.

4 Conclusion

Over the course of this semester, I have learned a great deal about the design process, and how complex things can be become. When I first looked at this project, I believed it would be simple to implement. However, the more I learn about the project, the more challenging it seems. As it currently stands, I believe I will be able to complete my tasks. However, there are still many unknowns. Because we are relying on the DV, we also need to rely on the driver associated with it. Despite these potential setbacks, I am excited to get start building the X-Calibrator, and I am confident that our final product will meet our client's needs.

References

[1] H.-S. Jans, "Activity cross-calibration of unsealed radionuclides utilizing a portable ion chamber," *Medical physics 43*, vol. 12, pp. 6536–6543, 2016.

[2] H.-S. Jans "Documents from Client Meetings."

[3] "3.2inch RPi LCD (B)." [Online]. Available: https://www.waveshare.com/wiki/3.2inch_RPi_LCD_(B)

[4] "Raspberry Pi Touchscreen Layout." [Online]. Available: https://www.raspberrypi.org/documentation/hardware/display/7InchDisplayDrawing-14092015.pdf

[5] "Building a Touchscreen Interface for Raspberry Pi." [Online]. Available: https://humanizing.tech/diy-raspberry-pi-touchscreen-d27165942bb0

[6] "Raspberry Pi Noticeboard using 7" Touchscreen display." [Online]. Available: https://maker.pro/raspberry-pi/projects/raspberry-pi-noticeboard-using-7-touchscreen-display

[7] "TkInter Python Documentation." [Online]. Available: https://wiki.python.org/moin/TkInter

[8] "Octave Website." [Online]. Available: https://www.gnu.org/software/octave/

[9] "RPi Serial Connection." [Online]. Available: https://elinux.org/RPi_Serial_Connection

[10] "Pythonic Documentation." [Online]. Available: https://wiki.octave.org/Pythonic