Matthew Buckman, Ethan Smith, Colton Leopold

Computer Science 370

Professor Shrideep Pallickara

December 6th, 2023

From Fish to Fuse: A Workstation for Breadboard Version Control and Analogue Readings

Introduction

For the term project, our group ended up creating a hobbyist workstation for breadboard development. While not the project we initially set out to create, this workstation successfully allows users to test the voltage of a project, add their own analogue sensors, and take progress pictures all within a user-friendly GUI. At the beginning of this process our team set out to create a fish tank monitoring system. We intended to use a Raspberry Pi and two sensors (pH sensor and camera) to check the pH levels in the water and through image analysis determine the turbidity in the tank. This project was unfortunately scrapped because the pH monitor, we had purchased did not function properly. This was particularly frustrating because not only was the pH monitor faulty, but it had already eaten up a lot of our time as attempting to read its input was particularly challenging. This challenge comes from the fact that the pH monitor provides analog inputs, but the Raspberry Pi 's GPIO Pi ns are designed for digital I/O ("Physical Computing with Python."). This issue was resolved by using a ADC (analogue to digital converter) on a breadboard. In an effort to detect the issue with our pH sensor, on the same breadboard as the converter, we added a voltmeter to ensure we were sending the right amounts of power to the correct locations. While this development helped us discover the fault in our sensor, it also gave us a starting point for a new project idea. Since we found the voltmeter so useful in our own development, we thought this setup may be beneficial to other hobbyists doing similar projects. To include another sensor, we thought it would be a good idea to include the camera as a way to save progress on the user's project. We see this as the tangible GitHub, instead of saving a version history of code this workstation can save a version history of physical changes. The failure of our initial project and some of the tools we used during that development led us to the final project we are presenting: a hobbyist workstation for breadboard development with progress picture, volt measurement, and customizable port capabilities that can be operated with a custom GUI using a Raspberry Pi, voltmeter, and camera.

Problem Characterization

Our project seeks to solve two problems which are descripted in this section:

The first problem is a lack of version control for hardware development. As computer scientists, many of us are familiar with one of the most powerful version control systems available today, GitHub. While GitHub is a powerful tool for version control it is limited to software. Code lends itself to keeping an in-depth history because it is created and stored on computers. When it comes to hardware, its physical nature presents a huge hurdle in maintaining a sufficient history of its changes. Not only is there more room for human error but there are also physical changes that must be reversed/redone; no simple back button or click on the version history. When trying to find systems for hardware version control, almost all the systems available are for the drafting process of a project such as a CAD (computer aided design) version control system for designing printed circuit boards (PCBs) (Peterson). There does not seem to be specific tools for maintaining a history of a physical project outside of general project management guidelines. While our project is constrained by hardware's physical limitations, we can improve hardware version history. In conjunction with the frequent and (generally) easily reversible changes on a breadboard we looked to solve this problem by developing version control for breadboard hobbyists.

Our second problem to solve is to improve the Raspberry Pi 's inability to take in analogue inputs. The difference between analogue and digital input is that digital input is in the form of a binary code while analogue is a continuous signal (A_K_Mishra). Both types of input have their strengths but for measuring specific input values, such as volts or values from a sensor, analogue is preferable since digital can only represent true or false (1 or 0) at any given instance. The Raspberry Pi 's GPIO pins are only designed to take in binary input values which prevents users from collecting a wide range of statistics. The initial instinct we had for solving this problem was to use an Arduino in conjunction with the Pi since our original pH sensor had documentation on how to use it with Arduino; which does read analogue values. To be more economical, though, an Arduino overcompensates and is overpriced for solving this problem, so we decided to look elsewhere. The only element of the Arduino we really needed was the analogue to digital converter (ADC) chip which allows for analogue input on the Raspberry Pi. Considering this, we decided to solve the analogue input problem by using an ADC chip and breadboard circuits for users to have analog input capabilities.

Proposed Solution and Implementation Strategy

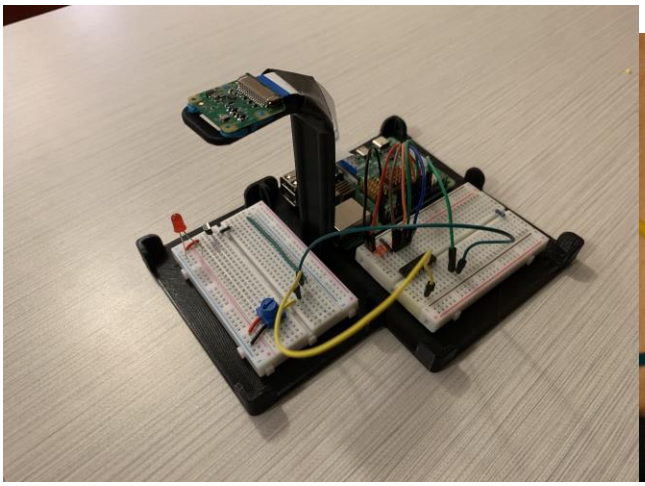
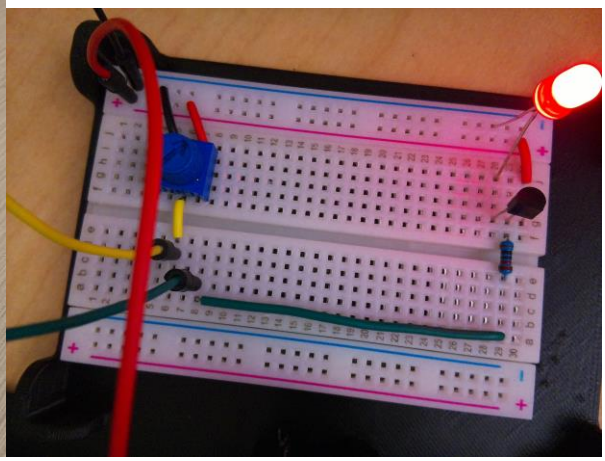
To solve these two problems, we planned to use a Raspberry Pi camera module as well as an Adafruit’s 10-bit, 8-inputs, MPC3008. These two components were chosen mainly for their low-price points and their support and easy integration with the Pi. We planned to add programmability to these components through Python as both components specifically had support for Python; the camera with the libcamera and the OS module and the ADC with its own circuit Python libraries. Connecting the camera module to the Pi was simple as the Pi has its own slot for the camera. One hiccup in the implementation was the number of deprecated applications and out of date documentation online for the camera module, we ended up going down a few implementation routes only to find out that the method we were trying to use was deprecated. There also was some trouble with adding the python programmability to the camera as well. A

few dependencies for the camera python module were not on the Pi, possibly because we reformatted our python environment for the ADC, which we'll cover more later on in this report. Since importing the proper camera modules in our python scripts was not working, we found a simple workaround, which was to use the Python OS module to create a request for the OS to run the libcamera-jpeg application which worked great and made our coding much simpler and more streamlined as we were familiar with the libcamera application. The implementation of the ADC was more complicated since this specific part relies on many of the Adafruit and Circuit python libraries as well as requiring some specific breadboard circuitry to function. The way this ADC works is it sends signals to specific GPIO Pi ns on the Pi based on the voltage values for each of the 8 Pi ns. For the Pi to properly read in the ADC values the 4 outputs of the ADC must specifically be connected to the Pi 's SCLK, MISO, MOSI and D5. The ADC is also powered by the Pi's 3.3-volt output pins. To read the values on the PI the Adafruit Blinka library was used to implement the GPIO input and a specific python library for this ADC was used to convert the raw GPIO values into readable bits. To use these libraries our python environment had to be changed to be setup using Adafruit's Raspberry Pi setup which provides many classes that represent circuit hardware. Within python defining a pin on the ADC is as simple as defining a new AnalogIn() class with the specified pin and ADC settings. Testing is the first implementation of this part was quite simple as well, we just hooked up a simple potentiometer which can control voltage output to one of the circuits and twisted it while a python script printed the values to the terminal. One major limit of this part was that the highest value read in is 3.3 volts but many hardware projects on the PI include 5.5v so extra resistors may need to be used to limit the voltage to a readable value. After verifying that we correctly connected both components to the Pi, we boiled down both components into their basic functionalities and wrote a python module for each component (pins.py and camera.py) to use in our own application. We also wanted to leave these modules open-ended and basic just to supply a basic interface and allow anyone using the pi/workstation to add analogue GPIO programming and camera behavior in their own circuitry. For the basic version control systems, we created a simple graphical and non-graphical application to allow users to read in and display the live voltage values of each pin on the ADC as well as take pictures with the camera. A mount for the Pi, camera and ADC was also created, this allows us to position and focus the camera to take pictures of breadboards. To provide clear example of how someone may use the libraries, we also created a simple breadboard circuit and Python script (example.py) to read in an analog value from a potentiometer, convert it to a digital one that the PI can read, once the Pi reads a value above a specific threshold, the Pi will send a signal to an N-type transistor to turn on a LED and then take a picture of the circuit using the camera. This script uses the time library, our Pins and Camera as well as the LED class from the gpiozero library. This code is heavily commented as well, to make it clear to anyone looking for examples.

To make the workstation more friendly to hobbyists and beginners, we created an application to read in and display the ADC values and control the camera. This application was written in Python and has both a Graphical (GUI.py) and non-Graphical (nonGUI.py) version in case users are using ssh without X-11 forwarding. The non-graphical program takes advantage of Python's threading libraries to read and display the voltage values while also accepting IO from the user. The graphical application is much more complicated and useful as it takes advantage of

Python's Tkinter library (Graphical application description). Being new to GUI development, Tkinter was a good choice as it contains a fair amount of behind-the-scenes logic allowing the programmer to use some simple methods to spawn elements on the Tk interface which resembles a simple webpage. The logic as to how this GUI works is relatively similar for each type of element. In the example of a button, a programmer would call the tk.Button method with the first input being the name of the Tk interface that the button should be displayed on, followed by customizations. This button example: picbutton = tk.Button(root, text="Take Picture", command=picture, font=custom_font) is displayed on the Tk interface named root with the text "Take Picture" in the font custom_font (in our GUI.py this custom_font is a size parameter). The "command" section defines what should happen upon pressing the button, setting it equal to picture calls the picture function. The picture function we defined calls the method takeImage(userInput) from our script Camera.py with input provided in a text box above the button; this allows the user to take and name a picture of their workstation. Using this logic, the programmer can then call picbutton.pack() to define when the element should be finished and set to the Tk interface ("Graphical User Interfaces with TK.").

Implementation Images

(Left: picture of the workstation with ADC breadboard and example breadboard project)
(Right: picture of example breadboard from Raspberry Pi camera while running example.py)

Conclusion

Throughout our project's production, we had to overcome a few challenges. To start, we had to design and implement our project. Our original project idea and goal had to be changed to a voltmeter with camera capabilities for working and saving progress on a breadboard. Originally, the project we had planned was for it to be a small part of an aquaponics system. This project would serve as the start of the aquaponics system where plants and fish would be monitored to provide a healthy ecosystem that would be used to provide sustainable vegetation and fish. The system was originally going to comprise two monitoring components. The first

component was a camera that monitors the entire system and detects visual changes or problems that would come up within the system. The second component was a potential hydrogen (pH) sensor. This would read the pH of the water inside a fish tank and monitor it so that fish could live in a healthy and sustainable environment. Alongside the pH sensor was also the ability to monitor the water's temperature and its dissolved oxygen. Again, this would have helped with promoting a sustainable, safe, and healthy environment for the fish that was originally going to be part of our aquaponics system. The pH sensor however was nonfunctional.

Once we started the project, we found that the pH sensor that was shipped to us did work with our Raspberry Pi but did not function in reading the pH of the water. We did try to calibrate the pH monitor and fix it, however, no matter what we used or tried the pH sensor would not read the pH of the liquids correctly. Well, this was a massive problem for us. Due to the project deadline coming up, we did not have enough time to go ahead and order another one in the hopes that it would work and could be calibrated correctly. Not to mention if we did order another one, we would be going in a bind for our coding for our sensor readings and would have had to hope that our code would have worked first try. This would not have been a good decision on our part. Thus, it was not feasible and from there as a group, we decided to change our project idea and goal. Thus, coming up with a voltmeter and a camera.

Once we voted for our project, Matthew worked on the Raspberry Pi and the breadboard used to measure the voltage being inputted into the functions to do these measurements as well as the functions for the camera to work. Meanwhile, Ethan and Colton worked on creating a graphical user interface (GUI) for the voltmeter and camera. The GUI features a menu that allows the user to set the channel for each channel in the breadboard. This allows the user to measure the voltage within each channel and in each pin. For the GUI, there is also a text box entry that would allow the users to name the file of the picture being taken as well as a button that could be clicked to have the camera take a picture. That way any user using the project could take a picture and save their progress for their breadboard if they needed to disassemble it or if they needed it for other uses. However, the only way we can present this window for the GUI is through the X11 port forwarding that is integrated with MobaXterm.

Despite the changes that were incurred during this project, as well as the changes in our project's goals. We were successful in creating a voltmeter that works with a camera and were able to meet our end goals for this project. Not to mention the use of a GUI makes the voltmeter much simpler and helps a lot in terms of understanding the voltage reading coming from each channel. Alongside that, the use of a camera does help with saving one's progress when implementing a breadboard. If we were to do another project, the first step our group would take would be to check the components in our project and make sure that they are not faulty or broken. This check would have saved a lot of hassle our group had to deal with at the beginning of this project. Not to mention if our group did receive a faulty component then we would have enough time to replace it or to switch to a different part from another manufacturer. Besides checking the components for any issues, there is nothing else that our group would like to change, and the process of the project went very smoothly.

References

"Adafruit Python MCP3008." *GitHub*, 20 Dec. 2018, github.com/adafruit/Adafruit_Python_MCP3008.

A_K_Mishra. "Difference between Digital and Analog System." *GeeksforGeeks*, 29 Mar. 2023, www.geeksforgeeks.org/difference-between-digital-and-analog-system/.

"Graphical User Interfaces with TK." *Python Documentation*, docs.python.org/3/library/tk.html. Accessed 5 Dec. 2023.

Peterson, Zachariah. "Best Practices in Hardware Version Control Systems." *Altium*, 21 Oct. 2020, resources.altium.com/p/best-practices-hardware-version-control-systems.

"Physical Computing with Python." *Projects.Raspberry Pi .Org*, projects.raspberry Pi .org/en/projects/physical-computing/13. Accessed 5 Dec. 2023.

"What Is a Version Control System?" *What Is a Version Control System?*, resources.pcb.cadence.com/blog/what-is-a-version-control-system. Accessed 5 Dec. 2023.