WEEK 2:

Date: 09/02/2022

Total hours: 12

Description of design efforts:

I spent this week designing a pre-amp circuit for piezo contact microphones, as well as writing software which can interact directly with the framebuffer devices of a RaspberryPi SBC. Additionally, I was able to retrieve the first images from our depth camera. Finally, as part of daily team meetings, I contributed to our project's functional description.

Piezo Pre-Amp:

One of the key features of our project is its ability to track the score of a game of table tennis. In order to do this reliably, it won't be enough to use a consumer-grade depth camera alone. In addition to video data, RACHEL will take in audio from various contact microphones attached to the table tennis table (herein "TTT"). This week, I adapted a design by Richard Mudhar [1] for a pre-amplifier circuit specifically for piezoelectric disks. The pre-amp is necessary because the signal we want to measure is very small and won't be discernible by our microcontroller's ADC at its original amplitude. Piezoelectric elements output voltage proportional to the pressure in the crystal, and because a table tennis ball is so small and light, it isn't able to exert much pressure on the crystal through the playing surface of the TTT. My adapted schematic is shown in Figure 1.

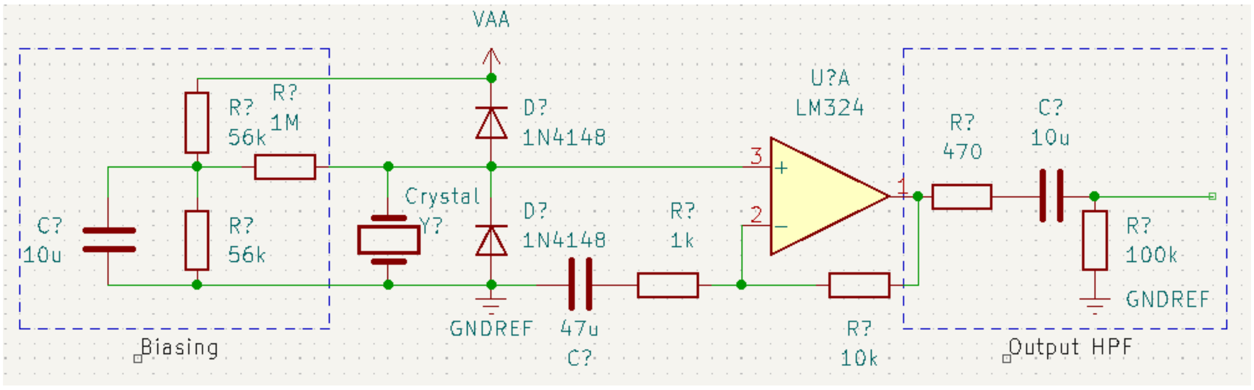

Figure 1: A schematic of a pre-amplifier circuit designed to work with piezoelectric elements

In the figure above, one can see two boxes. On the left is circuitry to apply a biasing voltage to the piezo, which allows the output signal to be balanced. Without this biasing, the negative portions of the output signal would be clipped off by the protection diodes shown in the middle of the diagram. On the right is a passive high-pass filter and a resistor. I'm not exactly sure of the purpose of the 470Ω resistor, as that is a design element taken from [1], but I presume that it serves to limit the output power of the amplifier for very high-frequency signals. It's likely that the component values in this schematic may need to be changed from those in the original design, because I opted for an op-amp which is more readily available to the team than the component specified in [1].

Framebuffer Manipulation:

Generally, when one wants to write a graphical program for Linux, they take advantage of the graphics libraries and windowing systems included in most distributions. Due to serious technical difficulties with relation to the windowing system and desktop environment on the RaspberryPi SBC in our design, I was interested in writing our software such that it could be run without any desktop environment or windowing system. This led me to spend many hours of the week following a series of articles [2][3] detailing the basic interface used to draw to displays on Linux: the framebuffer. A framebuffer is essentially an array of raw pixel data which is shown on the connected display device. By writing to this array, it is possible to control each individual pixel of the display. This week I was able to write several small test programs to practice interacting with framebuffers and with the low-level IO of a Linux system in general. Shown in Figure 2 is the output of a small program based on [2], which makes

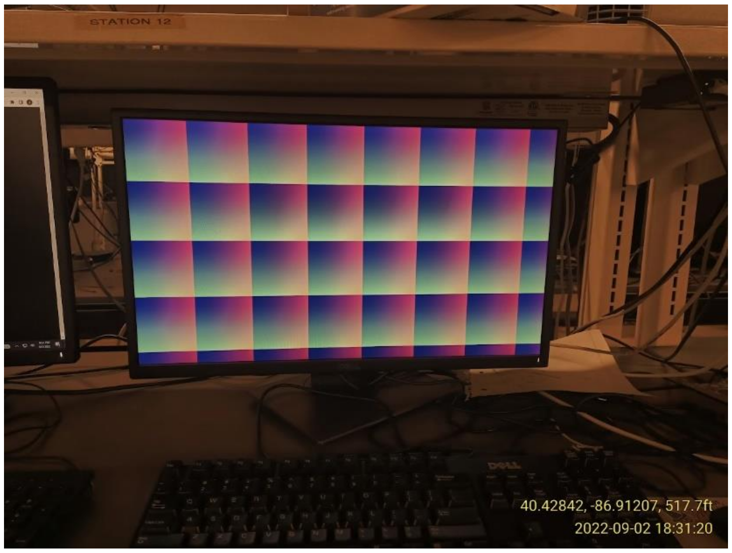

Figure 2: The output of a small program showcasing the use of the framebuffer. The red and green values shown on the screen depend on a particular pixel's coordinates, and the blue value fades through all 256 of its possible values.

use of the framebuffer interface to write directly to the display.

Depth Camera Interfacing and OpenNI2:

Our project is currently based on an Asus Xtion PRO Live depth camera, which will be used in conjunction with audio data to track the ball and ultimately keep score. In order to interact with depth cameras, it is common in the field of robotics to use a library known as OpenNI2. This week, I very slightly modified the source of this library to allow it to build successfully on Raspbian Bullseye, the distribution running on our RaspberryPi. After a successful build and install to the Pi, I was able to use a sample program provided with the code to retrieve the first images from our depth camera. Unfortunately, I'm not able to share any of these images at this time, because the desktop environment on the Pi is currently inoperable and this sample program relies on it to function.

As a solution to this problem (and a natural progression from using sample code to developing our own codebase), I have been working on a program to read data from the camera and write it directly to the framebuffer. This program will likely serve as the basis for all the game graphics we intend to display, as well as the ball tracking capability inherent to our project. At this time, my intention is to either a) finish my version of the viewer by early next week and move on to incorporating OpenCV blob tracking, or b) replace the RaspberryPi in our design with my Thinkpad T420 (with its working graphical environment) to allow me to use more standard graphics tools.

References:

[1] https://www.richardmudhar.com/blog/piezo-contact-microphone-hi-z-amplifier-low-noise-version/

[2] http://raspberrycompote.blogspot.com/2012/12/low-level-graphics-on-raspberry-pi-part_9509.html

[3] https://medium.com/@avik.das/writing-gui-applications-on-the-raspberry-pi-without-a-desktop-environment-8f8f840d9867