Lab 1: Setting Up piTFT

By Alec Newport [acn55] and Eldor Bekpulatov [eb654]
Thursday Lab - Section 403
October 3, 2019

Introduction

This lab was designed to get us used to the Raspberry Pi development environment and to get our feet wet working with the Unix operating system. Lab consisted of three parts. First part of the lab primarily focused on configuring the Rasbpian OS in our Raspberry Pi 3 boards and assembling our lab kit for ECE5725. Second part consisted of installing software for video playbacks and drivers for the piTFT display module. The final part of the lab included writing python scripts that polled the physical buttons installed on the piTFT board to control the video playback using named pipes.

Design and Testing

Section 1

We started by physically constructing the Raspberry Pi/TFT system. We placed the Pi in a case to protect it from both physical and ESD damage. We attached a ribbon cable to the TFT to allow access to the GPIO pins and attached the TFT to the Pi. Finally, we plugged in all the peripherals: the formatted micro SD card, display, keyboard, mouse, and Ethernet cable. After getting our setup checked by the TAs, we plugged in the power cable to turn on the Pi. See Figure 1 for the completed system.

Figure 1: Completed Raspberry Pi/piTFT System

As part of Homework #1, we formatted the microSD card and installed the latest release of the Linux kernel. So once we power on the Pi, it booted to the desktop. With the system up and running, we began to configure the system. The lab handout had a detailed step-by-step guide to do this manually. The following describe the steps we took and why:

1. We changed the password for security
2. We changed the hostname to easily distinguish our Pi from all the others on the network
3. We set the location and timezone so the system would have the correct date and time

4. We set the keyboard configuration to that corresponding to our keyboard
5. We enabled ssh to allow us to access the Pi remotely
6. We expanded the filesystem to encompass all available space in the microSD card
7. We set the audio output to always go through the onboard audio jack to allow us to use speakers with the system
8. We updated raspi-config to the latest version to retrieve any security improvements or bug fixes
9. We ran ifconfig -a to get the ip address to enable us to ssh into the Pi (see Figure 2)

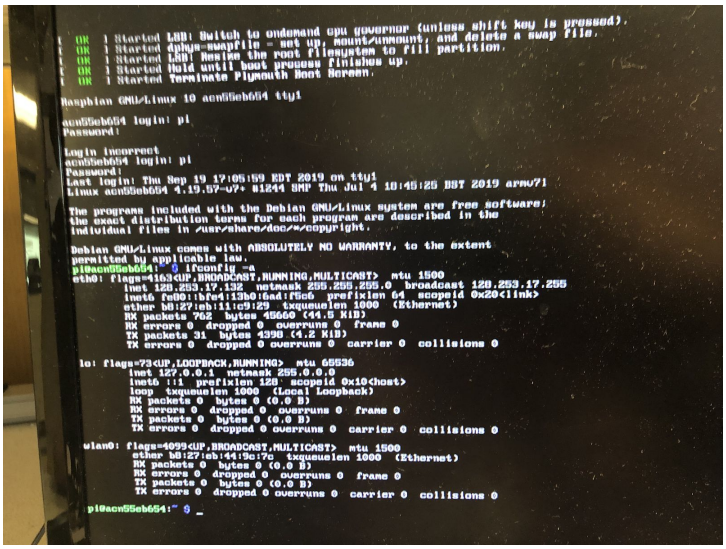

Figure 2: ifconfig -a

10. We updated and upgraded the kernel to the latest version to get any patches available. Latest available version of the kernel came out to be version 4.19.66-v7+ (see Figure 3)

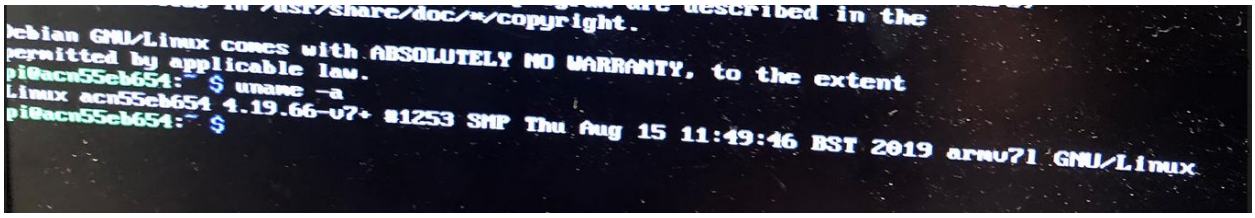

Figure 3: uname -a

11. We also installed Vim terminal editor
12. Last but not least, we backed up our MicroSD card, just in case!

Section 2

Having configured many all of the settings in section one of the lab, we now needed to install and configure piTFT display module driver. This section was also given as a step-by-step guide in the lab handout.

To start off, we installed a list of libraries/applications to support the piTFT using `apt-get` and `pip`:

- bc
- fbi
- git
- python-pip
- python-smbus
- python-spidev
- evtest
- Libts-bin
- evdev

The `python-pip` application is used to install python packages on the Pi. The evtest and `libts-bin` applications are used for testing the piTFT.

Next, we edited the /boot/config.txt with administrative access. In this config file we specified the appropriate Pi model, rotation of the screen, and frequency in which the pi updates the piTFT screen. To see our changes resonate, we rebooted our Pi as instructed and ran the dmesg command. The output was as expected, meaning both the instructed lines specified in the handout were observed in our Pi's output.

Next, we created/edited existing udev rules to assign touchscreen an event number which is then used to make accessing the touchscreen more straightforward. All of these rules had to be created with administrative access. Once all the rules were added, we had to unload and reload the touchscreen module driver to see the changes. In order to achieve that the following commands were run in sequence: sudo rmmod stmpe_ts and sudo modprobe stmpe_ts. To check our results we ran ls –l /dev/input/touchscreen and as expected touchscreen was associated with event number (see Figure 4).

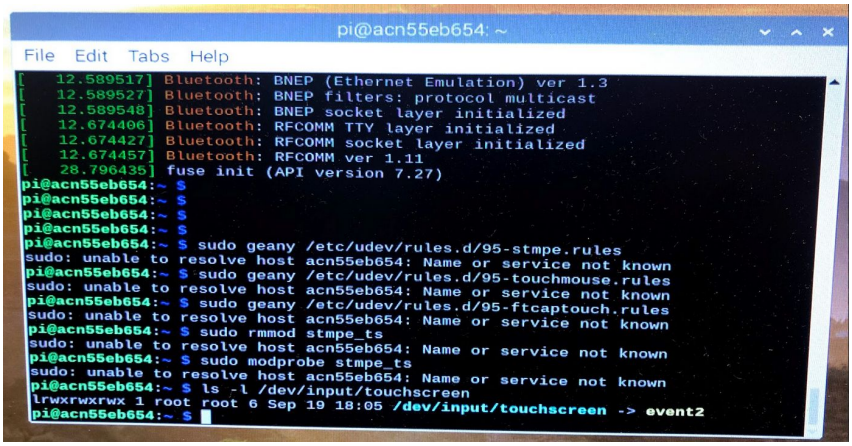

Figure 4: ls –l /dev/input/touchscreen

Now that we had the piTFT configured and recognized and touch features accessed, we were instructed to calibrate the touch screen. This was done through adding a new file in /etc/pointercal file. The calibration values were provided in the lab handout.

Final step in setting up the piTFT was to set up the command line terminal when loaded on the piTFT. Some of the aspects that needed to be specified were given in the lab handout and the included the the font type and font size. This was to give a more compact, readable font on the smaller piTFT screen. Unfortunately, we could not get a picture of the terminal window loaded on the piTFT. The last step was to reboot and backup the drive!

Section 3

In the second week, we used the system we set up to control video playback using mplayer and a fifo. A fifo was used because it acts as a portal of communication between the Python code controlling the video playback and mplayer itself. We sent commands in plain text to the fifo, just as we would from the command line, and mplayer read them from the fifo and acted upon them.

The first Python script we wrote, fifo_test.py, simply waited for commands from the user, translated them into the exact syntax desired by mplayer, and sent them to the fifo. To do this, the script prints a prompt to the screen and waits for user input. When the user hits the Enter key, sending the string to Python, the script determines which mplayer command it corresponds to, if any, and, if it represents a valid command, sends the corresponding mplayer string to video_fifo.

Next, we confirmed that we were able to receive signals from a button on the TFT using a script called one_button.py. This script simply polls the input pin for one of the TFT buttons and prints out a message when the value on that pin goes low, indicating a button press. The button circuit has a pull-up resistor connected to the pin as well, so when the button is not pressed, the signal on the pin will be high, and nothing is printed.

We then expanded one_button.py into four_buttons.py, which, as the name suggests, polls all four buttons on the TFT to watch for presses and prints a corresponding message. One difference, however, is that when a button press is detected on GPIO19, the script terminates. This demonstrates actual control of the script using a button, rather than just printing text. It also allows complete control of the execution of the script using just the buttons, so we don't have to press Ctrl-C to end the script when we want to.

The final Python script we wrote, video_control.py, combines everything we did in the previous scripts. It polls the buttons on the TFT and sends a command to mplayer through video_fifo on a press. One of the buttons pauses/plays the video, one fast forwards 10 seconds, one skips back 10 seconds, and one quits both mplayer and the script itself. In this last case, mplayer must quit before the script because if the script terminated before mplayer, there would be nothing left running to send a quit command to the fifo.

Finally, we wrote a bash script to make starting mplayer and the video_control.py script easier. Since this normally requires multiple commands, the command to start mplayer correctly is relatively long, and there's no reason to start video_control.py without mplayer running, it makes sense to create a bash script, which automates all of this, simply by the user typing its name. This script uses absolute addresses for all required files, so it can be moved anywhere on the system and still run correctly. We finished early in our lab section and checked out with the TAs.

Conclusions

This lab went smoothly for us for the most part. The only issue we had was getting the button presses on the TFT to register correctly. We decided against implementing a debouncing state machine in software because of the significant additional time required to do so. We didn't find anything wrong/confusing about the lab.

Code

Code stored on server at /home/Lab1/acn55_eb654_Lab1