ECE5725

Lab Report：Lab 1

AUTHORS: Hehong Li, Chenghui Li

NETID: hl778, cl2228

LAB SECTION: Thursday

DATE: 9th Sept. & 16th Sept.

● Introduction

The aim of the first lab is to achieve the control of video displaying and audio through raspberry pi and software programming. The PiTFT display and RPi 4 are assembled together at first and using the Linux commands to complete the basic configuration of RPi. After the initial configuration, a video *bigbuckbunny.mp4* is displayed both with the mplayer application and the piTFT. The audio and video playback of the mplayer are totally controlled by a FIFO and the python files containing FIFO commands. Four buttons next to the screen are connected to the GPIO pins to achieve certain functions (*'pause'*, *'backward/forward'*, *'quit'*) while pressed and the results are recorded to analyze the properties of this control system. The final task is enabling python control file to work in the background and the video played in the foreground for controlling the video with the piTFT buttons. This report will focus on the detailed procedure of the experiments done in week1 and week2 and the observations we found during the testing.

● Design and Testing

The design and testing section will be divided into two parts. The first one the methods to perform this experiment of lab 1, the second one is the results represented by pictures and the observations and discussions based on the results.

- Design Methods

First, the main design steps were listed and explained as following and the procedure were divided into two different parts according to the date of the lab. It was separated into two different weeks and the progresses in each lab are described.

Week 1: Basic Configurations & Video Play

In the first step, all components in the lab kit including RP4, SD card and the TFT screen were assembled together connected to the HDMI monitor, keyboard and mouse, and charged with the cable. Then the Raspbian kernel was installed initially and the *raspi-config* tool are used for setting up. Due to the use of HDMI monitor, the initial install of the Raspbian Buster Kernel was directly on the Raspbian Desktop. The *'System Options'* option in the tool was used to set the unique hostname and configure WIFI connection of the RPI. The *'Localization'* option changed the locale settings to 'en_us. UTF8 UTF8' and set the timezone to *'Eastern'*. Then the plugged keyboard is configured to *'English (US)'* and the audio output was set to be 'Headphones' in the 'System Options' settings. Next, the Raspbian kernel was upgraded by the Linux *sudo* commands and it

was rebooted to start the upgraded kernel. After that, personal laptop was connected to the IP of 'wlan0' WIFI connection and the SD card was backup.

The python packages were installed on the Pi through the python-pip application and using the *evtest* and *libts-bin* application to test the piTFT. After the basic configuration, piTFT info was added in the config.txt by adding certain lines to the end of the */boot/config.txt* file. When completion, the command *dmesg* was used to check the piTFT by finding the "stmpe-spi" and "graphics fb1" lines. Then a *'udev'* rule was generated by adding the certain lines to different *.rules* files and the driver for the touch screen was unloaded and reloaded. Next, the initial piTFT calibration was set by editing the file */etc/pointercal* and the Linux console window on piTFT was started on the piTFT. The piTFT blanking was turned off when the console is displayed by adding an entry in */etc/rc.local* and some lines were added at the end of the file. After that, the Linux console should start on the piTFT if all procedures work well.

The final step is to run a video form the RPi on the PiTFT through the mplayer application. Two speakers were connected with the RPi and the audio playback was operating correctly by the speakers. The testing results of mplayer on piTFT, a console window and the laptop were compared with each other to find the differences on both the video and audio. After completion of all the steps of week 1, the lab was finished and the SD card was stored.

Week 2: Controlling Video Playback with External Devices

The video/audio playback controls of the mplayer were explored by analyzing the commands and these options were decided to be map into the control 'panel'. To control the mplayer, a FIFO file was created to run the video and execute certain instructions. The main procedure to do so is booting the RPi on the TFT, running *startx* to operate on the monitor and using two console windows to run and control the mplayer by commands to the FIFO. After completion of the control by a FIFO, a python file *fifo_test.py* was created to send some valid commands to the FIFO setup and control the mplayer. The next step is to detect and obtain the values of input from GPIO pins. This is implemented by setting different functions to the GPIO pins connected to the button, then the buttons were enabled to control the video display through the commands written in python files *one_button.py* and *four_button.py*, where the first one is for testing. Then for controlling the myplayer using a python program, a python file *video_control.py* was created and the four buttons in the piTFT kit was programmed to connected to four various mplayer actions. Finally, a bash script *start_video* was generated to launch the mplayer and the video control file. After all the operations, the control file will be arranged to run background using *'&'* command and the video display will run foreground.

- **Testing Results**

Then comes to the result testing part. This section was also divided to two parts based on two weeks for the experiment. And some figures were presented in the following pages to visualize the results and some problems met during the lab are also discussed.

Week 1: Basic Configurations & Video Play

We first installed and configured the Raspberry Pi, after power-up, we used basic commands like *"whoami"*, *"hostname"*, *"uname -a"* to get the information of our raspberry Pi:

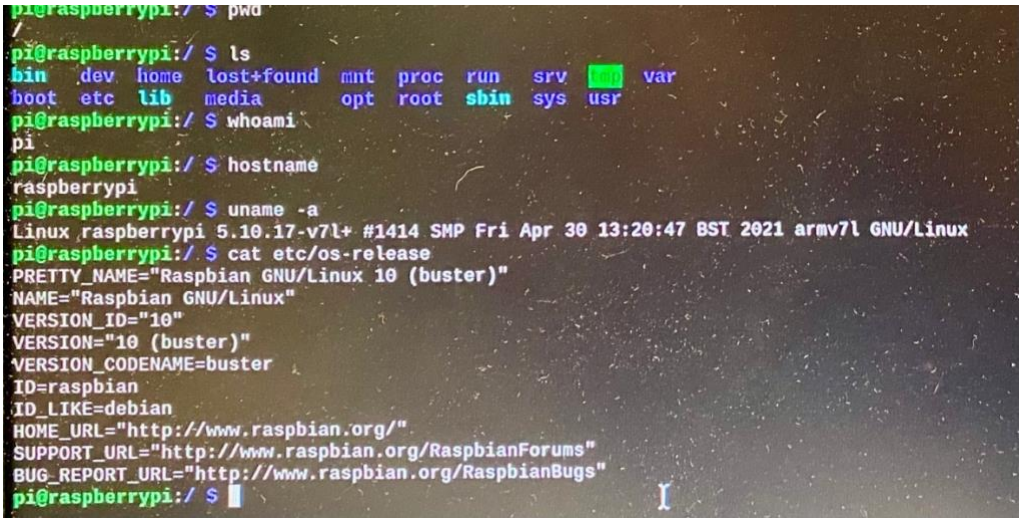

Figure 1: Some Simple Checks

It could be noticed that the Linux version is not up-to-date, and we have not changed the host name yet. After we updated it and change the user information, we can get:

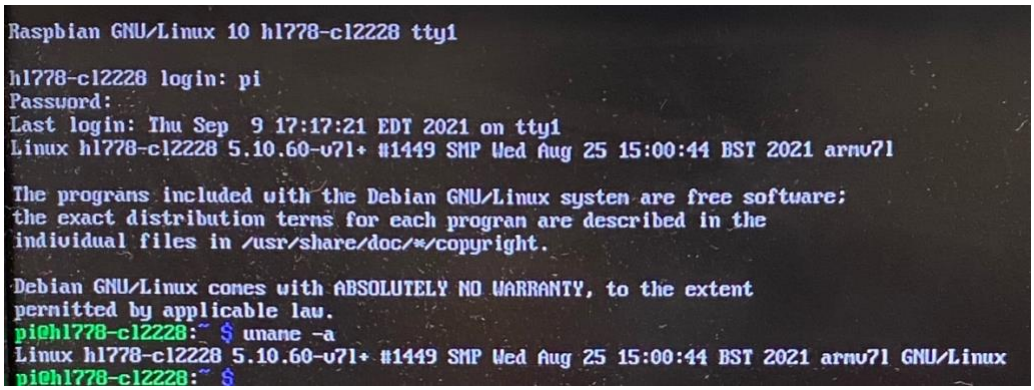

Figure 2: Check for the Kernel Version

The above figure shows that the hostname is changed to *"hl778-cl2228"* (netids of our team), and the Linux version is up-to-date now, then the *"htop"* command is used to show the running processes:

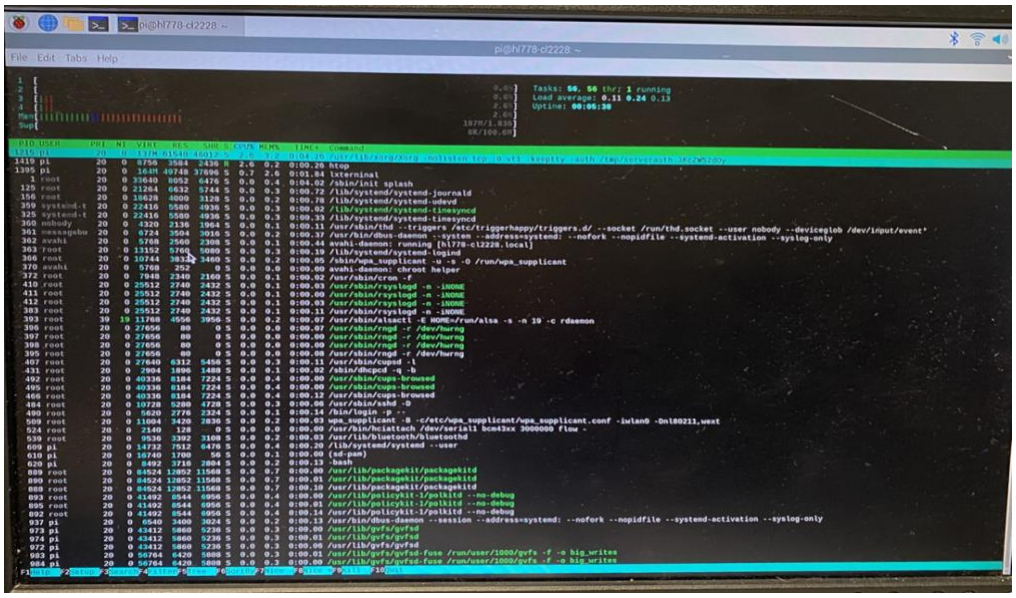

Figure 3: Running Processes

The next step was that to connect the pi using *ssh* from our laptops, so we used the command *"ifconfig -a"* to find out the IP address of the raspberry pi:

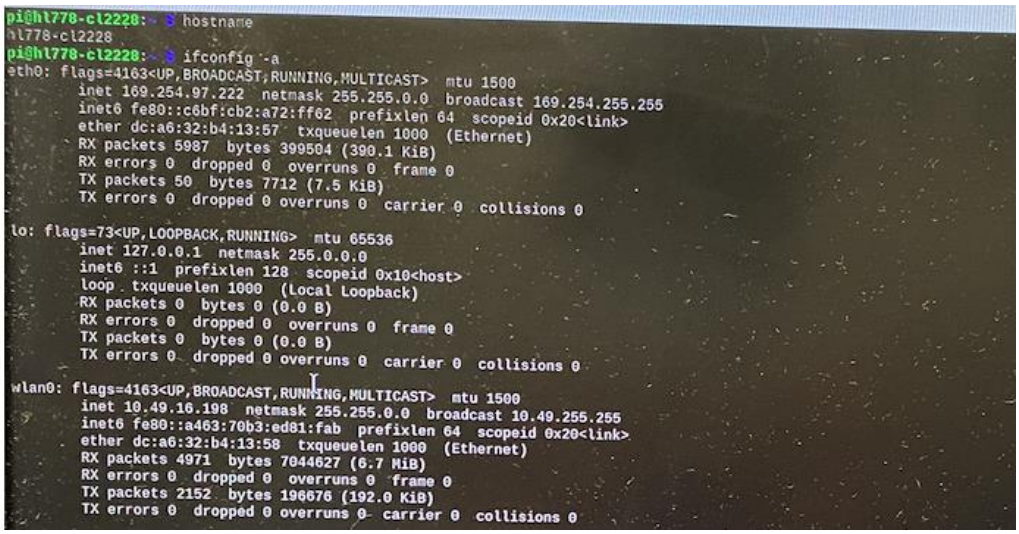

Figure 4: Check the network connections

The WLAN IP address of our raspberry pi is 10.49.16.198, and we successfully got access to the pi using *ssh* on my laptop with *"ssh @pi10.49.16.198"*. Next, we checked the CPU information of the pi by using *"cat /proc/cpuinfo"*:

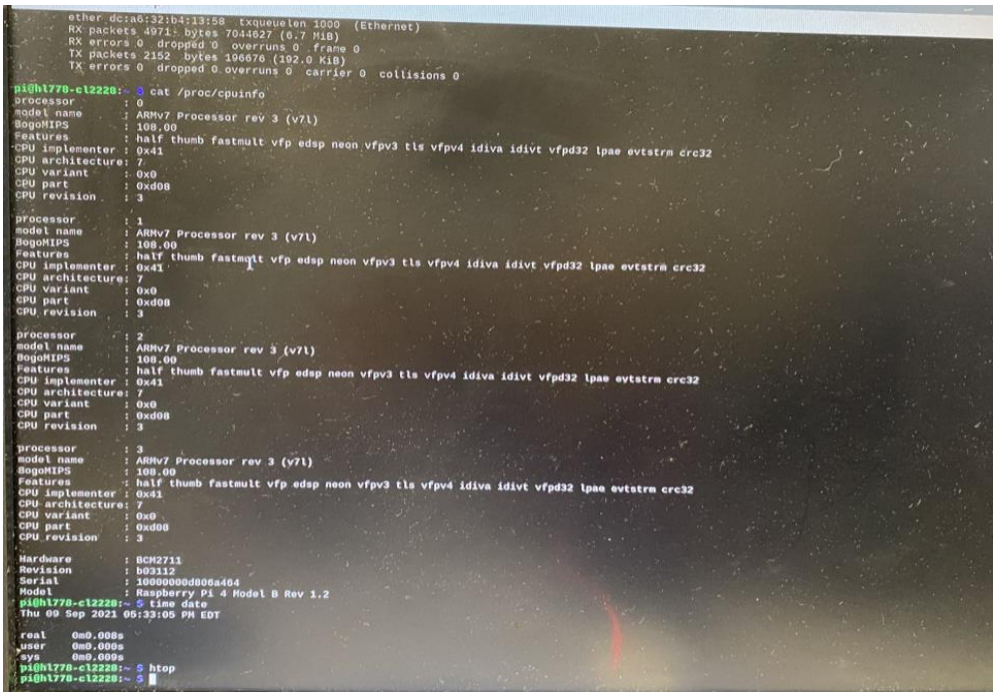

Figure 5: Checking the number of cores

From the above figure, it is indicated that the number of cores is four. After this, we used about half an hour to back up the pi. First, we used *"sudo shutdown -h now"* to shut down the pi, but we saw that the piTFT was still on, we asked TAs and found out as long as the green light by the pi is off, we can pull off the power.

The system set up was done, we proceeded to set up the piTFT. First, we used *"sudo apt-get install -y bc fbi git python-pip python-smbus python0spidev evtest libts-bin"* and *"sudo pip install evdev"* to install the necessary software.

After adding piTFT info to config.txt, we used *"dmesg"* to check whether the configuration is right:

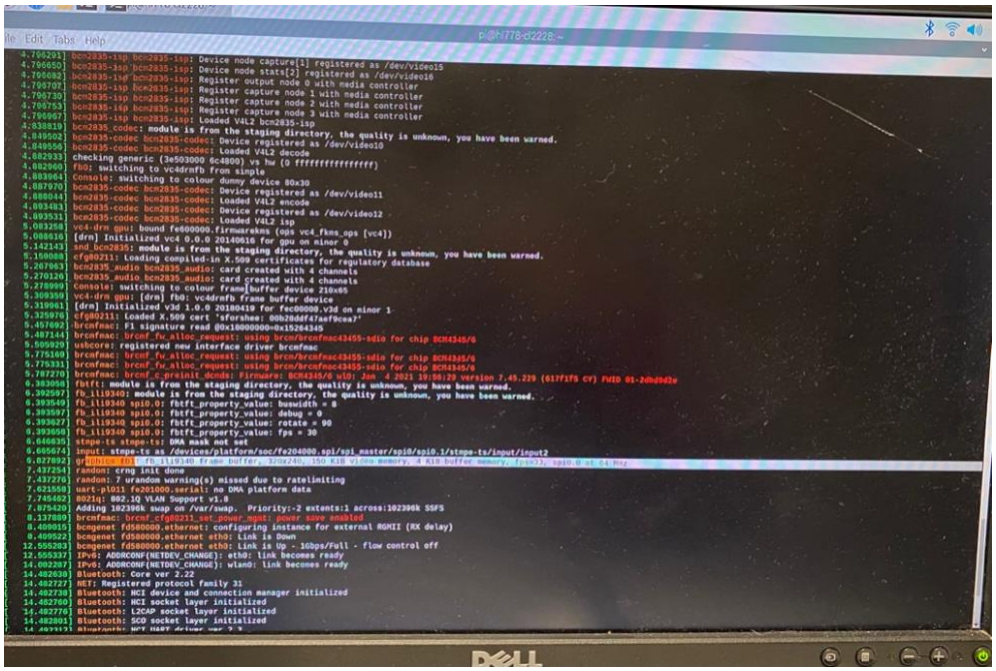

Figure 6: Configuration Check (Part.1)

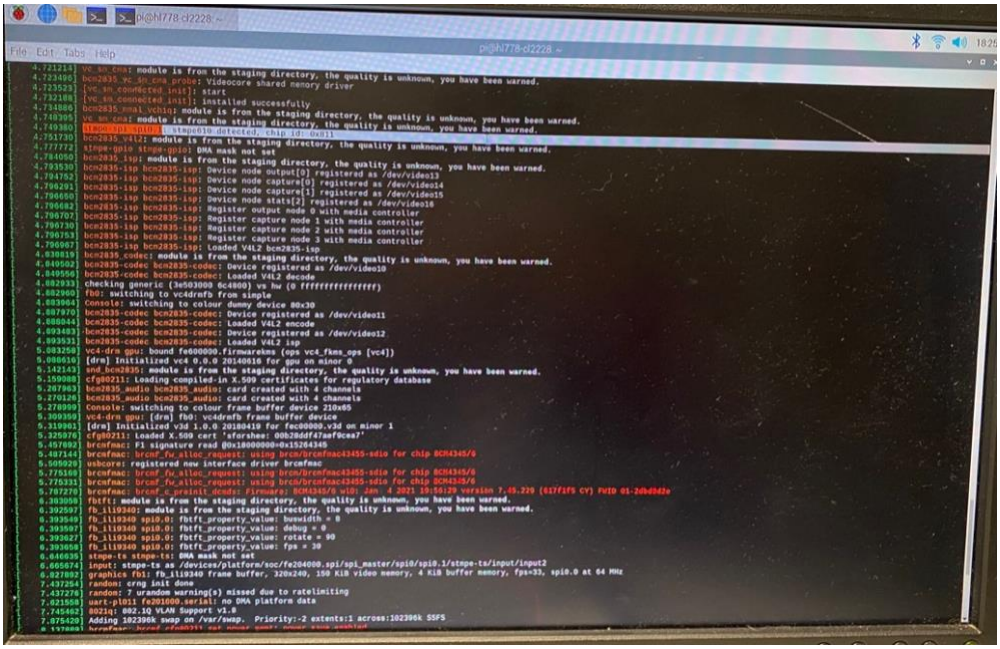

Figure 7: Configuration Check (Part.2)

We checked the link started with *"stmpe-spi"* and *"graphics fb1"* and made sure they are all set. The screen size is 320x240, 4KB buffer, fps=33, 64MHz.

After setting up the piTFT, we added udev rules to *"/etc/udev/rules.d/95-stmpe.rules"*, *"/etc/udev/rules.d/95-touchmouse.rules"*, and *"/etc/udev/rules.d/95-ftcaptouch.rules*" to catch events of the piTFT. We made a typo mistake here by typing *"touchmose"* rather than

"touchmouse", thanks to TAs helped us find it. After that, we rebooted the Pi and tested whether the piTFT was running correctly. We detected events caused by touching the screen:

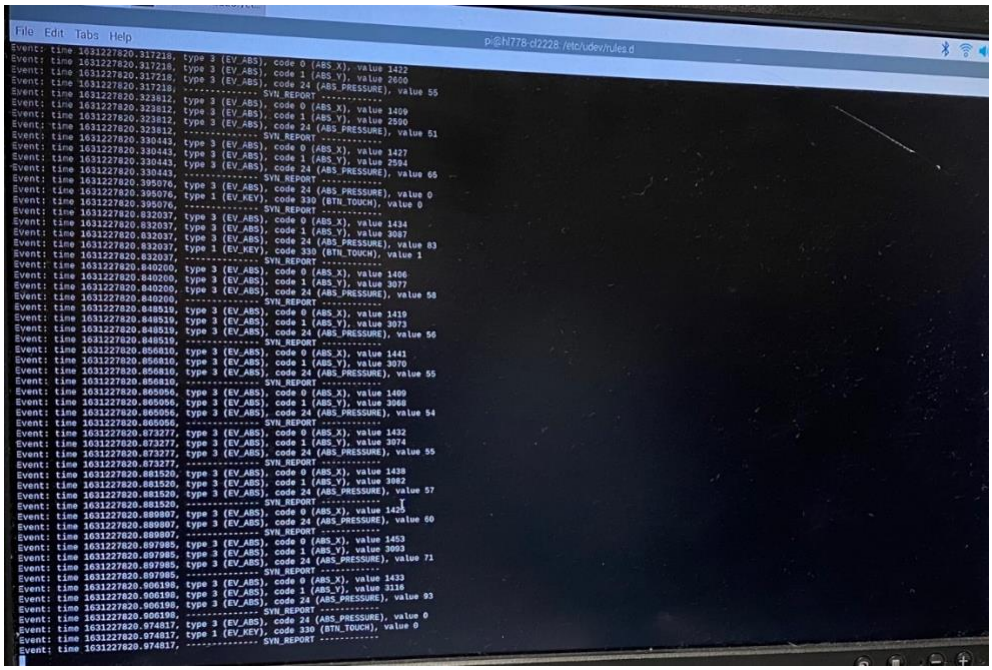

Figure 8: Event Detection

We unloaded the driver and reloaded again according to the instruction, after a serious of configurations were set right, we finally reached the step to play the video by the command *"sudo SDL_VIDEOFRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -vo sdl -framedrop bigbuckbunny32p.mp4"*. Again, we had a typo here by typing *"sd1"* rather than *"sdl"*, a bug that hard to detect. With the help from TAs, we finally found this bug and ran the video successfully.

When the video was running in different methods (with *sudo* and without *sudo*) and on different platforms (piTFT, desktop and *ssh* window). Then the differences appear which were that the mplayer could run the video with the audio successfully on all of these three platforms without *sudo* command. However, for the commands without *sudo*, only the one on the desktop using *startx* has the audio and the others only display the video without audio.

Week 2: Controlling Video Playback with External Devices

First, we created a fifo file at *"/home/pi/0916/video_fifo"*, and started the video with *"mplayer -input file=/home/pi/0916/video_fifo bigbuckbunny320p.mp4"*. We tested the fifo by using another *ssh* connect on my laptop to *"echo "pause" > /home/pi/0916/video_fifo"*, the video paused successfully, meant that the fifo worked!

Next, we created a python script called *"fifo_test.py"* which reads the input and if the input is a single *"q"*, the program sends a command to the fifo to quit the mplayer. And if the input to python is *"p"*, the program sends a command to the fifo to pause the mplayer.

After testing the python, we started to work with GPIO, the first stop was to write a python program to detect whether a button is pressed. We created a file called *"on_button.py"* to detect whether the GPIO 17 works, which shows no error in testing.

Next, we moved forward to test the four buttons using a python program called *"four_buttons.py"*, which initiated the GPIO and using a forever loop to detect if a button is pressed. We made a small mistake here that let the while loop contain the GPIO setup statements, which works but it is not a right choice.

After debugging our *"four_buttons.py"*, we started to create a *"video_control.py"* to control the video by four buttons on the piTFT. Similarly, we used a while loop that runs forever until a break. It keeps detecting our input, if we press the 17 button, the program excutes *"echo "pause" > /home/pi/0916/video_fifo"* to pause the video; when we pressed the 22 button, the program executed *"echo "seek 10" > /home/pi/0916/video_fifo"* to fast forward 10 seconds; ; when we pressed the 23 button, the program executed *"echo "seek -10" > /home/pi/0916/video_fifo"* to rewind 10 seconds; and when we pressed the 23 button, the program executed *"echo "quit" > /home/pi/0916/video_fifo"* to quit the mplayer as well as break the while loop of python, leading the end of the program.

The python program went well and we did not meet any bug, so me moved to the last step—using a bash shell to run the video in the front and run the python program in the back so that we can use four buttons on the piTFT to control the video with the python program running in the backend. We put the program in the backed by *"python /home/pi/0916/video_control.py &"*. We wrapped this statement and the mplayer statement to the bash shell. Overall, we did not make any bug and these steps went great magically, so we finish the whole lab.

● Conclusions

Week 1: Basic Configurations & Video Play

We actually went quiet well at the first-half part of the task, including installing and configuring the Raspberry Pi. We did not meet any problems in connecting the Raspberry Pi to Cornell WiFi, this is lucky because I heard from my friend Jinyang who worked in Lab Wednesday that they had some daunting troubles about the WiFi which took them near an hour.

But after having set up the Linux system and installed the necessary applications, we have trouble in configuring the PiTFT, mainly because typo, especially when it came the the command *"sudo SDL_VIDEOFRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -vo sdl -framedrop bigbuckbunny32p.mp4"*. We spent at least half an hour with troubles regarding this command, and professor also talked about my silly mistakes at the lecture next day, this is an important lesson that I really need to get rid of my bad habit of typo. Thanks much to my partner Hehong for not killing me for my typos. I also learnt from the week one that it would be much more convenient and faster to use SSH on my laptop than use the keyboard and the monitor at the lab. I improved by doing this at the week two and it turned out to be super efficient!

Although backup is important, it really took a long time to back up the Raspberry Pi during the lab time (we used about half an hour on it). This is the main reason that we nearly missed the checkout time that day, but the good side is that I saw a wonderful night view at Cornell that day and it was the first time I was still at Cornell after sum goes down.

Week 2: Controlling Video Playback with External Devices

Week 2 is remarkable success as we finished the lab ahead of the schedule by more than an hour! In week 2, the main knowledge I learned is how to use script, fifo, and python to control the mplayer. Since I have many experience in writing scripts in Linux and python programs, this was not a hard job for me so we went quite fast! Fifo is magical and the reference on Canvas is useful! Thanks to great help from my partner, professor and TAs!

● Code

Our codes have been uploaded to the server, at the path: /home/lab1/Th_cl2228_hl778_Lab1.