

Lab1 Report

Lab section 402

Xinyun Guo (xg284) Zhuoheng Li (zl764)

2019-09-25

Table of Content

Introduction	3
Design and Testing	3
Part 1	3
1.1 Assemble the lab kit for ECE5725	3
1.2 Initiate Raspbian Kernel install	3
1.3 Update the Raspbian kernel.	4
Part 2	5
2.1 Install drivers and software for the piTFT display	5
2.1.1 Install python packages on the Pi	5
2.1.2 Install evdev on Pi	5
2.2 Configure drivers and software for the piTFT display	6
2.2.1 Add piTFT info to config.txt (appendix)	6
2.2.2 Run dmesg to check for piTFT	6
2.2.3 Add udev rules	6
2.2.4 Use evtest to verify touchscreen operation	6
2.2.5 Set initial piTFT calibration	7
2.2.6 Start console window on piTFT	7
2.3 Install and configure drivers and software for video and audio playback	8
2.3.1 Load the sample video	8
2.3.2 Install mplayer	8
2.3.3 Test video playback on the piTFT	8
Part 3	9
3.1 Explore Mplayer	9
3.2 Control the mplayer to with fifo	9
3.3 Use Python to control the mplayer with a FIFO	10
3.4 Get input from a button connected to GPIO	10
3.5 Get input from four buttons connected to a GPIO	10
3.6 Control player through a FIFO using a python script	11
3.7 Bash script	11
Conclusion	13
Appendix	13

Introduction

In lab 1, we developed Python scripts to control video and audio playback on the screen of Raspberry Pi using physical buttons installed on the piTFT board. To achieve this goal, we carried out several steps including installing Linux kernel, configuring settings for piTFT display, and programming using Python scripts, which will be fully explained in the following sections.

Design and Testing

1. Part 1

1.1 Assemble the lab kit for ECE5725

To implement the project, we first assembled the Raspberry Pi kits, including installing the breakout cable to the TFT, connecting all external devices, and inserting the SD card. After that, we were able to power up the Raspberry Pi and continue.

1.2 Initiate Raspbian Kernel install

In this step, we learn about how to initiate the Raspberry. Using the command `'uname -a'`, we can have a clear view of the version of the kernel. Using command `'sudo raspi-config'`, we can change configurations of the Raspberry, including time zone, network setting, and audio setting which allows us to play the video with headphones in the following tasks.

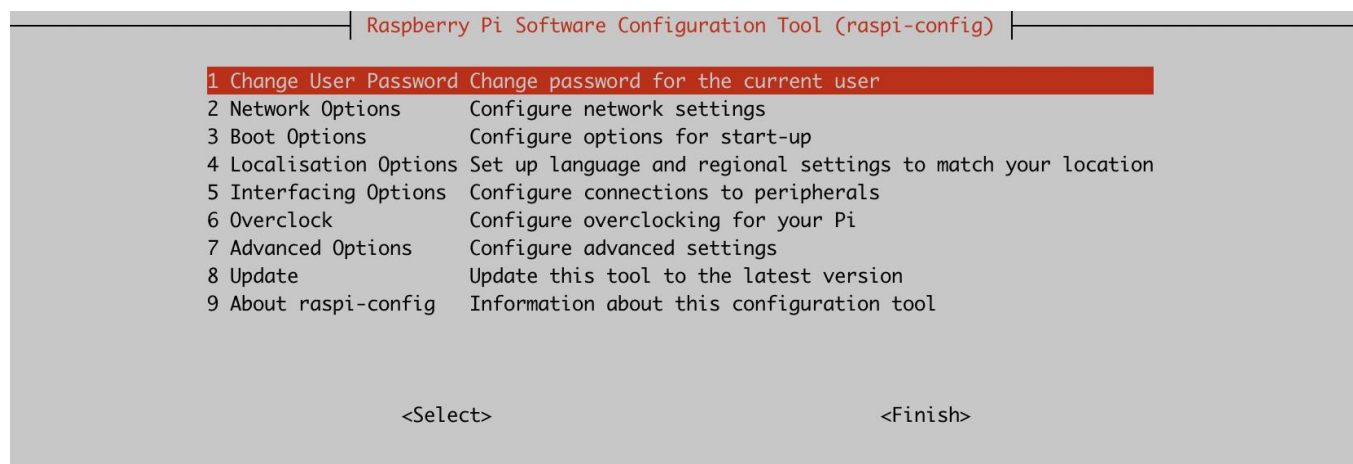


Figure 1 the configuration console

1.3 Update the Raspbian kernel.

we upgraded the Raspbian kernel to the latest buster release 4.14.66 with the following two commands:

- `'sudo apt-get update'`
- `'sudo apt-get upgrade'`

After the upgrade process, the `'uname -a'` will show our kernel has been updated.

```
Linux xg284-zl764 4.19.66-v7+ #1253 SMP Thu Aug 15 11:49:46 BST 2019 armv7l GNU/Linux
```

Also, using command `'ifconfig -a'`, we can find the IP address for the 'eth0' Ethernet adapter which allows us to log in remotely using our laptop.

```
pi@xg284-zl764:~ $ ifconfig -a
enxb827eb1c75e3: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether b8:27:eb:1c:75:e3 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.43.73 netmask 255.255.255.0 broadcast 192.168.43.255
    inet6 2607:fb90:2ed3:44fd:e842:7adc:d840:1716 prefixlen 64 scopeid 0x0<global>
    inet6 fe80::c572:8834:db7e:3547 prefixlen 64 scopeid 0x20<link>
    ether b8:27:eb:49:20:b6 txqueuelen 1000 (Ethernet)
    RX packets 150 bytes 14559 (14.2 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 131 bytes 20625 (20.1 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figure 2 info-config of Pi

Part 2

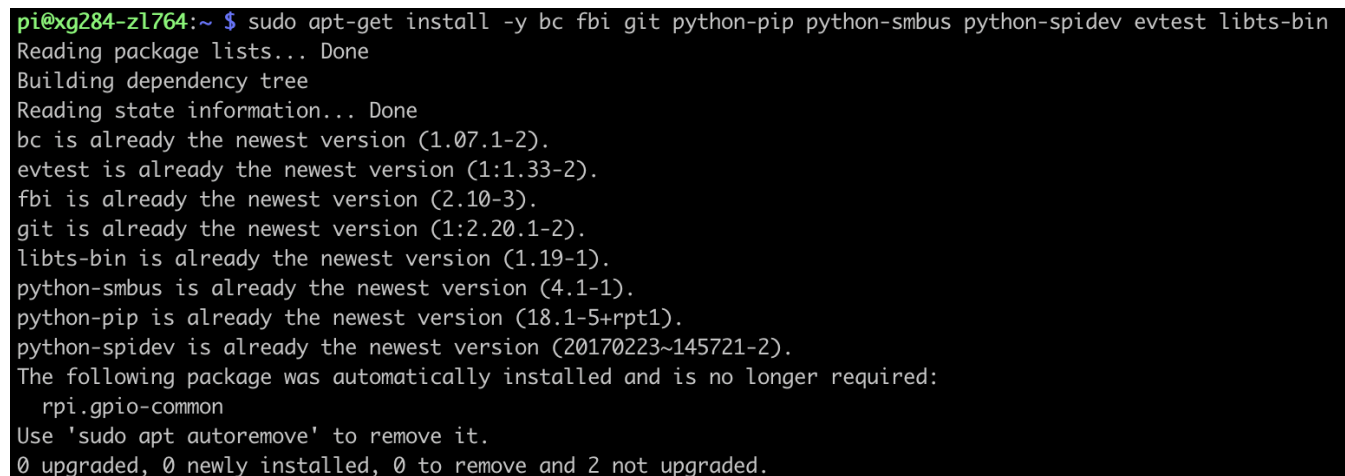
2.1 Install drivers and software for the piTFT display

2.1.1 Install python packages on the Pi

We used the python-pip application to install python packages on the Pi by running the command: `sudo apt-get install -y bc fbi git python-pip python-smbus python-spidev evtest libts-bin`

After running the command, we used the evtest and libts-bin applications to test the piTFT.

Testing:



```
pi@xg284-zl764:~ $ sudo apt-get install -y bc fbi git python-pip python-smbus python-spidev evtest libts-bin
Reading package lists... Done
Building dependency tree
Reading state information... Done
bc is already the newest version (1.07.1-2).
evtest is already the newest version (1:1.33-2).
fbi is already the newest version (2.10-3).
git is already the newest version (1:2.20.1-2).
libts-bin is already the newest version (1.19-1).
python-smbus is already the newest version (4.1-1).
python-pip is already the newest version (18.1-5+rpt1).
python-spidev is already the newest version (20170223~145721-2).
The following package was automatically installed and is no longer required:
  rpi.gpio-common
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
```


Figure 3 kernel has been updated

From the screenshot, we can see that the python packages have been installed successfully.

2.1.2 Install evdev on Pi

Further, we installed the evdev on Pi. We run the command: `sudo pip install evdev`

Testing:



```
pi@xg284-zl764:~ $ sudo pip install evdev
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: evdev in /usr/local/lib/python2.7/dist-packages (1.2.0)
```

From the screenshot, we can see that the evdev has been installed successfully.

Installing collected packages: evdev

Successfully installed evdev-1.2.0

2.2 Configure drivers and software for the piTFT display

After installation of the software we need for the piTFT display, we need to change the configurations of those drivers and software.

2.2.1 Add piTFT info to config.txt (appendix)

Testing:

After changes, we rebooted RPi, the PiTFT screen started out with white and switched to black as the boot sequence finishes. This indicated the initial changes worked correctly.

2.2.2 Run dmesg to check for piTFT

At this point, we run `dmesg` to check whether the modules for the touch screen are installed correctly.

Testing:

```
pi@xg284-zl764:~$ dmesg
[ 0.000000] Booting Linux on physical CPU 0x0
[ 0.000000] Linux version 4.19.66-v7+ (dom@buildbot) (gcc version 4.9.3 (crosstool-NG crosstool-ng-1.22.0-88-g8460611)) #1253 SMP
[ 0.000000] CPU: ARMv7 Processor [41061024] revision 4 (ARMv7), cp 10.53824
```

2.2.3 Add udev rules

Next, we created files to support the use of touch on the pitft.

Edit `/etc/udev/rules.d/95-stmpe.rules`

Edit `/etc/udev/rules.d/95-touchmouse.rules`

Edit `/etc/udev/rules.d/95-ftcaptouch.rules`

By doing this, 'udev' rule will be created which will assign the touchscreen an eventX number.

Testing:

To check the touchscreen's operation, we unloaded the driver and reload it, and then run: `ls -l /dev/input/touchscreen`

The touchscreen now was associated with an event number as shown in the figure below:

```
pi@xg284-zl764:/ $ ls -l /dev/input/touchscreen
lrwxrwxrwx 1 root root 6 Oct  1 18:43 /dev/input/touchscreen -> event4
```

2.2.4 Use evtest to verify touchscreen operation

With the above changes, the evtest utility may be used to check the operation of the touchscreen.

Run: `sudo evtest /dev/input/touchscreen`

Once evtest runs, we can tap the screen several times and the data associated with each tap will display as shown in the figure below.

Testing:

```
pi@xg284zl764:/etc/udev/rules.d $ sudo evtest /dev/input/touchscreen
sudo: unable to resolve host xg284zl764: Name or service not known
Input driver version is 1.0.1
Input device ID: bus 0x18 vendor 0x0 product 0x0 version 0x0
Input device name: "stmpe-ts"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 330 (BTN_TOUCH)
  Event type 3 (EV_ABS)
    Event code 0 (ABS_X)
      Value      0
      Min        0
      Max       4095
    Event code 1 (ABS_Y)
      Value      0
      Min        0
      Max       4095
    Event code 24 (ABS_PRESSURE)
      Value      0
      Min        0
      Max       255
Properties:
Testing ... (interrupt to exit)
```

Figure 4 touchscreen

2.2.5 Set initial piTFT calibration

In this step, we set information for initial calibration of the piTFT by editing the file `/etc/poointercal`

2.2.6 Start console window on piTFT

Next, we changed the following configurations to allow the Linux console window to start on the piTFT.

This was achieved by modifying `/boot/cmdline.txt`. Also, modifying entries in

`/etc/default/console-setup` helped to set the font and font size for displaying a more readable console on the small piTFT screen. Last, we added an entry in `/etc/rc.local` to turn off piTFT blanking when the console is displayed.

Testing:

After all the configuration changes, the Linux console should start on the PiTFT.

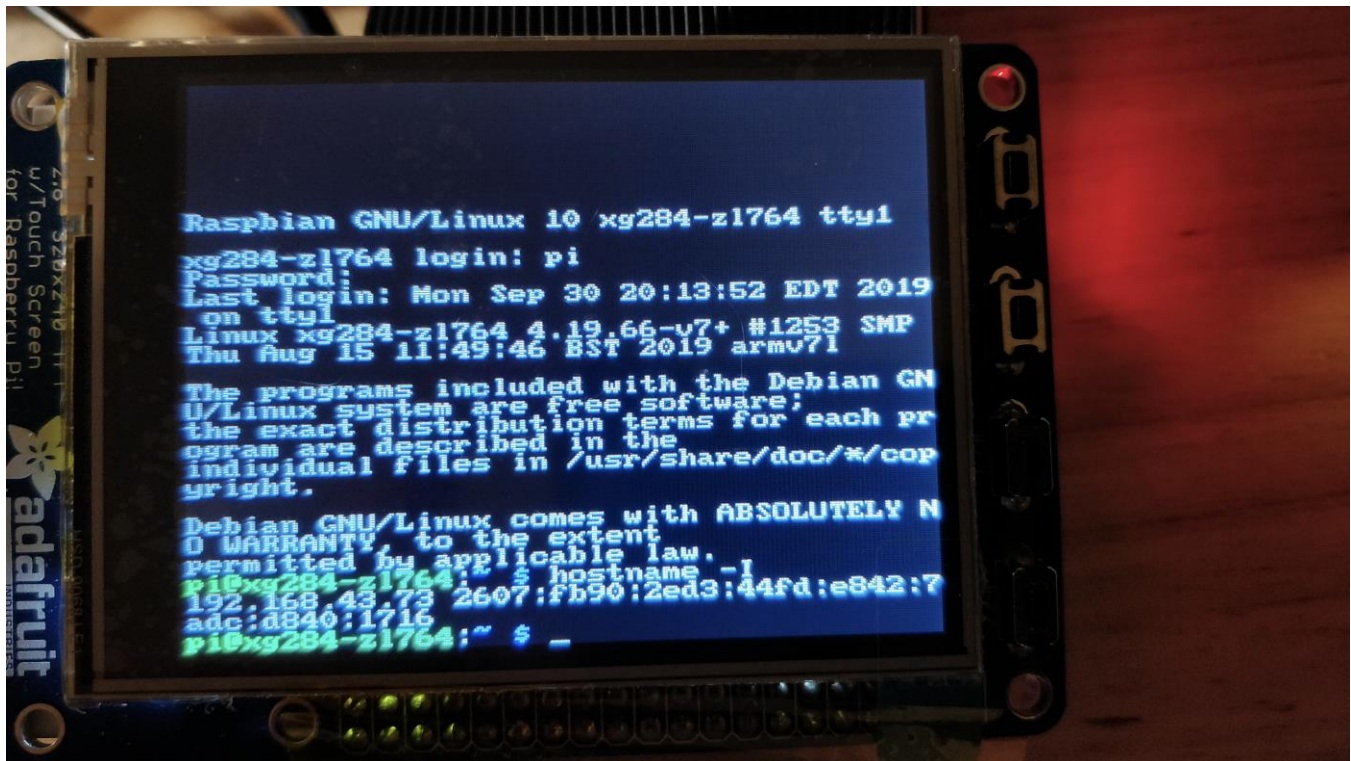


Figure 5 Linux console starts on PiTFT

2.3 Install and configure drivers and software for video and audio playback

2.3.1 Load the sample video

We first download the sample video file into our folder by running the following command:

```
wget http://adafruit-download.s3.amazonaws.com/bigbuckbunny320p.mp4
```

2.3.2 Install mplayer

Next, we installed the mplayer into our Pi with the following commands:

```
sudo apt-get update
```

```
sudo apt-get install mplayer
```

2.3.3 Test video playback on the piTFT

Finished the installation, we can play the sample video on piTFT window by running the following

commands: `sudo SDL_VIDEODRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -vo sdl -framedrop bigbuckbunny320p.mp4`

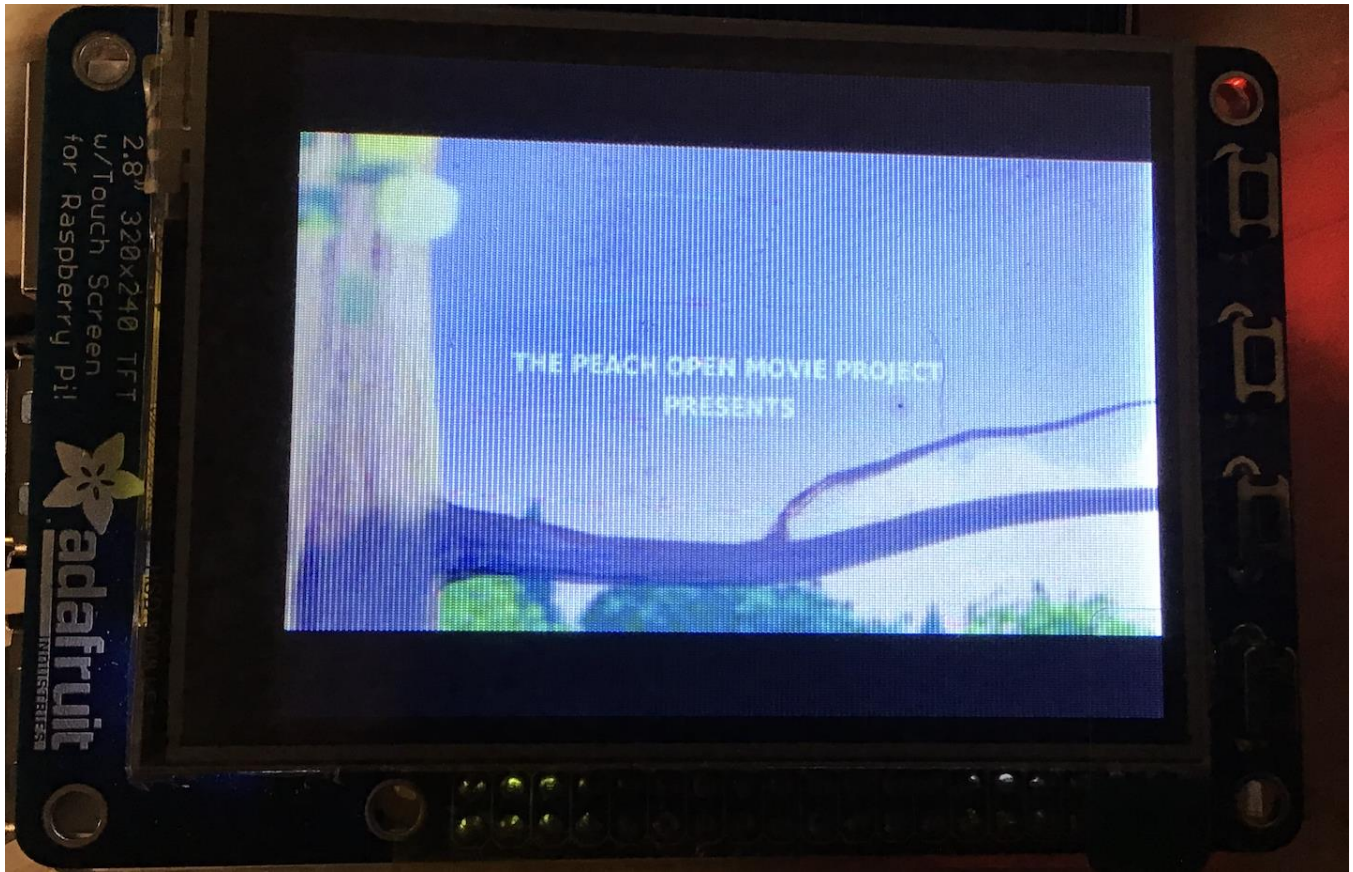


Figure 6 play the video on piTFT screen

Part 3

In this section, we learned about how to control the mplayer with a FIFO, and writing python scripts to control the video and audio playback on Pi using physical buttons installed on the piTFT board.

3.1 Explore Mplayer

In this step, we ran command `mplayer` to explore the options of the mplayer for playback control, including pause, skip ahead and back, and stop.

3.2 Control the mplayer to with fifo

Firstly, we create a fifo file “test.fifo” by entering the command:

```
mkfifo /home/pi/lab1/test_fifo
```

To let the mplayer controlled by the fifo, we used the command:

```
mplayer -input file=/home/pi/test_fifo bunny_video.mp4
```

This command will run the mplayer to play the “bunny_video.mp4” and it also lets the mplayer respond to the command sent to fifo. And we can use “echo” command to Sent “pause” command into fifo:

```
Echo “pause” > /home/pi/lab1/test_fifo
```

3.3 Use Python to control the mplayer with a FIFO

In this step, we wrote a python routine and ran it in the foreground. The routine will listen to the input from the keyboard. The python code we wrote is shown in figure 1 in the appendix below. We use `raw_input` to receive the input from the keyboard and use an if-else statement to judge which command it is. Then, we send the corresponding command to fifo and the mplayer will be controlled as expected. Testing the `fifo_test.py`, the mplayer responded to users' commands and successfully started, paused, and quit the video as expected.

3.4 Get input from a button connected to GPIO

To get input from the GPIO, we first need to import the GPIO modules. Then, we need to set up the GPIO which requires us to specify which GPIO pin you want to use, in this case, GPIO22. Then we use a loop to listen to the input from the GPIO22. Since the process ran in the foreground, when the button is pressed, in the console, “Button GPIO22 has been pressed” will show up. In addition, the `time.sleep(0.2)` is used to set a separation time between two triggering events. Otherwise, pi will detect several triggering events. The code is shown in figure 2 in the appendix.

Testing the `one_button.py`, once the button GPIO22 is pressed, the message “Button GPIO22 has been pressed” will display as expected.

3.5 Get input from four buttons connected to a GPIO

In this step, we wrote the code based on 3.4. The difference is that we set up four buttons instead of one button. The pin numbers of the four buttons are 17, 22, 23, 27. Then, we wrote a loop to listen to the input of the four buttons. If the button is clicked, the corresponding text will be shown in the console. Also, we set the GPIO 27 as the “quit” button, so if GPIO 27 is pressed, the process will terminate. The code is shown in figure 3 in the appendix.

Testing the `four_buttons.py`, when a button is pressed, the message will display that the corresponding button is pressed. Once the button (GPIO27) is pressed, the python program will terminate as expected.

3.6 Control player through a FIFO using a python script

In this step, we need to add a subprocess to each GPIO, and the subprocess will run when the corresponding button is pressed.

For GPIO 17, we add a subprocess for pausing the video. This command will replace all the codes in fifo with a pause. And the mplayer will read the pause command and pause the video.

```
echo "pause" > /home/pi/video_fifo
```

For GPIO 22, we send "seek 10 0" to the fifo which is a mplayer command to fast forward the playing video by 10 seconds.

```
echo "seek 10 0" > /home/pi/video_fifo
```

For GPIO 23, we send "seek -10 0" to the fifo to rewind the video by 10 seconds.

```
echo "seek -10 0" > /home/pi/video_fifo
```

For GPIO 27, we send the quit command to fifo to quit the mplayer. After that, this python program will be terminated as well. Thus, we added a break after executing the quit subprocess.

```
echo quit[0] > /home/pi/fifo
```

```
break
```

The code is shown in figure 4 in the appendix.

Testing the video_control.py, when a button is pressed, the corresponding command will be sent to fifo, and the mplayer will respond as expected. For instance, when the GPIO 23 is pressed, the video will rewind by 10 seconds.

3.7 Bash script

This is the final step of lab1. In this step, our goal is to write a bash script to assemble the python code we wrote before into this script. This allows us to control the mplayer after executing the script. To do this, we need to run the mplayer and also need to control the playing of the video. In this script, we first run the video_control.py python code in the background.

```
python /home/pi/lab1/video_control.py
```

The first command of the script should run in the background. It is because the second command will not be executed if the first command is run in the foreground. We chose to run the python code video_control first because if we run this python program after the mplayer command, the mplayer process will become an "orphan". It is because that after running the subprocess to quit mplayer, both the script and video_control.py will be terminated. While the subprocess is still running there to quit

mplayer. Therefore, we should run the video_control.py first in the background and then run mplayer in the foreground. The code is shown in figure 5 in the appendix.

Testing of the four functions “pause”, “fast forward”, “rewind”, and “quit” showed that all the functions were performed as expected.

Result :

Following the lab procedure, the video and audio control performed as expected. When we run the bash script, the python program video_control will run background and the video will start playing foreground. When a button is pressed, the video will respond to corresponding functions, including “pause”, “fast forward”, “rewind”, and “quit”.

Our team met the goals outlined in the lab description and completed each week’s checkout on time.

Conclusion

In lab 1, we completed most steps smoothly, and most tasks performed well as we expected.

During the lab, we encountered a problem that there was no sound when we ran the mplayer to play the video. In the beginning, we could only hear a very little sound. And we checked the audio settings and reset them according to the instructions. However, it was still not working. After that, we thought it may be the problem with the earphone. So, we changed to a new earphone but still couldn't work. Finally, we found that we muted the raspberry pi! Of course, there was no sound. When we turned on the audio, it finally worked.

In the week2, when we were writing the bash script, we put the two commands in the wrong order at the beginning. That is, we put the running mplayer command before running video_control.py command. That made the button didn't work because the video was running in the foreground. Under this order, the video_control.py would only be executed when the video ended. After noticing the error, we thought about several possibilities that could lead to this error. We first checked the path of fifo to make sure the command would be sent to the fifo and would be read by mplayer. However, we found there was no problem with the path of fifo. After that, we thought about the differences between running a process in the foreground and background. We realized that the process running in the background should be placed before the one running in the foreground. Then, we adjust the order of the two commands and buttons worked well.

We think the lab description is clear and organized, and we are able to follow each step during the lab. For suggestions, we think it will be nice if we can have some time left to wrap up at the end of each lab.

Appendix

```
# fifo_test.py Xinyun Guo xg284 Zhuoheng Li zl764 lab1 09-25-2019
import subprocess
while 1:
    # get the input from the keyboard
    cmd = raw_input ("Input the command")
    # print "OK" on console and print the output of the subprocess after sending pause command to fifo
    if cmd == "pause":
        print ("OK")
        my_cmd = 'echo "pause" > /home/pi/video_fifo'
        print subprocess.check_output(my_cmd, shell=True)
    # print the output of the subprocess after sending quit command to fifo
    elif cmd == "quit":
        print subprocess.check_output('echo "quit" > /home/pi/video_fifo', shell=True)
        break
```

Figure 1 Python code for fifo_test.py

```
#one_button.py Xinyun Guo xg284 & Zhuoheng Li zl764 lab1 09-25-2019
import RPi.GPIO as GPIO
import time

# set up the GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(22,GPIO.IN, pull_up_down=GPIO.PUD_UP)

while True:
    time.sleep(0.2)
    # when the GPIO22 is pressed, print
    if (not GPIO.input(22)) :
        print ("Button GPIO22 has been pressed")
```

Figure 2 Python code for one_button.py

```
#four_button.py Xinyun Guo xg284 & Zhuoheng Li z1764 lab1 09-25-2019
import time
import RPi.GPIO as GPIO

#set up all the four buttons
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(22,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(23,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(27,GPIO.IN,pull_up_down=GPIO.PUD_UP)

while True:
    time.sleep(0.2)
    if (not GPIO.input(17)):
        print("Button 17 has been pressed")
    elif(not GPIO.input(22)):
        print("Button 22 has been pressed")
    elif(not GPIO.input(23)):
        print("Button 23 has been pressed")
    #if GPIO27 is pressed, print and break out the loop
    elif(not GPIO.input(27)):
        print("Button 27 has been pressed")
        break
```

Figure 3 Python code for four_buttons.py

```
#video_control.py Xinyun Guo xg284 Zhuoheng Li z1764 lab1 09-25-2019
import RPi.GPIO as GPIO
import time
import subprocess

#set up the four GPIOs
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(22,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(23,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(27,GPIO.IN,pull_up_down=GPIO.PUD_UP)

while True:
    time.sleep(0.2)
    #pressing GPIO17 will pause the video
    if (not GPIO.input(17)):
        print subprocess.check_output('echo "pause" > /home/pi/video_fifo', shell=True)
    #pressing GPIO22 will fast forward 10s
    elif(not GPIO.input(22)):
        print subprocess.check_output('echo "seek 10 0" > /home/pi/video_fifo', shell=True)
    #pressing GPIO23 will rewind 10s
    elif(not GPIO.input(23)):
        print subprocess.check_output('echo "seek -10 0" > /home/pi/video_fifo', shell=True)
    #pressing GPIO27 will quit the mplayer
    elif(not GPIO.input(27)):
        print subprocess.check_output('echo "quit [0]" > /home/pi/video_fifo', shell=True)
        break
```

Figure 4 Python code for video_control.py


```
#start_video.sh Xinyun Guo xg284 Zhuoheng Li zl764 lab1 09-25-2019
#!/bin/bash
#shell script: run echo

#run the video_control.py in the background first
python /home/pi/lab1/video_control.py &

#run the mplayer in the foreground
sudo SDL_VIDEODRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -input file=/home/pi/video_fifo -vo sdl -framedrop /home/pi/bigbuckbunny320p.mp4
```

Figure 5 code for start_video.sh