ECE 5725: Lab 2 Report
Thursday Lab Session
10/16/2020

Jiaqi Sun (js3599) and Pranav Gupta (ppg22)

Introduction

Lab2 is an expansion of video control from Lab1 and also a practice experiment of designing PyGame programs. In the first part , we expanded the video control functions by adding more external buttons. Both the video control script with the rolling loop and callback function were modified for practice. The Linux “perf” utility was introduced to measure the performance of the scripts we had developed. The timeout function was added for the convenience of performance comparison between the two scripts. The third part was developing a PyGame Bounce Program. We first designed a program in which two balls that didn’t interface with each other moved on the screen and bounced back when they touched the edge of the screen. Then we expanded the python code to a ball collision program where the balls were no longer “transparent” to each other and bounced as they collided.

In the second week, we expanded the PyGame program by controlling it with the piTFT touch screen. The first job was to install and downgrade the wheezy files. After the files were successfully installed, we tested the coordinates with the code provided in class and started developing touch screen programs. A quit button program was first created where there was a “quit” button displayed on the screen and the program was ended by touch the button. We then expanded the function of the script by displaying the coordinates of each tap on the screen. We added another button and the ball collision animation to the third program. Hitting the “start” button on the screen would play the ball collision animation and tapping any location on the screen would display the coordinates of the tap. The last program was a two level PyGame program. The coordinates were displayed in the first level, and we could control speed and play of the ball animation in the second level.

Design and Testing

Video Control from Lab 1

In the first part of week 1, we expanded the function of video control by adding two more buttons on the piTFT. There were four buttons on the piTFT and they served functions of “Pause”, “Fast forward 10 sec”, “Rewind 10 sec” and “Quit” in the video control program. We added the two more buttons by connecting the physical buttons to the GPIO pins of the Rpi. The circuit of buttons connection is shown in fig 1. The GPIO pin connected to a 1k Ohm resistor, paralleling with a 10 k Ohm resistor connecting to the 3.3 Volt source. Without the 1k Ohm resistor, there would be infinite high current flowing through the GPIO pin and ending up burning the RPi. The selection of resistor was decided by maximum current allowed through the GPIO pin. There were 3.3 V on all the pins and GPIO pins could bear a maximum current of 50 mA. Therefore, 1k Ohm resistor could be connected to the GPIO pin.

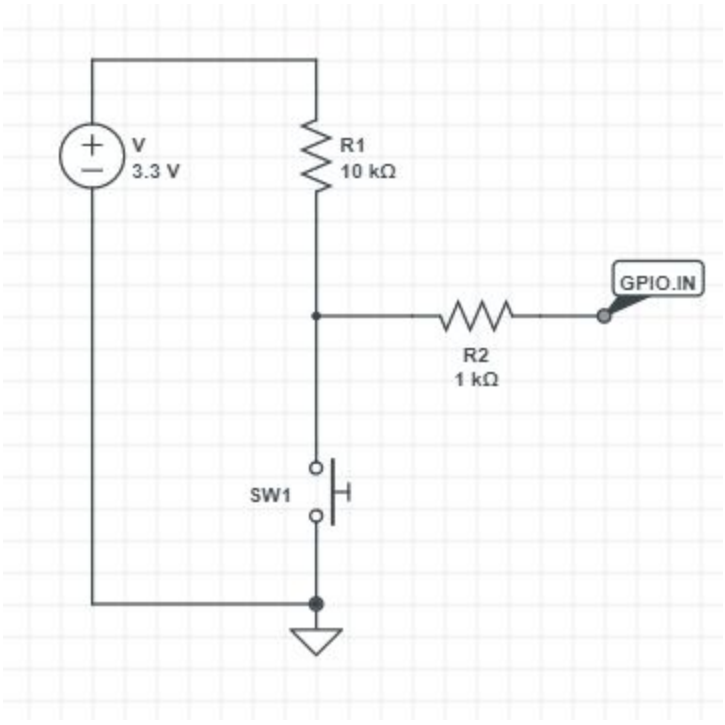


Fig 1: Connect push buttons to GPIO pins to control the video play

The two extra buttons served functions of “Fast forward 30 sec” and “Rewind 30 sec”. The GPIO pins we used were GPIO16 and GPIO26. The button connection was shown in fig 2.

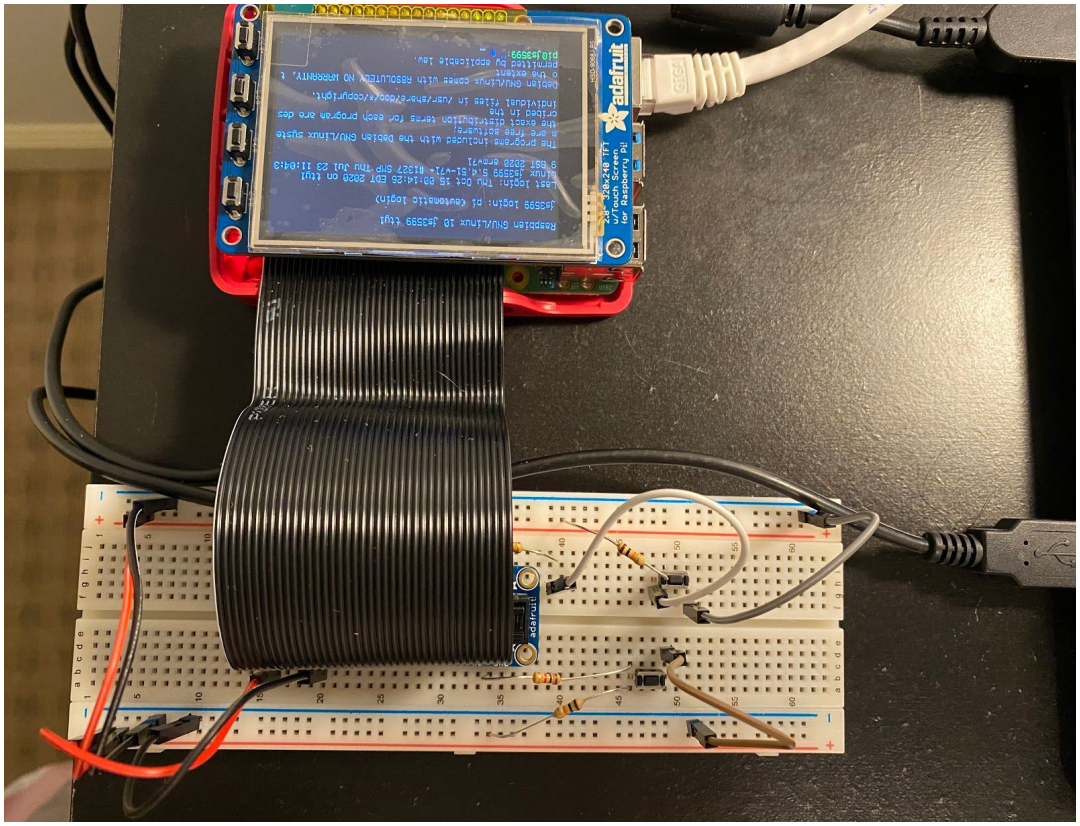


Fig 2: Circuit connection of the two extra buttons

In the video_control.py program, the two GPIO pins were set up as high and detected if they were low in the while loop. Once the GPIO pins were detected low, it means the buttons were pressed and the commands were sent to fifo test through subprocess. The video control callback program was modified as well. Two callback functions and two GPIO.add_event_detect commands were added in the scripts. Fig 3 and Fig 4 showed the simulation of more_video_control.py. The video fast forwarded for 30 seconds when button 26 was pressed and rewinded for 30 seconds when button 16 was pressed. The start_video and start_video bash script were also modified to use the scripts.

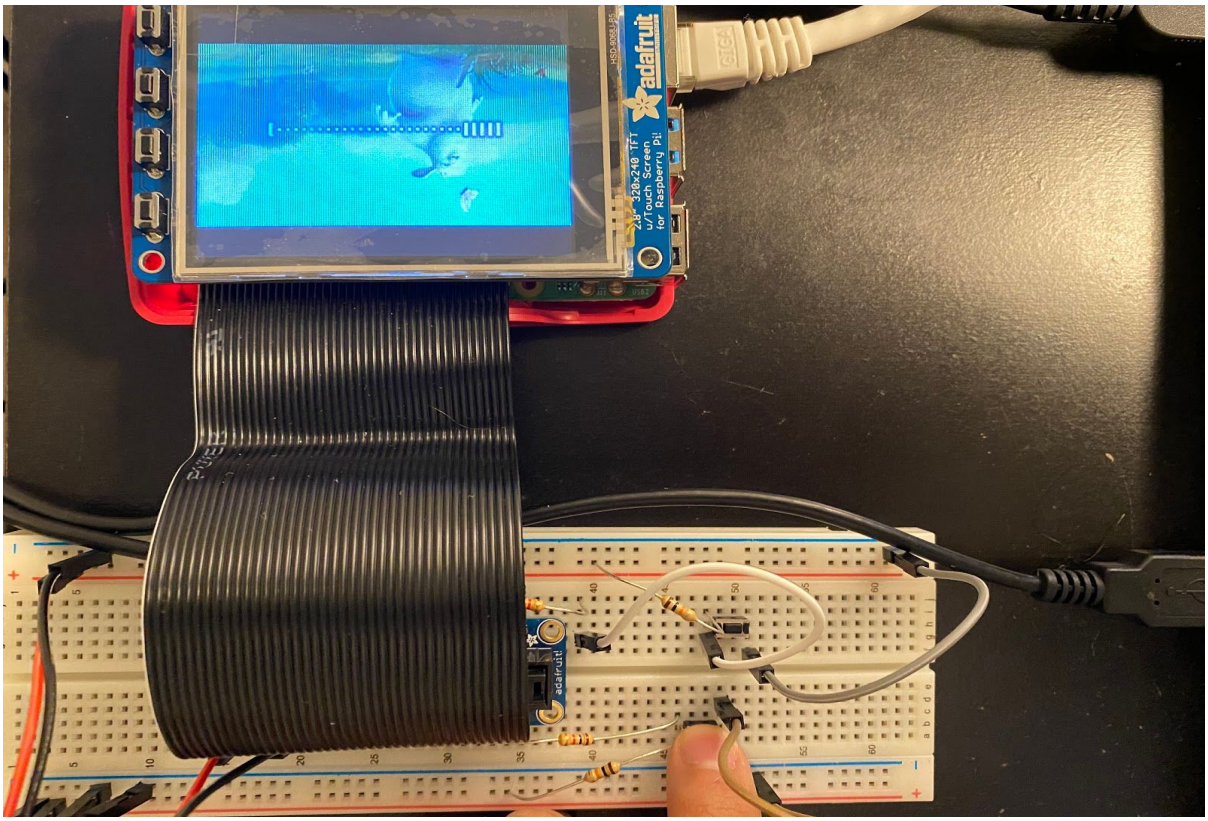


Fig 3: Fast Forwarded 30 seconds when button 26 was pressed

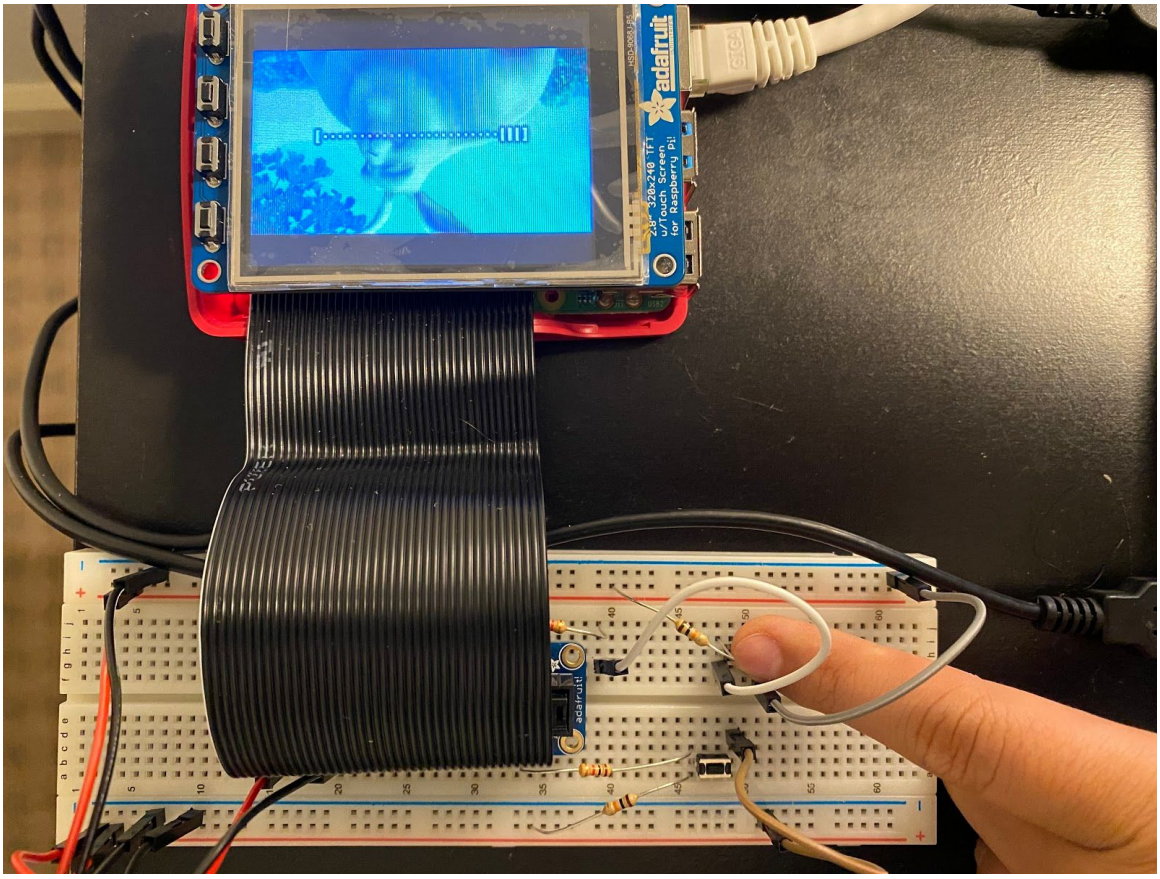


Fig 4: Rewinded 30 seconds when button 16 was pressed

In the next part, we moved to the performance measurement with “perf” utilities. The “perf” was installed with command “sudo apt-get install linux-perf-4.18”. Simply installing perf doesn’t work as apt install tries to locate a perf version 5.4 that doesn’t exist, because it tries to find a perf version with the same number as the Linux kernel. To measure the performance of a certain script, we could simply run the command “sudo perf_4.18 stat -e task-clock,context-switches,cpu-migrations,page-faults,cycles,instructions python program.py”. A few statistics were measured in the command. To fairly compare the performance of more_video_control.py and more_video_control_cb.py, both the programs have to run for a fixed time and quit. We could measure the time running in a rolling loop by using the time.time() function. We first measured the start time of the program and checked how long had passed with each iteration by code if time.time()-start>10. To set timeout with the callback program, we added a timeout argument to the wait_for_edge function. As long as the quit button wasn’t pressed, the wait_for_edge function would wait for a certain amount of time and then timed out. The performance measurement of more_video_control_perf.py and more_video_control_cb_perf.py running in 10 seconds is shown in fig 5 and 6.More performance measurements were done by decreasing the polling loop times. The polling loop times were decreased from 200 milliseconds to 20 milliseconds, 2 milliseconds, 200 microseconds, 20 microseconds and finally with no sleep. The performance measurements were shown in fig 7, 8, 9, 10, 11. All the performance statistics that were measured were summarized in table 1.

```
pi@js3599:~/lab2_files_f20 $ sudo perf_4.18 stat -e task-clock,context-switches
cpu-migrations,page-faults,cycles,instructions python more_video_control_perf.p
time = 10.0164871216

 Performance counter stats for 'python more_video_control_perf.py':

         53.710240      task-clock (msec)         #    0.005 CPUs utilized

                51      context-switches          #    0.950 K/sec

                 0      cpu-migrations            #    0.000 K/sec

               646      page-faults               #    0.012 M/sec

          69136860      cycles                    #    1.287 GHz

          47559210      instructions              #    0.69  insn per cycle


      10.066156328 seconds time elapsed

       0.033797000 seconds user
       0.022531000 seconds sys
```

Fig 5: Performance of more_video_control_perf.py in 10 seconds (sleep(0.2))

```
pi@js3599:~/lab2_files_f20 $ sudo perf_4.18 stat -e task-clock,context-switches
cpu-migrations,page-faults,cycles,instructions python more_video_control_cb_per
.py
Falling edge detected on port 27

Performance counter stats for 'python more_video_control_cb_perf.py':

         73.954223      task-clock (msec)         #    0.007 CPUs utilized

                11      context-switches          #    0.149 K/sec

                 3      cpu-migrations            #    0.041 K/sec

              1521      page-faults               #    0.021 M/sec

          86132062      cycles                    #    1.165 GHz

          54900886      instructions              #    0.64  insn per cycle


     10.090261772 seconds time elapsed

      0.037804000 seconds user
      0.039532000 seconds sys
```

Fig 6: Performance of more_video_control_cb_perf.py in 10 seconds

```
pi@js3599:~/lab2_files_f20 $ sudo perf_4.18 stat -e task-clock,context-switches,
cpu-migrations,page-faults,cycles,instructions python more_video_control_perf.py
time = 10.0179610252

 Performance counter stats for 'python more_video_control_perf.py':

       139.579402      task-clock (msec)         #    0.014 CPUs utilized

             642      context-switches          #    0.005 M/sec

               0      cpu-migrations            #    0.000 K/sec

             689      page-faults               #    0.005 M/sec

       108749916      cycles                    #    0.779 GHz

        80008517      instructions              #    0.74  insn per cycle


     10.229177717 seconds time elapsed

      0.062822000 seconds user
      0.087951000 seconds sys
```

Fig 7: Performance of more_video_control_perf.py in 10 seconds (sleep(0.02))

```
pi@js3599:~/lab2_files_f20 $ sudo perf_4.18 stat -e task-clock,context-switches
cpu-migrations,page-faults,cycles,instructions python more_video_control_perf.p
time = 10.0019550323

 Performance counter stats for 'python more_video_control_perf.py':

       406.025680      task-clock (msec)         #    0.040 CPUs utilized

             4819      context-switches          #    0.012 M/sec

               14      cpu-migrations            #    0.034 K/sec

              646      page-faults               #    0.002 M/sec

       333224407      cycles                    #    0.821 GHz

       264995455      instructions              #    0.80  insn per cycle


     10.051409809 seconds time elapsed

      0.313816000 seconds user
      0.151096000 seconds sys
```

Fig 8: Performance of more_video_control_perf.py in 10 seconds (sleep(0.002))

```
p1@js3599:~/lab2_files_f20 $ sudo perf_4.18 stat -e task-clock,context-switches
cpu-migrations,page-faults,cycles,instructions python more_video_control_perf.p
time = 10.0001649857

 Performance counter stats for 'python more_video_control_perf.py':

       2131.343693      task-clock (msec)         #    0.212 CPUs utilized

             30315      context-switches          #    0.014 M/sec

                 0      cpu-migrations            #    0.000 K/sec

               646      page-faults               #    0.303 K/sec

        1623660772      cycles                    #    0.762 GHz

        1446579887      instructions              #    0.89  insn per cycle


      10.071492402 seconds time elapsed

       0.168155000 seconds user
       2.323605000 seconds sys
```

Fig 9: Performance of more_video_control_perf.py in 10 seconds (sleep(0.0002))

```
p1@js3599:~/lab2_files_f20 $ sudo perf_4.18 stat -e task-clock,context-switches
cpu-migrations,page-faults,cycles,instructions python more_video_control_perf.p
time = 10.0000340939

 Performance counter stats for 'python more_video_control_perf.py':

       3327.772033      task-clock (msec)         #    0.331 CPUs utilized

             88179      context-switches          #    0.026 M/sec

                 0      cpu-migrations            #    0.000 K/sec

               647      page-faults               #    0.194 K/sec

        4558505939      cycles                    #    1.370 GHz

        4115265870      instructions              #    0.90  insn per cycle


      10.062380235 seconds time elapsed

       1.464092000 seconds user
       2.426211000 seconds sys
```

Fig 10: Performance of more_video_control_perf.py in 10 seconds (sleep(0.00002))

```
pi@js3599:~/lab2_files_f20 $ sudo perf_4.18 stat -e task-clock,context-switches
cpu-migrations,page-faults,cycles,instructions python more_video_control_perf.p
time = 10.0000100136

 Performance counter stats for 'python more_video_control_perf.py':

     10059.487962      task-clock (msec)         #    1.000 CPUs utilized

              23      context-switches          #    0.002 K/sec

               0      cpu-migrations            #    0.000 K/sec

             641      page-faults               #    0.064 K/sec

      15045319432      cycles                    #    1.496 GHz

      18222954709      instructions              #    1.21  insn per cycle


     10.061915346 seconds time elapsed

      9.912369000 seconds user
      0.150035000 seconds sys
```

Fig 11: Performance of more_video_control_perf.py in 10 seconds (No sleep!)

sleep	Task-clock	content-switches	cpu-migrations	page-faults	cycles	instructions
more_video_control_perf.py						
0.2	53.71	51	0	646	6.91e7	4.76e7
0.02	93.49	498	0	645	9.90e7	6.83e7
0.002	406.03	4819	14	646	3.33e8	2.65e8
0.0002	2132.34	30315	0	646	1.62e9	1.45e9
0.00002	3327.77	88179	0	647	4.56e9	4.12e9
No	10059.48	23	0	641	1.50e10	1.82e10
more_video_control_cb_perf.py						
No	73	11	3	1521	8.61e7	5.49e7

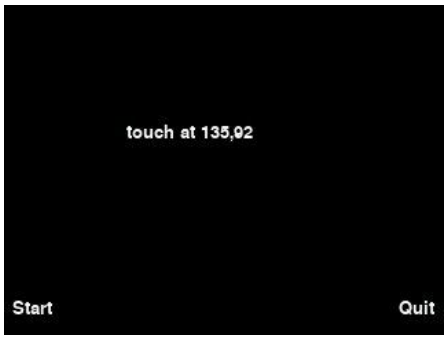
Table 1: Summary of Performance statistics

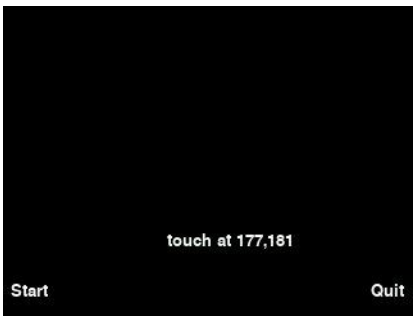
We could see that as sleep time decreased, the program execution time went higher, so were the times that the linux process scheduler switched, the number of fetch-decode-execute cycles and instructions were run for the process.The cpu-migration time and page faults were not influenced too much. This may be because as the sleep time decreased more cycles of the

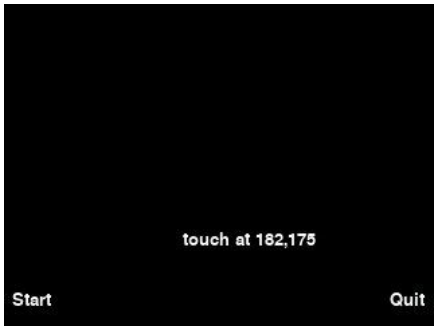
polling loop were running and caused the rise of those performance statistics. The page-faults of the more_video_control_cb_perf.py was about one time higher than that of more_video_control_perf.py with 200 milliseconds sleep time which meant more virtual memory was copied to physical memory. The callback routine involved less context switches.

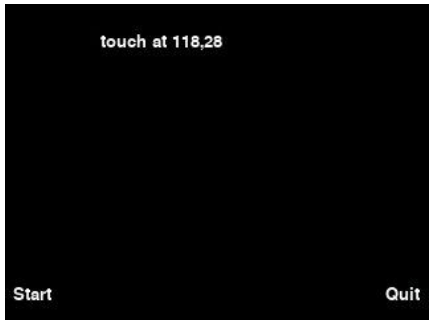
The development of the PyGame programs required the downgrade of the distro from Buster to Wheezy. Due to the open source nature of the Raspbian repo, we guess the synchrony between the touchscreen drivers and the Linux kernel was broken somehow. But then the downgrade to Wheezy is tricky as well. We enabled Wheezy packages via /etc/apt/sources.list.d/wheezy.list, set Wheezy as the default package source via /etc/apt/apt.conf.d/10defaultRelease, and most importantly set the priority of libsdl (SDL driver library for the touchscreen) from wheezy higher than the one from Buster via /etc/apt/preferences.d/libsdl . We were careful so as to not copy-paste and accidentally include bad characters. Still the touchscreen drivers gave nonsensical screen coordinates, either displaying completely different coordinates on pressing the same location again and again, or giving the maxed out answer (319,239) regardless of where we pressed on the screen. Professor Skovira suspects something fishy with the Lab 1 commands we ran, as it is unlikely otherwise that both of us (Jiaqi and Pranav) faced the same problem on our respective Pis. Ultimately we used a backup image sent to us by Professor Skovira with a functioning Wheezy downgrade and then we were able to get correct coordinates from the touchscreen. After the downgrade to Wheezy the version of the sdl driver also downgraded, as per the output of dpkg -l.

We then implemented the required touch button programs. The screenshots of screen_coordinates.py were saved using the pygame.image.save function, and have been attached here:

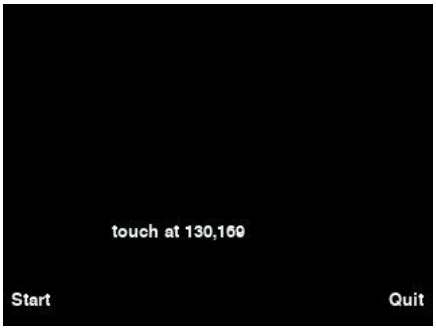


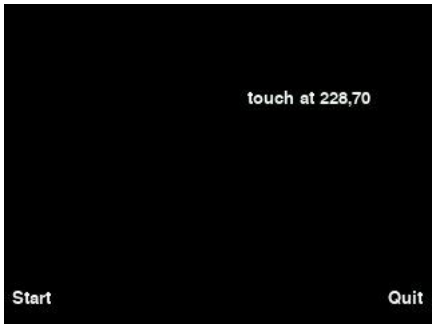



touch at 182,175
Start
Quit


touch at 118,28
Start
Quit


touch at 279,15
Start
Quit

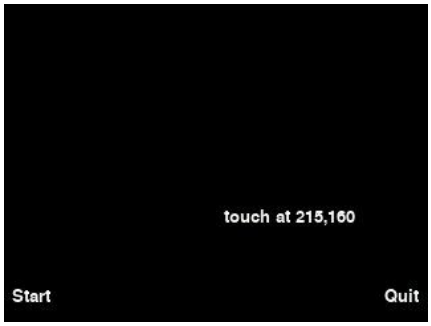

touch at 130,169
Start
Quit


touch at 228,70
Start
Quit

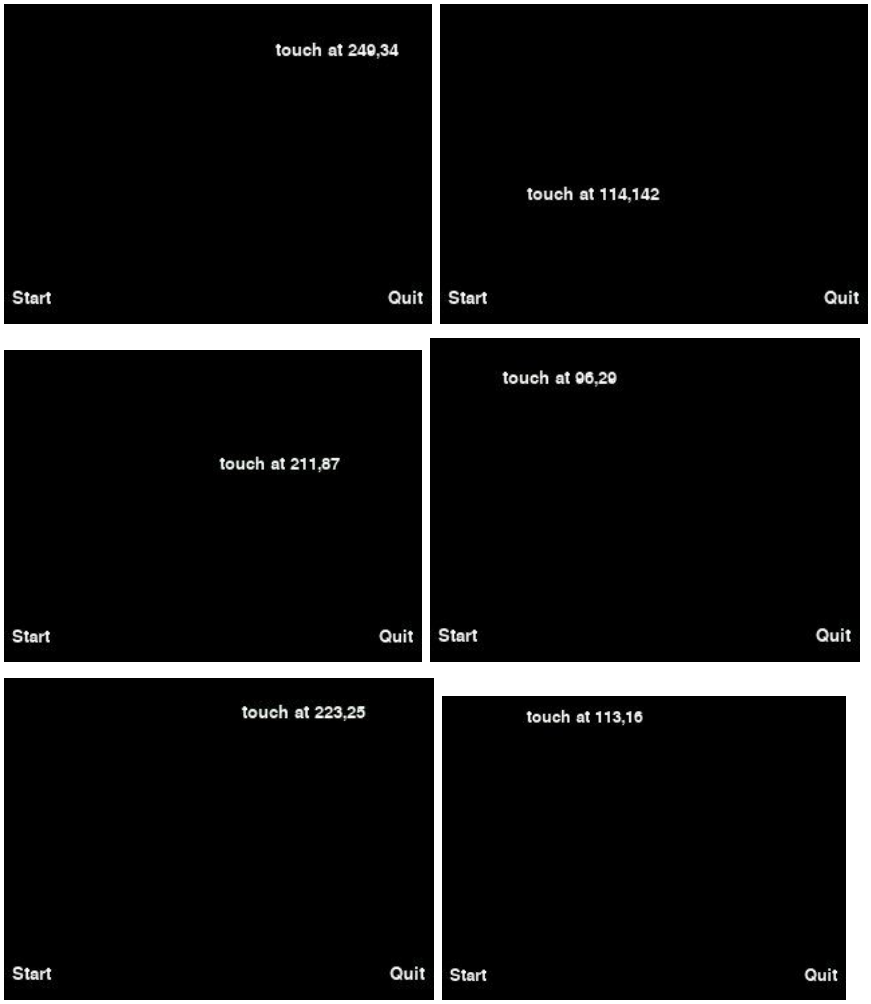

touch at 58,74
Start
Quit


ouch at 39,149
Start
Quit


touch at 235,46
Start
Quit


touch at 215,160
Start
Quit

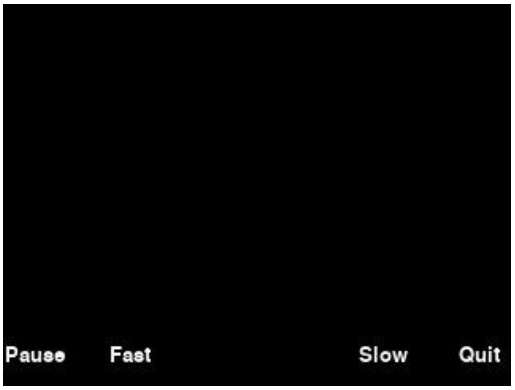

touch at 66,50
Start
Quit



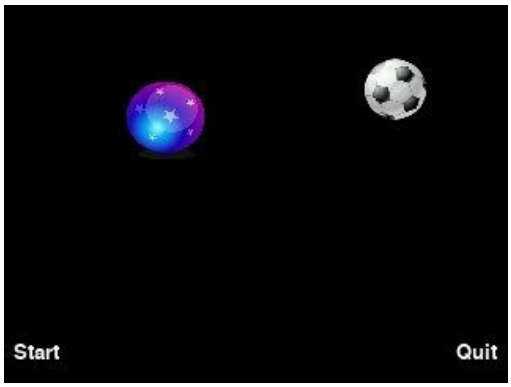
We implemented 4 programs in this part of the lab: namely **quit_button.py**, **screen_coordinates.py**, **two_button.py** running with two_collide.py functions, and **control_two_collide.py** running with two_collide.py functions.

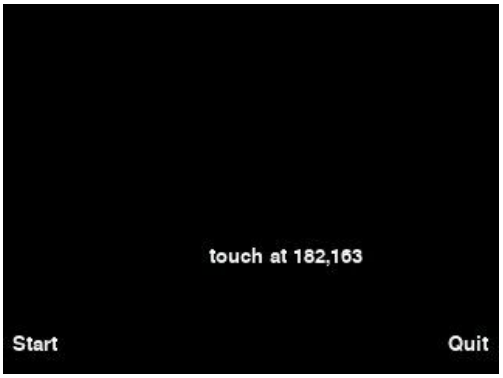
Each program had a GPIO bail out button (button #27 on the piTFT screen was chosen for this purpose) and a timeout of 30 seconds.

Results



Screenshots from control_two_collide.py





Screenshots from two_button.py

Conclusion

In this lab we learned how to use pygame for displaying graphics and videos on the piTFT. We also learned how to use the touchscreen for controlling the piTFT. We learned about callbacks, polling loops and events in pygame. The Wheezy downgrade was exasperating, and we were unable to track what file in the Linux kernel was messed up. We are curious to know the root cause behind the bizarre behavior of the touchscreen. Moreover, we observed that the /dev/input/touchscreen doesn't show up despite running the fix_touschscreen Shell script to activate the smpe_ts module and waiting for a while. In such cases, we resorted to simply rebooting the Pi and by some stroke of luck the touchscreen would start functioning again.

References

[1] Stack Overflow,
https://stackoverflow.com/questions/6087484/how-to-capture-pygame-screen