ECE5725 Lab 2

Outline

In Lab2, we will go through the following steps:

- Add external buttons to the RPi setup to expand the control functions of video_control.py from Lab1
- Compare performance of video_test.py in lab 1 and modified versions of this program.
- Develop general methods for checking performance of applications on the Pi with an eye towards performance tuning.
- Develop some test programs using the PyGame library and expand these programs into control panels for animated videos.

Lab safety

We have all been in lab before and handled electronic components. Please apply all the cautions that you have learned including the items below:
Integrated electronics are REALLY susceptible to static discharge.

- Avoid walking around while holding bare components
- Confine assembly to the grounded mats on the lab bench
- Make sure you ground yourself by staying in contact with the mat.

Personal safety

- Safety goggles are REQUIRED for soldering

Experimental Safety

- If something on your board is
- Really hot
- Smoking
- Just doesn't look right
- Immediately unplug power
- Train yourself so that this is your first reaction – know where the power switch/cutoff is for your experiment.

Experimental assembly

- Before adding any hardware to the system, power should be OFF
- Before powering up the equipment, please have staff check your added circuit

video_control.py application from Lab1

A few more buttons

In the final step of Lab 1 (step 6), you created the video_control.py to control buttons on the piTFT. This section includes some modifications to this program.
As a first step, make sure the original program, as defined in Lab 1, is working as designed. A short description includes the button functions:
On the piTFT:

- Pause
- Fast-forward 10 seconds
- Rewind 10 seconds
- Quit

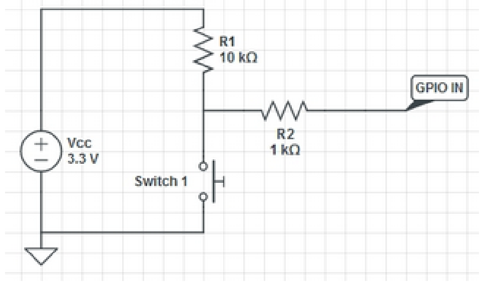
Reference step 5 and 6 in Lab 1, week 2 for additional details.

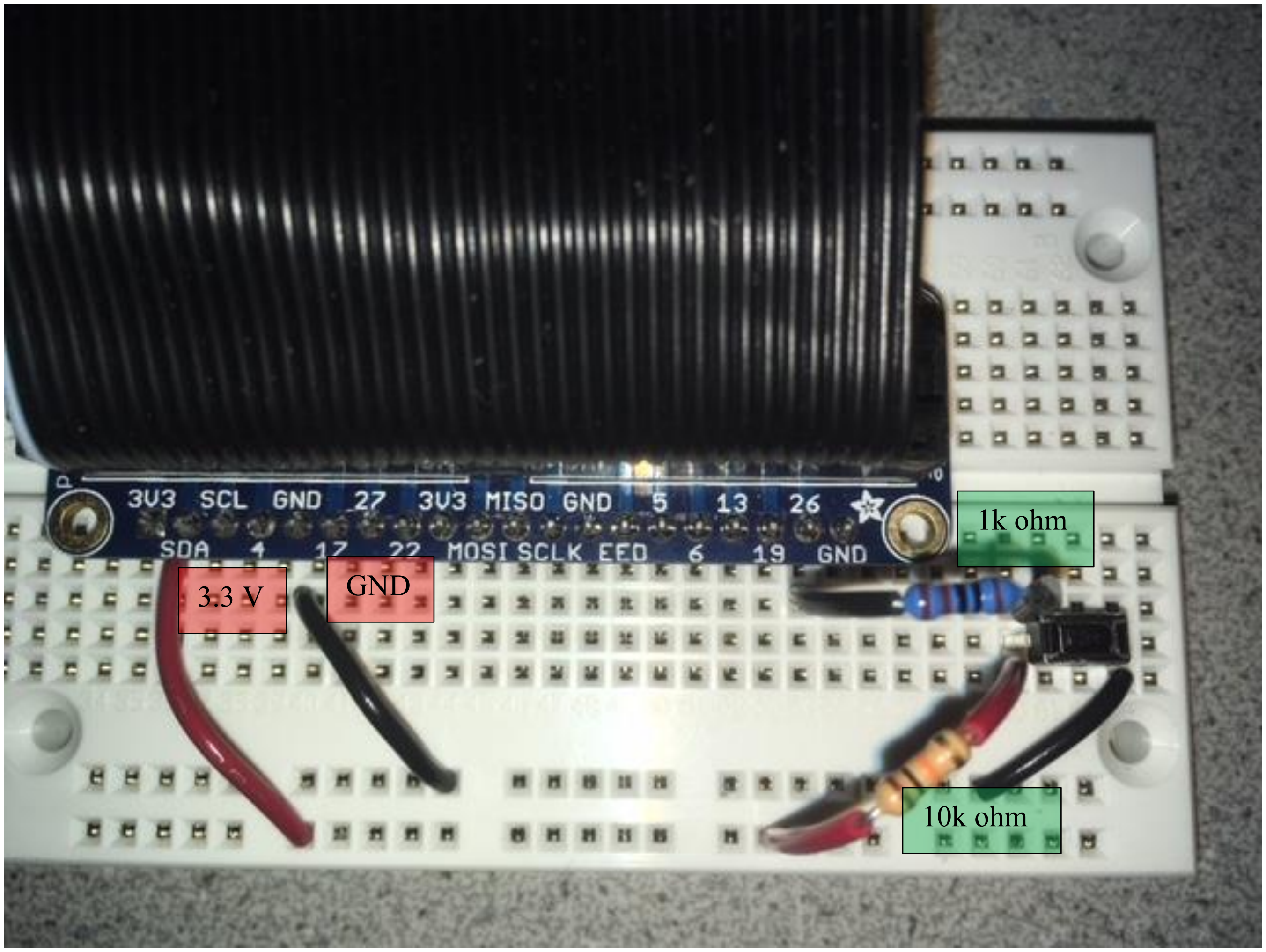
Using the breakout cable ('Pi-cobbler'), add two additional buttons to the RPi setup used for playing the video on the PiTFT.
Consider which GPIO pins to use, based on what free pins are available

- Once the cable is connected to the protoboard, check that the polarity of the plug is correct. Using a voltmeter, check that the +5 pins and several of the +3.3 pins are in the correct locations (according to the indications on the cable header). If not, or if you are unsure, please check in with the TA.
- Use the safe development techniques discussed in class, so as not to 'cook' the GPIO pins if something goes wrong.
- Once the buttons are wired correctly;

  STOP and please check with the TA to ensure correct connections and GPIO selection before turning on the raspberry pi.

A photo of a setup connected correctly using the following circuit:



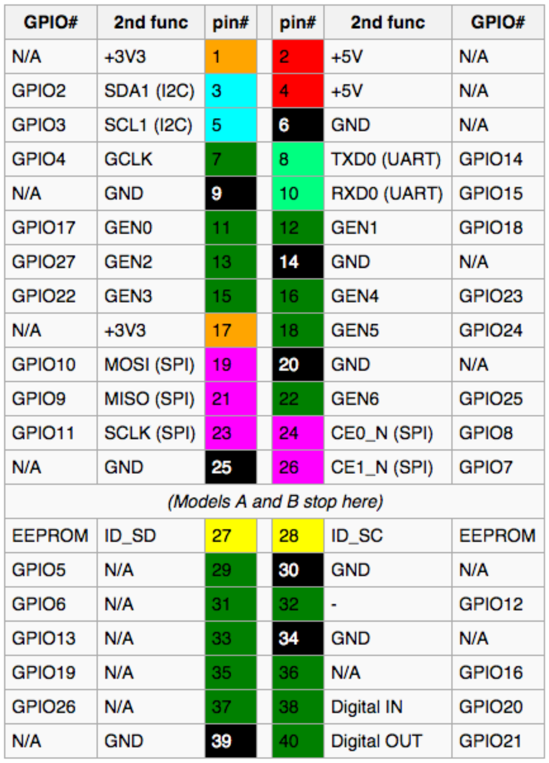


Notes:

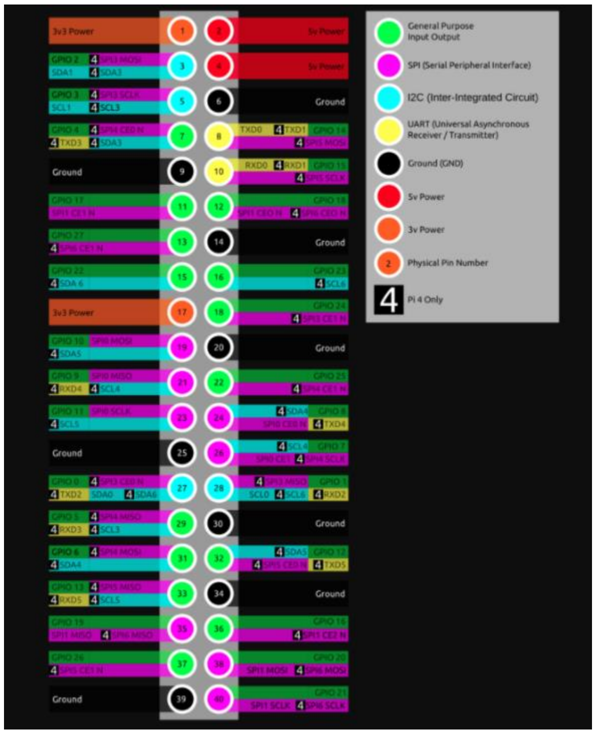
- The white stripe on the piCobbler breakout cable is on the same side as the piTFT buttons
- On the protoboard, you can read the pin assignments (starting from the left) 3.3, SDA, SCL, 4, GND, etc….
- The photo is a bit distorted on the left edge: The red wire is connected to 3.3 V (the pin labeled 3V3), **NOT SDA**

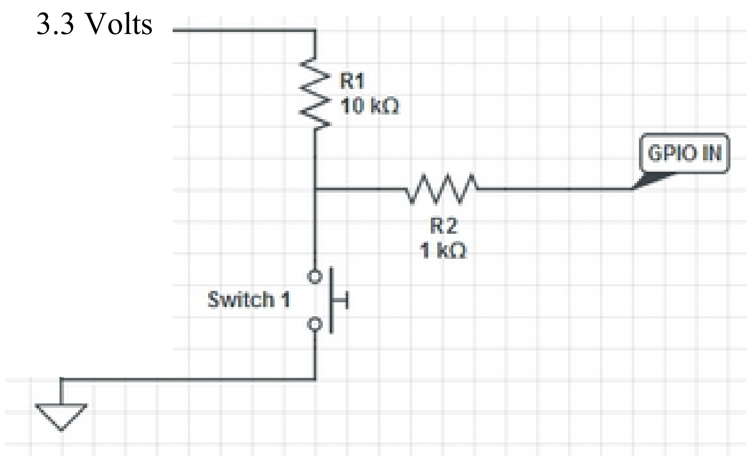
- Note that the numbering on the piCobbler corresponds to BCM numbering (for example pin '26' on the piCobbler is GPIO26)
- The above example has a single button wired into GPIO26.
- Note wire insulation used on bare resistor leads to prevent shorts (please see the'Lab2 parts' video on Canvas for a demonstration)

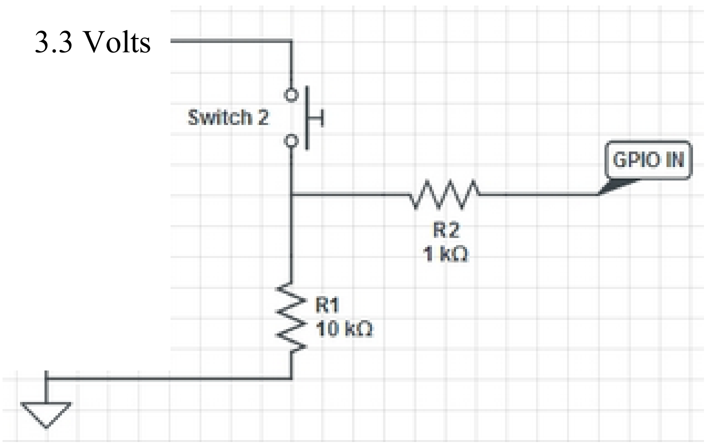
Some references:



GPIO#	2nd func	pin#	pin#	2nd func	GPIO#
N/A	+3V3	1	2	+5V	N/A
GPIO2	SDA1 (I2C)	3	4	+5V	N/A
GPIO3	SCL1 (I2C)	5	6	GND	N/A
GPIO4	GCLK	7	8	TXD0 (UART)	GPIO14
N/A	GND	9	10	RXD0 (UART)	GPIO15
GPIO17	GEN0	11	12	GEN1	GPIO18
GPIO27	GEN2	13	14	GND	N/A
GPIO22	GEN3	15	16	GEN4	GPIO23
N/A	+3V3	17	18	GEN5	GPIO24
GPIO10	MOSI (SPI)	19	20	GND	N/A
GPIO9	MISO (SPI)	21	22	GEN6	GPIO25
GPIO11	SCLK (SPI)	23	24	CE0_N (SPI)	GPIO8
N/A	GND	25	26	CE1_N (SPI)	GPIO7
(Models A and B stop here)					
EEPROM	ID_SD	27	28	ID_SC	EEPROM
GPIO5	N/A	29	30	GND	N/A
GPIO6	N/A	31	32	-	GPIO12
GPIO13	N/A	33	34	GND	N/A
GPIO19	N/A	35	36	N/A	GPIO16
GPIO26	N/A	37	38	Digital IN	GPIO20
N/A	GND	39	40	Digital OUT	GPIO21


General Purpose Input Output
SPI (Serial Peripheral Interface)
I2C (Inter-Integrated Circuit)
UART (Universal Asynchronous Receiver / Transmitter)
Ground (GND)
5v Power
3v Power
Physical Pin Number
4
Pi 4 Only





- **Stop and check your wiring with a TA.**
- Once checked, power up and extend four_buttons.py to create ‘six_buttons.py’ to make sure you can correctly read the existing four, and two new buttons
- Extend ‘video_control.py’ to create ‘more_video_control.py’ to add two new functions to video control; fast forward 30 seconds and rewind 30 seconds
- Modify the ‘start_video’ bash script to use the new code, more_video_control.py.

Step2: interrupt callbacks

Modification of video_control.py

Once more_video_control.py is running, copy the verified file into a new file, more_video_control_cb.py. In this code, modify the button presses to use threaded callback interrupt routines for button presses. Note:

- Most (all) of the buttons can be modified to use callback routines
- The polling loop used in more_video_control.py will require alteration
- The quit function may be a special case.
- For each button, you will need to:
    - Setup a callback routine
    - Define an event that accesses the callback routine

Once this routine has been redesigned, verify that it operates correctly in a bash script, similar to the operation of start_video from lab1. Name this new bash script start_video_cb. Confirm that the operation of all buttons in start_video_cb is identical to the function in the original start_video script.

Backup your python files on server or to your own laptop using scp at this point.

Performance measurement with ‘perf’ utilities

This section explores the Linux ‘perf’ utility to measure performance of applications on the R-Pi. We will be looking into the performance difference between polling loop and interrupt versions of programs implemented in Python during lab.

Step1: Install Perf with the following commands:

Try the command:

```
perf --help
```

And it should **fail**! The message will tell you that your system is missing the correct version of perf. Example:

```
pi@RPI-jfs9:~ $ perf --help
/usr/bin/perf: line 13: exec: perf_6.1: not found
E: linux-perf-6.1 is not installed.
```

We are also going to install the latest version of perf by running:

```
sudo apt-get install linux-perf
```

perf will take a few minutes to install. Here is a snip of the installation:

```
pi@RPI-jfs9:~ $ sudo apt-get install linux-perf
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  libfuse2
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  libopencsd0 linux-perf-5.10
Suggested packages:
  linux-doc-5.10
The following NEW packages will be installed:
  libopencsd0 linux-perf linux-perf-5.10
0 upgraded, 3 newly installed, 0 to remove and 82 not upgraded.
Need to get 2,237 kB of archives.
After this operation, 5,800 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://deb.debian.org/debian bullseye/main arm64 libopencsd0 arm64 0.14.4-1 [165 kB]
Get:2 http://deb.debian.org/debian bullseye/main arm64 linux-perf-5.10 arm64 5.10.223-1 [2,071 kB]
Get:3 http://deb.debian.org/debian bullseye/main arm64 linux-perf arm64 5.10.223-1 [1,124 B]
Fetched 2,237 kB in 1s (2,255 kB/s)
Selecting previously unselected package libopencsd0:arm64.
(Reading database ... 99091 files and directories currently installed.)
Preparing to unpack .../libopencsd0_0.14.4-1_arm64.deb ...
Unpacking libopencsd0:arm64 (0.14.4-1) ...
Selecting previously unselected package linux-perf-5.10.
Preparing to unpack .../linux-perf-5.10_5.10.223-1_arm64.deb ...
Unpacking linux-perf-5.10 (5.10.223-1) ...
Selecting previously unselected package linux-perf.
Preparing to unpack .../linux-perf_5.10.223-1_arm64.deb ...
Unpacking linux-perf (5.10.223-1) ...
Setting up libopencsd0:arm64 (0.14.4-1) ...
Setting up linux-perf-5.10 (5.10.223-1) ...
Setting up linux-perf (5.10.223-1) ...
Processing triggers for man-db (2.9.4-2) ...
Processing triggers for libc-bin (2.31-13+rpt2+rpi1+deb11u10) ...
```

Note the bold sections in the above output. These messages indicate that the 64 bit version of perf_5.10 was installed on the system.

Step2:

Test that the latest version of perf is operating as designed by running:

```
pi@RPI-jfs9:~ $ perf --help
/usr/bin/perf: line 13: exec: perf_6.1: not found
E: linux-perf-6.1 is not installed.
```

As suspected, the 6.1 version of perf was not installed. Instead, check:

```
pi@RPI-jfs9:~ $ perf_5.10 –help
pi@RPI-jfs9:~ $ perf_5.10 list
pi@RPI-jfs9:~ $ perf_5.10 --version
perf version 5.10.223
```

Notes:

1. Perf is able to track a number of different hardware and software events. Take a look at the output of 'list', above, to get an idea of these events perf is able to track.
2. Keep in mind that not all of these measurements are available on all platforms.

Step3: Create a python file named calibrate_v1.py containing the following statements:

```
import time
time.sleep(0.2)  # sleep
```

Step4: Run the following:

```
sudo perf_5.10 stat -e task-clock,context-switches,cpu-migrations,page-faults python calibrate_v1.py
```

This will show the following performance statistics:

- Task-clock: program execution time in milliseconds
- Context-switches: how many times the Linux process scheduler switched control between running processes
- cpu-migration: how many times process was moved to a different cpu (or core)
- page-faults: How many times a part of the processes virtual memory was copied to physical memory

This test gives a baseline set of statistics for a simple python code. Record the results.

Performance measurement of video_control.py

This section includes a series of tests to begin to characterize performance of the polling and interrupt versions of the video_control routines. Plan to record results from all runs for comparison.

In order to fairly compare the polling and interrupt versions of video_control, consider how you might need to modify the codes to perform for a fixed amount of time; that is, all codes in the experiment should run for a fixed time of 10 seconds (for example), then quit. The idea is to determine how much overhead may be present in the programs associated with button control; not the functions called by the button presses but, rather, the logic used to establish the control for the buttons.

Step1: Modify more_video_control.py to introduce fixed timing prior to the perf run. Rename this function 'more_video_control_perf.py'. Run the perf tools for more_video_control_perf.py , using a perf call that records default statistics:

```
sudo perf_5.10 stat python more_video_control_perf.py
```

In the polling loop, make sure to start with a 'sleep' value of 200 milliseconds (time.sleep(0.2)).

Step2: Modify more_video_control_cb.py to introduce a fixed run-time prior to running the perf tool. Once modified, run the perf tools, for more_video_control_cb_perf.py

Step3: make successive runs of more_video_control_perf.py, changing polling loop times to:

- 20 milliseconds
- 2 milliseconds
- 200 microseconds
- 20 microseconds
- with no sleep statement at all

Step4: Note changes to the perf measurements and deduce impacts of changes and compare the results between successive runs of more_video_control_perf.py and with more_video_control_cb_perf.py. Include discussion in the Lab report.

PyGame: Bounce Program

Notes for bounce programs:

- For easier control while debugging all piTFT programs:
  - **Implement a physical 'bail out' button.** This button may be one of the piTFT buttons or an external button. Hitting this button should end the program.
  - **Also implement a code time-out that will end the program after a given time interval.** For example, when you are first starting to develop the code, you may want to set this timeout to 30 seconds. This time-out can be lengthened, or eliminated, as your code stabilizes.
  - These methods will prevent excessive power off/power on restarts of a hung RPi system.
- Remember, from the lecture examples, that you'll need to consider the following environment variables when running on the piTFT:

```
import os
os.putenv('SDL_VIDEODRIVER', 'fbcon')
os.putenv('SDL_FBDEV', '/dev/fb1')
```

1. On your Raspberry Pi, implement the 'Bounce' code in python using pygame (you can find pygame documentation in the Canvas 'References' section). bounce.py should be run on both the monitor and the piTFT.
2. Extend this code to include 2 balls, two_bounce.py, where each ball moves on the screen at a different speed. Note that the two bouncing balls are designed to be 'transparent' to one another. That is, although a ball bounces off the 'walls' at the edge of the screen, balls may pass over each other when they intersect. Design this code to run on both the monitor and the piTFT.

3. Expand two_bounce.py to create two_collide.py, so that the balls alter their trajectories as they collide with one another.
   - o There are many techniques for collision of pygame objects. Have a look at the Pygame reference document for possible calls to help with detecting collisions (have a look at https://tinyurl.com/ybvmkrky)
   - o For physically correct 2-dimensional collisions, you can also refer to the following references:
     - ▪ https://scipython.com/blog/two-dimensional-collisions/
     - ▪ https://en.wikipedia.org/wiki/Elastic_collision
     - ▪ Note that you can also check the 'Dynamics' section in: http://people.ece.cornell.edu/land/courses/ece4760/labs/f2016/lab3_particle_beam.html

For another suggestion on how to alter velocity after collisions. [1]

4. Implement a version of two_collide.py that runs on both the monitor and the piTFT. Design this code to **exit when one of the piTFT buttons is hit (see note above!)**

Example of a two_bounce.py screen:

Demonstrate your work to the TA; note that **all demonstrations should be on the piTFT**. Demonstrate all Codes to the TA including:

- six_buttons.py (note; OK to run this code on the monitor)
- more_video_control.py and modified start_video script
- more_video_control_cb.py and start_video_cb bash script
- demonstrate perf runs with modified and more_video_control_perf.py more_video_control_cb_perf.py
- bounce.py, two_bounce.py and two_collide.py

Backup your SD card – Important as next section has some detailed changes.

Also, in anticipation of possible changes during Lab2 week2, please plan to save all of your lab code and data. One method would be to place all code/data in a respective Lab1 or Lab2 directory and then use the scp command to place these directories on the ECE5725 server. Alternatively, you can save the Lab directories on USB installed memory.

Week 2:

piTFT touch control

If you haven't done it yet, backup your original SD card to preserve the system from Lab 2 week 1 and Lab 1, weeks 1 and 2.

To move on to touch control for our system, we will be moving to a new 'driver' for the capacitive touch piTFT. First, an outline of the changes. Next, a step by step guide to update your system.

The outline of the upcoming detailed steps:

- Check evdev installation
- Download files for the Capacitive piTFT driver
- Configure the system and test the driver and touch capability

That's the plan! Let's go through these steps in detail.

Step 1: Check evdev installation

The evdev application is used to collect touch events from the piTFT. In past semesters evdev was pre-installed in the OS. To check for this installation, run:

```
sudo pip3 install evdev
```

The leading 'sudo'will install evdev with correct permissions to run the upcoming applications.

Here is an example evdev install:

```
pi@RPI-jfs9:~/Lab2/ $ sudo pip3 install evdev
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting evdev
  Downloading evdev-1.7.1.tar.gz (30 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
Building wheels for collected packages: evdev
  Building wheel for evdev (PEP 517) ... done
  Created wheel for evdev: filename=evdev-1.7.1-cp39-cp39-linux_aarch64.whl size=91042
sha256=c47b203d52c5c3c996240f575341fe4fd954800302f5108b80bbd89069def32e
  Stored in directory:
/root/.cache/pip/wheels/c8/58/5c/733f9fed824427c7f7c846c6d1b3949a6a85e7ee3b42f52720
Successfully built evdev
Installing collected packages: evdev
Successfully installed evdev-1.7.1
```

Step 2: Download files for the Capacitive piTFT driver

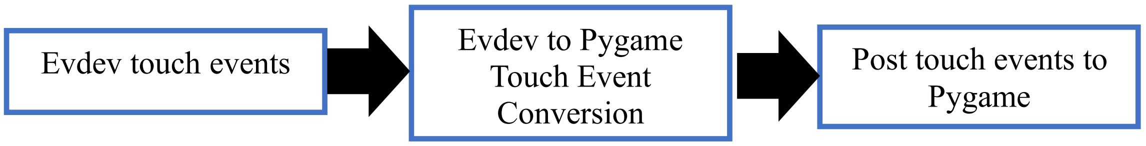
To install the codes necessary to drive the touchscreen run the following:

```
        cd /home/pi/lab2  # change to your lab2 directory (whatever you call it!)
        git clone https://github.com/n4archive/pitft_touchscreen
        git clone https://github.com/n4archive/pigame
        git clone https://github.com/nift4/Raspberry-Pi-Testing
```

These commands will install the git repos in your lab 2 directory. A quick overview:

- 'pitft_touchscreen': collects touch events from evdev
- 'pigame': converts evdev touch events to pygame touch events
- 'Raspberry-Pi-Testing': Provides codes to check the installation of the system.

The new idea for grabbing touch events is illustrated in the figure from Lecture:


Evdev touch events
Evdev to Pygame Touch Event Conversion
Post touch events to Pygame

Step 3: Configure the system and test the driver and touch capability

To get this running on the test system, I copied the following python codes from the respective git repos to the 'Raspberry-Pi-Testing' directory:

From 'pitft_touchscreen', copy the file pitft_touchscreen.py
From 'pigame', copy the file pigame.py

Important Note: You will notice in the sample python code an 'import pigame' is included. This will include the pigame.py file which requires the pitft_touchscreen.py file. **For any code you create using touch in the pitft, these 2 files should be copied into the directory.**

As an example for the sdl.py test application, here are the contents of the Raspberry-Pi_Testing directory once I copied the two files.

```
pi@RPI-jfs9:~/Lab2/Raspberry-Pi-Testing $ pwd
/home/pi/Lab2/Raspberry-Pi-Testing
pi@RPI-jfs9:~/Lab2/Raspberry-Pi-Testing $ ls -l
total 40
-rw-r--r-- 1 pi pi  193 Aug 13 15:54 detectbtn.py
-rw-r--r-- 1 pi pi  230 Aug 13 15:54 menu.sh
-rw-r--r-- 1 pi pi 5955 Aug 13 15:54 pigame.py
-rw-r--r-- 1 pi pi 4259 Aug 13 15:54 pitft_touchscreen.py
-rw-r--r-- 1 pi pi  715 Aug 13 15:54 README.md
-rw-r--r-- 1 pi pi 2375 Aug 13 15:54 sdlkit.py
-rw-r--r-- 1 pi pi 1732 Aug 13 15:54 sdl.py
-rw-r--r-- 1 pi pi  401 Aug 13 15:54 test.py
```

Before you test the 'sdl' application, make sure to download the /home/Lab/lab2_files_s25 from the class server. From this directory, copy sdl_v2.py into the Raspberry-Pi-Testing directory.

Also, before you run sdl_v2.py, please check:

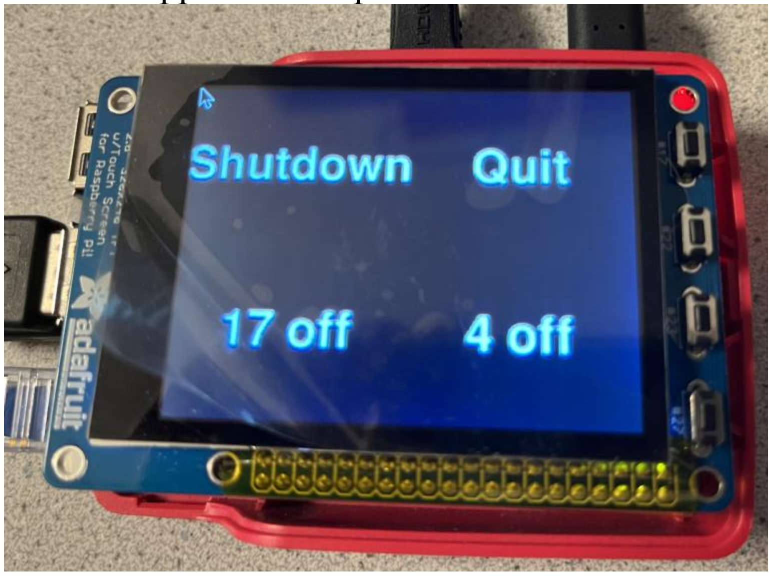
```
pi@RPi-jfs9:~ $ ls -l /dev/input/touchscreen
lrwxrwxrwx 1 root root 6 Sep  4 08:55 /dev/input/touchscreen -> event3
```

If there is **NO** entry for /dev/input touchscreen, please refer to the steps in the Lab 1 guide on page 34 for creating the correct link for your system.

Once everything is ready, try the command:

```
      sudo python sdl_v2.py
```

If all is well, a Menu should appear on the piTFT with a number of touch buttons:



Hitting the 'quit' touch button should return you to the command line window. sdl_v2.py is a great, short(!) test code to have a look at to understand how touch events work in this new system.

What follows is an annotated version (notes in green text) of this test program to indicate the essential components necessary to recognize touch events on the system:

```
pi@RPi-jfs9:~/Lab2/Raspberry-Pi-Testing $ cat sdl_v2.py
import pygame,pigame  # import pigame (which imports pitft_touchscreen)
from pygame.locals import *
import os
from time import sleep
#Colours
WHITE = (255,255,255)
os.putenv('SDL_VIDEODRV','fbcon')    # two environment variables for piTFT display
os.putenv('SDL_FBDEV', '/dev/fb1')
os.putenv('SDL_MOUSEDRV','dummy')      # Environment variables for touchscreen
os.putenv('SDL_MOUSEDEV','/dev/null')
os.putenv('DISPLAY','')

pygame.init()
pitft = pigame.PiTft()  #create a pitft object for
                        # moving touch events from evdev to pygame
lcd = pygame.display.set_mode((320, 240))
lcd.fill((0,0,0))

font_big = pygame.font.Font(None, 50)

touch_buttons = {'Shutdown':(80,60), 'Quit':(240,60), '17 off':(80,180), '4 off':(240,180)}

for k,v in touch_buttons.items():
    text_surface = font_big.render('%s'%k, True, WHITE)
    rect = text_surface.get_rect(center=v)
    lcd.blit(text_surface, rect)

pygame.display.flip()


code_run = True
while code_run:
    pitft.update()  # each time through loop, query the system for evdev to pygame events
    # Scan touchscreen events
    for event in pygame.event.get():
        if(event.type is MOUSEBUTTONDOWN):
            x,y = pygame.mouse.get_pos()
            print(x,y)
        elif(event.type is MOUSEBUTTONUP):
            x,y = pygame.mouse.get_pos()
            print(x,y)
            #Find which quarter of the screen we're in
            if y > 120:
                if x < 160:
                    print("17off")
                else:
                    print("4off")
            else:
                if x < 160:
                    pygame.quit() ; import sys
                    os.system("sudo poweroff") ; sys.exit(0)
                else:
                    pygame.quit()  # exit pygame
                    code_run = False
    sleep(0.1)
del(pitft)  # delete the pitft object
```

Important Note: In the above sdl_v2.py code note that, at exit, the code runs:

```
del(pitft)
```

If this call is not included, your code will not quit correctly, if at all. The suspicion is the del(pitft) deletes the pitft object which the system expects for exit.

Initial Testing

Once the above changes and initial testing have been completed, work may proceed on adding touch screen controls to the Lab2 files. The idea for the following codes will be to use touch as a mouse click to press on-screen buttons controlling your software. It is MUCH easier to begin this process by displaying on the monitor rather than the piTFT. Using the monitor, you will be able to observe the impact of code changes and any debug/log information by using a console window. In addition, you can add print statements to a running program to indicate response to events ('pause button hit' for example).

Control for monitor use is determined by the environment variables:

```
os.putenv('SDL_VIDEODRIVER','fbcon')  # two environment variables for piTFT display
os.putenv('SDL_FBDEV', '/dev/fb1')
os.putenv('SDL_MOUSEDRV','dummy')     # Environment variables for touchscreen
os.putenv('SDL_MOUSEDEV','/dev/null')
os.putenv('DISPLAY','')
```

If you have added these into your python program, comment them out to display the on the monitor instead of on the piTFT. You will also be using:
`pygame.mouse.set_visible(False)`

Which will turn off the mouse cursor. When debugging on the monitor, comment out this line as well which will set Mouse Visible to True, by default, allowing you to easily use the mouse to click the on-screen buttons.

Once you have your code debugged and running on the monitor, you can switch to the actual piTFT. Un-comment the five environment variable settings AND the pygame.mouse.set.visible(False) line which will hide the cursor for touchscreen operation. If you now launch from either the monitor or an ssh window (on your laptop) the application should run on the piTFT.

As a final test, the code should be started from the command line on the piTFT to show the embedded operation of the application. Optionally, if you do not have a USB keyboard available, you can demonstrate this running using ssh. [2]

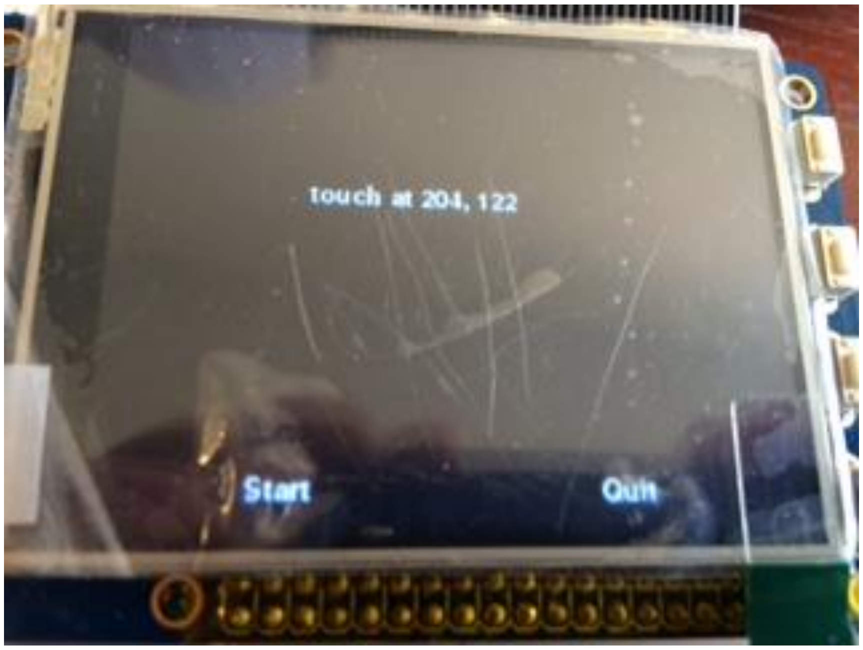
- Design a python application, quit_button.py, that displays a single 'quit' button on the lower edge of the screen. The program should be designed so that touching the 'quit' button ends the program and returns to the Linux console screen. Initially, the quit function may be implemented by touching at any location on the screen.

For easier control while debugging all touch screen programs:

- Implement a physical 'bail out' button. This button may be one of the piTFT buttons or an external button. Hitting this button should end the program.
- Also implement a code time-out that will end the program after a given time interval. For example, when you are first starting to develop the code, you may want to set this timeout to 30 seconds. This time-out can be lengthened, or eliminated, as your code stabilizes.

These methods will prevent excessive power off/power on restarts of a hung RPi system.

- Expand quit_button.py into a second python application, screen_coordinates.py.



The above example shows a screenshot of some test code to display button press (you do not need to implement the start button for this step, only the quit button)

The operation of screen_coordinates.py should be:

- Display a single quit button at the bottom of the screen
- Tapping any location on the screen still display 'Touch at x, y' where x, y show the screen coordinates of the hit. Note that the screen should be updated correctly for successive screen hits.
- As a quick guide, the upper-left of the screen is coordinate (0,0), lower-right is (320,240), upper-right is (320,0) and lower-left is (0,240)
- Tapping the 'quit' button will exit the program. Note that you may need to iterate over several runs of screen_coordinates.py to refine the coordinates of the quit button.
- All hits should be displayed on the Linux console as well
- Plan to include a copy of 20 screen taps, ranging over the extent of the piTFT screen, in your lab report. Think about how best to collect these data on screen taps; what are some simple ways to save all twenty events?
- Implement a physical bail-out button, and a time-out, for this code.

What if screen_coordinates.py doesn’t work?

sdl_v2.py, quit_button.py and screen_coordinates.py are the true tests of the correct function of touch operation on the piTFT. In spite of all of the above checks during install steps, there may still be problems. The symptoms of incorrect coordinates include:

- evtest (from Lab1) runs correctly (because this tests to see if linux is recognizing the piTFT touch.
- screen_coordinates runs correctly on a monitor (because the piTFT is different from the monitor and requires configuration to work.)

From experience, there are several classes of problem:

Cannot get quit_button or screen_coordinates running on piTFT:

- Check ‘ ls –l /dev/fb* ’ If this only shows /def/fb0, then make sure your code is directed to this device (not /dev/fb1). If both /dev/fb0 and /dev/fb1 are defined, then direct your code to /dev/fb1 to run on the piTFT
- Make sure you run python codes preceded with sudo (sudo python3 quit_button.py, for example)
- Check that the environment variables are set correctly (no typos) AND uncommented (watch out for DEV v DRV)
- Make sure that pygame.init() occurs **after** the definition of the environment variables.
- Make sure the python codes ‘pigame.py’ and ‘pitft_touchscreen.py’ have been copied into the directory where your applications are running.
- Try running on an ssh console window (from your laptop) and/or directly on the piTFT console window (if you have a USB keyboard attached to the piTFT)
- Check ‘ls –l /dev/input/touchscreen’. This should point at ‘eventN’. If /dev/input/touchscreen is not defined, recheck the instructions for the udev rule in lab1

Screen_coordinates runs on the piTFT, but the coordinates are incorrect:

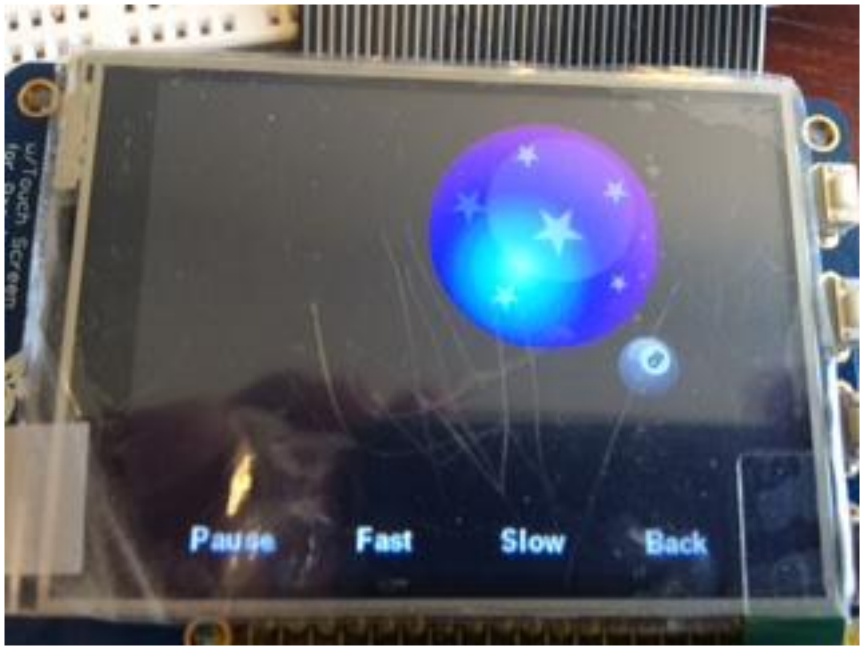
- Try rebooting the RPi and re-test
- Double check for correct entries in the environment variables in your python code. In particular:
    - MAKE SURE the Environment Variables are spelled correctly (DEV v DRV !)
    - MAKE SURE os calls to set environment variables are called BEFORE pygame.init()

Assume you get these codes working, you may proceed with the lab. If not, please discuss the issues with staff.

- Design a python program 'two_button.py' with the following functions:

    - Two, on-screen buttons are displayed 'start' and 'quit'
    - Hitting 'start' begins playback of two_collide.py
    - Hitting 'quit' ends the program and returns to the Linux console screen.
    - Hitting any other location on the screen displays screen coordinates.
    - The start and quit buttons should be displayed on the screen, and operate whenever they are displayed, during the entire time the program is running (including while the animation is playing)
    - Note that button placement should be designed so as not to interfere with video playback.
    - Implement a physical bail-out button, and a time-out, for this code.

- Design a python program control_two_collide.py with the following functions:



Example screenshot showing implementation of second level touch controls

- As in ‘two_button.py’, start and quit are implemented with identical functions. Screen coordinates should be displayed if hits occur outside of start and quit buttons. This is the ‘level 1’ menu.
- Once the animation begins to play, a second level of button controls, shown above on the ’level 2’ menu, should be displayed including the following buttons and associated functions:
  - Pause/restart: pause a running animation. Restart a paused animation
  - Faster: speed up the animation by a fixed amount
  - Slower: slow the animation by a fixed amount
  - Back: stop the animation and return to the ‘top’ menu screen which implements the start and quit buttons.
- Note that button placement should not interfere with the running animation. In particular, watch out for the placement of level 1 ‘Quit’ and level2 ‘Back’. Also, coordinates should be detected so as to separate button function (that is, avoid a single tap causing multiple buttons to be hit)
- Implement a physical bail-out button, and a time-out, for this code.

Backup your SD card.

Demonstrate all codes running on the piTFT to the TA:

- sdl_v2.py
- bounce.py, two_bounce.py and two_collide.py from week 1
- Quit_button.py
- Screen_coordinates.py
- Two_button.py running with two_collide.py animation
- Control_two_collide.py running with two_collide.py animation
- Correct function of 'bail out' button for all the above codes.

IMPORTANT: Backup your SD card.

References:

[1] From ECE4760, Lab 3, Professor Bruce Land. Link suggested by Junyin Chen, ECE5725, Fall 2016 student, Cornell MEng 2016.

[2] This section on debugging with the monitor and piTFT paraphrases a note posted by Junyin Chen, ECE5725, Fall 2016 student, Cornell MEng 2016.

[3] Detailed instructions for running PyGame touch features on a piTFT gathered from user forums and consolidated in a single document by Adafruit:
https://learn.adafruit.com/adafruit-pitft-28-inch-resistive-touchscreen-display-raspberry-pi/pitft-pygame-tips