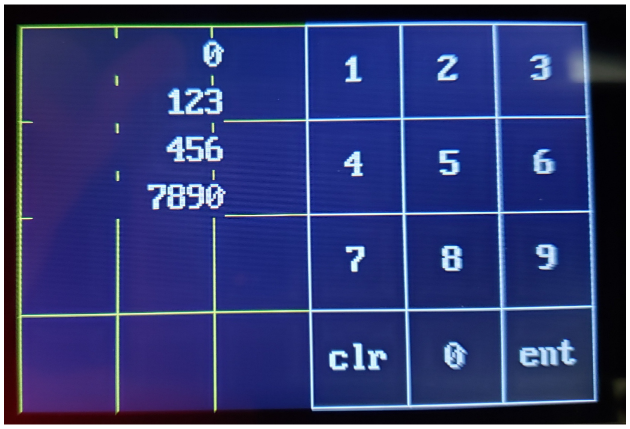
28 Touch Screens



Introduction

The ST7796S display includes both a graphics TFT display as well as a resistive touch input. With the touch input, you can set it up so the (x,y) coordinates of the touch screen match up with the graphics screen:

- (0,0) is the upper left corner
- (479,319) is the lower right corner

Multiple touch-points are supported. In theory, interrupts can be set up for when you touch the screen. In the following code, polling is used:

- Each time you call *touch.read_points()*, the number of touch points is returned along with the (x,y) coordinates of each touch point.

This lecture goes over using the touch-screen along with some basic program:

- Displaying the (x,y) location of up to 3 touch points (base code)
- Using slider bars to change the RGB color of the display
- Playing draw poker, allowing you to select which cards to discard, and
- Programming a numeric keypad with the touch screen

The base code comes from MicroPython Libraries

- https://docs.openmv.io/library/omv.gt911.html#module-gt911

gt911.py library

In order for the following routines to work, *gt911.py* must be downloaded to your Pi-Pico board. This is located on Bison Academy and comes from MicroPython Libraries

Base Code

The base code

- Initializes the touch screen

- Sets it for up to 3 touch points
- If you touch the screen, the (x,y) location of each touch point is then displayed in the shell window

```
from time import sleep_ms
from machine import Pin, I2C
from gt911 import GT911

rst_pin = Pin(10, Pin.OUT)
irq_pin = Pin(11)
sda_pin = Pin(8)
scl_pin = Pin(9)

touch = GT911(
    I2C(0, scl=scl_pin, sda=sda_pin, freq=100_000),
    rst_pin,
    irq_pin
)

touch.init(touch_points=3, refresh_rate=50)

while(1):
    num_points, points_data = touch.read_points()
    msg = ''
    for i in range(num_points):
        msg = msg + str(i) + ': ('  + str(points_data[i][0])
        msg = msg + ',' + str(points_data[i][1]) + ')   '
    if(num_points > 0):
        print(msg)
```

When you run this code,

- Up to 3 touch points are accepted
- Each time you touch another point, another column is added
- The (x,y) location of each touch-point is displayed
- You get multiple reads while you are touching the display

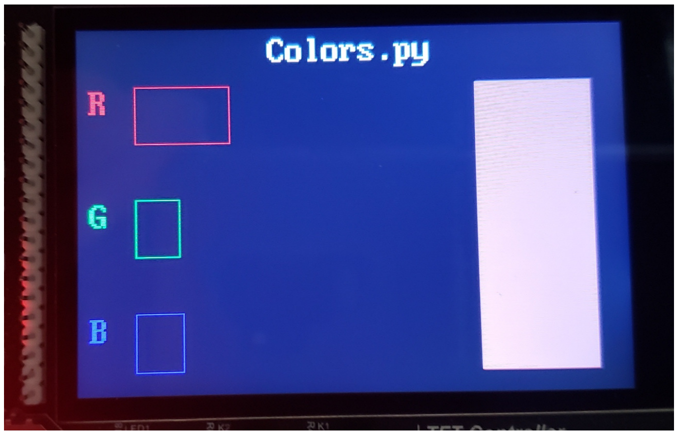
```
Shell Window
0: (263,77)
0: (263,77)
0: (263,77)
0: (263,77)  1: (92,185)
0: (263,77)  1: (92,185)
0: (263,77)  1: (91,185)
0: (263,77)  1: (473,229)  2: (92,185)
0: (263,78)  1: (473,229)  2: (92,185)
0: (264,79)  1: (468,232)  2: (94,187)
0: (265,83)  1: (455,237)  2: (112,195)
0: (269,87)  1: (447,241)  2: (135,200)
0: (272,91)  1: (434,245)  2: (150,200)
0: (427,246)  1: (157,199)
0: (413,248)  1: (168,195)
0: (409,249)  1: (171,194)
0: (405,251)  1: (176,191)
0: (404,252)  1: (181,186)
0: (404,252)
0: (142,165)
0: (142,165)
```

With this code, you can use the touch screen as an input to the Pi-Pico

Example 1: RGB Sliders

Use the touch-screen to set the RGB color for a solid rectangle drawn on the display.

- Display the RGB levels with bar graphs
- By touching (or sliding) the rectangles, each color's contribution can be adjusted from 0% (0) to 100% (255) intensity.



Code: A single touch-point is used in this (and all subsequent) programs. The x-coordinate determines the intensity

- <50 = 0 (0%)
- 50 < x < 300: 0-255 (0% to 100%)
- >300 = 255 (100%)

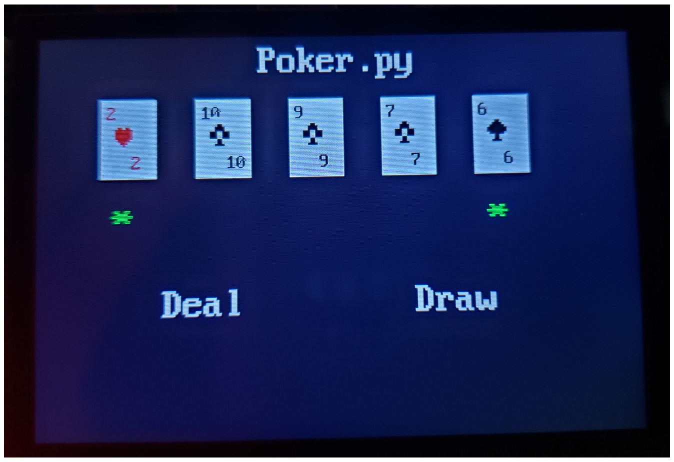
The y-coordinate determines which color you're referring to

- red: 0 < y < 100
- green: 100 < y < 200
- blue: y > 200

Polling reads the touch-screen as fast as the program can keep up, with the bar-graphs and solid rectangle redrawn each pass.

```
LCD.Init()
LCD.Clear(Black)

R = 0;
G = 0;
B = 0;

LCD.Text2('R', 10, 50, Red, Black)
LCD.Text2('G', 10, 150, Green, Black);
LCD.Text2('B', 10, 250, Blue, Black);

while True:
    num_points, points_data = touch.read_points()
    if(num_points > 0):
        tx = points_data[0][0]
        ty = points_data[0][1]
        print(tx, ty)
        if(ty < 100):
            LCD.Box(50, 50, R+50, 100, Black)
            R = max(0, min(250, tx - 50))
            LCD.Box(50, 50, R+50, 100, Red)
        elif(ty < 200):
            LCD.Box(50, 150, G+50, 200, Black)
            G = max(0, min(250, tx - 50))
            LCD.Box(50, 150, G+50, 200, Green)
        else:
            LCD.Box(50, 250, B+50, 300, Black)
            B = max(0, min(250, tx - 50))
            LCD.Box(50, 250, B+50, 300, Blue)

        Color = LCD.RGB(R, G, B)
        LCD.Solid_Box(350, 50, 450, 300, Color)
```

Example 2: Draw Poker



Here, a game of draw poker is played on the touch screen.

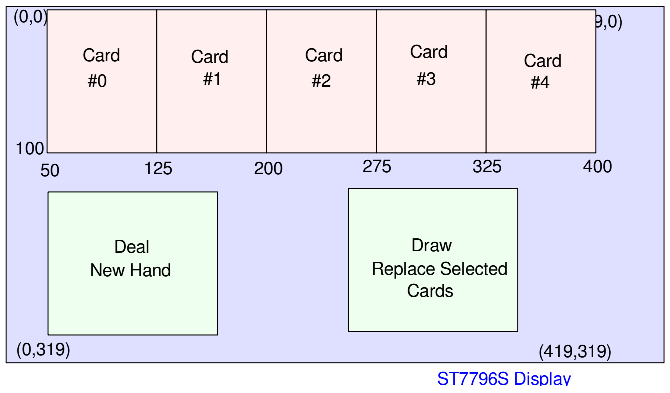
- When the game starts, the deck is shuffled and the top 5 cards are dealt out.

```
Deck = LCD.Shuffle()
Hand = [0]*5;
Value = [0]*5
Suit = [0]*5


for i in range(0,5):
    Hand[i] = Deck[i]
    Value[i] = (Hand[i] % 13) + 1
    Suit[i] = int(Hand[i] / 13) + 1
    LCD.Card(Value[i], Suit[i], 50+i*75, 50)

    LCD.Text2('Deal', 100, 200, White, Black)
    LCD.Text2('Draw', 300, 200, White, Black)

Draw = [0]*5
ptr = 5;
```

Once you have your hand the player can select which cards are to be discarded (marked with a green star) and which are to be kept. Touching the card toggles the select / deselect for that card.

The code which determines which card you're selecting by dividing the screen in to regions

- Touching within a region toggles that card being selected
- Touching outside this region is ignored



```
while(1):
    num_points, points_data = touch.read_points()
    if(num_points > 0):
        tx = points_data[0][0]
        ty = points_data[0][1]
        card = round( (tx-50) / 75)
        card = max(0, min(4, card))
        if(ty < 150):
            Draw[card] = 1 - Draw[card]
```

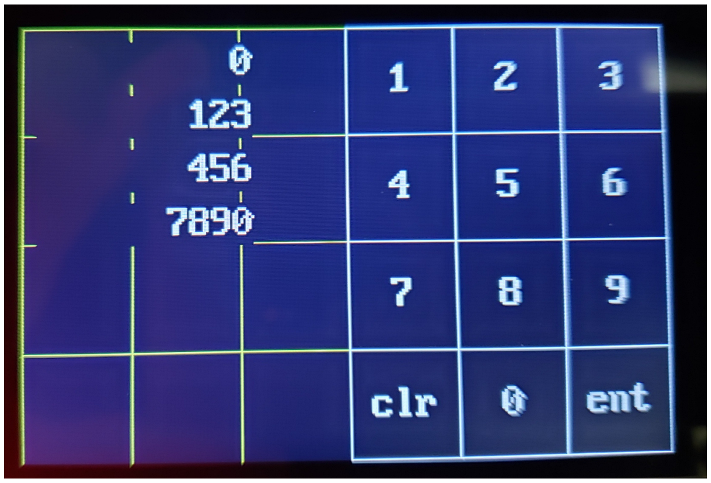
Touching in the region of (250,200) to (350,250) results in discarding the selected cards and replacing them with the top cards of the deck. As written, you can continue to replace cards over and over (multiple draw steps are allowed with this code).

```
        #Draw
        if( (tx > 250)*(tx < 350)*(ty > 200)*(ty < 250) ):
            for i in range(0,5):
                if(Draw[i] == 1):
                    Hand[i] = Deck[ptr]
                    ptr = ptr + 1
                Draw[i] = 0
                Beeper.value(1)
                sleep_ms(10)
                Beeper.value(0)
```

Touching in the region (50,200) to (150,250) results in reshuffling the deck and drawing 5 new cards (new game).

```
           # Deal
        if( (tx > 50)*(tx < 150)*(ty > 200)*(ty < 250) ):
            Deck = LCD.Shuffle()
            for i in range(0,5):
                Hand[i] = Deck[i];
            ptr = 5
            Draw = [0]*5
            for i in range(0,2):
                Beeper.value(1)
                sleep_ms(10)
                Beeper.value(0)
                sleep_ms(100)
```

Example 4: Numeric Keypad



The final sample code uses the touch-screen as a numeric keypad:

- You must press and release a button for it to register (no sliding)
- Numbers 0..9 append that number to the right of the message
- *clr* removes the rightmost number
- *Enter* pushes that number onto a stack

Several subroutines are defined in this code.

Place_Button() places a button along with its label at location (bx, by). The total size of the numeric keypad is (x,y). For example, the keypad shown above is a 6x4 keypad. Number '1' is at location (0,3)

- Top left corner is (0,0)
- Bottom right corner is (5,3)

```
def Place_Button(bx, by, x, y, Label):
    n = len(Label)
    Sx = 480//x
    Sy = 320//y
    x0 = bx*Sx
    y0 = by*Sy
    LCD.Box(x0, y0, min(479, x0 + Sx), min(319, y0 +  Sy), White)
    LCD.Text2(Label, x0+Sx//2-8*n, y0+Sy//2-16, White, Black)
```

Draw_Keyboard() draws the keyboard and labels each button.

```
def Draw_Keyboard(x,y):
    Sx = 480//x
    Sy = 320//y
    for i in range(0,y+1):
        LCD.Line(0,i*Sy,479,i*Sy, Grey)
    for i in range(0,x+1):
        LCD.Line(i*Sx,0,i*Sx,319, Grey)

    Place_Button(3, 0, x, y, '1')
    Place_Button(4, 0, x, y, '2')
    Place_Button(5, 0, x, y, '3')

    Place_Button(3, 1, x, y, '4')
    Place_Button(4, 1, x, y, '5')
    Place_Button(5, 1, x, y, '6')

    Place_Button(3, 2, x, y, '7')
    Place_Button(4, 2, x, y, '8')
    Place_Button(5, 2, x, y, '9')

    Place_Button(3, 3, x, y, 'clr')
    Place_Button(4, 3, x, y, '0')
    Place_Button(5, 3, x, y, 'ent')
```

Read_Keypad(x,y): waits until you press and release a button, It then returns the (x,y) location of that key.

- (x,y) are the dimensions of the keypad. In this example, this is a 6x4 keypad.
- The while() loops attempt to wait until you press then wait until you release the button.
- The sleep_ms() statements slow down the sampling rate to avoid double reads. The number 30ms was found by trial-and-error.

```
def Read_Keypad(x, y):
    Sx = 480//x
    Sy = 320//y
    num_points, points_data = touch.read_points()
    while(num_points == 0):
        num_points, points_data = touch.read_points()
        sleep_ms(30)
    tx = points_data[0][0] // Sx
    ty = points_data[0][1] // Sy
    while(num_points > 0):
        num_points, points_data = touch.read_points()
        sleep_ms(30)
    return(tx, ty)
```

Finally, the main routine checks which key was pressed

- Each time you press and release a button, the (x,y) location of that key is returned
- The (x,y) location determines what you do
    - For numbers, append that value to the right of variable X
    - For clr, remove the rightmost value (do an integer divide by 10)
    - For enter, push the numbers on the stack

The new value of X is then displayed to the left of the keypad. If a push-command was executed (changing Y, Z, and T), those values are also updated. This just speeds up the display function.

```
while(1):
    [x, y] = Read_Keypad(6,4)
    print(x, y)
    if(y==0):
        if(x==3):
            X = 10*X + 1
        elif(x==4):
            X = 10*X + 2
        elif(x==5):
            X = 10*X + 3
    :
    :
    if(flag == 1):
       LCD.Number2(T, 9, 0, 10, 10, White, Black)
       LCD.Number2(Z, 9, 0, 10, 50, White, Black)
       LCD.Number2(Y, 9, 0, 10, 90, White, Black)
       flag = 0
    LCD.Number2(X, 9, 0, 10,130, White, Black)
```

Once you have a keypad, you can use this for entering numbers, building a calculator, etc.

Summary

The touch screen is actually pretty easy to use with polling.

- Each time you call touch.read_points(), the number of touch points and their corresponding (x,y) locations is returned.
- Each time you call Read_Keypad(), the program waits until you press and release on the screen. The (x,y) location of the corresponding key is then returned.

What you do with this is kind of limited by the creativity of the programmer.