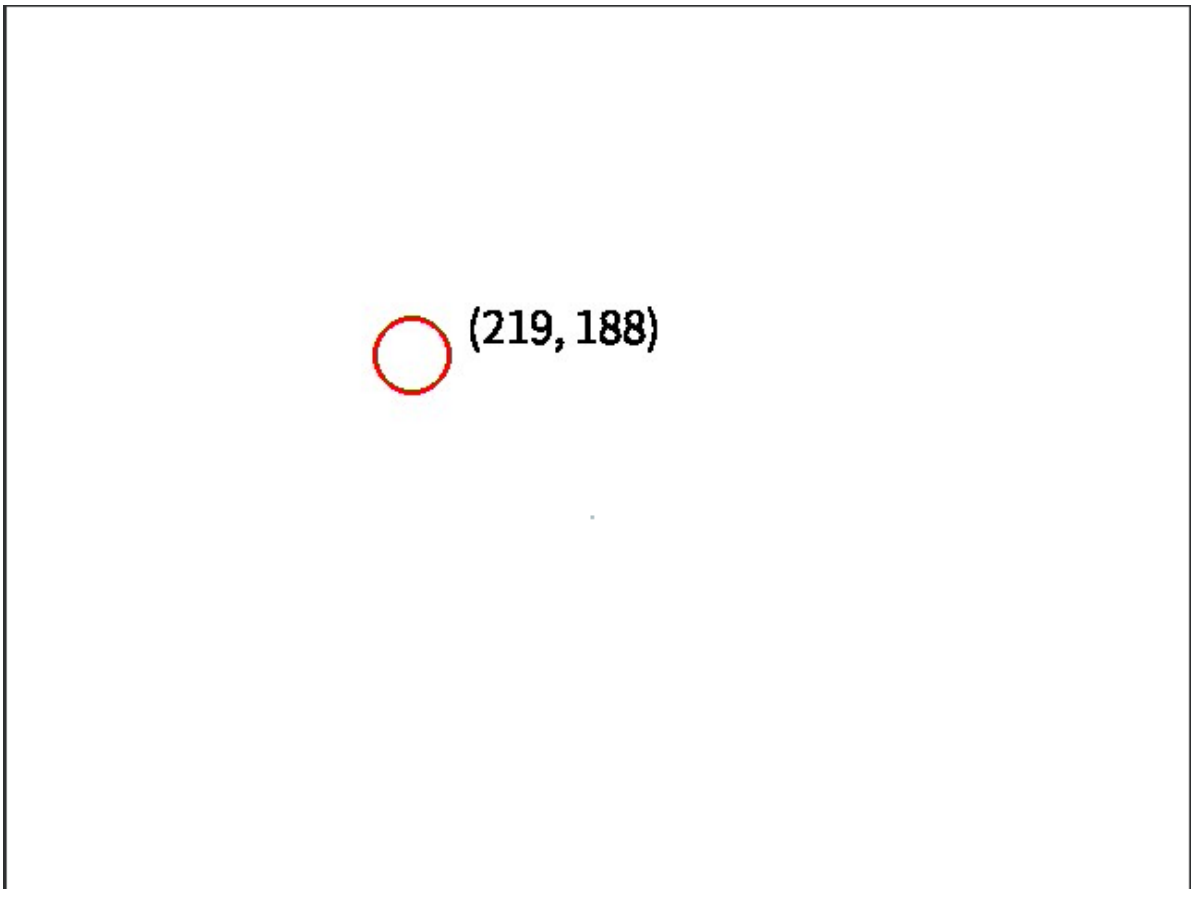
# Touch point Detection



## Example Results

The example source code file for this section is named [1.show_touch_info.py]

Connect to the IDE, run the code in the IDE, and click anywhere on the screen to see the touch point information appear.




## Code Explanation

This section will detail the functionality and implementation logic of `1.show_touch_info.py` to help users understand how to capture touch point information using the touch sensor and display it on the screen.

## 1. Importing Necessary Modules

```
import time, os, urandom, sys
from media.display import *
from media.media import *
from machine import TOUCH
```

- `time`, `os`, `urandom`, `sys`: These are system-related modules used for time control, operating system functions, random number generation, and system operations. - `media.display` and `media.media`: For screen display and media resource management.
- `machine.TOUCH`: For initializing and operating the touch sensor.

## 2. Initializing the touch sensor and screen resolution

```
tp = TOUCH(0)
DISPLAY_WIDTH = 640
DISPLAY_HEIGHT = 480
```

- `TOUCH(0)`: Initializes the touch sensor, `tp` is used to read touch data later.
- `DISPLAY_WIDTH` and `DISPLAY_HEIGHT`: Define screen resolution constants, which are 640x480 respectively.

## 3. Define touch point display function

```
def touch_point_display():
    print("Touch point display test")
```

- `touch_point_display()` is the main function responsible for handling touch point detection and display.

### 3.1 Creating and Initializing the Background Image

```
img = image.Image(DISPLAY_WIDTH, DISPLAY_HEIGHT, image.RGB888)

def clear_and_reset_background():
    """Clear the image and reset to white background"""
    img.clear()
    img.draw_rectangle(0, 0, DISPLAY_WIDTH, DISPLAY_HEIGHT, color=(255,255,255),
fill=True)

clear_and_reset_background()
```

- Create an image object `img` in RGB888 format, sized to the screen resolution.
- Define the `clear_and_reset_background()` function to clear the image contents and reset the screen to a white background.
- Call this function during initialization to ensure the screen is initially white.

### 3.2 Initializing the Display and Media Manager

```
Display.init(Display.ST7701, width = DISPLAY_WIDTH, height = DISPLAY_HEIGHT,
to_ide = True)
MediaManager.init()
```

- Initialize the display using the ST7701 driver, set the resolution to 640x480, and enable IDE output ( `to_ide = True` ).
- Initialize the media manager to manage display-related resources.

## 4. Touch Detection and Processing Logic

### 4.1 Defining Touch State Variables

```
current_touch_active = False
last_touch_down_time = 0
```

- `current_touch_active` : A Boolean value that records whether a touch is currently active.
- `last_touch_down_time` : Records the time of the last touch-down event.

### 4.2 Main Loop: Reading and Processing Touch Data

```
while True:
    os.exitpoint()
    point = tp.read(1)
```

- Use `tp.read(1)` to read touch data, and `point` to store touch point information.

#### 4.2.1 Handling Touch Press Events

```
if len(point):
    pt = point[0]
    if pt.event == TOUCH.EVENT_DOWN:
        clear_and_reset_background()
        circle_radius = 20
        img.draw_circle(pt.x, pt.y, circle_radius, color=(255, 0, 0),
thickness=3)
        coord_text = f"({pt.x}, {pt.y})"
        text_x = pt.x + 30
        text_y = pt.y - 30
        if text_x + 100 > DISPLAY_WIDTH:
            text_x = pt.x - 100
        if text_y < 20:
            text_y = pt.y + 40
        img.draw_string_advanced(text_x, text_y, 24, coord_text, color=(0, 0,
0))
        current_touch_active = True
        last_touch_down_time = time.time()
        print(f"Touch detected at: {coord_text}")
```

- When a touch down event ( `TOUCH.EVENT_DOWN` ) is detected:
    - Clear the previous image and reset the background to white.
    - Draw a red circle with a radius of 20 at the touch point.

- Calculate and display the touch point's coordinates in text, adjusting the text position based on the touch point to ensure it stays within the screen boundaries.
- Update the touch state variable `current_touch_active` to `True` and record the touch down time.
- Print the touch point coordinates to the console.

#### 4.2.2 Handling Touch Up Events

```
elif pt.event == TOUCH.EVENT_UP:
    current_touch_active = False
```

- When a touch up event (`TOUCH.EVENT_UP`) is detected, `current_touch_active` is set to `False`, indicating that the touch has ended.

### 4.3 Update Display

```
Display.show_image(img)
time.sleep(0.05)
```

- Use `Display.show_image(img)` to update the screen to show the current image.
- `time.sleep(0.05)` updates every 50 milliseconds to avoid performance issues caused by excessively fast refreshes.

## 5. Exception Handling and Resource Release

```
except KeyboardInterrupt as e:
    print("User stop: ", e)
except BaseException as e:
    print(f"Exception {e}")

Display.deinit()
os.exitpoint(os.EXITPOINT_ENABLE_SLEEP)
time.sleep_ms(100)
MediaManager.deinit()
```

- Catch `KeyboardInterrupt` (user interrupt) and `BaseException` (other exceptions) and print relevant information.
- Call `Display.deinit()` to deinitialize the display.
- Call `MediaManager.deinit()` to release media resources.

## 6. Main program entry

```
if __name__ == "__main__":
    os.exitpoint(os.EXITPOINT_ENABLE)
    touch_point_display()
```

- Enable the exit point (`EXITPOINT_ENABLE`) and call the `touch_point_display()` function to start the touch point display test.