

Template Matching

- Template matching (TM) is a basic technique for locating a template image within a larger target image. It involves sliding the template across the target image and calculating a similarity score at each position.
- Template matching has a variety of applications in computer vision, for example:
 - Object detection: Template matching can be used to identify objects within a larger image by searching for areas that closely match a predefined template.
 - Facial recognition: Template matching techniques can be employed in security systems to recognize faces by comparing facial features to stored templates.
 - Navigation of mobile robots: Template matching can be used to help robots navigate by recognizing landmarks and features in their environment.
 - Quality control: Template matching can be used to inspect products for defects by comparing them to a standard template.
 - Medical imaging: Template matching is applicable in medical imaging for tasks such as image registration and identifying anatomical structures.
 - Activity recognition: Template matching can be used to recognize and classify human activities in videos by comparing the motion patterns to predefined templates.
 - Vehicle Counting: Template matching can be used to count vehicles in traffic monitoring applications.

Template Matching

- In essence, TM is relatively basic method for object localization.
 - We need the **template** image and the **source** (input) image.
 - We need compare this template vs. overlapped image regions.



template



source

Template Matching



Template Matching

How to compare the template image vs. the source image?

Template Matching

How to compare the template image vs. the source image?

SAD - Sum of absolute differences

Template Matching

How to compare the template image vs. the source image?

SAD - Sum of absolute differences

$$SAD = \sum_{x,y} |I_1[x, y] - I_2[x, y]|$$

Template Matching

How to compare the template image vs. the source image?

SAD - Sum of absolute differences

$$SAD = \sum_{x,y} |I_1[x, y] - I_2[x, y]|$$

SSD - Sum of square differences

Template Matching

How to compare the template image vs. the source image?

SAD - Sum of absolute differences

$$SAD = \sum_{x,y} |I_1[x, y] - I_2[x, y]|$$

SSD - Sum of square differences

$$SSD = \sum_{x,y} (I_1[x, y] - I_2[x, y])^2$$

Template Matching

How to compare the template image vs. the source image?

SAD - Sum of absolute differences

$$SAD = \sum_{x,y} |I_1[x, y] - I_2[x, y]|$$

SSD - Sum of square differences

$$SSD = \sum_{x,y} (I_1[x, y] - I_2[x, y])^2$$

CC- Cross correlation

Template Matching

How to compare the template image vs. the source image?

SAD - Sum of absolute differences

$$SAD = \sum_{x,y} |I_1[x, y] - I_2[x, y]|$$

SSD - Sum of square differences

$$SSD = \sum_{x,y} (I_1[x, y] - I_2[x, y])^2$$

CC- Cross correlation

$$CC = \sum_{x,y} (I_1[x, y] \cdot I_2[x, y])$$

Template Matching

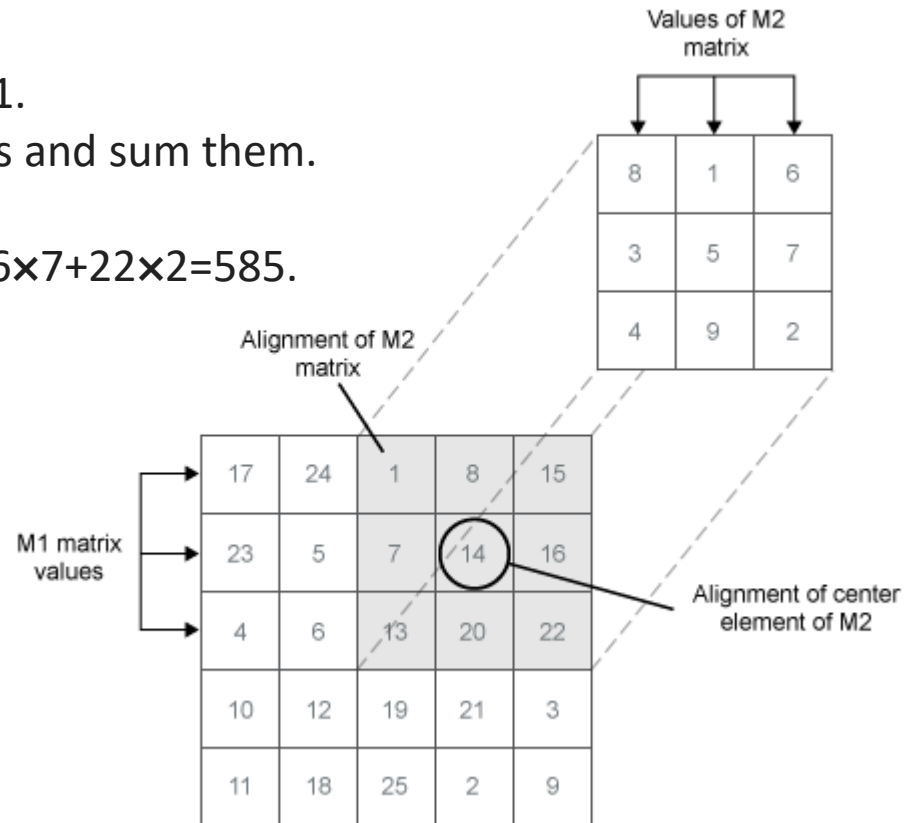
Example of cross correlation.

Suppose that M2 is on top of the matrix M1.

Compute the element-by-element products and sum them.

The answer should be:

$$1 \times 8 + 7 \times 3 + 13 \times 4 + 8 \times 1 + 14 \times 5 + 20 \times 9 + 15 \times 6 + 16 \times 7 + 22 \times 2 = 585.$$



Template Matching

- Template matching is a basic technique for locating a template image within a larger target image. It involves sliding the template across the target image and calculating a similarity score at each position. The location with the highest similarity score indicates the best match. OpenCV's **cv2.matchTemplate()** function can be used for this process, taking the template and target images as inputs and producing a result matrix.
- The **cv2.matchTemplate function in OpenCV** takes three parameters:
 - The input image that contains the object to be detected.
 - The template of the object.
 - The template matching method.
- The function returns a result matrix, where each element corresponds to a position in the target image and contains a value indicating the degree of similarity between the template and the content of the target image at that position.

Template Matching

Several comparison methods are implemented in OpenCV:

- TM_SQDIFF
- TM_CCORR
- TM_CCOEFF
- TM_SQDIFF_NORMED
- TM_CCORR_NORMED
- TM_CCOEFF_NORMED

Template Matching

- Several comparison methods are implemented in OpenCV:

- TM_SQDIFF
$$R(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2$$

- TM_CCORR
$$R(x, y) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))$$

Cross-Correlation (TM_CCORR) and its limitations:

The result is highly dependent on the brightness and contrast of both the template and the image region. For example:

- A bright region in the image (e.g., a white patch) will produce a high correlation value, even if it doesn't match the template pattern.
- Variations in lighting or contrast between the template and image can lead to incorrect matches.⁶⁵

Template Matching

- There are also the **normalized** versions (that address some of the problems):
https://docs.opencv.org/4.x/df/dfb/group_imgproc_object.html#ga586ebfb0a7fb604b35a23d85391329be

- TM_SQDIFF_NORMED
$$R(x, y) = \frac{\sum_{x',y'} (T(x', y') - I(x + x', y + y'))^2}{\sqrt{\sum_{x',y'} T(x', y')^2 \cdot \sum_{x',y'} I(x + x', y + y')^2}}$$

- TM_CCORR_NORMED
$$R(x, y) = \frac{\sum_{x',y'} (T(x', y') \cdot I(x + x', y + y'))}{\sqrt{\sum_{x',y'} T(x', y')^2 \cdot \sum_{x',y'} I(x + x', y + y')^2}}$$

- The results are scaled to a consistent range (e.g. 0 to 1), making it easier to compare different regions of an image or different templates.

Template Matching

- There are also the **normalized** versions (that address some of the problems):
https://docs.opencv.org/4.x/df/dfb/group_imgproc_object.html#ga586ebfb0a7fb604b35a23d85391329be

- TM_SQDIFF_NORMED
$$R(x, y) = \frac{\sum_{x',y'} (T(x', y') - I(x + x', y + y'))^2}{\sqrt{\sum_{x',y'} T(x', y')^2 \cdot \sum_{x',y'} I(x + x', y + y')^2}}$$

- TM_CCORR_NORMED
$$R(x, y) = \frac{\sum_{x',y'} (T(x', y') \cdot I(x + x', y + y'))}{\sqrt{\sum_{x',y'} T(x', y')^2 \cdot \sum_{x',y'} I(x + x', y + y')^2}}$$

- After the function finishes the comparison, the best matches can be found as global minimums (when **TM_SQDIFF** was used) or maximums (when **TM_CCORR** or **TM_CCOEFF** was used) using the **minMaxLoc** function.

Template Matching

- **The Correlation Coefficient (cv2.TM_CCOEFF / cv2.TM_CCOEFF_NORMED)** method calculates the correlation coefficient between the template and the target image.
- A higher value indicates a better match. This method matches a template relative to its mean against the image relative to its mean. Therefore, a perfect match will be 1, and a perfect mismatch will be -1.

$$R(x, y) = \sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))$$

TM_CCOEFF

Python: cv.TM_CCOEFF

where

$$T'(x', y') = T(x', y') - 1/(w \cdot h) \cdot \sum_{x'', y''} T(x'', y'')$$

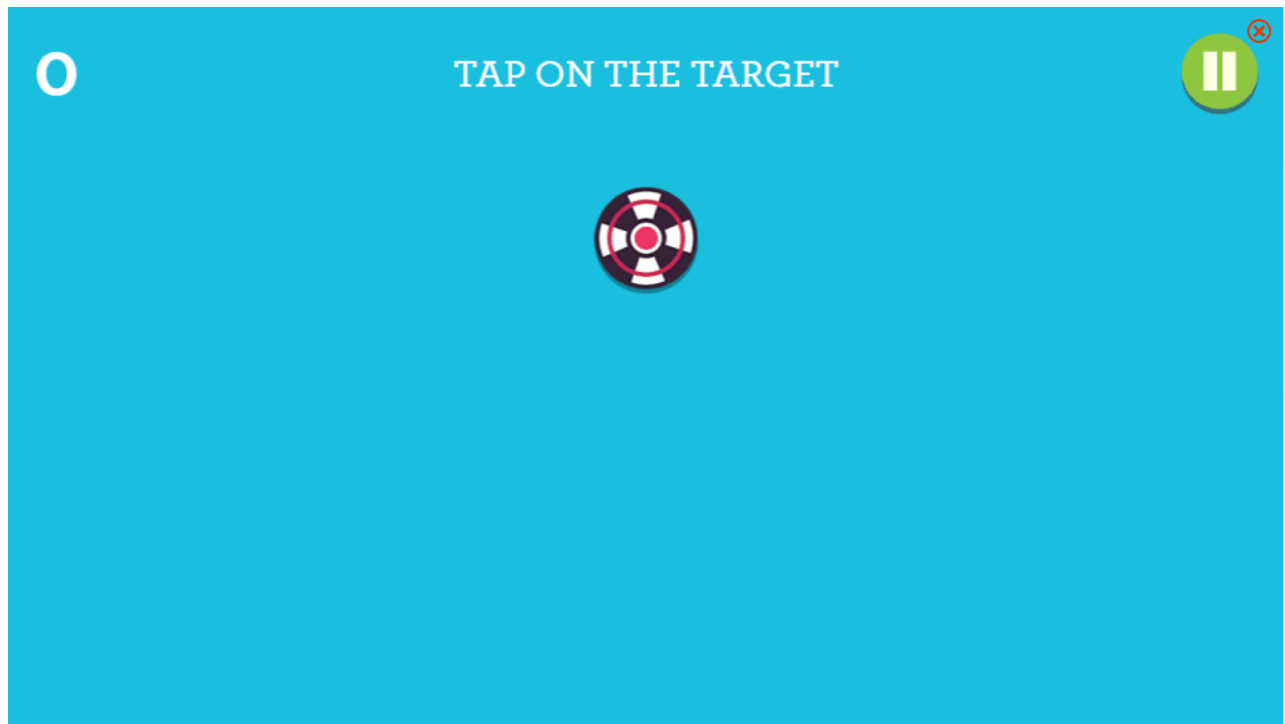
$$I'(x + x', y + y') = I(x + x', y + y') - 1/(w \cdot h) \cdot \sum_{x'', y''} I(x + x'', y + y'')$$

TM_CCOEFF_NORMED

Python: cv.TM_CCOEFF_NORMED

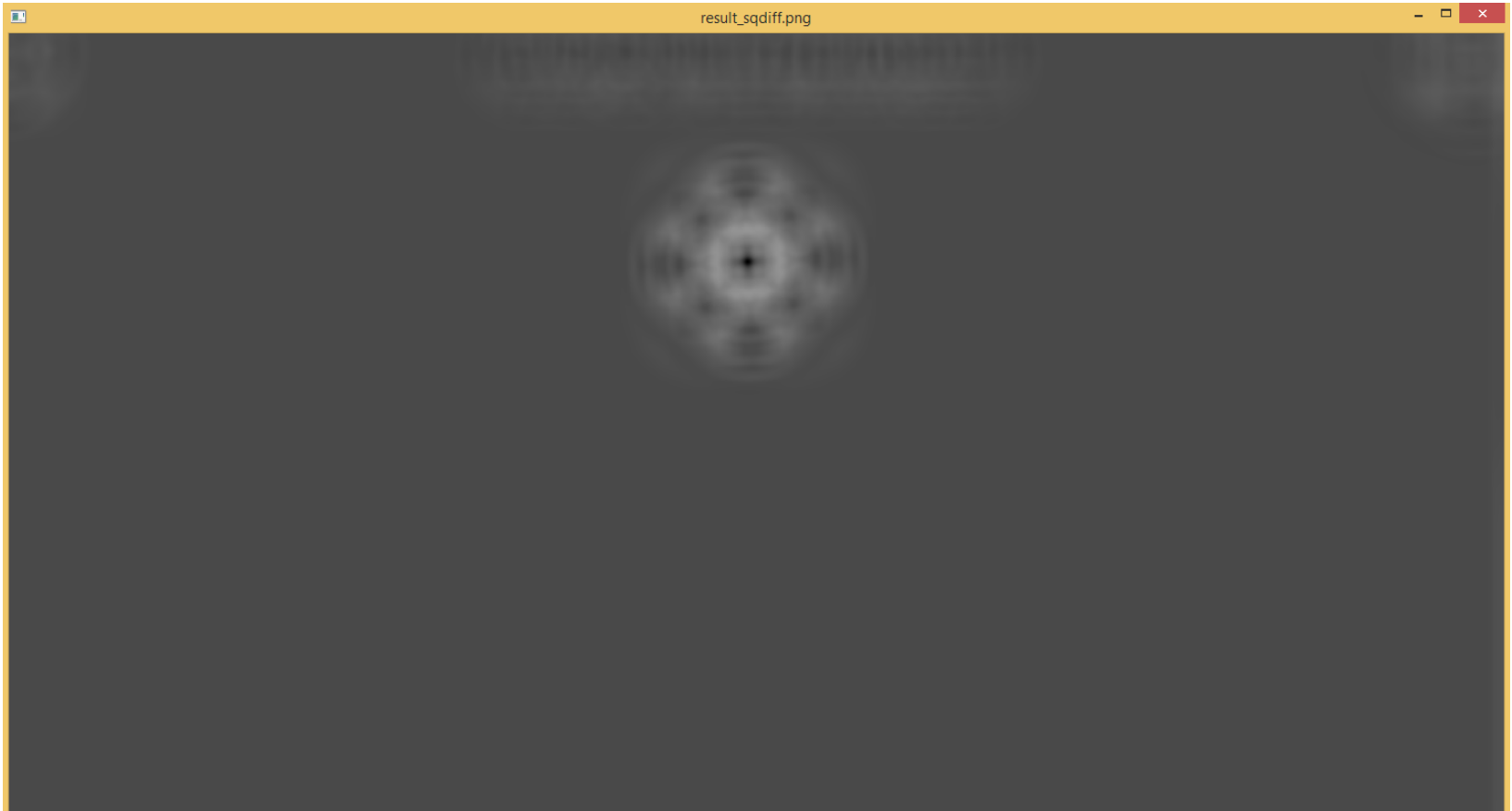
$$R(x, y) = \frac{\sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} T'^2(x', y') \cdot \sum_{x', y'} I'^2(x + x', y + y')}}^2$$

Template Matching



Template Matching

TM_SQDIFF_NORMED



Template Matching

TM_CCORR_NORMED



Template Matching

- Example – preparing phase:

The screenshot shows a Python IDE with a project named 'template_02'. The code in 'main.py' defines a function 'template_simple()' that reads two images, converts them to grayscale, and prints their dimensions. The 'Run' console shows the output: 'src w h 1600 900' and 'temp w h 131 136'. Two windows are open: 'src_mat' showing a large grayscale image of a target with the text 'TAP ON THE TARGET' and a pause button, and 'temp_mat' showing a smaller grayscale image of a template (a target symbol).

```

16
17 def template_simple():
18
19     cv2.namedWindow("src_mat", 0)
20     cv2.namedWindow("temp_mat", 0)
21
22     src_mat = cv2.imread("source.png")
23     temp_mat = cv2.imread("template.png")
24
25     src_mat = cv2.cvtColor(src_mat, cv2.COLOR_BGR2GRAY)
26     temp_mat = cv2.cvtColor(temp_mat, cv2.COLOR_BGR2GRAY)
27
28     src_mat_h = src_mat.shape[0]
29     src_mat_w = src_mat.shape[1]
30     print("src w h", src_mat_w, src_mat_h)
31
32     temp_mat_h = temp_mat.shape[0]
33     temp_mat_w = temp_mat.shape[1]
34     print("temp w h", temp_mat_w, temp_mat_h)
35
36     cv2.imshow("src_mat", src_mat)
37     cv2.imshow("temp_mat", temp_mat)
38     cv2.waitKey(0)
    
```

Run: main x

```

src w h 1600 900
temp w h 131 136
    
```

Template Matching

- Example – localization phase:

The screenshot shows an IDE with a Python script named `main.py` in a project called `template_02`. The code performs template matching on a source image (`src_mat`) using a template (`temp_mat`). The localization phase is highlighted, showing the calculation of the bounding box and center point of the match.

```

42     result = cv2.matchTemplate(src_mat, temp_mat, cv2.TM_CCORR_NORMED)
43
44     min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
45     print("max_val", round(max_val, 2))
46
47     top_left = max_loc
48     bottom_right = (top_left[0] + temp_mat_w, top_left[1] + temp_mat_h)
49
50     center_x = int((top_left[0] + bottom_right[0])/2)
51     center_y = int((top_left[1] + bottom_right[1])/2)
52
53     src_mat_copy = src_mat.copy()
54     cv2.namedWindow("src_mat_copy", 0)
55
56     cv2.rectangle(src_mat_copy, top_left, bottom_right, 255, 5)
57     cv2.circle(src_mat_copy, (center_x, center_y), 5, 255, -1)
58
59     cv2.imshow("src_mat_copy", src_mat_copy)
60     cv2.waitKey(0)
61
62
63

```

The `src_mat_copy` window displays the source image with a red bounding box around the target (a coin) and a red circle at its center. The text "TAP ON THE TARGET" is visible in the window. The console output shows:

```

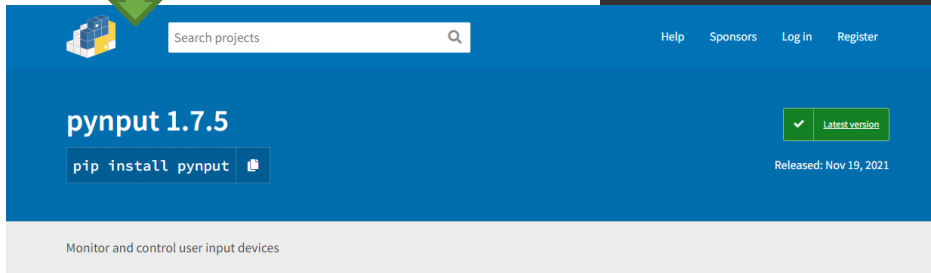
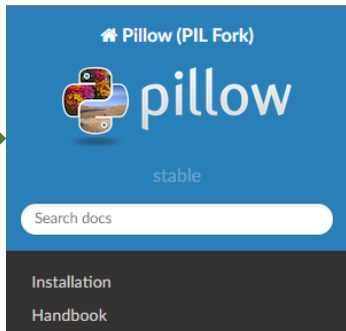
temp w h 131 136
max_val 1.0

```

Template Matching

- taking screenshots :

```
import cv2
from PIL import ImageGrab
import numpy as np
from pynput.mouse import Button, Controller
```



Docs » Reference » ImageGrab Module

[Edit on GitHub](#)

ImageGrab Module

The `ImageGrab` module can be used to copy the contents of the screen or the clipboard to a PIL image memory.

New in version 1.1.3.

```
PIL.ImageGrab.grab(bbox=None, include_layered_windows=False, all_screens=False, xdisplay=None) \[source\]
```

Take a snapshot of the screen. The pixels inside the bounding box are returned as an "RGBA" on macOS, or an "RGB" image otherwise. If the bounding box is omitted, the entire screen is copied.

New in version 1.1.3: (Windows), 3.0.0 (macOS), 7.1.0 (Linux (X11))

Parameters

- bbox** – What region to copy. Default is the entire screen. Note that on Windows OS, the top-left point may be negative if `all_screens=True` is used.
- include_layered_windows** – Includes layered windows. Windows OS only.
New in version 6.1.0.
- all_screens** – Capture all monitors. Windows OS only.

Template Matching

- Save screenshot for creating the template :

```

def simple_capture():

    image_grab = ImageGrab.grab(bbox=None) # x, y, w, h
    screen_mat = np.array(image_grab)

    screen_mat_h = screen_mat.shape[0]
    screen_mat_w = screen_mat.shape[1]
    print("screen_mat w h", screen_mat_w, screen_mat_h)

    screen_mat = cv2.cvtColor(screen_mat, cv2.COLOR_BGR2GRAY)
    cv2.imwrite("screen.jpg", screen_mat)
  
```


Template Matching

- pynput example :

```
1 import cv2 as cv
2 import numpy as np
3 from PIL import ImageGrab
4 from pynput.mouse import Button, Controller
5 import time
6
7 mouse = Controller()
8 n_frame = 0
9 x=21
10 y=648
11 while(True):
12     image_grab = ImageGrab.grab(bbox=None)
13     image_grab_np = np.array(image_grab)
14     cv.imwrite(f"out/{n_frame}.jpg", image_grab_np)
15     print('The current pointer position is {0}'.format(mouse.position))
16     mouse.position = (x, y)
17     mouse.press(Button.left)
18     mouse.release(Button.left)
19     n_frame += 1
20     time.sleep(5)
```