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 relativelly 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

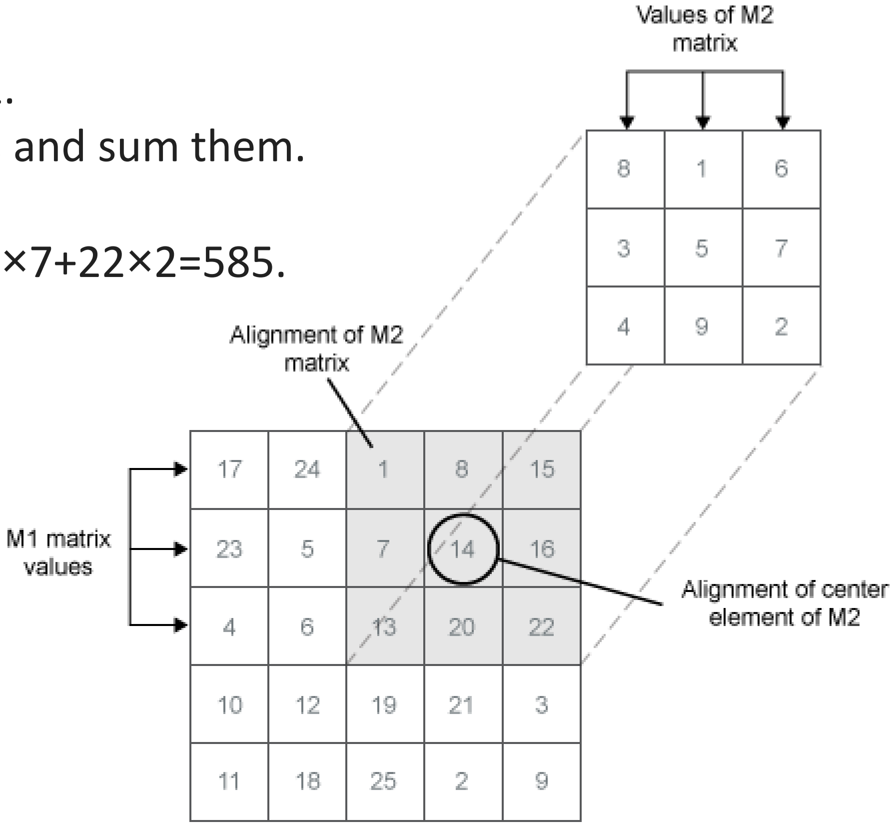
$$SSD = \sum_{x,y} (I_1[x,y] - I_2[x,y])^2$$

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:
1×8+7×3+13×4+8×1+14×5+20×9+15×6+16×7+22×2=585.



https://www.mathworks.com/help/signal/ref/xcorr2.html

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.

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.

TM_CCOEFF
Python: cv.TM_CCOEFF

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

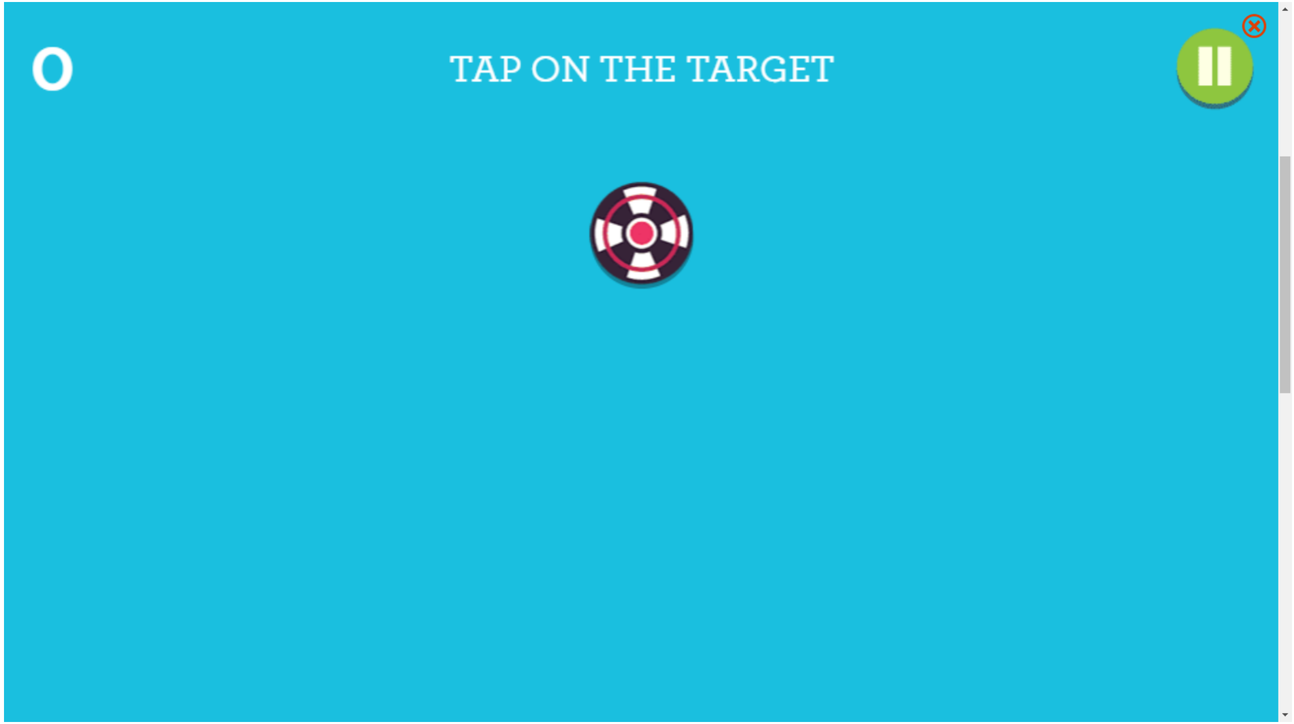
where

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

TM_CCOEFF_NORMED
Python: cv.TM_CCOEFF_NORMED

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

Template Matching



https://www.addictinggames.com/shooting/dart-master

Template Matching

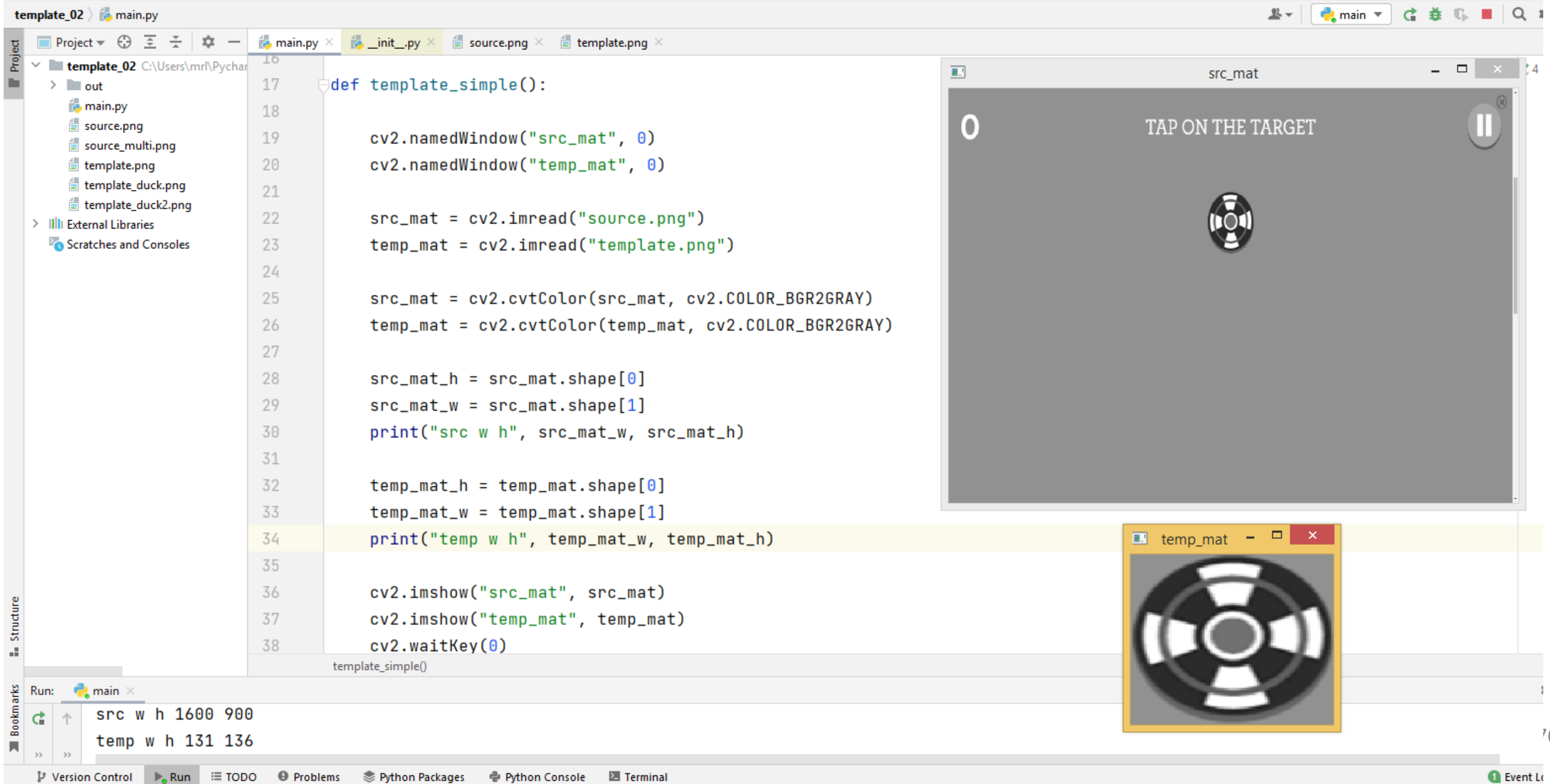
TM_SQDIFF_NORMED

Template Matching

TM_CCORR_NORMED

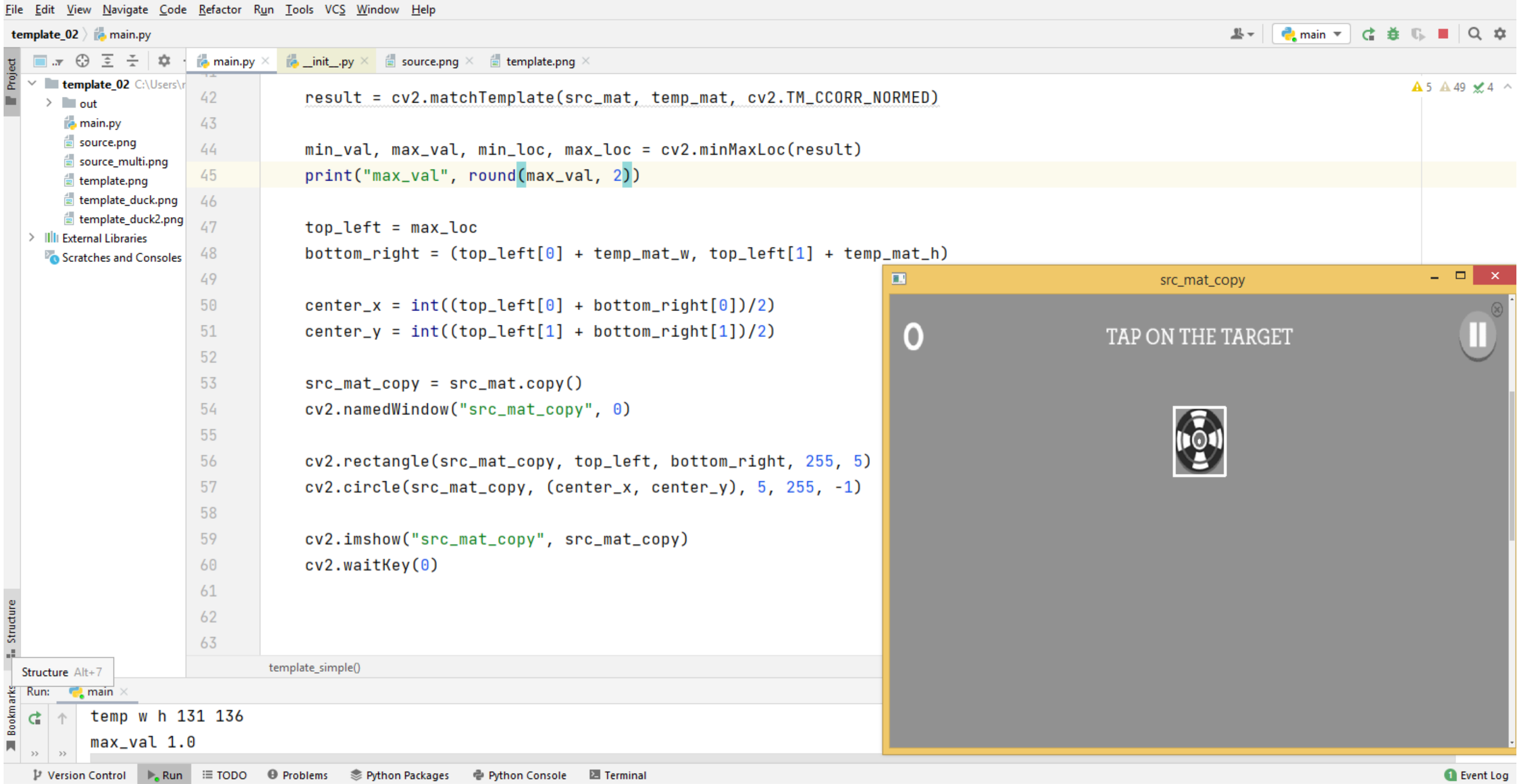
Template Matching

- Example – preparing phase:



Template Matching

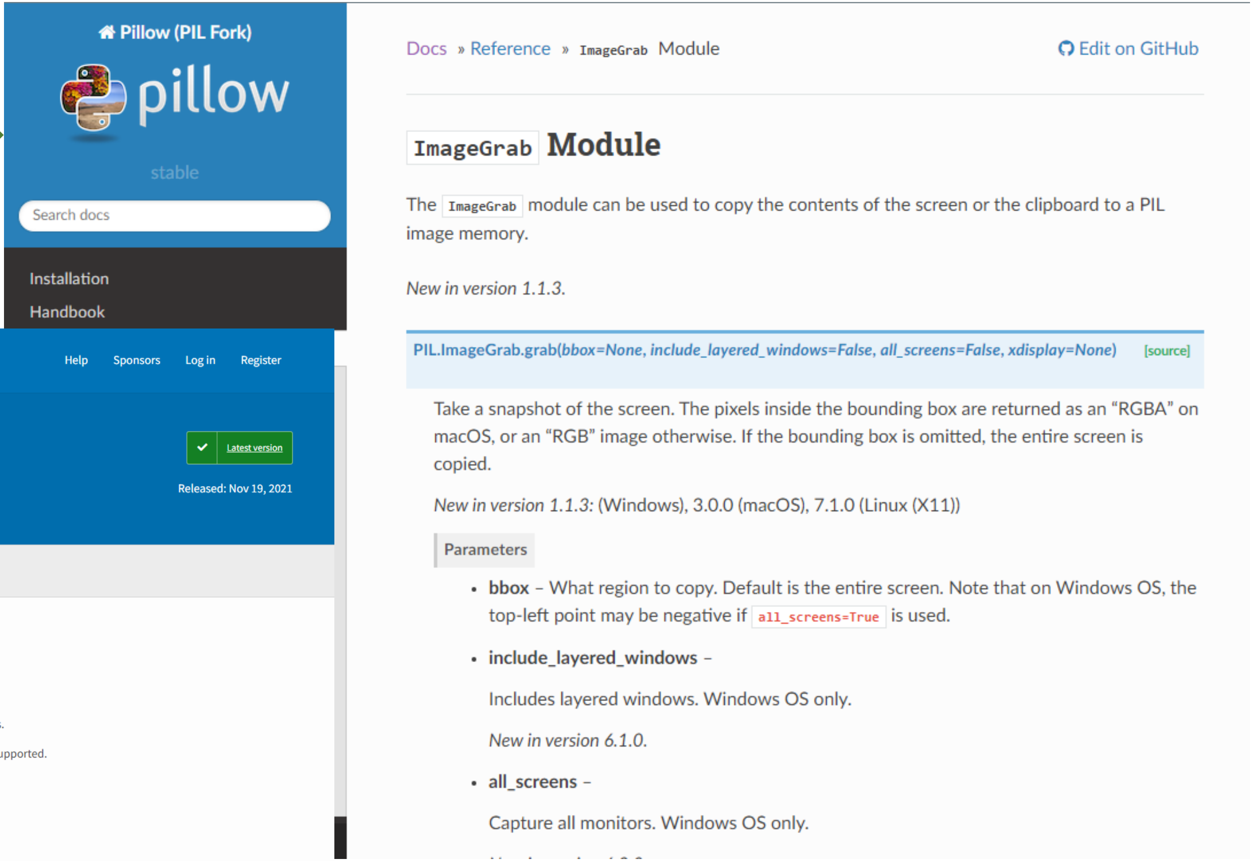
- Example – localization phase:

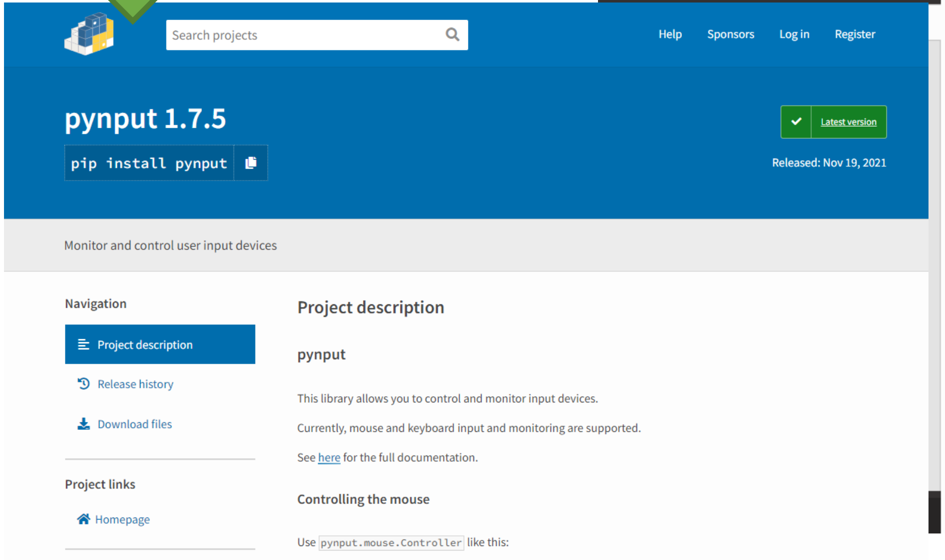


Template Matching

- taking screenshots :

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





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 :

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

mouse = Controller()
n_frame = 0
x=21
y=648
while(True):
    image_grab = ImageGrab.grab(bbox=None)
    image_grab_np = np.array(image_grab)
    cv.imwrite(f"out/{n_frame}.jpg", image_grab_np)
    print('The current pointer position is {0}'.format(mouse.position))
    mouse.position = (x, y)
    mouse.press(Button.left)
    mouse.release(Button.left)
    n_frame += 1
    time.sleep(5)
```