

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет программной инженерии и компьютерной техники

Мобильные системы компьютерного зрения

Лабораторная работа №2

Преподаватель: Денисов Алексей Константинович

**Выполнили: Демьяновский Савелий Игоревич,
Юров Максим Алексеевич,
Р4115**

**Санкт-Петербург
2024**

Содержание

Цель работы	3
Текст программы	3
Результаты работы	8
Вывод	8
Источники	8

Цель работы

Реализовать простейшие алгоритмы сопоставления изображений.

Текст программы

Программа состоит из нескольких файлов:

main.py:

```
if __name__ == "__main__":
    template_image_path = 'images/templates/inscription_temp.png'
    target_image_path = 'images/targets/inscription.png'

    template_img = cv2.imread(template_image_path, cv2.IMREAD_GRAYSCALE)
    target_img = cv2.imread(target_image_path, cv2.IMREAD_GRAYSCALE)

    template_result = template_matching(target_img, template_img)
    keypoint_result = keypoint_matching(target_img, template_img)

    plt.figure(figsize=(15, 5))

    plt.subplot(*args: 1, 3, 1)
    show_image(plt, cv2.cvtColor(template_result, cv2.COLOR_BGR2RGB), title: 'Template Matching')

    plt.subplot(*args: 1, 3, 2)
    show_image(plt, cv2.cvtColor(template_img, cv2.COLOR_BGR2RGB), title: 'Template image')

    plt.subplot(*args: 1, 3, 3)
    show_image(plt, cv2.cvtColor(keypoint_result, cv2.COLOR_BGR2RGB), title: 'Keypoints Matching')

    plt.show()
```

В этом файле получаем изображение, делаем его черно-белым, отправляем его в функции прямого поиска изображения и поиска ключевых точек (SIFT). После отображаем результаты (пример будет ниже).

template_matching.py:

```
def template_matching(img, template): 2 usages
    img_copy = img.copy()
    w, h = template.shape[:2]

    result = cv2.matchTemplate(img, template, cv2.TM_CCOEFF_NORMED)
    _, _, _, max_loc = cv2.minMaxLoc(result)

    top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)

    cv2.rectangle(img_copy, top_left, bottom_right, (0, 0, 0), 10)

💡 return img_copy
```

Здесь реализован наш алгоритм шаблонного сопоставления на изображении, с использованием метода нормальной корреляции. Функция cv2.matchTemplate ищет похожее на шаблон template место в изображении img и возвращает результат в виде матрицы, где каждый элемент указывает на некую степень совпадения шаблона с соответствующей областью исходного изображения. Метод cv2.TM_CCOEFF_NORMED используется для нормализации коэффициента корреляции, что делает процесс более устойчивым к изменениям освещенности.

Функция cv2.minMaxLoc используется для нахождения наименьшего и наибольшего значений в матрице result, а также их позиций. Здесь нас интересует max_loc — позиция наибольшего значения, которая указывает на лучшее совпадение шаблона с изображением.

Определяется верхний левый угол совпадения (top_left), и рассчитывается нижний правый угол (bottom_right) с учетом размеров шаблона. Эти координаты используются для очерчивания области совпадения на изображении.

keypoint_matching.py:

```
def keypoint_matching(img, template, min_matches=10):  2 usages
    sift = cv2.SIFT_create()

    kp1, des1 = sift.detectAndCompute(template, None)
    kp2, des2 = sift.detectAndCompute(img, None)

    if des1 is None or des2 is None: ...

    bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key=lambda x: x.distance)

    if len(matches) >= min_matches:
        src_pts = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
        dst_pts = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)

        M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
        if M is None:
            print("Homography computation failed.")
            return img

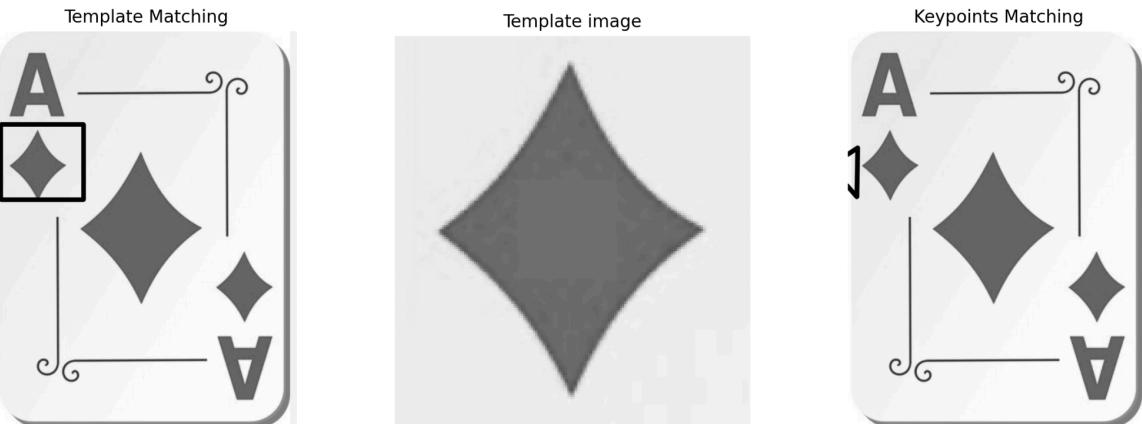
        h, w = template.shape
        pts = np.float32([[0, 0], [0, h - 1], [w - 1, h - 1], [w - 1, 0]]).reshape(-1, 1, 2)
        dst = cv2.perspectiveTransform(pts, M)

        img_with_template = cv2.polylines(img.copy(), [pts], isClosed=True, color=(0, 0, 0), thickness=10, cv2.LINE_AA)
        return img_with_template
    else:
        print(f"Not enough matches are found - {len(matches)}/{min_matches}")
        return img
```

Здесь мы с помощью SIFT ищем ключевые точки и вычисляем дескрипторы, далее сопоставляем дескрипторы с помощью BFMatcher, после сортируем совпадения - чем меньше расстояние, тем ближе совпадение. У нас есть порог в количестве совпадений, если их достаточно, то рисуем рамку.

Результаты





Template Matching



Template image



Keypoints Matching



Template Matching



Template image



Keypoints Matching



Template image



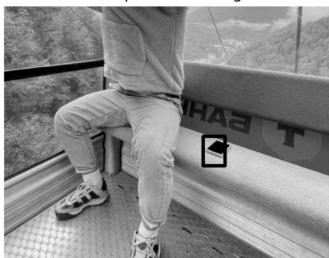
Template Matching



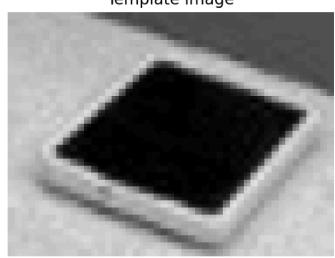
Keypoints Matching



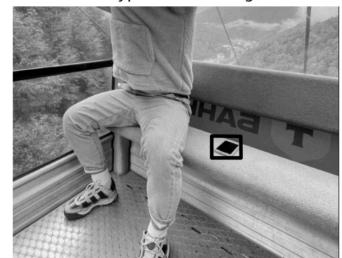
Template Matching



Template image



Keypoints Matching





Результаты работы

Как мы можем увидеть из этих 10 экспериментов, с большинством ситуаций справилось оба метода, keypoint matching, не считая осечки на втором эксперименте,правлялся лучше - он всегда находил и верно очерчивал искомый артефакт, в отличие от template matching, который также почти всегда находил нужное место, но часто неверно его выделял.

Насчет осечки в случае с картами: это случилось из-за маленького количества текстур.

Вывод

В результате выполнения работы мы реализовали простейшие алгоритмы по сопоставлению изображений и сравнили их. Так мы убедились в преимуществах сопоставления по ключевым точкам, но увидели недостатки подобных методах - они требуют достаточное количество текстур.

Источники

<https://habr.com/ru/articles/106302/>