МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра АПУ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Программирование систем реального времени»

Студенты гр. 2392 _____ Жук Ф.П.

Преподаватель _____ Гульванский В.В.

Санкт-Петербург

2025

Цель работы.

Вам представлен датасет с паттерном, который находится у вас в архиве. Его надо найти на картинке из датасета. Паттерн выглядит так:

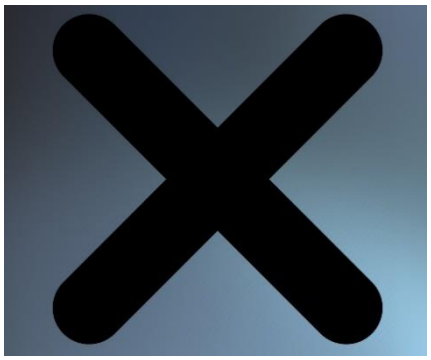

Рис. 1 - Патерн

1. Для изображений 1-100 по аналогии с поиском границ необходимо сделать матрицу, которая позволит найти этот паттерн.

2. Для всех остальных изображений необходимо найти любым, рассказанном на лекции, способом этот паттерн. Необходимо описать ваши подходы по нахождению этого паттерна и представить результаты в виде файла, как показано на скриншоте ниже:

Ход работы.

Для первых 300 изображений был выбран метод с поиском контуров с помощью библиотеки cv2.

Для поиска центров была написана функция, продемонстрированная на рисунке 1.

```python
def find_centers(img):

    # бинаризация изображения
    _, thresh = cv2.threshold(img, 40, 255, cv2.THRESH_BINARY)

    # поиск контуров
    # cv2.RETR_EXTERNAL - внешних контуров
    # cv2.CHAIN_APPROX_SIMPLE - упрощения контуров
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)


    centers = []
    for cnt in contours:
        # вычисление моментов контура
        M = cv2.moments(cnt)
        # вычисление центра
        if M['m00'] != 0:
            cx = int(M['m10'] / M['m00'])
            cy = int(M['m01'] / M['m00'])
            centers.append((cx, cy))

    return centers
```
✓ 0.0s

Рис. 1 – Функция поиска

Ход работы функции:

1. На вход продаться черно белое изображение (функция чтения cv2.imread(PATH_IMAGE, cv2.IMREAD_GRAYSCALE))

2. Далее проводиться бинаризация изображения (пример работы функции продемонстрирован на рисунке 2).

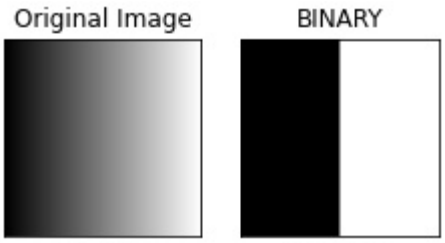

Рис. 2 - пример работы cv2.threshold с параметром cv2.THRESH_BINARY

3. Далее был выполнен поиск контуров на бинаризованом изображении. Он был выполнен с параметрами .
   Функция извлекает контуры из бинарного изображения с помощью алгоритма “ Topological Structural Analysis of Digitized Binary Images by Border Following”. Алгоритм сканирует изображение построчно в поисках граничного пикселя. Пиксель является граничным, если среди его 4-соседей есть хотя бы один фоновый

пиксель. При обнаружении пикселя, принадлежащего новой границе, запускается процедура трассировки. Алгоритм обходит границу по часовой стрелке для внешних границ объектов или против часовой стрелки для границ отверстий внутри объектов, запоминая координаты пикселей границы. Трассировка продолжается, пока алгоритм не вернется в стартовый пиксель.

Параметры:

- cv2.RETR_EXTERNAL – выделяет вешний контур;
- cv2.CHAIN_APPROX_SIMPLE - упрощает контур.

4. Далее рассчитывается момет изображения. Моменты изображения в компьютерном зрении — определённые средневзвешенные значения интенсивности пикселей изображения, или функция таких моментов. Для всех точек контура вычисляются моменты порядка (p+q). Для контура — суммирование по всем его точкам.

$$m_{pq} = \sum_{x} \sum_{y} x^p y^q$$

Где в итоге m_{00} - площадь объекта, m_{10} - суммы x, m_{00} - суммы y.

5. Зная площадь объекта, суммы x, суммы y рассчитываем центр объекта.

Пример работы кода (рис. 3).

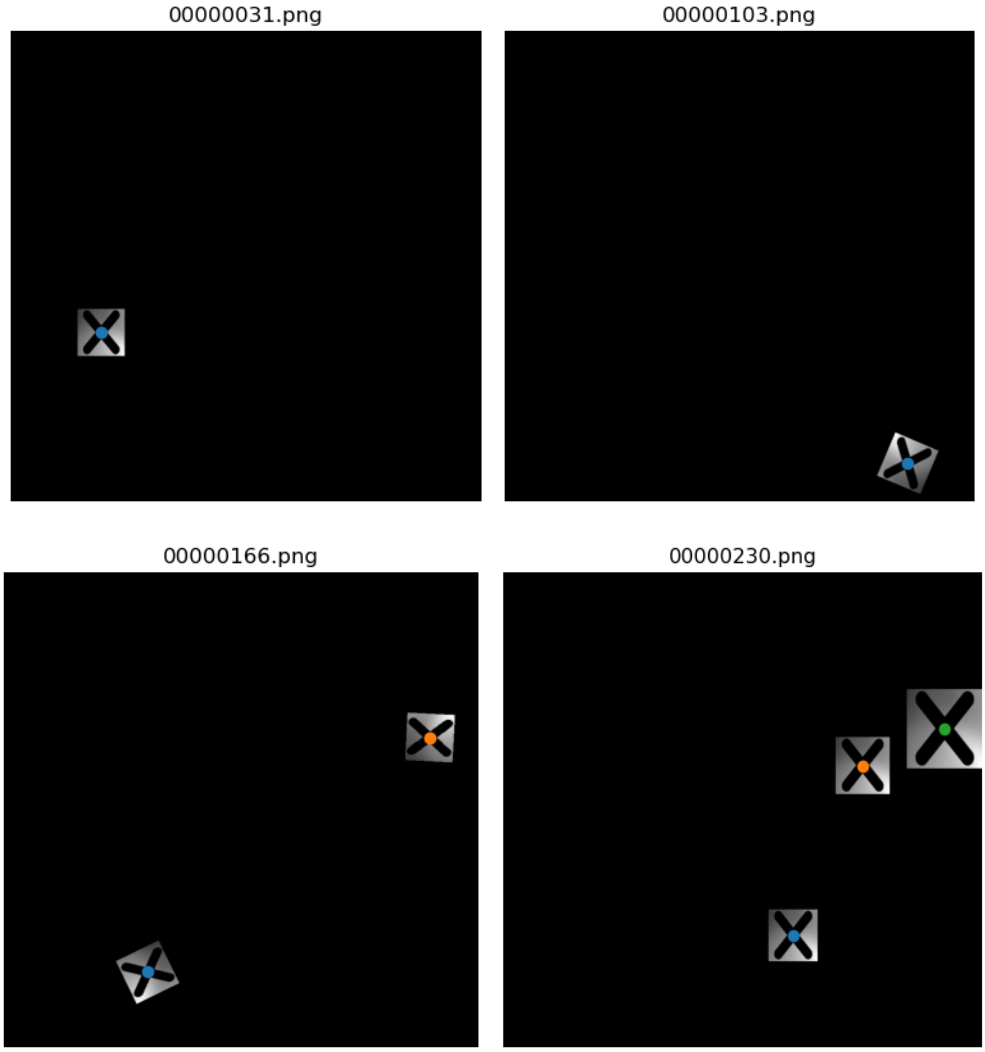

Рис. 3 - Пример работы

Для изображений 300–400 был выбран симбиоз алгоритмов.

Первый алгоритм – это свертка изображения с последующим нормированием (рис. 4), для фильтрации изображений.

```
def filter2D(img, kernel):
    img = cv2.filter2D(img, -1, kernel)
    cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

    return img
```

0.0s

Рис. 4 – Алгоритм свертки

Пример сверток продемонстрирован на рисунке 5.

Рис. 5 - Пример сверток

Для работы был выбран фильтр черного, он оставляет только пиксели, которые имею яркость 0. Данный фильтр упростит дальнейшую детекцию.

Второй алгоритм это SIFT. Это алгоритм для поиска и описания локальных особенностей на изображениях. Его основная задача — находить характерные точки, которые устойчивы к масштабированию, повороту и частично к изменению освещения.

Он работает по следующему алгоритму:

1. Поиск экстремумов в пространстве масштабов. Изображение размазывается с разными масштабами, ищутся точки, которые являются максимумом или минимумом среди соседей.
2. Определение точного положения ключевой точки. Для каждой найденной точки уточняется положение и отбрасываются слабые и неустойчивые точки, например: на краях.
3. Определение ориентации. Для каждой точки вычисляется направление градиента, чтобы сделать описание инвариантным к повороту.
4. Построение дескриптора. Вокруг каждой точки строится вектор признаков, который описывает локальную структуру изображения.

Для поиска совпадающих дескрипторов между пятерном и изображением, используется алгоритм BFMatcher. Это простой способ сопоставления дескрипторов между двумя изображениями. Он сравнивает каждый дескриптор первого изображения со всеми дескрипторами второго и находит наиболее похожие пары.

Дальше проходит отсеивание ложных совпадения и оставить только действительно похожие участки. Если таких участков меньше 4, то этот участок отсеивается.

Далее полученный участок подается в следующий алгоритм.

В качестве третьего алгоритма выступает cv2.matchTemplate (сегмент кода рис. 6). Это метод поиска и нахождения местоположения шаблонного изображения на участке изображения, для более точного нахождения положения объекта. Она просто скользит по шаблонному изображению и сравнивает шаблон и фрагмент входного изображения под шаблонным изображением.

```
# проходим по участкам изображения, где найдено совпадение
for pt in good_pts:

    # вычисление участка изображения для поиска шаблона
    x, y = int(pt[0]), int(pt[1])
    x1 = max(0, x - int(tw))
    y1 = max(0, y - int(th))
    x2 = min(img.shape[1], x + int(tw))
    y2 = min(img.shape[0], y + int(th))

    # проверка на выход за границы изображения
    roi = img[y1:y2, x1:x2]
    if roi.shape[0] < th or roi.shape[1] < tw:
        continue

    # сопоставление шаблона
    res = cv2.matchTemplate(roi, rotated_tmpl, cv2.TM_CCOEFF_NORMED)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

    if max_val > match_thresh:
        center_x = x1 + max_loc[0] + tw // 2
        center_y = y1 + max_loc[1] + th // 2

        # проверка на близость к уже найденным центрам
        centers.append((center_x, center_y)) if not any(np.sqrt((center_x - cx) ** 2 + (center_y - cy) ** 2) < 25 for cx, cy in centers) else None
```

Рис. 6 – Фрагмент кода

Таже для поиска изменённых размеров и случаев с поворотом изображения, для каждого этапа проводиться масштабирование и поворот шаблона.

Что позволяет применять данный код для каждого участка.

В итоге мы получаем центр нашего изображения.

Пример работы продемонстрирован на рисунке 7.

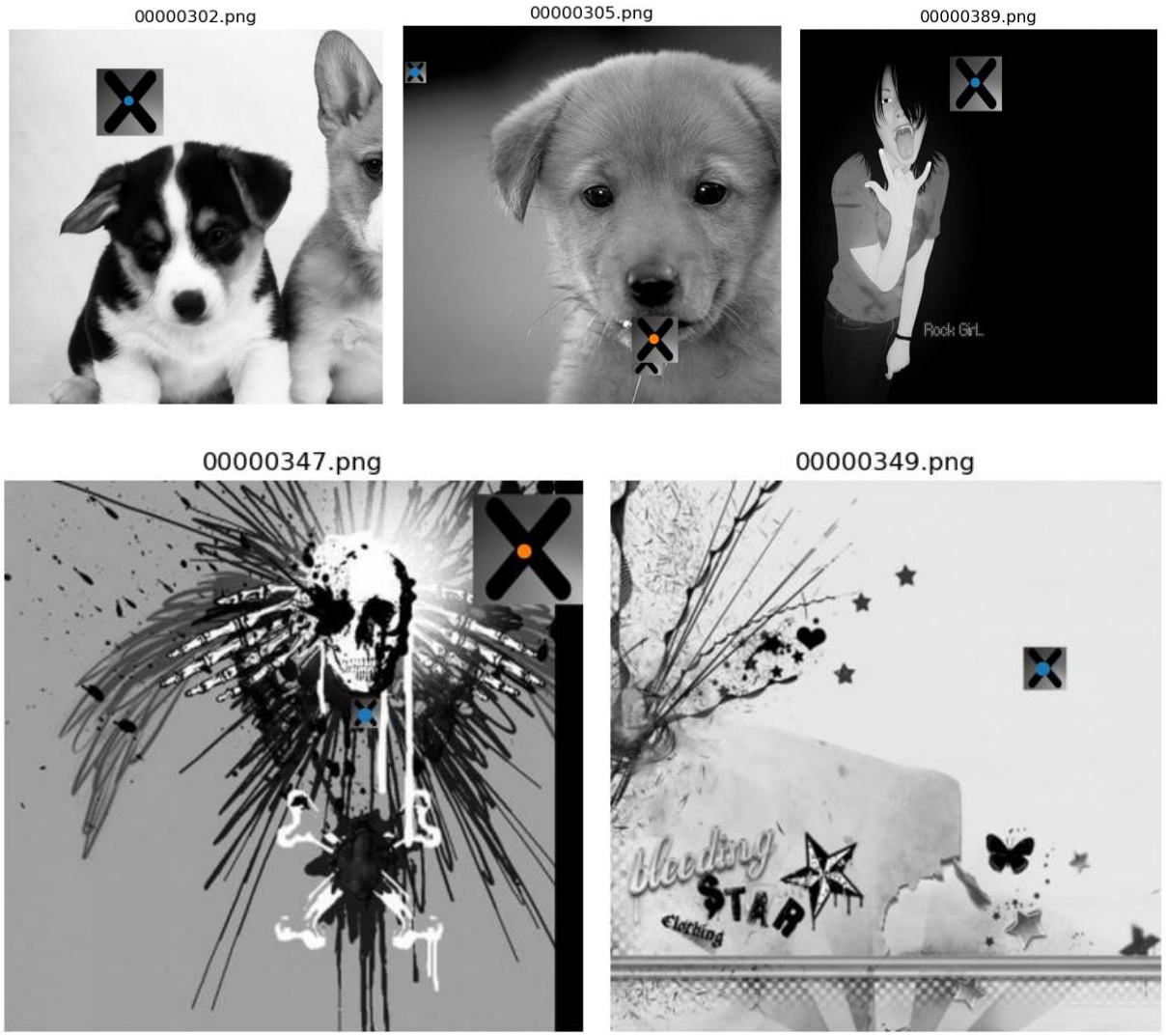

Рис. 7 – Результат работы кода

Выводы.

В ходе лабораторной работы было проведено исследование основных алгоритмов компьютерного зрения. Были определены центры паттернов на 400 изображениях. В ходе работы применялись корреляция с шаблоном, методы сопоставления ключевых точек SIFT, предобработка изображения различными фильтрами и созданной матрицей для выделения паттерна.

В процессе экспериментов были протестированы различные ядра свёртки для выделения характерных особенностей изображений, а также реализованы методы бинаризации и поиска контуров для автоматического определения центров объектов. Для повышения точности поиска паттернов использовались масштабирование шаблона и фильтрация ложных совпадений с помощью правила Лоу при сопоставлении дескрипторов SIFT.

Результаты работы были сохранены в виде таблицы с координатами найденных центров, что позволяет использовать полученные данные для дальнейшего анализа или обучения моделей машинного обучения. Проведённая работа показала эффективность комплексного подхода, сочетающего классические методы обработки изображений и современные алгоритмы поиска ключевых точек.