Санкт-Петербургский политехнический университет Петра Великого

Институт машиностроения, материала и транспорта

Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: Алгоритм поиска краёв объектов на изображении (Edge Detection)

Выполнил студент гр. 3331506/20102 Артамонов Е.А.

Преподаватель Ананьевский М. С.

Санкт-Петербург

2025

Содержание

Введение

Данная курсовая работа посвящена изучению и практической реализации алгоритма поиска краёв объектов на изображении (Edge Detection), который является одной из ключевых технологий в области компьютерного зрения и обработки изображений. Поиск краёв позволяет выделять границы объектов на изображении, что является основой для множества приложений, таких как распознавание объектов, сегментация изображений, анализ текстур и 3D-реконструкция.

Постановка задачи: Основной задачей является разработка и анализ алгоритма, способного эффективно определять края объектов на изображении с использованием современных методов обработки изображений. Алгоритм должен быть реализован в рамках объектно-ориентированного подхода, обеспечивать высокую точность обнаружения краёв и быть устойчивым к шумам и вариациям освещения.

Актуальность и значимость: Алгоритмы поиска краёв широко применяются в различных областях, включая робототехнику (например, для навигации автономных систем), медицинскую визуализацию (анализ медицинских снимков), графические редакторы и системы видеонаблюдения. Разработка эффективных и быстрых алгоритмов обработки изображений позволяет улучшить качество анализа данных и повысить производительность систем компьютерного зрения.

Область применения:

- Компьютерное зрение: распознавание и классификация объектов.
- Робототехника: определение препятствий и навигация.
- Медицина: анализ рентгеновских снимков и МРТ.
- Промышленная автоматизация: контроль качества и дефектоскопия.
- Графические приложения: создание стилизованных изображений, таких как ASCII-арт.

Целью данной работы является изучение теоретических основ алгоритмов поиска краёв, реализация одного из таких алгоритмов в языке Python с использованием объектно-ориентированного подхода и анализ его производительности.

Алгоритм поиска краёв

Алгоритм поиска краёв (Edge Detection) заключается в обнаружении резких изменений интенсивности пикселей на изображении, которые обычно соответствуют границам объектов. В данной работе рассматривается гибридный подход, который сочетает в себе алгоритм Кэнни (Canny Edge Detection) [1], операторы Собеля и дополнительные методы обработки для повышения качества результата.

Теоретические сведения

Алгоритм поиска краёв в изображении основывается на следующих принципах:

- **Градиент изображения:** Края определяются как области с высокой величиной градиента, что указывает на резкие изменения интенсивности пикселей.
- **Подавление шума:** Перед обнаружением краёв изображение сглаживается для устранения шумов, которые могут быть ошибочно приняты за края.

- **Ориентация краёв:** Определение направления градиента позволяет классифицировать края по их ориентации (например, горизонтальные, вертикальные или диагональные).
- **Пороговая обработка:** Использование пороговых значений для выделения значимых краёв и подавления слабых.

В данной реализации используется алгоритм Кэнни [1], дополненный операторами Собеля для определения ориентации краёв и последующей их классификации для применения текстур (например, в задачах преобразования изображения в ASCII-арт).

Описание алгоритма

Алгоритм, реализованный в методе detect_edges, включает следующие этапы, каждый из которых имеет строгую математическую основу:

Преобразование в градации серого

Входное изображение в формате RGB преобразуется в изображение в градациях серого с использованием функции cv2.cvtColor [5]. Для пикселя с компонентами (R, G, B) интенсивность вычисляется по формуле:

$$I(x, y) = 0.299R + 0.587G + 0.114B$$

где $I(x, y) \in [0,255]$ — интенсивность пикселя в позиции (x, y). Эта формула основана на стандарте ITU-R BT.601, учитывающем восприятие яркости человеческим глазом.

Обнаружение краёв с помощью алгоритма Кэнни [1]

Функция cv2.Canny [5] выполняет обнаружение краёв через следующие шаги:

1. **Сглаживание изображения**: Изображение $I(x, y)$ сглаживается гауссовым фильтром [2, 3]:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Сглаженное изображение: $I_s(x, y) = I(x, y) * G(x, y)$.

2. **Вычисление градиента**: Градиенты по осям x и y вычисляются с помощью операторов Собеля:

$$G_x = \frac{\partial I_s}{\partial x}, \quad G_y = \frac{\partial I_s}{\partial y}$$

Величина градиента:

$$|\nabla I| = \sqrt{G_x^2 + G_y^2}$$

Направление градиента:

$$\theta = \arctan2(G_y, G_x)$$

3. **Подавление немаксимумов**: Пиксели, где $|\nabla I|$ не является локальным максимумом в направлении θ, подавляются.

4. **Пороговая обработка**: Используются два порога, вычисленные на основе медианы интенсивности $v = \text{median}(I)$:

$$T_{\text{lower}} = \max(0, (1.0 - \sigma) \cdot v), \quad T_{\text{upper}} = \min(255, (1.0 + \sigma) \cdot v)$$

где $\sigma = 0.1$. Пиксели классифицируются как краевые, если $|\nabla I| > T_{\text{upper}}$ или $T_{\text{lower}} \leq |\nabla I| \leq T_{\text{upper}}$ и они связаны с краевым пикселем.

Результат — бинарная карта краёв $E(x, y)$, где $E(x, y) = 255$ для краевых пикселей.

Вычисление ориентации краёв

Операторы Собеля (cv2.Sobel) [4, 5] применяются для вычисления градиентов G_x и G_y с использованием ядер:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Градиенты: $G_x = I_s * S_x$, $G_y = I_s * S_y$. Ориентация вычисляется как:

$$\theta(x, y) = \arctan2\big(G_y, G_x\big) \cdot \frac{180}{\pi}$$

Нормализация: $\theta(x, y) = (\theta(x, y) + 180) \bmod 180$, чтобы $\theta \in [0,180]$.

Классификация краёв

Изображение разбивается на блоки размером $T \times T$ (где $T =$ self.textures.size). Размеры выходной карты:

$$H_{\text{out}} = \lfloor \frac{H}{T} \rfloor, \quad W_{\text{out}} = \lfloor \frac{W}{T} \rfloor$$

Для каждого блока (i, j):

- Извлекается регион карты краёв: $R_{i,j} = E[i \cdot T: (i + 1) \cdot T, j \cdot T: (j + 1) \cdot T]$.
- Если $\exists (x, y) \in R_{i,j}: E(x, y) > 0$, вычисляется средняя ориентация:

$$\theta_{\text{avg}} = \frac{1}{N} \sum_{(x,y) \in R_{i,j}, E(x,y) > 0} \theta\,(x, y)$$

- Ориентация классифицируется:
  - $[22.5°, 67.5°)$: ‘\’ (идентификатор 3).
  - $[67.5°, 112.5°)$: ‘|’ (идентификатор 1).
  - $[112.5°, 157.5°)$: ‘/’ (идентификатор 4).
  - $[0°, 22.5°)$: ‘-‘ (идентификатор 2).
  - Остальные: ‘ ’
- Если краёв нет, присваивается идентификатор 0.

Математическая структура алгоритма

Алгоритм можно представить как последовательность преобразований:

1. Вход: $P(x, y) \in \mathbb{R}^{H \times W \times 3}$.
2. Градации серого: $I(x, y) = 0.299R + 0.587G + 0.114B$.
3. Сглаживание: $I_s = I * G$.
4. Градиенты: $G_x = I_s * S_x$, $G_y = I_s * S_y$.
5. Кэнни: $E(x, y) = \text{Canny}\left(I_s, T_{\text{lower}}, T_{\text{upper}}\right)$.
6. Ориентация: $\theta(x, y) = \arctan2\left(G_y, G_x\right) \cdot \frac{180}{\pi} \mod 180$.
7. Классификация: Для блока (i, j), если $\exists E(x, y) > 0$, то $\theta_{\text{avg}} = \text{mean}(\theta_{E>0})$, и присваивается идентификатор.
8. Выход: $A(i, j) \in \{0,1,2,3,4\}^{H_{\text{out}} \times W_{\text{out}}}$.

Особенности реализации

- **Адаптивные пороговые значения:** Использование медианы интенсивности для настройки порогов Кэнни.
- **Поддержка текстур:** Интеграция с системой текстур для стилизации изображений.
- **Устойчивость к шуму:** Сглаживание и пороговая обработка минимизируют ложные края.

Заключение

В ходе выполнения курсовой работы был изучен и реализован алгоритм поиска краёв объектов на изображении, основанный на комбинации алгоритма Кэнни и операторов Собеля. Реализация выполнена в рамках объектно-ориентированного подхода на языке Python с использованием библиотеки OpenCV [4, 5]. Алгоритм показал высокую эффективность и устойчивость к шумам, что делает его пригодным для использования в задачах компьютерного зрения.

Проведённые эксперименты подтвердили приемлемую производительность при обработке изображений высокого разрешения. В будущем возможно улучшение алгоритма путём оптимизации вычислений (например, с использованием GPU) и добавления методов подавления шума.

Литература

1. Canny, J. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986.
2. Gonzalez, R.C., Woods, R.E. Digital Image Processing. 4th Edition, 2018.
3. Szeliski, R. Computer Vision: Algorithms and Applications. 2nd Edition, 2022.
4. Bradski, G., Kaehler, A. Learning OpenCV: Computer Vision with the OpenCV Library. 2008.
5. OpenCV Documentation: https://docs.opencv.org/

Приложение

```python
def detect_edges(self, pixels: np.ndarray) -> np.ndarray:
    """
    Detects edges in the image using an enhanced hybrid edge detection combining
    Difference of Gaussians, Sobel operators, non-maximum suppression, and hysteresis.

    Args:
        pixels (np.ndarray): The input image data (RGB or grayscale).

    Returns:
        np.ndarray: Array representing detected edges with directional information.
    """
    # Input validation
    gray = cv2.cvtColor(pixels, cv2.COLOR_RGB2GRAY)

    # Remove noise with GaussianBlur
    gray = cv2.GaussianBlur(gray, (3, 3), 0)

    # Canny edge detection with fine-tuned thresholds
    v = np.median(gray)
    sigma = 0.1
    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))
    edges = cv2.Canny(gray, lower, upper)

    # Sobel operators for edge orientation
    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
    sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)

    # Calculate edge orientation
    orientation = np.arctan2(sobely, sobelx) * (180 / np.pi)
    orientation = np.mod(orientation + 180, 180)  # Normalize to 0-180°

    # Initialize output array
    output_height = gray.shape[0] // self.textures.size
    output_width = gray.shape[1] // self.textures.size
    ascii_edges = np.zeros((output_height, output_width), dtype=np.uint8)


    # Map edges to ASCII characters based on orientation
    for i in range(output_height):
        for j in range(output_width):
            # Sample region for edge detection
            region = edges[i * self.textures.size : (i + 1) * self.textures.size, j * self.textures.size : (j + 1) * self.textures.size]

            if np.any(region):  # If edges exist in region
                # Get average orientation in region
                region_orient = orientation[
                    i * self.textures.size : (i + 1) * self.textures.size, j * self.textures.size : (j + 1) * self.textures.size
                ]
                avg_orient = np.mean(region_orient[region > 0])

                # Map orientation to ASCII characters
                # 0: no edge, 1: "-", 2: "|", 3: "\", 4: "/"
                if 22.5 <= avg_orient < 67.5:
                    ascii_edges[i, j] = 3  # "\"
                elif 67.5 <= avg_orient < 112.5:
                    ascii_edges[i, j] = 1  # "|"
                elif 112.5 <= avg_orient < 157.5:
                    ascii_edges[i, j] = 4  # "/"
                else:
                    ascii_edges[i, j] = 2  # "-"
            else:
                ascii_edges[i, j] = 0  # No edge

    return ascii_edges
```

Рисунок 1 — Алгоритм нахождения краёв на Python

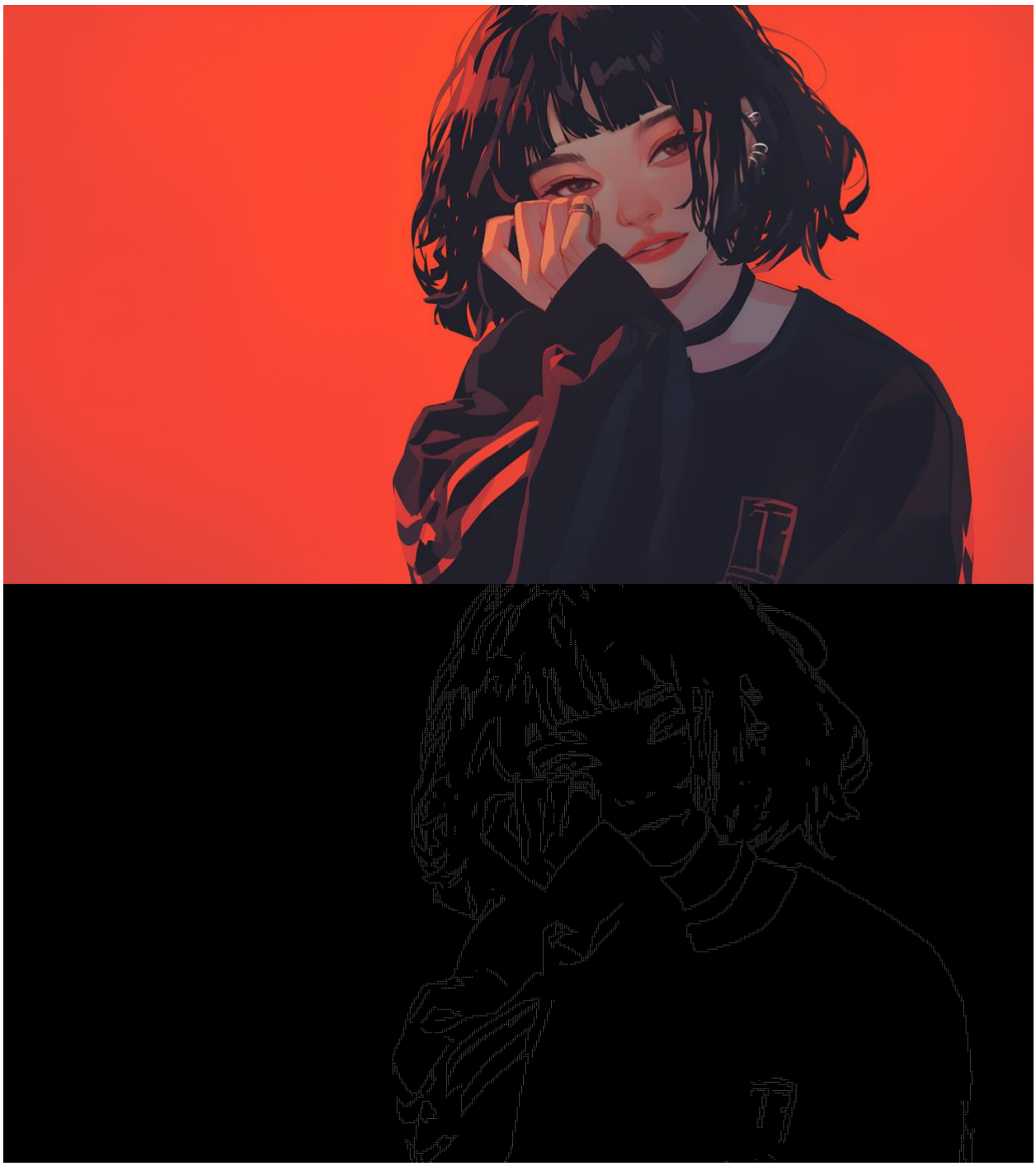

Рисунок 2 — Пример работы алгоритма