Метод предобработки изображения кисти руки в системе распознавания жестовых символов

Танцевов Г. М.[a]

[a] *МГТУ им. Н. Э. Баумана, Москва, Россия*

Аннотация

Здесь дается краткая аннотация, характеризующая главную проблематику исследования, а также наиболее значимые полученные результаты.

Ключевые слова: ключевое слово 1, ключевое слово 2, ключевое слово 3

1. Введение

Одной из перспективных задач машинного зрения является распознавание жестовых символов. Актуальность данной темы обусловлена множеством сфер применения: от новых методов взаимодействия с ПК до систем распознавания жестовых языков. Как правило, такие системы состоят из трех частей:

1. Получение данных о жесте
2. Предобработка данных
3. Классификация

В качестве исходных данных можно использовать снимок с камеры, например, смартфона. В этом случае, для упрощения классификации, важен этап предобработки данных. На данном этапе необходимо выделить на изображении основные признаки исходного жеста. Алгоритмы, применимые для достижения данной цели, можно разделить на следующие группы:

- Выделение контура фигуры
- Выделение силуэта кисти руки
- Построение скелета кисти руки

2. Выделение контура фигуры

Для выделения контура кисти руки можно использовать операторы преобразования изображения. К таким методам можно отнести:

Электронная почта: email_tantsevov@gmail.com (Танцевов Г. М.)

- Оператор Собеля
- Оператор Прюитт
- Перекрестный оператор Робертса
- Оператор Кэнни

Рассмотрим каждый подробнее:

2.1. Оператор Собеля

Основная идея оператора Собеля[1] заключается в вычислении градиента освещенности каждой точки изображения. Вычисление производится примерное с помощью свертки изображения двумя сепарабельными целочисленными фильтрами размера 3x3 в вертикальном и горизонтальном направлениях. Благодаря этому вычисление работа данного оператора имеет низкие трудозатраты. В результате получаются два новых изображения Gx и Gy, в каждой точке которого записано приближенное значение производных по x и по y соответственно. Пусть A - исходное изображение, тогда вычисляются они следующим образом:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{1}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2}$$

Определение данных матриц на языке Python выглядит следующим образом:

```
sobelx = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]
sobely = [[1, 2, 1], [0, 0, 0], [-1, -2, -1]]
```

Алгоритм 1: Определение матриц свертки оператора Собелья

В итоге значение градиента вычисляется как $G = \sqrt{G_x^2 + G_y^2}$, а его направление как $\theta = \arctan(\frac{G_x}{G_y})$.

Для получения результата применем матрицы к изображению:

```
for row in range(self.width-len(sobelx)):
    for col in range(self.height-len(sobelx)):
        gx = 0
        gy = 0
        for i in range(len(sobelx)):
            for j in range(len(sobely)):
                val = mat[row+i, col+j] * lin_scale
                gx += sobelx[i][j] * val
                gy += sobely[i][j] * val
        pixels[row+1, col+1] = int(math.sqrt(gx*gx + gy*gy))
```

Алгоритм 2: Свертка изображения оператором Собеля

Результат показывает скорость изменения яркости изображения в конкретной точке, т.е. вероятность ее нахождения на границе изображения.

2.2. Оператор Прюитт

Принцип работы[2] аналогичен оператору Собеля, используются только другие ядра свертки:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{3}$$

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{4}$$

Отсюда реализация данного оператора отличается от реализации оператора Собеля только определением матриц:

```
prewittx = [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]]
prewitty = [[1, 1, 1], [0, 0, 0], [-1, -1, -1]]
```

Алгоритм 3: Определение матриц свертки оператора Прюитт

2.3. Перекрестный оператор Робертса

Данный алгоритм[3] для каждого пиксела вычисляет сумму квадратов разниц со смежным ему диагональным пикселем. Данную операцию можно представить в виде свертки изображения двумя ядрами размера 2x2:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{5}$$

Подход данного метода не сильно отличается от двух рассмотренных ранее, в следствии чего его реализация выглядит следующим образом:

```
robertsx = [[1,0],[0,-1]]
robertsy = [[0,1],[-1,0]]
lin_scale = .7
for row in range(self.width-len(robertsx)):
    for col in range(self.height-len(robertsy)):
        Gx = 0
        Gy = 0
        for i in range(len(robertsx)):
            for j in range(len(robertsy)):
                val = mat[row+i, col+j] * lin_scale
                Gx += robertsx[i][j] * val
                Gy += robertsy[i][j] * val
        pixels[row+1,col+1] = int(math.sqrt(Gx*Gx + Gy*Gy))
```

Алгоритм 4: Реализация перекрестного оператора Робертса

В результате получается изображение пространственного градиента исходного изображения, где точки с наибольшим значением соответствуют границе.

2.4. Оператор Кэнни

Данный фильтр[4] был разработан с учетом удовлетворения следующих свойств:

- хорошее обнаружение (Кэнни трактовал это свойство как повышение отношения сигнал/шум);
- хорошая локализация (правильное определение положения границы);
- единственный отклик на одну границу.

Алгоритм состоит из пяти последовательных шагов:

1. Размытие изображения для удаления лишнего шума.
2. Поиск градиентов, для определения границ с максимальным значением градиента.
3. Подавление не-максимумов, т.е. для границ берутся исключительно локальные максимумы.
4. Определение потенциальных границ с помощью двойной пороговой фильтрации.
5. Трассировка области неоднозначности

Для получения размытого изображения можно использовать стандартную функцию *convolve*() из пакета $scipy.ndimage.filters$. Для поиска градиентов подойдет оператор Собеля, описанный выше.

Реализация подавления не-максимумов:

```
def non_max_suppression(img, D):
    M, N = img.shape
    Z = np.zeros((M, N), dtype=np.int32)
    angle = D * 180. / np.pi
    angle[angle < 0] += 180
    for i in range(1, M - 1):
        for j in range(1, N - 1):
            try:
                q = 255
                r = 255
                # angle 0
                if (0 <= angle[i, j] < 22.5) or
                (157.5 <= angle[i, j] <= 180):
                    q = img[i, j + 1]
                    r = img[i, j - 1]
                # angle 45
                elif 22.5 <= angle[i, j] < 67.5:
                    q = img[i + 1, j - 1]
                    r = img[i - 1, j + 1]
                # angle 90
                elif 67.5 <= angle[i, j] < 112.5:
                    q = img[i + 1, j]
                    r = img[i - 1, j]
                # angle 135
                elif 112.5 <= angle[i, j] < 157.5:
                    q = img[i - 1, j - 1]
                    r = img[i + 1, j + 1]
                if (img[i, j] >= q) and (img[i, j] >= r):
                    Z[i, j] = img[i, j]
                else:
                    Z[i, j] = 0
            except IndexError as e:
                pass
    return Z
```

Алгоритм 5: Функция подавления не-максимумов

Реализация опредления потенциальных границ:

```
def threshold(self, img):
    highThreshold = img.max() * self.highThreshold
    lowThreshold = highThreshold * self.lowThreshold
    M, N = img.shape
    res = np.zeros((M, N), dtype=np.int32)
    weak = np.int32(self.weak_pixel)
    strong = np.int32(self.strong_pixel)
    strong_i, strong_j = np.where(img >= highThreshold)
    zeros_i, zeros_j = np.where(img < lowThreshold)
    weak_i, weak_j = np.where((img <= highThreshold)
        & (img >= lowThreshold))
    res[strong_i, strong_j] = strong
    res[weak_i, weak_j] = weak
    return res
```

Алгоритм 6: Функция определения потенциальных

Финальным шагом является трассировка областей неоднозначности:

```
def hysteresis(self, img):
    M, N = img.shape
    weak = self.weak_pixel
    strong = self.strong_pixel
    for i in range(1, M - 1):
        for j in range(1, N - 1):
            if img[i, j] == weak:
                try:
                    if ((img[i + 1, j - 1] == strong)
                        or (img[i + 1, j] == strong)
                        or (img[i + 1, j + 1] == strong)
                        or (img[i, j - 1] == strong)
                        or (img[i, j + 1] == strong)
                        or (img[i - 1, j - 1] == strong)
                        or (img[i - 1, j] == strong)
                        or (img[i - 1, j + 1] == strong)):
                        img[i, j] = strong
                    else:
                        img[i, j] = 0
                except IndexError as e:
                    pass
    return img
```

Алгоритм 7: Трассировка области неоднозначности

Стоит обратить внимание, что описанные выше методы принимают на вход изображение в серых тонах. То есть нулевым шагом данных методов можно указать преобразование изображения из цветного в черно-белое.

3. Выделение силуэта кисти руки

Помимо классических методов определения границ можно использовать сегментациб по цвету кожи [5]. Данный метод преобразует RGB изображение в бинарное с помощью фильтрации пикселей по цвету, близкому к цвету кожи. Для улучшения работы алгоритма перед фильтрацией изображение переводят в цветовое пространство YCrCb, в котором различные цвета кожи расположены близко друг к другу [6].

Пример фильтрации изображения:

```
import math

for i in range(0, hand.shape[0]):
    for j in range(0, hand.shape[1]):
        if (math.sqrt(
                (int(imgYCC[i, j, 0]) - avg_skin_color[0]) ** 2 +
                (int(imgYCC[i, j, 1]) - avg_skin_color[1]) ** 2 +
                (int(imgYCC[i, j, 2]) - avg_skin_color[2]) ** 2) <= error):
            imgYCC[i, j] = [255, 255, 255]
        else:
            imgYCC[i, j] = [0, 0, 0]
```

Алгоритм 8: Фильтрация изображения по цвету кожи

В большинстве случаев после бинаризации на изображении присутствуют шумы и артефакты, вызванные тем, что на фоновой части изображения находились пиксели, попадающие в ограничения фильтра. Для их устранения можно использовать морфологические операции "расширение"и "сужение"[7]:

```
kernel = np.ones((7, 7), np.uint8)
imgYCC = cv2.dilate(imgYCC, kernel, iterations=1)
imgYCC = cv2.erode(imgYCC, kernel, iterations=1)
```

Алгоритм 9: Применение к изображению операций расширение и сужение

4. Построение скелета кисти руки

Для ускорения процесса классификации жеста руки можно использовать скелетную модель. Данный тип входных данных в силу своей специфики может упростить вычисление признаков, необходимых классификатору.

Для построения скелета кисти можно использовать метод построения скелета выпуклой фигуры [7].

В качестве выпуклой фигуры можно использовать результат работы метода, описанного в разделе 3.

В данном методе предлагается поиск скелета с помощью морфологической операции "сужение". Данная операция применяется до тех пор, пока последующие применение не приведет к очищению изображения. Пример реализации:

```
img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2GRAY)
size = np.size(img)
skel = np.zeros(img.shape, np.uint8)

ret, img = cv2.threshold(img, 127, 255, 0)
element = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
done = False

while not done:
    eroded = cv2.erode(img, element)
    temp = cv2.dilate(eroded, element)
    temp = cv2.subtract(img, temp)
    skel = cv2.bitwise_or(skel, temp)
    img = eroded.copy()
    zeros = size - cv2.countNonZero(img)
    if zeros == size:
        done = True
skel = cv2.cvtColor(skel, cv2.COLOR_GRAY2BGR)
for i in range(0, skel.shape[0]):
# looping through the rows( height)
    for j in range(0, skel.shape[1]):
# looping through the columns(width)
        if (skel[i, j] == [255, 255, 255]).all():
            skel[i, j] = [255, 0, 0]
            orig_img[i, j] = [255, 0, 0]
```

Алгоритм 10: Фильтрация изображения по цвету кожи

Проблемой данного метода являются побочные ветви скелета, образованные из-за возможной зашумленности или неточности фигуры. Другим недостатком можно считать отсутствие гарантии обеспечения связного набора пикселей для всего скелета, или обеспечения одинаковой ширины ветвей во всем скелете.

Скелет кисти можно построить на основании ключевых точек, получаемых с помощью нейронной сети [8]. Данная нейронная сеть определяет на изображении 22 ключевых точки, 21 из которых относятся к кисти руки, а 22 отмечает фон. Пример расположения точек представлен на рисунке 1.

Далее для построения скелета необходимо соединить полученные точки в последовательностях, описанной в таблице 1.

5. Методология

В этом разделе описывается методология исследования.

5.1. Название параграфа

Пример простой формулы:

$$f(x) = ax + b \tag{6}$$

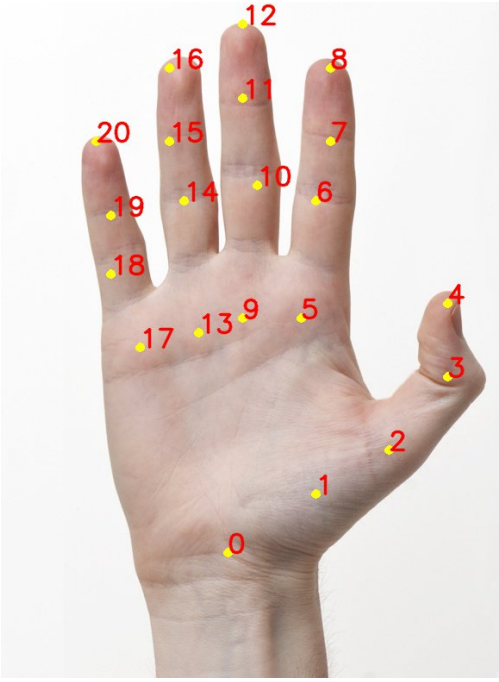

Рис. 1: Ключевые точки кисти руки

Таблица 1: Сравнение алгоритмов выделения источников

Ветвь скелета	Последовательность точек
Большой палец	$0 \to 1 \to 2 \to 3 \to 4$
Указательный палец	$0 \to 5 \to 6 \to 7 \to 8$
Средний палец	$0 \to 9 \to 10 \to 11 \to 12$
Безымянный палец	$0 \to 13 \to 14 \to 15 \to 16$
Мизинец	$0 \to 17 \to 18 \to 19 \to 20$

5.2. Название параграфа

Пример более сложной формулы:

$$f_{GND}(0, \sigma, k) = \frac{\phi(y)}{\sigma - kP_t}, \tag{7}$$

$$y = \begin{cases} -\frac{1}{k}\log\left[1 - \frac{kP_t}{\sigma}\right], & k \neq 0 \\ \frac{P_t}{\sigma}, & k = 0 \end{cases},$$

6. Данные

В этом разделе приводится описание использованных в работе данных, обосновывается их выбор, а также указываются источники их получения. Общераспространенной практикой является анализ описательной статистики, позволяющей сделать какие-то предположения еще до получения результатов исследования.

7. Результаты

В данном разделе приводятся основные полученные в работе результаты, а также выполняется их детальный анализ.

7.1. Название параграфа

Таблица 2: Пример простой таблицы, содержащей описательную статистику.

Параметр	Название колонки	Название колонки
Среднее, μ	0.79	0.98

Пояснения: Здесь даются пояснения к таблице.

7.2. Название параграфа

Таблица 3: Пример более сложной таблицы, содержащей оценки параметров модели.

Параметр	*Название колонки*	*Название колонки*	*Название колонки*
Группа 1			
μ	0.30^{***} (0.01)	0.30^{***} (0.01)	0.30^{***} (0.01)
ϕ	0.30^{***} (0.01)	0.30^{***} (0.01)	0.30^{***} (0.01)
Группа 2			
μ	0.40^{*} (0.17)	0.40^{*} (0.17)	0.40^{*} (0.17)
ϕ	0.40^{*} (0.17)	0.40^{*} (0.17)	0.40^{*} (0.17)

Пояснения: В скобках приведены стандартные ошибки параметров. Уровни значимости: *** – 1%, ** – 5%, * – 10%.

7.3. Название параграфа

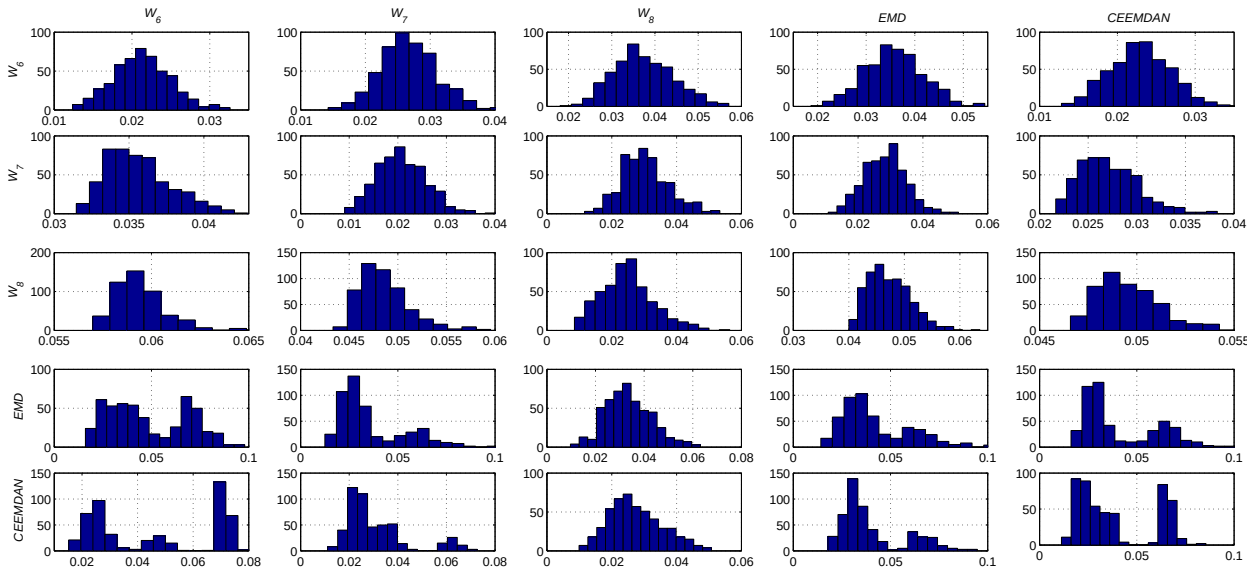

Рис. 2: Название рисунка.

8. Заключение

В данном разделе обобщаются полученные результаты и делаются основные выводы исследования.

9. Благодарности

Раздел содержит благодарности людям или организациям, которые оказали существенную помощь различного характера при написании работы.

В качестве базового, можно использовать следующий шаблон:

Авторы выражают благодарность {ФИО} за ценные комментарии по содержанию работы. Ответственность за все ошибки и неточности, допущенные авторами, лежит исключительно на них самих.

ПриложениеА. Название приложения

Здесь размещается содержимое приложений к работе. В общем случае их может быть более одного.

Список литературы

[1] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

[2] Judith M. S. Prewitt. Object enhancement and extraction picture processing and psychopictorics. *Academic*, pages 75–149, 1970.

[3] Lawrence Roberts. *Machine Perception of Three-Dimensional Solids*. 01 1963.

[4] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8:679 – 698, 12 1986.

[5] S.L. Phung, Abdesselam Bouzerdoum, and Douglas Chai. Skin segmentation using color and edge information. pages 525 – 528 vol.1, 08 2003.

[6] Joshi Siddharth and Srivastava Gaurav. Face detection. *E368: Digital Image Processing*, pages 101 – 112, 2003.

[7] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson Prentice Hall, third edition, 2008.

[8] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. 04 2017.