Лабораторная работа №7
Морфологический анализ изображений

Цель работы

Освоение принципов математической морфологии в области обработки и анализа изображений.

Методические рекомендации

До начала работы студенты должны ознакомиться с функциями среды MATLAB или библиотеки OpenCV для работы с бинарной морфологией и с основными операциями и положениями бинарной морфологии. Лабораторная работа рассчитана на 4 часа.

Теоретические сведения

Термин *морфология* дословно переводится как «наука о форме». Морфологический анализ используется во многих областях знаний, в т.ч. и в обработке изображений. Морфологический анализ является относительно универсальным подходом, поэтому стоит обособленно от всех рассмотренных ранее методов. С использованием математической морфологии при обработке изображений можно осуществлять фильтрацию шумов, сегментацию объектов, выделение контуров, реализовать поиск заданного объекта на изображении, вычислить «скелет» образа и т.д. Рассмотрим основные операции бинарной морфологии над изображением A структурным элементом B:

1. Дилатация (расширение, наращивание): $A \oplus B$, в MATLAB выполняется функцией `imdilate(A,B)`, расширяет бинарный образ A структурным элементом B;

2. Эрозия (сжатие, сужение): $A \ominus B$, в MATLAB выполняется функцией `imerode(A,B)`, сужает бинарный образ A структурным элементом B;

3. Открытие (отмыкание, размыкание, раскрытие): $(A \ominus B) \oplus B$, в MATLAB выполняется функцией `imopen(A,B)`, удаляет внешние дефекты бинарного образа A структурным элементом B;

4. Закрытие (замыкание): $(A \oplus B) \ominus B$, в MATLAB выполняется функцией `imclose(A,B)`, удаляет внутренние дефекты бинарного образа A структурным элементом B,

где $\oplus$ и $\ominus$ — сложение и вычитание Минковского, соответственно.

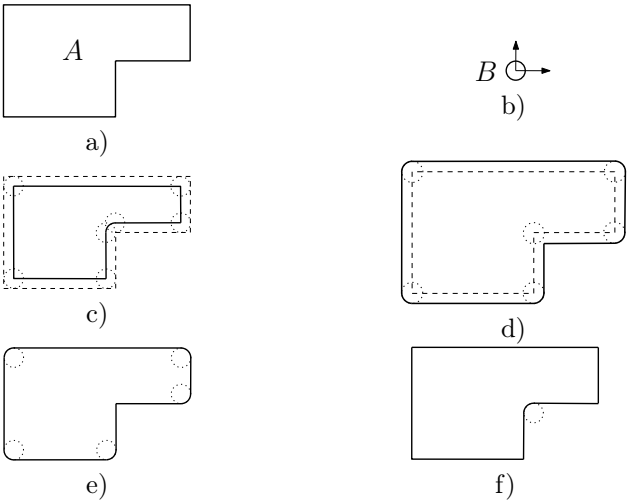


Рис. 7.1 — Результат выполения операций бинарной морфологии: а) бинарный образ A, б) дисковый структурный элемент B, в) операция дилатации, г) операция эрозии, д) операция открытия, е) операция закрытия.

В среде MATLAB имеются следующие полезные функции, часто используемые совместно с морфологическими операциями:

1. `strel()` — задает структурный элемент;

2. `bwmorph(A,operation,n)` — применяет морфологическую фильтрацию `operation` к образу `A` `n` раз;

3. `bwhitmiss(A,B,C)` — выполняет операцию «hit & miss», определяющую пиксели, окрестности которых совпадают по форме со структурным элементом `B` и не совпадают со структурным элементом `C`;

4. `bwareaopen(A,dim)` — удаляет объекты на изображении `A`, содержащие менее `dim` пикселей;

5. `bwselect()` — выбирает определенные объекты на изображении;

6. `bwarea()` — вычисляет площадь объектов;

7. `bwlabel()` — выполняет маркировку связных объектов бинарного изорбражения;

8. `bweuler()` — вычисляет число Эйлера бинарного изображения.

В библиотеке OpenCV имеются следующие функции, предназначенные для работы с морфологией и морфологическими операциями (перечисленные далее имена функций представлены для языков программирования C++ и Python соответственно):

1. `cv::getStructuringElement(shape, size, anchor)`, `cv2.getStructuringElement(shape, size, anchor)` — задает структурный элемент заданной формы *shape* и размера *size*. Если параметр центральной точки *anchor* не задан, то центром структурного элемента становится его геометрический центр. Форма структурного элемента может быть одной из следующих:

   (a) `cv::MORPH_RECT`, `cv2.MORPH_RECT` — структурный элемент прямоугольной формы размера *size*.

   (b) `cv::MORPH_CROSS`, `cv2.MORPH_CROSS` — структурный элемент в форме креста размера *size*.

   (c) `cv::MORPH_ELLIPSE`, `cv2.MORPH_ELLIPSE` — структурный элемент эллиптической формы, вписанный в прямоугольник размера *size*.

2. `cv::morphologyEx(A, C, op, B, anchor, iters, borderType, borderValue)`, `cv::morphologyEx(A, op, B, anchor, iters, borderType, borderValue)` — применяет морфологическую операцию *op* к изображению A используя структурный элемент B *iters* раз. Если параметр *iters* не задан, то морфологическая операция выполняется 1 раз. Также можно задать центральную точку структурного элемента *anchor* и метод обработки граничных точек изображения. В реализации на языке C++ результирующее изображение C передается вторым аргументом функции, тогда как в реализации на Python оно является возвращаемым значением. Используя данную функцию, можно выполнить следующие типы морфологических операций:

   (a) `cv::MORPH_ERODE`, `cv2.MORPH_ERODE` — эрозия. Для выполнения эрозии можно также использовать функцию `cv::erode()` в C++ и `cv2.erode()` в Python.

   (b) `cv::MORPH_DILATE`, `cv2.MORPH_DILATE` — дилатация. Для выполнения дилатации можно также использовать функцию `cv::dilate()` в C++ и `cv2.dilate()` в Python.

   (c) `cv::MORPH_OPEN`, `cv2.MORPH_OPEN` — открытие. $open(A, B) = erode(dilate(A, B), B)$

   (d) `cv::MORPH_CLOSE`, `cv2.MORPH_CLOSE` — закрытие. $close(A, B) = erode(dilate(A, B), B)$.

   (e) `cv::MORPH_GRADIENT`, `cv2.MORPH_GRADIENT` — морфологический градиент. Результатом данной операции является разница между дилатацией и эрозией. $gradient(A, B) = dilate(A, B) - erode(A, B)$.

   (f) `cv::MORPH_TOPHAT`, `cv2.MORPH_TOPHAT` — операция «top hat». Результатом данной операции является разница между изображением и его открытием. $tophat(A, B) = A - open(A, B)$.

   (g) `cv::MORPH_BLACKHAT`, `cv2.MORPH_BLACKHAT` — операция «black hat». Результатом данной операции является разница между закрытием изображения и им самим. $blackhat(A, B) = close(A, B) - A$.

(h) `cv::MORPH_HITMISS`, `cv2.MORPH_HITMISS` — операция «hit & miss».

3. `cv::connectedComponents(A, labels, connectivity)`, `cv2.connectedComponents(A)`
— формирует список всех связных компонент изображения A в матрице *labels*, которая хранит индекс компоненты для каждого пикселя исходного изображения. Возвращает количество связных компонент изображения в C++ и кортеж объектов, хранящий количество связных компонент и матрицу индексов компонент *labels*, в Python.

4. `cv::connectedComponentsWithStats(A, labels, stats, centers)`,
`cv2.connectedComponentsWithStats(A)` — то же самое, что и *connectedComponents*(), но кроме индексов компонент она также считает статистику для каждой компоненты. Возвращает количество связных компонент изображения в C++ и кортеж объектов, хранящий количество связных компонент, матрицу индексов компонент *labels*, статистику *stats* и центры *centers*, в Python. Статистика *stats* представляет из себя двумерную матрицу, где первое измерение — индекс компоненты (*label*), а второе — индекс параметра. Тип данных матрицы — 32 битные целые числа со знаком (*int*32). Центры *centers* — двумерная матрица, где первое измерение — это индекс компоненты (*label*), а второе 0 для оси X и 1 для оси Y. Статистика включает в себя следующие данные:

   (a) `cv::CC_STAT_LEFT`, `cv2.CC_STAT_LEFT` — координата самого левого пикселя компоненты (X);

   (b) `cv::CC_STAT_TOP`, `cv2.CC_STAT_TOP` — координата самого верхнего пикселя компоненты (Y);

   (c) `cv::CC_STAT_WIDTH`, `cv2.CC_STAT_WIDTH` — горизонтальный размер компоненты (ширина);

   (d) `cv::CC_STAT_HEIGHT`, `cv2.CC_STAT_HEIGHT` — вертикальный размер компоненты (высота);

   (e) `cv::CC_STAT_AREA`, `cv2.CC_STAT_AREA` — размер компоненты в пикселях.

Функция `bwareaopen(A, dim)` из MATLAB реализуется с использованием функции `connectedComponentsWithStats()` библиотеки OpenCV. В Приложении 5.1 представлен пример ее программной реализации на языках программирования C++ и Python.

Примеры использования морфологических операций

Выделение границ объектов

Границы объектов могут быть выделены с использованием следующего подхода:

1. $C = A - (A \ominus B)$ — формирование внутреннего контура;
2. $C = (A \oplus B) - A$ — формирование внешнего контура.

Разделение «склеенных» объектов

Одним из примеров использования морфологических операций над изображением может быть задача разделения склеившихся на изображении объектов. Задача может быть решена с достаточной степенью точности при помощи последовательного выполнения нескольких раз фильтра сжатия, а затем максимально возможного расширения полученного результата. Пересечение исходного изображения с обработанным позволит разделить склеенные объекты.

Листинг 7.1. Разделение «склеенных» объектов в среде MATLAB.

```
I = imread('pic.jpg');
t = graythresh(I);
Inew = im2bw(I,t);
Inew = ~Inew;
BW2 = bwmorph(Inew,'erode',7);
BW2 = bwmorph(BW2,'thicken',Inf);
Inew = ~(Inew & BW2);
```

Листинг 7.2. Разделение «склеенных» объектов с использованием библиотеки OpenCV и языка программирования C++.

```
Mat I = imread("pic.jpg",
  cv::IMREAD_GRAYSCALE);
Mat Inew;
cv::threshold(I, Inew, 160, 255,
  cv::THRESH_BINARY_INV);
Mat B = cv::getStructuringElement(
  cv::MORPH_ELLIPSE, cv::Size(5, 5));
// Erosion
Mat BW2;
cv::morphologyEx(Inew, BW2, cv::MORPH_ERODE,
  B, cv::Point(-1, -1), 14,
  cv::BORDER_CONSTANT, cv::Scalar(0));
// Dilation
Mat D, C, S;
Mat T = Mat::zeros(Inew.rows, Inew.cols,
  Inew.type());
int pix_num = Inew.rows * Inew.cols;
while (cv::countNonZero(BW2) < pix_num)
{
  cv::morphologyEx(BW2, D, cv::MORPH_DILATE,
    B, cv::Point(-1, -1), 1,
    cv::BORDER_CONSTANT, cv::Scalar(0));
  cv::morphologyEx(D, C, cv::MORPH_CLOSE,
    B, cv::Point(-1, -1), 1,
    cv::BORDER_CONSTANT, cv::Scalar(0));
  S = C - D;
  cv::bitwise_or(S, T, T);
  BW2 = D;
}
// Closing for borders
cv::morphologyEx(T, T, cv::MORPH_CLOSE, B,
  cv::Point(-1, -1), 14,
  cv::BORDER_CONSTANT, cv::Scalar(255));
// Remove borders from an image
cv::bitwise_and(~T, Inew, Inew);
```

Листинг 7.3. Разделение «склеенных» объектов с использованием библиотеки OpenCV и языка программирования Python.

```
I = cv2.imread("pic.jpg",
  cv2.IMREAD_GRAYSCALE);
ret, Inew = cv2.threshold(I, 160, 255,
  cv.THRESH_BINARY_INV)
B = cv2.getStructuringElement(
  cv2.MORPH_ELLIPSE, (5, 5))
# Erosion
BW2 = cv2.morphologyEx(Inew,
  cv2.MORPH_ERODE, B, iterations = 14,
  borderType = cv2.BORDER_CONSTANT,
  borderValue = (0))
# Dilation
T = np.zeros_like(Inew)
while cv2.countNonZero(BW2) < BW2.size:
  D = cv.dilate(BW2, B,
    borderType = cv2.BORDER_CONSTANT,
    borderValue = (0))
  C = cv.morphologyEx(D, cv2.MORPH_CLOSE, B,
    borderType = cv2.BORDER_CONSTANT,
    borderValue = (0))
  S = C - D
  T = cv.bitwise_or(S, T)
  BW2 = D
# Closing for borders
T = cv2.morphologyEx(T, cv2.MORPH_CLOSE, B,
  iterations = 14,
  borderType = cv2.BORDER_CONSTANT,
  borderValue = (255))
# Remove borders from an image
Inew = cv2.bitwise_and(~T, Inew)
```

Сегментация методом управляемого водораздела

Рассмотрим еще один из примеров применения математической морфологии к задаче сегментации изображения. В подходе сегментации по водоразделам изображение рассматривается как карта высот, на котором интенсивности пикселей описывают высоты относительно некоторого уровня. На такую «высотную местность» «льет

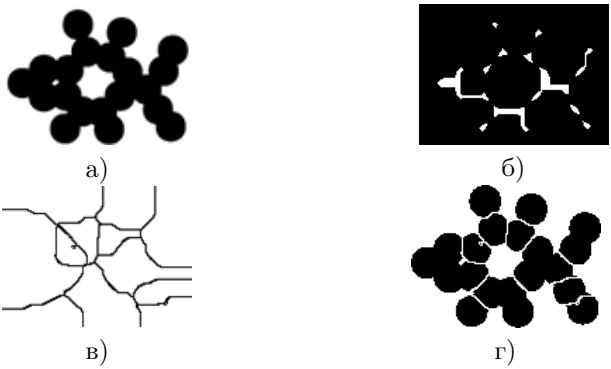


Рис. 7.2 — Разделение «склеенных» объектов: а) исходное изображение, б) эрозия бинарного изображения, в) расширение объектов, г) результат разделения.

дождь», образуя множество *водосборных бассейнов*. Постепенно вода из переполненных бассейнов переливается, и бассейны объединяются в более крупные (см. рис. 5.3). Места объединения бассейнов отмечаются как линии водораздела. Если «дождь» остановить рано, тогда изображение будет сегментировано на мелкие области, а если поздно — на крупные. В таком подходе все пиксели подразделяются на три типа:

1. *локальные минимумы*;
2. находящиеся *на склоне* (с которых вода скатывается в один и тот же локальный минимум);
3. *локальные максимумы* (с которых вода скатывается более чем в один минимум).

При реализации данного метода необходимо определить водосборные бассены и линии водораздела путем обработки локальных областей и вычисления их характеристик. Алгоритм сегментации состоит из следующих шагов:

1. Вычисление функции сегментации. Как правило, для этого используется градиентное представление изображения.

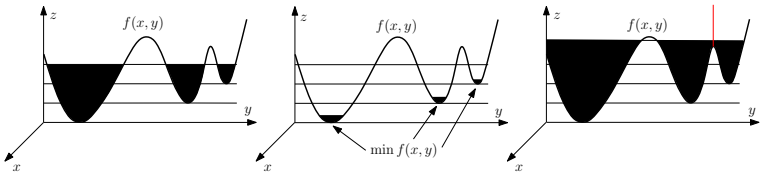


Рис. 7.3 — Водораздел областей.

2. Вычисление маркеров переднего плана на основании связности пикселей каждого объекта.

3. Вычисление маркеров фона, представляющих пиксели, не являющиеся объектами.

4. Модифицирование функции сегментации с учетом взаиморасположения маркеров переднего плана и фона.

В результате работы алгоритма будет получена маска, где пиксели одинаковых сегментов будут помечены одинаковыми метками и будут образовывать связную область. Предположим, задано изображение, представленное на рис. 7.4.

Рис. 7.4 — Исходное изображение.

Выполним морфологическую фильтрацию изображения с использованием базовых морфологических операций и морфологической реконструкции `imreconstruct()`. Функция `imcomplement()` вычисляет изображение-дополнение к изображению-аргументу и представляет собой инвертированное изображение.

Листинг 7.4. Сегментации методом водораздела в среде MATLAB.

```
1 rgb = imread('pic.jpg');
2 A = rgb2gray(rgb);
3 B = strel('disk',6);
4 C = imerode(A,B);
5 Cr = imreconstruct(C,A);
6 Crd = imdilate(Cr,B);
7 Crdr = imreconstruct(imcomplement(Crd), ...
8     imcomplement(Cr));
9 Crdr = imcomplement(Crdr);
```

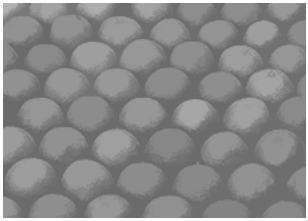


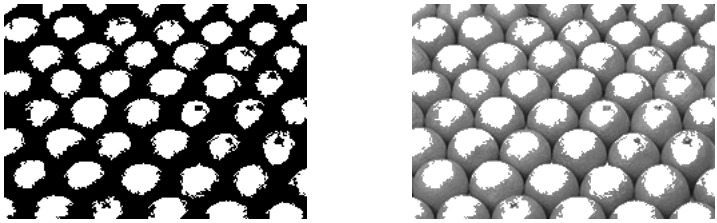
Рис. 7.5 — Отфильтрованное изображение.

После этого определим локальные максимумы функцией `imregionalmax()` для определения маркеров переднего плана (*foreground markers* или `fgm`) и, для наглядности, наложим маркеры на исходное изображение:

```
10 fgm = imregionalmax(Crdr);
11 A2 = A;
12 A2(fgm) = 255;
```

Как видно из представленных изображений, маркеры сильно изрезаны. Для сглаживания маркеров переднего плана воспользуемся следующей последовательностью морфологических операций:

```
13 B2 = strel(ones(5,5));
14 fgm = imclose(fgm,B2);
15 fgm = imerode(fgm,B2);
16 fgm = bwareaopen(fgm,20);
17 A3 = A;
18 A3(fgm) = 255;
```



а) б)

Рис. 5.6 — Маркеры переднего плана: а) вычисленные, б) наложенные на исходное изображение.

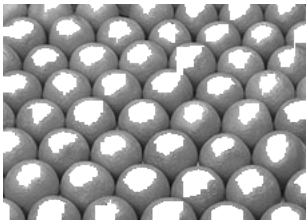
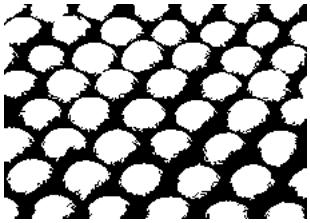


Рис. 7.7 — Отфильтрованные маркеры переднего плана.

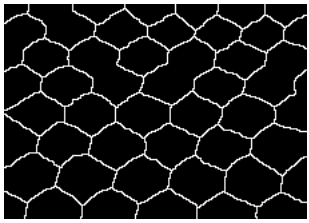
На следующем шаге требуется определить маркеры фона (*background markers* или `bgm`). Для этого бинаризуем отфильтрованное изображение `Crdr`, найдем евклидово расстояние от каждого черного до ближайшего белого пикселя функцией `bwdist()` и вычислим матрицу `L`, содержащую метки сегментов, полученных методом водораздела с использованием функции `watershed()`:

```
19 bw = imbinarize(Crdr);
20 D = bwdist(bw);
21 L = watershed(D);
22 bgm = L == 0;
```

Затем требуется модифицировать функцию сегментации. Для этого можно воспользоваться функцией `imimposemin()`, которая точно определяет локальные минимумы изображения. Благодаря этому функция корректирует значения градиентов на изображении и уточняет расположение маркеров переднего плана и фона. Перед



а)

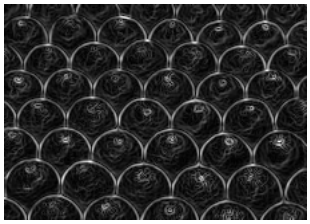


б)

Рис. 5.8 — а) бинаризованное изображение, б) маркеры фона.

этим нужно определить градиентное представление изображения, которое и будет скорректировано:

```
23 hy = fspecial('sobel');
24 hx = hy';
25 Ay = imfilter(double(A), hy, 'replicate');
26 Ax = imfilter(double(A), hx, 'replicate');
27 grad = sqrt(Ax.^2 + Ay.^2);
28 grad = imimposemin(grad, bgm | fgm);
```



а)

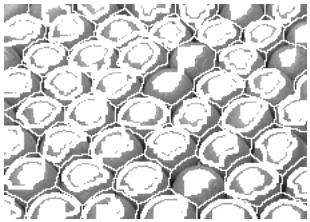
б)

Рис. 7.9 — Градиентное представление изображения: а) исходное, б) модифицированное.

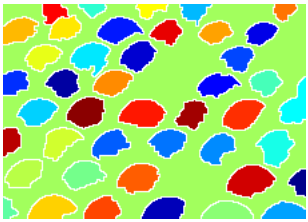
После этого выполняется операция сегментации уточненного градиентного представления изображения на основе водораздела и визуализируется результат работы алгоритма:

```
29 L = watershed(grad);
30 A4 = A;
```

```
31 A4(imdilate(L == 0, ...
32     ones(3,3)) | bgm | fgm) = 255;
33 Lrgb = label2rgb(L, 'jet', 'w', 'shuffle');
```



а)



б)

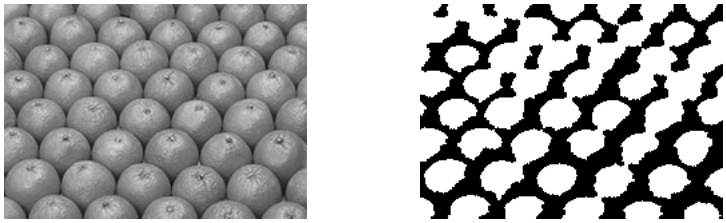
Рис. 7.10 — Маркеры и границы, наложенные на: а) исходное изображение, б) представленное в цветах rgb.

В наборе библиотеке OpenCV отсутствуют некоторые из функций MATLAB, которые использовались при реализации сегментации методом управляемого водораздела, однако можно реализовать аналогичный алгоритм. Прежде всего следует прочитать входное изображение и преобразовать его в черно-белое представление для поиска маркеров.

Листинг 7.5. Сегментация методом водораздела с использованием библиотеки OpenCV и языка программирования C++.

```
1   Mat I, I_gray, I_bw;
2   I = imread("pic.jpg", cv::IMREAD_COLOR);
3   cv::cvtColor(I, I_gray, cv::COLOR_BGR2GRAY);
4   cv::threshold(I_gray, I_bw, 0, 255,
5     cv::THRESH_BINARY + cv::THRESH_OTSU);
6   bwareaopen(I_bw, I_bw, 20, 4);
7   Mat B = cv::getStructuringElement(
8     cv::MORPH_ELLIPSE, cv::Size(5, 5));
9   cv::morphologyEx(I_bw, I_bw,
10    cv::MORPH_CLOSE, B);
```

Для определения маркеров переднего плана можно использовать преобразование евклидова расстояния, которое для каждого пикселя исходного изображения вычисляет расстояние от него



а) б)

Рис. 5.11 — а) полутоновое изображение, б) отфильтрованное черно-белое изображение.

до ближайшего черного пикселя. Это преобразование выполняется путем вызова функции `cv::distanceTransform()` библиотеки OpenCV:

```
11   Mat I_fg;
12   double I_fg_min, I_fg_max;
13   cv::distanceTransform(I_bw, I_fg,
14     cv::DIST_L2, 5);
```

Теперь становится возможно определить маркеры переднего плана как связные компоненты изображения, полученного дополнительной фильтрации, используя функцию `cv::connectedComponents()`:

```
15   cv::minMaxLoc(I_fg, &I_fg_min, &I_fg_max);
16   cv::threshold(I_fg, I_fg, 0.6 * I_fg_max,
17     255, 0);
18   img_fg.convertTo(img_fg, CV_8U,
19     255.0 / I_fg_max);
20   Mat markers;
21   int num = cv::connectedComponents(I_fg,
22     markers);
```

Область фонового маркера можно найти, выполнив алгоритм водораздела, используя полученные ранее маркеры переднего плана, и выбрав граничную область в качестве фона. Это решение хорошо работает в случае, если объекты расположены плотно, и область фона трудно выделить.

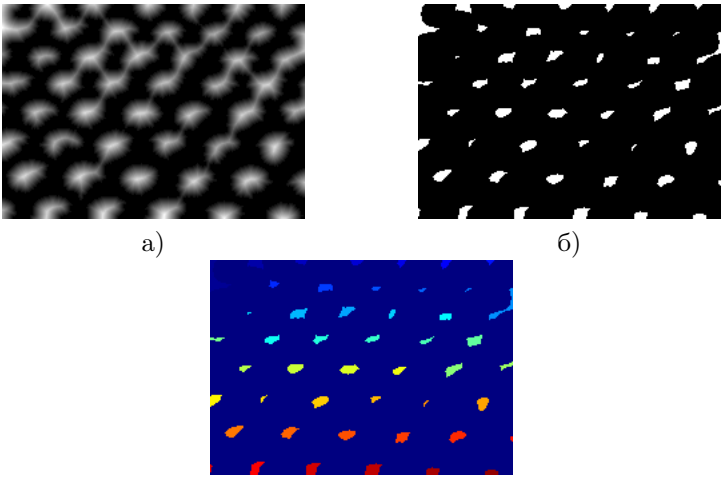


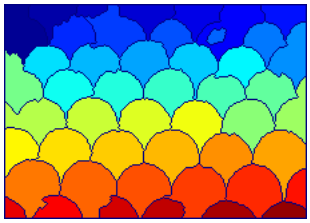
Рис. 7.12 — Определение маркеров переднего плана: а) преобразование евклидова расстояния, б) область маркеров переднего плана, в) маркеры переднего плана.

```
23   Mat I_bg = Mat::zeros(I_bw.rows, I_bw.cols,
24     I_bw.type());
25   Mat markers_bg = markers.clone();
26   cv::watershed(I, markers_bg);
27   I_bg.setTo(Scalar(255), markers_bg == -1);
```

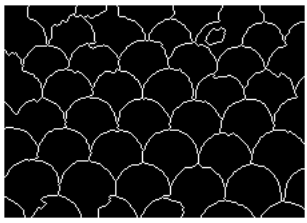
Далее необходимо определить «неопределенную» область, которая должна быть заполнена алгоритмом водораздела. Эту область можно найти путем вычитания области маркеров переднего плана из области, обратной области маркера заднего плана:

```
28   Mat I_unk;
29   cv::bitwise_not(I_bg, I_unk);
30   cv::subtract(I_unk, I_fg, I_unk);
```

Теперь получены все данные, необходимые для работы алгоритма водораздела. Осталось объединить все полученные маркеры в единую матрицу:



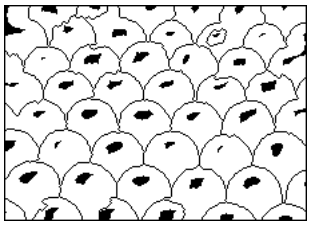
а)



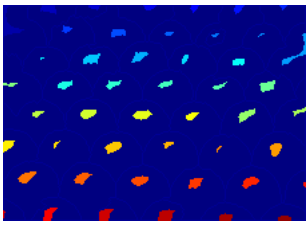
б)

Рис. 5.13 — Определение маркера фона: а) результат сегментации по маркерам переднего плана, б) маркер фона.

```
31   markers += 1;
32   markers.setTo(Scalar(0), I_unk == 255);
```



а)



б)

Рис. 7.14 — Маркеры: а) неопределенная область, б) объединение всех маркеров.

И запустить алгоритм, используя функцию `cv::watershed()`:

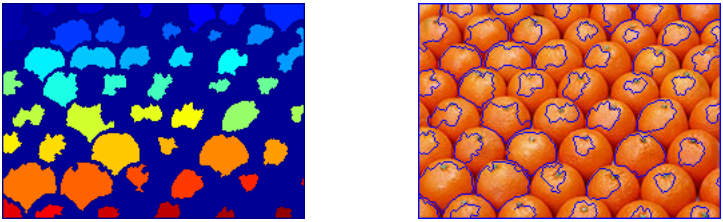
```
33   cv::watershed(I, markers);
```

Для визуализации результатов сегментации можно использовать цветовую карту JET:

```
34   Mat markers_jet;
35   markers.convertTo(markers_jet, CV_8U,
36     255.0 / (num + 1));
37   cv::applyColorMap(markers_jet,
38     markers_jet, cv::COLORMAP_JET);
```

И выделить сегментированные области на исходном изображении синим цветом.

```
39   I.setTo(Scalar(255, 0, 0), markers == -1);
```



а) б)

Рис. 7.15 — Маркеры и границы, наложенные на: а) изображение с использованием цветовой карты JET, б) исходное изображение.

В Приложении 7.2 представлена программная реализация данного метода на языке программирования Python.

Другим решением для определения маркера фона может быть последовательная дилатация отфильтрованного черно-белого изображения. Инверсия результата расширения должна дать нам область, которая обязательно будет являться фоном и может быть использована в качестве маркера фона. Это решение хорошо работает в случае, если область фона достаточно велика и не станет пустой после операции дилатации.

Листинг 7.6. Альтернативное решения для определения маркера фона с использование библиотеки OpenCV.

```
1   Mat I_bg;
2   cv::dilate(I_bw, I_bg, B,
3   cv::Point(-1, -1), 3);
4   cv::bitwise_not(I_bg, I_bg);
```

Порядок выполнения работы

1. *Базовые морфологические операции.* Выбрать произвольное изображение, содержащее дефекты формы (внутренние «дырки» или внешние «выступы») объектов. Используя базовые

морфологические операции, полностью убрать или минимизировать дефекты.

2. *Разделение объектов.* Выбрать произвольное бинарное изображение, содержащее перекрывающиеся объекты. Использовать операции бинарной морфологии для разделения объектов. Выделить контуры объектов.

3. *Сегментация.* Выбрать произвольное изображение, содержащее небольшое число локальных минимумов. Выполнить сегментацию изображения по водоразделам.

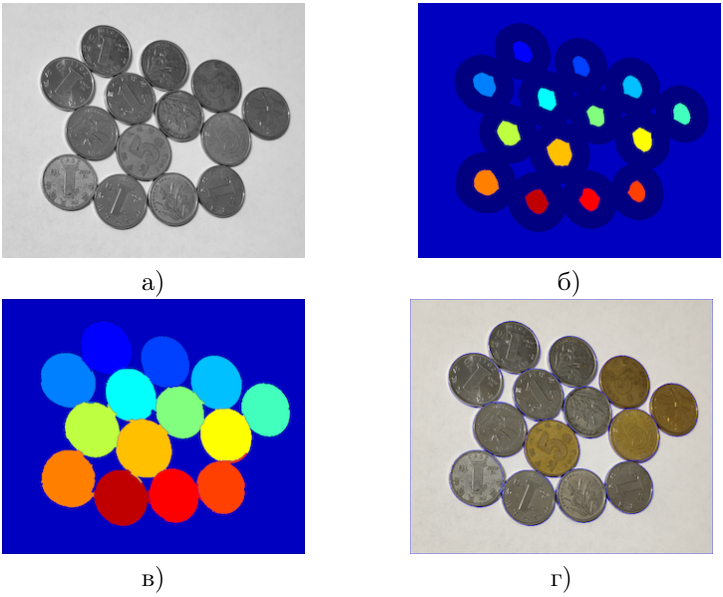


Рис. 7.16 — Использование альтернативного решения для определения маркера фона : а) черно-белое изображение, б) маркеры, в) результат сегментации, г) результат сегментации, наложенный на исходное изображение.

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Теоретическое обоснование математической морфологии для анализа изображений.
4. Ход выполнения работы:
   (a) Исходные изображения;
   (b) Листинги программных реализаций;
   (c) Комментарии;
   (d) Результирующие изображения.
5. Выводы о проделанной работе.

Вопросы к защите лабораторной работы

1. Включает ли результат открытия в себя результат закрытия?
2. Какой морфологический фильтр необходимо применить, чтобы убрать у объекта выступы?
3. Каким образом с помощью морфологических операций можно найти контур объекта?
4. Что такое *морфология*?

Приложение 7.1. Реализация функции `bwareaopen()` из среды MATLAB средствами библиотеки OpenCV

Листинг 7.7. Реализация функции `bwareaopen()` из среды MATLAB с использованием библиотеки OpenCV и языка программирования C++.

```
1   void bwareaopen(const Mat &A, Mat &C,
2     int dim, int conn = 8)
3   {
4     if (A.channels() != 1 &&
5         A.type() != CV_8U &&
6         A.type() != CV_32F)
7       return;
8     // Find all connected components
9     Mat labels, stats, centers;
10    int num =
11      cv::connectedComponentsWithStats(A,
12      labels, stats, centers, conn);
13    // Clone image
14    C = A.clone();
15    // Check size of all connected components
16    vector<int> td;
17    for (int i = 0; i < num; i++)
18      if (stats.at<int>(i,
19          cv::CC_STAT_AREA) < dim)
20        td.push_back(i);
21    // Remove small areas
22    if (todel.size() > 0)
23      if (img.type() == CV_8U)
24      {
25        for (int i = 0; i < C.rows; i++)
26          for (int j = 0; j < C.cols; j++)
27            for (int k = 0; k < td.size();
28                 k++)
29              if (labels.at<int>(i, j) ==
30                  td[k])
31              {
32                C.at<uchar>(i, j) = 0;
33                continue;
34              }
35      }
36    else
37      {
38        for (int i = 0; i < C.rows; i++)
```

```
39            for (int j = 0; j < C.cols; j++)
40              for (int k = 0; k < td.size();
41                   k++)
42                if (labels.at<int>(i, j) ==
43                    td[k])
44                {
45                  C.at<float>(i, j) = 0;
46                  continue;
47                }
48        }
49    }
```

Листинг 7.8. Реализация функции `bwareaopen()` из среды MATLAB с использованием библиотеки OpenCV и языка программирования Python.

```
1   def bwareaopen(A, dim, conn = 8):
2     if img.ndim > 2:
3       return None
4     # Find all connected components
5     num, labels, stats, centers = \
6       cv2.connectedComponentsWithStats(A,
7       connectivity = conn)
8     # Check size of all connected components
9     for i in range(num):
10      if stats[i, cv2.CC_STAT_AREA] < dim:
11        A[labels == i] = 0
12    return A
```

Приложение 7.2. Сегментация методом управляемого водораздела средствами OpenCV

Листинг 7.9. Сегментация методом управляемого водораздела с использованием библиотеки OpenCV и языка программирования Python.

```
1   # Read an image
2   # Convert to grayscale and to BW
3   # Filter
```

```
4  I = cv2.imread("pic.jpg", cv.IMREAD_COLOR)
5  I_gray = cv.cvtColor(I, cv2.COLOR_BGR2GRAY)
6  ret, I_bw = cv.threshold(I_gray, 0, 255,
7    cv2.THRESH_BINARY + cv2.THRESH_OTSU)
8  I_bw = bwareaopen(I_bw, 20, 4)
9  B = cv2.getStructuringElement( \
10   cv2.MORPH_ELLIPSE, (5, 5))
11 I_bw = cv2.morphologyEx(I_bw, \
12   cv2.MORPH_CLOSE, B)
13
14 # Do distance transformation
15 # Find foreground location
16 # Define foreground markers
17 I_fg = cv2.distanceTransform(I_bw,
18   cv2.DIST_L2, 5)
19 ret, I_fg = cv.threshold(I_fg,
20   0.6 * I_fg.max(), 255, 0)
21 ret, markers = cv.connectedComponents(I_fg)
22
23 # Find background location
24 I_bg = np.zeros_like(I_bw)
25 markers_bg = markers.copy()
26 markers_bg = cv2.watershed(I, markers_bg)
27 I_bg[markers_bg == -1] = 255
28
29 # Define undefined area
30 I_unk = cv2.subtract(~I_bg, I_fg)
31 # Define all markers
32 markers = markers + 1
33 markers[I_unk == 255] = 0
34 # Do watershed
35 # Prepare for visualization
36 markers = cv2.watershed(I, markers)
37 markers_jet = cv2.applyColorMap(
38   (markers.astype(np.float32) * 255 /
39   (ret + 1)).astype(np.uint8),
40   cv2.COLORMAP_JET)
41 I[markers == -1] = (0, 0, 255)
```

Список литературы

[1] *Журавель И.М.* Краткий курс теории обработки изображений: [Электронный ресурс]. URL: https://hub.exponenta.ru/post/kratkiy-kurs-teorii-obrabotki-izobrazheniy734. (Дата обращения: 22.03.2022).

[2] *MATLAB Documentation.* Computer Vision System Toolbox: [Электронный ресурс]. URL: https://www.mathworks.com/help/vision/index.html. (Дата обращения: 22.03.2022).

[3] *Шаветов С.В.* Основы технического зрения: учебное пособие / С.В. Шаветов. — Санкт-Петербург: НИУ ИТМО, 2017. — 86 с. — Текст: электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/110455 (дата обращения: 22.03.2022). — Режим доступа: для авториз. пользователей.

[4] *Старовойтов В.В.* Цифровые изображения: от получения до обработки / Старовойтов В.В., Голуб Ю.И. — Минск: ОИПИ НАН Беларуси, 2014. — 202 с. — ISBN 978-985-6744-80-1.

[5] *Визильтер Ю.В.* Обработка и анализ изображений в задачах машинного зрения: Курс лекций и практических занятий / Визильтер Ю.В., Желтов С.Ю., Бондаренко А.В., Ососков М.В., Моржин А.В. — М.: Физматкнига, 2010. — 672 с. — ISBN 978-5-89155-201-2.

[6] *Визильтер Ю.В.* Обработка и анализ цифровых изображений с примерами на LabVIEW IMAQ Vision / Визильтер Ю.В., Желтов С.Ю., Князь В.А., Ходарев А.Н., Моржин А.В. — М.: ДМК Пресс, 2007. — 464 с. — ISBN 5-94074-404-4.