Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Отчет по лабораторной работе №4
«Морфологический анализ изображений»
по дисциплине «Техническое зрение»

Выполнил: студент гр. R3338,
Кирбаба Д.Д.
Преподаватель: Шаветов С.В.,
канд. техн. наук, доцент ФСУ и Р

Санкт-Петербург, 2022

Цель работы

Освоение принципов математической морфологии в области обработки и анализа изображений.

Теоретическое обоснование применяемых методов

Морфологический анализ – распространенный способ анализа и обработки изображений. С его помощью могут выполняться следующие действия: фильтрация шумов, сегментация объектов, выделение контуров, поиск заданного объекта на изображении, построение «скелета» образа.

В основе морфологического анализа лежат следующие действия:

- Дилатация (расширение) – расширение бинарного образа A структурным элементом B

$$A \oplus B, \qquad \oplus - \text{сложение Минковского};$$

- Эрозия (сужение) – сужение бинарного образа A структурным элементом B

$$A \ominus B, \qquad \ominus - \text{вычитание Минковского};$$

- Открытие – удаление внешних дефектов бинарного образа A структурным элементом B

$$(A \ominus B) \oplus B;$$

- Закрытие – удаление внутренних дефектов бинарного образа A структурным элементом B

$$(A \oplus B) \ominus B;$$

- Градиент – разница между дилатацией и эрозией.

С помощью морфологических операций могут быть выделены границы изображения:

- $C = A - (A \ominus B)$ – формирование внутреннего контура (в библиотеке OpenCV: *cv.MORPH_TOPHAT*);
- $C = (A \oplus B) - A$ – формирование внешнего контура (в библиотеке OpenCV: *cv.MORPH_BLACKHAT*).

Также можно успешно проводить разделение «склеенных» объектов. Для этого необходимо подействовать на изображение несколько раз фильтром сжатия, а затем максимально возможного расширения полученного результата. В завершении произвести пересечение исходного изображения с обработанным для получения результирующей картинки, в которой объекты будут разделены.

Еще один способ сегментации изображения с использованием морфологического анализа – сегментация методом управляемого водораздела. Суть алгоритма в следующем: изображение представляется в виде карты высот (в качестве высоты выступает интенсивность пикселей), затем на такую местность «льет дождь» образуя множество бассейнов и водоразделов. При реализации данного метода необходимо водосборные бассейны и линии водораздела определяются путем обработки локальных областей и вычисления их характеристик.

Ход выполнения работы

1. Базовые морфологические операции

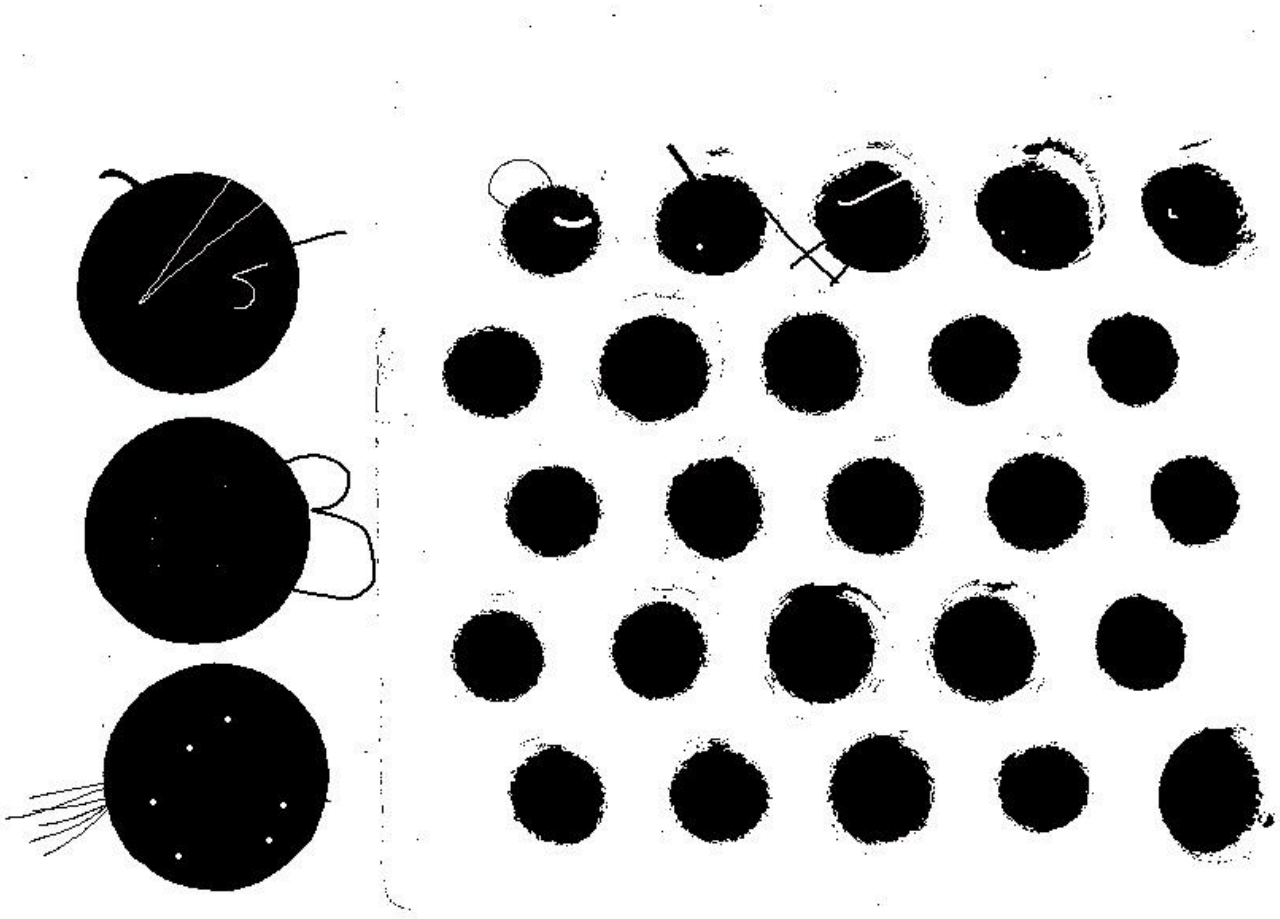

Figure 1: исходное изображение с дефектами формы.

1.1. Открытие

Операция открытия позволяет удалить внешние дефекты изображения. Она состоит из последовательного применения эрозии и дилатации.

Применим эрозию к изображению.

Listing 1. Применение эрозии на Python.

```
# Create struct element
el = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))

# Apply morph
res_img = cv.morphologyEx(img, cv.MORPH_ERODE, el,
borderType=cv.BORDER_CONSTANT, borderValue=255)
```

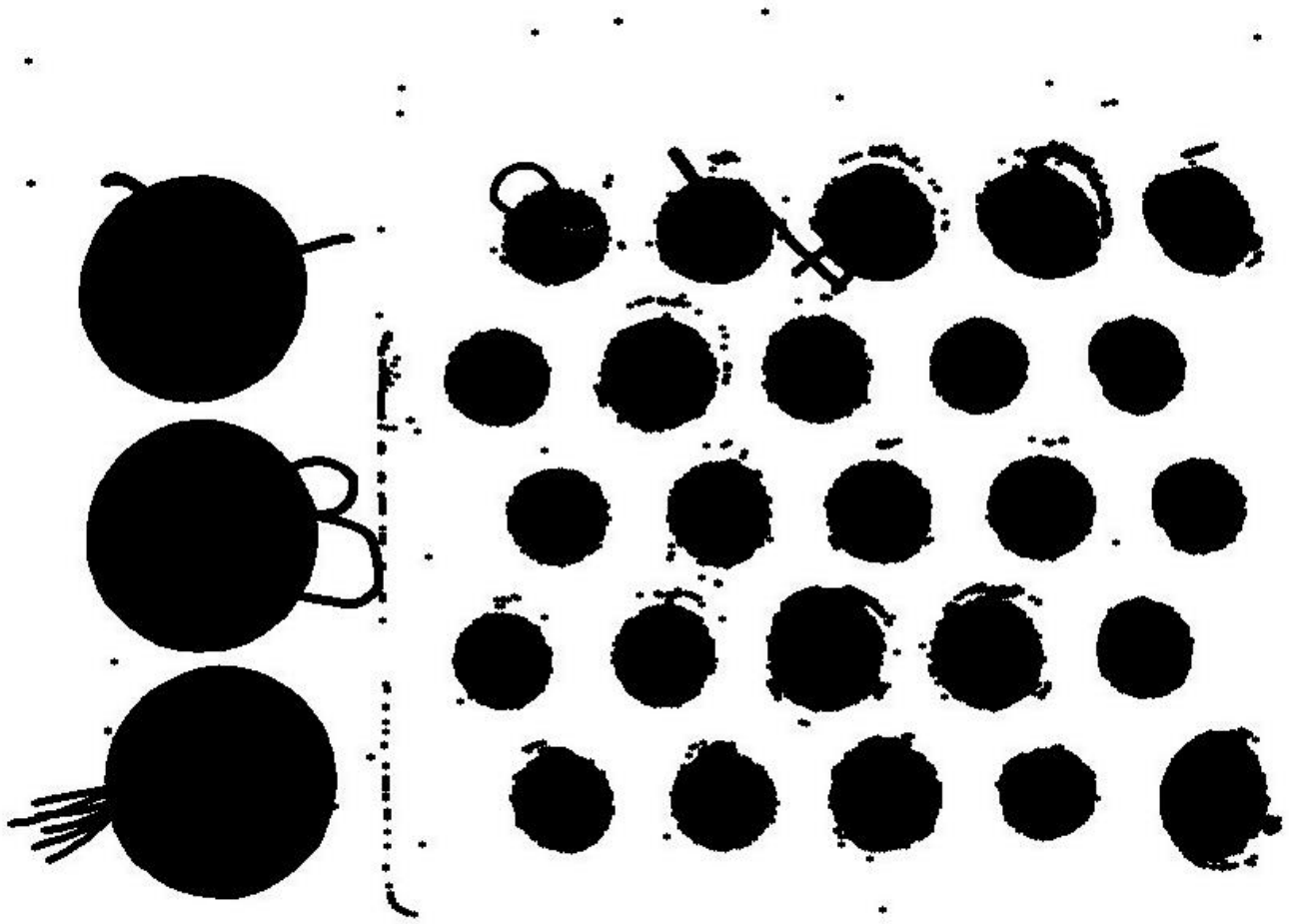

Figure 2: изображение после эрозии.

Так как операция эрозии состоит в том, что структурный элемент подставляется по контуру объектов, и результирующее изображение получается увеличенным, так ещё и мы удаляем все внутренние эффекты (только в том случае, если размер структурного элемента достаточен для перекрытия внутренних дефектов). Например, мы наблюдаем один объект, у которого по-прежнему имеются «дыры», значит, что для их удаления необходимо выбрать структурный элемент большего размера.

Однако необходимо помнить, что при больших размерах структурного элемента хотя и удаляются внутренние дефекты, однако формы объектов безвозвратно деформируются и сглаживаются.

Размер ядра является так называемым «гиперпараметром» и устанавливается исследователем на его усмотрение.

Для того, чтобы восстановить размеры объектов, необходимо применить операцию дилатации к полученному результату.

Listing 2. Применении дилатации на Python.

```
# Create struct element
el = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))

# Apply morph
res_img = cv.morphologyEx(img, cv.MORPH_DILATE, el,
borderType=cv.BORDER_CONSTANT, borderValue=255)
```

Стоит отметить также, что при применении функции *morphologyEx()* я также использую параметры границ изображения для более корректного применения морфологического фильтра на краях изображения. Так, я расширил границы пикселями с интенсивность равной интенсивностью фона (255). Так как в данном случае мы исследуем черные объекты на белом фоне (бинарное изображение).

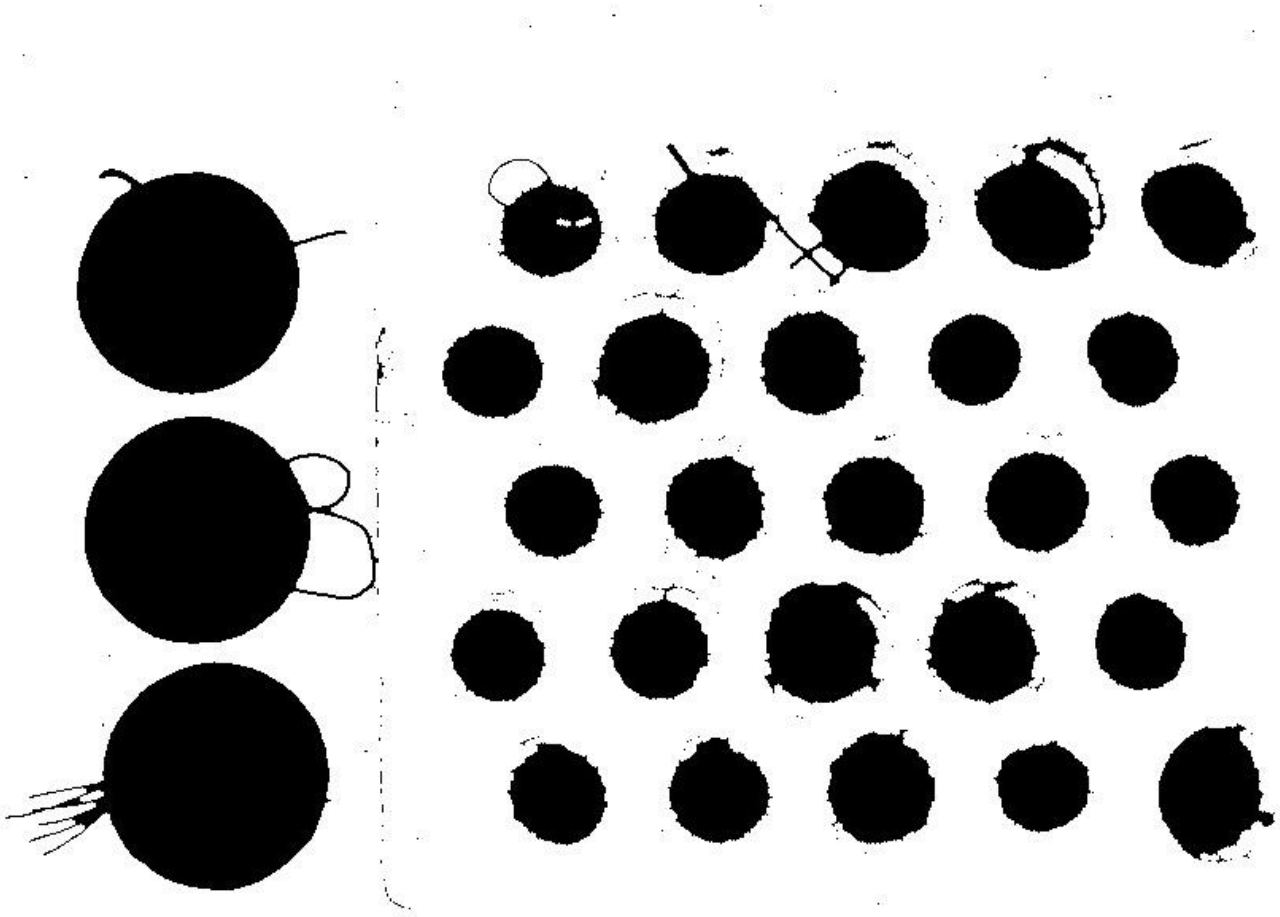

Figure 3: изображение после (эрозии+дилатации)=открытия.

Нам удалось удалить почти все внутренние дефекты, а также сохранить размеры объектов.

Стоит отметить, что мы можем использовать соответствующий параметр в функции $cv.morphologyEx()$ для вычисления открытия. А именно *cv.MORPH_OPEN*.

Listing 3. Применение открытия на Python.

```
# Create struct element
el = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))

# Apply morph
res_img = cv.morphologyEx(img, cv.MORPH_OPEN, el,
borderType=cv.BORDER_CONSTANT, borderValue=255)
```

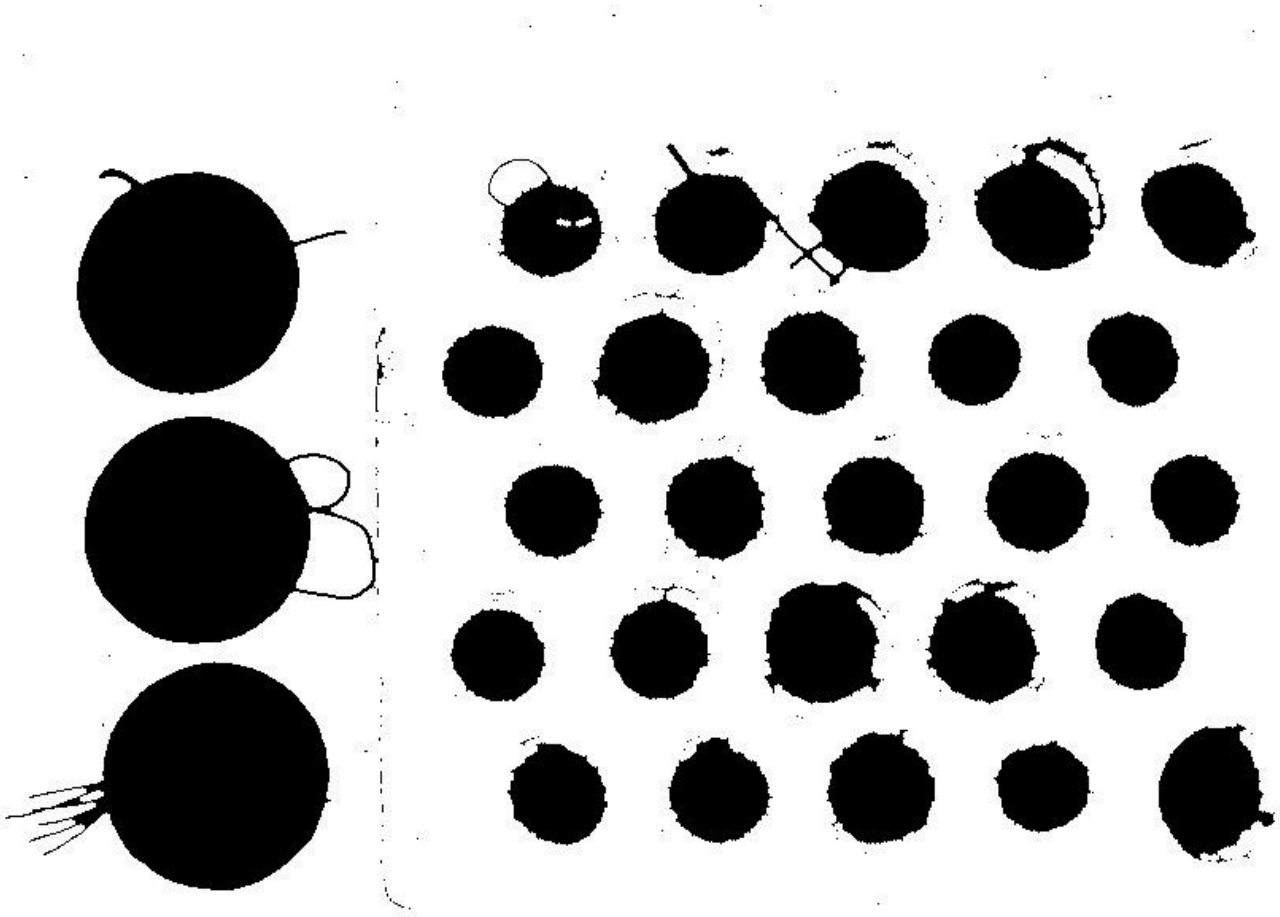

Figure 4: изображение после открытия.

Результаты одинаковы.

1.2. Закрытие

Операция закрытия позволяет удалить внешние дефекты объектов. Она состоит из последовательного применения дилатации и эрозии.

Применим данную морфологическую операцию к изображению без внутренних дефектов, полученному в предыдущем пункте.

Listing 4. Применение закрытия к изображению на Python.

```
# Create struct element
el = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))

# Apply morph
res_img = cv.morphologyEx(img, cv.MORPH_CLOSE, el,
borderType=cv.BORDER_CONSTANT, borderValue=255)
```

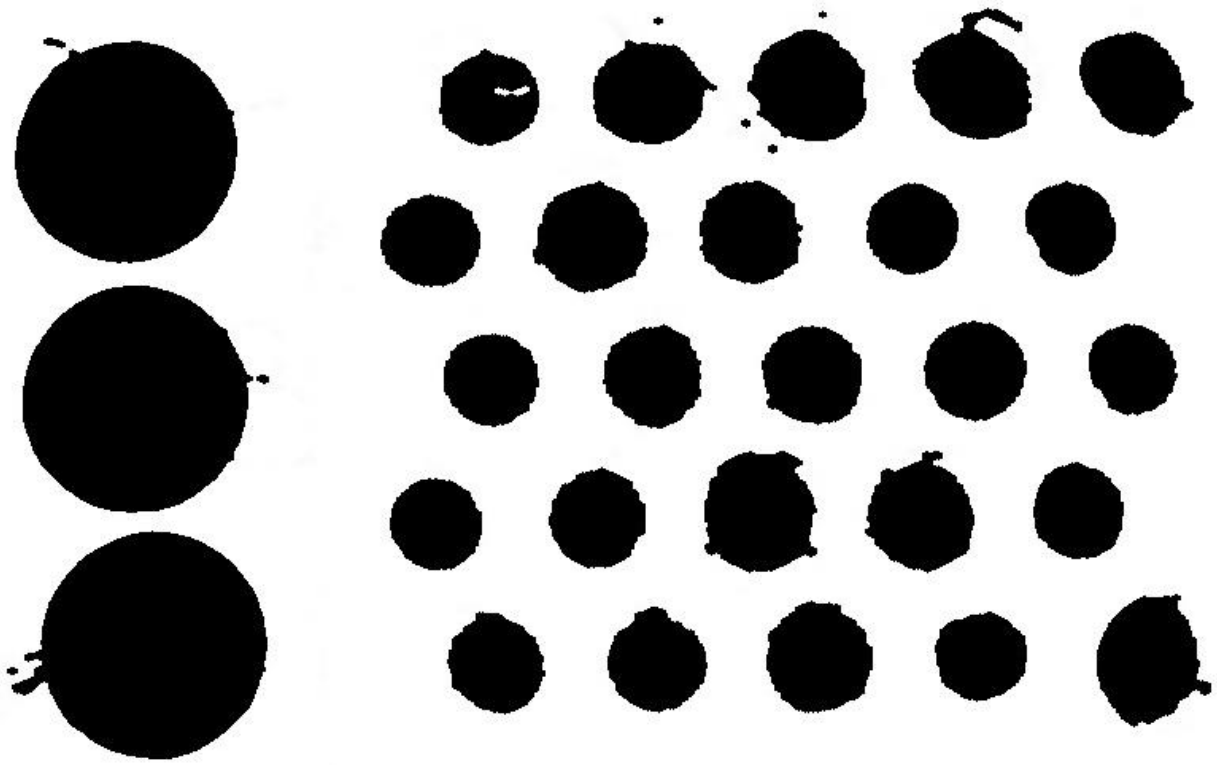

Figure 5: изображение после закрытия.

Видно, что количество внешних дефектов значительно уменьшилось. Однако тут та же проблема между выбором размера ядра фильтрации и детализацией объектов на изображении.

Суммируя результаты, операции закрытия и открытия можно успешно применять к изображениям для уменьшения как внутренних, так и внешних дефектов. Стоит отметить, что за счет вариации как формы, так и размера ядра фильтрации данный метод является довольно гибким.

2. Разделение объектов

Figure 6: исходное изображение с перекрывающимися объектами.

Пропустим изображение через пороговый фильтр для того, чтобы получить бинарное изображение.

Listing 5. Бинаризация изображения на Python.

```
ret, thres_img = cv.threshold(img, 20, 255, cv.THRESH_BINARY)
```

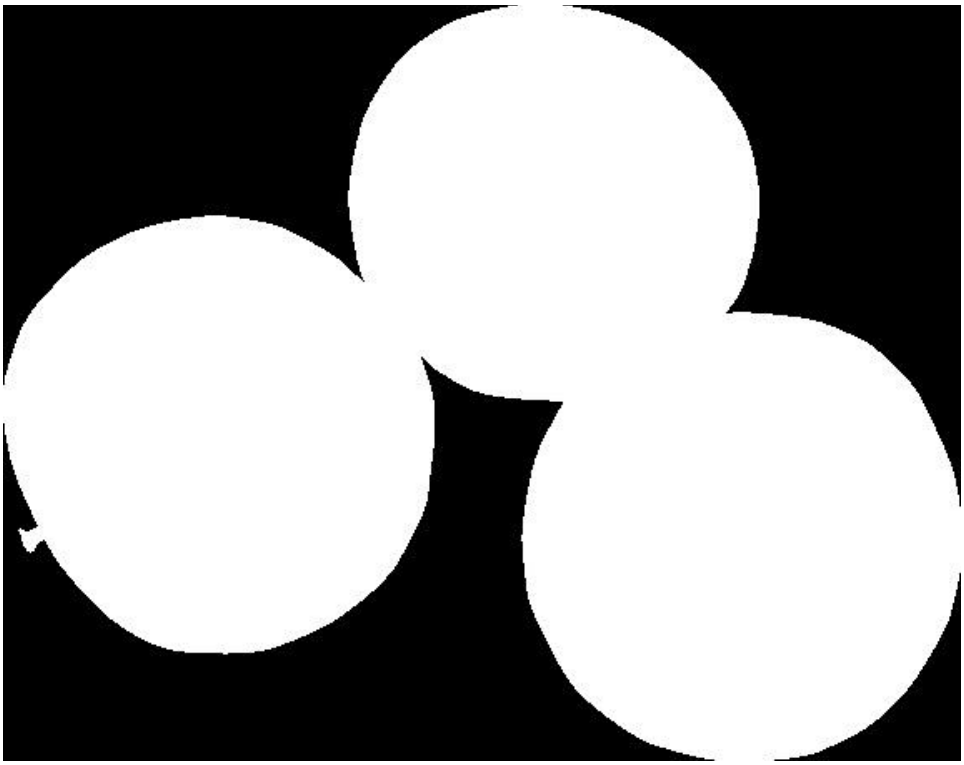

Figure 7: изображение в бинарном представлении.

Для разделения объектов, требуется в начале провести несколько операций эрозии для того, чтобы разделить объекты на изображении.

В нашем случае будем проводить эрозию до тех пор, пока не получим 3 разделенных объекта на изображении.

Listing 6. Эрозия изображения для отделения объектов на Python.

```
# Set structuring element
el = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))

working_img = np.copy(thres_img)  # Create working array
iter_counter = 0  # Counter

# Compute initial connected components map
retval, labels = cv.connectedComponents(thres_img)

while retval < 4:
    # Apply erosion
    working img = cv.morphologyEx(working img, cv.MORPH ERODE, el,
borderType=cv.BORDER_CONSTANT, borderValue=(0))
    retval, labels = cv.connectedComponents(working img)
    iter_counter += 1
```

Значение счетчика *iter_counter* в конце программы равно 26. Это значит, что для разделения нам понадобилось 26 операций эрозии.

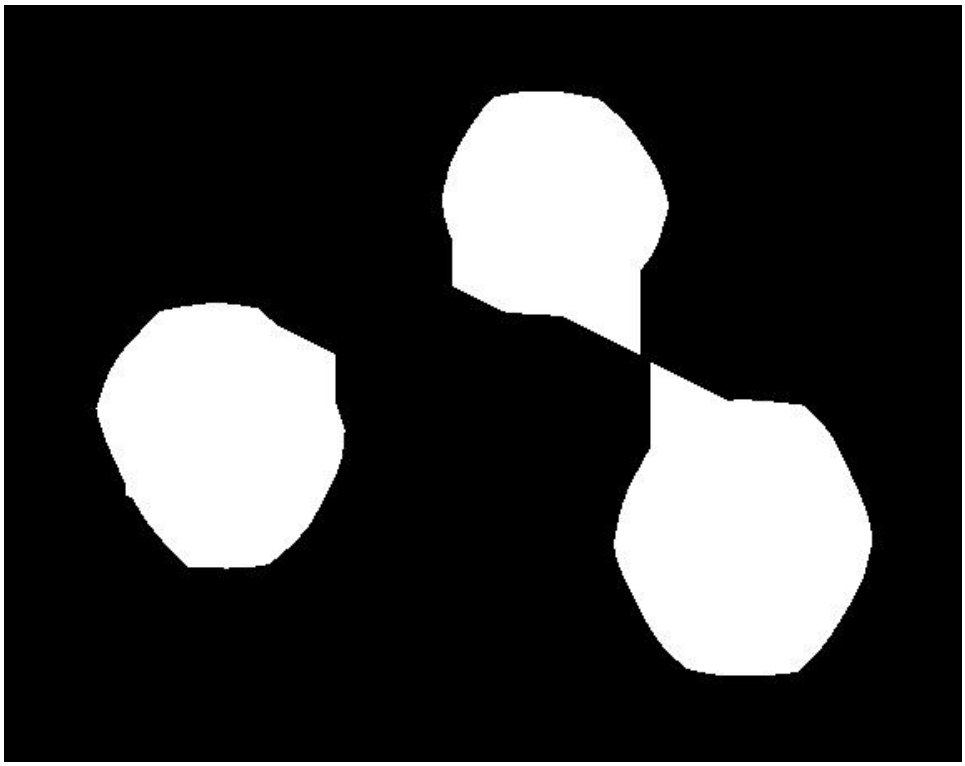

Figure 8: изображение после эрозии.

Как видим, действительно нам удалось отделить объекты друг от друга, однако форма объектов изменилась значительно.

Далее, найдем границы объектов используя следующий алгоритм.

Listing 7. Поиск границы объектов на Python.

```
# Find borders
img_borders = np.zeros_like(img)
while cv.countNonZero(working_img) < working_img.size:
    dilated_img = cv.morphologyEx(working_img, cv.MORPH_DILATE, el,
borderType=cv.BORDER_CONSTANT, borderValue=(0))
    closed_img = cv.morphologyEx(dilated_img, cv.MORPH_CLOSE, el,
borderType=cv.BORDER_CONSTANT, borderValue=(0))
    substracted_img = closed_img - dilated_img
    img_borders = cv.bitwise_or(substracted_img, img_borders)
    working_img = dilated_img
```

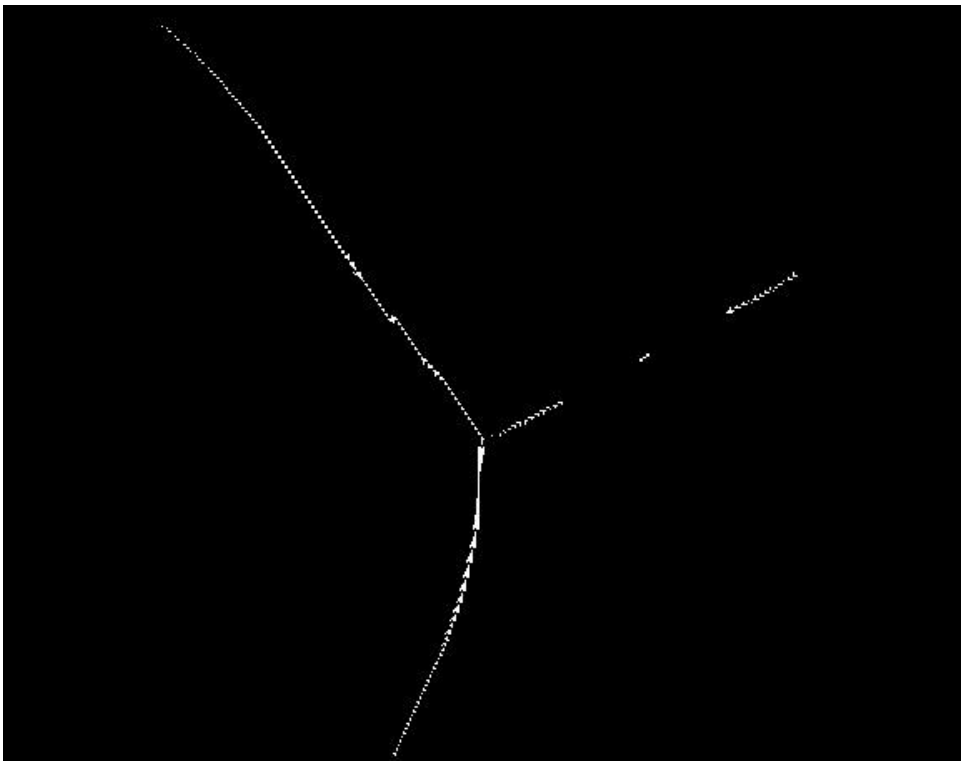

Figure 9: границы объектов.

Применим операцию закрытия для того, чтобы убрать внутренние дефекты границ, а именно сделать их сплошными линиями.

Listing 8. Применение закрытия к границам на Python.

```
# Closing for borders
img_borders = cv.morphologyEx(img_borders, cv.MORPH_CLOSE, el,
iterations=iter_counter, borderType=cv.BORDER_CONSTANT, borderValue=(0))
```

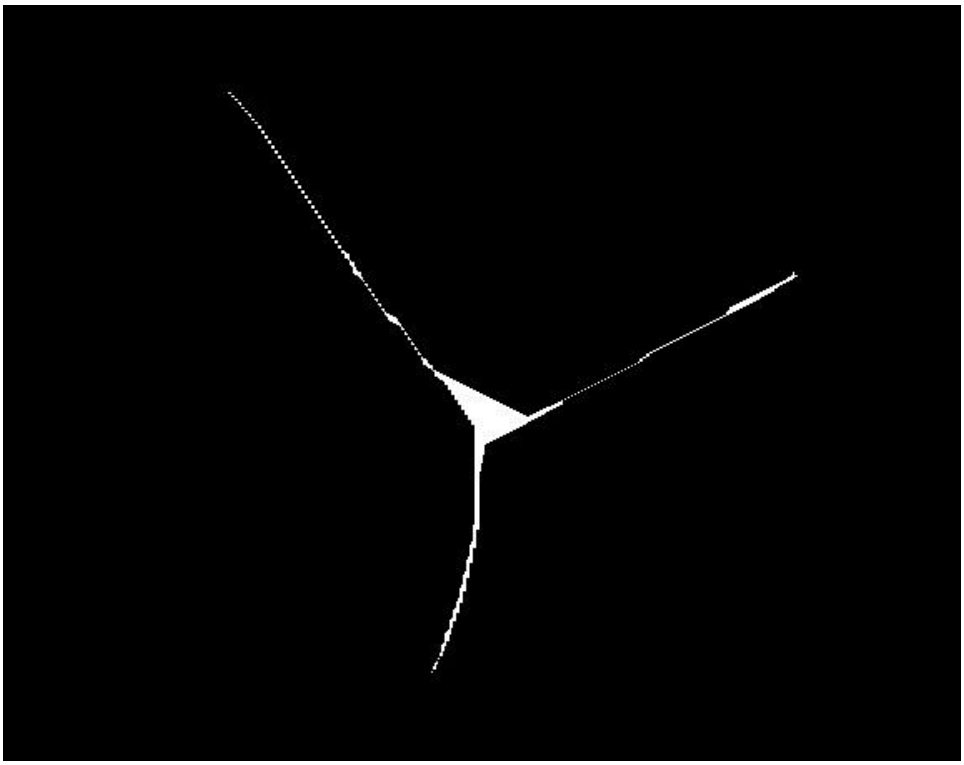

Figure 10: границы после открытия.

Как видим, границы стали сплошными линиями.

Стоит отметить, что для сохранения формы объектов необходимо применить равное количество раз как эрозию в начале алгоритма, так и закрытие тут (за это отвечает параметр *iter_counter*).

И на последнем шаге необходимо отделить удалить границу из исходного бинарного изображения.

Listing 9. Удаление границы с бинарного изображения на Python.

```
# Remove borders from image
img_res = cv.bitwise_and(~img_borders, thres_img)
```

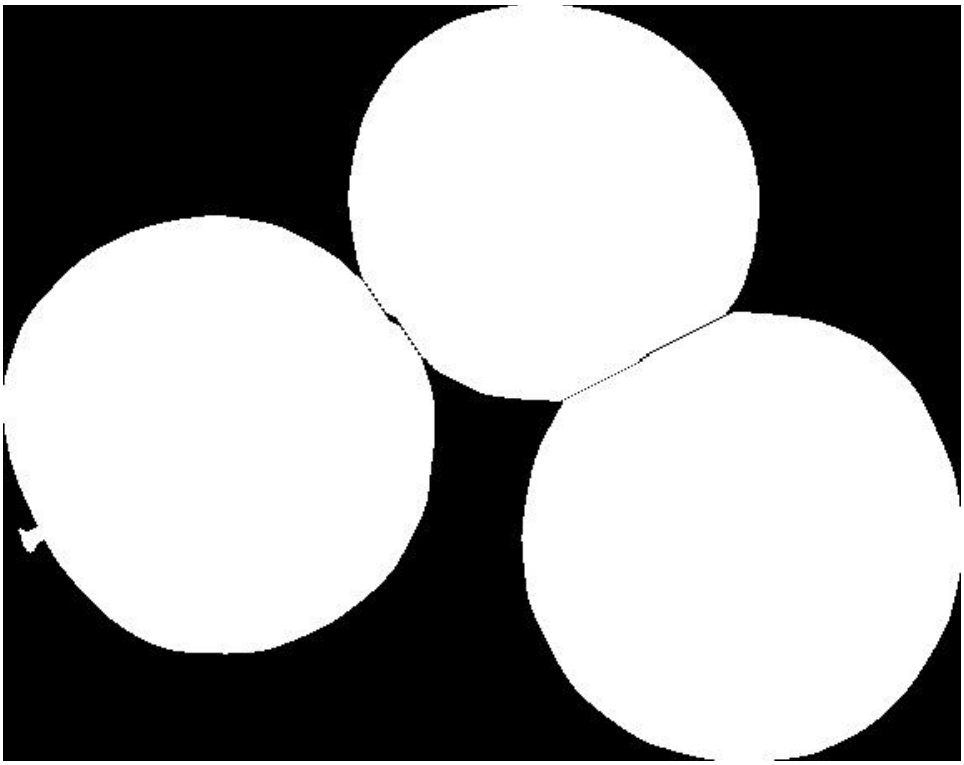

Figure 11: результат разделения объектов морфологическими операциями.

Итого, наблюдаем разделенные объекты.

Линия раздела является достаточно тонкой, что является плюсом работы алгоритма. Однако её можно настраивать, применяя морфологические операции (эрозия, дилатация…).

Результат работы программы хороший.

Смотря на исходное изображение в цвете, видно, что линия раздела проходит не по линии объектов. Однако это случилось, так как мы применяли фильтры над бинарным представлением изображений и соответственно другого результата получить не могли.

Задача разделения объектов на цветной (или чёрно-белой) картинке называется сегментацией и будет выполнена в следующем пункте, она является логичным продолжением и усложнением данного алгоритма.

3. Сегментация

Figure 12: исходное изображение для сегментации.

Будем применять алгоритм сегментации по водоразделам к данному изображению. Цель – отделить объекты друг от друга.

Для реализации данного метода необходимо определить водосборные бассейны и линии водораздела путем обработки локальных областей и вычисления их характеристик.

Переведем изображение в полутоновый вид.

Listing 10. Перевод изображения в полутоновый вид на Python.

```
# Convert to grayscale
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

Figure 13: изображение в grayscale.

Применим фильтрацию пороговым значением методом Оцу.

Listing 11. Фильтрация пороговым значением методом Оцу на Python.

```
# Apply threshold
ret_gray, img_threshold = cv.threshold(img_gray, 0, 255,
cv.THRESH_BINARY + cv.THRESH_OTSU)
```

Figure 14: изображение после пороговой фильтрации.

Как видно по исходному изображению – на нем отсутствует фон как таковой. Поэтому сложно подобрать такое пороговое значение, при котором возможно отделить достаточно точно хотя бы несколько объектов. Из-за этого в дальнейшем будут неточности, однако в целом работоспособность алгоритма будет доказана.

В итоге, в данном случае – фоном изображения будем считать объекты на заднем фоне с наименьшей интенсивностью.

Выполним некоторые преобразования для уменьшения дефектов.

Listing 12. Преобразования для уменьшения дефектов при сегментации по водоразделам на Python.

```
def bwareopen(img, dim, conn=8):
    if img.ndim > 2:
        return None
    # Find all connected components
    num, labels, stats, centers = cv.connectedComponentsWithStats(img,
connectivity=conn)
    # Check size of all connected components
    for i in range(num):
        if stats[i, cv.CC_STAT_AREA] < dim:
            img[labels == i] = 0
    return img
```

```
# Remove noise
working_img = bwareopen(img_threshold, 40, 4)
el = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
working_img = cv.morphologyEx(working_img, cv.MORPH_CLOSE, el,
borderType=cv.BORDER_REFLECT)
```

Figure 15: изображение после преобразований.

Теперь найдем те участки, которые точно являются объектами. Также необходимо отделить такие объекты друг от друга.

Listing 13. Нахождение положения объектов на Python.

```
# Find sure foreground
img_fg = cv.distanceTransform(cv.erode(working_img, el,
iterations=20), cv.DIST_L2, 5)
cv.normalize(img_fg, img_fg, 0, 255.0, cv.NORM_MINMAX)

ret_fg, img_fg = cv.threshold(img_fg, 0.4 * img_fg.max(), 255,
cv.THRESH_BINARY)
img_fg = np.clip(img_fg, 0, 255).astype(np.uint8)

ret, markers = cv.connectedComponents(img_fg)  # Location and markers
of sure foreground
```

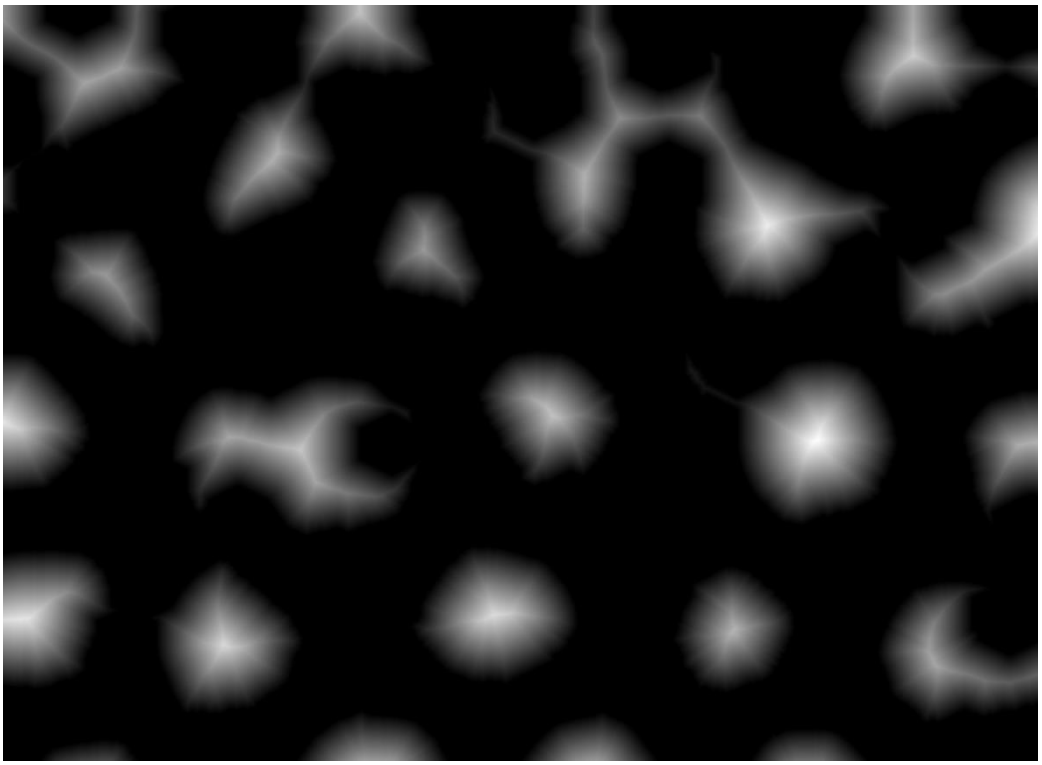

Figure 16: результат после distance transform без пороговой фильтрации.

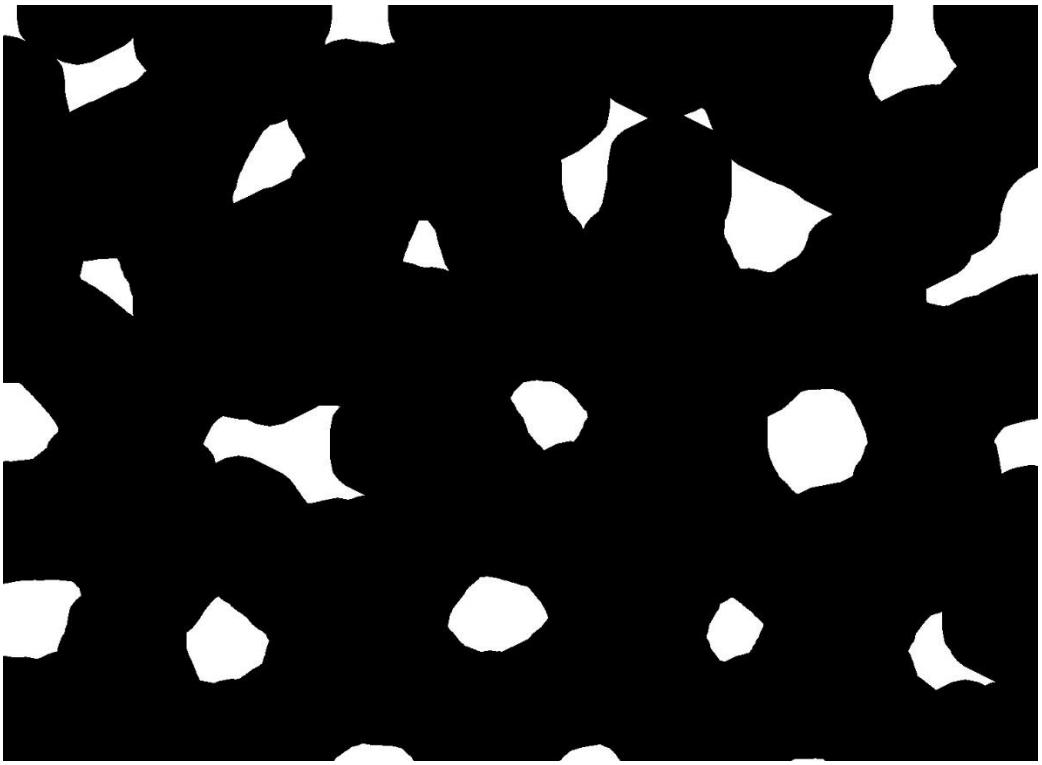

Figure 17: результат после distance transform после пороговой фильтрации.

Итого, благодаря применению эрозии некоторое количество раз, нам удалось отделить объекты, затем применили *distance transform*.

Вообще говоря, эрозия — это просто другой метод выделения области переднего плана, вот и все. Однако следует помнить, что в некоторых случаях необходимо только сегментация переднего плана, а не разделение взаимно соприкасающихся объектов. В этом случае вам не нужно использовать преобразование расстояния, достаточно эрозии.

То есть, в данном случае можно было ограничиться только эрозией, но для большей точности я использовал оба подхода последовательно.

Теперь найдем расположение фона и расположение неопределенных пикселей, которые будут определены к конкретной области с помощью сегментации водоразделом.

Listing 14. Нахождение маркеров и расположения неопределенных пикселей изображения на Python и визуализация методом JET областей до применения сегментации водоразделом.

```
# Find sure background
img_bg = cv.dilate(img_threshold, el, iterations=15)

# Finding unknown region
img_unknown = cv.subtract(img_bg, img_fg)

# Add one to all labels so that sure background is not 0, but 1
markers = markers + 1
# Now, mark the region of unknown with zero
markers[img_unknown == 255] = 0

# JET colormap areas before watershed segmentation
# The dark blue region shows unknown region. Sure coins are colored
with different values.
# Remaining area which are sure background are shown in lighter blue
compared to unknown region.
img_jet_before = cv.applyColorMap((markers.astype(np.float32) * 255 /
(ret + 1)).astype(np.uint8), colormap=cv.COLORMAP_JET)
cv.imshow("JET before watershed segmentation", img_jet_before)
```

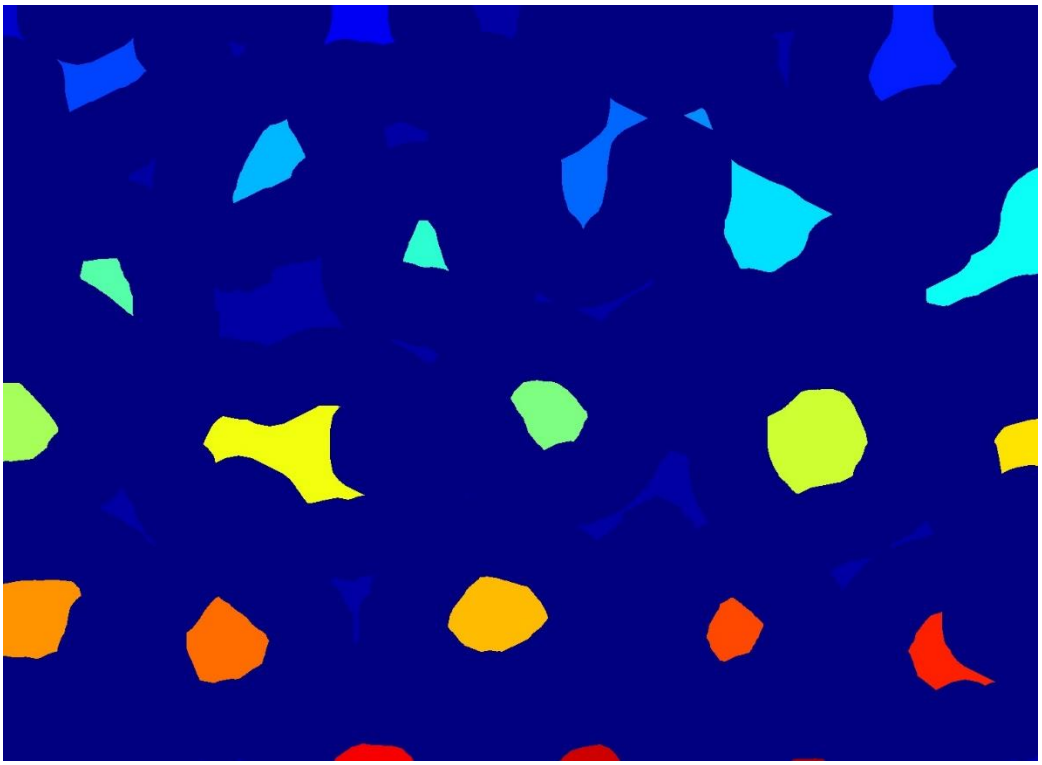

Figure 18: результат определения областей в цветовом представлении JET.

Темно-синяя область показывает неизвестную область. Объекты на переднем плане окрашены в разные цвета. Оставшаяся область, которая является фоном, показана более светлым синим цветом по сравнению с неизвестной областью.

Осталось лишь применить сегментацию и сделать визуализацию результата.

Listing 15. Применение сегментации и визуализация результата на Python.

```
# Do watershed
markers = cv.watershed(img, markers)

# Visualization
res_img = np.copy(img)
res_img[markers == -1] = 255
cv.imshow("Result", res_img)

# JET colormap areas after watershed segmentation
img_jet_after = cv.applyColorMap((markers.astype(np.float32) * 255 /
(ret + 1)).astype(np.uint8), colormap=cv.COLORMAP_JET)
cv.imshow("JET after watershed segmentation", img_jet_after)
```

Figure 19: границы сегментированных объектов на исзодном изображении.

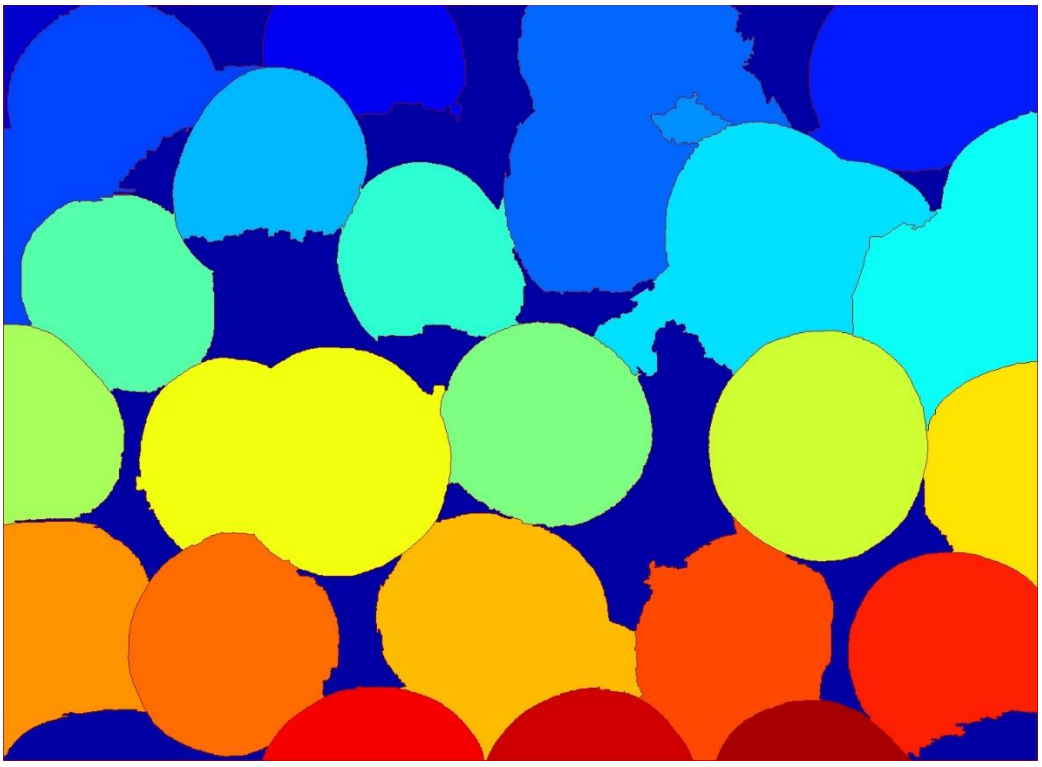

Figure 20: результат сегментации в цветовом представлении JET.

Итого результат сегментации достаточно хорош несмотря на то, что объекты сильно соприкасаются и накладываются друг на друга, а также отсутствует фон как таковой.

Таким образом, данный метод может применяться для сегментации объектов на изображении. Естественно, можно настраивать параметры алгоритма для получения необходимого результата.

Выводы

Изучены и опробованы базовые морфологические операции над бинарным изображением, такие как Дилатация, Эрозия, Открытие, Закрытие. С помощью этих операций было проведено удаление внутренних и внешних дефектов на изображении. Результат достаточно хороший, однако полностью удалить все дефекты не удалось.

Выполнено разделение объекта через операции бинарной морфологии. Сам алгоритм идейно прост, но крайне невыгоден по вычислительной сложности. Требуется повторять одну и ту же матричную операцию сотни раз. Результат работы разделения объектов как результат решения задачи сегментации оцениваю как средний, колоссальное количество мелких классов теряется во время эрозии.

Выполнена сегментация изображения методом водораздела. Все шаги алгоритма расписаны и предоставлены промежуточные результаты в процессе его работы. Результат работы сегментации методом водораздела можно оценить как хороший, классы теряются крайне редко, но и это правится параметрами.