哈爾濱工業大學

智能显微操作

实验一　斑马鱼图像处理

院系：航天学院四系

学号：1190302025

姓名：庄昇洋

哈尔滨工业大学　控制科学与工程系

1. 实验目的

本实验涉及图像处理领域，特别是涉及一种基于显微操作系统的斑马鱼姿态检测方法及系统。此实验主要探讨当前常见的几种滤波的方法，并要求对形态学膨胀等图像处理相关的基本概念进行了解，最终通过更改参数等方式经实践操作加深不同因素对图像处理效果的影响的认识。

2. 实验准备

该实验要求在电脑上安装 Python 和 opencv 安装包，具体操作方法为：下载并安装 anaconda3，然后在文件夹中配置环境，最后利用 jupyter notebook 进行实验。实验环境如图 1 所示。

图 1 实验环境

3. 实验内容

3.1 图像数据的处理与灰度图的生成

3.1.1 图像数据的处理的基本原理

用计算机进行图像处理的前提是图像必须以数字格式存储到计算机中，以数字格式存放的图像称为数字图像[1]。数字图像处理又称为计算机图像处理，它是指将图像信号转换成数字信号并利用计算机对其进行处理的过程。这一过程包括

对图像进行增强、除噪、分割、复原、编码、压缩、提取特征等内容，图像处理技术离不开计算机的发展。[1]

3.1.2 图像的滤波原理

原始实体资料变为数字图象在计算机中进行处理的时候，可能会产生各种各样的噪声，这些噪声可能是在进行数字转换过程中，因为输入设备的原因产生，也可能在对数字图象的处理中产生，噪声不一定是真实的声音，可以理解为影响人的视觉器官或系统传感器对所接收图象源信息进行理解或分析的各种因素。处理噪声的方法主要有以下几种[1]：

① 均值滤波器：适用于去除通过扫描得到的图象中的颗粒噪声。

② 自适应维纳滤波器：它能根据图象的局部方差来调整滤波器的输出，局部方差越大，滤波器的平滑作用越强。

③ 中值滤波器：基本原理是把数字图像或数字序列中一点的值用该点的一个领域中各点值的中值代换。 其主要功能是让周围象素灰度值的差比较大的像素改取与周围的像素值接近的值，从而可以消除孤立的噪声点，对于滤除图像的椒盐噪声非常有效。

④ 高斯滤波器：将在 3.2 节中详细介绍。

3.1.3 图像的灰度化原理和实现

将彩色图像转化成为灰度图像的过程成为图像的灰度化处理。彩色图像中的每个像素的颜色有 R、G、B 三个分量决定，而每个分量有 255 个中值可取，这样一个像素点可以有 1600 多万（255^3）的颜色的变化范围。而灰度图像是 R、G、B 三个分量相同的一种特殊的彩色图像，其一个像素点的变化范围为 255 种，所以在数字图像处理中一般先将各种格式的图像转变成灰度图像以使后续的图像的计算量变得少一些。灰度图像的描述与彩色图像一样仍然反映了整幅图像的整体和局部的色度和亮度等级的分布和特征。图像的灰度化处理可用三种方法来实现，分别为加权法、均值法、最大值法。

加权法是根据 YUV 的颜色空间中，Y 的分量的物理意义是点的亮度，由该值反映亮度等级，根据 RGB 和 YUV 颜色空间的变化关系可建立亮度与 R、G、B 三个颜色分量的对应关系为

$$GRAY = 0.3R + 0.059G + 0.11B$$

均值法的基本原理为

$$GRAY = \frac{(R + G + B)}{3}$$

最大值法的基本原理为

$$GRAY = \max\{R, G, B\}$$

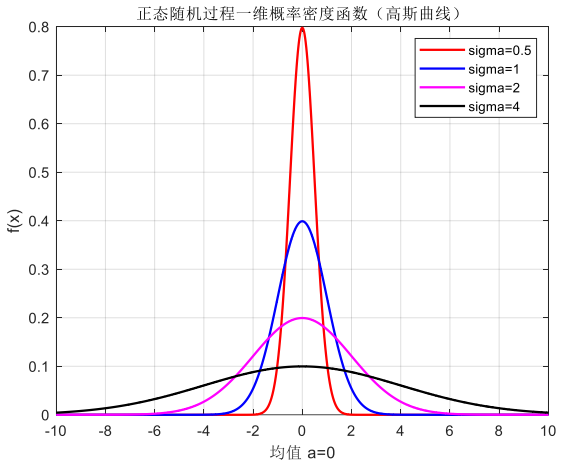
图像灰度化处理可以作为图像处理的预处理步骤，为之后的图像分割、图像识别和图像分析等上层操作做准备。

3.2 高斯滤波

高斯滤波就是对整幅图像进行加权平均的过程，每一个像素点的值，都由其本身和邻域内的其他像素值经过加权平均后得到[3]。高斯滤波的具体操作是：用一个模板（或称卷积、掩模）扫描图像中的每一个像素，用模板确定的邻域内像素的加权平均灰度值去替代模板中心像素点的值[4]。

一维高斯滤波函数及其图像为

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



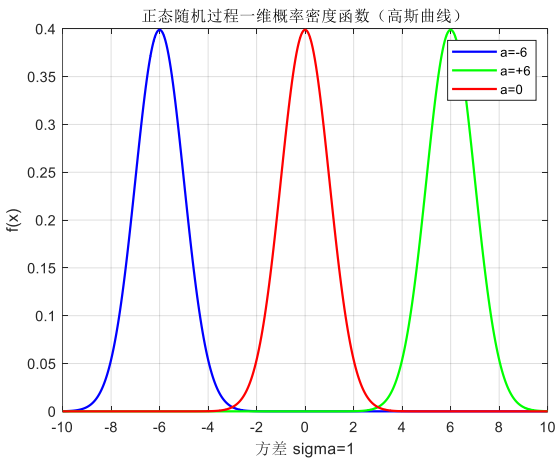


图 2 MATLAB 绘制一维高斯滤波函数

二维高斯滤波函数及其图像为

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

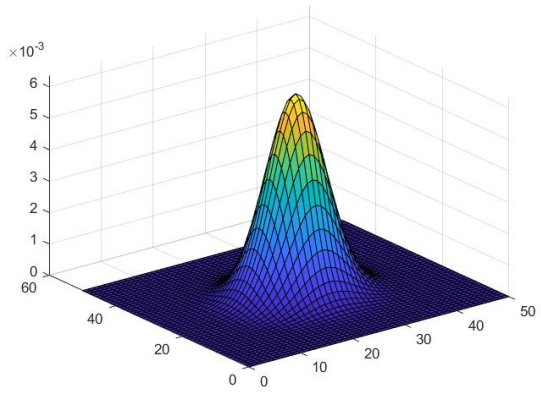
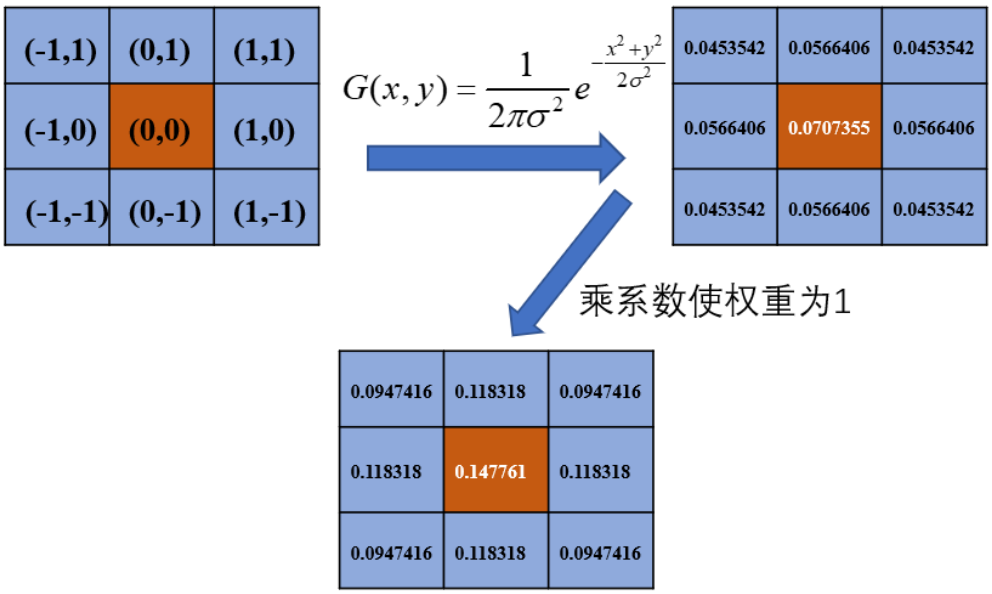


图 3 MATLAB 绘制二维高斯滤波函数

高斯滤波是基于模板的运算，我们称之为高斯核/卷积核。理论上，高斯分布在所有定义域上都有非负值，这就需要一个无限大的卷积核。实际上，仅需要取均值周围 3 倍标准差内的值，以外部份直接去掉即可。

高斯滤波的重要两步就是先找到高斯模板然后再进行卷积。

例如，我们假设中心点坐标是$(0,0)$，那么我们不妨取距离它最近的 8 个点坐标，为了便于计算，我们设定$\sigma=1.5$（影响图像的平滑效果），则模糊半径为 1 的卷积核计算如下。



根据上述卷积核，假设有 9 个像素点，则灰度值(0-255)的高斯滤波的计算方法如下。

最后，我们只需要移动相关核的中心元素，使它位于输入图像待处理像素的正上方，再将输入图像的像素值作为权重乘以相关核，将上面各步得到的结果相加作为输出即可得到高斯滤波后的图像。

高斯滤波后图像被平滑的程度取决于标准差。它的输出是邻域像素的加权平均，同时离中心越近的像素权重越高。因此，相对于均值滤波它的平滑效果更柔和，而且边缘保留的也更好。

高斯滤波具有以下 5 个重要特性：

① 一维二维高斯函数中 μ 是服从正态分布的随机变量的均值，称为期望或均值影响正态分布的位置，实际的图像处理应用中一般取 $\mu=0$；σ 是标准差，σ^2 是随机变量的方差，σ 定义了正态分布数据的离散程度，σ 越大，数据分布越分散，σ 越小，数据分布越集中。在图形或滤波效果上表现为：σ 越大，曲线越扁平，高斯滤波器的频带就越宽，平滑程度就越好，σ 越小，曲线越瘦高，高斯滤波的频带就越窄，平滑程度也越弱。

② 二维高斯函数具有旋转对称性，即滤波器在各个方向上的平滑程度是相同的. 一般来说，一幅图像的边缘方向是事先不知道的，因此，在滤波前是无法确定一个方向上比另一方向上需要更多的平滑. 旋转对称性意味着高斯平滑滤波器在后续边缘检测中不会偏向任一方向。

③ 高斯函数是单值函数。这表明，高斯滤波器用像素邻域的加权均值来代替该点的像素值，而每一邻域像素点权值是随该点与中心点的距离单调增减的。这一性质是很重要的，因为边缘是一种图像局部特征，如果平滑运算对离算子中心很远的像素点仍然有很大作用，则平滑运算会使图像失真。

④ 相同条件下，高斯卷积核的尺寸越大，图像的平滑效果越好，表现为图像越模糊，同时图像细节丢失的越多；尺寸越小，平滑效果越弱，图像细节丢失越少。

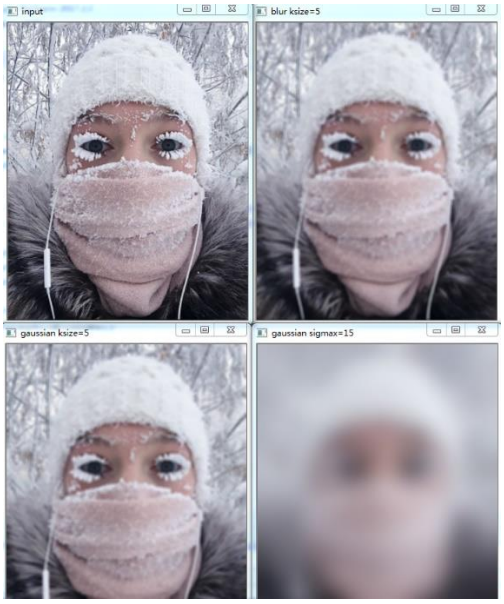


图 4 不同卷积核的高斯滤波效果图

3.3 图像的二值化操作

二值化就是让图像的像素点矩阵中的每个像素点的灰度值为 0（黑色）或者 255（白色），也就是让整个图像呈现只有黑和白的效果。在灰度化的图像中灰度值的范围为 0~255，在二值化后的图像中的灰度值范围是 0 或者 255。而一个像素点在灰度化之后的灰度值如何转化为 0 或者 255 就涉及到阀值的选取问题。

Python-OpenCV 中提供了阈值（threshold）函数，其格式为

threshold(src, thresh, maxval, type, dst=None)

src——指原图像，原图像应该是灰度图。

x——指用来对像素值进行分类的阈值。

y——指当像素值高于（有时是小于）阈值时应该被赋予的新的像素值。

method——阈值类型有以下 5 种。

① THRESH_BINARY：超过阈值的值为最大值，其他值是 0

② THRESH_BINARY_INV：超过阈值的值为 0，其他值为最大值

③ THRESH_TRUNC：超过阈值的值等于阈值，其他值不变

④ THRESH_TOZERO: 超过阈值的值不变，其他值为 0

⑤ THRESH_TOZERO_INV:超过阈值的值为 0，其他值不变

3.3.1 手动设置阈值

全局自定义阈值采用如下函数。

```
# 自定义阈值为 150,大于 150 的是白色 小于的是黑色
```

cv.THRESH_BINARY

opencv 代码如下：

```
ret, binary = cv.threshold(gray, 150, 255, cv.THRESH_BINARY)
    print("yuzhi: %s" % ret)
    cv.imshow("define_yuzhi", binary)
```

 # 自定义阈值为 150,大于 150 的是黑色 小于的是白色

cv.THRESH_BINARY_INV

opencv 代码如下：

```
ret, binary = cv.threshold(gray, 150, 255, cv.THRESH_BINARY_INV)
    print("yuzhi: %s" % ret)
    cv.imshow("define_rev", binary)
```

3.3.2 自适应阈值

全局自适应阈值采用如下函数。

#大律法，全局自适应阈值，参数 0 可改为任意数字但不起作用

cv.THRESH_BINARY | cv.THRESH_OTSU

opencv 代码如下：

```
ret, binary = cv.threshold(gray, 0, 255, cv.THRESH_BINARY | cv.THRESH_OTSU)
    print("yuzhi: %s" % ret)
    cv.imshow("OTSU", binary)
```

#TRIANGLE 法，全局自适应阈值，参数 0 可改为任意数字但不起作用，适用于单个波峰

cv.THRESH_BINARY | cv.THRESH_TRIANGLE

opencv 代码如下：

```
ret, binary = cv.threshold(gray, 0, 255,cv.THRESH_BINARY | cv.THRESH_TRIANGLE)
    print("yuzhi: %s" % ret)
    cv.imshow("TRIANGLE", binary)
```

3.4 形态学操作

形态学一般指生物学中研究动物和植物结构的一个分支。用数学形态学（也称图像代数）表示以形态为基础对图像进行分析的数学工具。基本思想是用具有一定形态的结构元素去度量和提取图像中的对应形状以达到对图像分析和识别的目的。形态学图像处理的数学基础和所用语言是集合论。形态学图像处理的应用可以简化图像数据，保持它们基本的形状特性，并除去不相干的结构。其还多用于图像与处理操作（去噪，形状简化）、图像增强（骨架提取，细化，凸包及物体标记）、物体背景分割及物体形态量化等场景中。值得注意的是，形态学操

作的对象是二值化图像。

形态学图像处理的基本运算有：膨胀、腐蚀、开操作和闭操作、击中与击不中变换，TOP-HAT 变换，黑帽变换等

形态学的应用有消除噪声、边界提取、区域填充、连通分量提取、凸壳、细化、粗化等；分割出独立的图像元素，或者图像中相邻的元素；求取图像中明显的极大值区域和极小值区域；求取图像梯度

3.4.1 形态学膨胀

膨胀是腐蚀运算的对偶运算，其作用是在结构元素的约束下将与目标区域相接触的背景合并到该目标物中，使目标边界向外部扩张，物体的面积增大了相应数量的点。

A 和 B 是两个集合，则 A 被 B 膨胀定义为

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \varnothing\} \text{ 或}$$

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \subseteq A\}$$

形态学膨胀具体操作方法是：拿一个宽 m,高 n 的矩形作为模板，对图像中的每一个像素 x 做如下处理：像素 x 置于模板的中心，根据模版的大小，遍历所有被模板覆盖的其他像素，修改像素 x 的值为所有像素中最大的值。这样操作的结果会将图像外围的突出点连接并向外延伸。如下图的操作过程。

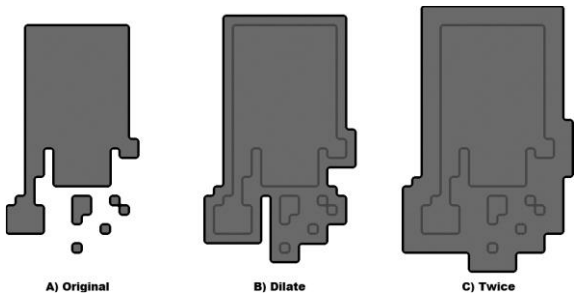


图 5 形态学膨胀原理图

需注意，背景为不同颜色时膨胀也是不同的效果。

3.4.2 形态学腐蚀

腐蚀是一种在结构元素约束下消除目标图形的部分边界点，使其边界向内部收缩的的算法，具有收缩目标区域的作用。

A 和 B 是两个集合，则 A 被 B 腐蚀定义为

$$A\Theta B=\{z\mid(\hat{B})_z\subseteq A\}$$

形态学腐蚀的具体操作方法是：拿一个宽 *m*,高 *n* 的矩形作为模板，对图像中的每一个像素 *x* 做如下处理：像素 *x* 置于模板的中心，根据模版的大小，遍历所有被模板覆盖的其他像素，修改像素 *x* 的值为所有像素中最小的值。这样操作的结果是会将图像外围的突出点加以腐蚀。如下图的操作过程：

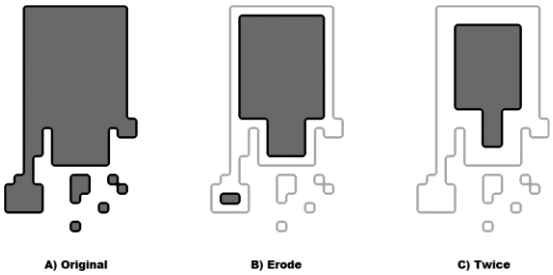


图 6 形态学腐蚀原理图

需注意，背景为不同颜色时腐蚀也是不同的效果。

3.4.3 形态学闭运算

闭运算指的是先膨胀后腐蚀，它的作用是填充目标内部的细小孔洞，将断开的邻近目标连接，在不明显改变物体面积和形状的情况下平滑其边界。

A 和 B 是两个集合，则使用结构元素 B 对集合 A 进行闭操作定义为

$$A\bullet B=(A\oplus B)\Theta B$$

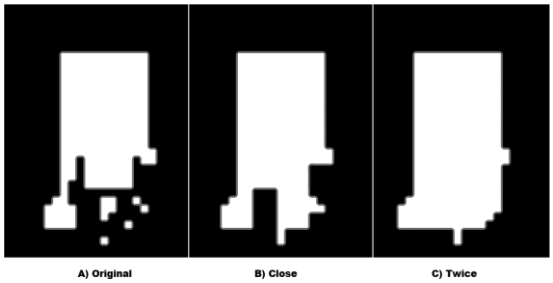


图 7 形态学闭运算原理图

3.4.4 形态学开运算

开运算指的是先腐蚀后膨胀，它的作用是消除图像中细小对象，在纤细点处分离物体和平滑较大物体的边界而有不明显改变其面积和形状。

A 和 B 是两个集合，则使用结构元素 B 对集合 A 进行开操作定义为

$$A\circ B=(A\Theta B)\oplus B$$

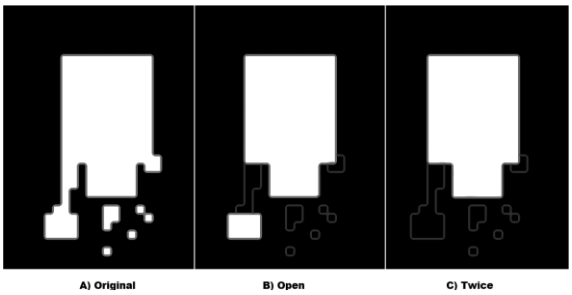


图 8 形态学开运算原理图

4. 代码内容

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import ipywidgets
from ipywidgets import interact
%matplotlib inline
```

1 数据读取，将彩色图像转换为灰度图像，使用 matplotlib 进行显示

```
img_origin = cv.imread("./data/test.jpg") #读取图像，存入 img_origin
img_origin[:,:,0],img_origin[:,:,1],img_origin[:,:,2]=img_origin[:,:,2],img_origin[:,:,1
],img_origin[:,:,0] #RGB 通道顺序调整，方便显示
img_gray=cv.cvtColor(img_origin,cv.COLOR_BGR2GRAY) #彩色图像转换为灰度
图像
plt.figure("Image",figsize=(10, 10)) # 建立画布，设置窗口大小
plt.imshow(img_origin) #显示图像
```

2 高斯滤波，模糊图像去噪声，

```
img_blur=img_gray.copy()

def IntSlider(value, **kwargs):
    """
    Creates an ipwidgets IntSlider with continuous update
    turned off
    """
    return ipywidgets.IntSlider(value, continuous_update=False, **kwargs)

def interactive_blur(x,y):
    global img_blur, img_gray
    fig = plt.figure("Blur",figsize=(20, 20))
    if img_blur is not None: plt.close(fig)
    img_blur = cv.GaussianBlur(img_gray, (2*int(x)+1, 2*int(y)+1), 0) #使用高斯滤
波处理输入 img_gray，输出为 img_blur

    plt.imshow(img_blur,cmap='gray')
```

```
interact(interactive_blur,
        x=IntSlider(value=0, min=0, max=50),
        y=IntSlider(value=0, min=0, max=50),
        );
```

3 图像二值化操作，白色为前景，黑色为背景

方法 1 手动设置阈值，输入为 img_blur，输出为 img_bin，阈值 thresh 由滑动条进行设置

```
img_bin = np.zeros_like(img_blur)
def interactive_manual(thresh):
    global img_blur,img_bin
    fig = plt.figure("Binary",figsize=(20, 20))
    if img_bin is not None: plt.close(fig)
    ret, img_bin = cv.threshold(img_blur, thresh, 255, cv.THRESH_BINARY_INV)
    plt.imshow(img_bin,cmap='gray')

interact(interactive_manual,
        thresh=IntSlider(value=20, min=0, max=255),
        );
```

方法 2 自适应阈值操作，输入为 img_blur，输出为 img_bin

```
img_bin = np.zeros_like(img_blur)
def interactive_adapt(blocksize,thresh):
    global img_blur,img_bin
    fig = plt.figure("Binary",figsize=(20, 20))
    if img_bin is not None: plt.close(fig)
    img_bin = cv.adaptiveThreshold(img_blur, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY_INV, 2*int(blocksize)+1, int(thresh))
    plt.imshow(img_bin,cmap='gray')

interact(interactive_adapt,
        blocksize=IntSlider(value=10, min=0, max=50),
        thresh=IntSlider(value=20, min=0, max=100),
        );
```

3 形态学操作，用于去噪和处理连通域形态，输入为 img_bin，输出为 img_morph

方法 1 形态学膨胀，参数 x 为 x 方向膨胀系数，参数 y 为 y 方向膨胀系数

```
img_morph = img_bin.copy()
def interactive_morph_dilate(x,y):
    global img_morph,img_bin
    fig = plt.figure("Binary",figsize=(20, 20))
    if img_morph is not None: plt.close(fig)
    se = cv.getStructuringElement(cv.MORPH_RECT, (x, y), (-1, -1))
```

```
    img_morph = cv.morphologyEx(img_bin, cv.MORPH_DILATE, se)
    plt.imshow(img_morph,cmap='gray')

interact(interactive_morph_dilate,
            x=IntSlider(value=1, min=1, max=100),
            y=IntSlider(value=1, min=1, max=100),
            );
```

方法 2 形态学腐蚀，参数 x 为 x 方向腐蚀系数，参数 y 为 y 方向腐蚀系数

```
img_morph = img_bin.copy()
def interactive_morph_erode(x,y):
    global img_morph,img_bin
    fig = plt.figure("Binary",figsize=(20, 20))
    if img_morph is not None: plt.close(fig)
    se = cv.getStructuringElement(cv.MORPH_RECT, (x, y), (-1, -1))
    img_morph = cv.morphologyEx(img_bin, cv.MORPH_ERODE, se)
    plt.imshow(img_morph,cmap='gray')

interact(interactive_morph_erode,
            x=IntSlider(value=1, min=1, max=100),
            y=IntSlider(value=1, min=1, max=100),
            );
```

方法 3 形态学闭运算，先膨胀后腐蚀

```
img_morph = img_bin.copy()
def interactive_morph_dilate_erode(x_dilate,y_dilate,x_erode,y_erode):
    global img_morph,img_bin
    fig = plt.figure("Binary",figsize=(20, 20))
    if img_morph is not None: plt.close(fig)
    se_dilate = cv.getStructuringElement(cv.MORPH_RECT, (x_dilate, y_dilate), (-1,
-1))
    img_morph = cv.morphologyEx(img_bin, cv.MORPH_DILATE, se_dilate)
    se_erode = cv.getStructuringElement(cv.MORPH_RECT, (x_erode, y_erode), (-1,
-1))
    img_morph = cv.morphologyEx(img_morph, cv.MORPH_ERODE, se_erode)

    plt.imshow(img_morph,cmap='gray')

interact(interactive_morph_dilate_erode,
            x_dilate=IntSlider(value=1, min=1, max=100),
            y_dilate=IntSlider(value=1, min=1, max=100),
            x_erode=IntSlider(value=1, min=1, max=100),
            y_erode=IntSlider(value=1, min=1, max=100),
            );
```

方法 4 形态学开运算，先腐蚀后膨胀

```
img_morph = img_bin.copy()
```

```
def interactive_morph_erode_dilate(x_erode,y_erode,x_dilate,y_dilate):
    global img_morph,img_bin
    fig = plt.figure("Binary",figsize=(20, 20))
    if img_morph is not None: plt.close(fig)
    se_erode = cv.getStructuringElement(cv.MORPH_RECT, (x_erode, y_erode), (-1,
-1))
    img_morph = cv.morphologyEx(img_bin, cv.MORPH_ERODE, se_erode)
    se_dilate = cv.getStructuringElement(cv.MORPH_RECT, (x_dilate, y_dilate), (-1,
-1))
    img_morph = cv.morphologyEx(img_morph, cv.MORPH_DILATE, se_dilate)

    plt.imshow(img_morph,cmap='gray')

interact(interactive_morph_erode_dilate,
            x_erode=IntSlider(value=1, min=1, max=100),
            y_erode=IntSlider(value=1, min=1, max=100),
            x_dilate=IntSlider(value=1, min=1, max=100),
            y_dilate=IntSlider(value=1, min=1, max=100)
            );
```

4 根据处理后的二值化图像进行目标定位，输出图像为 img_show

```
mask_and = np.zeros_like(img_bin)
img_findcontour=np.zeros_like(img_bin)
mask = np.zeros_like(img_bin)

height, width = img_bin.shape
cv.rectangle(mask_and, (int(width/4), int(height/4)), (int(width/4*3), int(height/4*3)),
(255), -1)#矩形
img_findcontour=cv.bitwise_and(img_morph,mask_and)

contours, hierarchy = cv.findContours(img_findcontour, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)
index = 0
max_area = 0
for c in range(len(contours)):
    area = cv.contourArea(contours[c])
    if area > max_area:
        max_area = area
        index = c
cv.drawContours(mask, contours, index, (255), -1)

img_show = img_origin.copy()

rect = cv.minAreaRect(contours[index])
box = cv.boxPoints(rect)
```

```
box = np.int0(box)
cv.drawContours(img_show,[box],0,(0,0,255),3)

plt.figure("img_show",figsize=(10, 10)) # 图像窗口名称
plt.imshow(img_show)
```

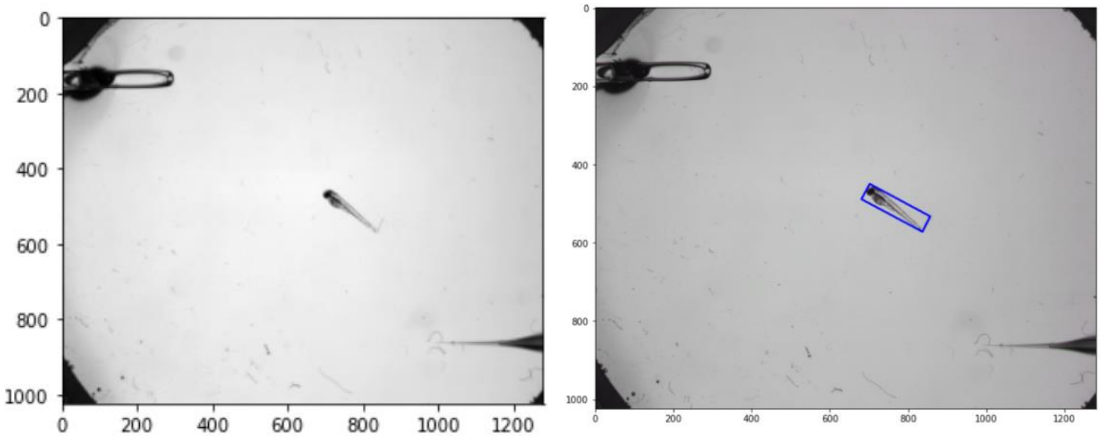
5 期望输出图像

```
img_expect = cv.imread("./data/expected.png") #读取图像，存入 img_origin
#
img_expect[:,:,0],img_expect[:,:,1],img_expect[:,:,2]=img_expect[:,:,2],img_expect[:,:,1],img_expect[:,:,0] #RGB 通道顺序调整，方便显示
plt.figure("Image",figsize=(10, 10)) # 建立画布，设置窗口大小
plt.imshow(img_expect) #显示图像
```

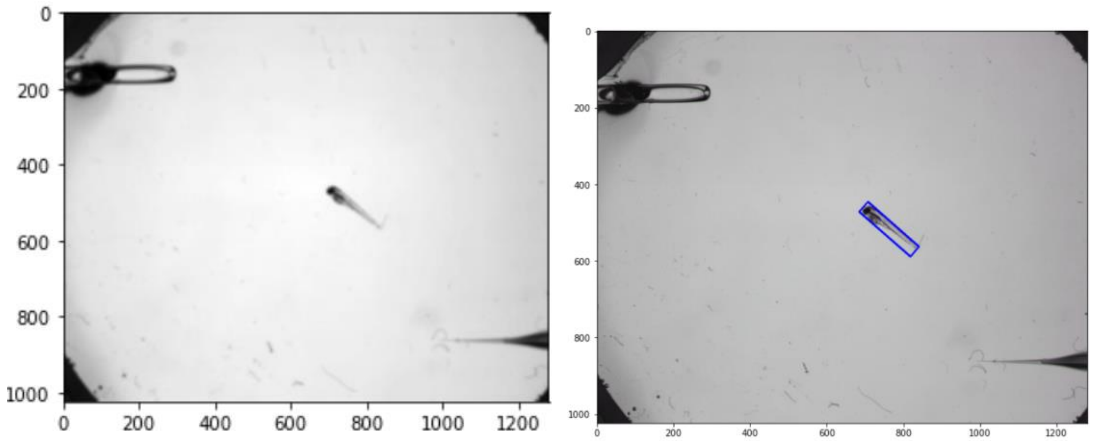
5. 实验结果及讨论

5.1 卷积核对输出图像的影响

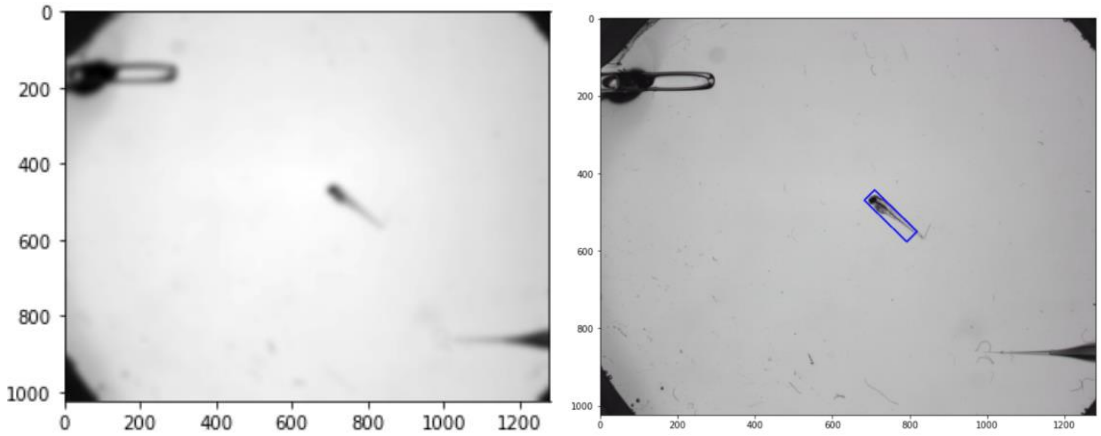
控制标准差 $\sigma=0$，改变卷积核矩阵的大小，图像二值化操作均采用自适应阈值（blocksize=50, thresh=15），形态学操作均采用形态学闭运算（x_dilate=7, y_dilate=7, x_erode=7, y_erode=7），可以得到图 10 所示的目标定位情况。



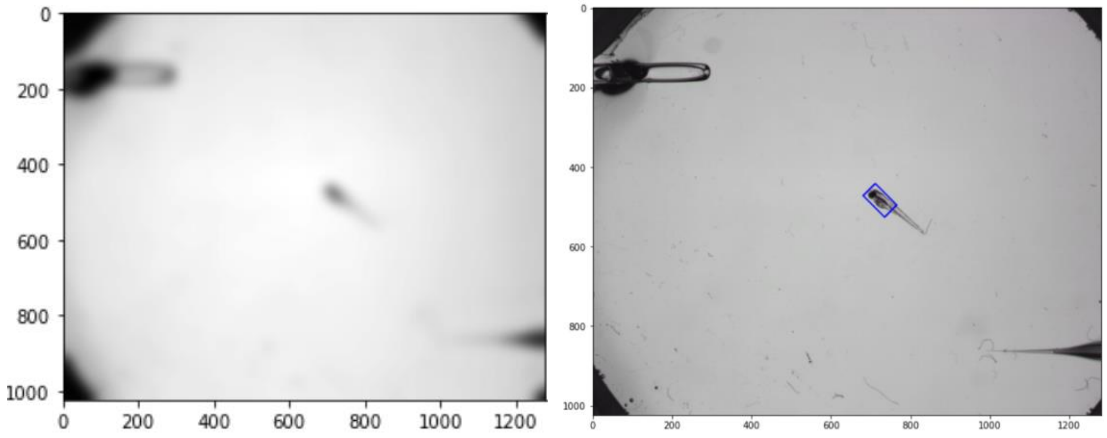
x=0, y=0



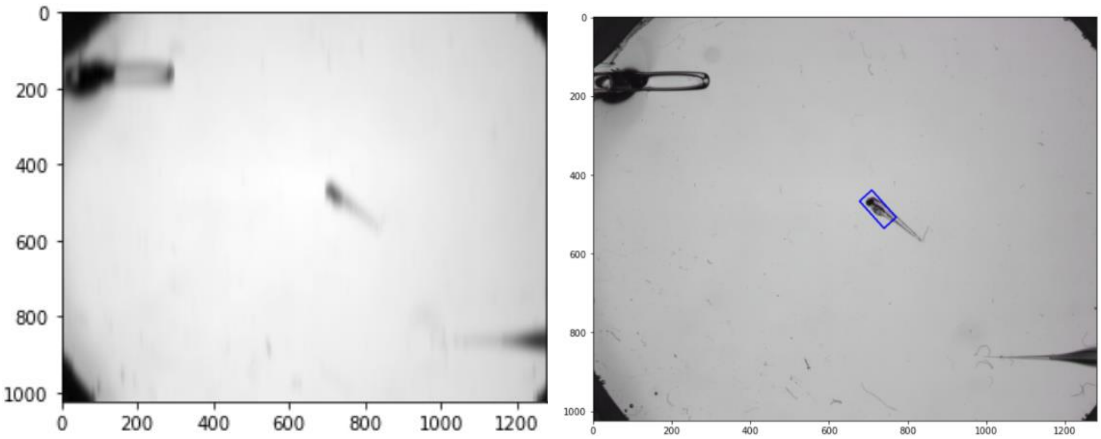
x=7, *y*=7



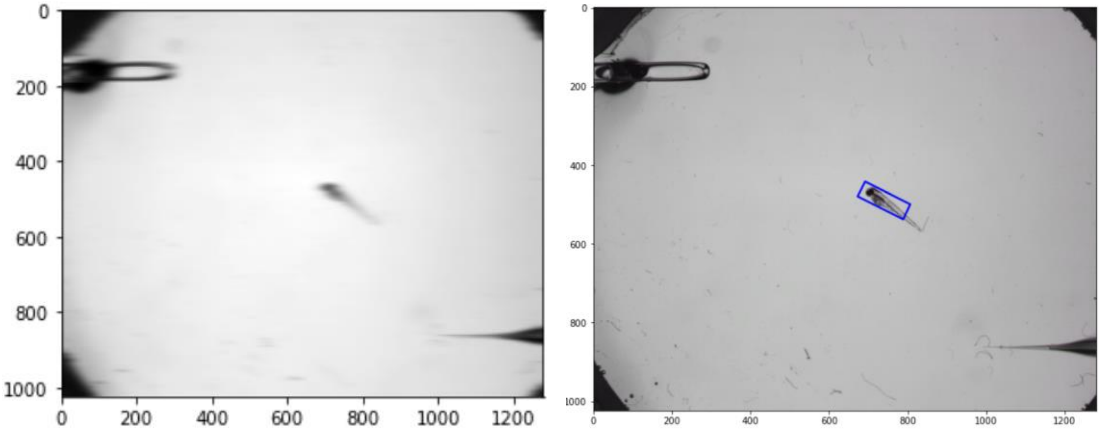
x=25, *y*=25



x=50, *y*=50



x=0, *y*=50



x=50, *y*=0

图 10　σ 固定时卷积核对图像的影响

从图 10 可以看出：

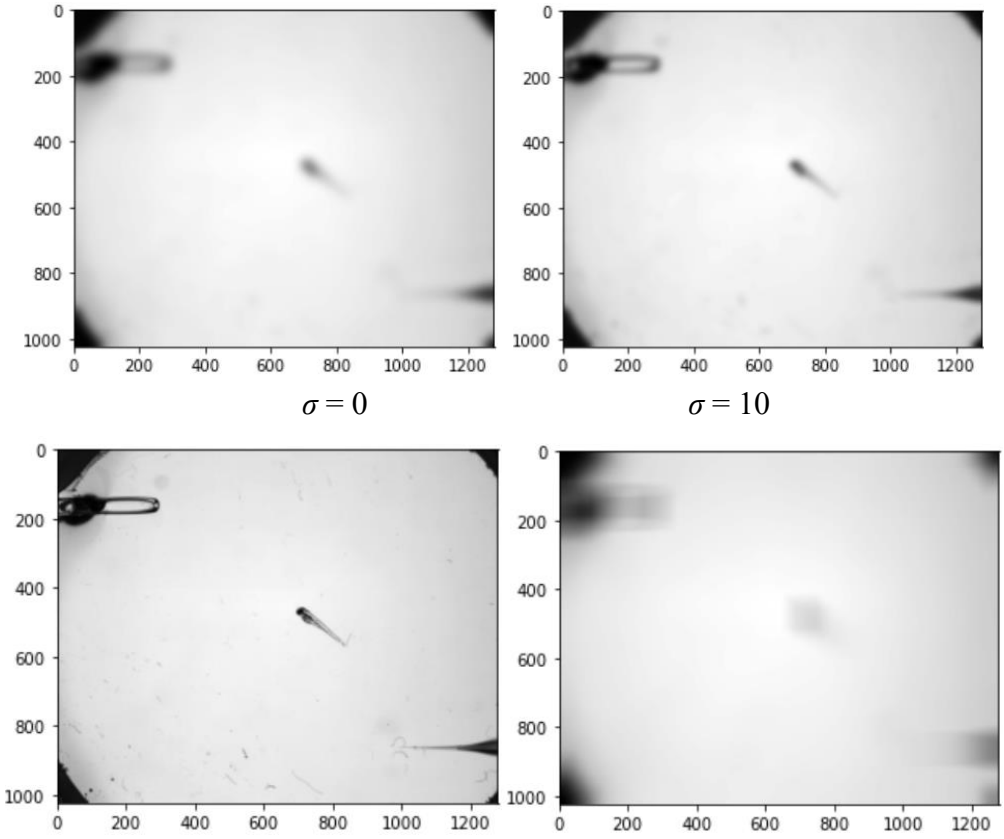
（1）**σ** 固定时，核越大，对图像数据处理的范围及其变化程度越大，图像也就越模糊。

（2）**σ** 固定时，核越小，对图像数据处理的范围及其变化程度越小，图像变化越小。

（3）**σ** 固定时，卷积核越大，相同条件下目标识别范围越小。且卷积核大小为 15×15 矩阵时高斯滤波平滑度最佳，相同条件下目标识别范围越精准。

5.2 标准差 σ 对输出图像的影响

控制卷积核为 101×101 矩阵，改变 σ，得图 9。



$\sigma = 0$　　$\sigma = 10$

$\sigma = 0.1$　　$\sigma = 100$

图 9　卷积核固定时 σ 对图像的影响

从图 9 可以看出：

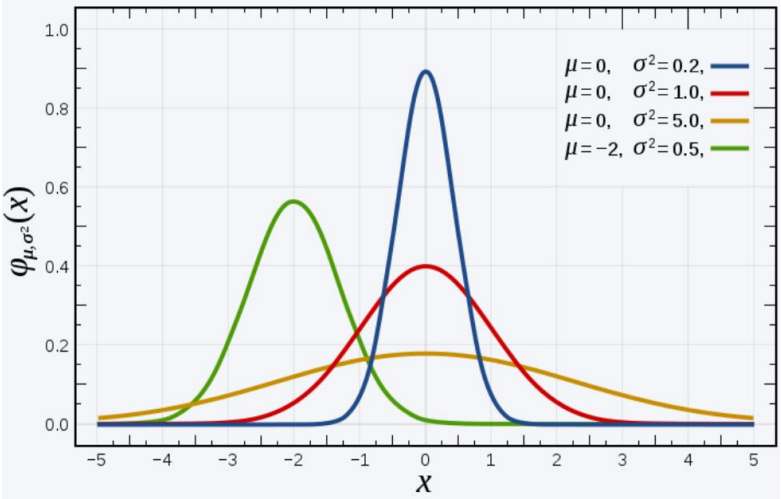
（1）在核大小固定的情况下，σ 值越大，权值分布越平缓。因此，邻域各个点的值对输出值的影响越大，最终结果造成图像越模糊。

（2）在核大小固定的情况下，σ 值越小，权值分布越突起。因此，邻域各个点的值对输出值的影响越小，图像变化也越小。假如中心点权值为 1，其他点权值为 0，那么最终结果是图像没有任何变化。

（3）由 5.1 结论可知，σ 值越大所造成的图形平滑度、模糊度增加会导致目标识别范围精准度下降。

因此，我们可以取 $\sigma = 0$，此时平滑度变化情况为最佳。

总结来说，高斯核可以看成是与中心距离负相关的权重。平滑时，调整 σ 实际是在调整周围像素对当前像素的影响程度，调大 σ 即提高了远处像素对中心像素的影响程度，滤波结果也就越平滑。高斯曲线随 σ 变化的曲线如下。



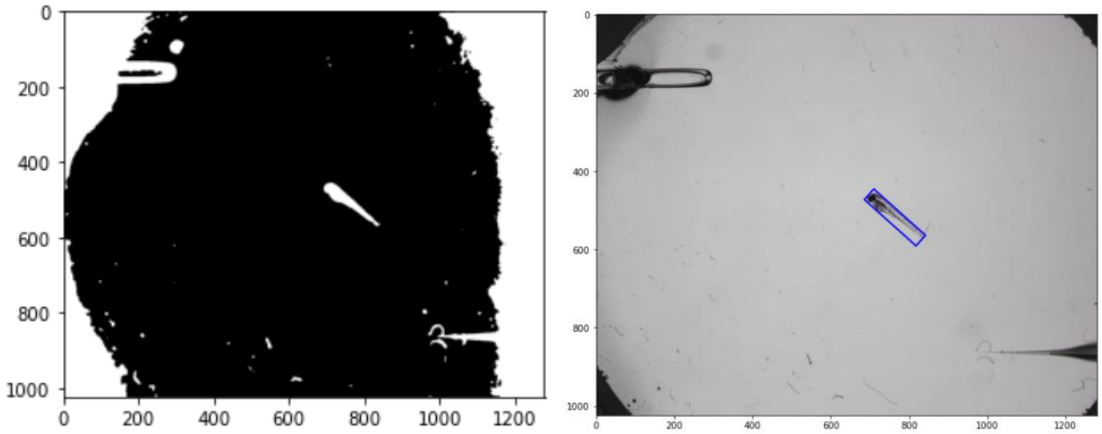
5.3 全局阈值与自适应阈值的效果比较

全局固定阈值是对整幅图像都是用一个统一的阈值来进行二值化。

局部自适应阈值则是根据像素的邻域块的像素值分布来确定该像素位置上的二值化阈值。这样做的好处在于每个像素位置处的二值化阈值不是固定不变的，而是由其周围邻域像素的分布来决定的。亮度较高的图像区域的二值化阈值通常会较高，而亮度较低的图像区域的二值化阈值则会相适应地变小。不同亮度、对比度、纹理的局部图像区域将会拥有相对应的局部二值化阈值。常用的局部自适应阈值有：局部邻域块的均值（所有像素周围的权值相同）、局部邻域块的高斯加权和（每个像素周围像素的权值则根据其到中心点的距离通过高斯方程得到）。在 opencv 中，局部邻域块的均值由 ADAPTIVE_THRESH_MEAN_C 实现。局部邻域块的高斯加权和由 ADAPTIVE_THRESH_GAUSSIAN_C 实现。

高斯滤波参数选取 5.1 和 5.2 中的最优值，下面我们讨论全局阈值与自适应阈值对最终结果的影响。

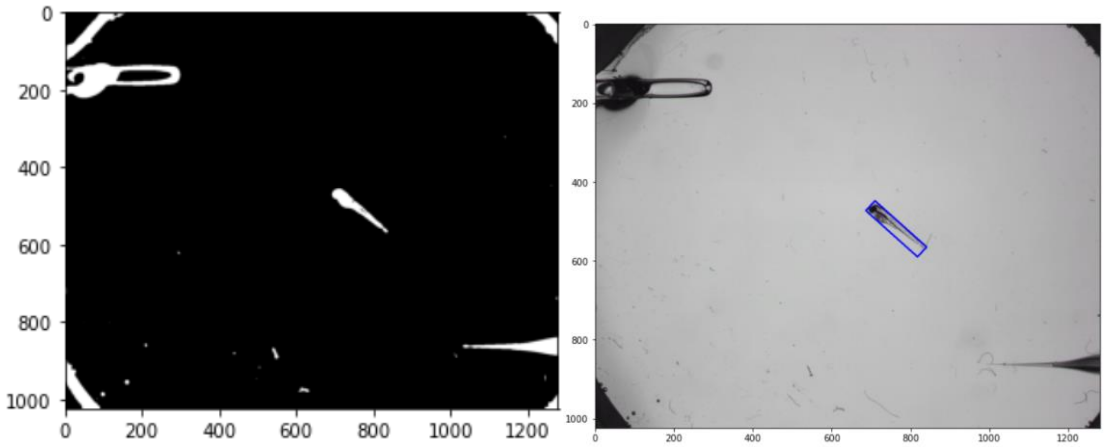
当使用全局阈值对图像进行二值化处理时，取阈值分割参数 thresh=174 时，二值化效果最佳。如若选取形态学闭运算，其参数设置为 x_dilate=8, y_dilate=8, x_erode=8, y_erode=8，则目标定位情况为



thresh=174

图 11 全局阈值

当使用自适应阈值对图像进行二值化处理时，取阈值分割参数 blocksize=50, thresh=18 时，二值化效果最佳。如若选取形态学闭运算，其参数设置为 x_dilate=8, y_dilate=8, x_erode=8, y_erode=8，则目标定位情况为



blocksize=50, thresh=18

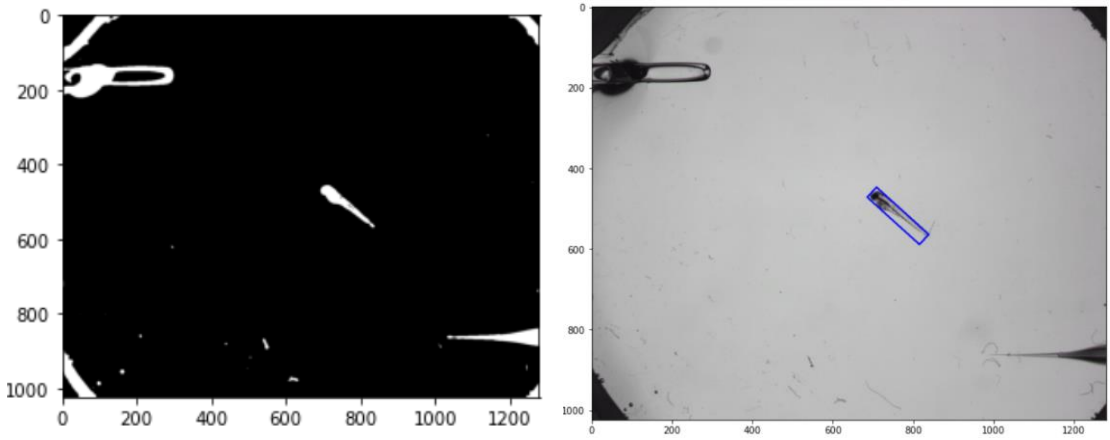
图 12 自适应阈值

由图 11 和图 12 可以看出，采用全局固定阈值的方法对光照不均匀区域的图像容易出现错误的二值分割，即出现错误的白色区域块。采用局部自适应阈值得到的图像更为清晰，二值化结果更符合实际需求。

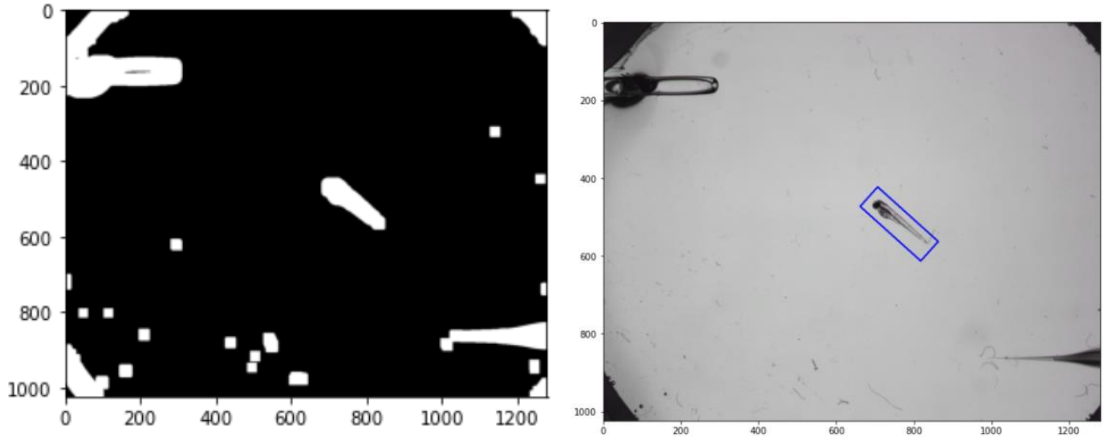
5.4 不同形态学操作对去噪和处理连通域效果的影响

5.4.1 形态学膨胀

高斯滤波参数选取 5.1 和 5.2 中的最优值，同时采用自适应阈值的方法进行二值化处理（blocksize=50, thresh=18），下面我们讨论不同形态学操作对最终结果的影响。



x=1, *y*=1

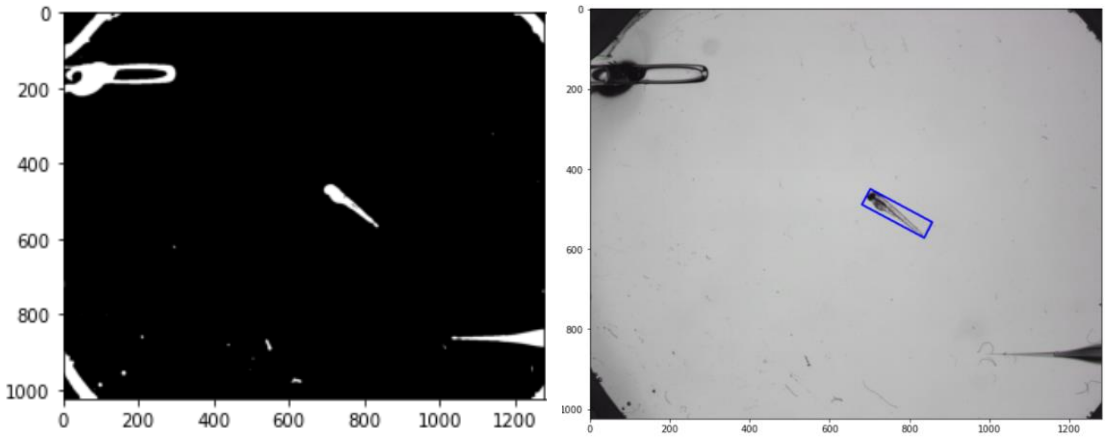


x=25, *y*=25

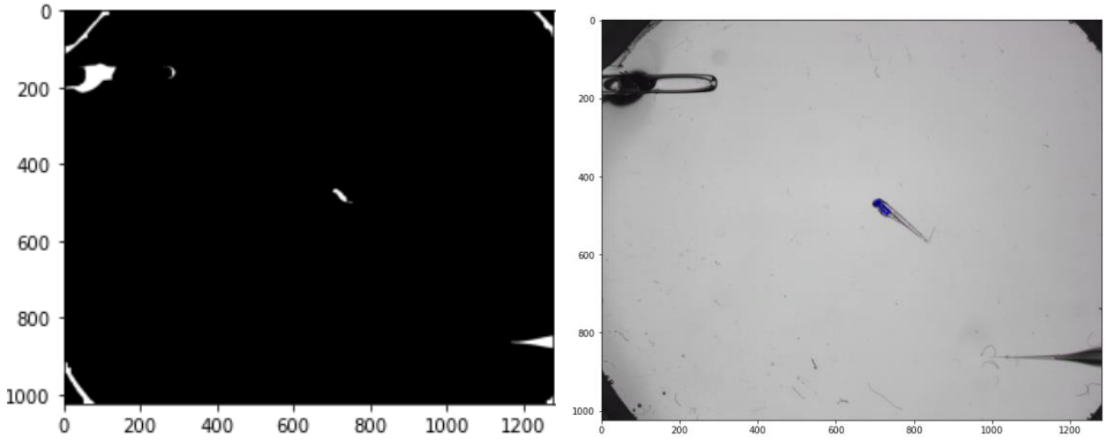
图 13 形态学膨胀不同参数设置对输出图像的影响

从图 13 可以看出，形态学膨胀操作将图像的轮廓加以膨胀。图 13 中，参数 *x* 为 *x* 方向的膨胀系数，参数 *y* 为 *y* 方向的膨胀系数，参数 *x*/*y* 越大，*x*/*y* 方向膨胀程度越大。可以看出，形态学膨胀会放大图像的细节部分。此外，从图 13 可以看出，形态学膨胀会使目标定位的范围扩大。

5.4.2 形态学腐蚀



x=1, *y*=1

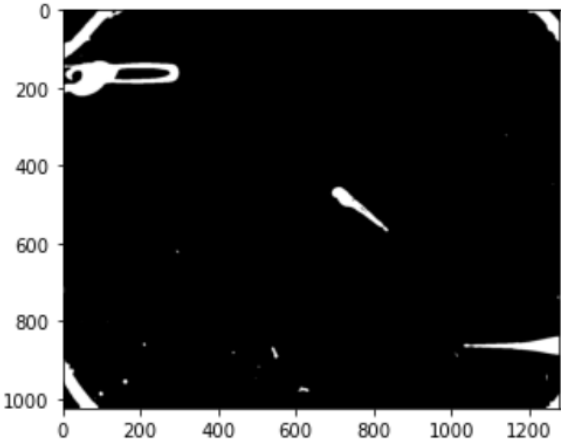


x=15, *y*=15

图 14 形态学腐蚀不同参数设置对输出图像的影响

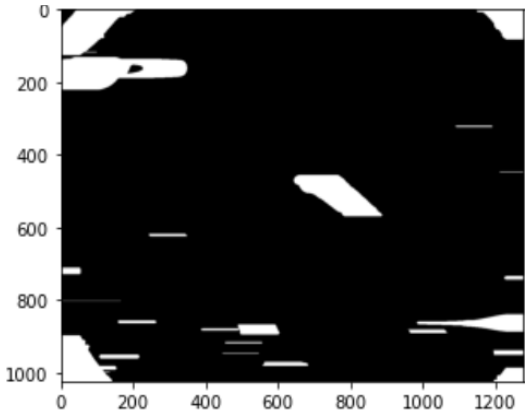
从图 14 可以看出，形态学腐蚀操作将物体的边缘加以腐蚀。图 14 中，参数 *x* 为 *x* 方向的腐蚀系数，参数 *y* 为 *y* 方向的腐蚀系数，参数 *x*/*y* 越大，*x*/*y* 方向腐蚀程度越大。可以看出，形态学腐蚀在一定程度内可以清除图像中的噪声干扰。此外，从图 14 可以看出，形态学腐蚀会使目标定位范围减小。

5.4.3 形态学闭运算



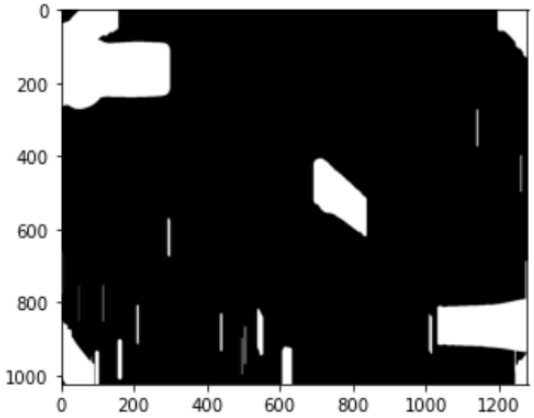
x_dilate=1, y_dilate=1

x_erode=1, y_erode=1



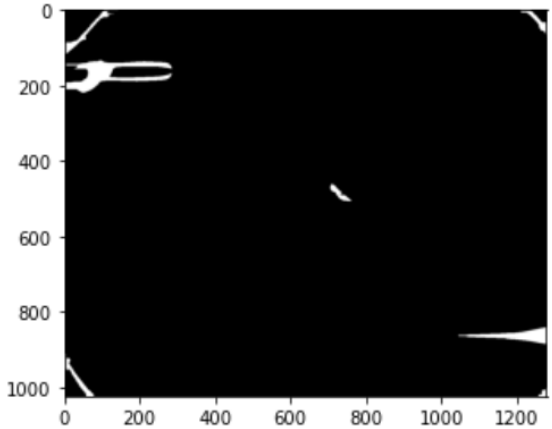
x_dilate=100, y_dilate=1

x_erode=1, y_erode=1



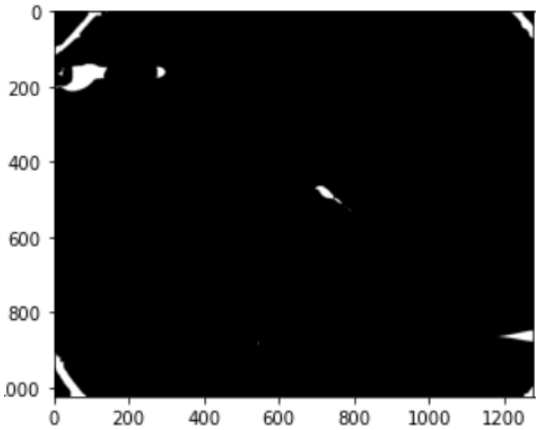
x_dilate=1, y_dilate=100

x_erode=1, y_erode=1



x_dilate=1, y_dilate=1

x_erode=27, y_erode=1



x_dilate=1, y_dilate=1

x_erode=1, y_erode=19

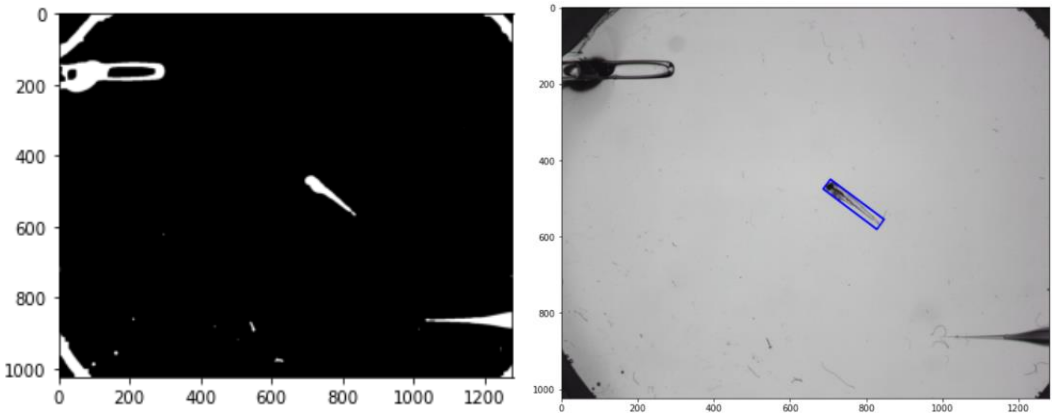
图 15 形态学闭运算不同参数设置对输出图像的影响

图 15 中，参数 x_dilate 为 x 方向膨胀系数，参数 y_dilate 为 y 方向膨胀系数。

参数 x_erode 为 x 方向腐蚀系数，参数 y_erode 为 y 方向腐蚀系数。可以看出，增大参数 x_dilate/y_dilate 的值，可以使图像的各部分在 x/y 方向扩大；增大参数 x_erode/y_erode 的值，可以使图像的各部分在 x/y 方向缩小。

经过适当的参数设置，可以得到图 16 所示的输出图像。形态学闭运算 k 恶意排除小型黑洞，突出比原图轮廓区域更暗的区域，更重要的是，它能将两个区域连接起来，形成连通域。

在本实验中，合适的形态学闭运算可以得到很好的目标定位效果。

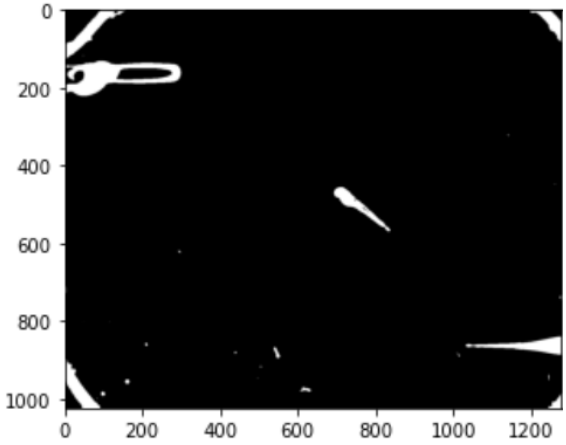


x_dilate=12, y_dilate=12

x_erode=13, y_erode=13

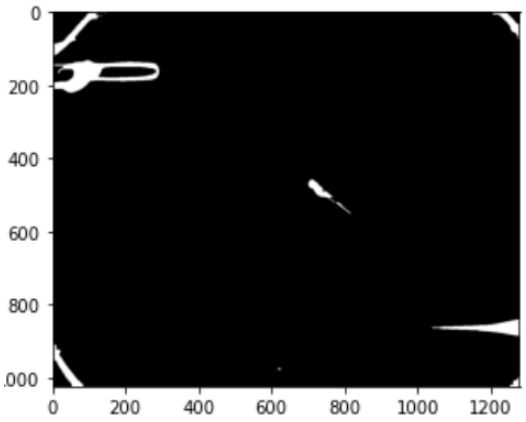
图 16 形态学闭运算输出图像

5.4.4 形态学开运算



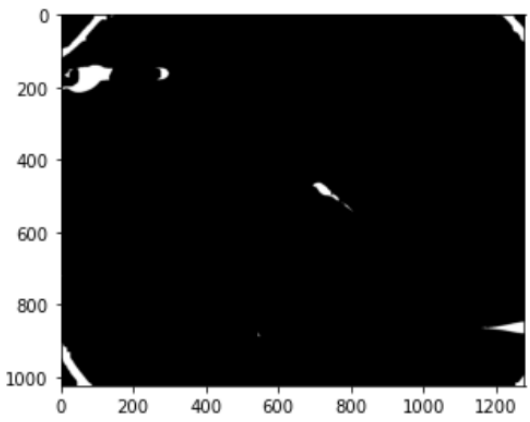
x_erode=1, y_erode=1

x_dilate=1, y_dilate=1



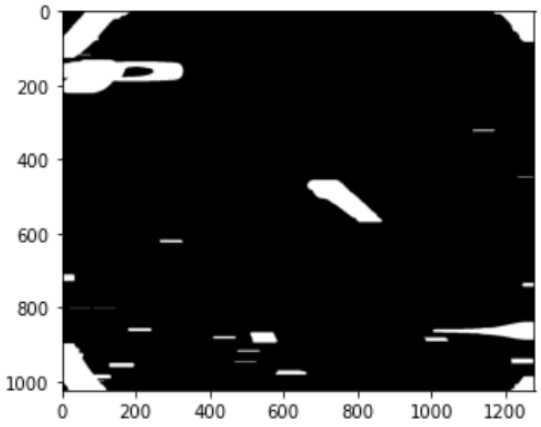
x_ erode=16, y_ erode=1

x_ dilate=1, y_ dilate=1



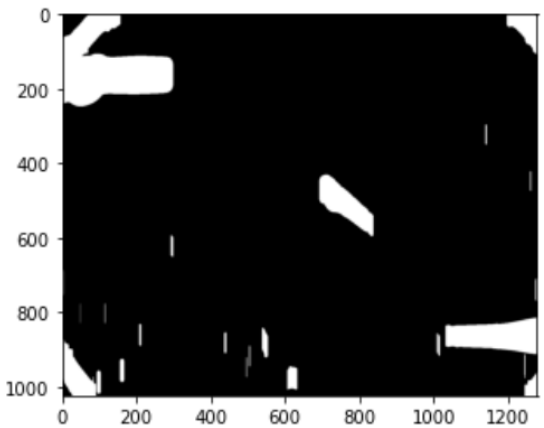
x_dilate=1, y_ erode=16

x_ dilate=1, y_ dilate=1



x_ erode =1, y_ erode =1

x_ dilate =58, y_ dilate =1



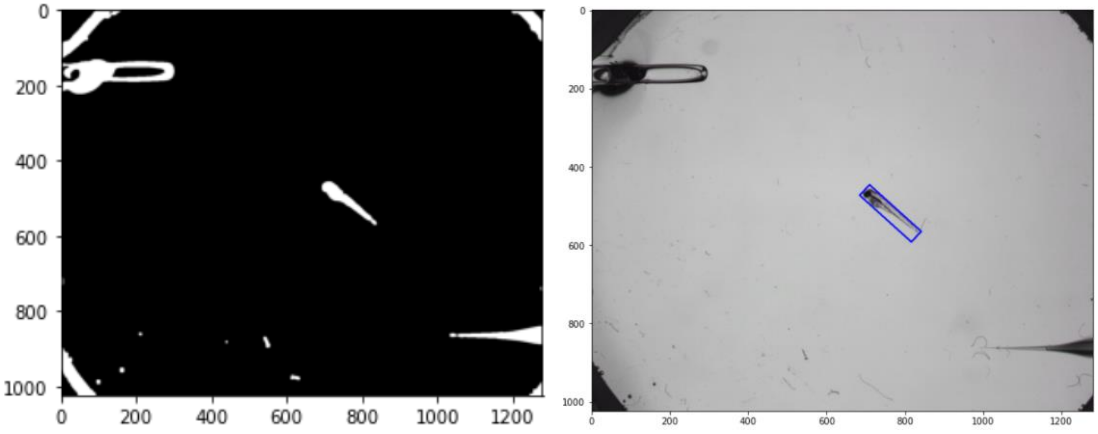
x_ erode =1, y_ erode =1

x_ dilate=1, y_ dilate=52

图 17 形态学开运算不同参数设置对输出图像的影响

从图 17 中看以看出，形态学开运算不同参数对输出图像的影响与形态学闭运算一致。通过选取合适的参数，可以得到图 17 所示的输出图像。

在本实验中，合适的形态学开运算同样可以得到很好的目标定位效果。



x_erode=4, y_erode=4

x_dilate=6, y_dilate=6

图 17 形态学开运算输出图像

形态学开运算可以放大裂缝和低密度区域，消除小物体，在平滑较大物体的边界时，不改变其面积。消除物体表面的突起。

5.5 图像最终输出结果

经过合适的处理并对目标中的斑马鱼幼鱼进行定位，对其轮廓使用最小包围矩形进行标注，同时滤除鱼身周围的灰尘干扰，可以得到图 18 所示的输出图像。其参数设置为

高斯滤波参数：$\sigma = 0$，卷积核大小 x=7, y=7

自适应阈值参数设置：blocksize=50, thresh=15

形态学闭运算参数设置：x_dilate=7, y_dilate=7, x_erode=7, y_erode=7

其与期望输出图像基本一致。

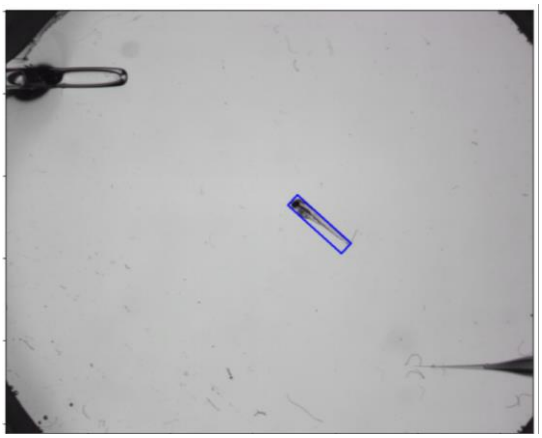


图 18 输出图像

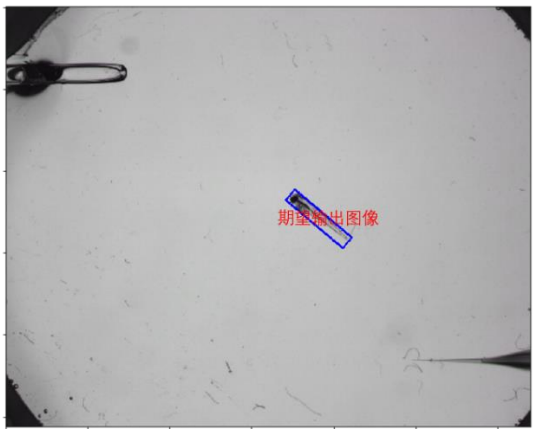


图 19 期望输出图像

6. 感想体会

显微操作目标定位过程让我第一次体会到了图像识别的魅力，感受到了其强大之处。我第一次深刻明白了计算机是如何根据图像的成像原理进行图像处理，并了解了图像灰度化原理的基本原理，进一步明白了各种图像滤波方法对最终图像处理的影响。图像滤波是让我感触最深的一个环节，通过巧妙的运算方式可以很好的达到图像处理的方法，让我不禁感叹计算机处理图像数据的强大之处。此外，图像二值化操作的过程同样令我印象深刻，我第一次体会到了不同阈值设置方法的影响，阈值的选取方式同样十分巧妙。在形态学操作的过程中，我也深刻了解了不同运算方式对最终图像输出的影响。总而言之，这是一次令我收获颇丰的课程。

参考文献

[1] 潘振赣,龚声蓉.浅谈数字图像处理技术的基本原理[J].电脑知识与技术,2010,6(06):1452-1453+1460.

[2] 姚希.数字图像处理技术及其应用[J].电子技术与软件工程,2017(18):87.

[3] 魏莹.高斯滤波 OpenMP 并行化[J].通讯世界,2015(10):194-195.

[4] 余胜威，丁建明，吴婷，魏健蓝编著．MATLAB 图像滤波去噪分析及其应用：北京航空航天大学出版社，2015.09