一、 立体水滴去除

1.1 项目设计方案与思路

这个项目是利用文献[1]的数据集，拟去除掉双目相机上水珠对于环境背景的干扰，也就是去噪的过程。这个项目的难点在于水珠噪声是透明的，与背景的差异很小（如图 7 所示，为训练集一个场景下左右相机在有无水滴情景下所拍摄的图像，以及计算的左右背景图像的视差图）。文献[3]对常用的去噪算法做了一个概述，针对本项目而言，使用传统频率域滤波是不适合的，因为频率域滤波是去除高频噪声的，水滴噪声属于低频噪声。而对于空域去噪方法，如双边滤波、非局部正则化、基于梯度的快速约束电视算法等虽然会去除水滴噪声，但是会对背景图造成极大的平滑，图像细节丢失太多，不适合采用。基于神经网络的自编码器算法在均匀规律噪声上效果较好，比较适合于单目图像恢复，但是对于多目图像会受到视差的影响，图像复原效果一般。

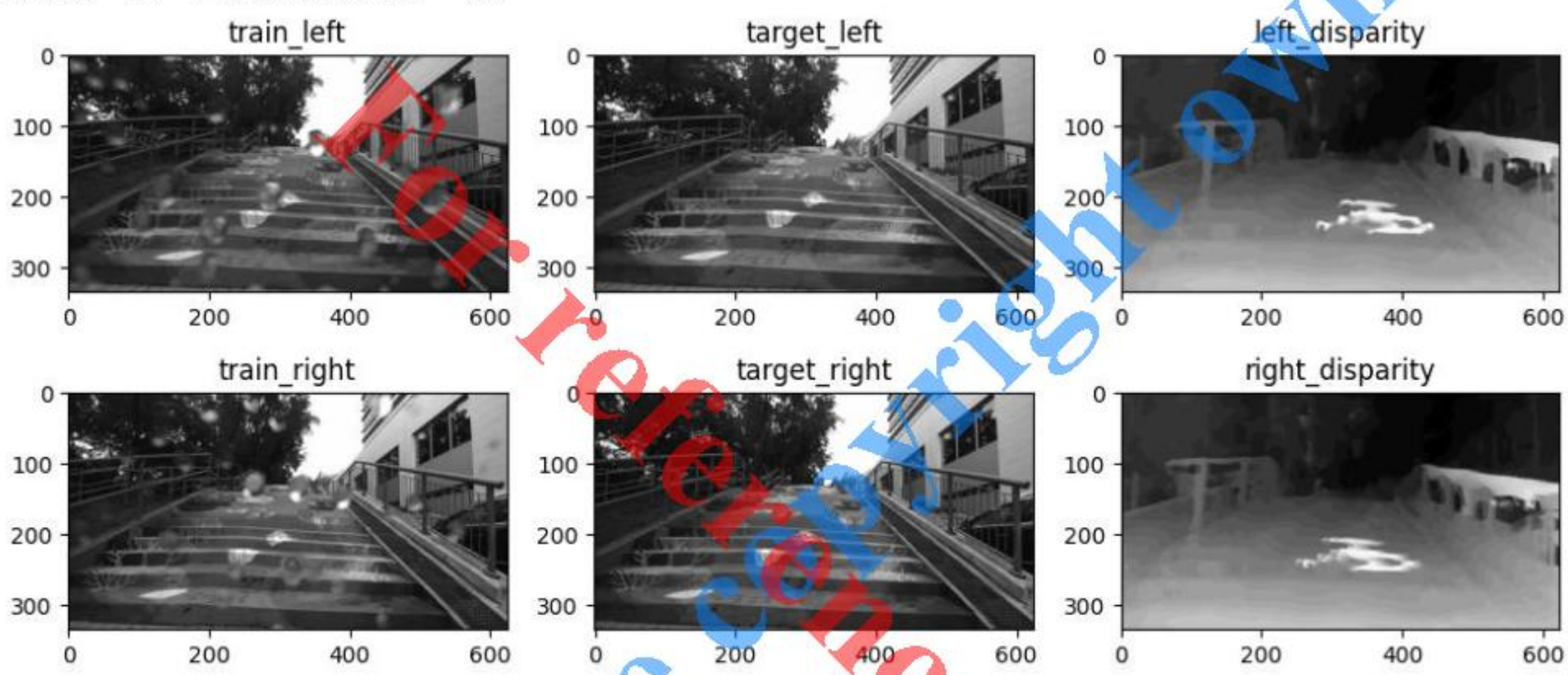

图 1 单样本数据

本文受到文献[2]的启发，提出了‘迭代视差引导修复算法’。即充分利用双目相机的特点，场景在不同相机中的成像位置是不同的，因此就可以通过对偶相机的场景来恢复水滴区域中应有的场景。那么重点问题就是左右图片中水滴噪声位置的确定了，在文献中也是打算采用对偶图像的视差来进行求解的，但是针对这个数据集发现无法这样求解，因为左右图像的水滴太大、太密，并且重合部分很多，如果还使用的是视差来进行查找，那么很明显左右相机的噪声区域会特别大，影响后续水滴的修复。（当然这里也反应出我的算法一个不可解决的问题，即对于左右相机重叠部分的水滴噪声去除效果不佳。因为传统方法无法持续搜索噪声区域，以及更新视差区域，如果使用深度学习方法来训练会达到最为理想的结果。）如果针对具体的任务，我的场景细节要求不高时，可以增加一个后处理操作，使用非局部均值去除噪声接着再次锐化图像就可以得到场景的基本结构。

1.2 涉及的理论与方法

1.2.1 视差

本方法所涉及到比较重要的一个概念就是图像的视差。要认识视差[4]，首先需要认识一下双目视觉系统（图 8），它两个相机的投影中心连线的距离为 b，也叫做基线，三维空间中任意一点 P 在左相机的成像点为P_L，在右相机的成像点为P_R。根据光的直线传播原理可以知道，三维空间点P就是两个相机的投影中心与成像点连线的交点。线段x_L和x_R分别是左右相

For reference only
Chinese the copyright owner

机成像点到左成像面的距离，则点 P 在左右相机的视差可以定义为：

$$d = |x_L - x_R| \tag{1}$$

$$Z = \frac{b * f}{x_L - x_R} = \frac{b * f}{d} \tag{2}$$

根据相似三角形理论，不难得到点 P 到投影中心平面的距离 Z，当点 P 在三维空间上移动时，点 P 在左右相机上的成像位置也会改变，从而视差也会发生相应变化，由(2)式可知，视差与三维空间上的点到投影中心平面的距离成反比，因此只要知道了某点的视差，就可以知道该点的深度信息。

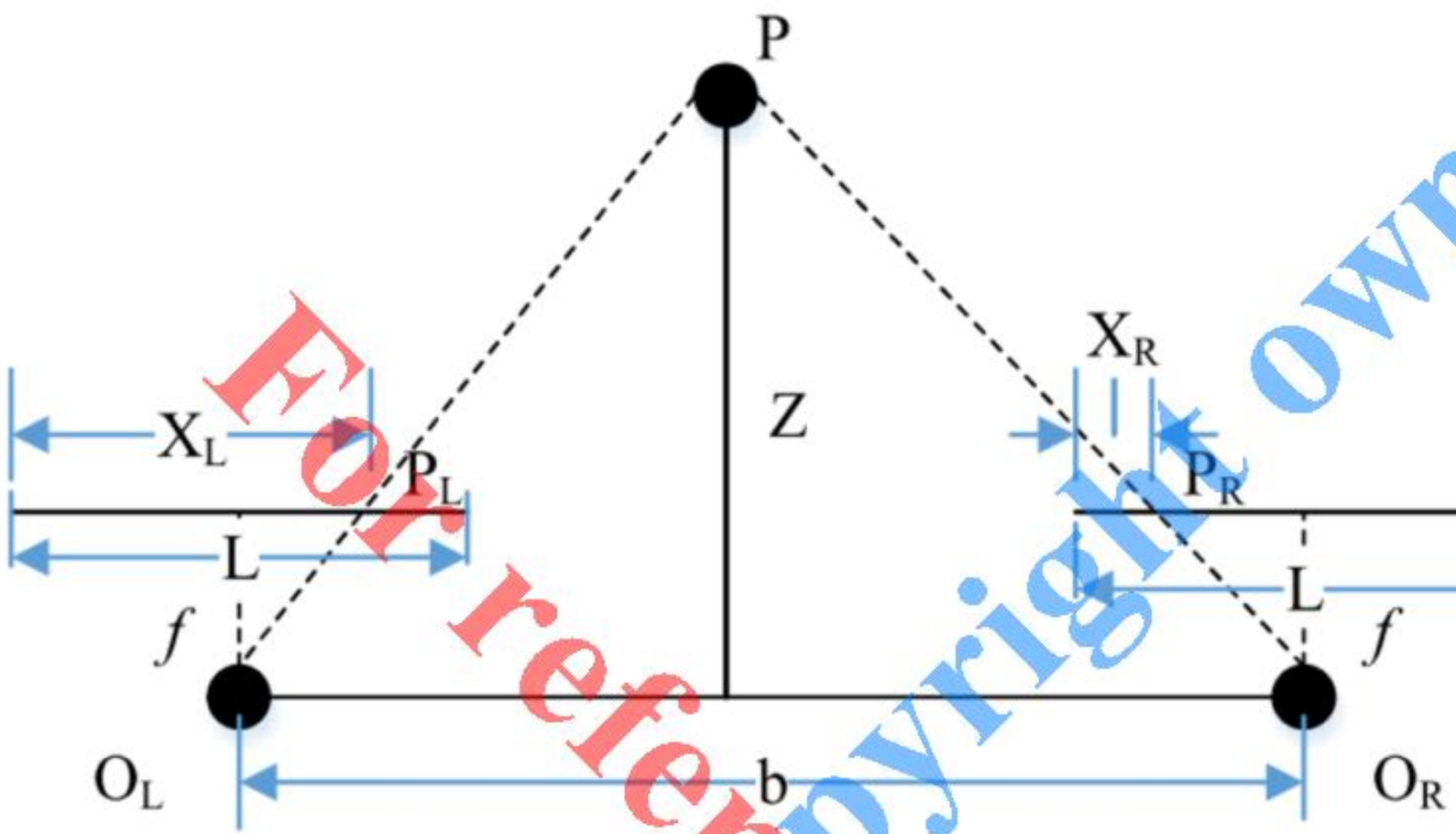

图 2 双目视觉系统

1.2.2 双目立体匹配

当三维空间与图像的对应关系确定后，要想对视差进行计算，就必须知道三维空间的点在左右图像上的对应关系，这就是立体匹配的目的。通过立体匹配技术，就可以明确左右图像中的点的对应关系，从而得到视差，恢复点的三维信息。

本文所使用的匹配方法基于以下几个约束：

- **极线约束。**在左图像中的一点，它在右图像上的 对应匹配点必定在某一条直线上，这条直线就是极线。
- **相似性约束。**在进行立体匹配时，点、线、块等元素一定具有相同或相似的属性。
- **唯一性约束。**对于待匹配图像，在原图像中至多对应一个点。一幅图像上的每个点只能与另一幅 图像上的唯一一个点一一对应，这样图像上的点至多有一个视差值。
- **左右一致性约束。**若左图像上的一点 P，其在右图像上的对应点为 Q，则右图像上的点 Q 在左图像上的对应点应该是点 P，如果这两点不是一一对应的，则匹配会不满足唯一性条件，说明匹配失败。

当然在实际使用时会灵活处理，比如处理左右一致性约束时会加上是否是噪声区域，以及只要对偶图像匹配点间的差值在可以接受的阈值内即可。

上面也提到过，由于使用传统方法无法通过视差优化来进行匹配，因此本文只能不断迭代匹配来尽量减小噪声区域的误差。其次水珠噪声的去除还受到硬件设备的制约，如图 9 所示。因为一个背景物体必须至少用一个相机来观察，因此基线长度需要满足式(5)。

For reference only
Chinese copyright owner.

$$b > \frac{zr}{z-l} \tag{3}$$

其中，b、l、z、r 分别为基线长度、摄像机与防护玻璃的距离、摄像机与最靠近摄像机的物体的距离、噪声大小。

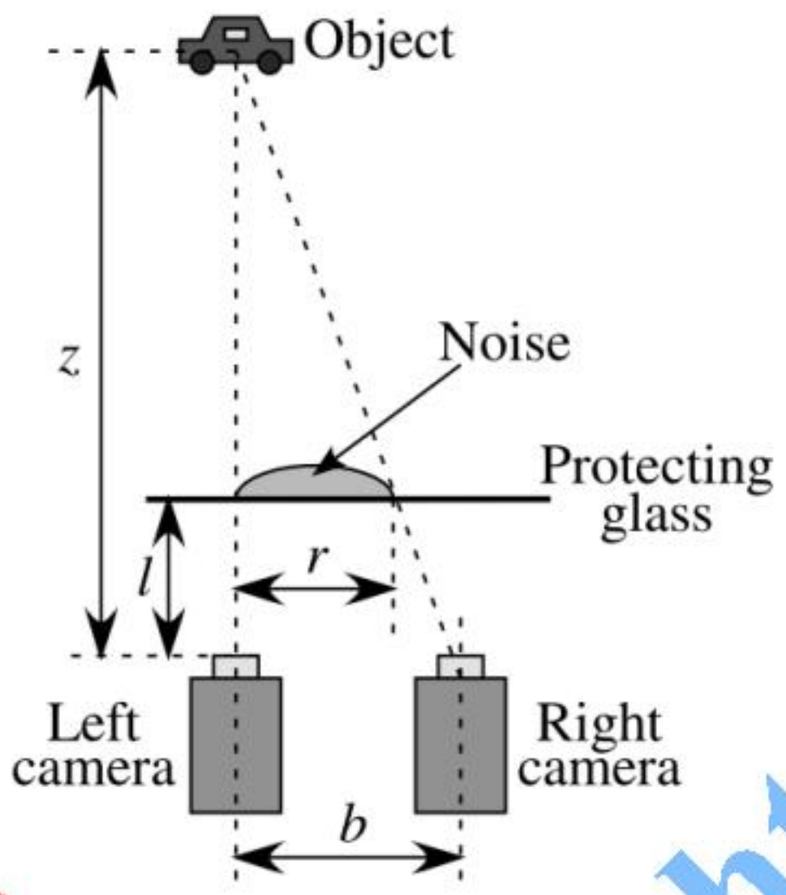

图 3 相机配置对水珠噪声移出的制约

1.2.3 方案设计

基于前面两节的基础，可以执行如图 10 所示的水滴去噪模型，整体思路较为简单。分为图像的预处理、模型求解以及图像后处理三个板块。

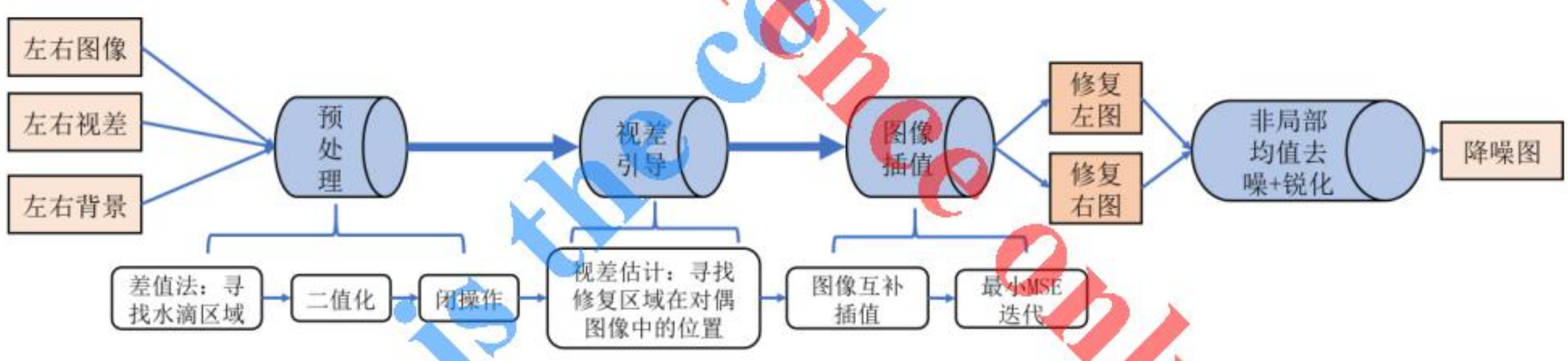

图 4 迭代视差引导修复算法流程图

在图像预处理阶段会使用图像的背景 Ground Truth 与水滴噪声图像进行作差以找到精确的水滴噪声区域，但处理后的差值图为灰度图像，因此需要进行二值化处理，考虑到水滴的透明特点，其灰度值与背景灰度值较为接近，因此直接使用自定义的下阈值进行二值化，这样子可能会产生不必要的噪声，因此还需要通过形态学闭操作来将小噪声都去除掉，避免干扰。接着进行视察引导算法，通过输入的左右 GT 图像的视差来对噪声区域在对偶图像中的位置进行预测；最后是图像插值部分，这一部分是替换水滴区域的像素值为其对偶图像对应区域的像素值，但对偶图像对应区域可能本身也会存在水珠，本文经过实验发现左右一致性约束原则在这里起到反作用的效果，因此针对于这部分的误差，我们只能期望其最小。因此引入了最小 MSE 迭代算法，通过不断交换对应视察位置的像素值来控制图像的 MSE 损失最小。

1.3 实验过程

在这一小节中，我们首先将从训练集中随机针对一个特定的场景进行处理，如图 11 所示

For reference only
Chime is the copyright owner.

为我们算法的输入数据。在处理完这个图像后将会从测试集中选择一个场景进行测试以验证算法的可行性。

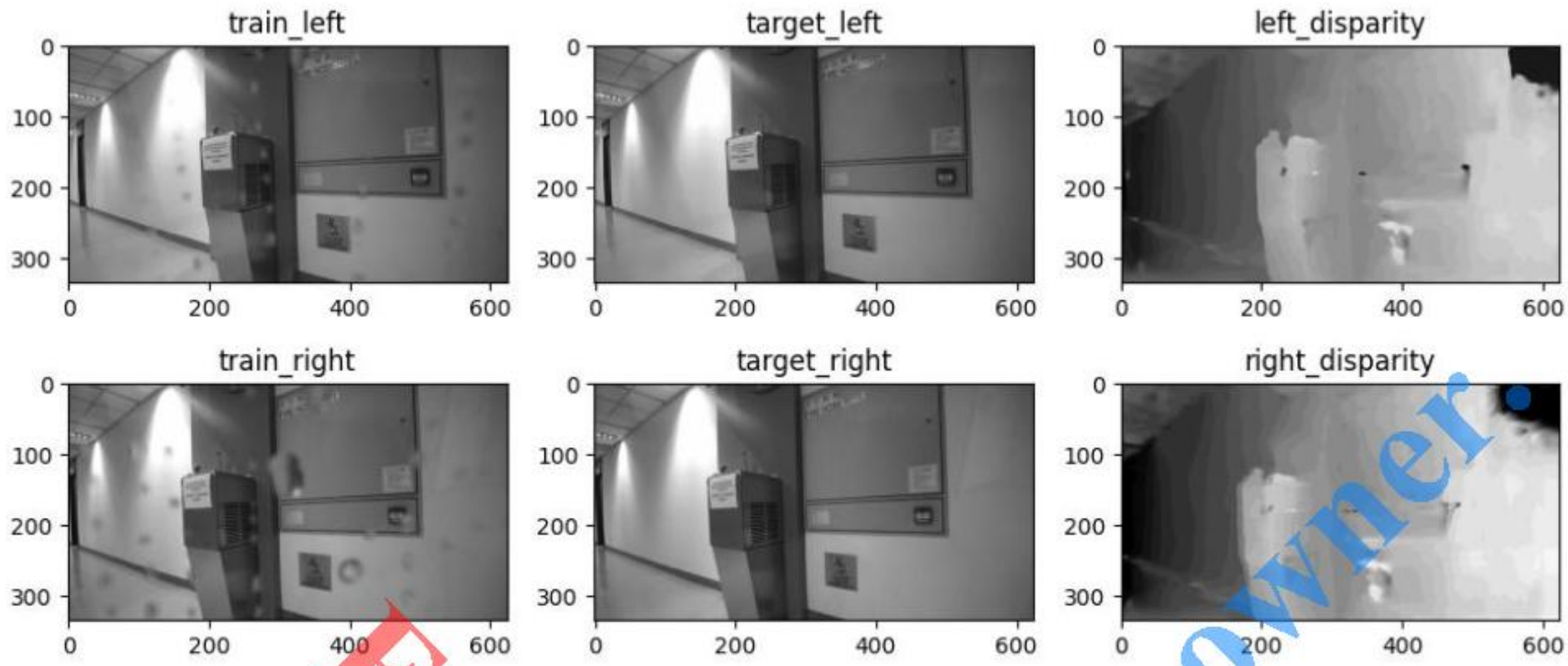

图 5 算法输入数据

1.3.1 图像预处理

首先对图 11 中的第 1 列图像与第 2 列图像进行差值计算，如图 12 所示为可视化结果：

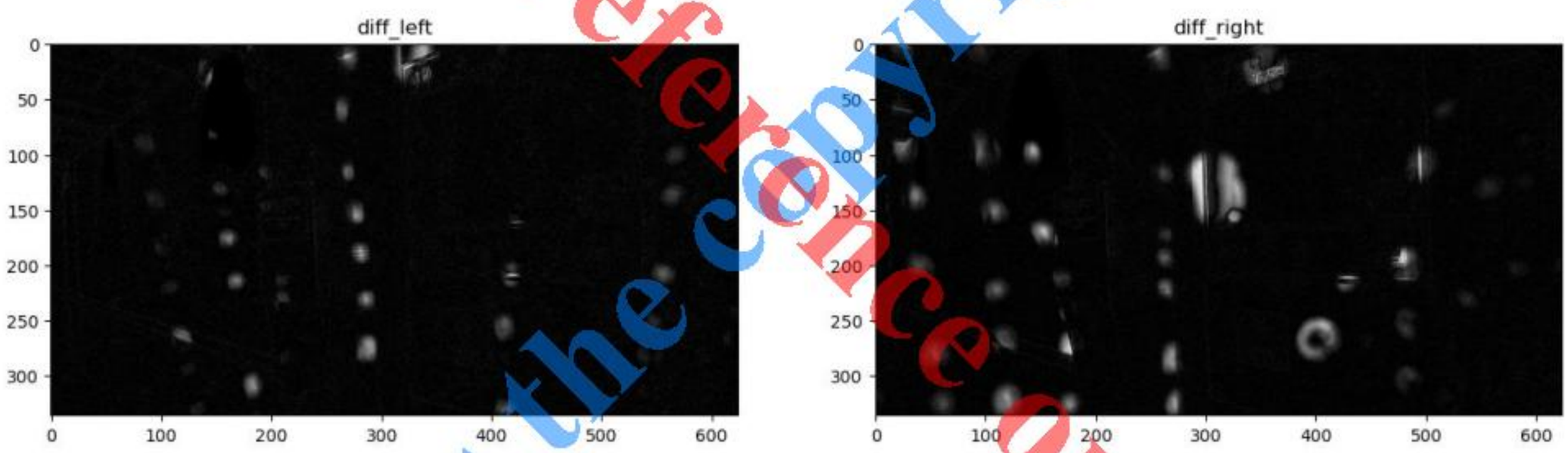

图 6 差值计算结果

接着将使用自定义阈值来对上面的差值进行二值化，然后进行闭操作，得到图 13 的结果。我们在原图上将找到的水滴区域绘制出来看看是否正确，如图 14 所示。

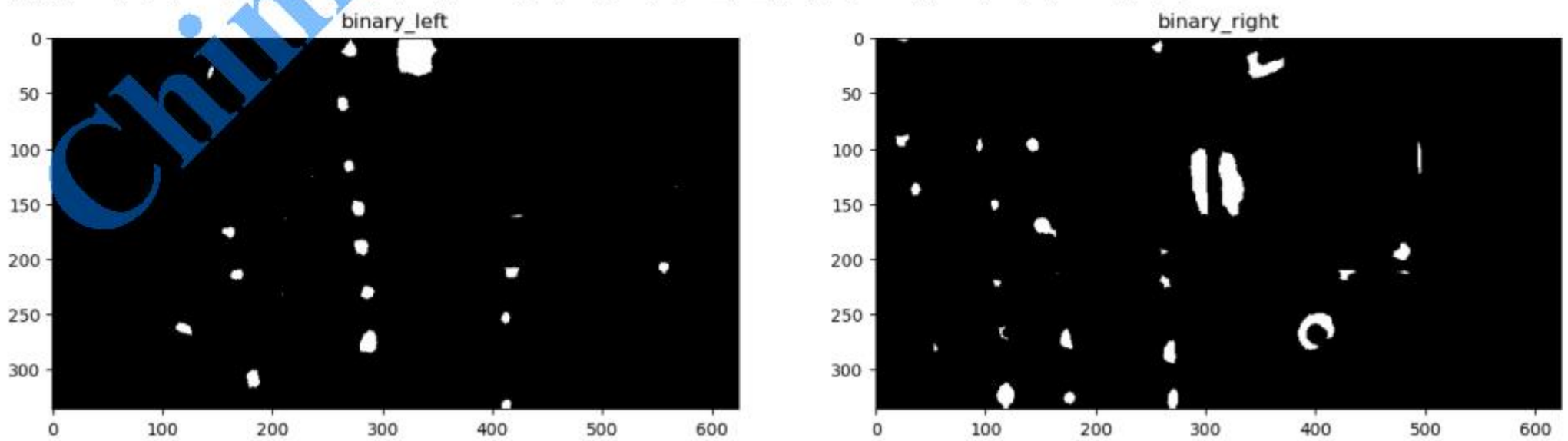

图 7 二值化图

For reference only
Chinese is the copyright owner

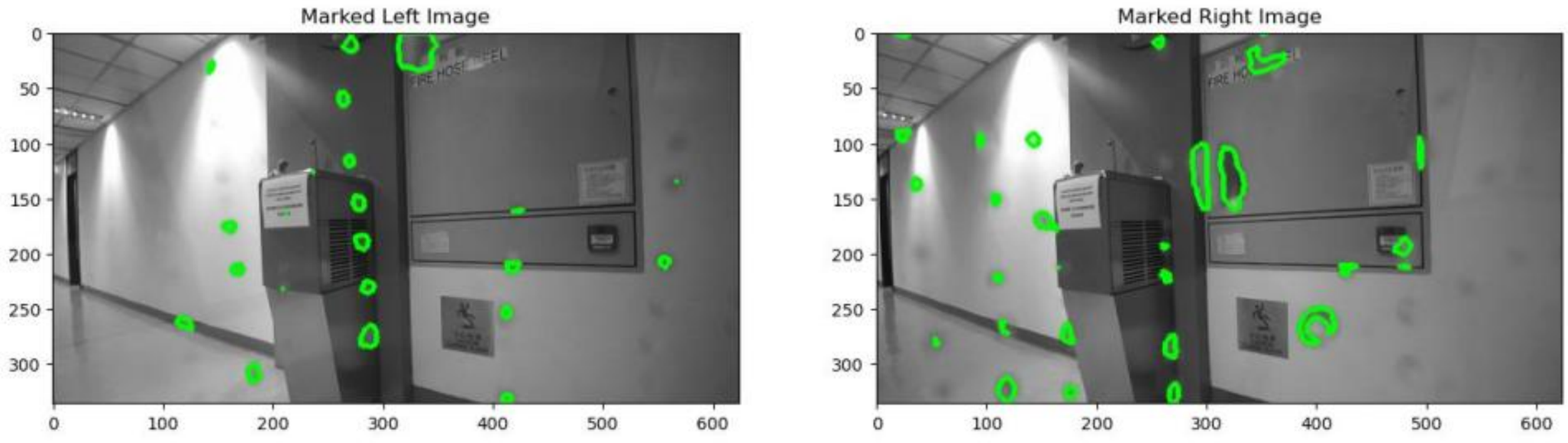

图 8 在原图上绘制水滴区域

预处理部分的核心代码如下：

```
diff_left = cv2.absdiff(noisy_left_image, background_left_image)
diff_right = cv2.absdiff(noisy_right_image, background_right_image)

_, binary_left = cv2.threshold(diff_left, 20, 255, cv2.THRESH_BINARY)
_, binary_right = cv2.threshold(diff_right, 30, 255, cv2.THRESH_BINARY)

kernel1 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (21,21))
kernel2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11,11))
binary_left = cv2.morphologyEx(binary_left, cv2.MORPH_CLOSE, kernel1)
binary_right = cv2.morphologyEx(binary_right, cv2.MORPH_CLOSE, kernel2)

# 查找轮廓
contours_left, _ = cv2.findContours(binary_left, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours_right, _ = cv2.findContours(binary_right, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# 在原图上绘制轮廓
marked_left_image = cv2.cvtColor(noisy_left_image, cv2.COLOR_GRAY2BGR)  # 转为彩色图像方便绘制
marked_right_image = cv2.cvtColor(noisy_right_image, cv2.COLOR_GRAY2BGR)

# 画出轮廓
cv2.drawContours(marked_left_image, contours_left, -1, (0, 255, 0), 4)  # 在左图上画轮廓，绿色线
cv2.drawContours(marked_right_image, contours_right, -1, (0, 255, 0), 4)  # 在右图上画轮廓，绿色线
```

For reference only
Chime is the copyright owner.

1.3.2 视差引导

我们在对偶图像上绘制出对应缺陷区域，如图 15 所示。可以发现对于左图重合的水滴噪声是较少的，而对于右图可能存在的水滴噪声较多，并且噪声区域也还有一部分由于差异太小无法检测出来。

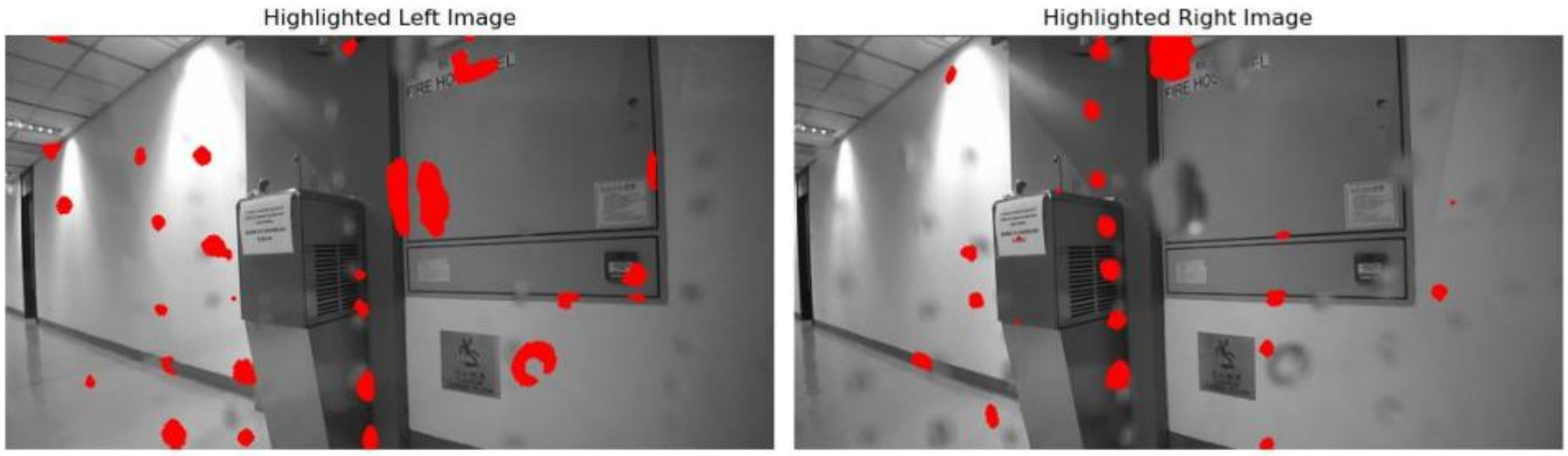

图 9 对偶缺陷区域

下面进行迭代视差引导修复算法的核心部分，即通过视差引导以最小化修复图像与 GT 图像的 MSE。如图 16 所示为处理后的结果。在差值图中，红色区域表示该像素的增强强度较高，可能是修复算法填补了噪声区域。而深蓝色区域表示修复过程中对该像素进行了较大的降噪或平滑，可能是噪声被消除后导致的细节丢失。因此红色区域越多，我们的修复效果越好。

图 10 迭代视差引导修复算法结果

视差引导部分的核心代码如下：

```
def inpaint_noise_with_disparity(left_image, right_image, left_disp, right_disp, binary_left, binary_right):
    """
    修复噪声图像，使用左图和右图的视差图，噪声位置通过噪声掩膜标记
    """
    # 创建恢复后的图像（首先拷贝原图）
    inpainted_left = left_image.copy()
    inpainted_right = right_image.copy()

    # 对左图中的噪声区域进行修复
    for u in range(binary_left.shape[0]):
        for v in range(binary_left.shape[1]):
            if binary_left[u, v] == 255:  # 如果是噪声区域
                # 获取左视差值
                left_disp_value = left_disp[u, v]
                # 使用右图的像素修复
```

For reference only
Chime is the copyright owner.

```python
                if v - int(left_disp_value) >= 0 and v - int(left_disp_value) < right_image.shape[1]:
                    inpainted_left[u, v] = right_image[u, v - int(left_disp_value)]

    # 对右图中的噪声区域进行修复
    for u in range(binary_right.shape[0]):
        for v in range(binary_right.shape[1]):
            if binary_right[u, v] == 255:  # 如果是噪声区域
                # 获取右视差值
                right_disp_value = right_disp[u, v]
                # 使用左图的像素修复
                if v + int(right_disp_value) >= 0 and v + int(right_disp_value) < left_image.shape[1]:
                    inpainted_right[u, v] = left_image[u, v + int(right_disp_value)]

    return inpainted_left, inpainted_right

def calculate_mse(imageA, imageB):
    err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
    err /= float(imageA.shape[0] * imageA.shape[1])
    return err

def iterative_inpaint(left_image, right_image, left_disp, right_disp, binary_left, binary_right, max_iters=5, threshold=1e-3):
    prev_left = left_image.copy()
    prev_right = right_image.copy()

    for iteration in range(max_iters):
        # 修复图像
        inpainted_left, inpainted_right = inpaint_noise_with_disparity(
            prev_left, prev_right, left_disp, right_disp, binary_left, binary_right
        )

        # 计算修复后图像的变化量
        delta_left = calculate_mse(inpainted_left , prev_left)
        delta_right = calculate_mse(inpainted_right , prev_right)
        print(f"Iteration {iteration + 1}: Delta Left = {delta_left}, Delta Right = {delta_right}")

        # 更新修复图像
        prev_left = inpainted_left.copy()
        prev_right = inpainted_right.copy()

        # 检查停止条件
        if delta_left < threshold and delta_right < threshold:
            print("Converged!")
            break

    return inpainted_left, inpainted_right
```

For reference only
Chime is the copyright owner.

```
inpainted_left_, inpainted_right_ = iterative_inpaint(noisy_left_image, noisy_right_image, left_disp, right_disp, binary_left, binary_right)
```

1.3.3 后处理

在对图像细节不是很重视的场合可以对上面的结果加上后处理以解决黑边缺陷，如图 17 所示为对修复后的图像进行了非局部均值与锐化操作，对比图 16 可以发现水滴区域的黑边也显著减少了。

图 11 非局部均值去噪+锐化结果

下面为核心代码：

```
# 使用非局部均值去噪
denoised_left = cv2.fastNlMeansDenoising(inpainted_left, None, 30, 7, 21)
denoised_right = cv2.fastNlMeansDenoising(inpainted_right, None, 20, 7, 21)

# 锐化图像（Unsharp Masking）
sharpened_left = cv2.addWeighted(denoised_left, 2, cv2.GaussianBlur(denoised_left, (0, 0), 3), -1, 0)
sharpened_right = cv2.addWeighted(denoised_right, 1.5, cv2.GaussianBlur(denoised_right, (0, 0), 3), -0.5, 0)
```

1.3.4 评价

由于我们拥有 GT 图像，因此可以使用 MSE、PSNR、SSIM 指数来对修复的图像质量做一个评价，表 1 所示为所得到的处理后与处理前结果的对比。

表 1 评价指标

Method	Left_image		Right_image	
	GT+Noise	GT+ Inpainted	GT+Noise	GT+ Inpainted
MSE	17.2008	11.2207	61.2678	25.6778
PSNR	35.7753	37.6306	30.2585	34.0352
SSIM	0.9508	0.9527	0.9236	0.9328

MSE 是用来衡量两幅图像之间的像素差异，数值越低说明两幅图像越接近。无论是左右图中，可以发现噪声映入了明显的像素差异，但是修复后的图像，尤其是右图像的 MSE 显著减少，说明在像素误差方面，本算法是修复较好的。

PSNR(峰值信噪比)：测量重建图像质量的指标，数值越高说明质量越好。虽然噪声降低了图像的质量，但是左右两张图的 PSNR 仍然保持在一个合理的范围内，但是通过图像修复后的图像质量会更好。

SSIM（结构相似性)：衡量两幅图像在结构信息上的相似性，数值在 [0,1] 之间，越接近

For reference only
Chime's the copyright owner.

1 表示图像越相似。两幅修复图像的结构相似性略微提升，说明了修复不仅降低了噪声，还较好的保留了图像的结构。

1.3.5 鲁棒性测试

下面我将对测试集的数据进行同样的算法处理，如图 18 所示为经过迭代视差引导修复算法得到的结果。

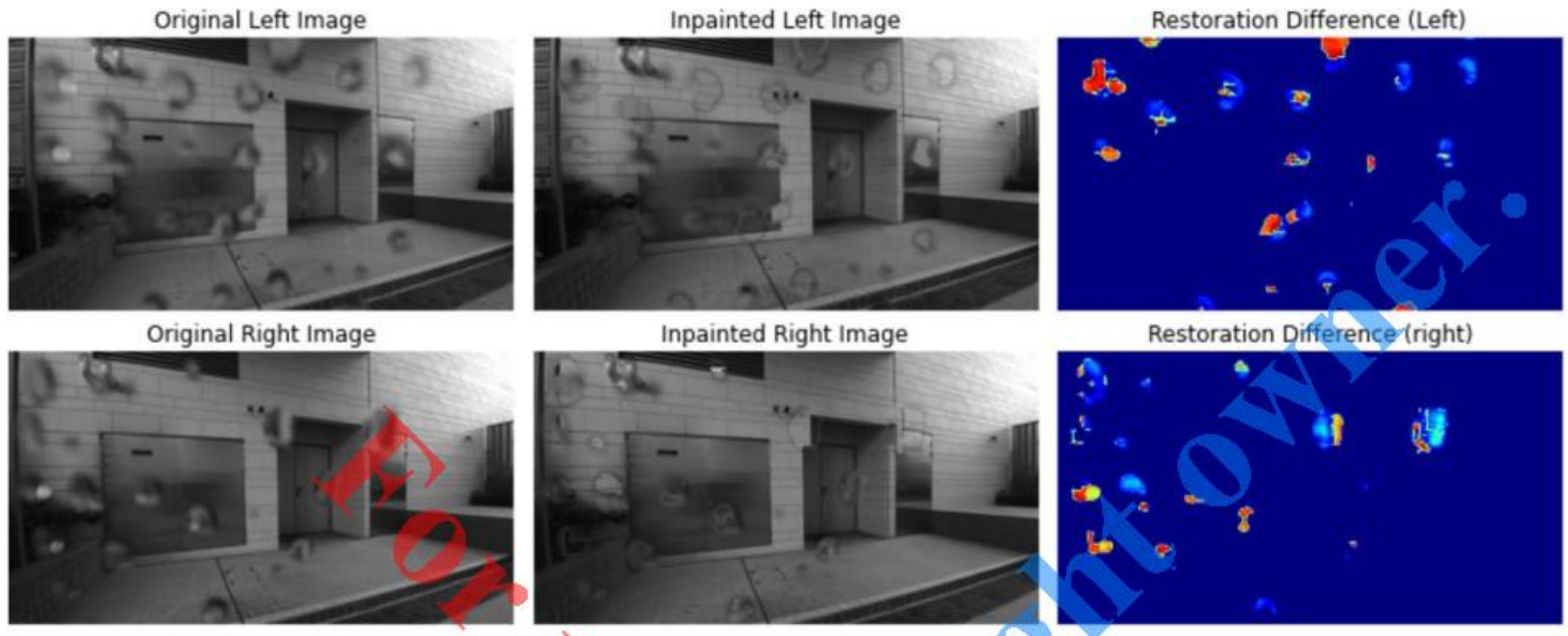

图 1 测试集数据集算法处理结果

处理得到的指标见表 2，可以发现算法对右相机较好。

表 2 测试集评价

Method	Left_image		Right_image	
	GT+Noise	GT+ Inpainted	GT+Noise	GT+ Inpainted
MSE	111.0927	83.0349	125.9688	76.4631
PSNR	27.6739	28.9382	27.1282	29.2963
SSIM	0.9126	0.9102	0.9251	0.9255

如图 19 所示为非局部均值去噪+锐化的结果。

图 2 非局部均值去噪+锐化结果

1.4 参考

[1] Z. Shi, N. Fan, D. -Y. Yeung and Q. Chen, "Stereo Waterdrop Removal with Row-wise Dilated Attention," 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 2021.

[2] A. Yamashita, Y. Tanaka and T. Kaneko, "Removal of adherent waterdrops from images acquired with stereo camera," 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, AB, Canada, 2005.

For reference only
Chime is the copyright owner.

[3] Fan, Linwei & Zhang, Fan & Fan, Hui & Zhang, Caiming. (2019). Brief review of image denoising techniques. Visual Computing for Industry, Biomedicine, and Art.
[4] https://aijishu.com/a/1060000000139727

二、附录

项目完整代码如下：

```
import zipfile
import os

# 恶性良性皮肤癌数据
files = ["/data/shixunfiles/92ee0df12de1a2559e344bc23fd77464_1731682879492.zip",
         "/data/shixunfiles/bf67a04d6abf6220610d9b1e14698d16_1731682656948.zip"]

# 创建目标文件夹
os.makedirs('/data/workspace/downloadfiles/data2/train', exist_ok=True)
os.makedirs('/data/workspace/downloadfiles/data2/test', exist_ok=True)

# 解压文件到目标文件夹
with zipfile.ZipFile(files[0], 'r') as zf:
    zf.extractall('/data/workspace/downloadfiles/data2/train')
with zipfile.ZipFile(files[1], 'r') as zf:
    zf.extractall('/data/workspace/downloadfiles/data2/test')

print("Files extracted successfully!")

import os
import cv2
import numpy as np

def load_data_by_scene(data_dir):
    # 初始化存储每个场景数据的列表
    scenes_left_inputs = []
    scenes_right_inputs = []
    scenes_left_targets = []
    scenes_right_targets = []

    image_dir = data_dir

    # 遍历每个场景文件夹（如 000, 001, ...）
    for folder in sorted(os.listdir(image_dir)):
        folder_path = os.path.join(image_dir, folder)
```

For reference only
Chime is the copyright owner.

```
        images = sorted(os.listdir(folder_path))
        # 标志变量，确保只读取一张噪声图像
        left_noise_read = False
        right_noise_read = False

        # 当前场景的输入数据
        left_inputs = []
        right_inputs = []
        left_target = None
        right_target = None

        # 遍历场景文件夹中的图片
        for img in images:
            img_path = os.path.join(folder_path, img)

            # 区分左右相机的真实图像和噪声图像
            if img == '000_0.png':  # 左相机真实图像
                left_target = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            elif img == '000_1.png':  # 右相机真实图像
                right_target = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            elif img.endswith('_0.png'):  # 左相机噪声图像
                left_inputs.append(cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
)
                left_noise_read = True  # 标记为已读取
            elif img.endswith('_1.png'):  # 右相机噪声图像
                right_inputs.append(cv2.imread(img_path, cv2.IMREAD_GRAYSCALE
))
                right_noise_read = True  # 标记为已读取

            # 如果左右噪声图像都已读取，提前结束循环
            if left_noise_read and right_noise_read:
                break

        # 将当前场景数据添加到场景列表中
        scenes_left_inputs.append(left_inputs)
        scenes_right_inputs.append(right_inputs)
        scenes_left_targets.append(left_target)
        scenes_right_targets.append(right_target)

    return scenes_left_inputs, scenes_right_inputs, scenes_left_targets, scen
es_right_targets

# 调用函数加载数据
data_path = r"Stereoscopic drop removal (Image denoising)\project\data"
train_left_inputs, train_right_inputs, train_left_targets, train_right_target
s = load_data_by_scene(os.path.join(data_path, "train\image"))
test_left_inputs, test_right_inputs, test_left_targets, test_right_targets =
load_data_by_scene(os.path.join(data_path,"test"))
print('ok')

def load_disparity_and_mask(data_dir):

    """加载视差图和掩膜图"""
```

For reference only
Chime is the copyright owner.

```python
    disparities = []
    disparities_r = []
    masks = []
    masks_r = []

    image_dir = os.path.join(data_dir, 'image')
    mask_dir = os.path.join(data_dir, 'mask')
    mask_r_dir = os.path.join(data_dir, 'mask_r')
    disparity_dir = os.path.join(data_dir, 'disparity')
    disparity_r_dir = os.path.join(data_dir, 'disparity_r')

    for folder in sorted(os.listdir(image_dir)):
        # 加载掩膜和视差图
        mask = cv2.imread(os.path.join(mask_dir, f"{folder}.png"), cv2.IMREAD_GRAYSCALE)
        mask_r = cv2.imread(os.path.join(mask_r_dir, f"{folder}.png"), cv2.IMREAD_GRAYSCALE)
        disparity = cv2.imread(os.path.join(disparity_dir, f"{folder}.png"), cv2.IMREAD_GRAYSCALE)
        disparity_r = cv2.imread(os.path.join(disparity_r_dir, f"{folder}.png"), cv2.IMREAD_GRAYSCALE)

        masks.append(mask)
        masks_r.append(mask_r)
        disparities.append(disparity)
        disparities_r.append(disparity_r)

    return disparities, disparities_r, masks, masks_r

data_path = r"Stereoscopic drop removal (Image denoising)\project\data\train"
disparities, disparities_r, masks, masks_r = load_disparity_and_mask(data_path)

# 检查数据
print(train_left_inputs[10][0].shape)
print(len(train_left_inputs))
print(train_right_inputs[0][0].shape)
print(train_left_targets[0].shape)
print(len(train_left_targets))
print(train_right_targets[0].shape)
print(disparities[0].shape)
print(len(disparities))
print(disparities_r[0].shape)
# 展平嵌套列表并匹配目标
def flatten_data(inputs, targets):
    flat_inputs = []
    flat_targets = []
    for scene_inputs, target in zip(inputs, targets):
        for img in scene_inputs:
            flat_inputs.append(img)  # 展平嵌套输入
            flat_targets.append(target)  # 将目标匹配到每张输入图像
    return flat_inputs, flat_targets
```

For reference only
Chime is the copyright owner.

```python

# 处理训练数据
flat_train_left_inputs, flat_train_left_targets = flatten_data(train_left_inputs, train_left_targets)
flat_train_right_inputs, flat_train_right_targets = flatten_data(train_right_inputs, train_right_targets)
flat_test_left_inputs, flat_test_left_targets = flatten_data(test_left_inputs, test_left_targets)
flat_test_right_inputs, flat_test_right_targets = flatten_data(test_right_inputs, test_right_targets)

# 差值法找出轮廓
background_left_image = flat_train_left_targets[0]
noisy_left_image = flat_train_left_inputs[0]
background_right_image = flat_train_right_targets[0]
noisy_right_image = flat_train_right_inputs[0]

diff_left = cv2.absdiff(noisy_left_image, background_left_image)
diff_right = cv2.absdiff(noisy_right_image, background_right_image)

plt.figure(figsize=(16, 12))
plt.subplot(1,2,1)
plt.imshow(diff_left,cmap='gray')
plt.title("diff_left")

plt.subplot(1,2,2)
plt.imshow(diff_right,cmap='gray')
plt.title("diff_right")
plt.show()

_, binary_left = cv2.threshold(diff_left, 20, 255, cv2.THRESH_BINARY)
_, binary_right = cv2.threshold(diff_right, 30, 255, cv2.THRESH_BINARY)

kernel1 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (21,21))
kernel2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11,11))
binary_left = cv2.morphologyEx(binary_left, cv2.MORPH_CLOSE, kernel1)
binary_right = cv2.morphologyEx(binary_right, cv2.MORPH_CLOSE, kernel2)

plt.figure(figsize=(16, 12))
plt.subplot(1,2,1)
plt.imshow(binary_left,cmap='gray')
plt.title("binary_left")

plt.subplot(1,2,2)
plt.imshow(binary_right,cmap='gray')
plt.title("binary_right")
plt.show()

# 查找轮廓
contours_left, _ = cv2.findContours(binary_left, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours_right, _ = cv2.findContours(binary_right, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

For reference only.
Chime is the copyright owner.

```python

# 在原图上绘制轮廓
marked_left_image = cv2.cvtColor(noisy_left_image, cv2.COLOR_GRAY2BGR)  # 转为彩色图像方便绘制
marked_right_image = cv2.cvtColor(noisy_right_image, cv2.COLOR_GRAY2BGR)

# 画出轮廓
cv2.drawContours(marked_left_image, contours_left, -1, (0, 255, 0), 4)  # 在左图上画轮廓，绿色线
cv2.drawContours(marked_right_image, contours_right, -1, (0, 255, 0), 4)  # 在右图上画轮廓，绿色线

#  显示结果
plt.figure(figsize=(16, 12))

# 左图结果
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(marked_left_image, cv2.COLOR_BGR2RGB))  # 转为RGB显示
plt.title("Marked Left Image")

# 右图结果
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(marked_right_image, cv2.COLOR_BGR2RGB))  # 转为RGB显示
plt.title("Marked Right Image")

plt.show()

# 迭代视差引导修复
left_disp = disparities[0]
right_disp = disparities_r[0]

# 标记对应区域的方法
def highlight_corresponding_regions(left_image, right_image, left_disp, right_disp, binary_left, binary_right):
    # 创建用于标记的图像副本
    highlighted_left = left_image.copy()
    highlighted_right = right_image.copy()

    # 转换为彩色图像以便标记
    highlighted_left = cv2.cvtColor(highlighted_left, cv2.COLOR_GRAY2BGR)
    highlighted_right = cv2.cvtColor(highlighted_right, cv2.COLOR_GRAY2BGR)

    # 在左图中标记右图对应的修复区域
    for u in range(binary_left.shape[0]):
        for v in range(binary_left.shape[1]):
            if binary_left[u, v] == 255:  # 如果是噪声区域
                left_disp_value = left_disp[u, v]
                # 计算对应的右图位置
                right_v = v - int(left_disp_value)
                if 0 <= right_v < right_image.shape[1]:
```

For reference only
Chime is the copyright owner.

```
                    cv2.circle(highlighted_right, (right_v, u), radius=2, color=(0, 0, 255), thickness=-1)

    # 在右图中标记左图对应的修复区域
    for u in range(binary_right.shape[0]):
        for v in range(binary_right.shape[1]):
            if binary_right[u, v] == 255:  # 如果是噪声区域
                right_disp_value = right_disp[u, v]
                # 计算对应的左图位置
                left_v = v + int(right_disp_value)
                if 0 <= left_v < left_image.shape[1]:
                    cv2.circle(highlighted_left, (left_v, u), radius=2, color=(0, 0, 255), thickness=-1)

    return highlighted_left, highlighted_right

# 调用函数
highlighted_left, highlighted_right = highlight_corresponding_regions(
    noisy_left_image, noisy_right_image, left_disp, right_disp, binary_left, binary_right
)

# 可视化标记结果
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

axes[0].imshow(cv2.cvtColor(highlighted_left, cv2.COLOR_BGR2RGB))
axes[0].set_title('Highlighted Left Image')
axes[0].axis('off')

axes[1].imshow(cv2.cvtColor(highlighted_right, cv2.COLOR_BGR2RGB))
axes[1].set_title('Highlighted Right Image')
axes[1].axis('off')

plt.tight_layout()
plt.show()

# 首先进行单次引导
def inpaint_noise_with_disparity(left_image, right_image, left_disp, right_disp, binary_left, binary_right):

    inpainted_left = left_image.copy()
    inpainted_right = right_image.copy()

    # 对左图中的噪声区域进行修复
    for u in range(binary_left.shape[0]):
        for v in range(binary_left.shape[1]):
            if binary_left[u, v] == 255:  # 如果是噪声区域
                # 获取左视差值
                left_disp_value = left_disp[u, v]
                # 使用右图的像素修复
                if v - int(left_disp_value) >= 0 and v - int(left_disp_value) < right_image.shape[1]:
```

For reference only
Chime is the copyright owner.

```
                    inpainted_left[u, v] = right_image[u, v - int(left_disp_value)]

    # 对右图中的噪声区域进行修复
    for u in range(binary_right.shape[0]):
        for v in range(binary_right.shape[1]):
            if binary_right[u, v] == 255:  # 如果是噪声区域
                # 获取右视差值
                right_disp_value = right_disp[u, v]
                # 使用左图的像素修复
                if v + int(right_disp_value) >= 0 and v + int(right_disp_value) < left_image.shape[1]:
                    inpainted_right[u, v] = left_image[u, v + int(right_disp_value)]

    return inpainted_left, inpainted_right

# 修复图像
inpainted_left, inpainted_right = inpaint_noise_with_disparity(noisy_left_image, noisy_right_image, left_disp, right_disp, binary_left, binary_right)

plt.tight_layout()
plt.show()

def calculate_mse(imageA, imageB):
    err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
    err /= float(imageA.shape[0] * imageA.shape[1])
    return err

def iterative_inpaint(left_image, right_image, left_disp, right_disp, binary_left, binary_right, max_iters=5, threshold=1e-3):

    prev_left = left_image.copy()
    prev_right = right_image.copy()

    for iteration in range(max_iters):
        # 修复图像
        inpainted_left, inpainted_right = inpaint_noise_with_disparity(
            prev_left, prev_right, left_disp, right_disp, binary_left, binary_right
        )

        # 计算修复后图像的变化量
        delta_left = calculate_mse(inpainted_left , prev_left)
        delta_right = calculate_mse(inpainted_right , prev_right)
        print(f"Iteration {iteration + 1}: Delta Left = {delta_left}, Delta Right = {delta_right}")

        # 更新修复图像
        prev_left = inpainted_left.copy()
        prev_right = inpainted_right.copy()

        # 检查停止条件
```

For reference only
Chime is the copyright owner.

```
        if delta_left < threshold and delta_right < threshold:
            print("Converged!")
            break

    return inpainted_left, inpainted_right

inpainted_left_, inpainted_right_ = iterative_inpaint(noisy_left_image, noisy_right_image, left_disp, right_disp, binary_left, binary_right)

# 使用非局部均值去噪
denoised_left = cv2.fastNlMeansDenoising(inpainted_left, None, 10, 7, 21)
denoised_right = cv2.fastNlMeansDenoising(inpainted_right, None, 20, 7, 21)

# 锐化图像（Unsharp Masking）
sharpened_left = cv2.addWeighted(denoised_left, 2, cv2.GaussianBlur(denoised_left, (0, 0), 3), -1, 0)
sharpened_right = cv2.addWeighted(denoised_right, 1.5, cv2.GaussianBlur(denoised_right, (0, 0), 3), -0.5, 0)

# 可视化去噪后的图像
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

axes[0].imshow(sharpened_left, cmap='gray')
axes[0].set_title('Denoised Left Image')
axes[0].axis('off')

axes[1].imshow(sharpened_right, cmap='gray')
axes[1].set_title('Denoised Right Image')
axes[1].axis('off')

plt.tight_layout()
plt.show()

# 读取原始 RGB 图像（假设为 original_rgb）
original_rgb = cv2.imread(r"000_0.png")

noisy_left_image_3ch = cv2.merge([noisy_left_image,noisy_left_image,noisy_left_image])
denoised_gray_3ch_1 = cv2.merge([sharpened_left, sharpened_left, sharpened_left])

# # 保留原始 RGB 的色彩信息，将去噪后的灰度图像与原图合并
restored_rgb_noise = cv2.addWeighted(original_rgb, 0.5, noisy_left_image_3ch, 0.5, 0)
restored_rgb = cv2.addWeighted(original_rgb, 0.5, denoised_gray_3ch_1, 0.5, 0)

fig, axes = plt.subplots(1, 2, figsize=(10, 4))

axes[0].imshow(restored_rgb_noise, cmap='gray')
axes[0].set_title('Denoised Left Image')
axes[0].axis('off')
```

For reference only
Chime is the copyright owner.

```
axes[1].imshow(restored_rgb, cmap='gray')
axes[1].set_title('Denoised Right Image')
axes[1].axis('off')

plt.tight_layout()
plt.show()

from skimage.metrics import structural_similarity as ssim

def calculate_psnr(imageA, imageB):
    mse = calculate_mse(imageA, imageB)
    if mse == 0:
        return float('inf')
    PIXEL_MAX = 255.0
    return 20 * np.log10(PIXEL_MAX / np.sqrt(mse))

def calculate_ssim(imageA, imageB):
    score, _ = ssim(imageA, imageB, full=True)
    return score

def evaluate_images(background_image, noisy_image, inpainted_image):
    # 计算 MSE
    mse_value_ori = calculate_mse(background_image, noisy_image)
    mse_value = calculate_mse(background_image, inpainted_image)

    # 计算 PSNR
    psnr_value_ori = calculate_psnr(background_image, noisy_image)
    psnr_value = calculate_psnr(background_image, inpainted_image)

    # 计算 SSIM
    ssim_value_ori = calculate_ssim(background_image, noisy_image)
    ssim_value = calculate_ssim(background_image, inpainted_image)

    # 打印结果
    print(f"Ori MSE: {mse_value_ori:.4f}, inpainted MSE: {mse_value:.4f}")
    print(f"Ori PSNR: {psnr_value_ori:.4f} dB, inpainted PSNR: {psnr_value:.4f}")
    print(f"Ori SSIM: {np.mean(ssim_value_ori):.4f}, inpainted SSIM: {np.mean(ssim_value):.4f}")

evaluate_images(background_left_image, noisy_left_image, inpainted_left_)
evaluate_images(background_right_image, noisy_right_image, inpainted_right_)
```

For reference only
Chime is the copyright owner.