

# 使用 SIFT 算法进行图像描述任务

## 一、SIFT 算法简介

SIFT (Scale-invariant feature transform) 算法是一种在计算机视觉领域中广泛应用的特征提取和匹配算法。它由 David Lowe 在 1999 年提出，并在 2004 年的论文中详细描述。

SIFT 算法的主要步骤包括尺度空间极值检测、关键点定位、方向确定、特征描述和特征匹配。其中，尺度空间极值检测使用高斯差分金字塔实现，用于检测图像中的关键点。关键点定位通过对尺度空间极值点进行筛选，剔除不稳定和低对比度的点。方向确定阶段为每个关键点分配一个主方向，用于后续特征描述。特征描述通过构建局部特征直方图描述关键点周围的图像局部结构。最后，特征匹配阶段用于寻找不同图像中具有相似特征的关键点。

SIFT 算法在图像匹配、目标识别、三维重建等领域广泛应用，并且具有很好的鲁棒性和不变性。接下来将使用 SIFT 算法完成一些图像描述任务，并对**实验结果、算法性能以及算法的适用性**进行分析。

## 二、SIFT 图像匹配及其拓展

### 2.1 使用 SIFT 进行图像匹配的算法思路

SIFT 算法本身就包括尺度空间极值检测、关键点定位、方向确定、特征描述和特征匹配。因此，可以直接使用 SIFT 算法实现图像匹配。

算法思路如下：

- # 创建 SIFT 对象
- # 检测特征点和计算特征描述符
- # 创建 FLANN 匹配器
- # 使用 knnMatch 进行特征点匹配
- # 对特征点进行筛选
- # 可视化匹配结果

### 2.2 使用 SIFT 进行图像匹配的 python 代码

SIFTmatch.py:

```
import cv2

def sift_matching(img1, img2):
    # 创建 SIFT 对象
    sift = cv2.xfeatures2d.SIFT_create()

    # 检测特征点和计算特征描述符
    keypoints1, descriptors1 = sift.detectAndCompute(img1, None)
    keypoints2, descriptors2 = sift.detectAndCompute(img2, None)

    # 创建 FLANN 匹配器
    flann = cv2.FlannBasedMatcher()

    # 使用 knnMatch 进行特征点匹配
    matches = flann.knnMatch(descriptors1, descriptors2, k=2)

    # 进行 Lowe's ratio test 进行筛选
    good_matches = []
    for m, n in matches:
        if m.distance < 0.7 * n.distance:
            good_matches.append(m)

    # 可视化匹配结果
    img_matches = cv2.drawMatches(img1, keypoints1, img2, keypoints2, good_matches, None)
```

```

return img_matches

# 读取图像
img1 = cv2.imread('image1.jpg')
img2 = cv2.imread('image2.jpg')

# 进行图像匹配
result = sift_matching(img1, img2)

# 显示匹配结果
cv2.namedWindow('SIFT Matching Result', cv2.WINDOW_NORMAL)
cv2.imshow('SIFT Matching Result', result)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

## 2.3 设计实验分析 SIFT 图像匹配的性能

### 2.3.1 实验设计思路

根据 SIFT 的原理，考虑到可能影响匹配效果的因素：

①图像质量：图像质量可能是影响 SIFT 匹配效果的关键因素之一。如果图像存在噪声、模糊、亮度不均匀等问题，可能会导致关键点提取和描述子计算过程中的误差，从而影响匹配效果。

②视角变化：SIFT 算法对于旋转和视角变化具有一定的鲁棒性，但如果图像存在较大的视角变化，可能会导致关键点提取和描述子计算过程中的错误，进而影响匹配结果。

③遮挡和遮蔽：如果图像中存在遮挡或遮蔽物，这些物体可能会对关键点的提取和匹配造成困扰，使得匹配效果下降。

④物体的纹理和结构：物体的纹理丰富度和结构复杂性对 SIFT 匹配效果也有影响。纹理不够丰富或结构不够明显的物体可能会导致关键点提取和匹配过程中的错误。

下面针对这四个因素设计实验，分析每种因素对匹配效果的影响，进一步分析 SIFT 算法的性能。

### 2.3.2 图像质量对匹配效果的影响

将 3 张拍摄内容和视角大致相同、图像质量不同的图片作为 SIFTmatch.py 的输入图片，分析图像质量对匹配效果的影响。

3 张图片名称为：quality1.jpg、quality2.jpg、quality3.jpg，其中 quality3.jpg 较模糊，3 张图片如图 1 所示：



图 1：依次为 quality1.jpg、quality2.jpg、quality3.jpg

将 quality1.jpg 和 quality2.jpg 作为输入图片，获得第一个运行结果，再将 quality1.jpg 和 quality3.jpg 作为输入图片，获得第二个运行结果，对两次运行结果进行对比分析。

两次运行结果如图 2、图 3 所示：

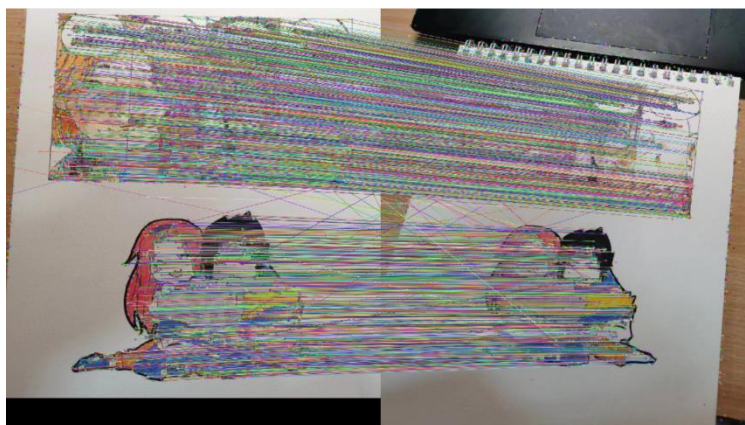


图 2: quality1.jpg 和 quality2.jpg 匹配结果

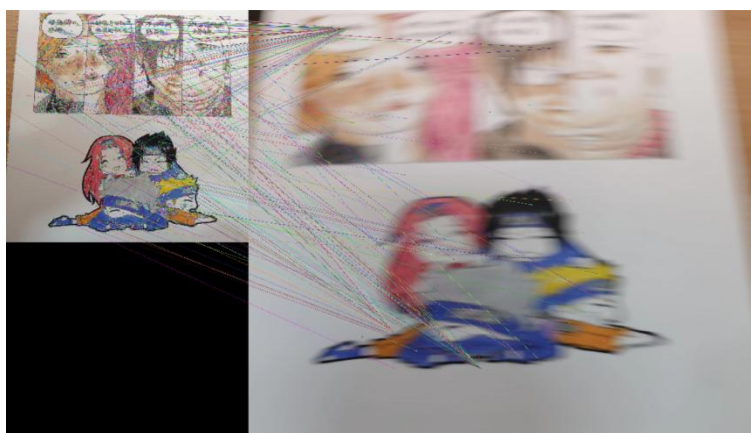


图 3: quality1.jpg 和 quality3.jpg 匹配结果

可以看出, quality1.jpg 和 quality2.jpg 匹配特征点数远大于 quality1.jpg 和 quality3.jpg 匹配特征点数, 前者匹配效果远好于后者。这说明图像质量是影响 SIFT 匹配效果的关键因素之一, 噪声和模糊会造成关键点提取和描述子计算过程中的误差, 从而影响匹配效果, 甚至无法实现匹配。

### 2.3.3 视角变化对匹配效果的影响

将 6 张拍摄内容和图像质量大致相同、视角不同的图片作为 SIFTmatch.py 的输入图片, 分析视角变化对匹配效果的影响。

6 张图片名称为: angle1.jpg、angle2.jpg、angle3.jpg、angle4.jpg、angle5.jpg、angle6.jpg, 其中 angle1.jpg、angle2.jpg、angle3.jpg、angle4.jpg 是在保持物体正面角度的情况下旋转视角, 而 angle5.jpg、angle6.jpg 则将视角旋转到物体的侧面, 6 张图片如图 4 所示:







图 4: 依次为 angle1.jpg、……、angle6.jpg

将 angle1.jpg 分别和 angle2.jpg、angle3.jpg、angle4.jpg、angle5.jpg、angle6.jpg 作为输入图片，获得五个运行结果，对五次运行结果进行对比分析。五次运行结果如图 5 到图 9 所示：

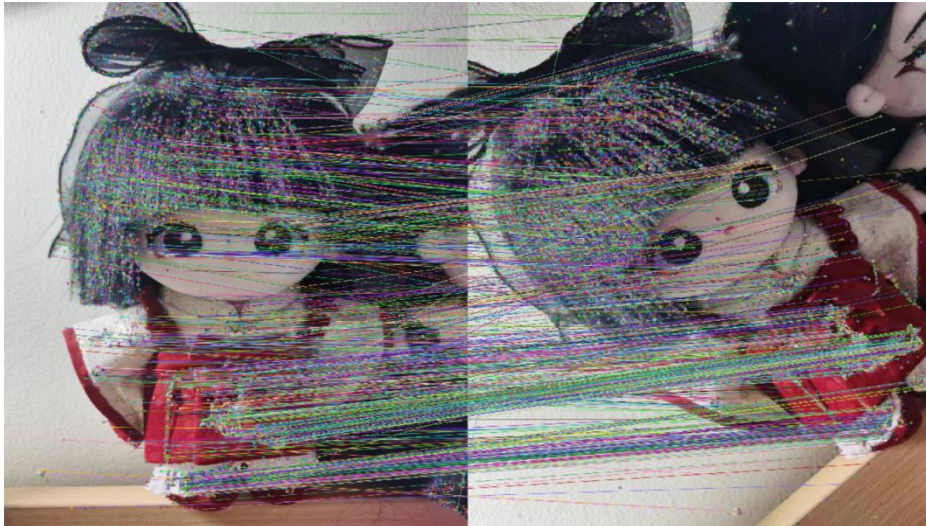


图 5: angle1.jpg 和 angle2.jpg 匹配结果

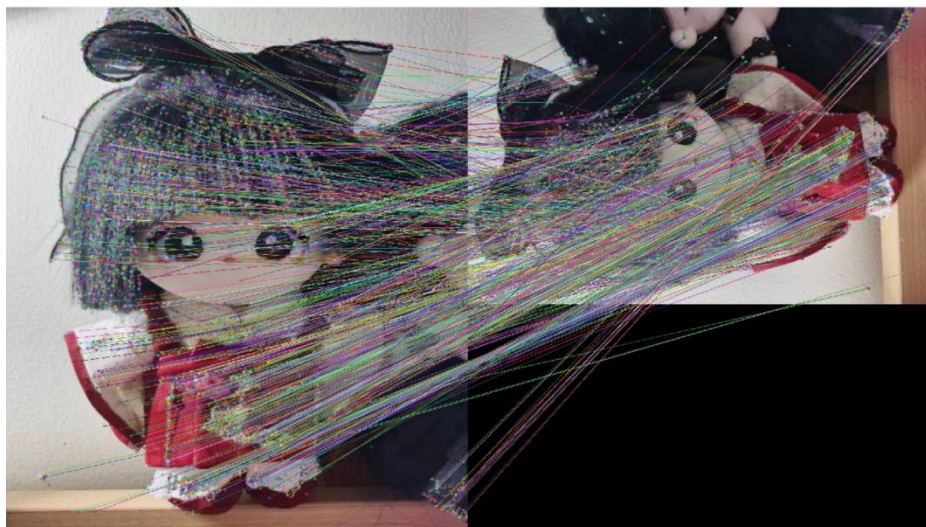


图 6: angle1.jpg 和 angle3.jpg 匹配结果



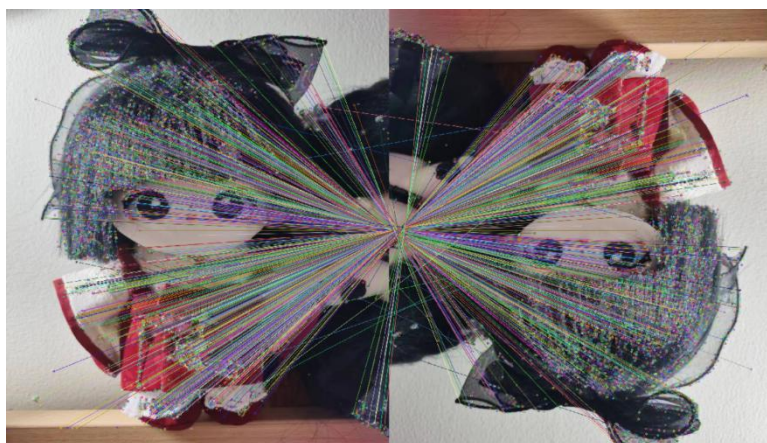


图 7: angle1.jpg 和 angle4.jpg 匹配结果



图 8: angle1.jpg 和 angle5.jpg 匹配结果



图 9: angle1.jpg 和 angle6.jpg 匹配结果

可以看出，angle1.jpg 和 angle2.jpg、angle3.jpg、angle4.jpg 匹配效果都很好，angle1.jpg 和 angle5.jpg 匹配效果次之，angle1.jpg 和 angle6.jpg 匹配结果最差。

这说明视角变化是影响 SIFT 匹配效果的关键因素之一，如果保持物体角度无大变化的情况下旋转视角，匹配效果不会变差，但若图像存在较大的视角变化，比如从正视图变为左视图，会影响匹配结果。

#### 2.3.4 遮挡和遮蔽对匹配效果的影响

将 3 张拍摄内容、图像质量和视角大致相同，遮挡程度不同的图片作为 SIFTmatch.py 的输入图片，分析遮挡和遮蔽对匹配效果的影响。

3 张图片名称为：hide1.jpg、hide2.jpg、hide3.jpg，遮挡程度依次增加，3 张图片如图 10 所示：



图 10：依次为 hide1.jpg、hide2.jpg、hide3.jpg

将 hide1.jpg 和 hide2.jpg 作为输入图片，获得第一个运行结果，再将 hide1.jpg 和 hide3.jpg 作为输入图片，获得第二个运行结果，对两次运行结果进行对比分析。

两次运行结果如图 11、图 12 所示：



图 11：hide1.jpg 和 hide2.jpg 匹配结果



图 12：hide1.jpg 和 hide3.jpg 匹配结果



可以看出, hide1.jpg 和 hide2.jpg 匹配特征点数远大于 hide1.jpg 和 hide3.jpg 匹配特征点数, 前者匹配效果远好于后者。这说明遮挡和遮蔽是影响 SIFT 匹配效果的关键因素之一, 如果图像中存在遮挡或遮蔽物, 这些物体可能会对关键点的提取和匹配造成困扰, 使得匹配效果下降。

### 2.3.5 物体的纹理和结构对匹配效果的影响

将 4 张纹理结构不同的图片作为 SIFTmatch.py 的输入图片, 分析物体的纹理和结构对匹配效果的影响。

4 张图片名称为: texture1.jpg、texture2.jpg、texture3.jpg、texture4.jpg, 前两张是纹理很少的桌面, 后两张是桌面上放置一个红色发卡, 4 张图片如图 13 所示:



图 13: 依次为 texture1.jpg、texture2.jpg、texture3.jpg、texture4.jpg

将 texture1.jpg 和 texture2.jpg 作为输入图片, 获得第一个运行结果, 再将 texture3.jpg 和 texture4.jpg 作为输入图片, 获得第二个运行结果, 对两次运行结果进行对比分析。

两次运行结果如图 14、图 15 所示:



图 14: texture1.jpg 和 texture2.jpg 匹配结果

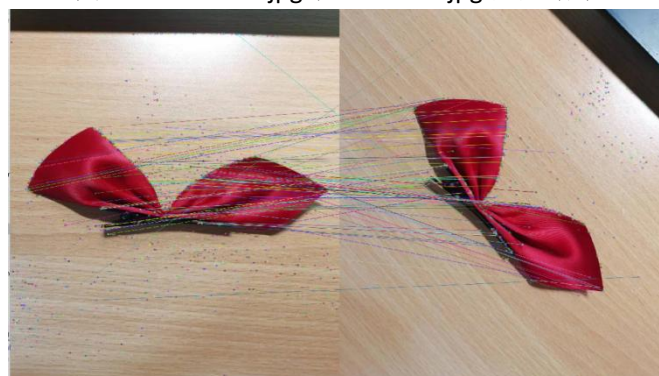


图 15: texture3.jpg 和 texture4.jpg 匹配结果

可以看出，texture3.jpg 和 texture4.jpg 匹配特征点数大于 texture1.jpg 和 texture2.jpg 匹配特征点数，后者匹配效果好于前者。这说明物体的纹理丰富度和结构复杂性对 SIFT 匹配效果也有影响，纹理不够丰富或结构不够明显的物体可能会导致关键点提取和匹配过程中的错误，使得匹配效果下降。

## 2.4 总结 SIFT 图像匹配算法的适用性

综合前面的分析，可以发现简单的 SIFT 匹配算法不适用于图像较模糊、视角变换较大、遮挡较大以及纹理过于简单的情况，这些问题有时可以通过将 SIFT 算法与其他方法结合的方式解决。

## 2.5 拓展一：立体匹配

根据 2.3.3 的分析结果推广，SIFT 算法本身并不能直接应用于立体匹配。SIFT 算法主要用于图像特征点的提取和匹配，以便进行图像间的配准、物体识别等任务。在立体匹配中，我们需要找到左右图像之间对应的像素点，而不是图像特征点之间的匹配。

立体匹配通常涉及计算视差（Disparity）或深度估计，目的是找到左右图像中对应像素点之间的水平偏移量。SIFT 算法无法直接提供视差信息，因为它是基于局部特征点描述符的。

但是，在立体匹配中提取特征点的步骤，依然可以使用 SIFT 算法。

一般来说，立体匹配的步骤如下：

- ①对左右两幅图像进行预处理，包括去噪、矫正、校正等操作。
- ②提取左右两幅图像中的特征点。可以使用 SIFT 算法、SURF 算法、ORB 算法等。
- ③对左右两幅图像中的特征点进行匹配，计算对应点的视差。常用的匹配算法有基于区域的匹配、基于特征描述子的匹配等。
- ④对视差图进行后处理，包括填充、滤波、去噪等，以得到更加平滑和准确的视差图。
- ⑤可视化视差图，将其显示出来并调整参数，以得到最佳的视差效果。

其中，视差算法的选择和参数设置对最终的视差图效果有很大的影响。在实际应用中，需要根据具体的场景和需求选择合适的算法和参数。

## 2.6 拓展二：目标识别

在 SIFT 图像匹配算法的基础上，可以稍作修改实现目标识别算法，但和 SIFT 图像匹配算法的适用性一样，在此基础上改造的目标识别算法不适用于图像较模糊、视角变换较大、遮挡较大以及纹理过于简单的情况，在目标变化多种多样的情况下准确度不佳。

使用 SIFT 进行目标识别的 python 代码如下：

SIFTrecognition.py:

```
import cv2

def sift_matching(query_img, target_img):
    # 创建 SIFT 对象
    sift = cv2.xfeatures2d.SIFT_create()

    # 检测特征点和计算特征描述符
    keypoints_query, descriptors_query = sift.detectAndCompute(query_img, None)
    keypoints_target, descriptors_target = sift.detectAndCompute(target_img, None)

    # 创建 FLANN 匹配器
    flann = cv2.FlannBasedMatcher()

    # 使用 knnMatch 进行特征点匹配
    matches = flann.knnMatch(descriptors_query, descriptors_target, k=2)

    # 进行 Lowe's ratio test 进行筛选
    good_matches = []
    for m, n in matches:
        if m.distance < 0.7 * n.distance:
```



```

        good_matches.append(m)

# 如果匹配对数超过一定阈值，则认为目标被识别
if len(good_matches) > match_threshold:
    return True
else:
    return False

# 读取查询图像和目标图像
query_img = cv2.imread('image3.jpg', 0) # 灰度图像
target_img = cv2.imread('image4.jpg', 0) # 灰度图像

# 设置匹配阈值
match_threshold = 10

# 使用 SIFT 进行目标识别
is_match = sift_matching(query_img, target_img)

# 打印结果
if is_match:
    print("目标被识别！")
else:
    print("目标未被识别！")

```

### 三、SIFT 目标跟踪

#### 3.1 使用 SIFT 进行目标跟踪的算法思路

- # 使用 SIFT 算法提取初始帧中目标对象的特征点和特征描述符。
- # 在下一帧图像中，使用 SIFT 算法检测并提取特征点和描述符。
- # 使用特征匹配算法（如 FLANN 匹配器）匹配两帧中的特征点。
- # 根据匹配结果，根据某种规则（例如 RANSAC 算法）筛选出符合要求的特征点匹配对。
- # 根据匹配对中的特征点位置，进行目标位置估计和跟踪。

#### 3.2 使用 SIFT 进行目标跟踪的 python 代码

SIFTtrack.py:

```

import cv2
import numpy as np

# 创建 SIFT 对象
sift = cv2.xfeatures2d.SIFT_create()

# 创建 FLANN 匹配器
flann = cv2.FlannBasedMatcher()

# 读取待追踪主体图像
init_frame = cv2.imread('spirit.jpg', 0) # 灰度图像

# 检测初始帧特征点和计算特征描述符
keypoints_init, descriptors_init = sift.detectAndCompute(init_frame, None)

# 创建视频捕获对象
cap = cv2.VideoCapture('spirit.mp4')

# 初始目标位置
bbox = (0, 0, 0, 0)

while True:
    # 读取下一帧图像
    ret, frame = cap.read()

    if not ret:
        break

    # 转换为灰度图像
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # 检测当前帧特征点和计算特征描述符
    keypoints_frame, descriptors_frame = sift.detectAndCompute(gray_frame, None)

    # 使用 FLANN 匹配器进行特征点匹配
    matches = flann.knnMatch(descriptors_init, descriptors_frame, k=2)

    # 筛选匹配对
    good_matches = []
    for m, n in matches:
        if m.distance < 0.7 * n.distance:

```

```

        good_matches.append(m)

# 获取匹配对中的特征点位置
src_points = np.float32([keypoints_init[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
dst_points = np.float32([keypoints_frame[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)

# 使用 RANSAC 算法估计变换矩阵
if len(good_matches) >= 4:
    M, _ = cv2.findHomography(src_points, dst_points, cv2.RANSAC, 5.0)

    # 计算目标位置（左上角坐标和宽高）
    h, w = init_frame.shape
    pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)

    if M is not None:
        dst = cv2.perspectiveTransform(pts, M)
        x, y, w, h = cv2.boundingRect(dst)
        # 更新目标位置
        bbox = (x, y, w, h)

# 绘制目标框
cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])), (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3])), (0, 255, 0), 2)

# 显示结果
cv2.imshow("Tracking", frame)

# 按下 ESC 键退出
if cv2.waitKey(1) == 27:
    break

# 释放资源
cap.release()
cv2.destroyAllWindows()

```

### 3.3 设计实验分析 SIFT 目标跟踪的性能

#### 3.3.1 实验设计思路

参考 2.3.1 的实验设计思路，针对图像质量、视角变化、遮挡和遮蔽这三个因素设计实验，分析每种因素对匹配效果的影响，进一步分析 SIFT 算法的性能。

实验所用视频以人偶为主体，以下是所有视频以及每个视频对应的待追踪主体图像：

- ①near2far\_good.mp4、near2far\_good.jpg：在图像质量好、视角变化小、无遮挡情况下，人偶在画面中由近及远移动；
- ②left2right\_good.mp4、left2right\_good.jpg：在图像质量好、视角变化小、无遮挡情况下，人偶在画面中由左向右移动；
- ③quality\_light.mp4、quality\_light.jpg：画面亮度由极暗变为正常；
- ④angle\_plane.mp4、angle\_plane.jpg：视角在平面旋转；
- ⑤angle\_stereo.mp4、angle\_stereo.jpg：视角在 3D 范围旋转；
- ⑥hide\_cover.mp4、hide\_cover.jpg：画面被遮挡。

#### 3.3.2 实验结果

对 3.3.1 中的 6 种情况运行程序，得到 6 个运行结果，运行结果是以对目标进行跟踪的视频形式呈现的。将运行结果录制成视频放置于文件夹中，视频经 5 倍加速处理，名称如下：near2far\_good 结果.mp4、left2right\_good 结果.mp4、quality\_light 结果.mp4、angle\_plane 结果.mp4、angle\_stereo 结果.mp4、hide\_cover 结果.mp4







|   |                   |                  |        |           |
|---|-------------------|------------------|--------|-----------|
|  | near2far_good结果   | 2023/11/24 19:22 | MP4 文件 | 21,054 KB |
|  | left2right_good结果 | 2023/11/24 19:32 | MP4 文件 | 26,495 KB |
|  | quality_light结果   | 2023/11/24 19:35 | MP4 文件 | 11,327 KB |
|  | angle_plane结果     | 2023/11/24 19:37 | MP4 文件 | 16,399 KB |
|  | angle_stereo结果    | 2023/11/24 19:39 | MP4 文件 | 20,943 KB |
|  | hide_cover结果      | 2023/11/24 19:41 | MP4 文件 | 7,965 KB  |

图 16：目标追踪实验结果

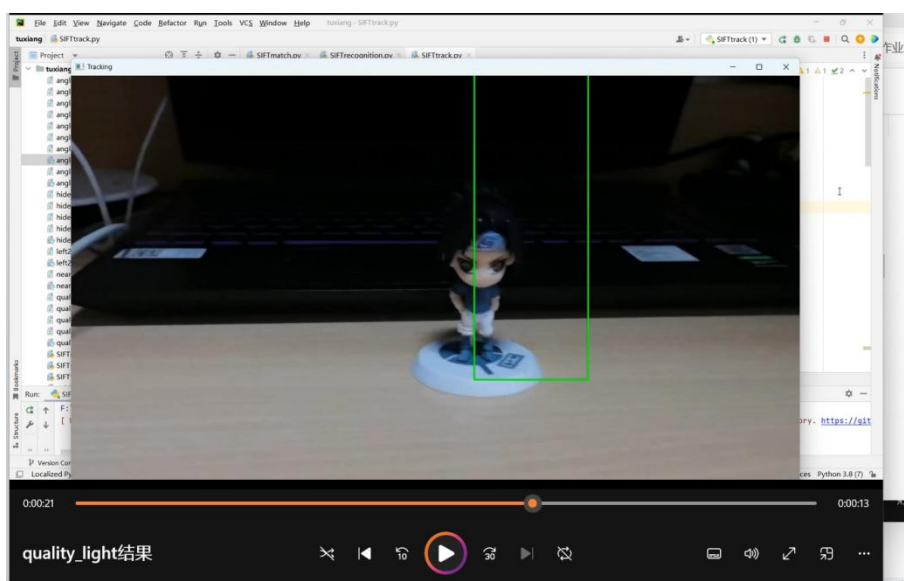
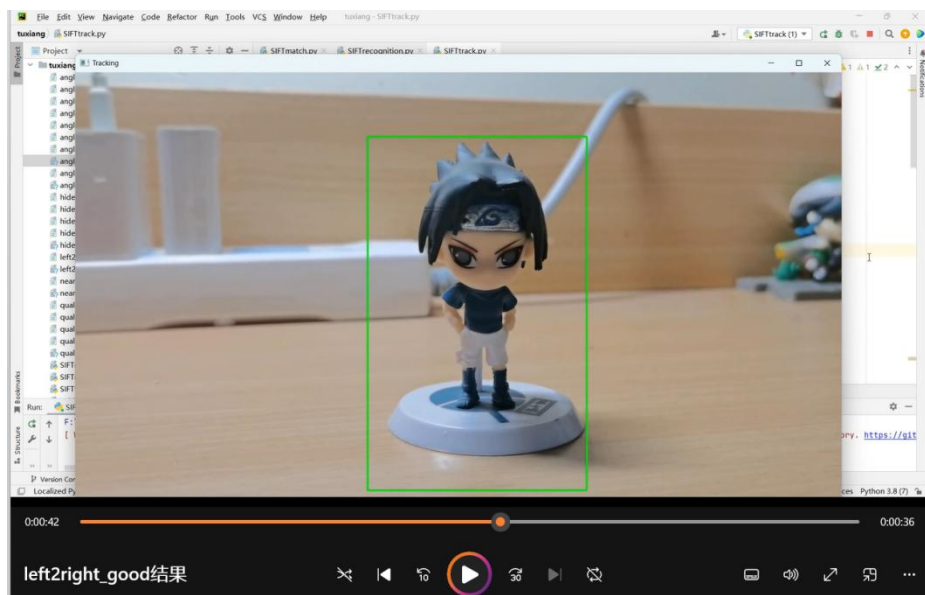
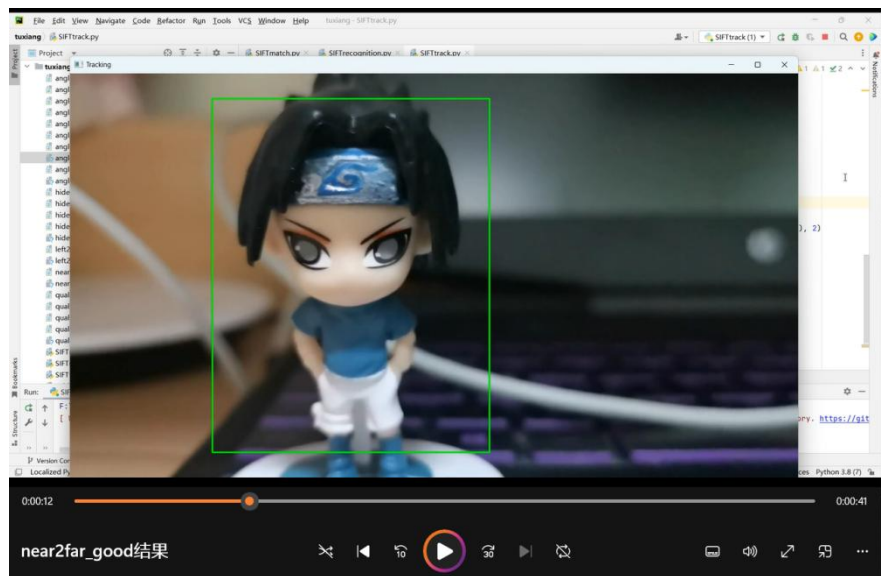


图 17：目标追踪实验结果截图



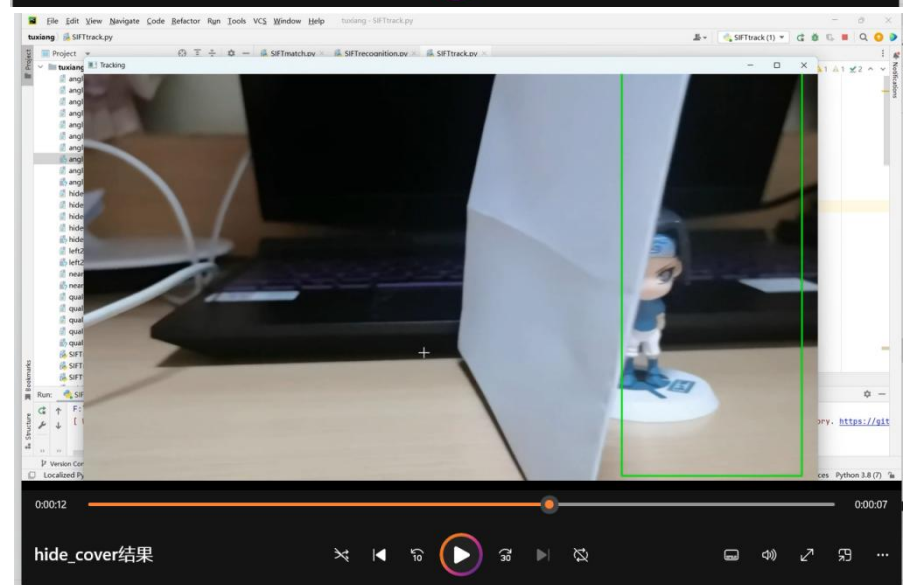
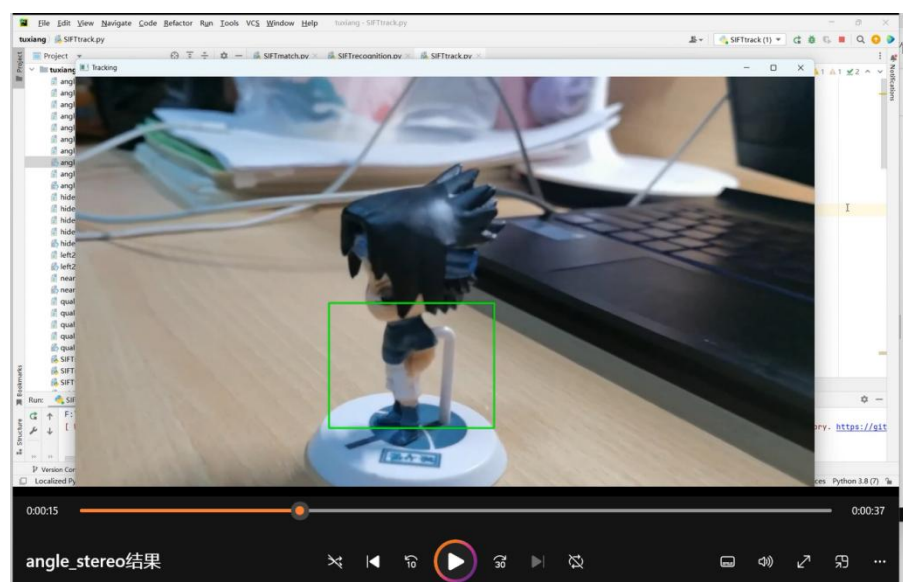
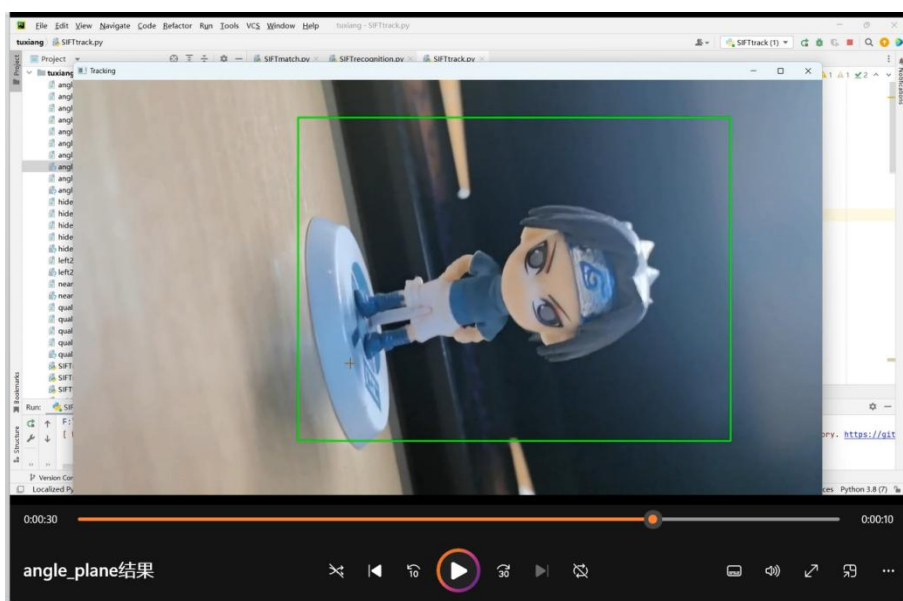


图 18: 目标追踪实验结果截图

### 3.3.3 实验结果分析

①near2far\_good.mp4、②left2right\_good.mp4、④angle\_plane.mp4 目标追踪效果较好；  
③quality\_light.mp4、⑤angle\_stereo.mp4、⑥hide\_cover.mp4 目标追踪效果很差。

这说明在直接使用简单的 SIFT 算法进行目标追踪时，追踪效果受到很大的限制。图像质量、视角变化和遮挡都会对追踪效果造成非常大的影响。

然而，这些问题可以在一定程度上被解决。

图像质量的影响：预先对视频进行处理，提升视频每一帧的图像质量，使视频亮度保持均匀，这样可以降低图像质量对追踪效果的限制；

视角变化的影响：完善对待追踪主体图像的特征提取，可以拍摄不同角度的待追踪主体图像，以应对视角变化；

### 3.4 总结 SIFT 目标跟踪算法的适用性

本实验中写的基于 SIFT 算法的目标跟踪方法相对较为简单且容易受到光照变化、视角变化和目標遮挡等因素的影响。现代的目标跟踪算法通常使用更复杂的技术和方法，例如深度学习的目标检测器和端到端的跟踪器等。

目标跟踪是一个连续的任务，在视频序列中实时检测和跟踪目标的位置。与 SIFT 不同，目标跟踪算法通常需要结合目标检测和运动估计的技术，以在连续帧中精确定位目标位置。

然而，可以基于 SIFT 算法构建更复杂的目标跟踪系统，通过在每帧中使用 SIFT 算法提取关键点和描述符，并使用匹配和运动估计技术来跟踪目标。这种方法可能不如现代目标跟踪算法那样准确和鲁棒，因为 SIFT 算法本身对于光照、视角和尺度变化相对较鲁棒，但对于快速运动、目标遮挡等情况可能会出现一些困难。

总之，虽然 SIFT 算法在特征提取和匹配方面具有良好的性能，但在目标跟踪任务中，需要结合其他更适合的算法和方法，以实现准确和鲁棒的目标跟踪。