

# 《计算机视觉导论》

## 大作业实验报告

### 1 问题背景和定义

#### 2 数据集的构建

2.1 三维重建样本采集

2.2 测试数据展示

#### 3 基础方案的实现

3.1 SIFT描述子

    3.1.1 方向分配

    3.1.2 建立描述符

    3.1.3 优点与特性

3.2 NN特征匹配算法

    3.2.1 Ratio Test

        Ratio Test

    3.2.2 NN-mutual

3.3 可视化与图像相似度比较

    3.3.1 投影效果的二维可视化

    3.3.2 基于感知哈希的相似度检测

        3.3.2.1 实现原理

        3.3.2.2 实现步骤

        3.3.2.3 代码实现

3.4 具体实现流程

    3.4.1 三维重建

    3.4.2 视觉定位（相机位姿估计）

3.5 效果展示

    3.5.1 三维重建效果

    3.5.2 视觉定位效果

#### 4 项目提升方案

4.1 新方法的调研

    4.1.1 新的特征提取方法

        4.1.1.1 基于传统视觉模型的特征提取

        4.1.1.2 基于深度学习的特征提取

        4.1.1.3 基于强化学习的特征提取

    4.1.2 基于学习的特征匹配

        4.1.2.1 传统的最近邻搜索算法

        4.1.2.2 PointCN

        4.1.2.3 OA-Net

        4.1.2.4 SuperGlue

        4.1.2.5 LightGlue

    4.1.3 hloc & pycolmap

4.2 提升方案的效果展示

    4.2.1 不同描述子下的三维点云模型展示

    4.2.2 不同算法的定位效果

4.3 鲁棒性检测

    4.3.2 较大位姿差异

4.3.3 有遮挡

4.3.4 夜间定位（极大光照差异）

## 5 结论与反思

5.1 基础解决方案

5.2 解决方案提升

# 1 问题背景和定义

重建和定位在VR导航等场景有着广泛的应用。基于图像对自身做定位的流程可以分为以下两个部分：

- 重建：基于一组多视角图像，恢复出场景的三维结构（点云与相机位姿）
- 定位：基于场景的重建，估计同一场景新采集图像的六自由度位姿。这里新采集图像可以是不同视角，以及不同时问拍摄的。

我们希望能够利用SFM来对校园的某一区域进行重建，来构建场景地图。并基于我们构建的场景点云地图来实现视觉定位，即将待定位图像与多张我们的重建图像进行特征匹配，最后再与三维点云模型匹配后求解PnP，恢复待定位的图像位姿。同时，我们还希望能够利用可视化的方法对比投影点云和实际图像的吻合程度来观察定位的准确性。

# 2 数据集的构建

## 2.1 三维重建样本采集

- 2023年12月8日傍晚采集
- 使用设备：组员手机后置摄像头
- 数据采集方法：使用4K, HDR围绕毛像底座走动录制360°+的环绕视频。
- 处理方法：总共4600+帧图片，最开始每隔100帧提取出一张图片，总共40+张图片，重建质量不好，最终通过探索，在速度和质量间权衡后采取每20帧提取一张，数据库总计233张图片。

处理视频数据对应的代码如下所示：

```
import cv2
import os

video_path = './data/4k-60.MOV'
save_path = './data/imgs'
skip_frames = 20 # 每20帧保存一次

if not os.path.exists(save_path):
    os.makedirs(save_path)

cap = cv2.VideoCapture(video_path)
if not cap.isOpened():
    print("打开视频流或文件时出错")

frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
fps = cap.get(cv2.CAP_PROP_FPS)
print("frame rate: ", fps)

# 总帧数
total_frames = cap.get(cv2.CAP_PROP_FRAME_COUNT)
print("total frames: ", total_frames)
```

```
counter = 0

while cap.isOpened():
    # 设置帧的位置
    cap.set(cv2.CAP_PROP_POS_FRAMES, counter * skip_frames)

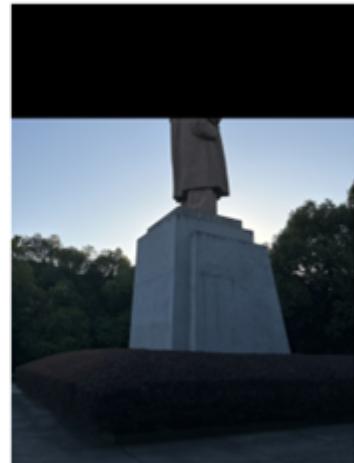
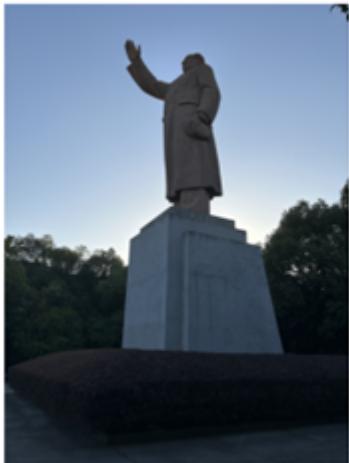
    ret, frame = cap.read()
    if ret:
        cv2.imwrite(os.path.join(save_path, str(int(counter)+1) + '.jpg'), frame)
        counter += 1
    else:
        break

cap.release()
```

## 2.2 测试数据展示

我们的测试数据具有以下特点：

- 不同日期采集的数据
- 不同光照条件下采集的数据（早晨、下午、夜晚）
- 有遮挡的采集数据
- 多角度不同机位的测试数据



## 3 基础方案的实现

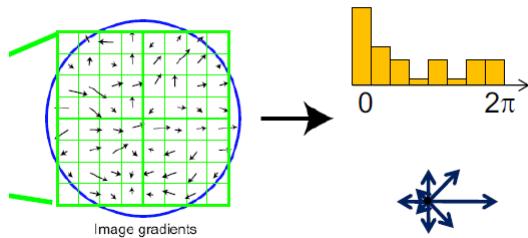
## 3.1 SIFT描述子

SIFT: Scale Invariant Feature Transform 对于图像缩放和旋转具有不变性

通过DoG找到了合法且稳定的 feature points 并确定了尺度后（因为使用了DoG，所以有尺度不变性），我们可以按照如下的步骤建立SIFT描述子。

### 3.1.1 方向分配

根据比例在关键点位置周围取一个邻域，并计算该邻域内每个像素的梯度大小和方向。随后我们创建一个包含36个bin的方向直方图，覆盖0-360度。假设某个点（在“方向收集区域”中）的梯度方向为 18.759 度，则它将进入 10-19 度的 bin。添加到容器中的“量”与该点的梯度大小成正比。对关键点周围的所有像素完成此操作后，直方图将在某个点出现峰值。



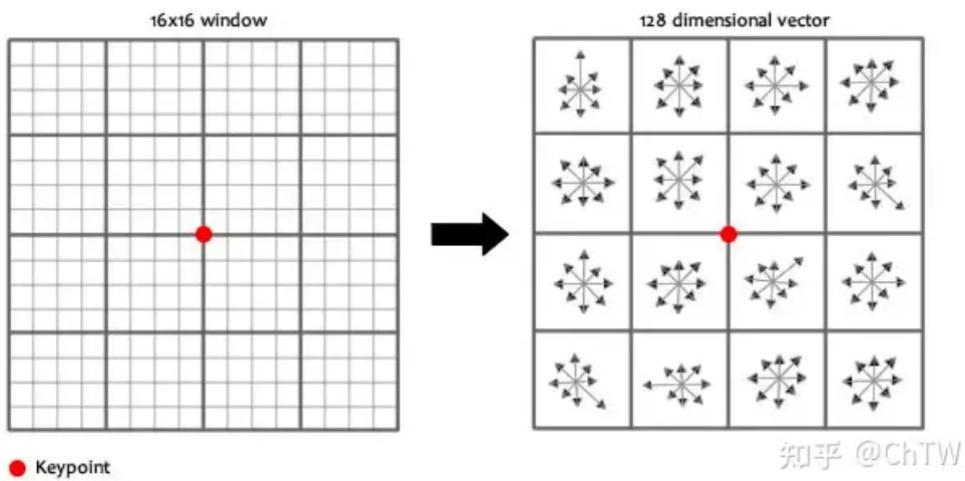
这样建立的基于梯度的直方图有以下特点：

- Captures important texture information 捕捉重要的纹理信息
- Robust to small translations /affine deformations 对微小的变换具有鲁棒性

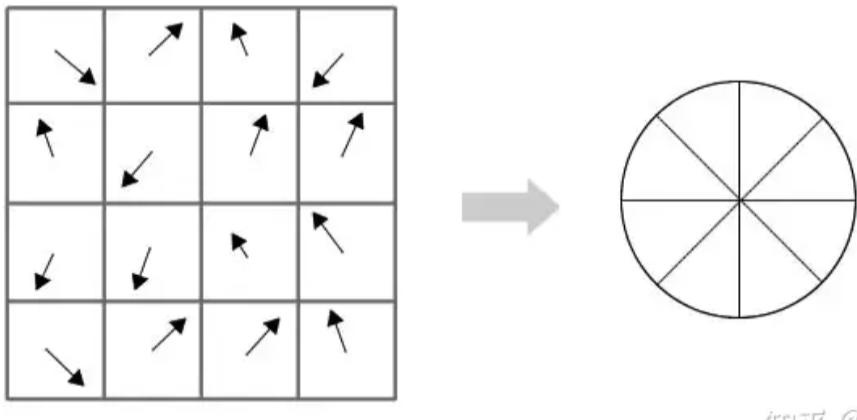
### 3.1.2 建立描述符

完成方向分配后，每一个key point 都有一个位置和对应的梯度方向（梯度方向取直方图中最高峰对应的方向）。然后我们计算关于每个关键点的局部图像区域的描述符。

为此，需要围绕关键点创建一个 16x16 的窗口。它被分成 16 个 4x4 大小的子块。



对于每个子块，创建 8 bin 方向直方图。



因此，我们还会得到这个 $4 \times 4 \times 8$ 方向给出的128个bin值。它被表示为特征向量以形成关键点描述符。（在opencv库中调用的SIFT描述子得到的结果就是一个128维向量，和这里得到了对应）

### 3.1.3 优点与特性

SIFT描述子具有缩放不变性和旋转不变性：

- 缩放不变性
  - scale invariant并不是SIFT描述子构建的方式给出的，而是在找到key points前就完成的。
  - 使用DoG方法进行特征点检测，可以确定每一个特征点的最佳尺度
- 旋转不变性：
  - 直接利用SIFT，若图像旋转之后，梯度最大方向肯定发生变化
  - 为了保证旋转不变性，我们对直方图进行标准化（统一将开始得到的最大梯度方向作为参照物，再此基础上旋转而不是基于原有的坐标系）

使用SIFT算子还可以有效处理光照依赖。

- 如果我们对较大的数字进行阈值处理，我们就可以实现光照独立性。因此，任何大于0.2的(128个中的)数字都将其更改为0.2。再次对生成的特征向量进行归一化。就得到与光照无关的特征向量！

关于SIFT描述子，有以下的特性：

- 局部性：我们获取的是局部特征，对于遮挡和杂乱具有鲁棒性
- 可以解决视角变换：缩放不变性和旋转不变性
- 可以处理光照的significant changes：阈值化后的光照独立性
- 独特性：单个特征可以与大型对象数据库匹配
- 数量：即使是小物体也可以生成许多特征
- 效率：接近实时性能，高效
- 可扩展性：可以很容易地扩展到各种不同的特征类型，每一种都可增加鲁棒性

## 3.2 NN特征匹配算法

NN特征匹配算法实际上是非常朴素和直观的。我们知道特征描述子最后会以向量的形式来表达，那我们直接使用向量间的L2距离来表示两个特征之间的相似度。如下所示：

$$\text{Distance}(f_1, f_2) = \|f_1 - f_2\|$$

但直接使用这样的距离检测，对于很相似的描述子可能会引发混淆。因此我们可以采用两种过滤混淆匹配的方式来防止错误匹配的发生。

### 3.2.1 Ratio Test

#### Ratio Test

对于图像1中的某一key point, 找到和它距离最近的描述子  $f_2$  和距离第二近的  $f'_2$ , 计算如下的 Ratio score:

$$t = \frac{\|f_1 - f_2\|}{\|f_1 - f'_2\|}$$

我们可以发现, Ambiguous matches有较大的 ratio score, 因此我们可以设定一个阈值, 在该key point 对应的 ratio score 大于该阈值时就删去该匹配。

### 3.2.2 NN-mutual

另一种方式是去找这一匹配中两个点分别对应的最近邻, 若:

- $f_2$  is the nearest neighbor of  $f_1$  in  $I_2$
- $f_1$  is the nearest neighbor of  $f_2$  in  $I_1$

则我们认为  $f_1, f_2$  为一组匹配。

## 3.3 可视化与图像相似度比较

### 3.3.1 投影效果的二维可视化

通过图像的信息读取内参矩阵, 通过位姿估计结果获得外参矩阵。利用内外参3维重建的点云坐标投影到像素点上。实现了投影特征点和投影整个稀疏点云两种模式。

```
def project_cloud_to_image(model, camera, image_dir, query, ret):
    qvec = ret['qvec'] # 旋转向量
    tvec = ret['tvec'] # 平移向量
    intrinsic = camera.calibration_matrix()
    extrinsic = np.eye(4)
    # 计算旋转矩阵 R
    theta = 2 * np.arccos(qvec[0]) # 角度
    axis = qvec[1:] / np.sin(theta/2) # 轴
    R = np.eye(3) * np.cos(theta) + (1 - np.cos(theta)) * np.outer(axis, axis) + np.sin(theta) *
    * np.array([[0, -axis[2], axis[1]], [axis[2], 0, -axis[0]], [-axis[1], axis[0], 0]])

    # 构建外参矩阵 P
    extrinsic = np.eye(4)
    extrinsic[:3, :3] = R
    extrinsic[:3, 3] = tvec

    # 所有特征点的点云坐标
    # inl_3d = np.array([model.points3D[pid].xyz for pid in np.array(log['points3D_ids'])])
    [ret['inliers']])
    # 所有点云的坐标
    inl_3d_ids = [pid for pid in model.points3D]
    inl_3d = np.array([model.points3D[pid].xyz for pid in np.array(inl_3d_ids)])
    # 将世界坐标系下的点云转换到相机坐标系下
    inl_3d = np.dot(inl_3d, R.T) + tvec
    # 将相机坐标系下的点云转换到像素坐标系下
    inl_2d = np.dot(inl_3d, intrinsic.T)
    # 从齐次坐标转换到非齐次坐标
    inl_2d = inl_2d[:, :2] / inl_2d[:, 2:]
```

```

# print(inl_2d.shape)
# 所有特征点的颜色
# inl_3d_color = np.array([model.points3D[pid].color for pid in
np.array(log['points3D_ids'])[ret['inliers']]])
# 所有点云的颜色
inl_3d_color = np.array([model.points3D[pid].color for pid in np.array(inl_3d_ids)])]

width = camera.width
height = camera.height
# 剔除超出图像范围的点云
inl_3d_color = inl_3d_color[(inl_2d[:, 0] >= 0) & (inl_2d[:, 0] < width) & (inl_2d[:, 1]
>= 0) & (inl_2d[:, 1] < height)]
inl_2d = inl_2d[(inl_2d[:, 0] >= 0) & (inl_2d[:, 0] < width) & (inl_2d[:, 1] >= 0) &
(inl_2d[:, 1] < height)]

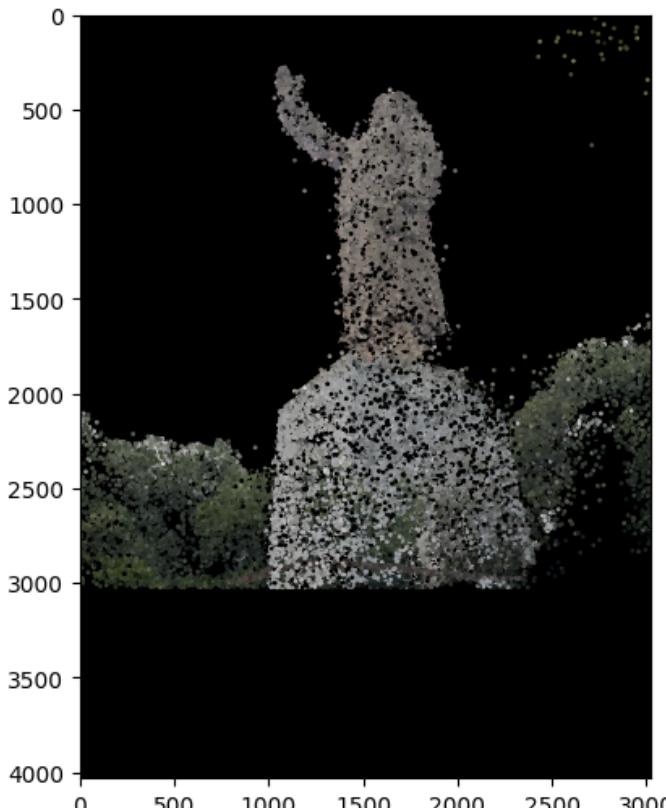

plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
plt.scatter(inl_2d[:, 0], inl_2d[:, 1], c=inl_3d_color / 255.0, s=1)
# 设置y轴的方向
plt.gca().invert_yaxis()

plt.imshow(np.zeros((width, height, 3)))
plt.subplot(1, 2, 2)
plt.imshow(read_image(image_dir / query))
plt.show()

```

本来想直接利用投影点的信息直接制作一张照片的，但通过实际试错后发现10w+的特征点对于4K分辨率下的图片信息实在太少了，导致做出来的图片全黑。最后采用根据投影点信息描散点来解决这个问题。

投影效果如下：



### 3.3.2 基于感知哈希的相似度检测

感知哈希算法（Perceptual Hash Algorithm，简称pHash）是哈希算法的一种，主要用来做相似图片的搜索工作。

#### 3.3.2.1 实现原理

感知哈希算法（pHash）首先将原图像缩小成一个固定大小的像素图像，然后将图像转换为灰度图像，通过使用离散余弦变换（DCT）来获取频域信息。然后，根据DCT系数的均值生成一组哈希值。最后，利用两组图像的哈希值的汉明距离来评估图像的相似度。

#### 3.3.2.2 实现步骤

- 缩小图像：将目标图像缩小为一个固定的大小（我们选择了  $128 \times 128$  的图像大小）。作用是去除各种图像尺寸和图像比例的差异，只保留结构、明暗等基本信息，目的是确保图像的一致性，降低计算的复杂度。
- 图像灰度化：将缩小的图像转换为灰度图像。
- 离散余弦变换（DCT）：感知哈希算法的核心是应用离散余弦变换。DCT将图像从空间域（像素级别）转换为频域，得到128的DCT变换系数矩阵，以捕获图像的低频信息。
- 计算灰度均值：计算DCT变换后图像块的均值，以便后面确定每个块的明暗情况。
- 生成二进制哈希值：如果块的DCT系数高于均值，表示为1，否则表示为0。
- Hamming距离计算：检查两张图片对应哈希值的汉明距离，并计算对应的相似度。

#### 3.3.2.3 代码实现

```
def compare_img_p_hash(img1, img2):  
    """  
        Get the similarity of two pictures via pHash  
        Attention: this is not accurate compare_img_hist() method, so use hist() method to  
        auxiliary comparision.  
        This method is always used for graphical search applications, such as Google  
        Image(Use photo to search photo)  
    •  
        :param img1:  
        :param img2:  
        :return:  
    """  
    return similarity(img1, img2)  
  
def get_img_p_hash(img):  
    """  
        Get the pHash value of the image, pHash : Perceptual hash algorithm(感知哈希算法)  
        :param img: img in MAT format(img = cv2.imread(image))  
        :return: pHash value  
    """  
    hash_len = 128  
  
    # GET Gray image  
    gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)  
  
    # Resize image, use the different way to get the best result  
    resize_gray_img = cv2.resize(gray_img, (hash_len, hash_len), cv2.INTER_AREA)  
  
    # Change the int of image to float, for better DCT  
    height, width = resize_gray_img.shape[:2]  
    vis0 = np.zeros((height, width), np.float32)  
    vis0[:height, :width] = resize_gray_img  
  
    # DCT: Discrete cosine transform(离散余弦变换)
```

```

vis1 = cv2.dct(cv2.dct(vis0))
vis1.resize(hash_len, hash_len)
img_list = vis1.flatten()

# Calculate the avg value
avg = sum(img_list) * 1.0 / len(img_list)
avg_list = []
for i in img_list:
    temp = '1' if i > avg else '0'
    avg_list.append(temp)
# Calculate the hash value
p_hash_str = ''
for x in range(0, hash_len * hash_len, 4):
    p_hash_str += '%x' % int(''.join(avg_list[x:x + 4]), 2)
return p_hash_str

def ham_dist(x, y):
    """
    Get the hamming distance of two values.
    hamming distance(汉明距)
    :param x:
    :param y:
    :return: the hamming distance
    """
    return sum([ch1 != ch2 for ch1, ch2 in zip(x, y)])

def similarity(img1, img2):
    """
    Get the similarity of two pictures via pHash
    """
    hash_img1 = get_img_p_hash(img1)
    hash_img2 = get_img_p_hash(img2)
    difference = 0
    assert len(hash_img1) == len(hash_img2)
    for i in range(len(hash_img1)):
        if hash_img1[i] != hash_img2[i]:
            difference += 1
    return 1 - difference / len(hash_img1)

```

## 3.4 具体实现流程

### 3.4.1 三维重建

采取 hloc 现成库进行稀疏点云的重建，重建后调用 pycolmap 读取重建后的模型。

```

import pycolmap
from pathlib import Path

output_path = Path("../")
image_dir = Path("../..../datasets/great_man/")

# 查看模型是否读取成功
model = pycolmap.Reconstruction("./")
print(model.summary())

```

```

for image_id, image in model.images.items():
    print(image_id, image)
    break

for point3D_id, point3D in model.points3D.items():
    print(point3D_id, point3D)
    break

for camera_id, camera in model.cameras.items():
    print(camera_id, camera.calibration_matrix())
    break

```

### 3.4.2 视觉定位（相机位姿估计）

视觉定位同样基于 hloc 封装实现。

配置特征描述子、特征匹配方法、结果保存路径以及特征提取和匹配：

```

from hloc import extract_features, match_features, pairs_from_exhaustive, visualization

features = output_path / 'features.h5'
matches = output_path / 'matches.h5'

feature_conf = extract_features.confs['disk']
matcher_conf = match_features.confs['disk+lightglue']
loc_pairs = output_path / 'pairs-loc.txt'

```

估计流程为：



位姿估计代码如下：

```

from hloc.localize_sfm import QueryLocalizer, pose_from_cluster

camera = pycolmap.infer_camera_from_image(image_dir / query)
print(camera.calibration_matrix()) # 相机内参矩阵

# 数据库中的参考图像
references_registered = [model.images[i].name for i in model.reg_image_ids()]
ref_ids = [model.find_image_with_name(n).image_id for n in references_registered]

conf = {
    'estimation': {'ransac': {'max_error': 12}},
    'refinement': {'refine_focal_length': True, 'refine_extra_params': True},
}

localizer = QueryLocalizer(model, conf)
ret, log = pose_from_cluster(localizer, query, camera, ref_ids, features, matches)
print(f'found {ret["num_inliers"]}/{len(ret["inliers"])} inlier correspondences.')
visualization.visualize_loc_from_log(image_dir, query, log, model)

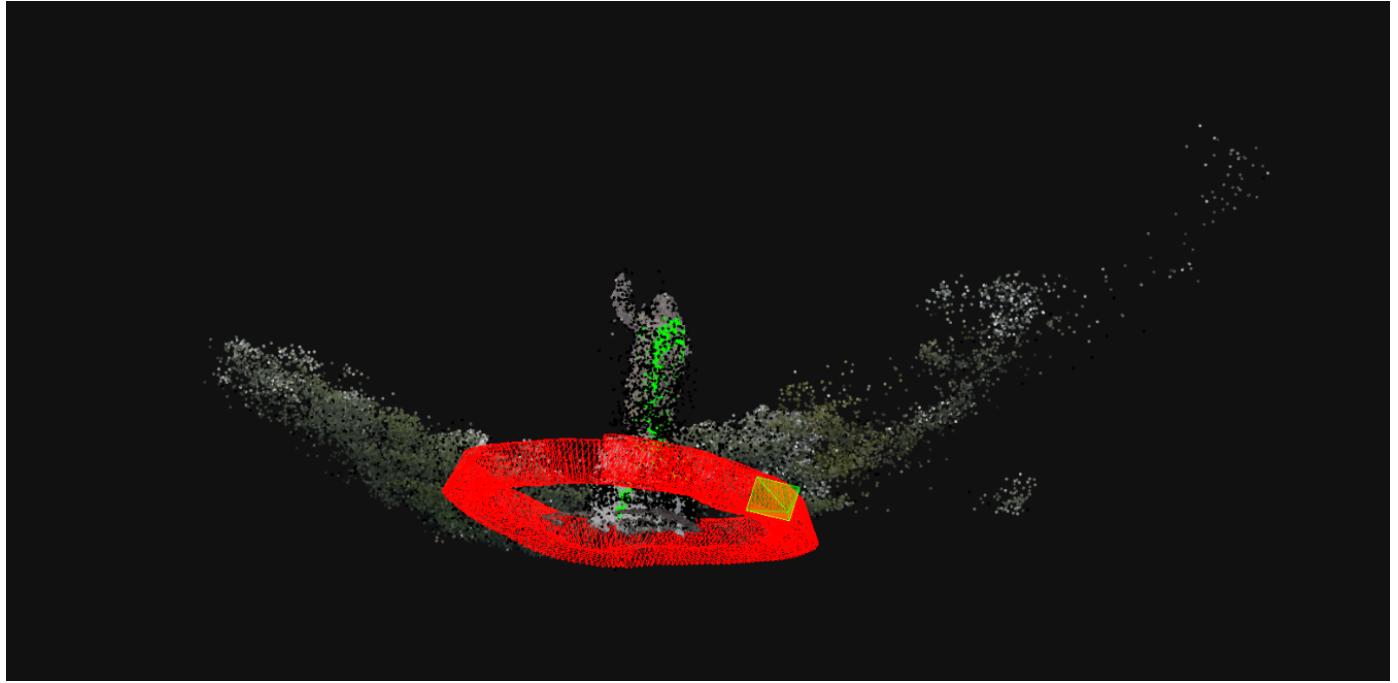
```

通过返回参数在3维模型中可视化相机位姿：

```

pose = pycolmap.Image(tvec=ret['tvec'], qvec=ret['qvec']) # 相机位姿
viz_3d.plot_camera_colmap(fig, pose, camera, color='rgba(0,255,0,0.5)', name=query, fill=True)
inl_3d = np.array([model.points3D[pid].xyz for pid in np.array(log['points3D_ids'])])
[ret['inliers']]])
viz_3d.plot_points(fig, inl_3d, color="lime", ps=1, name=query)
fig.show()

```



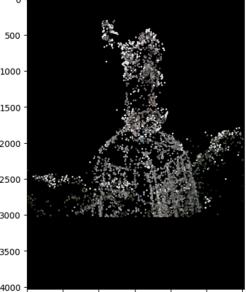
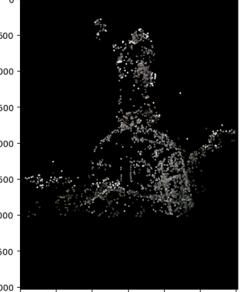
## 3.5 效果展示

### 3.5.1 三维重建效果

	SIFT Descriptor + NN-ratio	SIFT Descriptor + NN-mutual
三维重建耗时	18min12s	16min35s
三维重建模型效果		

### 3.5.2 视觉定位效果

	SIFT Descriptor + NN-ratio	SIFT Descriptor + NN-mutual
视觉定位时间	9.45s	10.27s
感知哈希算法相似度比较	0.5096	0.4543

	SIFT Descriptor + NN-ratio	SIFT Descriptor + NN-mutual
投影效果	 	 

## 4 项目提升方案

### 4.1 新方法的调研

#### 4.1.1 新的特征提取方法

##### 4.1.1.1 基于传统视觉模型的特征提取

- HOG: Histogram of oriented gradients (梯度方向直方图)

对于HOG特征描述子，选用梯度方向的分布作为特征。一张图像的梯度（x和y方向的导数）很有用因为在边缘和拐角（强度变化剧烈的区域）处的梯度幅值很大。而且我们知道边缘和拐角比其他平坦的区域包含更多关于物体形状的信息。

相比于SIFT而言，HOG算法通常用于描述整个图像，并使用同一尺度。HOG梯度使用领域容器进行规范化。

- SURF: Speeded Up Robust Features (加速鲁棒特征)

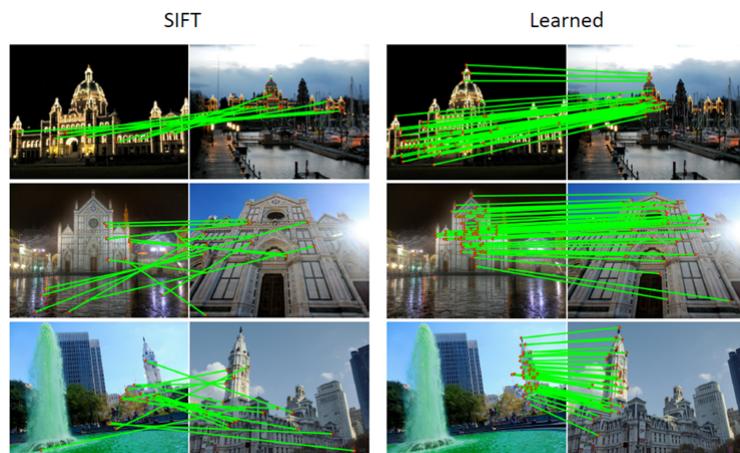
SURF是一种稳健的图像识别和描述算法。它是SIFT的高效变种，也是提取尺度不变特征，算法步骤与SIFT算法大致相同，但采用的方法不一样，要比SIFT算法更高效（正如其名）。SURF使用海森(Hessian)矩阵的行列式值作特征点检测并用积分图加速运算；SURF的描述子基于2D离散小波变换响应并且有效地利用了积分图。

SURF与SIFT描述子之间的比较如下所示：

比较项目	SIFT	SURF
尺度空间极值检测	使用高斯滤波器，根据不同尺度的高斯差(DOG)图像寻找局部极值	使用方形滤波器，利用海森矩阵的行列式值检测极值，并利用积分图加速运算
关键点定位	通过邻近信息插补来定位	与SIFT类似
方向定位	通过计算关键点局部邻域的方向直方图，寻找直方图中最大值的方向作为关键点的主要方向	通过计算特征点周围像素点x,y方向的哈尔小波变换，将x、y方向小波变换的和向量的最大值作为特征点方向
特征描述子	是关键点邻域高斯图像梯度方向直方图统计结果的一种表示，是 $16 \times 8 = 128$ 维向量	是关键点邻域2D离散小波变换响应的一种表示，是 $16 \times 4 = 64$ 维向量
应用中的主要区别	通常在搜索正确的特征时更加精确，当然也更加耗时	描述子大部分基于强度的差值，计算更快捷

#### 4.1.1.2 基于深度学习的特征提取

自AlexNet在ImageNet上的优异表现后，大家发现了深度学习和神经网络在图像处理中的优异表现，利用深度学习进行特征提取成为主流：

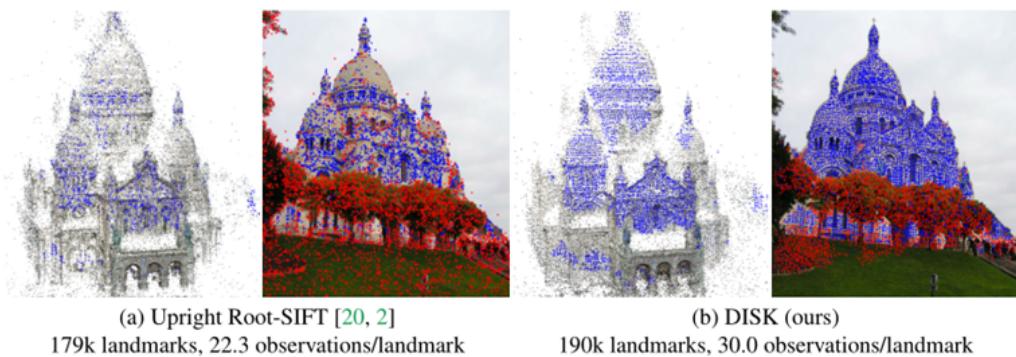


- **SuperPoint:** 自监督学习，同时提取特征点位置和描述子，而且，由于在模型中以位姿计算损失函数，所以提出的特征点和描述子在位姿解算方面更有优势。
  - **D2-Net:** 通过 CNN 得到特征图，然后同时计算描述符并检测特征点，可以密集提取描述符。
  - **R2D2:** D2-Net的优化，专为检测得分设计了一个损失，也就是局部的得分和对应局部的得分要相似，拉开局部最大值和均值差距作为正则化。
    - **Reliable:** 特征点在各种拍摄视角、光线、季节变化时依然能检测出来，并且能得到正确的特征点匹配
    - **Repeatable:** 针对纹理等重复性强的信息也可以提取特征并获得较好的匹配效果

#### 4.1.1.3 基于强化学习的特征提取

我们介绍基于强化学习的特征提取方法：DISK

- 依赖策略梯度算法的强化学习（蒙特卡罗采样 + 梯度下降）
  - 端到端可训练的方法，有更好的性能
  - 可非常密集的提取特征关键点，并能辨别特征点间的区别



我们希望我们重建的三维点云模型中有更多的特征点，以提高我们得到正确匹配的数目，从而最小化重投影误差得到更好的视觉定位效果。因此我们最后选取disk来作为我们的特征提取方法。

#### 4.1.2 基于学习的特征匹配

#### 4.1.2.1 传统的最近邻搜索算法

- 得到的匹配关系存在大量的误匹配（可能高达95%），因此需要识别并剔除错误匹配（outlier rejection）。
- 比例测试、相互最近邻和RANSAC等传统方法在视角变换大、亮度变换剧烈、存在遮挡等情况中无法取得满意效果。

#### 4.1.2.2 PointCN

- 一种基于学习的寻找最优匹配点的方法
- 利用多层感知机为每一对候选的匹配点进行加权
- 增加了上下文规范化层

#### 4.1.2.3 OA-Net

- PointCN的优化，强化了local/global context capture
- 为抓取局部特征，从GNN中引入可微池化操作
- 基于可微池化实现排序，方便反池化

#### 4.1.2.4 SuperGlue

- 基于图卷积神经网络的特征匹配算法
- 特征匹配：求解可微分最优化转移问题
- 损失函数通过GNN来构建
- 利用自注意和交叉注意力实现匹配

#### 4.1.2.5 LightGlue

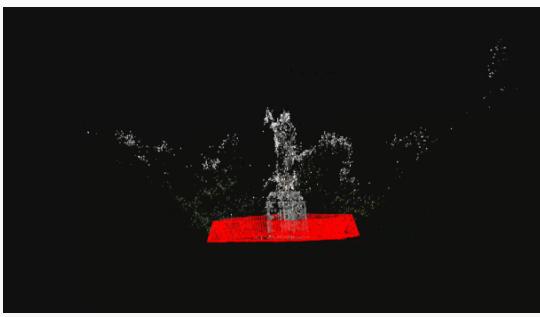
- 在准确性、效率和训练易用性方面优于SuperGlue
- 具有自适应的特性，可以根据图像对的难度进行灵活调整
- 特别适用于对延迟敏感的应用

### 4.1.3 hloc & pycolmap

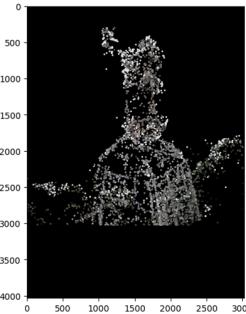
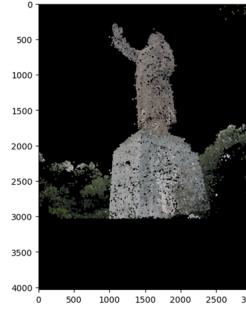
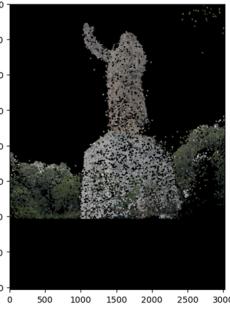
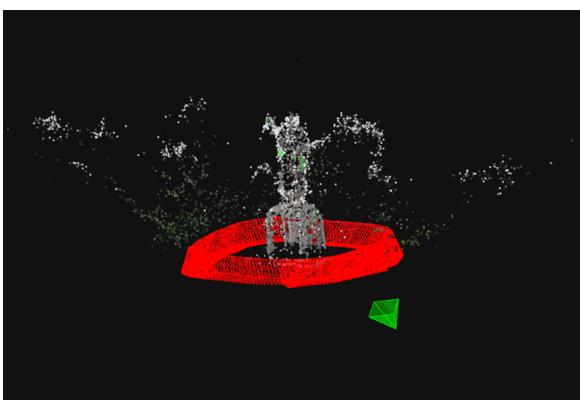
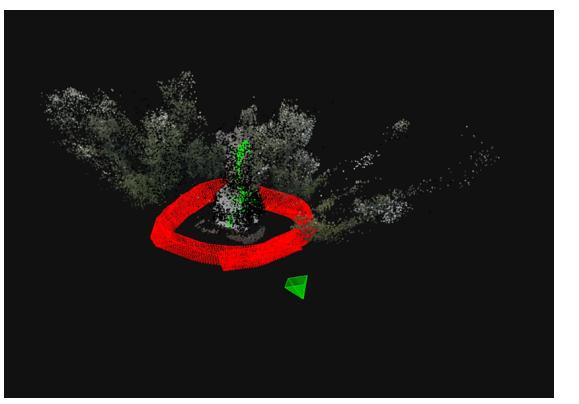
我们根据助教给出参考文献，发现了基于三维重建和视觉定位的框架 Hloc，而 Hloc 的实现基于 Pycolmap，因此我们的整体实验也在 pycolmap 和 Hloc 框架下进行。

## 4.2 提升方案的效果展示

### 4.2.1 不同描述子下的三维点云模型展示

	SIFT Descriptor + NN-ratio	DISK + NN-ratio
三维重建耗时	18min12s	29min27s
三维重建模型效果		

## 4.2.2 不同算法的定位效果

	SIFT Descriptor + NN-ratio	SIFT Descriptor + NN-mutual
视觉定位时间	9.45s	10.27s
感知哈希算法 相似度比较	0.5096	0.4543
投影效果	 	 
	DISK + NN-ratio	DISK + LightGlue
视觉定位时间	10.83s	19.15s
感知哈希算法 相似度比较	0.8269	0.7762
投影效果	 	 
视觉定位效果		

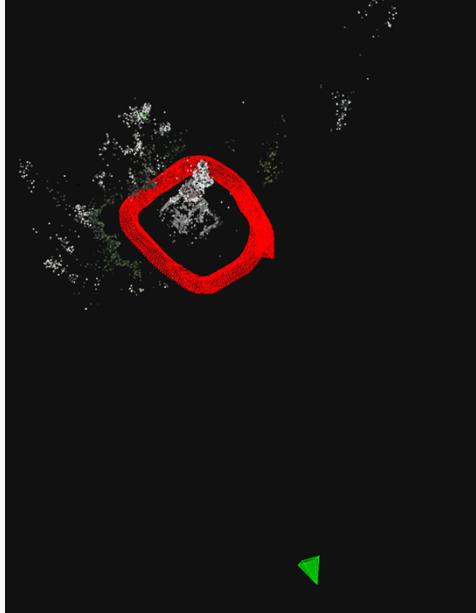
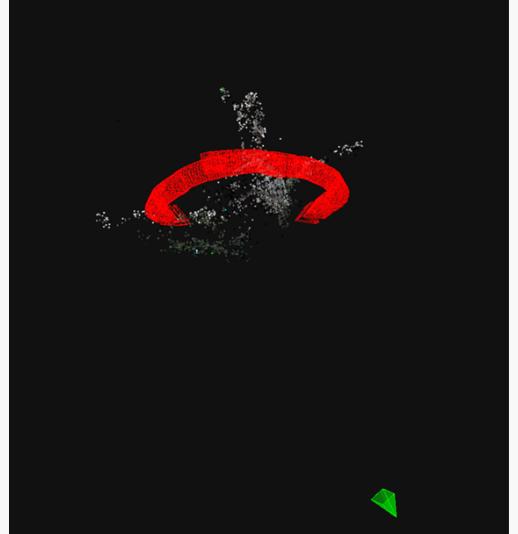
## 4.3 鲁棒性检测

### 4.3.2 较大位姿差异

待定位图像展示如下所示：

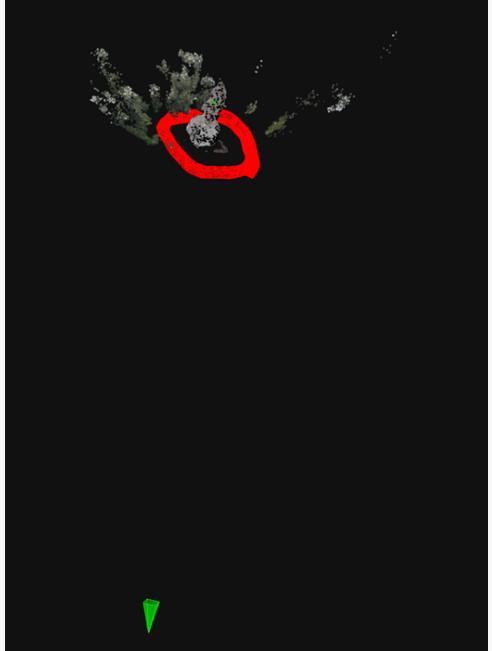


在传统视觉模型和传统匹配算法下的表现如下：

	SIFT Descriptor + NN-ratio	SIFT Descriptor + NN-mutual
感知哈希算法相似度比较	0.4745	0.3782
视觉定位效果		

在DISK特征提取和LightGlue特征匹配下的表现：

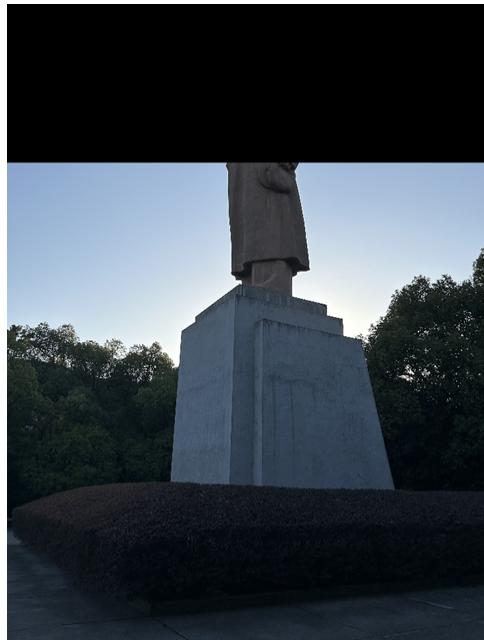
	DISK+NN-ratio	DISK+Lightglue
感知哈希算法相似度比较	0.5305	0.6632

	<b>DISK+NN-ratio</b>	<b>DISK+Lightglue</b>
视觉定位效果		

可以发现，在相机位姿和三维重建用到图像的位姿差异较大时，只有DISK + LightGlue保持了良好的定位效果，别的方法都存在严重的不足。

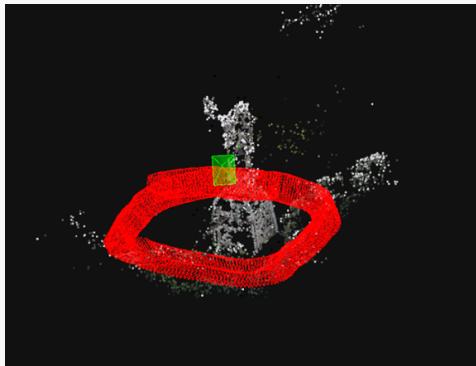
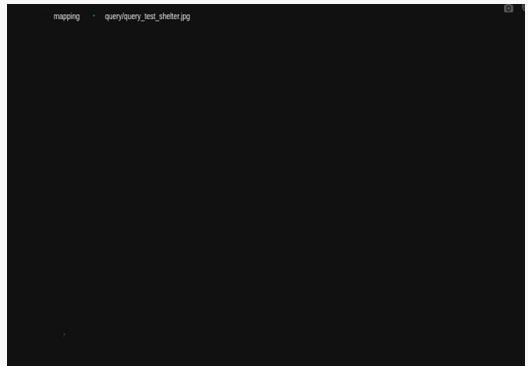
### 4.3.3 有遮挡

待定位图像展示如下所示：

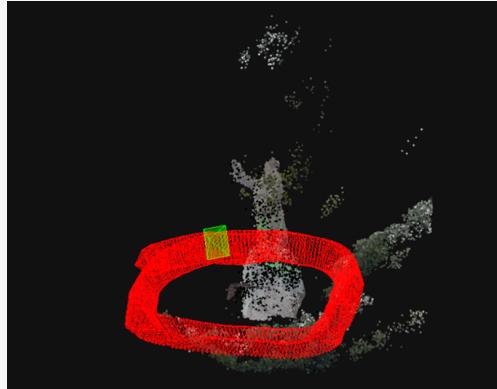
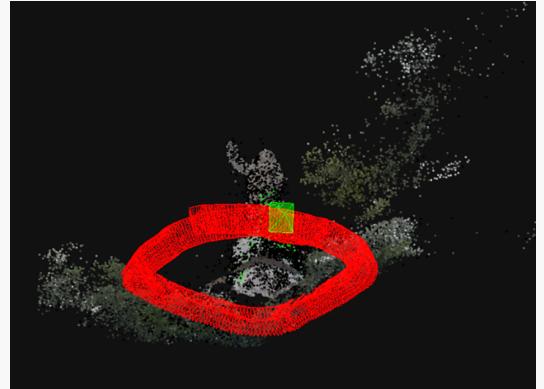


在传统视觉模型和传统匹配算法下的表现如下：

	<b>SIFT Descriptor + NN-ratio</b>	<b>SIFT Descriptor + NN-mutual</b>
感知哈希算法相似度比较	0.3576	失败

	SIFT Descriptor + NN-ratio	SIFT Descriptor + NN-mutual
视觉定位效果		

在DISK特征提取和LightGlue特征匹配下的表现：

	DISK+NN-ratio	DISK+Lightglue
感知哈希算法相似度比较	0.7883	0.7118
视觉定位效果		

可以发现，在有遮挡的情况下，只是简单的遮挡就对使用SIFT + NN构建的定位模型精度产生了较大影响；但对于使用DISK特征提取方法来进行的视觉定位，我们可以发现简单的遮挡对于模型的定位影响不大（可能是因为提取出的特征点比较密集），具有较强鲁棒性。

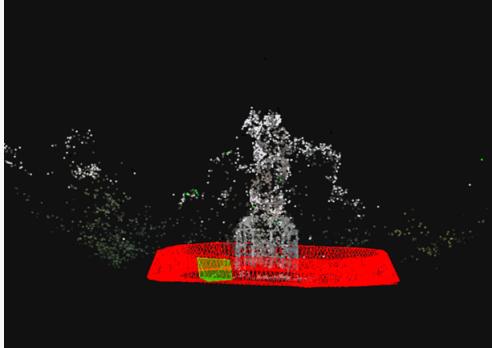
#### 4.3.4 夜间定位（极大光照差异）

待定位图像展示如下所示：

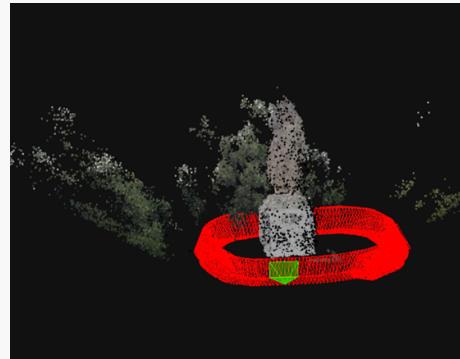
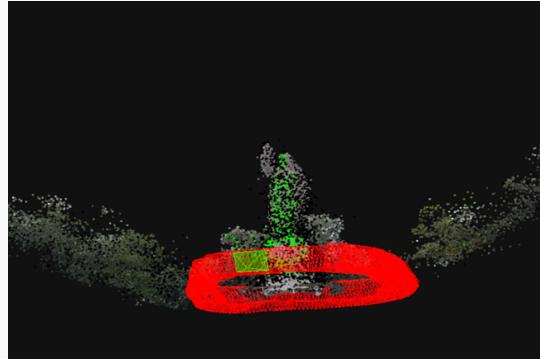


在传统视觉模型和传统匹配算法下的表现如下：

	SIFT Descriptor + NN-ratio	SIFT Descriptor + NN-mutual
--	----------------------------	-----------------------------

	SIFT Descriptor + NN-ratio	SIFT Descriptor + NN-mutual
感知哈希算法相似度比较	0.4187	匹配到了43/3918个特征，定位结果完全偏离
视觉定位效果		

在DISK特征提取和LightGlue特征匹配下的表现：

	DISK+NN-ratio	DISK+Lightglue
感知哈希算法相似度比较	0.8237	0.7834
视觉定位效果		

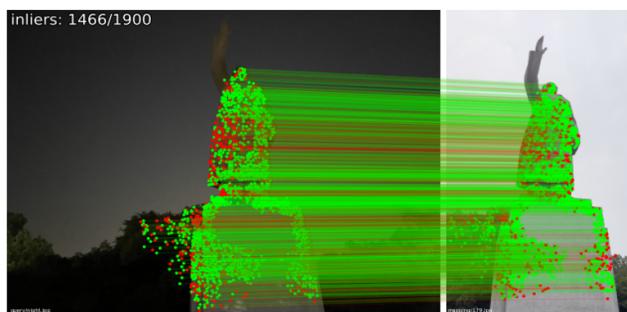
我们发现，虽然SIFT描述子本身已经对光照具有鲁棒性，但是在光照条件差距较大时，SIFT + NN来进行视觉定位得到的定位效果依然很差，提取出的特征点也很难进行匹配。

而对于DISK描述子进行的视觉匹配，我们发现使用NN-ratio算法匹配的特征数目也很少，只匹配到3对点，刚好可以P3P；但由于匹配本身的精确性，我们依旧可以较好地得到相机位姿信息实现定位。而使用DISK + lightglue来进行视觉定位，匹配到的特征点数目并没有受到光照条件的变化而有非常显著的影响。

## □ DISK+NN-ratio



## □ DISK + Lightglue



综上所述，我们发现DISK + LightGlue方法对于视觉定位最稳健，在角度差异大、有遮挡、光照条件差异大的情况下同样有很好的表现。

# 5 结论与反思

---

## 5.1 基础解决方案

---

- 基于SIFT+NN的传统视觉模型构建的三维点云相对稀疏，但也能完成基本的定位任务
- 基础解决方案的鲁棒性较差，在光照、位姿差异大或有遮挡时表现较差
- 使用感知Hash算法检测图像相似度在定位精度较高或较低时区分度不大

## 5.2 解决方案提升

---

- 调研了多种基于学习的特征提取方法，最终选择最适合用于点云定位的DISK描述子
- DISK描述子可非常密集的提取特征关键点，得到较为稠密的特征点云方便后续定位
- LightGlue匹配算法在速度上逊于NN算法，但自适应特性使其有更好的鲁棒性
- DISK + Lightglue 在三维重建和视觉定位中都有更好的表现