HW2

2053932 雷翔

Q0:

对于TUM数据集（rgbd_dataset_freiburg1_xyz）中的第（学号%30+40）张图像和第（学号%30+50）张图像之间通过某种特征匹配方法（不限）找到特征点，根据找到的特征点：

2053932 % 30 + 40 = 52 and 2053932 % 30 + 50 = 62

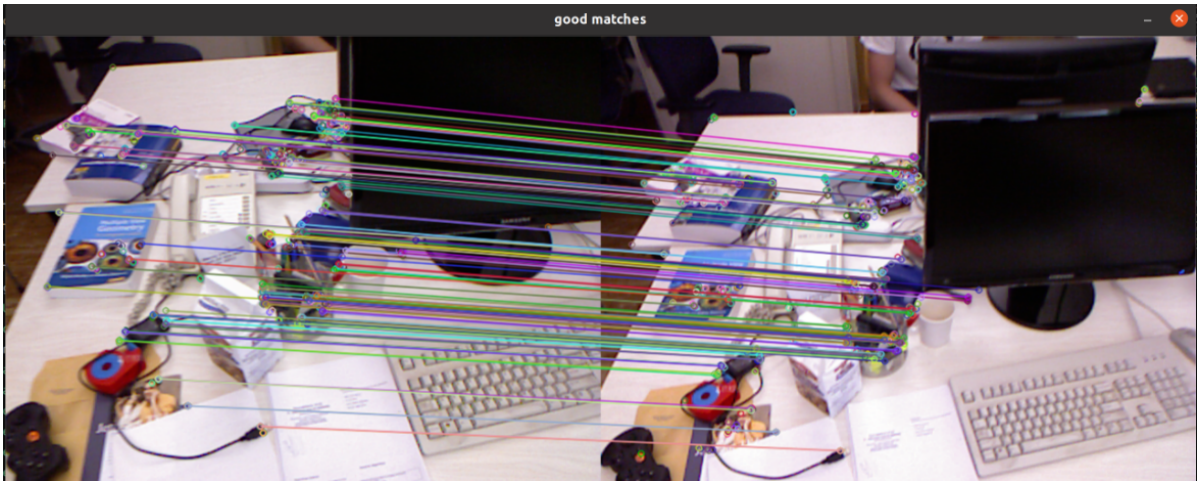
故选择第 52 张图片和第 62 张图片
第 52 张图片：1305031103.875361.png
第 62 张图片：1305031104.211283.png

第一步:检测 Oriented FAST 角点位置

第二步:根据角点位置计算 BRIEF 描述子

第三步:对两幅图像中的BRIEF描述子进行匹配，使用 Hamming 距离

匹配结果如下图：



Q1:

利用2D-2D对极约束计算E，F和H，进而分解得到R，t；

```
./pose_estimation_2d2d /home/lei/桌面/SLAM/slambook2/TUM/rgb/1305031103.875361.png
/home/lei/桌面/SLAM/slambook2/TUM/rgb/1305031104.211283.png
```

本质矩阵 $\mathbf{E}$：

$$\mathbf{E} = \begin{bmatrix} -0.009523805823273212 & -0.6616437055630591 & 0.2341097265746704 \\ 0.6520153881673575 & -0.01838206842971534 & 0.07098045690408235 \\ -0.2654893377374404 & -0.07962774148352751 & 0.002045032867384192 \end{bmatrix}$$

基础矩阵 $\mathbf{F}$：

$$\mathbf{F} = \begin{bmatrix} -5.19425056866933 \times 10^{-6} & 5.115074137663505 \times 10^{-5} & -0.06201324271894954 \\ -5.558374381605137 \times 10^{-5} & 3.360336409319307 \times 10^{-7} & -0.001041960934645596 \\ 0.06247539600084091 & -0.0002363912340255103 & 1 \end{bmatrix}$$

单应矩阵 $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} 1.092193412373497 & 0.0496063276248266 & -10.27578277032915 \\ 0.1183348478786654 & 1.105703573283429 & 30.788485384414 \\ 0.0003086792234843952 & 0.0002597284650868214 & 1 \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} 0.9995022449081169 & -0.02228731205085075 & -0.02232796777996576 \\ 0.02327221396012316 & 0.9987218289448967 & 0.04486772165325688 \\ 0.02129944790433358 & -0.04536500975981288 & 0.9987433851638079 \end{bmatrix}$$

$$\mathbf{t} = \begin{bmatrix} -0.1210746871526397 \\ 0.372826522369065 \\ 0.9199680996366555 \end{bmatrix}$$

这里通过旋转矩阵 $\mathbf{R}$ 计算出四元数，然后与 TUM 数据集中 groundtruch 真实位姿之间误差进行比较：

首先将上述旋转矩阵 $\mathbf{R}$ 计算出四元数，得 [-0.02256674 -0.01091099 0.0113942   0.99962086]

转换代码：

```python
"""
@File    :   R2Q.py
@Time    :   2024/05/28 21:06:03
@Author  :   Xiang Lei
@Version :   1.0
@Desc    :   Rotation matrix to quaternion
"""

import numpy as np


def rotation_matrix_to_quaternion(R):
    """
    Convert a rotation matrix to a quaternion.
    """
    qw = np.sqrt(1 + R[0, 0] + R[1, 1] + R[2, 2]) / 2
    qx = (R[2, 1] - R[1, 2]) / (4 * qw)
    qy = (R[0, 2] - R[2, 0]) / (4 * qw)
    qz = (R[1, 0] - R[0, 1]) / (4 * qw)
    return np.array([qx, qy, qz, qw])


R1 = np.array(
    [
        [0.9995022449081169, -0.02228731205085075, -0.02232796777996576],
        [0.02327221396012316, 0.9987218289448967, 0.04486772165325688],
        [0.02129944790433358, -0.04536500975981288, 0.9987433851638079],
    ]
)

Q1 = rotation_matrix_to_quaternion(R1)
print(Q1)
```

在 TUM 数据集中 groundtruth.txt 中找到最接近时间的位姿数据，并从中提取出四元数。

下面是根据两张图片找到的对应数据：

1305031103.8758 1.1769 0.6236 1.4946 0.6579 0.6492 -0.2920 -0.2460

1305031104.2159 1.2636 0.6289 1.6161 0.6475 0.6422 -0.2963 -0.2838

时间点 2 - 时间点 1 真实位姿差（用四元数）：[0.02259146 0.03212753 -0.00783342 0.99924089]

计算代码如下：

```
"""
@File    :   compute.py
@Time    :   2024/05/28 21:22:02
@Author  :   Xiang Lei
@Version :   1.0
@Desc    :   使用两个时间点的四元数，计算两个时间点之间的旋转四元数
"""

import numpy as np


def quaternion_conjugate(q):
    x, y, z, w = q
    return np.array([-x, -y, -z, w])


def quaternion_multiply(q1, q2):
    x1, y1, z1, w1 = q1
    x2, y2, z2, w2 = q2
    w = w1 * w2 - x1 * x2 - y1 * y2 - z1 * z2
    x = w1 * x2 + x1 * w2 + y1 * z2 - z1 * y2
    y = w1 * y2 - x1 * z2 + y1 * w2 + z1 * x2
    z = w1 * z2 + x1 * y2 - y1 * x2 + z1 * w2
    return np.array([x, y, z, w])


q1 = np.array([0.6579, 0.6492, -0.2920, -0.2460])

q2 = np.array([0.6475, 0.6422, -0.2963, -0.2838])

q1_conjugate = quaternion_conjugate(q1)

q_delta = quaternion_multiply(q2, q1_conjugate)


print("q_delta (relative quaternion):")
print(q_delta)
```

与真实结果误差：

MAE: 0.006295502500000002
MSE: 0.00011574142801482499
Cosine Similarity: 0.9997685280586092

计算代码：

```
"""
@File    :   change.py
```

```
@Time    :   2024/05/28 21:35:27
@Author  :   Xiang Lei
@Version :   1.0
@Desc    :   计算两个四元数之间的差距，使用 MAS，MSE 和 余弦相似度
"""

import numpy as np

q_real = np.array([0.02259146, 0.03212753, -0.00783342, 0.99924089])
q_pred1 = np.array([0.02256674, 0.01091099, -0.0113942, 0.99962086])


def compute_MAE(q1, q2):
    return np.mean(np.abs(q1 - q2))


def compute_MSE(q1, q2):
    return np.mean((q1 - q2) ** 2)


def compute_cosine_similarity(q1, q2):
    return np.dot(q1, q2) / (np.linalg.norm(q1) * np.linalg.norm(q2))


def compute_metrics(q_real, q_pred):
    MAE = compute_MAE(q_real, q_pred)
    MSE = compute_MSE(q_real, q_pred)
    cosine_similarity = compute_cosine_similarity(q_real, q_pred)
    return MAE, MSE, cosine_similarity


if __name__ == "__main__":
    MAE, MSE, cosine_similarity = compute_metrics(q_real, q_pred1)
    print("MAE:", MAE)
    print("MSE:", MSE)
    print("Cosine Similarity:", cosine_similarity)
```

Q2:

利用3D-2D PnP方法，以第（学号%30+40）张图像对应的深度图（depth）作为3D位置，求解R，t；

```
./pose_estimation_3d2d /home/lei/桌面/SLAM/slambook2/TUM/rgb/1305031103.875361.png
/home/lei/桌面/SLAM/slambook2/TUM/rgb/1305031104.211283.png /home/lei/桌
面/SLAM/slambook2/TUM/depth/1305031103.862379.png /home/lei/桌
面/SLAM/slambook2/TUM/depth/1305031104.194053.png
```

$$\mathbf{R} = \begin{bmatrix} 0.9993271651632516 & -0.02380935075574116 & -0.02789859823305237 \\ 0.02578752878725596 & 0.9970105384684507 & 0.07283535914579099 \\ 0.02608103383352796 & -0.07350578888386956 & 0.9969536993635835 \end{bmatrix}$$

$$\mathbf{t} = \begin{bmatrix} -0.01196559472435167 \\ 0.0277391066945159 \\ 0.1391525033085017 \end{bmatrix}$$

旋转矩阵 $\mathbf{R}$ 转四元数，结果为：[-0.03661601 -0.01350624 0.01240963 0.99916107]

与真实结果误差：

MAE: 0.03129102750000001
MSE: 0.001499438228187175
Cosine Similarity: 0.9970012538137822

Q3：

利用3D-3D ICP方法，分别通过SVD和非线性优化方法求解R，t；

```
./pose_estimation_3d3d /home/lei/桌面/SLAM/slambook2/TUM/rgb/1305031103.875361.png
/home/lei/桌面/SLAM/slambook2/TUM/rgb/1305031104.211283.png /home/lei/桌
面/SLAM/slambook2/TUM/depth/1305031103.862379.png /home/lei/桌
面/SLAM/slambook2/TUM/depth/1305031104.194053.png
```

使用 SVD 求解

$$\mathbf{R} = \begin{bmatrix} 0.9999387804176818 & 0.00789985081120928 & -0.007747759286362108 \\ -0.008348647967931777 & 0.9981805381149579 & -0.05971526945129002 \\ 0.007261920813829925 & 0.05977629702225834 & 0.9981853829927585 \end{bmatrix}$$

$$\mathbf{t} = \begin{bmatrix} 0.04173843758094892 \\ -0.03333071746973436 \\ -0.08795078312375026 \end{bmatrix}$$

旋转矩阵 $\mathbf{R}$ 转四元数，结果为：[0.0298867 -0.00375415 0.004064   0.99953798]

与真实结果误差：

MAE: 0.013842857499999998
MSE: 0.000370588087851125
Cosine Similarity: 0.9992588567227915

使用非线性优化方法求解

$$\mathbf{R} = \begin{bmatrix} 0.9999387804377085 & 0.007899852155451604 & -0.007747755331085923 \\ -0.008348649070264622 & 0.9981805381312308 & -0.05971526902516696 \\ 0.007261916788970622 & 0.05977629657287662 & 0.9981853830489512 \end{bmatrix}$$

$$\mathbf{t} = \begin{bmatrix} 0.04173844677017662 \\ -0.03333071480842063 \\ -0.0879506755796868 \end{bmatrix}$$

旋转矩阵 $\mathbf{R}$ 转四元数，结果为：[2.98866996e-02 -3.75415252e-03 -1.12251091e-04 9.99537981e-01]

与真实结果误差：

MAE: 0.012798795507249994
MSE: 0.00035010509341776774
Cosine Similarity: 0.9992998155575764

Q4:

利用数据集中提供的四元数计算这两张图像之间的R和t；

在 TUM 数据集中 groundtruth.txt 中找到最接近时间的位姿数据，并从中提取出四元数。

下面是根据两张图片找到的对应数据：

1305031103.8758 1.1769 0.6236 1.4946 0.6579 0.6492 -0.2920 -0.2460

1305031104.2159 1.2636 0.6289 1.6161 0.6475 0.6422 -0.2963 -0.2838

$$\mathbf{R} = \begin{bmatrix} 0.99781311 & 0.01710509 & 0.06384684 \\ -0.01420211 & 0.99885663 & -0.04564802 \\ -0.06455465 & 0.04464143 & 0.99691516 \end{bmatrix}$$

$$\mathbf{t} = \begin{bmatrix} 0.0867 \\ 0.0053 \\ 0.1215 \end{bmatrix}$$

```python
import numpy as np

def read_groundtruth(file_path):
    with open(file_path, 'r') as f:
        lines = f.readlines()

    data = []
    for line in lines:
        if not line.startswith('#'):
            values = list(map(float, line.strip().split()))
            data.append(values)
    return np.array(data)

def quaternion_to_rotation_matrix(q):
    w, x, y, z = q
    R = np.array([
        [1 - 2 * (y**2 + z**2), 2 * (x*y - z*w), 2 * (x*z + y*w)],
        [2 * (x*y + z*w), 1 - 2 * (x**2 + z**2), 2 * (y*z - x*w)],
        [2 * (x*z - y*w), 2 * (y*z + x*w), 1 - 2 * (x**2 + y**2)]
    ])
    return R

def normalize_quaternion(q):
    norm = np.linalg.norm(q)
    if norm == 0:
        return q
    return q / norm

def quaternion_conjugate(q):
    w, x, y, z = q
    return np.array([w, -x, -y, -z])

def quaternion_multiply(q1, q2):
    w1, x1, y1, z1 = q1
    w2, x2, y2, z2 = q2
    w = w1 * w2 - x1 * x2 - y1 * y2 - z1 * z2
```

```
    x = w1 * x2 + x1 * w2 + y1 * z2 - z1 * y2
    y = w1 * y2 - x1 * z2 + y1 * w2 + z1 * x2
    z = w1 * z2 + x1 * y2 - y1 * x2 + z1 * w2
    return np.array([w, x, y, z])

def compute_R_and_t(data, idx1, idx2):
    t1, tx1, ty1, tz1, qx1, qy1, qz1, qw1 = data[idx1]
    t2, tx2, ty2, tz2, qx2, qy2, qz2, qw2 = data[idx2]
    print(data[idx1], data[idx2])
    # 四元数和位置
    q1 = normalize_quaternion([qw1, qx1, qy1, qz1])
    q2 = normalize_quaternion([qw2, qx2, qy2, qz2])

    # 计算旋转矩阵
    q1_conj = quaternion_conjugate(q1)
    q_r = quaternion_multiply(q2, q1_conj)
    R = quaternion_to_rotation_matrix(q_r)

    # 计算平移向量
    t1 = np.array([tx1, ty1, tz1])
    t2 = np.array([tx2, ty2, tz2])
    t = t2 - t1

    return R, t

file_path = '/home/lei/桌面/SLAM/slambook2/TUM/groundtruth.txt'

data = read_groundtruth(file_path)

# 2053932
idx1 = 521
idx2 = 555

R, t = compute_R_and_t(data, idx1, idx2)

print("R=", R)
print("t=", t)
```

Q5：

分析说明这几种方法中误差产生的可能原因。

在图像处理和位姿估计中，误差是不可避免的。误差的产生可能是由于特征检测和匹配的错误、相机标定的不准确、图像噪声、算法的近似处理以及数据稀疏性等原因。这些误差会导致计算得到的旋转矩阵 $\mathbf{R}$ 和平移向量 $\mathbf{t}$ 的精度下降，从而影响整体的定位和建图效果。

误差分析：

1. **特征匹配误差**：在特征匹配过程中，由于图像质量、视角变化、光照变化等原因，特征点可能匹配错误或存在误匹配。这会直接导致计算得到的位姿估计不准确，进而影响旋转矩阵 $\mathbf{R}$ 和平移向量 $\mathbf{t}$ 的精度。解决方法：使用 RANSAC 来减少匹配错误的情况。
2. **相机标定误差**：相机的内参（如焦距、主点坐标）和外参（如位姿）如果不准确，会导致计算过程中引入系统误差，从而影响最终的位姿估计结果。

3. **图像噪声**：图像中存在的噪声会干扰特征点的检测和匹配，使得检测到的特征点位置不准确，匹配结果不可靠，导致位姿估计出现偏差。
4. **算法近似**：在计算本质矩阵 $\mathbf{E}$、基础矩阵 $\mathbf{F}$、单应矩阵 $\mathbf{H}$ 以及分解旋转矩阵 $\mathbf{R}$ 和平移向量 $\mathbf{t}$ 的过程中，通常会使用一些近似方法或假设。这些近似和假设会引入计算误差。
5. **数据稀疏性**：在 3D-2D 和 3D-3D 配准过程中，如果特征点或匹配点较少，计算结果的稳定性和准确性会受到影响。特征点越稀疏，位姿估计的误差就越大。
6. **时间差误差**：数据集中原始图像、深度图像和位姿数据之间存在时间差，这会导致在不同时间点采集的数据无法完全对齐，从而引入误差。这种时间差可能会导致旋转矩阵 $R\mathbf{R}$ 和平移向量 $t\mathbf{t}$ 的计算结果不准确。

方法分析：

方法1：2D-2D 对极约束法

利用 2D-2D 对极约束计算本质矩阵 $\mathbf{E}$、基础矩阵 $\mathbf{F}$ 和单应矩阵 $\mathbf{H}$，进而分解得到旋转矩阵 $\mathbf{R}$ 和平移向量 $\mathbf{t}$。此方法对特征匹配的依赖较大，匹配误差和图像噪声会直接影响计算结果。同时，基础矩阵和本质矩阵的计算涉及多次矩阵运算，容易引入近似误差。

方法2：3D-2D PnP 方法

以深度图作为 3D 位置，通过 PnP 方法求解旋转矩阵 $\mathbf{R}$ 和平移向量 $\mathbf{t}$。此方法将 2D 图像点和 3D 世界点对应起来，能够提供更精确的位姿估计，但受相机标定误差和深度图精度影响较大。此外，PnP 算法中的优化过程也可能引入数值误差。

方法3：3D-3D ICP 方法

通过 SVD 和非线性优化方法进行 3D-3D ICP 对齐，求解旋转矩阵 $\mathbf{R}$ 和平移向量 $\mathbf{t}$。此方法直接使用点云数据，对匹配的准确性要求较高。SVD 方法计算简便但对噪声敏感，非线性优化方法虽然精度更高但计算复杂度较大，容易受到初值选择的影响。

方法4：利用四元数计算

通过 TUM 数据集中提供的四元数直接计算旋转矩阵 $\mathbf{R}$ 和平移向量 $\mathbf{t}$。此方法依赖于四元数的准确性和对齐精度。四元数的误差可能来源于传感器噪声和对齐误差。此外，从四元数转换到旋转矩阵的过程中也可能引入数值误差。