一：算法流程

1：读取数据

2：对点云进行降采样 downsample

3：iss 特征点提取

4：fpfh 特征点描述 feature description

5：RANSAC Registration,初配准、得到初始旋转、平移矩阵

（1） Establish correspondences(point pairs) 建立 pairs

（2） select 4 pairs at each iteration,选择 4 对 corresponding 进行模型拟合

（3）iter 迭代， iter_match() ,选择出 vaild T

6：ICP for refined estimation,优化配准 ICP 对初始 T 进行迭代优化。

二：关键函数展示：

1：svd 求解选准平移矩阵

```
def solve_procrustes_transf(P,Q):    # 求解平移旋转矩阵 solve_procrustes_transformation
    up = P.mean(axis=0)
    uq = Q.mean(axis=0)

    # move to center:
    P_centered = P - up
    Q_centered = Q - uq

    U, s, V = np.linalg.svd(np.dot(Q_centered.T, P_centered), full_matrices=True, compute_uv=True)
    R = np.dot(U, V)
    t = uq - np.dot(R, up)

    # format as transform:
    T = np.zeros((4, 4))
    T[0:3, 0:3] = R
    T[0:3, 3] = t
    T[3, 3] = 1.0

    return T
```

2：ransac 初配准

```python
def ransac_match(
        idx_target,idx_source,
        pcd_source, pcd_target,
        feature_source, feature_target,
        ransac_params, checker_params
):
    #step5.1 Establish correspondences(point pairs) 建立 pairs
    matches = get_init_matches(feature_source, feature_target)   #通过 fpfh 建立的feature squre map 建立最初的 pairs

    ##build search tree on the target:
    search_tree_target = o3d.geometry.KDTreeFlann(pcd_target)

    N, _ = matches.shape
    idx_matches = np.arange(N)   #对每队 pair 打标签

    T = None                    #translation martix
    #step5.2 select 4 pairs at each iteration,选择4对corresponding 进行模型拟合
    proposal_generator = (
        matches[np.random.choice(idx_matches, ransac_params.num_samples, replace=False)] for _ in iter(int, 1)
    )
    ##step5.3 iter 迭代， iter_match() ,选择出 vaild T
    validator = lambda proposal: iter_match(idx_target,idx_source,pcd_source, pcd_target, proposal, checker_params)

    with concurrent.futures.ThreadPoolExecutor(max_workers=ransac_params.max_workers) as executor:
        for T in map(
                validator,
                proposal_generator        #map()是 Python 内置的高阶函数，它接收一个函数 f 和一个 list，并通过把函数 f 依次作用在 list 的每个元素上，得到一个新的 list 并返回。
        ):
            print(T)
            if not (T is None):
                break

    #set baseline
    print('[RANSAC ICP]: Get first valid proposal. Start registration...')
    best_result = icp_exact_match(
        pcd_source, pcd_target, search_tree_target,
        T,
        ransac_params.max_correspondence_distance,
        ransac_params.max_refinement
    )
```

```python
    # RANSAC:
    num_validation = 0
    for i in range(ransac_params.max_iteration):
        # get proposal:
        T = validator(next(proposal_generator))

        # check validity:
        if (not (T is None)) and (num_validation < ransac_params.max_validation):
            num_validation += 1

            # refine estimation on all keypoints:
            result = icp_exact_match(
                pcd_source, pcd_target, search_tree_target,
                T,
                ransac_params.max_correspondence_distance,
                ransac_params.max_refinement
            )

            # update best result:
            best_result = best_result if best_result.fitness > result.fitness else result

            if num_validation == ransac_params.max_validation:
                break

    return best_result
```

3：ICP 配准优化

```python
def icp_exact_match(
        pcd_source, pcd_target, search_tree_target,
        T,
        max_correspondence_distance, max_iteration
):
    # num. points in the source:
    N = len(pcd_source.points)

    # evaluate relative change for early stopping:
    result_prev = result_curr = o3d.pipelines.registration.evaluate_registration(
        pcd_source, pcd_target, max_correspondence_distance, T
    )

    for _ in range(max_iteration):
        # TODO: transform is actually an in-place operation. deep copy first otherwise the result will be WRONG
        pcd_source_current = copy.deepcopy(pcd_source)
        # apply transform:
        pcd_source_current = pcd_source_current.transform(T)

        # find correspondence:
        matches = []
        for n in range(N):
            query = np.asarray(pcd_source_current.points)[n]
            _, idx_nn_target, dis_nn_target = search_tree_target.search_knn_vector_3d(query, 1)

            if dis_nn_target[0] <= max_correspondence_distance:
                matches.append(
                    [n, idx_nn_target[0]]
                )
        matches = np.asarray(matches)

        #icp
        if len(matches) >= 4:
            # sovle ICP:
            P = np.asarray(pcd_source.points)[matches[:, 0]]
            Q = np.asarray(pcd_target.points)[matches[:, 1]]
            T = solve_procrustes_transf(P, Q)

            # evaluate:
            result_curr = o3d.pipelines.registration.evaluate_registration(
                pcd_source, pcd_target, max_correspondence_distance, T
            )

            # if no significant improvement:提前中止
            if shall_terminate(result_curr, result_prev):
                print('[RANSAC ICP]: Early stopping.')
                break

    return result_curr
```