## Abstract

为GICP增加约束，并且去除LAB的L分量，减少光照影响 ## 1. Introduction

## 2. Related Work

## 3. Method

### pixel to point cloud

最基本的图像变换点云，对于RGB图像只是深度值变成了 Variables: - : Pixel coordinates in the image plane. - : Coordinates of the principal point (optical center) in the image, typically given in pixels. - : Focal lengths of the camera in the and directions, respectively, typically given in pixels. - : Depth value at the pixel after scaling, given in meters. - : Depth value from the *original depth image* at pixel , typically given in the units used by the depth sensor. - : Scale factor to convert the depth values from the depth image units to meters. - : 3D point in *camera coordinates*, represented as a vector or

### grid down sample for depth image 优点: 不会引入噪声也能够加速点云转换，因为直接转换点云计算量很大，频繁的创建数据 *随机降采样是个彻头彻尾的反例，因为引入了噪声，但这种规整的降采样就不会* Variables: - : Pixel indices in the depth image. - : Original depth image. - : Scale factor to convert depth values from units to meters. - : Intrinsic camera matrix. - : Stride used for down sampling the image and depth values. 1. **Generate pixel indices for the entire depth image:**

1. **Downsampling by taking every nth pixel determined by the downsample\_stride:**
2. **Scale depth values to meters:**
3. **Calculate 3D coordinates in the camera coordinate system using the intrinsic parameters:**

* coordinate:
* coordinate:

1. **Concatenate coordinates to form the 3D points with a homogeneous coordinate:**

* The final array, points, is reshaped to ((-1, 4)) to flatten the point cloud into a two-dimensional array where each row represents a 3D point in homogeneous coordinates. ### point cloud rejector via image filter

1. 双边滤波（Bilateral Filter）：既保证能够出去离群值也能够保证不太影响图像边缘

import cv2  
image = cv2.imread('path\_to\_image')  
filtered\_image = cv2.bilateralFilter(image, d=9, sigmaColor=75, sigmaSpace=75)

其中，d 是领域直径，sigmaColor 控制颜色的高斯函数标准差，sigmaSpace 控制空间的高斯函数标准差。 2. 中值滤波: 但可能对边缘部分的离群值处理不太好，并且是用于减少噪声，特别是“椒盐”类型的噪声，但不清楚会不会引入

cv::Mat depthImage; // 假设这是你的深度图  
cv::medianBlur(depthImage, depthImage, 5); // 使用5x5的核进行滤波

没有太多的原创性，主要是应用层面迁移，从处理经典图像到深度图的适用，避免在pcd层面的rejector效率不高，毕竟是三维的，能在二维解决为什么要三维呢？并且也可以解决一小部分像素避免进行

### ABGICP

**核心算法** 1. 去除L分量，给定光源的情况下，尤其是室外，在不同角度下，同一个物体表面的彩色值是不一样的，可能会变成错误的噪音提供了错误的配准信息，使得算法收敛速度慢，或者求解不正确等等 2. 现有方法[1] 采用的是对LAB数值使用一个比例参数，协调LAB分量对于整个优化过程提供的影响程度，有点机械化，该论文使用了一系列的比例对齐技术，**椭圆正则化**[2]

gicp部分还没有进行比例对齐 #### BGR->LAB *RGB->LAB转换过程包括了颜色空间的转换和非线性的调整，以适应人眼对亮度的非线性感知*

import cv2  
# Load an image in BGR color space  
image = cv2.imread('path\_to\_your\_image.png')  
lab\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2LAB)

#### GICP

Math: 1. points

1. Extended points including the AB component
2. Assuming the error is normally distributed, focusing only on the spatial and AB components:
3. Define the loss function using Mahalanobis distance for spatial and AB components:
4. Objective function and MLE are then defined as:

## 4. Experimental Setup

### preprocess

#### grid down sample for depth image

**experiment data** - 原生数据集，不修改的

**evaluation** 1. 单纯[Point cloud - Open3D 0.18.0 documentation](https://www.open3d.org/docs/release/tutorial/geometry/pointcloud.html#Voxel-downsampling)Voxel-downsampling 2. 单纯image grid down sample 3. 混合两者

1. 配准都采用纯GICP算法情况下，*三者* 都可以比较 *配准精度*
2. 前两者在相同点云数情况下，应该是后者效率更高比较他们的*运算速度*，
3. 不出意外，实验证明，应该是混合两者运算 *效率和精度最高*

#### point cloud rejector via image filter

**experiment data** - 需要手动生成离群值，在干净数据集上面进行模拟因为可以验证离群处理的效果

[3] 该论文也是RGB-D相机但是直接在点云层面上进行预处理

import pclpy  
from pclpy import pcl  
  
def pc\_filter(pointcloud, cloud\_filtered):  
 # Create temporary PointClouds  
 temp = pcl.PointCloud.PointXYZRGBA()  
 temp2 = pcl.PointCloud.PointXYZRGBA()  
  
 # Distance based filter  
 passthrough = pcl.filters.PassThrough[pcl.PointXYZRGBA]()  
 passthrough.setInputCloud(pointcloud)  
 passthrough.setFilterFieldName("z")  
 passthrough.setFilterLimits(0.0, 3.0)  
 passthrough.filter(temp)  
  
 # Voxel grid filter  
 voxel\_filter = pcl.filters.VoxelGrid[pcl.PointXYZRGBA]()  
 voxel\_filter.setInputCloud(temp)  
 voxel\_filter.setLeafSize(0.005, 0.005, 0.005)  
 voxel\_filter.filter(temp2)  
  
 # Radius outlier removal  
 out\_rem = pcl.filters.RadiusOutlierRemoval[pcl.PointXYZRGBA]()  
 out\_rem.setInputCloud(temp2)  
 out\_rem.setRadiusSearch(0.01)  
 out\_rem.setMinNeighborsInRadius(10)  
 out\_rem.filter(cloud\_filtered)  
  
# Example usage  
pointcloud = pcl.PointCloud.PointXYZRGBA()  
cloud\_filtered = pcl.PointCloud.PointXYZRGBA()  
pc\_filter(pointcloud, cloud\_filtered)

**evaluation** - 该滤波方法生成点云时候和标准rejector生成的点云进行比较 - 同时和干净点云进行配准，评估 *配准精度*和 *运算帧率* 的影响

### registration

**experiment data** - 偶数个RGB图像，按比例放大数值，代表光线增强 **evaluation** 1. [Colored point cloud registration - Open3D 0.18.0 documentation](https://www.open3d.org/docs/release/tutorial/pipelines/colored_pointcloud_registration.html)和open3d的colored\_pointcloud\_registration方法系列进行比较精度，实现的是该论文的方法[4]，是RGB空间 2. [Point Cloud Library (PCL): pcl::GeneralizedIterativeClosestPoint6D Class Reference](https://pointclouds.org/documentation/classpcl_1_1_generalized_iterative_closest_point6_d.html)这应该是*对标的方案*[1],是这篇文章的实现，LAB空间额外的约束GICP算法 *从配准精度的角度计算时间角度考虑* ## 5. Results & Discussion

[1] M. Korn, M. Holzkothen, and J. Pauli, “Color supported generalized-ICP,” in *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, IEEE, 2014, pp. 592–599.

[2] S. Ha, J. Yeon, and H. Yu, “RGBD GS-ICP SLAM.” arXiv, Mar. 2024. Accessed: May 23, 2024. [Online]. Available: <https://arxiv.org/abs/2403.12550>

[3] H. Gupta, A. J. Lilienthal, H. Andreasson, and P. Kurtser, “NDT-6D for color registration in agri-robotic applications,” *Journal of Field Robotics*, vol. 40, no. 6, pp. 1603–1619, Sep. 2023, doi: [10.1002/rob.22194](https://doi.org/10.1002/rob.22194).

[4] J. Park, Q.-Y. Zhou, and V. Koltun, “Colored point cloud registration revisited,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 143–152.