

# Structure From Motion结题报告

唐子钰 2020212271 刘梓航 2020212297 马天驰 2020212299

## 基于增量法的 sfm

### 摘要

三维重建是指从二维图像中恢复物体或场景的三维模型的过程，是计算机视觉领域的一个重要研究方向。自动驾驶和增强现实等领域近年来取得了快速的进展，它们都需要对周围环境进行精确的三维重建，通常是通过结合激光雷达、结构光和其他特定传感器的深度数据来实现的。这些传感器的缺点是它们需要特殊的硬件，相比于只依赖于 RGB 相机系统的方法，它们虽然更有效，但也更复杂且难以获取和使用。三维重建有多种方法，其中一种是基于单张图像的增量式结构从运动（SFM）方法，即利用单张图像中的特征点和相机运动信息来逐步重建稀疏的点云和相机位姿。

本文介绍了一个简单版本的增量式 SFM 方法，并且嵌入实现了可视化的 PyQt 界面中，其实现了离线方式下稀疏点云的重建和位姿的计算，算法提供可调节参数，且算法精简各模块独立，可拓展性较强。其主要步骤如下：（1）提取特征：对输入图像进行特征提取和匹配，得到特征点对应关系；（2）视图对寻找：基于有序的方法寻找最优视图对，自动跳过匹配点稀疏的视图；（3）初始化 sfm：通过视图 0 寻找新视图，利用匹配点和相机内参估计本质矩阵，剔除错误匹配；将本质矩阵分解为旋转矩阵和平移向量，得到初始点云；（4）PnP 计算位姿：利用已重建的三维点和新视图的匹配点，求解新视图的相机姿态；（5）点云融合：利用三角量测方法，从两个投影矩阵和匹配点中重建新的三维点，并与已有点云融合；（6）BA 捆绑调整：对所有相机姿态和三维点进行非线性优化，最小化重投影误差。

本文实现了一个基于 OpenCV 和 PyQt 的增量式 SFM 系统，并在一些公开数据集上进行了实验，验证了其有效性和可行性。

### — Introduction

三维重建指对三维物体建立适合计算机表示和处理的**数学模型**，是在计算机环境下对其进行处理、操作和分析其性质的基础，也是在计算机中建立表达客观世界的虚拟现实的关键技术。

在计算机视觉中，获取场景的三维表示是一个具有挑战性的任务，主要原因在于二维图像只提供了有关三维世界的部分信息，即缺失深度信息；而多视图可以提供这一信息。多视图三维重建任务的目的用相机拍摄真实世界的物体、场景，并通过计算机视觉技术进行处理，从而得到物体的三维模型。

目前三维重建的工作主要分为两大类：基于传统多视图几何的传统方法和基于深度学习的方法。

### 传统方法

基于多视图几何的传统方法是指利用二维投影或影像恢复物体三维信息。这类方法按传感器是否主动向物体照射光源可以分为主动式和被动式两种。主动式方法需要使用特殊的设备，如激光扫描仪、结构光仪等，来获取物体表面的深度信息。被动式方法则只需要使用普通的相机，通过多张不同角度的

照片来重建物体的三维结构。被动式方法一般来说就是基于特征点的方法：先提取照片中的特征点，然后通过匹配、三角化、光束发平差等步骤来估计相机姿态和物体形状。

而传统的三维重建 pipeline 包括从相机标定、基础矩阵与本质矩阵估计、特征匹配到运动恢复结构 (SFM)；再从 SFM 到稠密点云重建 MVS、表面重建、纹理贴图等。

## 深度学习方法

基于深度学习的方法是指利用大量的训练数据（例如影像以及相对应的三维形状）来训练深度神经网络或卷积神经网络建构的模型，达到由单张或多张二维影像作为输入，推测三维形状。原理是利用深度神经网络从图像中提取特征，然后将这些特征映射到三维几何表示。这类方法可以使用不同类型的二维数据作为输入，如 RGB 影像、深度图等。也可以使用不同数量的输入影像，如单张或多张。深度学习的方法可以使用不同类型的数据来表示三维形状，如体素、点云等。

## sfm 基本流程

- 提取特征点并进行匹配
- 选择一组场景（两张图片），基于这两张图片估计相机位姿，并重建三维坐标点；
- 用光束发平差进行优化
- 对于剩下每一个的场景（图片），重复 2-3 步进行三维重建

## sfm 相关工作

### Incremental SFM

Incremental SFM 是一种从一组未标定的图像中逐步地估计相机姿态和场景几何的方法，基于逐步增量的方法更加鲁棒，因为能保证每张增量图像位姿和场景点的正确性，但是也有可能造成误差累积，导致相机轨迹飘移。

一些具有代表性的传统算法有：

- Pollefeys 等人提出了一种自标定的增量 SFM 方法，可以在不知道相机内参的情况下，从两张或多张图像中恢复出度量结构和运动。
- Snavely 等人提出了完全开源且稳定的 Bundler\_sfm，一个开源的增量 SFM 系统，可以从互联网上收集的大量照片中重建出三维场景。
- Wu 提出了 VisualSfM，一个基于 GPU 加速的增量 SFM 系统，可以在线性时间内处理大规模的图像集合。
- Schonberger 等人提出了 COLMAP，一个集成了增量和全局 SFM 方法的系统，可以从稀疏或稠密的图像集合中重建出高质量的三维模型。

### Global SFM

Global SFM 一次性地估计所有图像的相机姿态和场景几何，而不是逐步地添加新的图像。它通常先求得所有相机的旋转，然后再求得所有相机的平移和三维点。只需一次 BA，因此效率较高，但是其鲁棒性差，很容易受到错误匹配点的影响而导致重建失败。

代表性的传统算法有：

- Triggs 等人提出了一种基于因子分解的全局 SFM 方法，可以从正交投影下的多个视图中恢复结构和运动。
- Martinec 和 Pajdla 提出了一种基于  $L^\infty$  范数优化的全局 SFM 方法，可以从点匹配中估计全局运动。
- Jiang 等人提出了一种基于相似度平均的全局 SFM 方法，可以从无序图像中恢复大规模场景。

- Wilson 和 Snavely 提出了一种基于 1DSfM 的全局 SFM 方法，可以从特征轨迹中估计全局平移。

### hybrid SFM

hybrid SFM 结合了增量式和全局式 SFM 方法的优点，先对图像集合进行分割，然后对每个子集采用增量式或全局式 SFM 方法进行重建，并最终将所有子集进行融合。它可以处理大规模数据集，并且避免了误差累积和漂移问题。

一些具有代表性的传统算法：

- Havlena 等人提出了一种基于层次聚类树的混合 SFM 方法，可以从无标定图像中恢复层次结构和运动。
- Heinly 等人提出了一种基于多阶段优化的混合 SFM 方法，可以从粗到细地进行三维重建。
- Zhu 等人提出了一种基于 HSfM (Hybrid Structure-from-Motion) 框架的混合 SFM 方法，可以从局部增量到全局平均地进行三维重建。

### hierarchical SFM

hierarchical SFM 层次 SFM 先对原数据集进行划分，获得若干相互关联的子图，然后对每个子图进行并行的增量式或全局式 SFM 处理，并最终将所有子图进行合并。它可以提升大规模数据的处理效率，但是其鲁棒性较差，很容易受到子图划分、锚节点选择等因素的影响而导致重建失败。

一些具有代表性的传统算法有：

- Crandall 等人提出了一种基于图划分和并行优化的层次式 SFM 方法，可以从大规模互联网照片中重建结构和运动。
- Ni 等人提出了一种基于改进效率和鲁棒性的层次式 SFM 方法，可以从无标定图像中恢复结构和运动。
- Moulon 等人提出了一种基于正交最小生成树 (OMST) 和渐进式 BA (PBA) 的层次式 SFM 方法，可以从大规模无序图像中重建结构和运动。

## 二 进展

本次课程设计主要被拆分为 5 个板块的内容，即：

1. 前期技术路线调研：调研 sfm 相关技术路线，了解 sfm 各个分支领域的具体工作，以便选择合适的 baseline 实现。
2. 相关原理学习：待学习的具体原理主要包括如针孔相机模型、畸变模型、相机标定、对极几何、三角测量、SFM 系统设计，以及一些基本的线性代数、非线性优化数学知识比如 svd、非线性最小二乘优化、cholesky 分解、光束法平差等等。
3. sfm baseline 流程设计：设计一个 baseline 或者算法框架，包括 sfm 主要的系统设计和实现流程，寻找多视图数据集以评估。
4. 核心代码设计：完成整体的代码设计，要求实现 sfm 的基本功能：输入多视图图像重建相机位姿和场景三维点，要求达到稀疏重建的效果。
5. gui 界面设计：设计一个简单的 gui 界面，实现用户友好的数据集选择、参数选择，与后端连接正确运行设计的 sfm 核心算法，得到重建的稀疏点云以及位姿。

截至目前结题时间段，小组成员已成功所有技术路线，即包括前期技术路线的调研，相关原理的学习，sfm baseline 的设计，并实现核心代码和 gui 界面设计；

下面给出项目中期-结题的时间甘特图：



## 三 原理介绍

相机相关原理：

1. 基于小孔模型的相机成像模型；
2. 单目标定、张正友标定法、单应性；
3. 对极几何、本质矩阵 E、基础矩阵 F、位姿 RT；
4. 三角测量
5. PnP

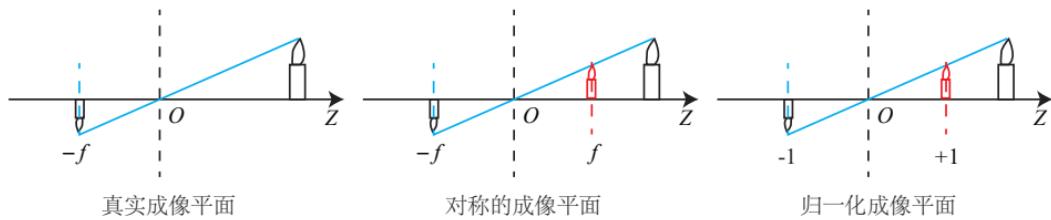
相关数学知识：齐次坐标、归一化相机平面坐标、旋转平移和坐标系变换、投影变换、畸变模型、单应性矩阵、cholesky 分解、超定方程的最小二乘解、svd、极大似然估计、LM 算法、对极几何、本质矩阵分解。

其他：sift 特征、pyqt 可视化

### 3.1 基于小孔模型的相机成像模型

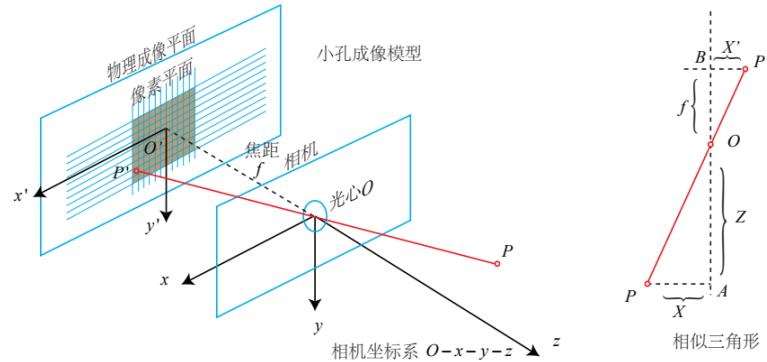
相机将三维世界中的坐标点（单位为米）映射到二维图像平面（单位为像素）的过程能够用一个几何模型进行描述。这个模型有很多种，其中最简单的称为针孔模型。针孔模型是很常用，而且有效的模型，它描述了一束光线通过针孔之后，在针孔背面投影成像的关系。我们用一个简单的针孔相机模型来对这种映射关系进行建模。同时，由于相机镜头上的透镜的存在，会使得光线投影到成像平面的过程中会产生畸变。因此，我们使用针孔和畸变两个模型来描述整个投影过程。

相机可以将三维空间中的坐标点映射到二维像素平面上，这个过程可以用一个几何模型来描述。在相机 area，最简单的描述方式被称为针孔模型。其描述了一束光线通过针孔后，在针孔背面投影成像的关系，如图，相机的软件算法可以自动翻转倒像，因此可以使用对称的正像来表示：



相机的坐标转换我定义为四个过程：

①世界坐标 $P_w \rightarrow$ ②相机坐标 $P_c \rightarrow$ ③图像坐标 $P_p \rightarrow$ ④像素坐标 $P_{pix}$



过程和中间推导略过，不考虑畸变等，给出针孔相机模型：

$$ZP_{uv} = Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K(RP_w + t) = KTP_w$$

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

$$f_x = fs_x, f_y = fs_y$$

其中  $K$ 、 $(R, T)$  分别为内外参， $P_{uv}$  和  $P_w$  为齐次像素坐标和世界坐标， $R, t$  被称为外部参数， $c_x, c_y$  即像素坐标系的中心点（过主光轴）。

### 3.2 单目标定

在给定 SfM 任务中，假设已经求解出内参系数  $K$ ，若未求出，需要在 SfM 过程中利用一些特殊优化算法通常是最小化 reprojection error 来不断求解优化  $K$ 。

单目标定是指利用一台摄像机和已知的标定板进行标定，求解出摄像机的内部参数（如焦距、主点坐标等）和外部参数（如旋转矩阵、平移向量等），从而建立摄像机的模型。标定板通常是一个已知尺寸的棋盘格，通过拍摄不同位姿下的标定板图像，可以得到摄像机的内外参数。

通常使用的方法是张正友标定法，通过确定的棋盘，构造棋盘角点和图像平面的单应性矩阵和坐标投影关系进行求解。

篇幅原因，这里仅列出关键公式：

1. 建立标定板图像平面投影模型：

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

1. 求解单应矩阵  $H$

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = H \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}, H = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} = \lambda K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix}$$

1. 求解内参 K:

建立 K 的约束等式:

$$\begin{cases} h_1^T K^{-T} K^{-1} h_2 = 0 \\ h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2 = 1 \end{cases}$$

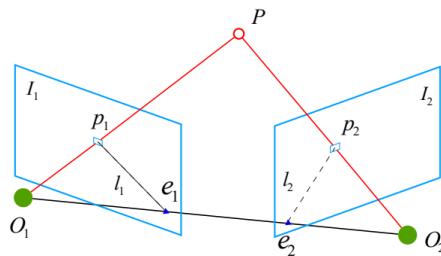
先求解  $K^{-T} K^{-1}$  再一一对应分解得到 K, 然后自然可以计算外参  $r_1, r_2$  和 T。

1. 最后再用非线性最小二乘法对所求参数进行优化。

### 3.3 对极几何

对极几何建模了两幅图像匹配点的关系, 通常用本质矩阵 E 和基础矩阵 F 来表示。

如图连线  $O_1 p_1$  和连线  $O_2 p_2$  在三维空间中会相交于点 P。这时候点  $O_1, O_2, P$  三个点可以确定一个平面, 称为极平面 (Epipolar plane)。 $O_1 O_2$  连线与像平面  $I_1, I_2$  的交点分别为  $e_1, e_2$ 。 $e_1, e_2$ , 称为极点 (Epipoles),  $O_1 O_2$  被称为基线 (Baseline)。称极平面与两个像平面  $I_1, I_2$  之间的相交线  $I_1, I_2$  为极线 (Epipolar line)。



具体来说, 对于匹配点  $p_1$  和  $p_2$ , 由关系:

$$s_1 p_1 = K P, \quad s_2 p_2 = K(RP + t)$$

可以推导出:

$$p_2^T K^{-T} t^\wedge R K^{-1} p_1 = 0$$

令其中本质矩阵 (Essential Matrix)  $E = t^\wedge R$ , 基础矩阵 (Fundamental Matrix)  $F = K^{-T} t^\wedge R K^{-1}$ , 对极约束给出了两个匹配点的空间位置关系, E 描述了空间两个相机的位姿关系, F 中包含了相机内参信息

用八点法求解本质矩阵 E (尺度等价) :

$$(u_1, v_1, 1) \begin{pmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{pmatrix} \begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} = 0$$

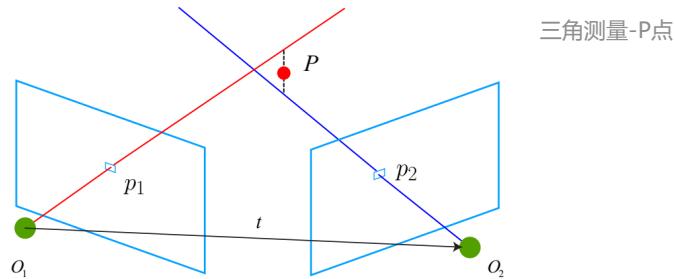
再基于反对称矩阵的块分解方法、E 的尺度和正负等价性, 对 E 进行 SVD 分解可得到位姿 R 和 T。

注意: 求解的位姿 t 和 R 由于 E 的尺度等价或者说样本未包含深度信息, 因此只能求解与尺度无关的相对位置关系约束, 并不能描述两个相机/图像之间的绝对位置关系, 即 t 的自由度仅为 2, E 的自由度为 5。

### 3.4 三角测量

三角测量是指，通过在两处观察同一个点的夹角，确定该点的距离。

在 sfm 增量法中，我们使用三角测量重建稀疏点云。



设  $x_1, x_2$  为两个特征点的归一化坐标，我们现在已知两相机的内参和位姿，那么它们满足：

$$s_2 \mathbf{x}_2 = s_1 \mathbf{R} \mathbf{x}_1 + \mathbf{t}$$

可以左乘  $\mathbf{x}_2$  的反对称矩阵得到：

$$s_2 \mathbf{x}_2^\wedge \mathbf{x}_2 = 0 = s_1 \mathbf{x}_2^\wedge \mathbf{R} \mathbf{x}_1 + \mathbf{x}_2^\wedge \mathbf{t}$$

构建了单独的关于  $s_1$  的方程，可以直接求得  $x_1$  的深度，同理可以求解  $x_2$  的深度。

或者可以由相机模型构建关于三维点 P 的线性方程组，显然自由度为 3，而两个匹配点每个点的 u、v 可以构建两个线性方程组，总共可以构建四个，然后求得最小二乘解。

### 3.5 PnP 问题

在 sfm 过程中，直接使用 PnP 来估计相机运动。

定义为已知 3D 点和像素点求解位姿  $R, T$ ，它描述了当我们知道 n 个 3D 空间点以及它们的投影位置时，如何估计相机所在的位姿。

由于已知具体点的绝对位置信息即深度信息，因此求得的  $T$  也是绝对的，对比本质矩阵的尺度等价性。

具体来说，PnP 问题有三种求解方法：

1. 直接线性变换 DLT：直接构建关于三维点的线性方程组，然后求方程组的最小二乘解。
2. 三对点估计法 P3P：仅使用三对匹配点将问题转化为 ICP 问题，即 3D-3D 点的匹配问题。
3. 非线性优化法 Bundle adjustment：把 PnP 问题构建成一个定义于李代数上的非线性最小二乘问题，是一种非线性优化算法。前面的线性算法通常是先求解相机位姿，再求解三维点，然后再求解位姿。而非线性优化将位姿、三维点都看作优化变量，最小化重投影误差放在一起优化。

且该方法通常放在线性求解初值后对结果进行进一步的优化。

这里仅介绍 DLT 方法：

将世界坐标变换为齐次坐标，定义增广矩阵  $[R|t]$  为一个  $3 \times 4$  的矩阵得到：

$$s \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = \begin{pmatrix} t_1 & t_2 & t_3 & t_4 \\ t_5 & t_6 & t_7 & t_8 \\ t_9 & t_{10} & t_{11} & t_{12} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

左侧是归一化的相机坐标，因此每个三维点可以得到两个约束：

$$u_1 = \frac{t_1X + t_2Y + t_3Z + t_4}{t_9X + t_{10}Y + t_{11}Z + t_{12}} \quad v_1 = \frac{t_5X + t_6Y + t_7Z + t_8}{t_9X + t_{10}Y + t_{11}Z + t_{12}}$$

从而得到两个等式约束方程：

$$\begin{aligned}\mathbf{t}_1^T \mathbf{P} - \mathbf{t}_3^T \mathbf{P} u_1 &= 0 \\ \mathbf{t}_2^T \mathbf{P} - \mathbf{t}_3^T \mathbf{P} v_1 &= 0\end{aligned}$$

因此至少六对点可以求解出位姿 R 和 T。

当 R 不满足正交矩阵时，可以对 R 进行 QR 分解用一个最好的旋转矩阵去近似 R。

### 3.6 sift

对于 sfm 过程特征点的提取和匹配，选择鲁棒性、稳定性较好的尺度不变特征 SIFT。

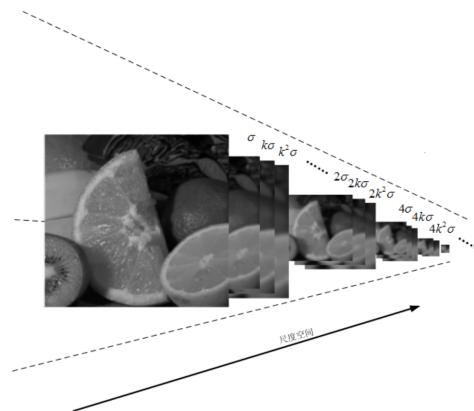
尺度不变特征转换(SIFT, Scale Invariant Feature Transform)是图像处理领域中的一种局部特征描述算法，sift 的实质就是在不同的尺度空间上寻找关键点，并计算出关键点的方向。

简单来说 sift 大概可以分为下面几个步骤：

- 尺度关键点检测：**搜索所有尺度图像，利用高斯微分函数来识别潜在的对于尺度和旋转不变的兴趣点，比如角点、边缘点等。
- 关键点定位：**在每个候选的位置上，通过一个拟合精细的模型来确定精确的特征点位置和尺度。
- 方向确定：**基于图像局部的梯度方向，分配给每个关键点位置一个或多个方向。所有后面的对图像数据的操作都相对于关键点的方向、尺度和位置进行变换，从而提供对于这些变换的不变性。
- 生成关键点描述符：**在关键点对应尺度的图像邻域内，选定多个不同的方向统计测量局部图像梯度的幅值作为关键点的描述符。

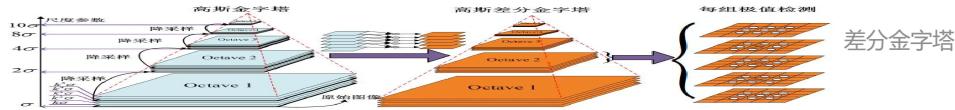
#### 3.6.1 尺度描述

sift 的尺度空间用高斯金字塔描述，如图：



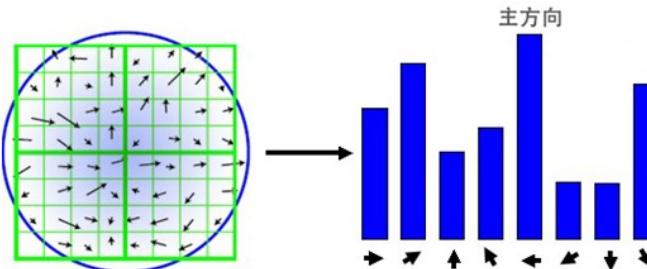
#### 3.6.2 关键点测量

sift 从 DOG 图中寻找极值点作为粗略的关键点，然后进一步利用泰勒展开求解亚像素精度的极值点。



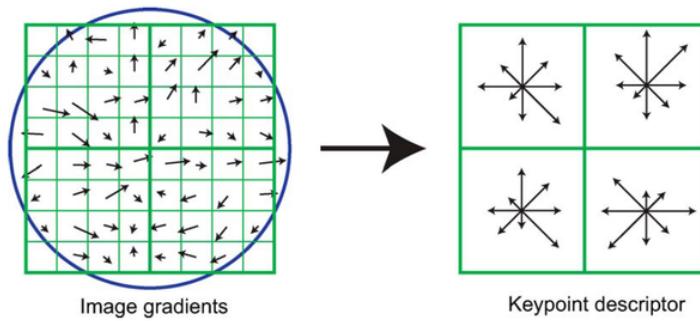
### 3.6.3 关键点方向

sift 提取关键点邻域的尺度图像梯度方向直方图，其中峰值最大的作为关键点的主方向：



### 3.6.4 关键点描述

最后 sift 根据关键点位置、尺度和方向，将尺度图像分块为  $4 \times 4$  个子区域，统计每个子区域 8 个方向的梯度高斯加权幅值作为关键点的描述符，然后再进行归一化去除光照的影响等。



综上可以得到 sift 提取的 128 维特征向量，sift 是一种稳定的具有较为的旋转、尺度、光照不变性的特征描述子，其也能在一定程度上不受视角、仿射变换和噪声的干扰。

### 3.6.5 sift 缺点

1. 特征点数量敏感，有时较少
2. 对边缘光滑的目标无法准确提取特征点，如图对模糊图像、边缘平滑图像、圆无能为力。

## 四 系统设计

### 4.1 sfm baseline 设计

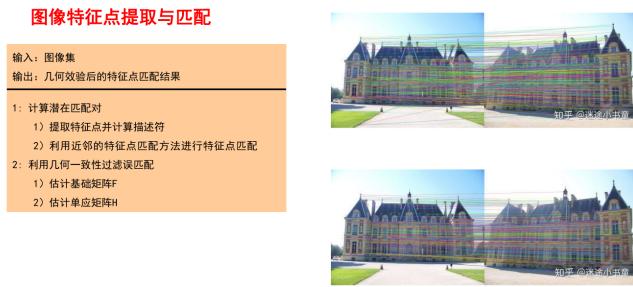
经过调研，sfm 中有多种基础的 baseline，我们选择实现基本的增量法 SFM。

下面介绍基本的系统设计思路即可能包括的模块，并详细介绍。

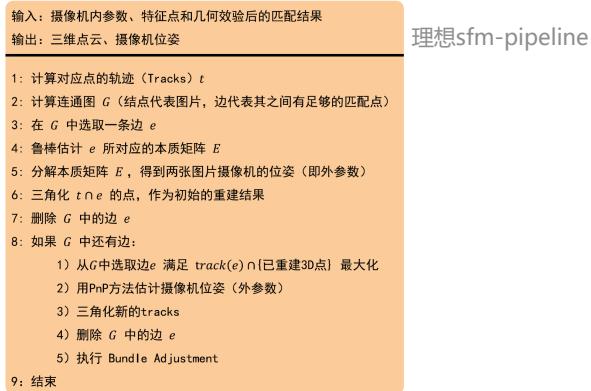
#### 4.1.1 数据预处理

数据预处理主要包括读入图像数据集，对输入图像进行 degradation，统一分辨率，然后提取图像特征点。sfm 并不要求实时性，所以通常使用可靠稳健的 SIFT (Scale-Invariant Feature Transform) 特征点，对应的描述子也称为 sift 描述子。

## 预处理



### 4.1.2 增量法 sfm



如图为增量法 sfm 标准 pipeline 的算法步骤，具体来说可以总结为下面几个步骤：

#### 1. 匹配图像对搜索

匹配图像对搜索指视图添加的逻辑，即确定视图重建的顺序，常见的方法如：

- 穷举法：对所有图像进行两两匹配
- 词汇树(Vocabulary Tree)：在这种匹配模式下，每幅图像都会通过空间重新排序的词汇树与其视觉近邻进行匹配。与视觉词袋相比，词汇树利用层次 kmeans 得到尽可能多的特征。层次 kmeans 的意思是将空间中的点用普通 kmeans 得到 k 个聚类中心，每个点都属于一个聚类中心代表的类。对于一类中的所有点，再递归的进行 kmeans，直到获得我们想要的聚类中心个数。这样的聚类方法相比普通 kmeans 聚类可以得到更多的聚类中心，同时花费的时间更加少。
- 序列匹配：和 SLAM 中一样，假设拍摄图一般是有序的，那么直接根据顺序信息进行匹配即可，当然，可能不同时序的图像之间也会存在匹配关系，所以通常会结合词汇树方式一同使用。

本次课设为了方便，考虑输入图像是有序的，即基于序列匹配的方法，不过我的开源代码各板块间独立性，后期可以继续拓展形成新的视图匹配对寻找方法。

#### 1. 误匹配点对过滤

由于遮挡、重复纹理等因素，sift 提取的特征点和描述子在匹配过程中会存在误匹配的情况，所以需要将错误的匹配点对剔除。

通常包括两种方法：比值测试和几何一致性检测。

比值测试：指的是特征点匹配时讲最近邻和次近邻相似度做比较过滤掉重复、错误的匹配对。

几何一致性检测：指的是利用 RANSAC 方法求解基础矩阵 F 或者单应矩阵 H 时，过滤 mask 掉异常点即外点，保留内点；算法对超参数的选择十分敏感特别是内点阈值 T。

参考论文即在该步骤提出了一种自适应阈值（基于 AC -ransac）的方法，大致是最小化残差期望，从而兼顾保证内点数量以及求解模型的质量，由于时间原因暂时没有研究进一步实现。

#### 1. 增量法 sfm 初始化

本质上就是选择初始视图进行位姿估计和点云重建，设为视图 0 和视图 1，其中会将视图 0 的位姿变换矩阵设为单位阵和 0 向量；该步骤很重要，初始位姿和三维点云直接决定了后续的重建效果，只有

初始位姿是通过分解本质矩阵 E 来得到位姿的，后面的增量过程利用已重建得到的三维点构造 PnP 问题进行位姿重建。

简单来说是指选取两张匹配的图像，设定其中一张图像的位姿为单位阵，然后通过它们之间的匹配点对估计出 E 矩阵，将 E 矩阵分解获得另一张图像的位姿。在估计出两张图像的位姿后，就可以通过三角化 (triangulation) 来生成三维点。

### 1. sfm 增量过程

不断增加视图  $0, 1, 2, \dots, i, i+1..$  视图  $i+1$  与视图  $i$  进行匹配，选择  $i-1$  和  $i$  匹配与  $i$  和  $i+1$  视图匹配的公共已重建三维点，使用 PnP 方法计算视图  $i+1$  的位姿，fusion 新增的三角化点云；每一步骤都要进行 bundle adjustment；

#### 1. 剔除异常点

对于所有重建的三维点，剔除显然的异常点，并最后全局进行一次 bundle adjustment。

#### 1. (可选) 全局 BA

对全局的三维点云和各视图位姿统一进行一次捆绑调整。

## 4.2 可视化功能界面

为了更好的展示 sfm 实验结果，使用 pyqt5 对整个系统进行抽象整理分块，利用 pyqt5 可视化，构造交互、结果展示界面，可以帮助 researcher 更好的理解整个过程。

pyqt5 是一个基于 Qt5 的 Python 接口，由一组 Python 模块构成。Qt5 是一个跨平台的 C++ 图形用户界面库，可以运行于 Unix, Windows, and Mac OS 等多个平台。pyqt5 本身拥有超过 620 个类和 6000 函数及方法，提供了一系列创建桌面应用的 UI 元素。

pyqt5 使用信号槽机制进行通信，用 python 语言编写跨平台的桌面应用程序，同时也可以与其他 Python 库或框架（如 numpy, pandas, scipy 等）配合使用，实现数据分析、机器学习、图形处理等功能。

### pyqt5 初始化方法

- 导入 pyqt5 模块，例如 QtCore, QtGui, QtWidgets 等。
- 创建一个 QApplication 对象，这是 pyqt5 应用程序的核心对象，它负责管理应用程序的事件循环、窗口系统集成、设置等。QApplication 对象只能创建一个，并且必须在创建任何其他 QWidget 对象之前创建。
- 创建 QMainWindow 设置 UI、slot、func 等；
- 最后用 QWidget 对象的 show() 方法，将窗口控件显示在屏幕上，并且调用 QApplication 对象的 exec\_() 方法，启动应用程序的事件循环。这两个方法必须在最后调用，并且 exec\_() 方法会阻塞程序的执行直到窗口关闭。

### pyqt5 主要模块、类、函数和方法

- QtCore 模块：包含了核心的非 GUI 的功能。主要和时间、文件与文件夹、各种数据、流、URLs、mime 类文件、进程与线程一起使用。其中最重要的类是 QObject 类，它是所有 pyqt5 对象的基类，提供了信号槽机制、属性系统、元对象系统等功能。另外还有 QTimer 类（定时器）、QThread 类（线程）、QProcess 类（进程）、 QFile 类（文件）、QUrl 类（URL）等常用类。
- QtGui 模块：包含了窗口系统、事件处理、2D 图像、基本绘画、字体和文字类。其中最重要的类是 QPainter 类，它是一个绘图工具，可以在 QWidget 或 QPixmap 等设备上绘制各种图形、文字、图像等。另外还有 QImage 类（图像）、QPixmap 类（位图）、 QFont 类（字体）、QColor 类（颜色）等常用类。

- QtWidgets 模块：包含了一系列创建桌面应用的 UI 元素。其中最重要的类是 QWidget 类，它是所有窗口控件的基类，提供了窗口的基本功能和属性。另外还有很多 QWidget 的子类，例如 QPushButton 类（按钮）、QLabel 类（标签）、QLineEdit 类（单行文本框）、QTextEdit 类（多行文本框）、QComboBox 类（下拉列表框）、QCheckBox 类（复选框）、QRadioButton 类（单选按钮）、QSlider 类（滑动条）、QProgressBar 类（进度条）、QCalendarWidget 类（日历控件）等常用控件。

## pyqt5 的高级特性、扩展性

- pyqt5 支持多线程编程，可以使用 QtCore 模块中的 QThread 类或 Python 标准库中的 threading 模块创建和管理线程，实现后台任务的并发执行。
- pyqt5 支持多进程编程，可以使用 QtCore 模块中的 QProcess 类或 Python 标准库中的 multiprocessing 模块创建和管理进程，实现任务的分布式执行。

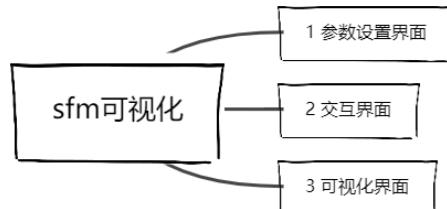
## QtDesigner

QtDesigner 是一个基于 Qt5 的图形用户界面设计工具，可以用来创建和编辑 PyQt5 应用程序的 UI 界面。它提供了一个可视化的拖拽和点击的方式，让开发者可以快速地布局和配置各种控件，如按钮、标签、文本框、下拉列表等，并且可以预览和测试界面的效果。

且 QtDesigner 可与 PyCharm 等 IDE 集成，方便开发者在同一个环境下进行界面设计和代码编写。也可以单独使用 QtDesigner 创建和保存.ui 文件，然后使用 pyuic5 工具将.ui 文件转换为.py 文件，供 Python 程序调用。

## 界面设计

本次基于 sfm 的可视化的设计主要利用了 QtDesigner，使用了 PyQt 的多线程特性，并重写鼠标点击事件过滤器，编写槽函数，点云、label 可视化、logging 嵌入等等。



### 4.2.1 参数设置模块

该界面用于用户输入或选择相关的参数，例如图像路径、相机内参、优化方法、sfm 过程相关参数等等。该界面可以使用表单、复选框、滑动条等控件来实现，每个参数都应该有一个合理的默认值和范围，以及一个简短的说明。该界面还应该恢复默认参数和保存参数的按钮，以固定算法的超参数。

### 4.2.2 交互模块

该界面用于用户与算法的交互，例如查看当前的进度、暂停或继续算法、保存或加载中间结果、调整参数等。该界面可以使用按钮、文本框等控件来实现，每个控件都应该有一个清晰的标签和功能。

且该界面还应该有一个日志窗口，用于显示算法的运行信息和错误提示。

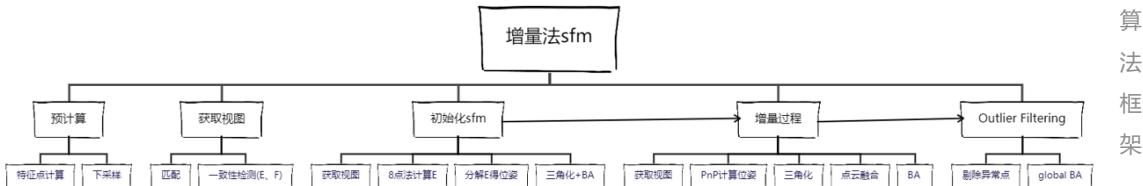
### 4.2.3 结果展示模块

该界面用于用户查看算法的输出结果，例如视图选择匹配点结果以及重建的三维点云和相机位姿。该界面可以使用 OpenGL 或其他图形库来实现，提供基本的渲染和交互功能，例如缩放、旋转等。

## 五 详细结构设计

## 5.1 Incremental SFM

算法框架主要包括五个部分：预算算、视图获取、初始化 sfm、增量过程、Outlier Filtering。



增量法 SFM, 核心步骤在于视图对寻找 (SLAM 假设顺序列视图强相关) 、特征点匹配、本质矩阵计算、几何一致性检测、本质矩阵分解、PnP 计算位姿、点云融合、BA 缆绑调整等。

世界坐标默认为视图 0 的相机坐标。

由于基础矩阵分解时尺度是齐次的，因此所有视图的尺度与初始视图 0、视图 1 分解 E 的平移向量 t 的尺度一致。这也间接说明了我们重建得到的是一个相对三维空间位置。

sfm 基本步骤：

1. 将所有图像 downscale 并提取所有图像的 sift 特征，得到特征点和描述子。
2. 初始化 sfm：计算视图 0 和某一视图的本质矩阵，一致性检测去除噪点，分解得到 T 和 R 并重建为初始化点云，视图 0 的相机坐标系为世界坐标系；
3. 不断增加视图 0,1,2,...,i,i+1.. 视图 i+1 与视图 i 进行匹配，选择 i-1 和 i 匹配与 i 和 i+1 视图匹配的公共已重建三维点计算 PnP 视图 i+1 的位姿，然后 fusion 新增点云；
4. 每一步骤都要进行 bundle adjustment。

### 5.1.1 预计算

####

预算算包括图像数据集读入、参数变量初始化和预处理如 downscale degradation。

```

class IncrementalSFM:
    def __init__(self, imgs_dir: str, downscale_factor: float = 1.0, log_handler=None):
        """
        初始化整个sfm的相关参数、中间变量，并读取图像，进行下采样处理。
        :param imgs_dir:
        :param downscale_factor:
        :param log_handler:
        """
    
```

Python

### 初始化

初始化即读入图像集合、相机内参，初始化 sfm 类等。

其中涉及到的参数和中间变量列举如下：

```



```

Python

## 特征描述子

sfm 并不要求实时性, 所以通常使用更为可靠稳健的 SIFT (Scale-Invariant Feature Transform) 特征点, 对应的描述子也称为 sift 描述子, sift 特征在之前已经详细的介绍了, 用 sift 特征衡量两个视图的相似性, 并用 sift 特征点来进行图像匹配, 寻找对应点。

```

def extract_feat(self, name='sift'):
    """
    提取所有img_list的特征点和描述子

    :param name:
    :return:
    """

```

Python

### 5.1.2 视图获取

即图像匹配对的搜索, 可以使用穷举法、视觉词袋、词汇树、暴力匹配、FLANN、序列匹配等, 为了方便实现, 我实现了基于序列匹配的方法, 即顺序图像对检索。

其中实现的 `get_nxt_view` 可以获取下一个连续视图, 且保证匹配点非稀疏, 且中途使用次优过滤阈值 `match_factor` 剔除误匹配点。

```

def get_nxt_view(self, match_factor=0.7):
    """
    获得下一个视图, 返回上一个视图和该视图匹配像素点和匹配的matches

    :param match_factor: 次优过滤因子
    :return: P_pix1, P_pix2, matches
    """

```

Python

### 5.1.3 初始化 sfm

初始 sfm 选择两个视图进行双目重建, 基准视图为视图 0。该步骤很重要, 基本直接决定了后续的重建效果。

算法: 计算视图 0 和某一视图的本质矩阵, 一致性检测去除噪点, 分解得到 T 和 R 并重建为初始化点云, 视图 0 的相机坐标系为世界坐标系, 且维护中间变量和参数。

期间关键使用了 opencv 提供的 `cv2.findEssentialMat` 函数和 `cv2.recoverPose`。

自己实现也很容易, 只需构造 correspondence\_matrix, 然后求最小二乘解 (svd/QR), 中间可能需要进行 scale 变换确保算法稳定性, 这里选择调库还使用了 RANSAC, 更加稳定可靠。

这里还提供了局部 BA 选项和 RANSAC 的阈值参数选择:

```
def initialize_sfm(self, if_local_BA=False, match_factor=0.7, threshold=0.4):
    """
    初始sfm选择两个视图进行双目重建, 基准视图为视图0。该步骤很重要, 基本直接决定了后续的重建效果。

    :param if_local_BA: 是否进行局部BA
    :param match_factor: 次优过滤因子
    :param threshold: 本质矩阵RANSAC阈值
    :return:
    """

```

Python

#### 5.1.3.1 视图获取

直接调用 5.1.2 模块介绍的视图获取方法 `get_nxt_view`, 获得 P\_pix1 和 P\_pix2 以及 matches。

#### 5.1.3.2 E 和位姿

见代码:

```
P_pix1, P_pix2, matches = self.get_nxt_view(match_factor=match_factor)
# 基于8点法和RANSAC计算本质矩阵, 并利用本质矩阵进行几何一致性检测得到mask。
E, mask = cv2.findEssentialMat(P_pix1, P_pix2, self.K, method=cv2.RANSAC, prob=0.999, threshold=0.4)

# 分解本质矩阵E=t^R : svd分解得到4个解, 并根据深度>0进行验证。还可以通过重投影进行一致性检测。
_, R, T, mask = cv2.recoverPose(E, P_pix1, P_pix2, self.K, mask=mask)
keep = np.where(mask > 0)[0]
P_pix1, P_pix2, matches = P_pix1[keep], P_pix2[keep], matches[keep]
```

Python

#### 5.1.3.3 三角化

在 `utils` 目录的 `utils.py` 文件中实现了三角化函数:

$s_1x_1 = KM_1X, s_2x_2 = KM_2X$

其中  $x_1, x_2$  是像素坐标的齐次形式,  $X$  是三维坐标的齐次形式,  $M_1 = [R_1, T_1], M_2 = [R_2, T_2], X = [X, 1]^T$ 。两个式子分别左乘  $x_1^T$  和  $x_2^T$  可以得到一个线性方程组。

```

def triangulation(R1, T1, R2, T2, P_pix1, P_pix2, K):
    """
    三角测量

    """
    pose_1 = np.matmul(K, np.hstack([R1, T1]))
    pose_2 = np.matmul(K, np.hstack([R2, T2]))
    P_w_h = cv2.triangulatePoints(pose_1, pose_2, P_pix1.T, P_pix2.T)
    return (P_w_h / P_w_h[3])[::3].T

```

Python

### 5.1.3.4 BA

bundle adjustment (光束法平差) 是一种优化方法，用于从多个视角的图像中提炼出最优的三维模型和相机参数（内参和外参）。它的基本思想是最小化每个特征点在图像上的重投影误差，即实际观测值与根据三维坐标和相机参数计算得到的预测值之间的差异。bundle adjustment 可以看作是一个非线性最小二乘问题，通常需要使用迭代法求解，例如高斯牛顿法、列文伯格-马夸尔特法等。

增量法 sfm 中的 bundle adjustment 主要有两种形式：局部 BA 和全局 BA。局部 BA 是指在每次添加新的视图时，只对当前视图和与之相关的一些视图和三维点进行优化，以减少计算量和内存消耗。全局 BA 是指在一定间隔或者最后阶段，对所有的视图和三维点进行一次全面的优化，以提高重建的精度和鲁棒性。

为了实现 bundle adjustment，我们需要定义以下几个要素：

- 观测值：每个特征点在图像上的像素坐标；
- 优化量：每个视图的相机姿态（旋转矩阵和平移向量）和每个三维点的空间坐标；
- 投影函数：根据相机内参、外参和三维点坐标，计算该点在图像上的预测坐标；
- 代价函数：根据观测值和预测值之间的误差，计算总体的代价或损失；
- 求解器：根据代价函数和优化量之间的关系，迭代地更新优化量，直到收敛或达到最大迭代次数。

可能还需要计算代价函数对优化量的偏导数或雅可比矩阵，以便求解器进行梯度下降或牛顿法等。这些偏导数可以通过解析法或自动微分法得到。

为了方便，我调用了 scipy 的非线性最小二乘法求解器即 `least_squares` 模块。

```

def local_bundle_adjustment(self, x, X, K, R, T):
    """
    局部光束法平差，针对一个位姿和其重建点进行优化，优化变量为位姿和重建点
    目标函数：将重投影误差(重投影坐标差)设置为残差，最小化残差平方和，默认调用简单的牛顿法优化

    :param x: 像素坐标
    :param X: 三维坐标
    :param K: 内参
    :param R: 位姿
    :param T: 位姿
    :return: BA后的三维点和位姿
    """

```

Python

### 5.1.4 增量过程

在初始化 sfm 的基础上，逐步增量重建点云和位姿，这是 sfm 的关键步骤。

```

def increment(self, if_local_BA=False, match_factor=0.7, threshold=0.4):
    """
    增量过程，即state=1，每步get_nxt_view()得到下一视图，并进行几何一致性检测
    获取当前视图已重建得到的三维点计算PnP得到位姿，再三角化，点云融合，BA优化...
    :param if_local_BA: 是否进行局部BA
    :param match_factor: 次优过滤阈值
    :param threshold: ransac阈值
    :return:
    """

```

Python

### 5.1.4.1 视图获取

同 5.1.2 视图获取

### 5.1.4.2 PnP

与初始化 sfm 不同，我们选择使用 PnP 利用重叠的三维点云计算位姿，以增强前后视图的依赖性，计算结果的鲁棒性和稳定性，直接计算出相对视图 0 的位姿，具体来说，使用 PnP 的原因：

- pnp 只需要知道相机内参，而不需要知道其他视图的位姿，因此更加灵活和鲁棒；
- pnp 可以利用已重建的三维点作为约束，而不需要重新匹配特征点，因此更加准确和高效；
- pnp 可以避免累积误差的问题，因为每次添加新视图时，都是基于世界坐标系进行优化，而不是基于相对坐标系；
- pnp 可以处理任意顺序的视图，而不需要按照时间或空间上的连续性，因此更加灵活和通用。

中间可以利用本质矩阵、基础矩阵进行几何一致性检测。

PnP 计算我选择基于迭代的方法，以增强结果稳定性。

```

# 获得三维点
obj_points, img_points = self.get_common_points(matches, P_pix2)
if len(obj_points) < 6:
    return
_, R_vector, T, keep = cv2.solvePnPRansac(obj_points, img_points, self.K, distCoeffs=np.zeros(4),
                                             flags=cv2.SOLVEPNP_ITERATIVE)

```

Python

### 5.1.4.2 BA 和点云融合

BA 同 5.1.3.4 模块的局部 BA 优化参数。

点云融合指融合未重建的三维点，并维护中间变量方便后续的重建过程，在 sfm 主类中，我定义了方法 `fusion_cur_view` 以实现：

```

def fusion_cur_view(self, Points, matches, P_pix_cur):
    """
    融合点云，重合的匹配点不融合，其他点融合到self.Points内并维护更新self.h

    :param Points: 当前视图所有match三角化的3D坐标
    :param matches: 当前视图与上一个视图的匹配
    :param P_pix_cur: 当前视图的像素坐标
    :return:
    """

```

Python

## 5.1.5 后处理

后处理包括 `select_global_points` 剔除异常点以及全局 BA，均是可选项，分别定义了方法

`select_global_points` 和 `global_bundle_adjustment`。

```
def select_global_points(self, threshold=0.5):
    """
    剔除一些outlier, 保留inlier
    稀疏重建中, 一般认为重投影误差小于0.5的点为inlier, 大于0.5的点为outlier
    处理的数据: self.Points, self.Colors, self.pts_info

    :param threshold: 保留inlier的阈值
    :return:
    """

def global_bundle_adjustment(self):
    """
    全局光束法平差,最小化所有位姿、重建点的重投影误差, 优化变量为R、T、X, 优化方法为LM
    输入参数包括: 相机内参K, 所有视图的位姿R、T, 所有视图的像素坐标P_pix, 所有视图的三维点P_w
    其中各个点的下标信息存储在pts_info中, pts_info的第一列为视图下标,第二、三列为像素坐标

    :return:
    """

```

Python

## 5.1.6 sfm 单步函数

sfm 的过程被我抽象成一个状态机, 每个 step 就是一个状态, 每个状态都有对应的任务目标  
sfm 状态:

- 1:待机
- 0: 完成特征提取
- 1: 完成初始化 sfm+ 增量重建 ing
- 2: 完成 step 增量重建
- 3: 完成 select 和全局 BA,over

```
def step(self, if_local_BA=False, if_global_BA=False, if_select=True, match_factor=0.7, threshold=0.5):
    """
    sfm单步

    :param if_local_BA: 局部BA
    :param if_global_BA: 全局BA
    :param if_select: 是否剔除outlier, 阈值固定为0.5
    :param match_factor: sift的最优、次优匹配点距离比例,
    :param threshold: ransac内点阈值
    :return:
    """

    if self.state == -1: # 待机状态: 提取特征

        elif self.state == 0: # 提取特征: 初始化sfm

            elif self.state == 1: # 初始化sfm: 增量sfm

                elif self.state == 2: # 完成增量sfm->全局BA
                    if if_select: # select

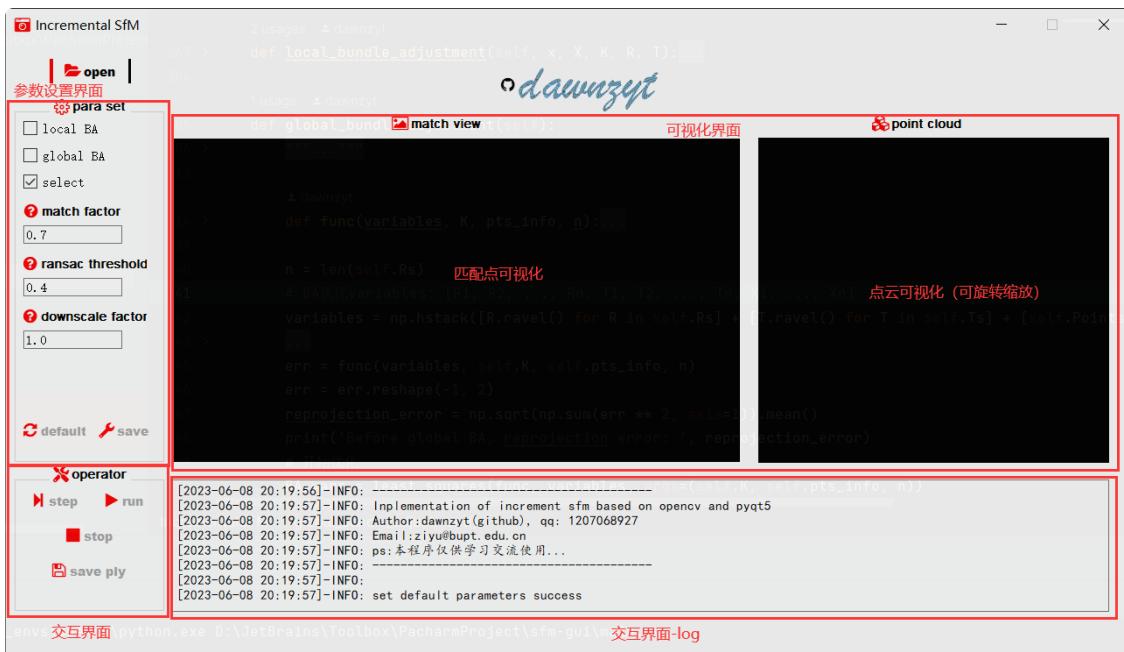
                        if if_global_BA: # 全局BA

```

Python

## 5.2 可视化交互界面

这里只介绍每个板块包括什么内容，可以做到什么，如何交互，以及一些 tooltips，具体实现见 github。



### 5.2.1 整体架构

整体交互界面 ui 的架构：

```
from sfm_ui import Ui_MainWindow # QtDesigner ui

# 单步sfm线程
class StepWorker(QThread, QObject):

# 连续运行sfm线程
class RunWorker(QThread, QObject):

# logging的处理器内嵌QPlainTextEdit,以记录日志传来时同步更新QPlainTextEdit以display。
class QPlainTextEditLogger(logging.Handler):

# sfm-gui主窗口
class MainUi(QMainWindow, Ui_MainWindow):
    def __init__(self):
        self.initUI()
        self.setStyle()
        self.initLog()
        self.setMatchLabel(img)
    @pyqtSlot()
    ...# 各种btn_clicked槽函数
    # 重写事件过滤器, 保证在sfm运行时, 按钮不可点击, 除了stop_btn
    def eventFilter(self, obj, event):
if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = MainUi()
    demo.show()
    app.installEventFilter(demo) # 为app安装事件过滤器
    sys.exit(app.exec_())

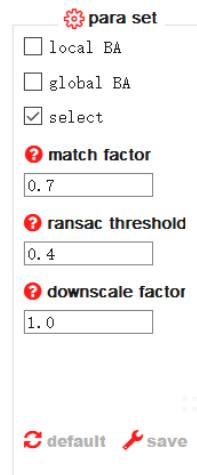
```

Python

其中一些有意思的 tricks 比如多线程执行 sfm 算法，重新 logging.handler 实时同步记录 log 并可视化到 QPlainTextEdit 中，以及一些槽函数、事件过滤器的交互，其他不用多说，都是朴素的实

现。

## 5.2.2 参数界面



如图包括可设置的参数，即：

`local BA`：优点：速度快，缺点：通常情况下鲁棒性强，但某些视图容易发生误差偏移

`global BA`：优点：精度高，缺点：计算量大，速度慢

`select`：剔除 global 异常点

`match factor`：sift 的最优、次优匹配点距离比例，分两种情况：1. 图片分辨率高，特征点数量多，此时需剔除一些错误关键点，即设置较小的 `match_factor`；2. 图片分辨率低，特征点数量少，此时需保留更多的关键点用以 RANSAC，即设置较大的 `match_factor`

`ransac threshold`：ransac 内点阈值，越大内点越多，精确度越低，建议遵循默认值

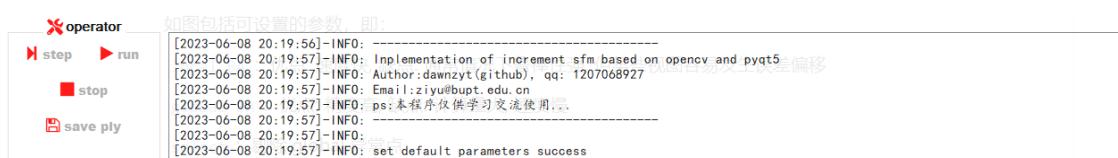
`downscale factor`：降采样因子，大于等于 1.0，用来加速，越大精确度越低

两个 btn：

`default`：恢复设置默认参数

`save`：保存参数

## 5.2.3 交互界面



一些可交互的 btn 以及 log 记录算法过程中的一些 debug information。

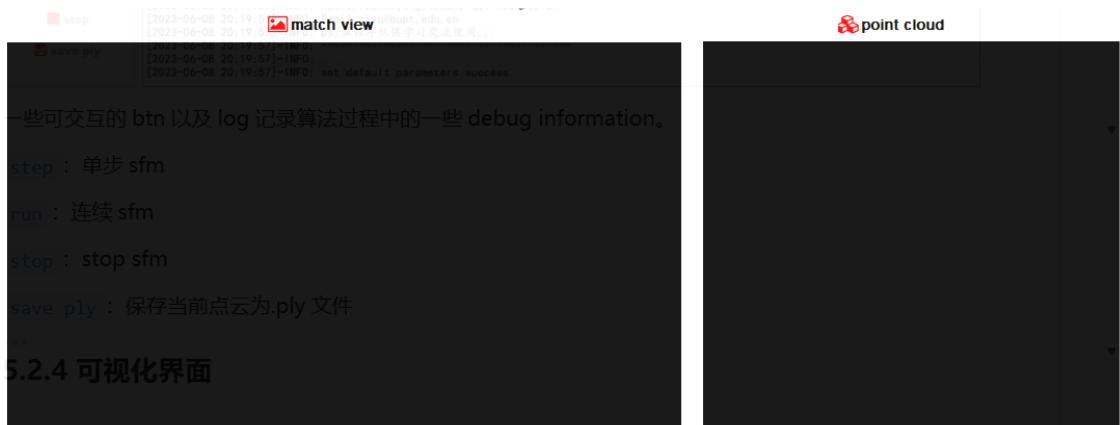
`step`：单步 sfm

`run`：连续 sfm

`stop`：stop sfm

`save ply`：保存当前点云为.ply 文件

## 5.2.4 可视化界面



包括匹配点的可视化和稀疏重建三维点云的可视化，见结果展示与分析。

## 六 结果展示与分析

### Visual Geometry Group - University of Oxford

Computer Vision group from the University of Oxford

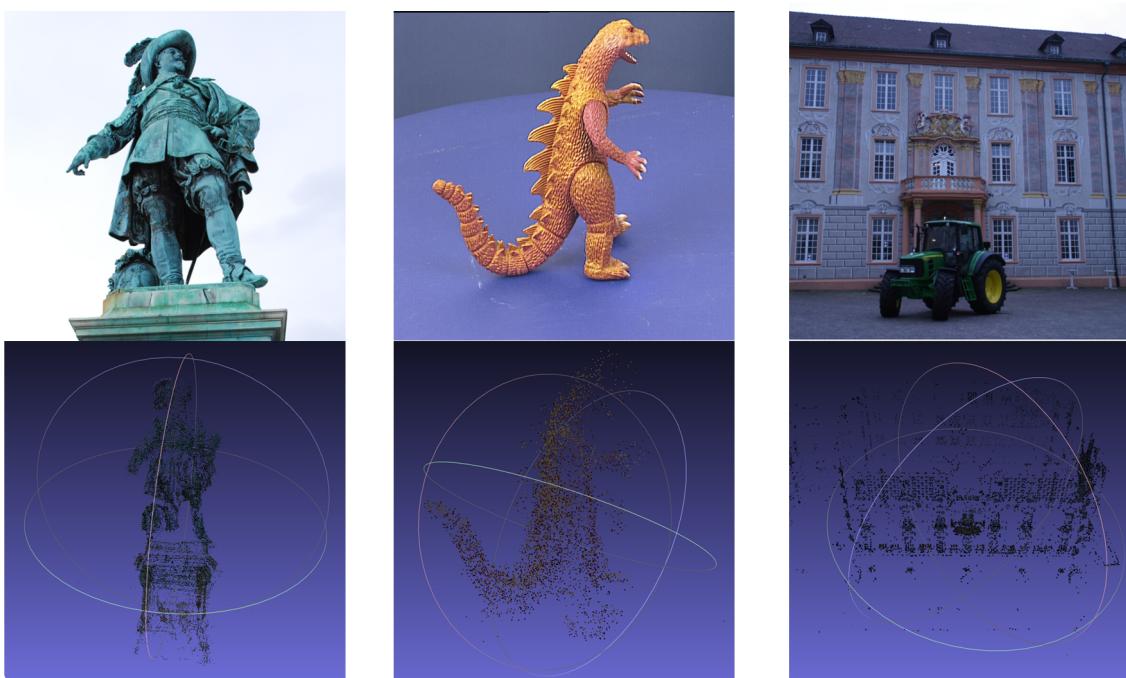
[www.robots.ox.ac.uk](http://www.robots.ox.ac.uk)

测试时，我们输入的是单场景的多视图图像数据，且包括内参 K.txt。

在可视化界面可以单击 `open` 来选择数据集。

这些图像将输入到我设计的 incremental sfm 算法中，逐步重建出稀疏点云并计算位姿。

如图是几个多视图数据集的重建结果，`save ply` 后用 meshlab 可视化：



在我们的实验中，使用 bundle adjustment 会占用大量的时间，除非先下采样，bundle adjustment 可以将 reprojection error 降低 3 个数量级以上，效果好的前提是 sift 匹配特征点足够鲁棒。

下面给出可视化界面的实验结果截图，选择 `castle-P30` 数据集，如图：



参数选择为默认参数，下采样因子为 1.0，即原分辨率，接下来单步进行。

- step: 提取特征



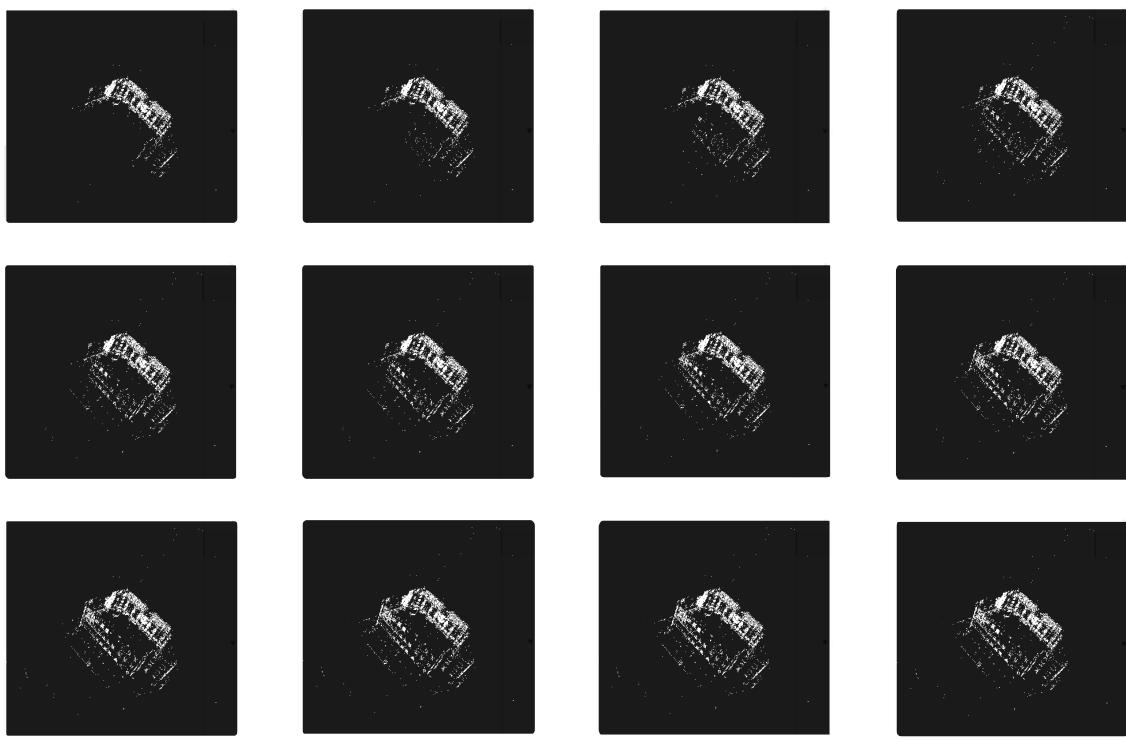
参数选择为默认参数，下采样因子为 1.0，即原分辨率，接下来单步进行。

- step: 初始化 sfm

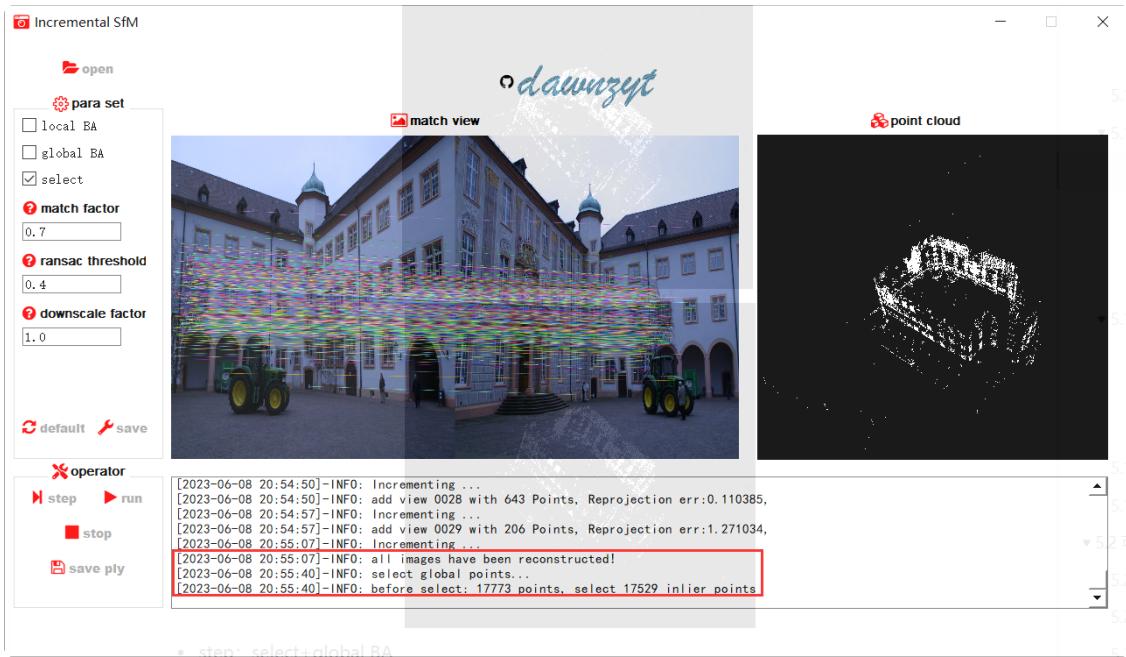


- step: 增量过程



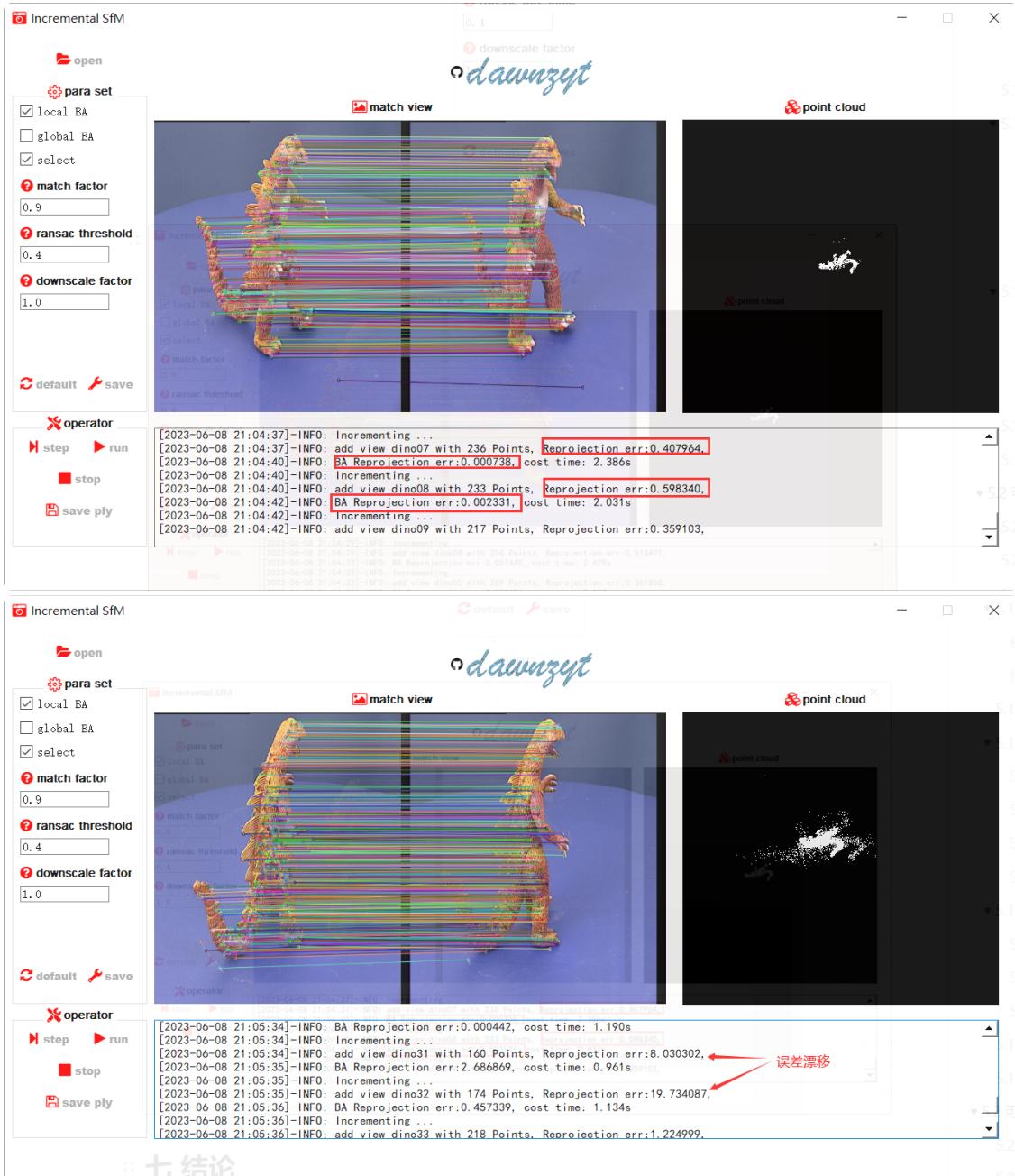


- step: select+global BA



综上，完成了整个 sfm 稀疏点云重建和离线位姿计算

接下来我用 [dino](#) 恐龙数据集为例子来展示局部 BA 的过程 (match\_factor 修改为 0.9 增加特征点)，图像分辨率仅为 [720x576](#)。



如图，在前期 sfm 时 BA 后的重建效果稳定，但后期局部 BA 优化后的结果与真实值相差很大，导致重建的地图发生严重的变形和偏移，如图可以看到后期视图的 error 突增，且得到随机分布的离群重建点。

主要原因在于：

- 视图之间的匹配点质量不高，存在大量的错误匹配或者重复匹配，导致优化过程中产生错误的约束和损失函数；
- 视图之间的视差太小，导致三角化和优化过程中产生不稳定的解或者退化的情况；

一些解决方法：

- 提高视图之间的匹配点质量，使用更稳定和鲁棒的特征提取和匹配方法，剔除错误匹配或者重复匹配；
- 增加视图之间的视差，选择更合适的相机运动方式和拍摄角度，使得三角化和优化过程更稳定和可靠；

## 七 结论

增量法 sfm 是一种基于图像的三维重建算法，它通过逐步添加新的视角和优化相机姿态和场景结构来恢复相机参数和三维点坐标。这种方法的优点是鲁棒性高，重建精度高，但缺点是对初始相机对的选取和相机的添加顺序敏感，大场景会产生累计误差和场景漂移，而且重复进行捆绑调整效率低。

本课设工作详细介绍我们设计的 sfm 算法框架和 PyQt 可视化交互界面框架，其中 sfm 框架实现了一个 simple version 的增量法 sfm，能成功计算稀疏重建点云和视图位姿，并且提供较多的可调节参数，保证能得到较为鲁棒、准确的结果；然后对于交互界面利用 PyQt 嵌入 sfm 算法模块，并基于 PyQt 的特性实现了简介有效、用户友好的可视化交互界面，能实现完整流程。

但从运动结构的恢复的稀疏表示缺乏一些明确的定量指标来进行评估。且我设计的模型有一些缺点，如假设视图按编号有序，图像必须按某种顺序传递给模型，这意味着两个图像的重叠部分必须到某种程度才能重建。不过这部分我保留了视图获取模块，可以进一步实现更多的图像对搜索算法进行拓展。

综上，我认为我们完整实现了基本的 structure from motion 算法，并得到了较为满意的结果。

## 八 未来期望

总之，我对该次工作较为满意，不过仍有许多拓展、完善的空间，比如可以：

全局式 sfm、分层 sfm、分布式 sfm 等，比较它们的效率和精度，并探索它们在不同场景下的适用性  
嵌入实现更多的图像匹配对搜索算法如词汇树等。

考虑相机的畸变模型，先对图像进行畸变校正再匹配。

可以考虑使用深度学习的方法来提高特征提取和匹配的效果，例如使用预训练的神经网络来提取特征描述子，或者使用学习的度量函数来计算特征之间的相似度；

从 2D 图像生成 3D 模型的稀疏表示是非常重要的一步，但实际所有应用都需要密集图像重建，后续可以考虑使用多视图立体（MVS）算法生成这种密集重建图像，形成更丰富的 pipeline。

对于人工设计 RANSAC 等可变参数的弊端，考虑引入 AC-RANSAC 等更鲁棒的方法。

根据实验结果中的分析，可以考虑计算两个视图之间的相对偏差角度来进一步考虑是否添加该视图。

## 附录-相关开源系统

### Images to 3D model pipelines

	Point cloud	Dense cloud	Surface	Textured
Bundler	Green	Red	Red	Red
CMVS	Red	Green	Red	Red
COLMAP	Green	CUDA only	CUDA only	Red
Meshlab	Red	Red	Green	Green
MVE	Green	Green	Green	Red
MVS-texturing	Red	Red	Red	Green
openMVG	Green	Red	Red	Red
openMVS	Red	Green	Green	Red
Theia	Green	Red	Red	Red
VisualFSM	Green	Green	Red	Red

上图中从左到右依次是：原始图像、稀疏点云重建、稠密点云重建、表面重建、纹理贴图。绿色表示该系统实现了对应模块的功能，红色表示未实现。

Bundler: <http://www.cs.cornell.edu/~snavely/bundler/>

CVMS: <https://www.di.ens.fr/cmvs/>

ColMap: <https://demuc.de/colmap/>

Meshlab: <https://www.meshlab.net/>

MVE: <https://www.gcc.tu-darmstadt.de/home/proj/mve/index.en.jsp>

MVS-Texturing: <https://www.gcc.tu-darmstadt.de/home/proj/texrecon/>

OpenMVG: <https://openmvg.readthedocs.io/en/latest/software/SfM/SfM/>

OpenMVS: <https://github.com/cdcseacave/openMVS>

Theia: <http://theia-sfm.org/>

VisualSFM: <http://ccwu.me/vsfm/>