第五节课习题

高翔

2021 年 9 月 9 日

1 习题说明

- 第 i 节课习题所有材料打包在 Li.zip 中，$\forall i = 1 \ldots 8$。
- 习题分为若干种：**计算类**习题，需要读者编程计算一个实际问题，我们会附有参考答案以供自测。**操作类**习题，会指导读者做一个具体的实验，给出中间步骤截图或结果。**简述类**习题则提供阅读材料，需要读者阅读材料后，回答若干问题。
- 每个习题会有一定的分值。每次习题分值加和为 10 分。你需要获得 8 分以上才能得到“通过”的评价。带 $*$ 的习题为附加题，会在总分之外再提供一定的分值，所以总和可能超过 10 分。换句话说，你也可以选择一道附加题，跳过一道正常题。
- 每道习题的给分由助教评判，简述类习题可能存在一定开放性，所以评分也存在主观因素。
- 请利用深蓝学院系统提交习题。每次习题我们会记通过与否。提交形式为 word 或 pdf 格式报告，如有编程习题请提交可编译的源码。
- 为方便读者，我通常会准备一些阅读材料，放在 books/或 papers/目录下。请读者按个人需求使用这些材料。它们多数是从网络下载的，如果侵犯到你的权利，请及时告诉我。
- 每个习题会标注大致用时，但视同学个人水平可能会有出入。
- 习题的完成情况会影响你对本课程内容的掌握程度，请认真、独立完成。**习题总得分较高的同学将获得推荐资格**。

备注：

- 本习题内容更新于 2021 年 9 月。考虑到大家的知识水平增加，本次更新增加了一些作业内容和难度。

2 ORB 特征点（4 分，约 3 小时）

ORB(Oriented FAST and BRIEF) 特征是 SLAM 中一种很常用的特征，由于其二进制特性，使得它可以非常快速地提取与计算 [1]。下面，你将按照本题的指导，自行书写 ORB 的提取、描述子的计算以及匹配的代码。代码框架参照 computeORB.cpp 文件，图像见 1.png 文件和 2.png。

2.1 ORB 提取

ORB 即 Oriented FAST 简称。它实际上是 FAST 特征再加上一个旋转量。本习题将使用 OpenCV 自带的 FAST 提取算法，但是你要完成旋转部分的计算。旋转的计算过程描述如下 [2]：

在一个小图像块中，先计算质心。质心是指以图像块灰度值作为权重的中心。

1. 在一个小的图像块 B 中，定义图像块的矩为：

$$m_{pq} = \sum_{x,y \in B} x^p y^q I(x,y), \quad p,q = \{0,1\}.$$

2. 通过矩可以找到图像块的质心：

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}).$$

3. 连接图像块的几何中心 O 与质心 C，得到一个方向向量 $\overrightarrow{OC}$，于是特征点的方向可以定义为：

$$\theta = \arctan(m_{01}/m_{10}).$$

实际上只需计算 m_{01} 和 m_{10} 即可。习题中取图像块大小为 16x16，即对于任意点 (u,v)，图像块从 $(u-8,v-8)$ 取到 $(u+7,v+7)$ 即可。请在习题的 computeAngle 中，为所有特征点计算这个旋转角。

提示：

1. 由于要取图像 16x16 块，所以位于边缘处的点（比如 $u<8$ 的）对应的图像块可能会出界，此时需要判断该点是否在边缘处，并跳过这些点。
2. 由于矩的定义方式，在画图特征点之后，角度看起来总是指向图像中更亮的地方。
3. std::atan 和 std::atan2 会返回弧度制的旋转角，但 OpenCV 中使用角度制，如使用 std::atan 类函数，请转换一下。

作为验证，第一个图像的特征点如图 1 所示。看不清可以放大看。

图 1: 带有旋转的 FAST

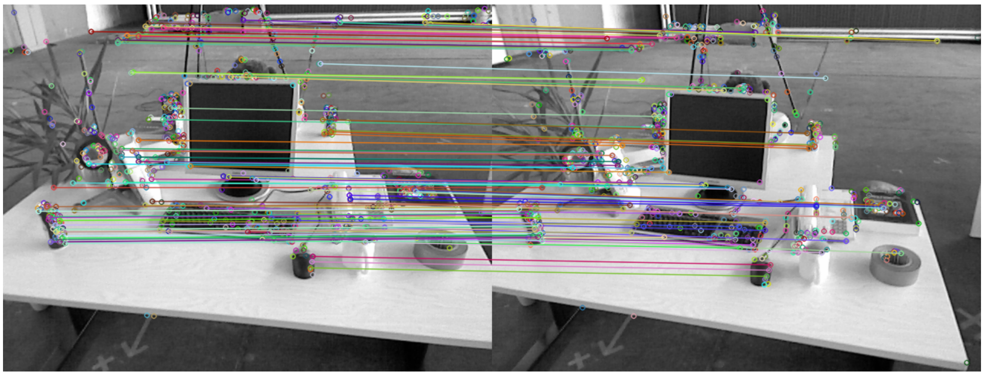

图 2: 匹配图像

2.2 ORB 描述

ORB 描述即带旋转的 BRIEF 描述。所谓 BRIEF 描述是指一个 0-1 组成的字符串（可以取 256 位或 128 位），每一个 bit 表示一次像素间的比较。算法流程如下：

1. 给定图像 I 和关键点 (u, v)，以及该点的转角 θ。以 256 位描述为例，那么最终描述子

$$\mathbf{d} = [d_1, d_2, \ldots, d_{256}].$$

2. 对任意 $i = 1, \ldots, 256$，d_i 的计算如下。取 (u, v) 附近任意两个点 $\mathbf{p}, \mathbf{q}$，并按照 θ 进行旋转：

$$\begin{bmatrix} {u_p}' \\ {v_p}' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} u_p \\ v_p \end{bmatrix}. \tag{1}$$

其中 u_p, v_p 为 $\mathbf{p}$ 的坐标，对 $\mathbf{q}$ 亦然。记旋转后的 $\mathbf{p}, \mathbf{q}$ 为 $\mathbf{p}', \mathbf{q}'$，那么比较 $I(\mathbf{p}')$ 和 $I(\mathbf{q}')$，若前者大，记 $d_i = 0$，反之记 $d_i = 1$[1]。

这样我们就得到了 ORB 的描述。我们在程序中用 256 个 bool 变量表达这个描述[2]。请你完成 computeORBDesc 函数，实现此处计算。注意，通常我们会固定 $\mathbf{p}, \mathbf{q}$ 的取法（称为 ORB 的 pattern），否则每次都重新随机选取，会使得描述不稳定。我们在全局变量 ORB_pattern 中定义了 $\mathbf{p}, \mathbf{q}$ 的取法，格式为 u_p, v_p, u_q, v_q。请你根据给定的 pattern 完成 ORB 描述的计算。

提示：

1. $\mathbf{p}, \mathbf{q}$ 同样要做边界检查，否则会跑出图像外。如果跑出图像外，就设这个描述子为空。
2. 调用 cos 和 sin 时同样请注意弧度和角度的转换。

2.3 暴力匹配

在提取描述之后，我们需要根据描述子进行匹配。暴力匹配是一种简单粗暴的匹配方法，在特征点不多时很有用。下面你将根据习题指导，书写暴力匹配算法。

所谓暴力匹配思路很简单。给定两组描述子 $\mathbf{P} = [p_1, \ldots, p_M]$ 和 $\mathbf{Q} = [q_1, \ldots, q_N]$。那么，对 $\mathbf{P}$ 中任意一个点，找到 $\mathbf{Q}$ 中对应最小距离点，即算一次匹配。但是这样做会对每个特征点都找到一个匹配，所以我们通常还会限制一个距离阈值 d_{max}，即认作匹配的特征点距离不应该大于 d_{max}。下面请你根据上述描述，实现函数 bfMatch，返回给定特征点的匹配情况。实践中取 $d_{max} = 50$。

[1]注意反过来记也可以，但是程序中要保持一致。

[2]严格来说可以用 32 个 uchar 以节省空间，但是那样涉及到位运算，本习题只要求掌握算法。

2.4 多线程 ORB

C++17 标准中带来了很多语言层面的并行化支持。它们对算法开发人员非常友好，我们可以很轻松地借助标准库的内容，就写出稳定、可靠的并行程序。在 ORB 这个例子中，很明显，角点方向的计算和描述子的计算都是很容易并行化的。请根据你在前两题中的结果，实现多线程并行化的 ORB 描述子计算过程，并比较多线程与单线程之间的性能差异。

为了方便起见，我为你搭建了计算角点部分的多线程计算方法框架，你只需填入关键代码部分即可。而对于计算描述子部分，请参考角度计算部分来完成。如果你的编译器还不支持 17 标准，请升级你的编译器。

提示：

1. 你需要按位计算两个描述子之间的汉明距离。
2. OpenCV 的 DMatch 结构，queryIdx 为第一图的特征 ID，trainIdx 为第二个图的特征 ID。
3. 作为验证，匹配之后输出图像应如图 2 所示。

最后，请结合实验，回答下面几个问题：

1. 为什么说 ORB 是一种二进制特征？
2. 为什么在匹配时使用 50 作为阈值，取更大或更小值会怎么样？
3. 暴力匹配在你的机器上表现如何？你能想到什么减少计算量的匹配方法吗？
4. 多线程版本相比单线程版本是否有提升？在你的机器上大约能提升多少性能？

3 从 E 恢复 R, t（3 分，约 1 小时）

我们在书中讲到了单目对极几何部分，可以通过本质矩阵 $\mathbf{E}$，得到旋转和平移 $\mathbf{R}, \mathbf{t}$，但那时直接使用了 OpenCV 提供的函数。本题中，请你根据数学原理，完成从 $\mathbf{E}$ 到 $\mathbf{R}, \mathbf{t}$ 的计算。程序框架见 code/E2Rt.cpp.

设 Essential 矩阵 $\mathbf{E}$ 的取值为（与书上实验数值相同）：

$$\mathbf{E} = \begin{bmatrix} -0.0203618550523477 & -0.4007110038118445 & -0.03324074249824097 \\ 0.3939270778216369 & -0.03506401846698079 & 0.5857110303721015 \\ -0.006788487241438284 & -0.5815434272915686 & -0.01438258684486258 \end{bmatrix}$$

. 请计算对应的 $\mathbf{R}, \mathbf{t}$，流程如下：

1. 对 $\mathbf{E}$ 作 SVD 分解：
$$\mathbf{E} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}.$$

2. 处理 $\mathbf{\Sigma}$ 的奇异值。设 $\mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \sigma_3)$ 且 $\sigma_1 \geq \sigma_2 \geq \sigma_3$，那么处理后的 $\mathbf{\Sigma}$ 为：
$$\mathbf{\Sigma} = \mathrm{diag}(\frac{\sigma_1 + \sigma_2}{2}, \frac{\sigma_1 + \sigma_2}{2}, 0). \tag{2}$$

3. 共存在四个可能的解：
$$\begin{aligned} \boldsymbol{t}_1^\wedge &= \boldsymbol{U}\boldsymbol{R}_Z(\tfrac{\pi}{2})\mathbf{\Sigma}\boldsymbol{U}^T, \quad \boldsymbol{R}_1 = \boldsymbol{U}\boldsymbol{R}_Z^T(\tfrac{\pi}{2})\boldsymbol{V}^T \\ \boldsymbol{t}_2^\wedge &= \boldsymbol{U}\boldsymbol{R}_Z(-\tfrac{\pi}{2})\mathbf{\Sigma}\boldsymbol{U}^T, \quad \boldsymbol{R}_2 = \boldsymbol{U}\boldsymbol{R}_Z^T(-\tfrac{\pi}{2})\boldsymbol{V}^T. \end{aligned} \tag{3}$$

   其中 $\boldsymbol{R}_Z(\frac{\pi}{2})$ 表示沿 Z 轴旋转 90 度得到的旋转矩阵。同时，由于 $-\boldsymbol{E}$ 和 $\boldsymbol{E}$ 等价，所以对任意一个 $\boldsymbol{t}$ 或 $\boldsymbol{R}$ 取负号，也会得到同样的结果。因此，从 $\boldsymbol{E}$ 分解到 $\boldsymbol{t}, \boldsymbol{R}$ 时，一共存在**四个**可能的解。请打印这四个可能的 $\mathbf{R}, \mathbf{t}$。

提示：用 AngleAxis 或 Sophus::SO3 计算 $\boldsymbol{R}_Z(\frac{\pi}{2})$。

注：实际当中，可以利用深度值判断哪个解是真正的解，不过本题不作要求，只需打印四个可能的解即可。同时，你也可以验证 $\mathbf{t}^\wedge\mathbf{R}$ 应该与 $\mathbf{E}$ 只差一个乘法因子，并且与书上的实验结果亦只差一个乘法因子。

3D->2D的PnP

4 用 G-N 实现 Bundle Adjustment 中的位姿估计（3 分，约 2 小时）

Bundle Adjustment 并不神秘，它仅是一个目标函数为重投影误差的最小二乘。我们演示了 Bundle Adjustment 可以由 Ceres 和 g2o 实现，并可用于 PnP 当中的位姿估计。本题，你需要自己书写一个高斯牛顿法，实现用 Bundle Adjustment 优化位姿的功能，求出相机位姿。严格来说，这是 Bundle Adjustment 的一部分，因为我们仅考虑了位姿，没有考虑点的更新。完整的 BA 需要用到矩阵的稀疏性，我们留到第七节课介绍。

假设一组点的 3D 坐标为 $\mathbf{P} = \{\mathbf{p}_i\}$，它们在相机中的坐标为 $\mathbf{U} = \{\mathbf{u}_i\}$，$\forall i = 1, \ldots n$。在文件 p3d.txt 和 p2d.txt 中给出了这两组点的值。同时，设待估计的位姿为 $\mathbf{T} \in \mathrm{SE}(3)$，内参矩阵为：

$$\mathbf{K} = \begin{bmatrix} 520.9 & 0 & 325.1 \\ 0 & 521.0 & 249.7 \\ 0 & 0 & 1 \end{bmatrix}.$$

请你根据上述条件，用 G-N 法求出最优位姿，初始估计为 $\mathbf{T}_0 = \mathbf{I}$。程序 GN-BA.cpp 文件提供了大致的框架，请填写剩下的内容。

在书写程序过程中，回答下列问题：

1. 如何定义重投影误差？
2. 该误差关于自变量的雅可比矩阵是什么？
3. 解出更新量之后，如何更新至之前的估计上？

作为验证，最后估计得到的位姿应该接近：

$$\mathbf{T}^* = \begin{bmatrix} 0.9978 & -0.0517 & 0.0399 & -0.1272 \\ 0.0506 & 0.9983 & 0.0274 & -0.007 \\ -0.0412 & -0.0253 & 0.9977 & 0.0617 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

这和书中使用 g2o 优化的结果很接近[3]。

[3]但是书中由于代码中错误地设置了 depth scale（应该为 5000，实际输入了 1000），所以应该说和修正后结果相近。

5 * 用 ICP 实现轨迹对齐（2 分，约 2 小时）

在实际当中，我们经常需要比较两条轨迹之间的误差。第三节课习题中，你已经完成了两条轨迹之间的 RMSE 误差计算。但是，由于 ground-truth 轨迹与相机轨迹很可能不在一个参考系中，它们得到的轨迹并不能直接比较。这时，我们可以用 ICP 来计算两条轨迹之间的相对旋转与平移，从而估计出两个参考系之间的差异。

图 3: vicon 运动捕捉系统，部署于场地中的多个红外相机会捕捉目标球的运动轨迹，实现快速定位。

设真实轨迹为 $\mathbf{T}_g$，估计轨迹为 $\mathbf{T}_e$，二者皆以 $\mathbf{T}_{WC}$ 格式存储。但是真实轨迹的坐标原点定义于外部某参考系中（取决于真实轨迹的采集方式，如 Vicon 系统可能以某摄像头中心为参考系，见图 3），而估计轨迹则以相机出发点为参考系（在视觉 SLAM 中很常见）。由于这个原因，理论上的真实轨迹点与估计轨迹点应满足：

$$\mathbf{T}_{g,i} = \mathbf{T}_{ge}\mathbf{T}_{e,i} \tag{4}$$

其中 i 表示轨迹中的第 i 条记录，$\mathbf{T}_{ge} \in \mathrm{SE}(3)$ 为两个坐标系之间的变换矩阵，该矩阵在整条轨迹中保持不变。$\mathbf{T}_{ge}$ 可以通过两条轨迹数据估计得到，但方法可能有若干种：

1. 认为初始化时两个坐标系的差异就是 $\mathbf{T}_{ge}$，即：

$$\mathbf{T}_{ge} = \mathbf{T}_{g,1}\mathbf{T}_{e,1}^{-1}. \tag{5}$$

2. 在整条轨迹上利用最小二乘计算 $\mathbf{T}_{ge}$：

$$\mathbf{T}_{ge} = \arg\min_{\mathbf{T}_{ge}} \sum_{i=1}^{n} \left\| \log\left(\mathbf{T}_{gi}^{-1}\mathbf{T}_{ge}\mathbf{T}_{e,i}\right)^{\vee} \right\|_2. \tag{6}$$

3. 把两条轨迹的平移部分看作点集，然后求点集之间的 ICP，得到两组点之间的变换。

其中第三种也是实践中用的最广的一种。现在请你书写 ICP 程序，估计两条轨迹之间的差异。轨迹文件在 compare.txt 文件中，格式为：

$$\mathrm{time}_e, \mathbf{t}_e, \mathbf{q}_e, \mathrm{time}_g, \mathbf{t}_g, \mathbf{q}_g,$$

其中 $\mathbf{t}$ 表示平移，$\mathbf{q}$ 表示单位四元数。请计算两条轨迹之间的变换，然后将它们统一到一个参考系，并画在 pangolin 中。轨迹的格式与先前相同，即以时间，平移，旋转四元数方式存储。

本题不提供代码框架，你可以利用之前的作业完成本题。图 4 显示了对准前与对准后的两条轨迹。

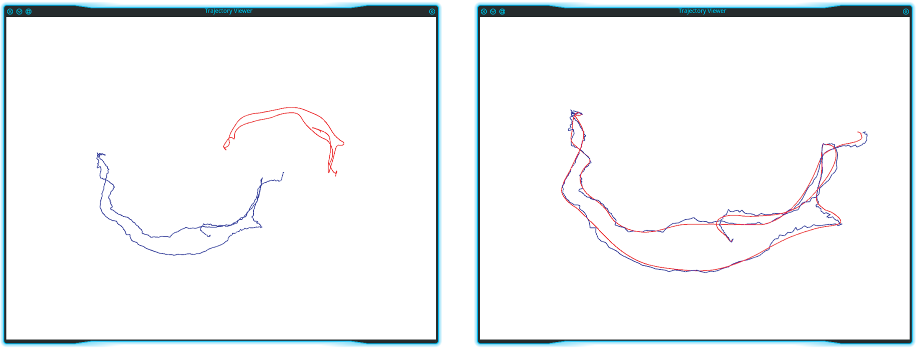

图 4: 轨迹对准前与对准后

Bibliography

[1] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in *2011 IEEE International Conference on Computer Vision (ICCV)*, pp. 2564–2571, IEEE, 2011.

[2] P. L. Rosin, "Measuring corner properties," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 291–307, 1999.