pcl_kinfu_app 目标函数修改设计报告

Changelog

版本号	变更人	变更说明	变更时间
V1.0	张琛	初稿	2015/01/18
V1.1	张琛	增加使用轮廓点集优化配准的实现方法描述	2015/01/23

目录

1. 引言

1.1. 编写目的

在理解 pcl_kinfu_app 程序的算法细节基础上，对现有的代码实现进行修改，在目标函数（cost function）上增加惩罚项，与原目标函数优化结果进行对比。

1.2. 问题描述

Kinect Fusion （pcl）开源实现中，用到 icp 算法对两个点集进行配准，求解点集之间的刚性变换 (R, t)。在 3D 场景的几何形状较为平凡时（e.g., 大的平面，对称的圆柱体、花瓶，etc.），icp 配准可能会出现累积偏移误差（drift）现象。可能需要修改原有的目标函数，增加新的惩罚项，约束优化空间进行求解，以消除偏移误差。

1.3. 任务目标

Pcl 中的 Kinect fusion 实现，采用了 POINT-TO-PLANE ICP 算法[1]，对物体（点云）的刚性变换进行求解。其目标函数为：

$$\widehat{\mathrm{M}}_{opt} = \arg\min \widehat{M} \, \Sigma \left(\left(\widehat{M} * s_i - d_i \right) * n_i \right)^2 \tag{1}$$

其中，$\mathrm{s_i} = \left(s_{ix}, s_{iy}, s_{iz}, 1\right)^T$是源数据点，$\mathrm{d_i} = \left(d_{ix}, d_{iy}, d_{iz}, 1\right)^T$是目标点集中的对应点；$n_\mathrm{i} = \left(n_{ix}, n_{iy}, n_{iz}, 0\right)^T$是 $\mathrm{d_i}$ 点处的单位法向量，M、$\mathrm{M_{opt}}$ 均为 4*4 3D 刚性变换矩阵。

我们尝试对式 (1) 进行修改：

1.3.1. 增加 RGB 颜色信息约束

目前有以下两种思路：

a) 若在 Σ **之外**增加一项，使目标函数变为：

$$\widehat{\mathrm{M}}_{opt} = \arg\min \widehat{M} \, [\Sigma \left(\left(\widehat{M} * s_i - d_i \right) * n_i \right)^2 + w * \Sigma \left(\mathrm{dist_{i,rgb}} \right)^2] \tag{2}$$

其中，$\left(\mathrm{dist_{i,rgb}}\right)^2$ 为第 i 对对应点之间的 RGB 空间的欧氏距离；w 为权重系数，用于平衡“+”前后两项间的权重比。

b) 若在 Σ **之内**进行修改，意味着仍使用 POINT-TO-PLANE ICP 相同的优化过程，所以约束信息必然是逐对应点的。这里仍使用 RGB 空间的欧氏距离，即将原来点集信息由 (x, y, z) 3D 空间扩展到 (x, y, z, r, g, b) 6D 空间进行 ICP 迭代。

~~此时，公式(1) 中，$\mathrm{s_i}, \mathrm{d_i}, \mathrm{n_i}$ 均为 6D 向量，~~

（没想通， 未解决）

输入：
1. OpenNI 驱动捕捉的 RGB-D 数据；
2. 修改过约束条件的目标函数；

输出：
1. 新的 (R, t) 序列；
2. 生成的 3D 场景模型；

1.3.2. 仅使用 depth 点云，但增加轮廓点的权重（杨学连之前的思路）

对原有点集 C 提取轮廓后，得到轮廓点集$\mathrm{C}^{\mathrm{outlier}}$，以及正常点$\mathrm{C}^{\mathrm{inlier}} = \mathrm{C} - \mathrm{C}^{\mathrm{outlier}}$ 。假设视窗中物体（或场景）的轮廓具有相对更高的精度，配准过程中，增加 C1 的权重 w：

$$\widehat{\mathrm{M}}_{opt} = \arg\min \widehat{M} \left[\Sigma\left(\left(\widehat{M} * s_i^{inl} - d_i^{inl}\right) * n_i^{inl}\right)^2 + w * \Sigma\left(\left(\widehat{M} * s_i^{outl} - d_i^{outl}\right) * n_i^{outl}\right)^2 \right. \quad (3)$$

其中，$\mathrm{s_i^{inl}}, d_i^{inl}, n_i^{inl}$, 分别为点集$\mathrm{C}^{\mathrm{inlier}}$对应的第 i 帧源数据点，对应点，法向量；$\mathrm{s_i^{outl}}, d_i^{outl}, n_i^{outl}$,则为$\mathrm{C}^{\mathrm{outlier}}$对应的第 i 帧源数据点，对应点，法向量.

1.4. 实现环境

OS: windows7 x64
IDE: visual studio 2010
Lang: C/C++
Device: XBOX 360 Kinect (Model 1414)
Drivers: OpenNI
3rd party LIBs: pcl, VTK, Boost, Eigen, FLANN, OpenNI

1.5. 参考文献

[1] Low K L. Linear least-squares optimization for point-to-plane icp surface registration[J]. Chapel Hill, University of North Carolina, 2004.
[2] Arun K S, Huang T S, Blostein S D. Least-squares fitting of two 3-D point sets[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 1987 (5): 698-700.

2. 实现步骤

原来的代码实现对公式(1)进行变换为：

$$\min_{x}|Ax-b|^2 \tag{4}$$

对应：

$$\mathrm{x}=\left(\alpha,\beta,\gamma,\mathrm{t_x},\mathrm{t_y},\mathrm{t_z}\right)^{\mathrm{T}} \tag{5}$$

$$\mathbf{A}=\begin{pmatrix} a_{11} & a_{12} & a_{13} & n_{1x} & n_{1y} & n_{1z} \\ a_{21} & a_{22} & a_{23} & n_{2x} & n_{2y} & n_{2z} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & n_{Nx} & n_{Ny} & n_{Nz} \end{pmatrix}, \text{其中} \quad \begin{aligned} a_{i1} &= n_{iz}s_{iy}-n_{iy}s_{iz}, \\ a_{i2} &= n_{ix}s_{iz}-n_{iz}s_{ix}, \\ a_{i3} &= n_{iy}s_{ix}-n_{ix}s_{iy}. \end{aligned} \tag{6}$$

$$\mathbf{b}=\begin{pmatrix} n_{1x}d_{1x}+n_{1y}d_{1y}+n_{1z}d_{1z}-n_{1x}s_{1x}-n_{1y}s_{1y}-n_{1z}s_{1z} \\ n_{2x}d_{2x}+n_{2y}d_{2y}+n_{2z}d_{2z}-n_{2x}s_{2x}-n_{2y}s_{2y}-n_{2z}s_{2z} \\ \vdots \\ n_{Nx}d_{Nx}+n_{Ny}d_{Ny}+n_{Nz}d_{Nz}-n_{Nx}s_{Nx}-n_{Ny}s_{Ny}-n_{Nz}s_{Nz} \end{pmatrix}. \tag{7}$$

则问题变为线性最小二乘问题。采用 SVD 分解方法得到：

$$\mathrm{x_{opt}}=\mathrm{A}^{+}b \tag{8}$$

进而得到 (R, t)

2.1. 增加 RGB 颜色信息约束的实现

对于 1.3.1-a) 提到的方法，由于增加了惩罚项 $w*\Sigma\left(\mathrm{dist_{i,rgb}}\right)^2]$，无法再转化为公式(4)~(8)的过程。可能的实现方案可以如下：

1. 设定初值 R=identity()；

2. 每次迭代，$\Sigma\left(\mathrm{dist_{i,rgb}}\right)^2$的计算方法为：记前一帧彩色图像为$\mathrm{F_{prev}}$，当前帧为$\mathrm{F_{curr}}$，上一帧得到的旋转为 $\mathrm{R_{prev}}$ ，则

$$\mathrm{dist_{i,rgb}}=R_{prev}*F_{prev}-F_{curr} \tag{9}$$

3. 采用自适应阈值方法，迭代得到较好的权重值 w。这里每次迭代需要评估结果，可能的方法有：

a) 使用 3D 扫描仪的 obj 点云作为 ground-truth 进行评估；

b) 以原来代码实现得到的 (R, t)序列作为 groundtruth 进行评估；

计划尝试使用优化算法库 levmar 进行实现。

2.2. 仅使用 depth 点云，但增加轮廓点的权重的实现

假设所有点权重相同，公式(1)可近似为公式(4)，采用线性最小二乘求解。而公式(3)中对轮廓点集$\mathrm{C^{outlier}}$赋予不同权重 w，等价于在公式(4)中，对 n*6 **矩阵 A 中的对应轮廓点的行**乘以 w，其余计算过程完全不变。因此，代码修改的主要工作集中在如何**标记出点集 C 中的轮**

廓点，对 cuda 中计算矩阵 A 的部分，判断某一行是否为轮廓点，如果是，乘以权重系数 w。

具体修改步骤如下：

1. 轮廓点提取：

采用 pcl 已有 api（关键词 outlier_removal）：

model_outlier_removal

radius_outlier_removal

statistical_outlier_removal

进行轮廓点提取，需要对可用的 API 效果对比测试（1day）

输入： raw-depth-data C

输出： a) C1, C2

b) **index_C2**，即 C2 在 C 中各点对应序号。

编码困难：

① pcl-oni-grabber 得到的是 organized datasets，能否直接得到 array of indices？

② 对于 unorganized datasets，能否直接得到 array of indices？不得不$O(N^2)$遍历吗？未解决

2. cpp 接口修改：

从 cpp 到 cu 文件的接口为 estimateCombined 函数：

```
void
estimateCombined (const Mat33& Rcurr, const float3& tcurr, const MapArr& vmap_curr, const MapArr& nmap_curr, const Mat33& Rprev_inv, const float3& tprev, const Intr& intr,
const MapArr& vmap_g_prev, const MapArr& nmap_g_prev, float distThres, float angleThres,
DeviceArray2D<float>& gbuf, DeviceArray<float>& mbuf, float* matrixA_host, float* vectorB_host);
```

其中， 传入数据主要为：vmap_curr，nmap_curr，vmap_g_prev，nmap_g_prev。需要将当前帧（curr）的 vmap 改为 vmap_inl, vmap_outl， nmap 改为 nmap_inl, nmap_outl；前一帧（prev）不用改。 轮廓点、正常点的计算已经由第一步求出。

3. GPU 求解矩阵 A, b 代码修改：

输入： C, **index_C2**（第一步求得）

操作： 向量 “float row[7];” 逐元素赋权值 w

输出： A, b 两矩阵（分别 6*6, 6*1）

3. 测试方法

分别使用原始 pcl_kinfu_app 以及改进后的 fusion 结果（obj mesh 文件）， 与 3D 扫描仪导出的 obj 点云进行 ICP 配准， 分别计算 RMS 误差。