

## ORB-SLAM2系统的是如何初始化的？

orb slam

ORB-SLAM2系统初始化可以总结做了两件事：

- 1、把读取到的图片数据给到跟踪线程
- 2、把保存跟踪生成的轨迹到文件(可视化轨迹看前面的文章)

## 源码解读

```
1  /**
2  定义ORB-SLAM2系统的主线程结构,其他各模块都由这里开始被调用
3  */
4  #include <string>
5  #include <opencv2/core/core.hpp>
6  #include "Tracking.h"
7  #include "FrameDrawer.h"
8  #include "MapDrawer.h"
9  #include "Map.h"
10 #include "LocalMapping.h"
11 #include "LoopClosing.h"
12 #include "KeyFrameDatabase.h"
13 #include "ORBvocabulary.h"
14 #include "Viewer.h"
15 #include "System.h"
16 #include "Converter.h"
17
18 #include <thread>
19 #include <pangolin/pangolin.h>
20 #include <iomanip>
21
22 namespace ORB_SLAM2
23 {
24 /**
25  * @param strVocFile
26  * @param strSettingsFile
27  * @param sensor
28  * @param buseViewer
29  * 传感器类型 eSensor mSensor
30 枚举类型用于表示本系统所使用的传感器类型
31  enum eSensor{
32      MONOCULAR=0,
33      STEREO=1,
34      RGBD=2
35  };
36  传感器类型
37  eSensor mSensor;
38  类变量前缀 m
39  指针类型变量前缀 p
40  进程前缀 t
41
42  * Initialize the SLAM system. It launches the Local Mapping, Loop Closing and Viewer th
43  构造函数初始化整个系统
44  * 创建可视化指针用Pangolin显示地图,相机位姿 Viewer* mpviewer
45  * mbActivateLocalizationMode bool 记录模式的变量
46  * mbDeactivateLocalizationMode bool 记录模式的变量
47  */
48  System::System(const string &strVocFile, //指定ORB字典文件的路径
49                  const string &strSettingsFile, //指定配置文件的路径
50                  const eSensor sensor, //指定配置文件的路径
51                  const bool buseViewer): //指定是否使用可视化界面
52      mSensor(sensor), //初始化传感器类型
53      mpViewer(static_cast<Viewer*>(NULL)), //可视化界面
54      mbReset(false), //复位标志
55      mbActivateLocalizationMode(false), //定位建图模式
56      mbDeactivateLocalizationMode(false) //定位模式
57  {
58      // 输出当前传感器类型
59      cout << "Input sensor was set to: ";
60
61      if(mSensor==MONOCULAR)
62          cout << "Monocular" << endl;
63      else if(mSensor==STEREO)
64          cout << "Stereo" << endl;
65      else if(mSensor==RGBD)
66          cout << "RGB-D" << endl;
67
68      // Check settings file
69      cv::FileStorage fsSettings(strSettingsFile.c_str(),cv::FileStorage::READ);
70      // 如果打开失败,输出调试信息
71      if(!fsSettings.isOpened()){
72          cerr << "Failed to open settings file at: " << strSettingsFile << endl;
73          exit(-1);
74      }
75
76      // Load ORB vocabulary
77      cout << endl << "Loading ORB vocabulary. This could take a while..." << endl;
78
79
80      /**
81      * ORB vocabulary used for place recognition and feature matching.
82      * 定义存储ORB字典的指针
83      * ORBVocabulary* mpVocabulary;
84      * 建立一个新的ORB字典
85      * typedef DBow2::TemplatedVocabulary<DBow2::FORB::TDescriptor, DBow2::FORB> ORBVoca
86      */
87      mpVocabulary = new ORBVocabulary();
88
89      // 获取字典加载状态
90      bool bvocLoad = mpVocabulary->loadFromTextFile(strVocFile);
91      // 如果加载失败,输出调试信息,然后退出
92      if(!bvocLoad){
93          cerr << "Wrong path to vocabulary. " << endl;
94          cerr << "Failed to open at: " << strVocFile << endl;
95          exit(-1);
96      }
97      // 加载成功
98      cout << "Vocabulary loaded!" << endl << endl;
99
100     //Create KeyFrame Database
101     /**
102     * KeyFrame database for place recognition (relocalization and loop detection).
103     * 定义存储关键帧数据库的指针,用于重定位和回环检测
104     * KeyFrameDatabase* mpKeyFrameDatabase;
105     * 构造函数
106     * KeyFrameDatabase::KeyFrameDatabase (const ORBVocabulary &voc):
107     */
108     mpKeyFrameDatabase = new KeyFrameDatabase(*mpVocabulary);
109
110     // Create the Map
111     /**
112     * Map structure that stores the pointers to all KeyFrames and MapPoints.
113     * 定义存储地图的指针,地图包括关键帧和地图点
114     * Map* mpMap;
115     * 地图构造函数
116     * Map::Map():mMaxKFid(0)
117     */
118     mpMap = new Map();
119
120     // Create Drawers. These are used by the Viewer
```

```

121 // 这里的帧绘制器和地图绘制器将会被可视化的viewer所使用
122 /**
123  * 创建帧显示指针
124  * FrameDrawer* mpFrameDrawer;
125  * FrameDrawer构造函数
126  * FrameDrawer::FrameDrawer(Map* pMap):mpMap(pMap)
127  */
128 mpFrameDrawer = new FrameDrawer(mpMap);
129
130 /**
131  * 创建地图显示指针
132  * MapDrawer* mpMapDrawer;
133  * MapDrawer构造函数
134  * MapDrawer::MapDrawer(Map* pMap, const string &strSettingPath):mpMap(pMap)
135  */
136 mpMapDrawer = new MapDrawer(mpMap, strSettingsFile);
137
138
139 /**
140  * Tracker. It receives a frame and computes the associated camera pose.
141  * It also decides when to insert a new keyframe, create some new MapPoints and
142  * performs relocalization if tracking fails.
143  * 跟踪接收帧和帧的位姿踪. 负责创建关键帧, 新地图点, 跟踪失败后的重定位.
144  * Tracking* mpTracker;
145  * 在主线程中初始化跟踪线程
146  * Initialize the Tracking thread
147  * (it will live in the main thread of execution, the one that called this construct
148  */
149 mpTracker = new Tracking(this,
150                          mpVocabulary,          // 字典
151                          mpFrameDrawer,        // 帧可视化
152                          mpMapDrawer,          // 地图可视化
153                          mpMap,                // 地图
154                          mpKeyFrameDatabase,    // 关键帧地图
155                          strSettingsFile,       // 设置文件路径
156                          mSensor);             // 传感器类型
157
158
159 /**
160  * Local Mapper. It manages the local map and performs local bundle adjustment.
161  * 创建局部建图指针器管理局部地图并且进行局部BA
162  * LocalMapping* mpLocalMapper;
163  * 初始化局部建图线程并运行
164  * Initialize the Local Mapping thread and launch
165  * LocalMapping构造函数
166  * LocalMapping::LocalMapping(Map *pMap, const float bMonocular):
167  */
168 mpLocalMapper = new LocalMapping(mpMap, mSensor==MONOCULAR);
169 /**
170  * 局部建图线程
171  * System threads: Local Mapping, Loop Closing, Viewer.
172  * The Tracking thread "lives" in the main execution thread that creates the System
173  * 系统在运行主进程跟踪的同时创建局部建图, 回环检测, 可视化线程
174  * std::thread* mptLocalMapping;
175  * std::thread* mptLoopClosing;
176  * std::thread* mptViewer;
177  */
178 mptLocalMapping = new thread(&ORB_SLAM2::LocalMapping::Run, mpLocalMapper);
179
180 /**
181  * Loop Closer. It searches loops with every new keyframe. If there is a loop it per
182  * a pose graph optimization and full bundle adjustment (in a new thread) afterwards
183  * 创建回环检测指针. 搜索回环执行位姿图优化并且开启新线程进行全局BA
184  * LoopClosing* mpLoopCloser;
185  * Initialize the Loop Closing thread and launch
186  * LoopClosing 构造函数
187  * LoopClosing::LoopClosing(Map *pMap, KeyFrameDatabase *pDB, ORBVocabulary *pVoc, c
188  */
189 mpLoopCloser = new LoopClosing(mpMap, mpKeyFrameDatabase, mpVocabulary, mSensor!=MONOCU
190
191 /**
192  * System threads: Local Mapping, Loop Closing, Viewer.
193  * The Tracking thread "lives" in the main execution thread that creates the System
194  * 系统在运行主进程跟踪的同时创建局部建图, 回环检测, 可视化线程
195  * std::thread* mptLocalMapping;
196  * std::thread* mptLoopClosing;
197  * std::thread* mptViewer;
198  * 创建回环检测线程
199  */
200 mptLoopClosing = new thread(&ORB_SLAM2::LoopClosing::Run, mpLoopCloser);
201
202 // Initialize the Viewer thread and launch
203 if(buseviewer){
204     mpViewer = new Viewer(this, mpFrameDrawer, mpMapDrawer, mpTracker, strSettingsFile);
205     // 新建viewer线程
206     mptViewer = new thread(&Viewer::Run, mpViewer);
207     // 给动跟设置对应的可视化
208     mpTracker->SetViewer(mpViewer);
209 }
210
211 // Set pointers between threads
212 // 设置进程间的指针
213 mpTracker->SetLocalMapper(mpLocalMapper);
214 mpTracker->SetLoopClosing(mpLoopCloser);
215
216 mpLocalMapper->SetTracker(mpTracker);
217 mpLocalMapper->SetLoopCloser(mpLoopCloser);
218
219 mpLoopCloser->SetTracker(mpTracker);
220 mpLoopCloser->SetLocalMapper(mpLocalMapper);
221 }
222
223 /**
224  * @brief 双目图像跟踪
225  * @param imLeft
226  * @param imRight
227  * @param timestamp
228  * @return
229  * 针对三种不同类型的传感器设计三种运动跟踪接口
230  * 彩色图像为CV_8UC3类型. 并且都将会被转换成灰度图像
231  * 跟踪接口返回估计的相机位姿. 如果跟踪失败则返回NULL
232  * Process the given stereo frame. Images must be synchronized(同步) and rectified(校正)
233  * Input images: RGB (CV_8UC3) or grayscale (CV_8U). RGB is converted to grayscale.
234  * Returns the camera pose (empty if tracking fails).
235  *
236  *
237  * const cv::Mat &imLeft,          //左目图像
238  * const cv::Mat &imRight,         //右目图像
239  * const double &timestamp;        //时间戳
240  */
241 cv::Mat System::TrackStereo(const cv::Mat &imLeft, const cv::Mat &imRight, const double
242 if(mSensor!=STEREO){
243     cerr << "ERROR: you called TrackStereo but input sensor was not set to STEREO."
244     exit(-1);
245 }
246
247 // Check mode change
248 {
249     unique_lock<mutex> lock(mMutexMode);
250     // 只开启定位模式
251     if(mbActivateLocalizationMode){
252         mpLocalMapper->RequestStop(); // 局部建图停止
253         // wait until Local Mapping has effectively stopped
254         while(!mpLocalMapper->isStopped()){
255             usleep(1000);
256         }
257         mpTracker->InformOnlyTracking(true); // 只跟踪
258         mbActivateLocalizationMode = false; // 设置定位模式状态
259     }
260     // 定位模式和建图模式同时开启
261     if(mbDeactivateLocalizationMode){
262         mpTracker->InformOnlyTracking(false); // 开启地图构建和跟踪模式
263         mpLocalMapper->Release(); // 局部建图工作

```

```

265     }
266 }
267
268 // Check reset, 检查是否有复位的操作
269 // Reset Flag 复位标志
270 // std::mutex mMutexReset;
271 // bool mBReset;
272 {
273     unique_lock<mutex> lock(mMutexReset); //上锁
274     //是否有复位请求
275     if(mBReset){
276         mpTracker->Reset(); //有, 跟踪复位
277         mBReset = false; // 清除标志
278     }
279 }
280
281 /**
282  * Tracker. It receives a frame and computes the associated camera pose.
283  * It also decides when to insert a new keyframe, create some new MapPoints and
284  * performs relocalization if tracking fails.
285  * 跟踪接收帧和帧的位姿. 负责创建关键帧, 新地图点, 跟踪失败后的重定位.
286  * Tracking* mpTracker;
287  *
288  * 输入左右目图像. 可以为RGB, BGR, RGBA, GRAY
289  * 1. 将图像转为mImGray和imGrayRight并初始化mCurrentFrame
290  * 2. 进行tracking过程
291  * 输出世界坐标系到该帧相机坐标系的变换矩阵
292  * cv::Mat Tracking::GrabImageStereo(
293  * const cv::Mat &imRectLeft, //左侧图像
294  * const cv::Mat &imRectRight, //右侧图像
295  * const double &timestamp) //时间戳
296  *
297  * 用矩阵Tcw来保存估计的相机位姿. GrabImageStereo运动估计函数
298  */
299 cv::Mat Tcw = mpTracker->GrabImageStereo(imLeft, imRight, timestamp);
300
301 // 运动跟踪状态上锁
302 unique_lock<mutex> lock2(mMutexState);
303
304 /**
305  * 获取运动跟踪状态
306  * Tracker. It receives a frame and computes the associated camera pose.
307  * It also decides when to insert a new keyframe, create some new MapPoints and
308  * performs relocalization if tracking fails.
309  * 跟踪接收帧和帧的位姿. 负责创建关键帧, 新地图点, 跟踪失败后的重定位.
310  * Tracking* mpTracker;
311  */
312 mTrackingState = mpTracker->mState;
313 // 获取当前帧跟踪到的地图点向量指针
314 mTrackedMapPoints = mpTracker->mCurrentFrame.mvpMapPoints;
315 // 获取当前帧跟踪到的关键帧特征点向量的指针
316 mTrackedKeyPointsUn = mpTracker->mCurrentFrame.mvKeysun;
317 // 返回获得的相机运动估计
318 return Tcw;
319 }
320
321 /**
322  * @brief RGBD图像跟踪
323  * @param im
324  * @param depthmap
325  * @param timestamp
326  * @return
327  */
328 * Process the given rgbd frame. Depthmap must be registered(配准|匹配) to the RGB frame.
329 * Input image: RGB (CV_8UC3) or grayscale (CV_8U). RGB is converted to grayscale.
330 * Input depthmap: Float (CV_32F).
331 * Returns the camera pose (empty if tracking fails).
332  *
333  * const cv::Mat &im, //彩色图像
334  * const cv::Mat &depthmap, //深度图像
335  * const double &timestamp; //时间戳
336  */
337 cv::Mat System::TrackRGBD(const cv::Mat &im, const cv::Mat &depthmap, const double &time
338 {
339     if(mSensor!=RGBD){
340         cerr << "ERROR: you called TrackRGBD but input sensor was not set to RGBD." << endl;
341         exit(-1);
342     }
343
344     // Check mode change
345     // Change mode flags 记录模式的变量
346     // std::mutex mMutexMode;
347     // bool mBActivateLocalizationMode;
348     // bool mBDeactivateLocalizationMode;
349     {
350         unique_lock<mutex> lock(mMutexMode);
351         if(mBActivateLocalizationMode){
352             // Local Mapper. It manages the local map and performs local bundle adjustment
353             // 局部建图和局部BA(bundle adjustment)
354             mpLocalMapper->RequestStop();
355
356             // wait until Local Mapping has effectively stopped
357             while(!mpLocalMapper->isStopped()){
358                 usleep(1000);
359             }
360
361             mpTracker->InformOnlyTracking(true);
362             mBActivateLocalizationMode = false;
363         }
364         if(mBDeactivateLocalizationMode){
365             mpTracker->InformOnlyTracking(false);
366             mpLocalMapper->Release();
367             mBDeactivateLocalizationMode = false;
368         }
369     }
370
371     // Check reset 检查复位
372     {
373         unique_lock<mutex> lock(mMutexReset);
374         if(mBReset){
375             mpTracker->Reset();
376             mBReset = false;
377         }
378     }
379
380     /**
381     * 获得相机位姿的估计
382     * cv::Mat Tracking::GrabImagerGBD(
383     * const cv::Mat &imRGB, //彩色图像
384     * const cv::Mat &imD, //深度图像
385     * const double &timestamp) //时间戳
386     * 输入左目RGB或RGBA图像和深度图
387     * 将图像转为mImGray和imDepth初始化mCurrentFrame
388     * 进行tracking过程
389     * 输出世界坐标系到该帧相机坐标系的变换矩阵
390     */
391     cv::Mat Tcw = mpTracker->GrabImagerGBD(im, depthmap, timestamp);
392
393     unique_lock<mutex> lock2(mMutexState);
394
395     // 跟踪状态 eTrackingState mState;
396     // Tracking state 记录跟踪状态
397     // int mTrackingState;
398     mTrackingState = mpTracker->mState;
399     /**
400     * MapPoints associated to keypoints, NULL pointer if no association.
401     * 每个特征点keypoints对应地图点MapPoint.
402     * 如果特征点没有对应的地图点, 将存储一个空指针
403     * std::vector<MapPoint*> mvpMapPoints;
404     * Frame mCurrentFrame; 跟踪线程的一个Current Frame(当前帧)
405     * std::vector<MapPoint*> mTrackedMapPoints;
406     * std::vector<cv::KeyPoint> mTrackedKeyPointsUn;

```

```

408 mTrackedMapPoints = mpTracker->mCurrentFrame.mvpMapPoints;
409 // std::vector<cv::KeyPoint> mvKeysUn; 校正mvkeys后的特征点
410 mTrackedKeyPointsUn = mpTracker->mCurrentFrame.mvKeysUn;
411 return Tcw;
412 }
413
414 /**
415  * @brief 单目图像跟踪
416  * @param im
417  * @param timestamp
418  * @return
419  *
420  * Process the given monocular frame
421  * Input images: RGB (CV_8UC3) or grayscale (CV_8U). RGB is converted to grayscale.
422  * Returns the camera pose (empty if tracking fails).
423  *
424  * const cv::Mat &im, //图像
425  * const double &timestamp); //时间戳
426 */
427 cv::Mat System::TrackMonocular(const cv::Mat &im, const double &timestamp){
428     if(mSensor!=MONOCULAR){
429         cerr << "ERROR: you called TrackMonocular but input sensor was not set to Monocular\n";
430         exit(-1);
431     }
432
433     // Check mode change
434     {
435         // 独占锁, 主要是为了mbActivateLocalizationMode和mbDeactivateLocalizationMode不会发生冲突
436         unique_lock<mutex> lock(mMutexMode);
437         // mbActivateLocalizationMode为true会关闭局部地图线程
438         if(mbActivateLocalizationMode){
439             mLocalMapper->RequestStop();
440             // wait until Local Mapping has effectively stopped
441             while(!mLocalMapper->isStopped()){
442                 usleep(1000);
443             }
444
445             // 局部地图关闭以后, 只进行追踪的线程. 只计算相机的位姿, 没有对局部地图进行更新
446             // 设置mbOnlyTracking为真
447             mpTracker->InformOnlyTracking(true);
448             // 关闭线程可以使别的线程得到更多的资源
449             mbActivateLocalizationMode = false;
450         }
451         // 如果mbDeactivateLocalizationMode=true, 局部地图线程就被释放, 关键帧从局部地图中删除
452         if(mbDeactivateLocalizationMode){
453             mpTracker->InformOnlyTracking(false);
454             mLocalMapper->Release();
455             mbDeactivateLocalizationMode = false;
456         }
457     }
458
459     // Check reset
460     {
461         unique_lock<mutex> lock(mMutexReset);
462         if(mbReset){
463             mpTracker->Reset();
464             mbReset = false;
465         }
466     }
467
468     // 获取相机位姿的估计结果
469     cv::Mat Tcw = mpTracker->GrabImageMonocular(im,timestamp);
470     unique_lock<mutex> lock2(mMutexState);
471     mTrackingState = mpTracker->mState;
472     mTrackedMapPoints = mpTracker->mCurrentFrame.mvpMapPoints;
473     mTrackedKeyPointsUn = mpTracker->mCurrentFrame.mvKeysUn;
474
475     return Tcw;
476 }
477
478 /**
479  * @brief 按照TUM格式保存相机运行轨迹并保存到指定的文件
480  * @param filename 用来存储轨迹的文件名字
481  * Save camera trajectory in the TUM RGB-D dataset format.
482  * NOTE Only for stereo and RGB-D. This method does not work for monocular.
483  * Call first shutdown()
484  * See format details at: http://vision.in.tum.de/data/datasets/rgbd-dataset
485  * 保存相机的运动轨迹(TUM数据集的格式, 保存轨迹操作要在Shutdown函数之后调用
486 */
487 void System::SaveTrajectoryTUM(const string &filename){
488     cout << endl << "Saving camera trajectory to " << filename << " ..." << endl;
489     // 只有在传感器为双目或者RGBD时才可以工作
490     if(mSensor==MONOCULAR){
491         cerr << "ERROR: SaveTrajectoryTUM cannot be used for monocular." << endl;
492         return;
493     }
494
495     // 从地图中获取所有的关键帧
496     vector<KeyFrame*> vpKFs = mpMap->GetAllKeyFrames();
497     // 根据关键帧生成的先后顺序(1id)进行排序
498     sort(vpKFs.begin(), vpKFs.end(), KeyFrame::1id);
499
500     // Transform all keyframes so that the first keyframe is at the origin.
501     // After a loop closure the first keyframe might not be at the origin.
502     // 获取第一个关键帧的位姿的逆, 并将第一帧作为世界坐标系
503     cv::Mat Two = vpKFs[0]->GetPoseInverse();
504
505     // 文件写入的准备工作
506     ofstream f;
507     f.open(filename.c_str());
508     // 输出浮点数的时候不使用科学计数法
509     f << fixed;
510
511
512     /**
513      * Frame pose is stored relative to its reference keyframe (which is optimized by bundle adjustment)
514      * We need to get first the keyframe pose and then concatenate the relative transformation
515      * Frames not localized (tracking failure) are not saved.
516      * 之前的帧位姿都是基于其参考关键帧的, 现在我们把它恢复
517      *
518      * For each frame we have a reference keyframe (lrit), the timestamp (lt) and a
519      * which is true when tracking failed (lbl).
520      * 参考关键帧列表
521      */
522     list<ORB_SLAM2::KeyFrame*>::iterator lrit = mpTracker->m1References.begin();
523     // 所有帧对应的时间戳列表
524     list<double>::iterator lt = mpTracker->m1FrameTimes.begin();
525     // 每帧的追踪状态组成的列表
526     list<bool>::iterator lbl = mpTracker->m1Lost.begin();
527     // 对于每一个m1RelativeFramePoses中的帧lit
528     for(list<cv::Mat>::iterator lit=mpTracker->m1RelativeFramePoses.begin(),
529         lend=mpTracker->m1RelativeFramePoses.end();
530         lit!=lend;
531         lit++, lrit++, lt++, lbl++)
532     {
533         // 如果该帧跟踪失败, 进行下一个
534         if(*lbl)
535             continue;
536
537         // 获取其对应的参考关键帧
538         KeyFrame* pKF = *lrit;
539
540         // 变换矩阵的初始化, 初始化为一个单位阵
541         cv::Mat Trw = cv::Mat::eye(4,4,CV_32F);
542
543         // If the reference keyframe was culled(删除), traverse the spanning tree to
544         // 查看当前使用的参考关键帧是否为bad
545         while(pKF->isBad()){
546             // 更新关键帧变换矩阵的初始值.
547             Trw = Trw*pKF->m1Tcp;
548             // 并且更新到原关键帧的父关键帧
549             pKF = pKF->GetParent();
550         }
551     }
552 }

```

```

550     }
551     // 最后一个Two是原点校正
552     // 最终得到的是参考关键帧相对于世界坐标系的变换
553     Trw = Trw*pkF->GetPose()*Two;
554
555     // 在此基础上得到相机当前帧相对于世界坐标系的变换
556     cv::Mat Tcw = (*1it)*Trw;
557     // 然后分解出旋转矩阵
558     cv::Mat Rwc = Tcw.rowRange(0,3).colRange(0,3).t();
559     // 分解出平移向量
560     cv::Mat twc = -Rwc*Tcw.rowRange(0,3).col(3);
561
562     // 用四元数表示旋转
563     vector<Float> q = Converter::toQuaternion(Rwc);
564
565     // 然后按照给定的格式输出到文件中
566     f << setprecision(6) << "lt << " << setprecision(9) << twc.at<float>(0) << " " << q[0] << " " << q[1] << " " << q[2] << " " << q[3] << endl;
567 }
568
569 // 关闭文件
570 f.close();
571 cout << endl << "trajectory saved!" << endl;
572 }
573
574 /**
575  * 保存关键帧的轨迹
576  * @param filename 用来存储轨迹的文件名字
577  * Save keyframe poses in the TUM RGB-D dataset format.
578  * NOTE This method works for all sensor input.
579  * Call first shutdown()
580  * See format details at: http://vision.in.tum.de/datasets/rgbd-dataset
581  * 保存相机的运动轨迹(TUM数据集的格式, 保存轨迹操作要在shutdown函数之后调用
582  */
583 void System::SaveKeyFrameTrajectoryTUM(const string &filename){
584     cout << endl << "Saving keyframe trajectory to " << filename << " ..." << endl;
585
586     //获取关键帧vector并按照生成时间对其进行排序
587     vector<KeyFrame*> vpKFs = mMap->GetAllKeyFrames();
588     sort(vpKFs.begin(), vpKFs.end(), KeyFrame::lId);
589
590     // Transform all keyframes so that the first keyframe is at the origin.
591     // After a loop closure the first keyframe might not be at the origin.
592     // cv::Mat Two = vpKFs[0]->GetPoseInverse();
593     ofstream f;
594     f.open(filename.c_str());
595     f << fixed;
596
597     // 遍历关键帧
598     for(size_t i=0; i<vpKFs.size(); i++)
599     {
600         // 获取该关键帧
601         KeyFrame* pKF = vpKFs[i];
602
603         // pKF->SetPose(pKF->GetPose()*Two);
604
605         if(pKF->isBad())
606             continue;
607
608         // 抽取旋转部分和平移部分, 前者使用四元数表示
609         cv::Mat R = pKF->GetRotation().t();
610         vector<float> q = Converter::toQuaternion(R);
611         cv::Mat t = pKF->GetCameraCenter();
612         //按照给定的格式输出到文件中
613         f << setprecision(6) << pKF->mTimeStamp << setprecision(7) << " " << t.at<float>(0) << " " << q[0] << " " << q[1] << " " << q[2] << " " << q[3] << endl;
614
615     }
616     f.close();
617     cout << endl << "trajectory saved!" << endl;
618 }
619
620 /**
621  * @brief 按照KITTI数据集的格式将相机的运动轨迹保存到文件中
622  * @param filename 用来存储轨迹的文件名字
623  * Save camera trajectory in the KITTI dataset format.
624  * NOTE Only for stereo and RGB-D. This method does not work for monocular.
625  * Call first shutdown()
626  * See format details at: http://www.cvlibs.net/datasets/kitti/eval_odometry.php
627  * 保存相机的运动轨迹(以KITTI数据集的格式, 保存轨迹操作要在shutdown函数之后调用
628  */
629 void System::SaveTrajectoryKITTI(const string &filename){
630     cout << endl << "Saving camera trajectory to " << filename << " ..." << endl;
631     if(mSensor==MONOCULAR){
632         cerr << "ERROR: SaveTrajectoryKITTI cannot be used for monocular." << endl;
633         return;
634     }
635     vector<KeyFrame*> vpKFs = mMap->GetAllKeyFrames();
636     sort(vpKFs.begin(), vpKFs.end(), KeyFrame::lId);
637
638     // Transform all keyframes so that the first keyframe is at the origin.
639     // After a loop closure the first keyframe might not be at the origin.
640     cv::Mat Two = vpKFs[0]->GetPoseInverse();
641
642     ofstream f;
643     f.open(filename.c_str());
644     f << fixed;
645
646     // Frame pose is stored relative to its reference keyframe (which is optimized by bundle adjustment)
647     // We need to get first the keyframe pose and then concatenate the relative transformation
648     // Frames not localized (tracking failure) are not saved.
649
650     // For each frame we have a reference keyframe (lrit), the timestamp (lt) and a
651     // which is true when tracking failed (lbl).
652     list<ORB_SLAM2::KeyFrame*>::iterator lrit = mpTracker->mIpReferences.begin();
653     list<double>::iterator lt = mpTracker->mIframeTimes.begin();
654     for(list<cv::Mat>::iterator lit=mpTracker->mIrelativeFramePoses.begin(), lend=mpTracker->mIrelativeFramePoses.end(); lit!=lend; ++lit, ++lrit, ++lt)
655     {
656         ORB_SLAM2::KeyFrame* pKF = *lrit;
657
658         cv::Mat Trw = cv::Mat::eye(4,4,CV_32F);
659
660         while(pKF->isBad()){
661             Trw = Trw*pKF->mTcp;
662             pKF = pKF->GetParent();
663         }
664
665         Trw = Trw*pKF->GetPose()*Two;
666
667         cv::Mat Tcw = (*1it)*Trw;
668         cv::Mat Rwc = Tcw.rowRange(0,3).colRange(0,3).t();
669         cv::Mat twc = -Rwc*Tcw.rowRange(0,3).col(3);
670
671         f << setprecision(9) << Rwc.at<float>(0,0) << " " << Rwc.at<float>(0,1) << " " << Rwc.at<float>(0,2) << " " << Rwc.at<float>(0,3) << " " << twc.at<float>(0,0) << " " << twc.at<float>(0,1) << " " << twc.at<float>(0,2) << " " << twc.at<float>(0,3) << endl;
672     }
673     f.close();
674     cout << endl << "trajectory saved!" << endl;
675 }
676
677 /**
678  * This stops local mapping thread (map building) and performs only camera tracking.
679  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
680  */
681 void System::ActivateLocalizationMode(){
682     unique_lock<mutex> lock(mMutexMode); // 上锁
683     mActivateLocalizationMode = true; // 设置标志
684 }
685
686 /**
687  * This stops local mapping thread (map building) and performs only camera tracking.
688  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
689  */
690 void System::DeactivateLocalizationMode(){
691     unique_lock<mutex> lock(mMutexMode); // 上锁
692     mActivateLocalizationMode = false; // 设置标志
693 }
694
695 /**
696  * This stops local mapping thread (map building) and performs only camera tracking.
697  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
698  */
699 void System::DeactivateLocalizationMode(){
700     unique_lock<mutex> lock(mMutexMode); // 上锁
701     mActivateLocalizationMode = false; // 设置标志
702 }
703
704 /**
705  * This stops local mapping thread (map building) and performs only camera tracking.
706  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
707  */
708 void System::DeactivateLocalizationMode(){
709     unique_lock<mutex> lock(mMutexMode); // 上锁
710     mActivateLocalizationMode = false; // 设置标志
711 }
712
713 /**
714  * This stops local mapping thread (map building) and performs only camera tracking.
715  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
716  */
717 void System::DeactivateLocalizationMode(){
718     unique_lock<mutex> lock(mMutexMode); // 上锁
719     mActivateLocalizationMode = false; // 设置标志
720 }
721
722 /**
723  * This stops local mapping thread (map building) and performs only camera tracking.
724  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
725  */
726 void System::DeactivateLocalizationMode(){
727     unique_lock<mutex> lock(mMutexMode); // 上锁
728     mActivateLocalizationMode = false; // 设置标志
729 }
730
731 /**
732  * This stops local mapping thread (map building) and performs only camera tracking.
733  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
734  */
735 void System::DeactivateLocalizationMode(){
736     unique_lock<mutex> lock(mMutexMode); // 上锁
737     mActivateLocalizationMode = false; // 设置标志
738 }
739
740 /**
741  * This stops local mapping thread (map building) and performs only camera tracking.
742  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
743  */
744 void System::DeactivateLocalizationMode(){
745     unique_lock<mutex> lock(mMutexMode); // 上锁
746     mActivateLocalizationMode = false; // 设置标志
747 }
748
749 /**
750  * This stops local mapping thread (map building) and performs only camera tracking.
751  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
752  */
753 void System::DeactivateLocalizationMode(){
754     unique_lock<mutex> lock(mMutexMode); // 上锁
755     mActivateLocalizationMode = false; // 设置标志
756 }
757
758 /**
759  * This stops local mapping thread (map building) and performs only camera tracking.
760  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
761  */
762 void System::DeactivateLocalizationMode(){
763     unique_lock<mutex> lock(mMutexMode); // 上锁
764     mActivateLocalizationMode = false; // 设置标志
765 }
766
767 /**
768  * This stops local mapping thread (map building) and performs only camera tracking.
769  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
770  */
771 void System::DeactivateLocalizationMode(){
772     unique_lock<mutex> lock(mMutexMode); // 上锁
773     mActivateLocalizationMode = false; // 设置标志
774 }
775
776 /**
777  * This stops local mapping thread (map building) and performs only camera tracking.
778  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
779  */
780 void System::DeactivateLocalizationMode(){
781     unique_lock<mutex> lock(mMutexMode); // 上锁
782     mActivateLocalizationMode = false; // 设置标志
783 }
784
785 /**
786  * This stops local mapping thread (map building) and performs only camera tracking.
787  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
788  */
789 void System::DeactivateLocalizationMode(){
790     unique_lock<mutex> lock(mMutexMode); // 上锁
791     mActivateLocalizationMode = false; // 设置标志
792 }
793
794 /**
795  * This stops local mapping thread (map building) and performs only camera tracking.
796  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
797  */
798 void System::DeactivateLocalizationMode(){
799     unique_lock<mutex> lock(mMutexMode); // 上锁
800     mActivateLocalizationMode = false; // 设置标志
801 }
802
803 /**
804  * This stops local mapping thread (map building) and performs only camera tracking.
805  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
806  */
807 void System::DeactivateLocalizationMode(){
808     unique_lock<mutex> lock(mMutexMode); // 上锁
809     mActivateLocalizationMode = false; // 设置标志
810 }
811
812 /**
813  * This stops local mapping thread (map building) and performs only camera tracking.
814  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
815  */
816 void System::DeactivateLocalizationMode(){
817     unique_lock<mutex> lock(mMutexMode); // 上锁
818     mActivateLocalizationMode = false; // 设置标志
819 }
820
821 /**
822  * This stops local mapping thread (map building) and performs only camera tracking.
823  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
824  */
825 void System::DeactivateLocalizationMode(){
826     unique_lock<mutex> lock(mMutexMode); // 上锁
827     mActivateLocalizationMode = false; // 设置标志
828 }
829
830 /**
831  * This stops local mapping thread (map building) and performs only camera tracking.
832  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
833  */
834 void System::DeactivateLocalizationMode(){
835     unique_lock<mutex> lock(mMutexMode); // 上锁
836     mActivateLocalizationMode = false; // 设置标志
837 }
838
839 /**
840  * This stops local mapping thread (map building) and performs only camera tracking.
841  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
842  */
843 void System::DeactivateLocalizationMode(){
844     unique_lock<mutex> lock(mMutexMode); // 上锁
845     mActivateLocalizationMode = false; // 设置标志
846 }
847
848 /**
849  * This stops local mapping thread (map building) and performs only camera tracking.
850  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
851  */
852 void System::DeactivateLocalizationMode(){
853     unique_lock<mutex> lock(mMutexMode); // 上锁
854     mActivateLocalizationMode = false; // 设置标志
855 }
856
857 /**
858  * This stops local mapping thread (map building) and performs only camera tracking.
859  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
860  */
861 void System::DeactivateLocalizationMode(){
862     unique_lock<mutex> lock(mMutexMode); // 上锁
863     mActivateLocalizationMode = false; // 设置标志
864 }
865
866 /**
867  * This stops local mapping thread (map building) and performs only camera tracking.
868  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
869  */
870 void System::DeactivateLocalizationMode(){
871     unique_lock<mutex> lock(mMutexMode); // 上锁
872     mActivateLocalizationMode = false; // 设置标志
873 }
874
875 /**
876  * This stops local mapping thread (map building) and performs only camera tracking.
877  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
878  */
879 void System::DeactivateLocalizationMode(){
880     unique_lock<mutex> lock(mMutexMode); // 上锁
881     mActivateLocalizationMode = false; // 设置标志
882 }
883
884 /**
885  * This stops local mapping thread (map building) and performs only camera tracking.
886  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
887  */
888 void System::DeactivateLocalizationMode(){
889     unique_lock<mutex> lock(mMutexMode); // 上锁
890     mActivateLocalizationMode = false; // 设置标志
891 }
892
893 /**
894  * This stops local mapping thread (map building) and performs only camera tracking.
895  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
896  */
897 void System::DeactivateLocalizationMode(){
898     unique_lock<mutex> lock(mMutexMode); // 上锁
899     mActivateLocalizationMode = false; // 设置标志
900 }
901
902 /**
903  * This stops local mapping thread (map building) and performs only camera tracking.
904  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
905  */
906 void System::DeactivateLocalizationMode(){
907     unique_lock<mutex> lock(mMutexMode); // 上锁
908     mActivateLocalizationMode = false; // 设置标志
909 }
910
911 /**
912  * This stops local mapping thread (map building) and performs only camera tracking.
913  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
914  */
915 void System::DeactivateLocalizationMode(){
916     unique_lock<mutex> lock(mMutexMode); // 上锁
917     mActivateLocalizationMode = false; // 设置标志
918 }
919
920 /**
921  * This stops local mapping thread (map building) and performs only camera tracking.
922  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
923  */
924 void System::DeactivateLocalizationMode(){
925     unique_lock<mutex> lock(mMutexMode); // 上锁
926     mActivateLocalizationMode = false; // 设置标志
927 }
928
929 /**
930  * This stops local mapping thread (map building) and performs only camera tracking.
931  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
932  */
933 void System::DeactivateLocalizationMode(){
934     unique_lock<mutex> lock(mMutexMode); // 上锁
935     mActivateLocalizationMode = false; // 设置标志
936 }
937
938 /**
939  * This stops local mapping thread (map building) and performs only camera tracking.
940  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
941  */
942 void System::DeactivateLocalizationMode(){
943     unique_lock<mutex> lock(mMutexMode); // 上锁
944     mActivateLocalizationMode = false; // 设置标志
945 }
946
947 /**
948  * This stops local mapping thread (map building) and performs only camera tracking.
949  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
950  */
951 void System::DeactivateLocalizationMode(){
952     unique_lock<mutex> lock(mMutexMode); // 上锁
953     mActivateLocalizationMode = false; // 设置标志
954 }
955
956 /**
957  * This stops local mapping thread (map building) and performs only camera tracking.
958  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
959  */
960 void System::DeactivateLocalizationMode(){
961     unique_lock<mutex> lock(mMutexMode); // 上锁
962     mActivateLocalizationMode = false; // 设置标志
963 }
964
965 /**
966  * This stops local mapping thread (map building) and performs only camera tracking.
967  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
968  */
969 void System::DeactivateLocalizationMode(){
970     unique_lock<mutex> lock(mMutexMode); // 上锁
971     mActivateLocalizationMode = false; // 设置标志
972 }
973
974 /**
975  * This stops local mapping thread (map building) and performs only camera tracking.
976  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
977  */
978 void System::DeactivateLocalizationMode(){
979     unique_lock<mutex> lock(mMutexMode); // 上锁
980     mActivateLocalizationMode = false; // 设置标志
981 }
982
983 /**
984  * This stops local mapping thread (map building) and performs only camera tracking.
985  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
986  */
987 void System::DeactivateLocalizationMode(){
988     unique_lock<mutex> lock(mMutexMode); // 上锁
989     mActivateLocalizationMode = false; // 设置标志
990 }
991
992 /**
993  * This stops local mapping thread (map building) and performs only camera tracking.
994  * 激活定位模式 局部建图部分不工作载入已有的地图 仅有跟踪部分工作
995  */
996 void System::DeactivateLocalizationMode(){
997     unique_lock<mutex> lock(mMutexMode); // 上锁
998     mActivateLocalizationMode = false; // 设置标志
999 }
1000

```

```

693 * This resumes local mapping thread and performs SLAM again.
694 * 取消只定位模式 局部建图部分工作, 跟踪部分工作.
695 */
696 void System::DeactivateLocalizationMode(){
697     unique_lock<mutex> lock(mMutexMode);
698     mbDeactivateLocalizationMode = true;
699 }
700
701 /**
702  * Returns true if there have been a big map change (loop closure, global BA)
703  * since last call to this function
704  * 记录地图是否改变的变量
705  */
706 bool System::MapChanged(){
707     static int n=0;
708     //其实整个函数功能实现的重点还是在这个GetLastBigChangeIdx函数上
709     int curr = mpMap->GetLastBigChangeIdx();
710     if(n<curr){
711         n=curr;
712         return true;
713     }
714     else
715         return false;
716 }
717
718 /**
719  * Reset the system (clear map)
720  * 复位SLAM系统就是清空地图
721  */
722 void System::Reset(){
723     unique_lock<mutex> lock(mMutexReset);
724     mbReset = true;
725 }
726
727 /**
728  * All threads will be requested to finish.
729  * It waits until all threads have finished.
730  * This function must be called before saving the trajectory.
731  * 关闭SLAM系统的所有线程, 然后保存轨迹到地图.
732  */
733 void System::Shutdown(){
734     // 对局部建图线程和闭环检测线程发送终止请求
735     mpLocalMapper->RequestFinish();
736     mpLoopCloser->RequestFinish();
737     // 使用可视化窗口
738     if(mpviewer){
739         // 向可视化窗口发送终止请求
740         mpviewer->RequestFinish();
741         while(!mpviewer->isFinished())
742             usleep(5000);
743     }
744     // wait until all thread have effectively stopped
745     while(!mpLocalMapper->isFinished() ||
746         !mpLoopCloser->isFinished() ||
747         mpLoopCloser->isRunningGBA())
748     {
749         usleep(5000);
750     }
751
752     if(mpviewer)
753         // 可视化窗口
754         pangolin::BindToContext("ORB-SLAM2: Map Viewer");
755 }
756
757 /**
758  * 获取当前帧的跟踪状态
759  * Information from most recent processed frame
760  * You can call this right after TrackMonocular (or stereo or RGBD)
761  * 记录最近的帧的跟踪状态 int GetTrackingState();
762  */
763 * std::mutex mMutexState;
764 *
765 * Tracking state 记录跟踪状态 int mTrackingState;
766 */
767 int System::GetTrackingState(){
768     unique_lock<mutex> lock(mMutexState);
769     return mTrackingState;
770 }
771
772 /**
773  * 获取跟踪到的地图点
774  * std::mutex mMutexState;
775  * std::vector<MapPoint*> mTrackedMapPoints;
776  * std::vector<MapPoint*> GetTrackedMapPoints();
777  */
778 vector<MapPoint*> System::GetTrackedMapPoints(){
779     unique_lock<mutex> lock(mMutexState);
780     return mTrackedMapPoints;
781 }
782
783 /**
784  * 获取跟踪到的关键帧上的关键点
785  * std::vector<cv::KeyPoint> GetTrackedKeyPointsUn();
786  * std::vector<cv::KeyPoint> mTrackedKeyPointsUn;
787  */
788 * The keypoint is characterized by the 2D position,
789 * a point feature found by one of many available keypoint detectors,
790 * such as cv::FAST, cv::StarDetector, cv::SURF, cv::SIFT,
791 */
792 vector<cv::KeyPoint> System::GetTrackedKeyPointsUn(){
793     unique_lock<mutex> lock(mMutexState);
794     return mTrackedKeyPointsUn;
795 }
796
797 } //namespace ORB_SLAM

```