实验 4 ORB_SLAM2 实验

一.mono_kitti.cc代码框架

1.读取图片目录

```
LoadImages(string(argv[3]), vstrImageFilenames, vTimestamps);
```

2.创建ORB_SLAM2系统对象

```
ORB_SLAM2::System SLAM(argv[1],argv[2],ORB_SLAM2::System::MONOCULAR,true);
```

函数的内部流程如下：

（1）创建了ORB词袋的对象

```
mpVocabulary = new ORBVocabulary();
```

（2）创建了关键帧的数据库

```
mpKeyFrameDatabase = new KeyFrameDatabase(*mpVocabulary);
```

（3）创建地图对象

```
mpMap = new Map();
```

（4）创建两个显示窗口

```
mpFrameDrawer = new FrameDrawer(mpMap);
mpMapDrawer = new MapDrawer(mpMap, strSettingsFile);
```

（5）初始化Tracking对象

```
mpTracker = new Tracking(this, mpVocabulary, mpFrameDrawer, mpMapDrawer,
                         mpMap, mpKeyFrameDatabase, strSettingsFile, mSensor);
```

（6）初始化Local Mapping对象，并开启线程运行

```
mpLocalMapper = new LocalMapping(mpMap, mSensor==MONOCULAR);
mptLocalMapping = new thread(&ORB_SLAM2::LocalMapping::Run,mpLocalMapper);
```

（7）初始化Loop Closing对象，并开启线程运行

```
mpLoopCloser = new LoopClosing(mpMap, mpKeyFrameDatabase, mpVocabulary,
mSensor!=MONOCULAR);
mptLoopClosing = new thread(&ORB_SLAM2::LoopClosing::Run, mpLoopCloser);
```

3.循环读取数据

```
for(int ni=0; ni<nImages; ni++)
{
    ...
}
```

for循环内部的流程如下：

（1）读取图片

```
im = cv::imread(vstrImageFilenames[ni],CV_LOAD_IMAGE_UNCHANGED);
```

（2）读取时间戳

```
double tframe = vTimestamps[ni];
```

（3）将图片传给SLAM系统

```
SLAM.TrackMonocular(im,tframe);
```

具体代码分析如下：

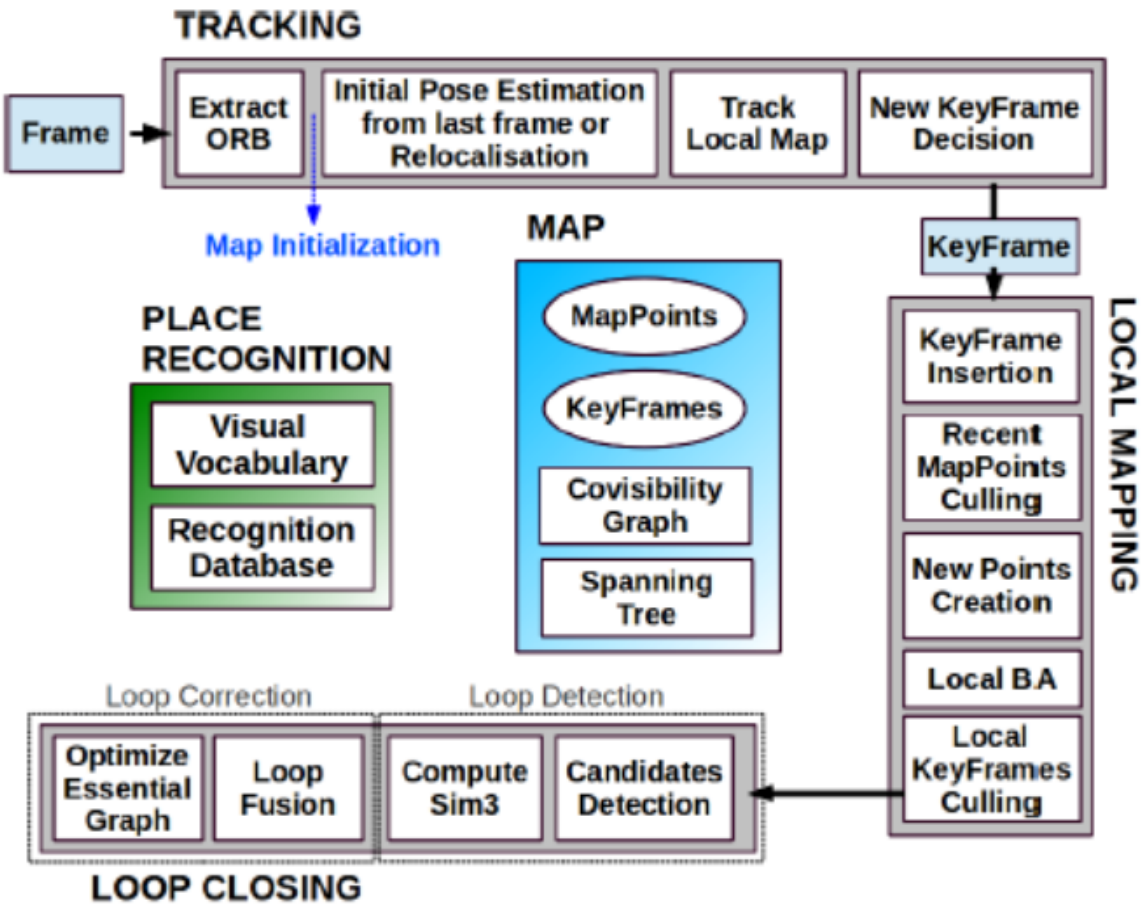
```
cv::Mat System::TrackMonocular(const cv::Mat &im, const double &timestamp)
{
    // 传感器不是单目摄像头、退出
    if(mSensor!=MONOCULAR)
    {
        cerr << "ERROR: you called TrackMonocular but input sensor was not set to
Monocular." << endl;
        exit(-1);
    }

    // Check mode change
    // 这一部分主要是对局部地图线程进行操作.
    // mbActivateLocalizationMode是是否停止局部地图线程
    // mbDeactivateLocalizationMode是是否清空局部地图.
    {
        // 独占锁，主要是为了mbActivateLocalizationMode和mbDeactivateLocalizationMode
        // 不会发生混乱，没有死锁或者在临界区
        unique_lock<mutex> lock(mMutexMode);
        // mbActivateLocalizationMode为true会关闭局部地图线程
        if(mbActivateLocalizationMode)
        {
```

```
            mpLocalMapper->RequestStop();
            // 设置local map的mbStopRequested，mbAbortBA为true.
            // 当这两个为true的时候，那么进行就会去关闭局部地图的线程

            // Wait until Local Mapping has effectively stopped
            // mbStopped为true，说明局部地图线程已经关闭了
            while(!mpLocalMapper->isStopped())
            {
                usleep(1000);
            }
            // 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
            // 设置mbOnlyTracking为真
            mpTracker->InformOnlyTracking(true);
            // 执行完当前的部分之和把mbActivateLocalizationMode再置回false.
            //当然这里设置mbActivateLocalizationMode为true的部分应该是没有新的关键帧和点云的时候
            // 关闭线程可以使得别的线程得到更多的资源
            mbActivateLocalizationMode = false;
        }
        // 如果mbDeactivateLocalizationMode是true
        // 设置mbActivateLocalizationMode为false
        // 局部地图线程就被释放，关键帧从局部地图中删除.
        // mbStopped和mbStopRequested被置为false.
        if(mbDeactivateLocalizationMode)
        {
            mpTracker->InformOnlyTracking(false);
            mpLocalMapper->Release();
            mbDeactivateLocalizationMode = false;
        }
    }

    // Check reset
    // 检查是否需要进行复位重置.
    {
        // 给mbReset加锁，防止被别的线程修改
        unique_lock<mutex> lock(mMutexReset);
        if(mbReset)
        {
            // mpViwer暂停，视图停止更新
            // 局部地图：mpLocalMapper和闭环检测：mpLoopClosing被停止.
            // Bow:mpKeyFrameDB和mpMap被清空
            // 就是把所有资源释放
            mpTracker->Reset();
            mbReset = false;
        }
    }
    // 可以看出上面这两部分都是对于各个线程状态的反馈.
    // 其实可以看做是对上一个状态的反馈.
    // 接下来的部分才是最重要的部分,获取数据，对各个线程的数据进行更新，对地图重新进行计算.
    return mpTracker->GrabImageMonocular(im,timestamp);
}
```



工作流程如下：

<1> Tracking 每一帧图像进行跟踪计算

（1）从图像中提取**ORB**特征，

（2）根据上一帧进行姿态估计，或者进行通过全局重定位初始化位姿，

（3）然后跟踪已经重建的局部地图，优化位姿，

（4）再根据一些规则确定新的关键帧。

<2> Mapping 地图构建

（1）加入关键帧，更新图，

（2）验证最近加入的地图点，去除outlier

（3）生成新的地图点

（4）局部bundle 调整（去除outlier）

（5）验证关键帧（去除重复帧）

<3> LoopClosing 闭环检测

（1）选取相似帧

（2）检测闭环，RANSAC计算内点数

> （3）融合三维点，更新图
>
> （4）图优化，传导变换矩阵，更新地图点

4.关闭SLAM系统

```
SLAM.Shutdown();
```

5.保存相机轨线

```
SLAM.SaveTrajectoryKITTI("CameraTrajectory.txt");
```

二.SLAM工作流程

<1> 读取传感器信息。

> 如相机采集现实世界中的图像信息，并对图像数据进行预处理。

<2> 视觉里程计(VO)，也称为前端。

> 给定相邻的图像，估算在获取到连续的相邻图像时，相机的位姿变化（相机的运动），即**R**和**t**，以此可以构建局部地图。
>
> 工作流程如下：
>
> （1）对当前帧提取关键点和描述子
>
> （2）如果系统没有初始化（若已经初始化，跳过该步），则以当前帧为参考帧，根据深度图来得到关键点的3D位置，返回（1）
>
> （3）根据参考帧和当前帧之间的运动情况
>
> （4）判断（3）中的估计是否正确，正确的话，把当前帧当作新的参考帧，返回（1），不正确的话，统计连续丢失的帧数，当丢失的帧数超过一个阈值，那么置VO状态为丢失，算法结束。如果没有超过阈值，返回（1）

<3> 后端优化

> 如果相机是在时刻运动的，后端可以接受不同时刻视觉里程计测量的相机位姿，并进行**回环检测**，获取到差异信息后，对它们进行优化，从而得到全局一致的轨迹和地图。
>
> 通俗的讲，前端视觉里程计得到的是一个**短时间**的轨迹和地图，但在**长时间**下，由于误差会进行**累计**，这时候得到的轨迹和地图是很不准确的，因此**长时间**下，我们需要**后端优化，来得到系统的状态量：位姿x和路标y**

<4> 回环检测

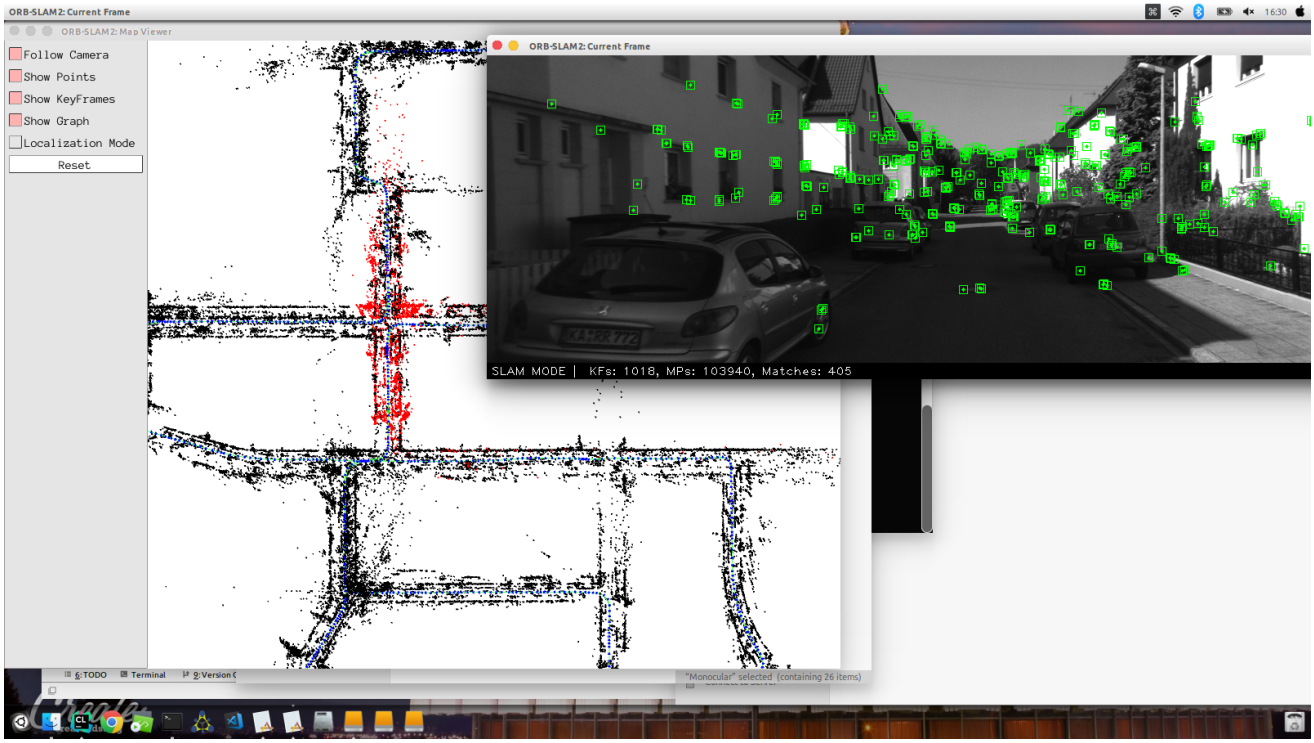
> 用以判断机器人是否曾经到达过先前的位置。如果到达过，则路径会产生回环，检测到回环后，将信息提供给后端进行处理。

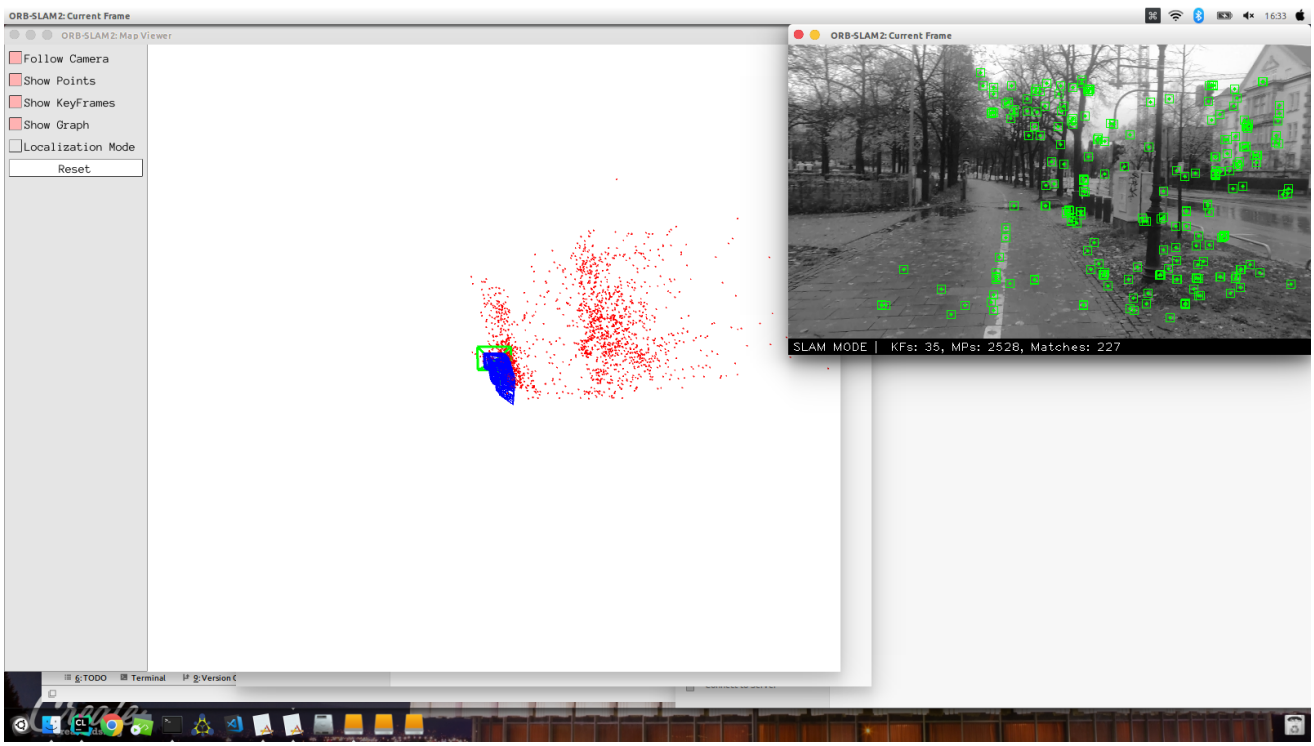
> 判断机器人是否曾经到达过先前的位置，若是回到了之前到达过的位置，那么可以为我们消除累积误差提供可能（因为相机在同一个地方会采集到相似的数据，这些数据间的约束条件可以有助于消除累计误差），另一方面，回环检测提供的当前数据与所有历史数据的关联，因此可以我们在跟踪某一特征点时丢失了后，可以利用回环检测进行重定位。基于外观的方法里主要是基于词袋的模型。

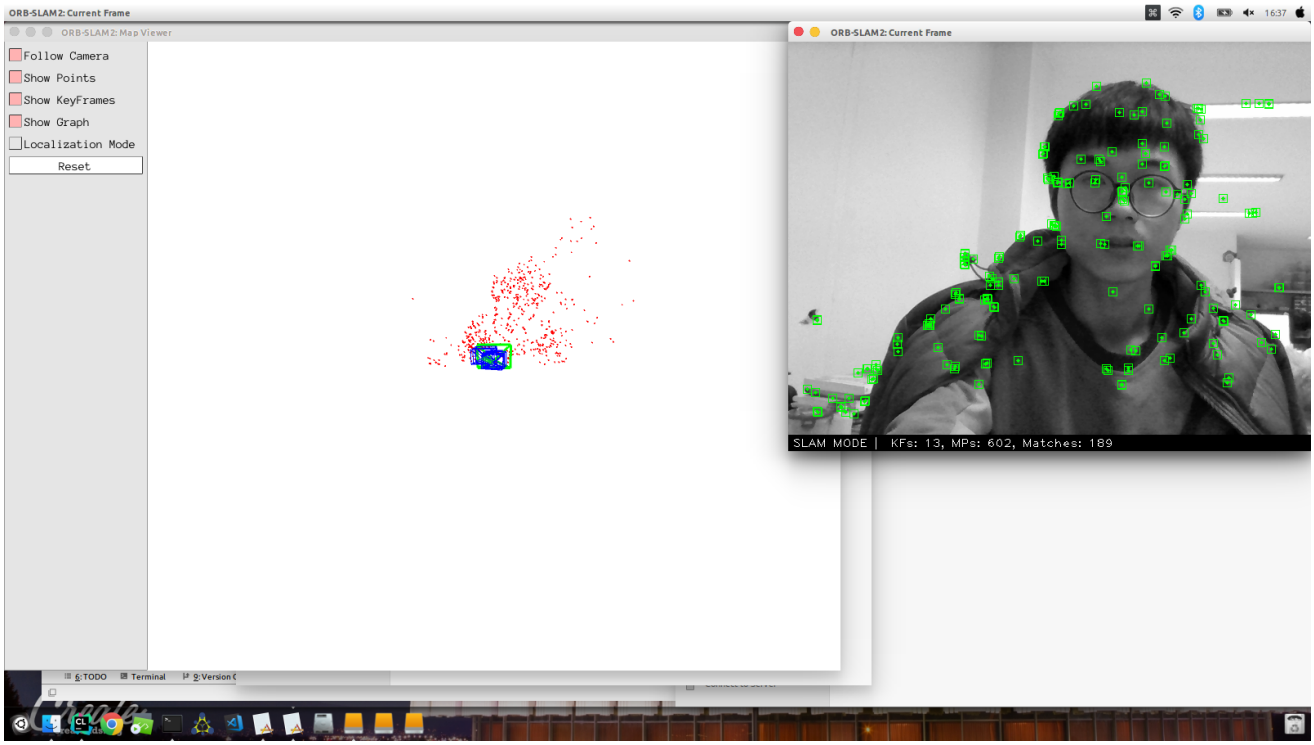
<5> 地图构建

> 结合前边的结果，估计机器人运动的轨迹，从而建立与任务要求对应的地图。

三.实验结果






ORB-SLAM2: Current Frame
ORB-SLAM2: Map Viewer
Follow Camera
Show Points
Show KeyFrames
Show Graph
Localization Mode
Reset
ORB-SLAM2: Current Frame
SLAM MODE | KFs: 13, MPs: 602, Matches: 189