HanLP Handbook

version 1.0

hankcs

April 20, 2018

Contents

vision workspace

Contents:

1.初始化Initializer.h

构建Initializer的类 该类主要完成的功能是： 初始化SLAM的R,t,及点云，计算Fundamental,Homography,以及分解Fundamental 和Homography,存储当前帧与参考帧的关键点以及特征匹配。三角化方法等等。

- 1 .用reference frame来初始化，这个reference frame就是SLAM正式开始的第一帧
- 2.用current frame,也就是用SLAM逻辑上的第二帧来初始化整个SLAM，得到最开始两帧之间的R t,以及点云
- 3. FindHomography

  假设场景为平面情况下通过前两帧求取Homography矩阵(current frame 2 到 reference frame 1),并得到该模型的评分
- 4. FindFundamental

  假设场景为非平面情况下通过前两帧求取Fundamental矩阵(current frame 2 到 reference frame 1),并得到该模型的评分
- 5. ComputeH21

  被FindHomography函数调用具体来算Homography矩阵
- 6. ComputeF21

  被FindFundamental函数调用具体来算Fundamental矩阵
- 7. CheckHomography

  被FindHomography函数调用，具体来算假设使用Homography模型的得分
- 8.CheckFundamental

  被FindFundamental函数调用，具体来算假设使用Fundamental模型的得分
- 9.ReconstructF

  分解F矩阵，并从分解后的多个解中找出合适的R，t
- 10.ReconstructH

  分解H矩阵，并从分解后的多个解中找出合适的R，t
- 11.Triangulate

  通过三角化方法，利用反投影矩阵将特征点恢复为3D点
- 12.Normalize

  归一化三维空间点和帧间位移t
- 13.CheckRT

  ReconstructF调用该函数进行cheirality check，从而进一步找出F分解后最合适的解
- 14.DecomposeE

  F矩阵通过结合内参可以得到Essential矩阵，该函数用于分解E矩阵，将得到4组解
- 15. 除了以上函数外，还有一些变量，主要用来存储参考帧和当前帧的特征点,以及记录匹配的点，相机内参，以及计算Fundamental 和Homography 矩阵时RANSAC迭代次数

  vector<cv::KeyPoint> mvKeys1; ///< 存储Reference Frame中的特征点

  vector<cv::KeyPoint> mvKeys2; ///< 存储Current Frame中的特征点

  vector<Match> mvMatches12; ///< Match的数据结构是pair,mvMatches12只记录Reference到Current匹配上的特征点对 vector<bool> mvbMatched1; ///< 记录Reference Frame的每个特征点在Current Frame是否有匹配的特征点

  cv::Mat mK; ///< 相机内参

  // Standard Deviation and Variance float mSigma, mSigma2; ///< 测量误差

  // Ransac max iterations int mMaxIterations; ///< 算Fundamental和Homography矩阵时RANSAC迭代次数

  // Ransac sets vector<vector<size_t> > mvSets; ///< 二维容器，外层容器的大小为迭代次数，内层容器大小为每次迭代算H或F矩阵需要的点

```
class Initializer
{
    typedef pair<int,int> Match;

public:

    // Fix the reference frame
    // 用reference frame来初始化，这个reference frame就是SLAM正式开始的第一帧
    Initializer(const Frame &ReferenceFrame, float sigma = 1.0, int iterations = 200);

    // Computes in parallel a fundamental matrix and a homography
    // Selects a model and tries to recover the motion and the structure from motion
    // 用current frame,也就是用SLAM逻辑上的第二帧来初始化整个SLAM，得到最开始两帧之间的R t,以及点云
    bool Initialize(const Frame &CurrentFrame, const vector<int> &vMatches12,
                    cv::Mat &R21, cv::Mat &t21, vector<cv::Point3f> &vP3D, vector<bool> &vbTriangulated);
```

主要函数成员

```
private:

    // 假设场景为平面情况下通过前两帧求取Homography矩阵(current frame 2 到 reference frame 1),并得到该模型的评分
    void FindHomography(vector<bool> &vbMatchesInliers, float &score, cv::Mat &H21);
    // 假设场景为非平面情况下通过前两帧求取Fundamental矩阵(current frame 2 到 reference frame 1),并得到该模型的评分
    void FindFundamental(vector<bool> &vbInliers, float &score, cv::Mat &F21);

    // 被FindHomography函数调用具体来算Homography矩阵
    cv::Mat ComputeH21(const vector<cv::Point2f> &vP1, const vector<cv::Point2f> &vP2);
    // 被FindFundamental函数调用具体来算Fundamental矩阵
    cv::Mat ComputeF21(const vector<cv::Point2f> &vP1, const vector<cv::Point2f> &vP2);

    // 被FindHomography函数调用，具体来算假设使用Homography模型的得分
    float CheckHomography(const cv::Mat &H21, const cv::Mat &H12, vector<bool> &vbMatchesInliers, float sigma);
    // 被FindFundamental函数调用，具体来算假设使用Fundamental模型的得分
    float CheckFundamental(const cv::Mat &F21, vector<bool> &vbMatchesInliers, float sigma);

    // 分解F矩阵，并从分解后的多个解中找出合适的R，t
    bool ReconstructF(vector<bool> &vbMatchesInliers, cv::Mat &F21, cv::Mat &K,
                      cv::Mat &R21, cv::Mat &t21, vector<cv::Point3f> &vP3D, vector<bool> &vbTriangulated, float minParallax, int minTriangulated);

    // 分解H矩阵，并从分解后的多个解中找出合适的R，t
    bool ReconstructH(vector<bool> &vbMatchesInliers, cv::Mat &H21, cv::Mat &K,
                      cv::Mat &R21, cv::Mat &t21, vector<cv::Point3f> &vP3D, vector<bool> &vbTriangulated, float minParallax, int minTriangulated);

    // 通过三角化方法，利用反投影矩阵将特征点恢复为3D点
    void Triangulate(const cv::KeyPoint &kp1, const cv::KeyPoint &kp2, const cv::Mat &P1, const cv::Mat &P2, cv::Mat &x3D);

    // 归一化三维空间点和帧间位移t
    void Normalize(const vector<cv::KeyPoint> &vKeys, vector<cv::Point2f> &vNormalizedPoints, cv::Mat &T);

    // ReconstructF调用该函数进行cheirality check，从而进一步找出F分解后最合适的解
    int CheckRT(const cv::Mat &R, const cv::Mat &t, const vector<cv::KeyPoint> &vKeys1, const vector<cv::KeyPoint> &vKeys2,
                       const vector<Match> &vMatches12, vector<bool> &vbInliers,
                       const cv::Mat &K, vector<cv::Point3f> &vP3D, float th2, vector<bool> &vbGood, float &parallax);

    // F矩阵通过结合内参可以得到Essential矩阵，该函数用于分解E矩阵，将得到4组解
    void DecomposeE(const cv::Mat &E, cv::Mat &R1, cv::Mat &R2, cv::Mat &t);


    // Keypoints from Reference Frame (Frame 1)
    vector<cv::KeyPoint> mvKeys1; ///< 存储Reference Frame中的特征点

    // Keypoints from Current Frame (Frame 2)
    vector<cv::KeyPoint> mvKeys2; ///< 存储Current Frame中的特征点

    // Current Matches from Reference to Current
    // Reference Frame: 1, Current Frame: 2
    vector<Match> mvMatches12; ///< Match的数据结构是pair,mvMatches12只记录Reference到Current匹配上的特征点对
    vector<bool> mvbMatched1; ///< 记录Reference Frame的每个特征点在Current Frame是否有匹配的特征点

    // Calibration
    cv::Mat mK; ///< 相机内参

    // Standard Deviation and Variance
    float mSigma, mSigma2; ///< 测量误差

    // Ransac max iterations
    int mMaxIterations; ///< 算Fundamental和Homography矩阵时RANSAC迭代次数

    // Ransac sets
    vector<vector<size_t> > mvSets; ///< 二维容器，外层容器的大小为迭代次数，内层容器大小为每次迭代算H或F矩阵需要的点

};
```

2.Frame帧类

Frame类中包含了MapPoint类和KeyFrame类

```
#include "MapPoint.h"
#include "Thirdparty/DBoW2/DBoW2/BowVector.h"
#include "Thirdparty/DBoW2/DBoW2/FeatureVector.h"
#include "ORBVocabulary.h"
#include "KeyFrame.h"
#include "ORBextractor.h"

class MapPoint;
class KeyFrame;
```

分别为双目摄像头，深度摄像头，单目摄像头三类构建帧类的复制构造函数

```
// Constructor for stereo cameras.
Frame(const cv::Mat &imLeft, const cv::Mat &imRight, const double &timeStamp, ORBextractor* extractorLeft, ORBextractor* extractorRight, ORBVocabulary* voc

// Constructor for RGB-D cameras.
Frame(const cv::Mat &imGray, const cv::Mat &imDepth, const double &timeStamp, ORBextractor* extractor,ORBVocabulary* voc, cv::Mat &K, cv::Mat &distCoef, co

// Constructor for Monocular cameras.
Frame(const cv::Mat &imGray, const double &timeStamp, ORBextractor* extractor,ORBVocabulary* voc, cv::Mat &K, cv::Mat &distCoef, const float &bf, const flo
```

抽取ORB特征

```
// Extract ORB on the image. 0 for left image and 1 for right image.
// 提取的关键点存放在mvKeys和mDescriptors中
// ORB是直接调orbExtractor提取的
void ExtractORB(int flag, const cv::Mat &im);
```

计算词袋BoW

```
// Compute Bag of Words representation.
// 存放在mBowVec中
void ComputeBoW();
```

设置相机位姿

```
// Set the camera pose.
// 用Tcw更新mTcw
void SetPose(cv::Mat Tcw);
```

从相机姿态中计算旋转，平移和相机中心矩阵

```
// Computes rotation, translation and camera center matrices from the camera pose.
void UpdatePoseMatrices();
```

得到相机中心点

```
// Returns the camera center.
inline cv::Mat GetCameraCenter()
{
    return mOw.clone();
}
// Returns inverse of rotation
```

得到旋转矩阵的逆矩阵

```
inline cv::Mat GetRotationInverse()
{
    return mRwc.clone();
}
```

判断路标点是否在视野中

```
// Check if a MapPoint is in the frustum of the camera
// and fill variables of the MapPoint to be used by the tracking
// 判断路标点是否在视野中
bool isInFrustum(MapPoint* pMP, float viewingCosLimit);
```

判断关键点是否在grid中

```
// Compute the cell of a keypoint (return false if outside the grid)
bool PosInGrid(const cv::KeyPoint &kp, int &posX, int &posY);

vector<size_t> GetFeaturesInArea(const float &x, const float  &y, const float  &r, const int minLevel=-1, const int maxLevel=-1) const;
```

判断左右图关键点是否match,如果match,计算深度信息并将左右关键点坐标存储

```
// Search a match for each keypoint in the left image to a keypoint in the right image.
// If there is a match, depth is computed and the right coordinate associated to the left keypoint is stored.
void ComputeStereoMatches();

// Associate a "right" coordinate to a keypoint if there is valid depth in the depthmap.
void ComputeStereoFromRGBD(const cv::Mat &imDepth);
```

将一个关键点从映射到3D世界坐标

```
// Backprojects a keypoint (if stereo/depth info available) into 3D world coordinates.
cv::Mat UnprojectStereo(const int &i);
```

3.keyframe 关键帧

3.keyframe 关键帧

这里有线程锁的概念，还不是很清楚这块

关键帧，和普通的Frame不一样，但是可以由Frame来构造 许多数据会被三个线程同时访问，所以用锁的地方很普遍

关键帧包含了地图，路标点，帧，关键帧数据库等类

```
class Map;
class MapPoint;
class Frame;
class KeyFrameDatabase;
```

设置Pose,得到Pose,Pose的逆矩阵，Get相机中心，Get双目相机中心，Get旋转矩阵，Get平移,计算BoW

```
// Pose functions
// 这里的set,get需要用到锁
void SetPose(const cv::Mat &Tcw);
cv::Mat GetPose();
cv::Mat GetPoseInverse();
cv::Mat GetCameraCenter();
cv::Mat GetStereoCenter();
cv::Mat GetRotation();
cv::Mat GetTranslation();

// Bag of Words Representation
void ComputeBoW();
```

图优化相关的一些函数

Covisibility graph是不同关键帧之间共享的可见点。

添加连接connection，删除连接，更新连接，更新最好的共享可见点.

添加子节点child，删除子节点，得到子节点

添加路标点MapPoint，删除路标点，得到路标点，

LoopEdge，

关键点 keypoint

```
// Covisibility graph functions
void AddConnection(KeyFrame* pKF, const int &weight);
void EraseConnection(KeyFrame* pKF);
void UpdateConnections();
void UpdateBestCovisibles();
std::set<KeyFrame *> GetConnectedKeyFrames();
std::vector<KeyFrame* > GetVectorCovisibleKeyFrames();
std::vector<KeyFrame*> GetBestCovisibilityKeyFrames(const int &N);
std::vector<KeyFrame*> GetCovisiblesByWeight(const int &w);
int GetWeight(KeyFrame* pKF);

// Spanning tree functions
void AddChild(KeyFrame* pKF);
void EraseChild(KeyFrame* pKF);
void ChangeParent(KeyFrame* pKF);
std::set<KeyFrame*> GetChilds();
KeyFrame* GetParent();
bool hasChild(KeyFrame* pKF);

// Loop Edges
void AddLoopEdge(KeyFrame* pKF);
std::set<KeyFrame*> GetLoopEdges();

// MapPoint observation functions
void AddMapPoint(MapPoint* pMP, const size_t &idx);
void EraseMapPointMatch(const size_t &idx);
void EraseMapPointMatch(MapPoint* pMP);
void ReplaceMapPointMatch(const size_t &idx, MapPoint* pMP);
std::set<MapPoint*> GetMapPoints();
std::vector<MapPoint*> GetMapPointMatches();
int TrackedMapPoints(const int &minObs);
MapPoint* GetMapPoint(const size_t &idx);

// KeyPoint functions
std::vector<size_t> GetFeaturesInArea(const float &x, const float  &y, const float  &r) const;
cv::Mat UnprojectStereo(int i);

// Image
bool IsInImage(const float &x, const float &y) const;

// Enable/Disable bad flag changes
void SetNotErase();
void SetErase();

// Set/check bad flag
void SetBadFlag();
```

```
bool isBad();

// Compute Scene Depth (q=2 median). Used in monocular.
float ComputeSceneMedianDepth(const int q);

static bool weightComp( int a, int b)
{
    return a>b;
}

static bool lId(KeyFrame* pKF1, KeyFrame* pKF2)
{
    return pKF1->mnId<pKF2->mnId;
}
```

下面的变量只可以单线程访问

包含了keyframe的ID号，时间戳，Grid,local mapping的一些变量,回环的一些变量

相机补偿的参数，等等

```
// The following variables are accesed from only 1 thread or never change (no mutex needed).
public:

    // nNextID名字改为nLastID更合适，表示上一个KeyFrame的ID号
    static long unsigned int nNextId;
    // 在nNextID的基础上加1就得到了mnID，为当前KeyFrame的ID号
    long unsigned int mnId;
    // 每个KeyFrame基本属性是它是一个Frame，KeyFrame初始化的时候需要Frame，
    // mnFrameId记录了该KeyFrame是由哪个Frame初始化的
    const long unsigned int mnFrameId;

    const double mTimeStamp;

    // Grid (to speed up feature matching)
    // 和Frame类中的定义相同
    const int mnGridCols;
    const int mnGridRows;
    const float mfGridElementWidthInv;
    const float mfGridElementHeightInv;

    // Variables used by the tracking
    long unsigned int mnTrackReferenceForFrame;
    long unsigned int mnFuseTargetForKF;

    // Variables used by the local mapping
    long unsigned int mnBALocalForKF;
    long unsigned int mnBAFixedForKF;

    // Variables used by the keyframe database
    long unsigned int mnLoopQuery;
    int mnLoopWords;
    float mLoopScore;
    long unsigned int mnRelocQuery;
    int mnRelocWords;
    float mRelocScore;

    // Variables used by loop closing
    cv::Mat mTcwGBA;
    cv::Mat mTcwBefGBA;
    long unsigned int mnBAGlobalForKF;

    // Calibration parameters
    const float fx, fy, cx, cy, invfx, invfy, mbf, mb, mThDepth;

    // Number of KeyPoints
    const int N;

    // KeyPoints, stereo coordinate and descriptors (all associated by an index)
    // 和Frame类中的定义相同
    const std::vector<cv::KeyPoint> mvKeys;
    const std::vector<cv::KeyPoint> mvKeysUn;
    const std::vector<float> mvuRight; // negative value for monocular points
    const std::vector<float> mvDepth; // negative value for monocular points
    const cv::Mat mDescriptors;

    //BoW
    DBoW2::BowVector mBowVec; ///< Vector of words to represent images
    DBoW2::FeatureVector mFeatVec; ///< Vector of nodes with indexes of local features

    // Pose relative to parent (this is computed when bad flag is activated)
    cv::Mat mTcp;
```

```
    // Scale
    const int mnScaleLevels;
    const float mfScaleFactor;
    const float mfLogScaleFactor;
    const std::vector<float> mvScaleFactors;// 尺度因子，scale^n，scale=1.2，n为层数
    const std::vector<float> mvLevelSigma2;// 尺度因子的平方
    const std::vector<float> mvInvLevelSigma2;

    // Image bounds and calibration
    const int mnMinX;
    const int mnMinY;
    const int mnMaxX;
    const int mnMaxY;
    const cv::Mat mK;
```

4.KeyframeDatabase

该类会用到KeyFrame和Frame两个类

```
class KeyFrame;
class Frame;
```

类定义

包含了添加，删除，清除，回环检测，重定位等函数

包含了ORB词典，索引文件等

```
class KeyFrameDatabase
{
public:

    KeyFrameDatabase(const ORBVocabulary &voc);

   void add(KeyFrame* pKF);

   void erase(KeyFrame* pKF);

   void clear();

   // Loop Detection
   std::vector<KeyFrame *> DetectLoopCandidates(KeyFrame* pKF, float minScore);

   // Relocalization
   std::vector<KeyFrame*> DetectRelocalizationCandidates(Frame* F);

protected:

  // Associated vocabulary
  const ORBVocabulary* mpVoc; ///< 预先训练好的词典

  // Inverted file
  std::vector<list<KeyFrame*> > mvInvertedFile; ///< 倒排索引，mvInvertedFile[i]表示包含了第i个word id的所有关键帧

  // Mutex
  std::mutex mMutex;
};
```

5.MapPoint 路标点,地图点

设置世界坐标,得到世界坐标

```
void SetWorldPos(const cv::Mat &Pos);
cv::Mat GetWorldPos();
```

归一化

```
cv::Mat GetNormal();
```

得到参考的关键帧

```
KeyFrame* GetReferenceKeyFrame();
```

观测点

```
std::map<KeyFrame*,size_t> GetObservations();
int Observations();
void AddObservation(KeyFrame* pKF,size_t idx);
void EraseObservation(KeyFrame* pKF);
```

关键帧的index

```
int GetIndexInKeyFrame(KeyFrame* pKF);
bool IsInKeyFrame(KeyFrame* pKF);

void SetBadFlag();
bool isBad();

void Replace(MapPoint* pMP);
MapPoint* GetReplaced();

void IncreaseVisible(int n=1);
void IncreaseFound(int n=1);
float GetFoundRatio();
inline int GetFound(){
    return mnFound;
}
```

计算描述子

```
void ComputeDistinctiveDescriptors();

cv::Mat GetDescriptor();

void UpdateNormalAndDepth();
```

计算最大，最小距离方差

```
float GetMinDistanceInvariance();
float GetMaxDistanceInvariance();
int PredictScale(const float &currentDist, KeyFrame*pKF);
int PredictScale(const float &currentDist, Frame* pF);
```

Tracking

TrackLocalMap - SearchByProjection中决定是否对该点进行投影的变量

mbTrackInView==false的点有几种:

- 已经和当前帧经过匹配（TrackReferenceKeyFrame，TrackWithMotionModel）但在优化过程中认为是外点
- 已经和当前帧经过匹配且为内点，这类点也不需要再进行投影
- 不在当前相机视野中的点（即未通过isInFrustum判断）

3D Descriptor

每个3D点也有一个descriptor

如果MapPoint与很多帧图像特征点对应（由keyframe来构造时），那么距离其它描述子的平均距离最小的描述子是最佳描述子

MapPoint只与一帧的图像特征点对应（由frame来构造时），那么这个特征点的描述子就是该3D点的描述子

```
public:
    long unsigned int mnId; ///< Global ID for MapPoint
    static long unsigned int nNextId;
    const long int mnFirstKFid; ///< 创建该MapPoint的关键帧ID
    const long int mnFirstFrame; ///< 创建该MapPoint的帧ID（即每一关键帧有一个帧ID）
    int nObs;

    // Variables used by the tracking
    float mTrackProjX;
    float mTrackProjY;
    float mTrackProjXR;
    int mnTrackScaleLevel;
    float mTrackViewCos;

    // TrackLocalMap - SearchByProjection中决定是否对该点进行投影的变量
    // mbTrackInView==false的点有几种:
    // a 已经和当前帧经过匹配（TrackReferenceKeyFrame，TrackWithMotionModel）但在优化过程中认为是外点
    // b 已经和当前帧经过匹配且为内点，这类点也不需要再进行投影
    // c 不在当前相机视野中的点（即未通过isInFrustum判断）
    bool mbTrackInView;
    // TrackLocalMap - UpdateLocalPoints中防止将MapPoints重复添加至mvpLocalMapPoints的标记
    long unsigned int mnTrackReferenceForFrame;
    // TrackLocalMap - SearchLocalPoints中决定是否进行isInFrustum判断的变量
    // mnLastFrameSeen==mCurrentFrame.mnId的点有几种:
    // a 已经和当前帧经过匹配（TrackReferenceKeyFrame，TrackWithMotionModel）但在优化过程中认为是外点
    // b 已经和当前帧经过匹配且为内点，这类点也不需要再进行投影
    long unsigned int mnLastFrameSeen;

    // Variables used by local mapping
    long unsigned int mnBALocalForKF;
    long unsigned int mnFuseCandidateForKF;

    // Variables used by loop closing
    long unsigned int mnLoopPointForKF;
```

```
    long unsigned int mnCorrectedByKF;
    long unsigned int mnCorrectedReference;
    cv::Mat mPosGBA;
    long unsigned int mnBAGlobalForKF;


    static std::mutex mGlobalMutex;

protected:

    // Position in absolute coordinates
    cv::Mat mWorldPos; ///< MapPoint在世界坐标系下的坐标

    // Keyframes observing the point and associated index in keyframe
    std::map<KeyFrame*,size_t> mObservations; ///< 观测到该MapPoint的KF和该MapPoint在KF中的索引

    // Mean viewing direction
    // 该MapPoint平均观测方向
    cv::Mat mNormalVector;

    // Best descriptor to fast matching
    // 每个3D点也有一个descriptor
    // 如果MapPoint与很多帧图像特征点对应（由keyframe来构造时），那么距离其它描述子的平均距离最小的描述子是最佳描述子
    // MapPoint只与一帧的图像特征点对应（由frame来构造时），那么这个特征点的描述子就是该3D点的描述子
    cv::Mat mDescriptor; ///< 通过 ComputeDistinctiveDescriptors() 得到的最优描述子

    // Reference KeyFrame
    KeyFrame* mpRefKF;

    // Tracking counters
    int mnVisible;
    int mnFound;

    // Bad flag (we do not currently erase MapPoint from memory)
    bool mbBad;
    MapPoint* mpReplaced;

    // Scale invariance distances
    float mfMinDistance;
    float mfMaxDistance;

    Map* mpMap;

    std::mutex mMutexPos;
    std::mutex mMutexFeatures;
};
```

6.Map地图

地图负责管理关键帧，路标点的功能

在地图中添加关键帧，添加路标点，删除路标点，删除关键帧，设置参考路标点，获得所有关键帧，过得参考的地图点

```
class Map
{
public:
    Map();

    void AddKeyFrame(KeyFrame* pKF);
    void AddMapPoint(MapPoint* pMP);
    void EraseMapPoint(MapPoint* pMP);
    void EraseKeyFrame(KeyFrame* pKF);
    void SetReferenceMapPoints(const std::vector<MapPoint*> &vpMPs);

    std::vector<KeyFrame*> GetAllKeyFrames();
    std::vector<MapPoint*> GetAllMapPoints();
    std::vector<MapPoint*> GetReferenceMapPoints();

    long unsigned int MapPointsInMap();
    long unsigned  KeyFramesInMap();

    long unsigned int GetMaxKFid();

    void clear();

    vector<KeyFrame*> mvpKeyFrameOrigins;

    std::mutex mMutexMapUpdate;

    // This avoid that two points are created simultaneously in separate threads (id conflict)
    std::mutex mMutexPointCreation;
```

```
protected:
    std::set<MapPoint*> mspMapPoints; ///< MapPoints
    std::set<KeyFrame*> mspKeyFrames; ///< Keyframs

    std::vector<MapPoint*> mvpReferenceMapPoints;

    long unsigned int mnMaxKFid;

    std::mutex mMutexMap;
};
```

7.ORBExtractor

请详细阅读ORB特征提取的论文，搞懂它的内部算法。

ExtractorNode

```
class ExtractorNode
{
public:
    ExtractorNode():bNoMore(false){}

    void DivideNode(ExtractorNode &n1, ExtractorNode &n2, ExtractorNode &n3, ExtractorNode &n4);

    std::vector<cv::KeyPoint> vKeys;
    cv::Point2i UL, UR, BL, BR;
    std::list<ExtractorNode>::iterator lit;
    bool bNoMore;
};
```

ORBextractor

```
class ORBextractor
{
public:

    enum {HARRIS_SCORE=0, FAST_SCORE=1 };

    ORBextractor(int nfeatures, float scaleFactor, int nlevels,
                 int iniThFAST, int minThFAST);

    ~ORBextractor(){}

    // Compute the ORB features and descriptors on an image.
    // ORB are dispersed on the image using an octree.
    // Mask is ignored in the current implementation.
    void operator()( cv::InputArray image, cv::InputArray mask,
      std::vector<cv::KeyPoint>& keypoints,
      cv::OutputArray descriptors);

    int inline GetLevels(){
        return nlevels;}

    float inline GetScaleFactor(){
        return scaleFactor;}

    std::vector<float> inline GetScaleFactors(){
        return mvScaleFactor;
    }

    std::vector<float> inline GetInverseScaleFactors(){
        return mvInvScaleFactor;
    }

    std::vector<float> inline GetScaleSigmaSquares(){
        return mvLevelSigma2;
    }

    std::vector<float> inline GetInverseScaleSigmaSquares(){
        return mvInvLevelSigma2;
    }

    std::vector<cv::Mat> mvImagePyramid;

protected:

    void ComputePyramid(cv::Mat image);
    void ComputeKeyPointsOctTree(std::vector<std::vector<cv::KeyPoint> >& allKeypoints);
    std::vector<cv::KeyPoint> DistributeOctTree(const std::vector<cv::KeyPoint>& vToDistributeKeys, const int &minX,
                                           const int &maxX, const int &minY, const int &maxY, const int &nFeatures, const int &level);

    void ComputeKeyPointsOld(std::vector<std::vector<cv::KeyPoint> >& allKeypoints);
```

```
    std::vector<cv::Point> pattern;

    int nfeatures;
    double scaleFactor;
    int nlevels;
    int iniThFAST;
    int minThFAST;

    std::vector<int> mnFeaturesPerLevel;

    std::vector<int> umax;

    std::vector<float> mvScaleFactor;
    std::vector<float> mvInvScaleFactor;
    std::vector<float> mvLevelSigma2;
    std::vector<float> mvInvLevelSigma2;
};
```