ORBSLAM2学习（五）：DBoW2源码分析（OrbDatabase部分）

2018年06月04日 15:12:41 Mega_Li 阅读数 574 更多

0 前言

接着上一篇博客链接继续做学习记录，这一次分析demo中OrbDatabase的应用和原理。

1 工程运行结果

demo中main函数内容如下所示。

```
int main()
{
  vector<vector<cv::Mat > > features;
  loadFeatures(features);// 提取特征
  testVocCreation(features);// 建立OrbVocabulary对象
  wait();
  testDatabase(features);// 利用前面建立好的OrbVocabulary对象建立OrbDatabase对象并用于图像的匹配
  return 0;
}
```

重点是testDatabase()部分，先看一下程序的运行结果（只截图了testDatabase()运行效果部分）

```
Creating a small database...
... done!
Database information:
Database: Entries = 4, Using direct index = no. Vocabulary: k = 9, L = 3, Weight
ing = tf-idf, Scoring = L1-norm, Number of words = 697
Querying the database:
Searching for Image 0. 4 results:
<EntryId: 0, Score: 1>
<EntryId: 2, Score: 0.0982032>
<EntryId: 1, Score: 0.0681918>
<EntryId: 3, Score: 0.048554>
Searching for Image 1. 4 results:
<EntryId: 1, Score: 1>
<EntryId: 3, Score: 0.202284>
<EntryId: 2, Score: 0.115883>
<EntryId: 0, Score: 0.0681918>
Searching for Image 2. 4 results:
<EntryId: 2, Score: 1>
<EntryId: 3, Score: 0.140616>
<EntryId: 1, Score: 0.115883>
<EntryId: 0, Score: 0.0982032>
Searching for Image 3. 4 results:
<EntryId: 3, Score: 1>
<EntryId: 1, Score: 0.202284>
<EntryId: 2, Score: 0.140616>
<EntryId: 0, Score: 0.048554>

Saving database...
... done!
Retrieving database once again...
... done! This is:
Database: Entries = 4, Using direct index = no. Vocabulary: k = 9, L = 3, Weight
ing = tf-idf, Scoring = L1-norm, Number of words = 697
```


结果表明程序建立了一个OrbDatabase对象，然后利用它对输入图像在database中检索最为相似的图像。demo表明了OrbDatabase的用途，这VSLAM中回环检测的作用就是在已经保存了关键帧的database中寻找与输入图像相似的图像作为备选图像。

2 代码分析

2.1 准备知识

先介绍DBoW2中的两个概念：正向索引（Direct index）和逆向索引（Inverse index）

正向索引：相对于图像而言，保存一幅图像中每个特征的索引和它在字典中所属的单词索引。意义在于当对两幅图像做涉及到特征点的匹配时，“属于同一单词索引的特征更有可能匹配“的设定规则，来加速匹配。

逆向索引：相对于字典中的单词而言，保存该单词出现过的图像的图像索引，和该单词在图像中的权重。意义在于加速在OrbDatabase中[illegible]图像的计算。

然后介绍这两种概念在DBoW2中的数据结构构造。对于正向索引，每幅图像对应一个FeatureVector对象，看一下FeatureVector的定义是class FeatureVector: public std::map<NodeId, std::vector<unsigned int> >，可以知道它就是一个NodeId和特征索引序列——对应的std::map类。代码addFeature()操作用于更新这个map，且通过insert()操作保证了map中的内容是按照关键字（NodeId）升序排列的。

OrbDatabase中会加入多个图像记录，因此定义了 typedef std::vector<FeatureVector> DirectFile表示多条记录的正向索引。

对于逆向索引，每一个索引对应一个IFRow对象，查看IFRow的定义为 typedef std::list<IFPair> IFRow，那么再查看IFPair的定义如下。

```
struct IFPair
{
  EntryId entry_id;// database中的记录id
  WordValue word_weight;// 该IFPair代表的单词在记录id代表图像中的权重
  IFPair(){}
  IFPair(EntryId eid, WordValue wv): entry_id(eid), word_weight(wv) {}
  inline bool operator==(EntryId eid) const { return entry_id == eid; }
};
```

它其实就是一个<图像id，单词权值>的记录，由于OrbDatabase中有多条记录，所以一个单词对应着多条记录，使用std::list<IFPair> IFRow表示建立了的OrbVocabulary对象中包含多个单词，因此定义typedef std::vector<IFRow> InvertedFile代表逆向索引，且InvertedFile的元素数量等于OrbVocabulary中单词的数量。

2.2 代码分析

testDatabase中的函数定义如下。

```
void testDatabase(const vector<vector<cv::Mat > > &features)
{
  cout << "Creating a small database..." << endl;
  // 加载之前创建并保存的OrbVocabulary文件
  OrbVocabulary voc("small_voc.yml.gz");
  OrbDatabase db(voc, false, 0); //将OrbVocabulary文件作为形参创建OrbDatabase对象

  // 向OrbDatabase中增加记录（代表图像的特征描述子）
  for(int i = 0; i < NIMAGES; i++)
  {
    db.add(features[i]);
  }
  cout << "... done!" << endl;
  cout << "Database information: " << endl << db << endl;

  // 利用OrbDatabase按照相似度高低查询图像
  cout << "Querying the database: " << endl;
  QueryResults ret;
  for(int i = 0; i < NIMAGES; i++)
  {
    db.query(features[i], ret, 4);
    cout << "Searching for Image " << i << ". " << ret << endl;
  }
  cout << endl;

  // 保存
  cout << "Saving database..." << endl;
  db.save("small_db.yml.gz");
  cout << "... done!" << endl;

  // 加载
  cout << "Retrieving database once again..." << endl;
  OrbDatabase db2("small_db.yml.gz");
  cout << "... done! This is: " << endl << db2 << endl;
}
```

1）创建OrbDatabase对象

```
template<class TDescriptor, class F>
template<class T>
TemplatedDatabase<TDescriptor, F>::TemplatedDatabase
```

```
  (const T &voc, bool use_di, int di_levels) 5    : m_voc(NULL), m_use_di(use_di), m_dilevels(di_levels)
{
  setVocabulary(voc);
  clear();
}
```

查看代码发现构造函数中主要做了：将传入的voc赋值给OrbDatabase中的字典成员变量m_voc，然后清空正向索引m_dfile，为逆向索引m_ifile根据m_voc中单词的数量分配空间。

2）向OrbDatabase对象中增加图像记录

```
  for(int i = 0; i < NIMAGES; i++)
  {
    db.add(features[i]);
  }
```

实际上传入的是从图像中提取得到的特征描述子，查看OrbDatabase的add函数，这里根据形参，会先把传入的特征利用m_voc转换为BowVector对象，然后调用temp
TDescriptor, class F>
EntryId TemplatedDatabase<TDescriptor, F>::add(const BowVector &v,
const FeatureVector &fv)函数，该函数的具体内容如下：

```
template<class TDescriptor, class F>
EntryId TemplatedDatabase<TDescriptor, F>::add(const BowVector &v,
  const FeatureVector &fv)
{
  EntryId entry_id = m_nentries++;// 保存database中记录数量
  BowVector::const_iterator vit;
  std::vector<unsigned int>::const_iterator iit;

  if(m_use_di)// demo中这里不执行
  {
    // update direct file
    if(entry_id == m_dfile.size())
    {
      m_dfile.push_back(fv);
    }
    else
    {
      m_dfile[entry_id] = fv;
    }
  }
  // 实际上就是在构造逆向索引的内容
  for(vit = v.begin(); vit != v.end(); ++vit)
  {
    const WordId& word_id = vit->first;
    const WordValue& word_weight = vit->second;

    IFRow& ifrow = m_ifile[word_id];
    ifrow.push_back(IFPair(entry_id, word_weight));
  }
  return entry_id;
}
```

发现其实函数中做的就是利用m_voc中已经生成的单词和输入图像转换得到的BowVector构造逆向索引。

3）利用OrbDatabase检索图像

之后程序中利用构造好的OrbDatabase检索database中与输入图像相似度高的图像。

```
  QueryResults ret;
  for(int i = 0; i < NIMAGES; i++)
  {
    db.query(features[i], ret, 4);
    cout << "Searching for Image " << i << ". " << ret << endl;
  }
```

QueryResults用于保存检索结果，具体定义查看代码即可。调用OrbDatabase的query()函数进行检索，查看代码发现函数中首先把传入的特征和换为BowVector，然后根据m_voc的评分规则调用不同的query分支（后面发现其实query中就是利用m_voc中计算图像间相似度的规则来做检索），的是queryL1()分支，内容如下。

```
template<class TDescriptor, class F>
void TemplatedDatabase<TDescriptor, F>::queryL1(const BowVector &vec,
  QueryResults &ret, int max_results, int max_id) const
{
  BowVector::const_iterator vit;
  typename IFRow::const_iterator rit;
  std::map<EntryId, double> pairs;// 保存输入图像与EntryId对应图像之间的相似度
  std::map<EntryId, double>::iterator pit;
  for(vit = vec.begin(); vit != vec.end(); ++vit)
  {
    const WordId word_id = vit->first;
    const WordValue& qvalue = vit->second;
    const IFRow& row = m_ifile[word_id];// 利用之前记录的单词word_id在database的各个图像的权值
    for(rit = row.begin(); rit != row.end(); ++rit)// 计算输入图像与database中各个图像的相似度
    {
      const EntryId entry_id = rit->entry_id;
      const WordValue& dvalue = rit->word_weight;
      if((int)entry_id < max_id || max_id == -1)
      {
        double value = fabs(qvalue - dvalue) - fabs(qvalue) - fabs(dvalue);// 计算规则与OrbVocabulary中相同
        pit = pairs.lower_bound(entry_id);
        if(pit != pairs.end() && !(pairs.key_comp()(entry_id, pit->first)))
        {
          pit->second += value;
        }
        else
        {
          pairs.insert(pit,
            std::map<EntryId, double>::value_type(entry_id, value));
        }
      }

    } // for each inverted row
  } // for each query word

  ret.reserve(pairs.size());
  for(pit = pairs.begin(); pit != pairs.end(); ++pit)
  {
    ret.push_back(Result(pit->first, pit->second));
  }

  std::sort(ret.begin(), ret.end());// 升序排列，由于前面计算的score带有负号，因此score值越小相似度越高

  if(max_results > 0 && (int)ret.size() > max_results)
    ret.resize(max_results);
  QueryResults::iterator qit;
  for(qit = ret.begin(); qit != ret.end(); qit++)
    qit->Score = -qit->Score/2.0;// 真正的score，介于[0,1]之间，值越大代表相似度越高
}
```

4）保存与加载

主要利用cv::FileStorage进行读写，不再叙述。

3. 小结

总结一下DBoW2中OrbDatabase和OrbVocabulary各自的用途。

OrbVocabulary需要一个图像集合输入，它会提取图像特征然后做聚类操作，形成一个很多表征一类特征的“单词”组成的词典。有了这个，它将两幅输入图像做“图像-特征-BowVector对象”的转换，之后计算相似度。

OrbDatabase构造时需要一个OrbVocabulary对象作为输入，目的是得到已经聚类生成的“单词”。之后通过向OrbDatabase中加入图像i方两个映射关系：正向索引和逆向索引，设计它们的目的都是为了加速匹配计算。逆向索引可加速在database中多幅图像中寻找与输入图像最为相误程。正向索引主要用在计算输入图像与备选图像间特征匹配关系时加快匹配速度。