# 使用Ceres库做BA优化相机姿态

《视觉SLAM十四讲从理论到实践》第七章使用g2o库对相机姿态进行优化，下面是自己基于Ceres库的实现，结果与课本有一定误差，将实现过程记录下来，便于后续参考。

## 1.Ceres使用步骤

### 1.1定义误差结构体

在误差结构体中实现重投影误差的计算。对于每一组二维图像点和三维空间点，二维图像点是观测值，是固定的，三维空间点及相机姿态是优化变量。因此在结果体中定义两个观测变量\_observed\_x和\_observed\_y，并在结构体的构造函数中传入。接着对**operator()**函数进行实现，该函数是一个模板函数，输入参数是优化变量指针和误差指针，这里的优化变量是pose和point，前者是一个6维向量，分别是旋转向量（3维）和平移向量（3维），后者是空间点，是一个3维向量，误差是一个2维向量，分别是x方向的重投影误差和y方向的重投影误差，注意这里误差是估计值与观测值之差，而没有进行平方。

|  |
| --- |
| struct ReprojectionError  **{**  //传入观测值  ReprojectionError**(**double observed\_x**,** double observed\_y**)**  **:** \_observed\_x**(**observed\_x**),** \_observed\_y**(**observed\_y**){}**  template**<**typename T**>**  //需要优化的变量：姿态([0,1,2] are the angle-axis rotaiton, [3, 4, 5] are the translation)及空间点坐标  bool **operator()(**const T**\*** const pose**,** const T**\*** const point**,** T**\*** residuals**)** const  **{**  //将空间点变换到相机坐标系下  T p**[**3**];**  ceres**::**AngleAxisRotatePoint**(**pose**,** point**,** p**);**  p**[**0**]** **+=** pose**[**3**];**  p**[**1**]** **+=** pose**[**4**];**  p**[**2**]** **+=** pose**[**5**];**  //再投影到图像平面  T u **=** \_fx **\*** p**[**0**]** **/** p**[**2**]** **+** \_cx**;**  T v **=** \_fy **\*** p**[**1**]** **/** p**[**2**]** **+** \_cy**;**  //计算重投影误差  residuals**[**0**]** **=** **(**\_observed\_x **-** u**);**  residuals**[**1**]** **=** **(**\_observed\_y **-** v**);**  **return** **true;**  **}**  private**:**  //观测值  double \_observed\_x**,** \_observed\_y**;**  //相机参数  double \_fx **=** 520.9**,** \_fy **=** 521.0**,** \_cx **=** 325.1**,** \_cy **=** 249.7**;**  **};** |

### 1.2 创建问题并向其添加每项数据的残差块

使用Problem创建问题，对每项观测数据，创建一个CostFunction，创建方式如代码中所示，其中2为残差的维度，6为姿态的维度，3为空间点的维度，前者为输出变量，后2者为输入变量，其顺序参照误差结构体中的顺序。

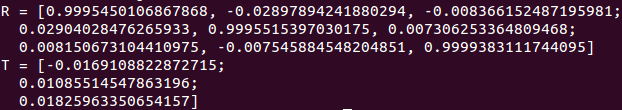
|  |
| --- |
| //1.创建问题  Problem problem**;**  //2.根据输入数据创建CostFuctor，添加到problem中，同时传入优化变量  **for(**int i **=** 0**;**i**<**points\_2d**.**size**()** **&&** i **<** points\_3d**.**size**();**i**++)**  **{**  double observed\_x **=** points\_2d**[**i**].**x**;**  double observed\_y **=** points\_2d**[**i**].**y**;**  CostFunction**\*** cost\_function **=** **new** AutoDiffCostFunction**<**ReprojectionError**,** 2**,** 6**,** 3**>(new** ReprojectionError**(**observed\_x**,** observed\_y**));**  problem**.**AddResidualBlock**(**cost\_function**,** **NULL,** pose**,** points **+** 3 **\*** i**);**  **}** |

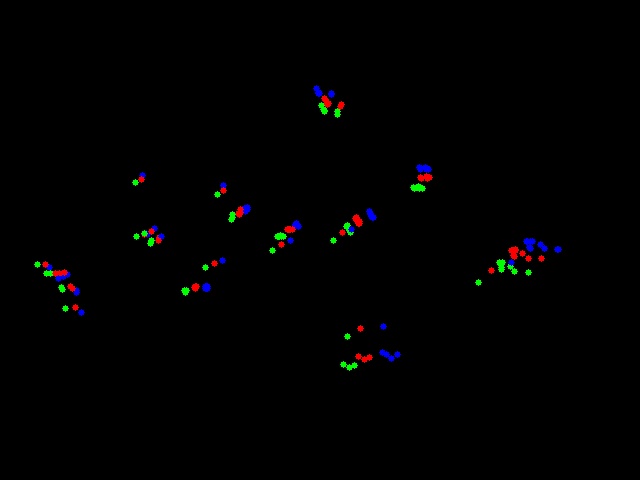
### 1.3 优化

优化的代码比较固定，如下：

|  |
| --- |
| Solver**::**Options options**;**  options**.**linear\_solver\_type **=** ceres**::**DENSE\_SCHUR**;**  options**.**minimizer\_progress\_to\_stdout **=** **true;**  Solver**::**Summary summary**;**  Solve**(**options**,** **&**problem**,** **&**summary**);** |

## 2.实验结果





图中绿色点是观测值，蓝色点为未优化前的重投影值，红色为优化后的重投影值，可见优化后重投影误差有明显减小，但仍未与观测值重合。实验结果与课本结果存在一定误差，未知道改善方法。

## 3.程序文件

### 3.1 comment.h

|  |
| --- |
| #ifndef COMMENT\_H  #define COMMENT\_H  #include <cmath>  #include <cstdio>  #include <iostream>  #include "ceres/ceres.h"  #include "ceres/rotation.h"  #include <iostream>  #include <opencv2/core/core.hpp>  #include <opencv2/features2d/features2d.hpp>  #include <opencv2/highgui/highgui.hpp>  #include <opencv2/calib3d/calib3d.hpp>  **using** **namespace** std**;**  **using** **namespace** cv**;**  **using** ceres**::**AutoDiffCostFunction**;**  **using** ceres**::**CostFunction**;**  **using** ceres**::**Problem**;**  **using** ceres**::**Solver**;**  **using** ceres**::**Solve**;**  void find\_feature\_matches **(**  const Mat**&** img\_1**,** const Mat**&** img\_2**,**  std**::**vector**<**KeyPoint**>&** keypoints\_1**,**  std**::**vector**<**KeyPoint**>&** keypoints\_2**,**  std**::**vector**<** DMatch **>&** matches **);**  // 像素坐标转相机归一化坐标  Point2d pixel2cam **(** const Point2d**&** p**,** const Mat**&** K **);**  void bundleAdjustment **(**  vector**<** Point3f **>** points\_3d**,**  const vector**<** Point2f **>** points\_2d**,**  const Mat**&** K**,**  Mat**&** R**,** Mat**&** t **);**  struct ReprojectionError  **{**  //传入观测值  ReprojectionError**(**double observed\_x**,** double observed\_y**)**  **:** \_observed\_x**(**observed\_x**),** \_observed\_y**(**observed\_y**){}**  template**<**typename T**>**  //需要优化的变量：姿态([0,1,2] are the angle-axis rotaiton, [3, 4, 5] are the translation)及空间点坐标  bool **operator()(**const T**\*** const pose**,** const T**\*** const point**,** T**\*** residuals**)** const  **{**  //将空间点变换到相机坐标系下  T p**[**3**];**  ceres**::**AngleAxisRotatePoint**(**pose**,** point**,** p**);**  p**[**0**]** **+=** pose**[**3**];**  p**[**1**]** **+=** pose**[**4**];**  p**[**2**]** **+=** pose**[**5**];**  //再投影到图像平面  T u **=** \_fx **\*** p**[**0**]** **/** p**[**2**]** **+** \_cx**;**  T v **=** \_fy **\*** p**[**1**]** **/** p**[**2**]** **+** \_cy**;**  //计算重投影误差  residuals**[**0**]** **=** **(**\_observed\_x **-** u**)** **\*** **(**\_observed\_x **-** u**);**  residuals**[**1**]** **=** **(**\_observed\_y **-** v**)** **\*** **(**\_observed\_y **-** v**);**  **return** **true;**  **}**  private**:**  //观测值  double \_observed\_x**,** \_observed\_y**;**  //相机参数  double \_fx **=** 520.9**,** \_fy **=** 521.0**,** \_cx **=** 325.1**,** \_cy **=** 249.7**;**  **};**  #endif |

### 3.2 comment.cpp文件

|  |
| --- |
| #include "comment.h"  void find\_feature\_matches **(** const Mat**&** img\_1**,** const Mat**&** img\_2**,**  std**::**vector**<**KeyPoint**>&** keypoints\_1**,**  std**::**vector**<**KeyPoint**>&** keypoints\_2**,**  std**::**vector**<** DMatch **>&** matches **)**  **{**  //-- 初始化  Mat descriptors\_1**,** descriptors\_2**;**  // used in OpenCV3  // Ptr<FeatureDetector> detector = ORB::create();  //Ptr<DescriptorExtractor> descriptor = ORB::create();  // use this if you are in OpenCV2  Ptr**<**FeatureDetector**>** detector **=** FeatureDetector**::**create **(** "ORB" **);**  Ptr**<**DescriptorExtractor**>** descriptor **=** DescriptorExtractor**::**create **(** "ORB" **);**  Ptr**<**DescriptorMatcher**>** matcher **=** DescriptorMatcher**::**create **(** "BruteForce-Hamming" **);**  //-- 第一步:检测 Oriented FAST 角点位置  detector**->**detect **(** img\_1**,**keypoints\_1 **);**  detector**->**detect **(** img\_2**,**keypoints\_2 **);**  //-- 第二步:根据角点位置计算 BRIEF 描述子  descriptor**->**compute **(** img\_1**,** keypoints\_1**,** descriptors\_1 **);**  descriptor**->**compute **(** img\_2**,** keypoints\_2**,** descriptors\_2 **);**  //-- 第三步:对两幅图像中的BRIEF描述子进行匹配，使用 Hamming 距离  vector**<**DMatch**>** match**;**  // BFMatcher matcher ( NORM\_HAMMING );  matcher**->**match **(** descriptors\_1**,** descriptors\_2**,** match **);**  //-- 第四步:匹配点对筛选  double min\_dist**=**10000**,** max\_dist**=**0**;**  //找出所有匹配之间的最小距离和最大距离, 即是最相似的和最不相似的两组点之间的距离  **for** **(** int i **=** 0**;** i **<** descriptors\_1**.**rows**;** i**++** **)**  **{**  double dist **=** match**[**i**].**distance**;**  **if** **(** dist **<** min\_dist **)** min\_dist **=** dist**;**  **if** **(** dist **>** max\_dist **)** max\_dist **=** dist**;**  **}**  printf **(** "-- Max dist : %f \n"**,** max\_dist **);**  printf **(** "-- Min dist : %f \n"**,** min\_dist **);**  //当描述子之间的距离大于两倍的最小距离时,即认为匹配有误.但有时候最小距离会非常小,设置一个经验值30作为下限.  **for** **(** int i **=** 0**;** i **<** descriptors\_1**.**rows**;** i**++** **)**  **{**  **if** **(** match**[**i**].**distance **<=** max **(** 2**\***min\_dist**,** 30.0 **)** **)**  **{**  matches**.**push\_back **(** match**[**i**]** **);**  **}**  **}**  **}**  Point2d pixel2cam **(** const Point2d**&** p**,** const Mat**&** K **)**  **{**  **return** Point2d  **(**  **(** p**.**x **-** K**.**at**<**double**>** **(** 0**,**2 **)** **)** **/** K**.**at**<**double**>** **(** 0**,**0 **),**  **(** p**.**y **-** K**.**at**<**double**>** **(** 1**,**2 **)** **)** **/** K**.**at**<**double**>** **(** 1**,**1 **)**  **);**  **}**  void bundleAdjustment **(**  vector**<** Point3f **>** points\_3d**,**  const vector**<** Point2f **>** points\_2d**,**  const Mat**&** K**,**  Mat**&** R**,** Mat**&** t **)**  **{**  cout **<<** "enter bundledAdjustment" **<<** endl**;**  //0.优化姿态  double pose**[**6**]** **=** **{**0**,** 0**,** 0**,** 0**,** 0**,** 0**};**  double **\***points **=** **new** double**[**points\_3d**.**size**()** **\*** 3**];**  int j **=** 0**;**  **for(**int i **=** 0**;**i**<**points\_3d**.**size**();**i**++)**  **{**  points**[**j**++]** **=** points\_3d**[**i**].**x**;**  points**[**j**++]** **=** points\_3d**[**i**].**y**;**  points**[**j**++]** **=** points\_3d**[**i**].**z**;**  **}**  //1.创建问题  Problem problem**;**  //2.根据输入数据创建CostFuctor，添加到problem中，同时传入优化变量  **for(**int i **=** 0**;**i**<**points\_2d**.**size**()** **&&** i **<** points\_3d**.**size**();**i**++)**  **{**  double observed\_x **=** points\_2d**[**i**].**x**;**  double observed\_y **=** points\_2d**[**i**].**y**;**  CostFunction**\*** cost\_function **=** **new** AutoDiffCostFunction**<**ReprojectionError**,** 2**,** 6**,** 3**>(new** ReprojectionError**(**observed\_x**,** observed\_y**));**  problem**.**AddResidualBlock**(**cost\_function**,** **NULL,** pose**,** points **+** 3 **\*** i**);**  **}**  //优化  Solver**::**Options options**;**  options**.**linear\_solver\_type **=** ceres**::**DENSE\_SCHUR**;**  options**.**minimizer\_progress\_to\_stdout **=** **true;**  Solver**::**Summary summary**;**  Solve**(**options**,** **&**problem**,** **&**summary**);**  cout **<<** summary**.**FullReport**()** **<<** endl**;**  cout **<<** "pose: " **<<** pose**[**0**]** **<<** " "**<<** pose**[**1**]** **<<** " "**<<** pose**[**2**]** **<<** " "  **<<** pose**[**3**]** **<<** " "**<<** pose**[**4**]** **<<** " "**<<** pose**[**5**]** **<<** endl**;**  Mat r **=** **(**Mat\_**<**double**>(**3**,** 1**)** **<<** pose**[**0**],** pose**[**1**],** pose**[**2**]);**  t **=** **(**Mat\_**<**float**>(**3**,** 1**)** **<<** pose**[**3**],** pose**[**4**],** pose**[**5**]);**  cv**::**Rodrigues **(**r**,** R**);**  **delete** **[]** points**;**  points **=** **NULL;**  cout **<<** "leave bundledAdjustment" **<<** endl**;**  **}** |

### 3.3 main.cpp

|  |
| --- |
| #include "comment.h"  int main**(**int argc**,** char**\*** argv**[])**  **{**  **if** **(** argc **!=** 5 **)**  **{**  cout**<<**"usage: pose\_estimation\_3d2d img1 img2 depth1 depth2"**<<**endl**;**  **return** 1**;**  **}**  //-- 读取图像  Mat img\_1 **=** imread **(** argv**[**1**],** CV\_LOAD\_IMAGE\_COLOR **);**  Mat img\_2 **=** imread **(** argv**[**2**],** CV\_LOAD\_IMAGE\_COLOR **);**  vector**<**KeyPoint**>** keypoints\_1**,** keypoints\_2**;**  vector**<**DMatch**>** matches**;**  find\_feature\_matches **(** img\_1**,** img\_2**,** keypoints\_1**,** keypoints\_2**,** matches **);**  cout**<<**"一共找到了"**<<**matches**.**size**()** **<<**"组匹配点"**<<**endl**;**  // 建立3D点  Mat d1 **=** imread **(** argv**[**3**],** CV\_LOAD\_IMAGE\_UNCHANGED **);** // 深度图为16位无符号数，单通道图像  Mat K **=** **(** Mat\_**<**double**>** **(** 3**,**3 **)** **<<** 520.9**,** 0**,** 325.1**,** 0**,** 521.0**,** 249.7**,** 0**,** 0**,** 1 **);**  vector**<**Point3f**>** pts\_3d**;**  vector**<**Point2f**>** pts\_2d**;**  **for** **(** DMatch m**:**matches **)**  **{**  ushort d **=** d1**.**ptr**<**unsigned short**>** **(**int **(** keypoints\_1**[**m**.**queryIdx**].**pt**.**y **))** **[** int **(** keypoints\_1**[**m**.**queryIdx**].**pt**.**x **)** **];**  **if** **(** d **==** 0 **)** // bad depth  **continue;**  float dd **=** d**/**5000.0**;**  Point2d p1 **=** pixel2cam **(** keypoints\_1**[**m**.**queryIdx**].**pt**,** K **);**  pts\_3d**.**push\_back **(** Point3f **(** p1**.**x**\***dd**,** p1**.**y**\***dd**,** dd **)** **);**  pts\_2d**.**push\_back **(** keypoints\_2**[**m**.**trainIdx**].**pt **);**  **}**  cout**<<**"3d-2d pairs: "**<<**pts\_3d**.**size**()** **<<**endl**;**  //BA求解相机姿态及点的空间坐标  Mat R**,** t**;**  bundleAdjustment**(**pts\_3d**,** pts\_2d**,** K**,** R**,** t**);**  cout **<<** "R = " **<<** R **<<** endl**;**  cout **<<** "T = " **<<** t **<<** endl**;**  **return** 0**;**  **}** |

### 3.4 CMakeLists.txt

|  |
| --- |
| **cmake\_minimum\_required**(**VERSION** 2.8 **FATAL\_ERROR**)  #set(CMAKE\_BUILD\_TYPE Release)  **project**(test)  **find\_package**(Ceres **REQUIRED**)  **include\_directories**(${CERES\_INCLUDE\_DIRS})  **find\_package**(OpenCV **REQUIRED**)  **include\_directories**(${OpenCV\_INCLUDE\_DIRS})  **link\_directories**(${OpenCV\_LIBRARY\_DIRS})  **aux\_source\_directory**(. SRC\_LIST)  **add\_executable**(${PROJECT\_NAME} ${SRC\_LIST})  **target\_link\_libraries**(test ${OpenCV\_LIBS})  **target\_link\_libraries**(test ${CERES\_LIBRARIES}) |