山东大学 计算机科学与技术 学院

计算机视觉 课程实验报告

|  |  |  |
| --- | --- | --- |
| 学号：201600301304 | 姓名：贾乘兴 | 班级：人工智能16 |
| 实验题目：特征检测与匹配 | | |
| 实验内容：   1. SIFT特征检测与匹配   1. Sift（尺度不变特征变换），全称为Scale Invariant Feature Transform  Sift提取图像的局部特征，在尺度空间寻找极值点，并提取出其位置、尺度、方向信息。Sift特征具有以下的特点：  1).对旋转、尺度缩放、亮度变化保持不变性，对视角变化、噪声等也存在一定程度的稳定性；  2).独特性，信息量丰富，适用于在海量特征数据中进行快速，准确的匹配；  3).多量性，即使少数几个物体也可以产生大量的Sfit特征向量；  4).可扩展性，可以很方便的与其他形式的特征向量进行联合；  Sfit算法的实质是在不同的尺度空间上查找关键点（特征点），计算关键点的大小、方向、尺度信息，利用这些信息组成关键点对特征点进行描述的问题。Sift所查找的关键点都是一些十分突出，不会因光照，仿射便函和噪声等因素而变换的“稳定”特征点，如角点、边缘点、暗区的亮点以及亮区的暗点等。匹配的过程就是对比这些特征点的过程，这个流程可以用下图表述：  ../../9a.png  2. SIFT特征提取算法  首先生成高斯金字塔，逐层将图像进行高斯滤波然后重采样，不断缩小图像大小，生成金字塔，随后进行高斯金字塔的差分，得到DoG金字塔。对于得到的DoG金字塔，进行空间的极值点的检测，即对周围的26邻域进行比较。  ../../9b.png  DoG值对噪声和边缘比较敏感，所以在尺度空间中检测到的局部极值点还要经过进一步的筛选，去除不稳定和错误检测出的极值点，另一点就是在构建高斯金字塔过程中采用了下采样的图像，在下采样图像中提取的极值点对应在原始图像中的确切位置稳定的极值点是在不同尺度空间下提取的，这保证了关键点的尺度不变性。为关键点分配方向信息所要解决的问题是使得关键点对图像角度和旋转具有不变性。方向的分配是通过求每个极值点的梯度来实现的。  梯度大小计算如下  ../../9c.png  梯度角的计算如下  ../../9d.png  分配给关键点的方向并不直接是关键点的梯度方向，而是按照一种梯度方向直方图的方式给出的。具体的方法是：计算以关键点为中心的邻域内所有点的梯度方向，当然梯度方向一定是在0~360°范围内，对这些梯度方向归一化到36个方向内，每个方向代表了10°的范围。然后累计落到每个方向内的关键点个数，以此生成梯度方向直方图。  ../../9e.png  将梯度方向直方图中纵坐标最大的项代表的方向分配给当前关键点作为主方向，若在梯度直方图中存在一个相当于主峰值80%能量的峰值，则将这个方向认为是关键点的辅方向。辅方向的设计可以增强匹配的鲁棒性，Lowe指出，大概有15%的关键点具有辅方向，而恰恰是这15%的关键点对稳定匹配起到关键作用。对关键点的描述是后续实现匹配的关键步骤，描述其实就是一种以数学方式定义关键的过程。描述子不但包含关键点，也包括关键点周围对其有贡献的邻域点。  ../../9f.png  最后进行特征点匹配，特征点的匹配是通过计算两组特征点的128维的关键点的欧式距离实现的。欧式距离越小，则相似度越高，当欧式距离小于设定的阈值时，可以判定为匹配成功。  3. 代码  #include **<iostream>**  #include **<vector>**  #include **<opencv2/xfeatures2d.hpp>**  #include **<opencv2/highgui.hpp>**  **using namespace** cv;  **using namespace** std;  **int** main()  {  Mat img1 = imread(**"/Users/apple/Desktop/13.jpg"**, ***IMREAD\_GRAYSCALE***);  Mat img2 = imread(**"/Users/apple/Desktop/14.png"**, ***IMREAD\_GRAYSCALE***);  **if** ((img1.data == **NULL**)||(img2.data ==**NULL**))  {  cout << **"No exist"** << endl;  **return** -1;  }  Ptr<Feature2D> feature = xfeatures2d::SIFT::create(1000);  vector<KeyPoint> keypoints\_1, keypoints\_2;  Mat descriptors\_1, descriptors\_2;  feature->detectAndCompute(img1, Mat(), keypoints\_1, descriptors\_1 );  feature->detectAndCompute(img2, Mat(), keypoints\_2, descriptors\_2 );  drawKeypoints(img1, keypoints\_1, img1);  drawKeypoints(img2, keypoints\_2, img2);  FlannBasedMatcher matcher;  std::vector< DMatch > matches;  matcher.match( descriptors\_1, descriptors\_2, matches );  **double** max\_dist = 0; **double** min\_dist = 100;  **for**( **int** i = 0; i < descriptors\_1.rows; i++ )  {  **double** dist = matches[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 );  std::vector< DMatch > good\_matches;  **for**( **int** i = 0; i < descriptors\_1.rows; i++ )  {  **if**( matches[i].distance <= max(2\*min\_dist, 0.02) )  {  good\_matches.push\_back( matches[i]);  }  }  Mat img\_matches;  drawMatches( img1, keypoints\_1, img2, keypoints\_2,  good\_matches, img\_matches, Scalar::all(-1), Scalar::all(-1),  vector<**char**>(), DrawMatchesFlags::***NOT\_DRAW\_SINGLE\_POINTS*** );  namedWindow(**"Good Matches"**,0);  resizeWindow(**"Good Matches"**,800,800);  imshow( **"Good Matches"**, img\_matches );  imwrite(**"/Users/apple/Desktop/SI.jpg"**,img\_matches);  **for**( **int** i = 0; i < (**int**)good\_matches.size(); i++ )  {  printf( **"-- Good Match** [**%d**] **Keypoint 1: %d  -- Keypoint 2: %d  \n"**, i, good\_matches[i].queryIdx, good\_matches[i].trainIdx );  }  waitKey(0);  **return** 0;  }  4. 实现结果  sift.jpg   1. SURF特征检测与匹配   1. Sift算法的优点是特征稳定，对旋转、尺度变换、亮度保持不变性，对视角变换、噪声也有一定程度的稳定性；缺点是实时性不高，并且对于边缘光滑目标的特征点提取能力较弱。SURF对SIFT进行了一定的改进，具体算法如下  1).构建Hessian，生成所有的兴趣点，用于特征的提取；  2).构建尺度空间  3).特征点定位  4).特征点主方向分配  5).生成特征点描述子  6).特征点匹配  该算法更加高效的实现了特征检测与匹配，且性质较好  2. 代码  #include **<iostream>**  #include **<vector>**  #include **<opencv2/xfeatures2d.hpp>**  #include **<opencv2/highgui.hpp>**  **using namespace** cv;  **using namespace** std;  **int** main()  {  Mat img1 = imread(**"/Users/apple/Desktop/13.jpg"**, ***IMREAD\_GRAYSCALE***);  Mat img2 = imread(**"/Users/apple/Desktop/14.png"**, ***IMREAD\_GRAYSCALE***);  **if** ((img1.data == **NULL**)||(img2.data ==**NULL**))  {  cout << **"No exist"** << endl;  **return** -1;  }  Ptr<Feature2D> feature = xfeatures2d::SURF::create(1000);  vector<KeyPoint> keypoints\_1, keypoints\_2;  Mat descriptors\_1, descriptors\_2;  feature->detectAndCompute(img1, Mat(), keypoints\_1, descriptors\_1 );  feature->detectAndCompute(img2, Mat(), keypoints\_2, descriptors\_2 );  drawKeypoints(img1, keypoints\_1, img1);  drawKeypoints(img2, keypoints\_2, img2);  FlannBasedMatcher matcher;  std::vector< DMatch > matches;  matcher.match( descriptors\_1, descriptors\_2, matches );  **double** max\_dist = 0; **double** min\_dist = 100;  **for**( **int** i = 0; i < descriptors\_1.rows; i++ )  {  **double** dist = matches[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 );  std::vector< DMatch > good\_matches;  **for**( **int** i = 0; i < descriptors\_1.rows; i++ )  {  **if**( matches[i].distance <= max(2\*min\_dist, 0.02) )  {  good\_matches.push\_back( matches[i]);  }  }  Mat img\_matches;  drawMatches( img1, keypoints\_1, img2, keypoints\_2,  good\_matches, img\_matches, Scalar::all(-1), Scalar::all(-1),  vector<**char**>(), DrawMatchesFlags::***NOT\_DRAW\_SINGLE\_POINTS*** );  namedWindow(**"Good Matches"**,0);  resizeWindow(**"Good Matches"**,800,800);  imshow( **"Good Matches"**, img\_matches );  imwrite(**"/Users/apple/Desktop/SI.jpg"**,img\_matches);  **for**( **int** i = 0; i < (**int**)good\_matches.size(); i++ )  {  printf( **"-- Good Match** [**%d**] **Keypoint 1: %d  -- Keypoint 2: %d  \n"**, i, good\_matches[i].queryIdx, good\_matches[i].trainIdx );  }  waitKey(0);  **return** 0;  }  3. 结果  surf.jpg   1. ORB特征检测与匹配 2. ORB特征检测与匹配   ORB特征是目前最优秀的特征提取与匹配算法之一  1.1 Fast 特征提取  **步骤一**：粗提取。该步能够提取大量的特征点，但是有很大一部分的特征点的质量不高。下面介绍提取方法。从图像中选取一点P，如上图1。我们判断该点是不是特征点的方法是，以P为圆心画一个半径为3pixel的圆。圆周上如果有连续n个像素点的灰度值比P点的灰度值大或者小，则认为P为特征点。一般n设置为12。为了加快特征点的提取，快速排出非特征点，首先检测1、9、5、13位置上的灰度值，如果P是特征点，那么这四个位置上有3个或3个以上的的像素值都大于或者小于P点的灰度值。如果不满足，则直接排出此点。  **步骤二**：机器学习的方法筛选最优特征点。简单来说就是使用ID3算法训练一个决策树，将特征点圆周上的16个像素输入决策树中，以此来筛选出最优的FAST特征点。  **步骤三**：非极大值抑制去除局部较密集特征点。使用非极大值抑制算法去除临近位置多个特征点的问题。为每一个特征点计算出其响应大小。计算方式是特征点P和其周围16个特征点偏差的绝对值和。在比较临近的特征点中，保留响应值较大的特征点，删除其余的特征点。  **步骤四**：特征点的尺度不变形。建立金字塔，来实现特征点的多尺度不变性。设置一个比例因子scaleFactor和金字塔的层数nlevels。将原图像按比例因子缩小成nlevels幅图像。缩放后的图像为：I’= I/scaleFactork(k=1,2,…, nlevels)。nlevels幅不同比例的图像提取特征点总和作为这幅图像的oFAST特征点。  **步骤五**：特征点的旋转不变性。ORB算法提出使用矩（moment）法来确定FAST特征点的方向。也就是说通过矩来计算特征点以r为半径范围内的质心，特征点坐标到质心形成一个向量作为该特征点的方向。矩定义如下：    其中，I(x,y)为图像灰度表达式。该矩的质心为：    假设角点坐标为*O*，则向量的角度即为该特征点的方向。计算公式如下：    1.2 **改进BRIEF算法—rBRIEF**  首先建立300k个特征点测试集。对于测试集中的每个点，考虑其31x31邻域。这里不同于原始BRIEF算法的地方是，这里在对图像进行高斯平滑之后，使用邻域中的某个点的5x5邻域灰度平均值来代替某个点对的值，进而比较点对的大小。这样特征值更加具备抗噪性。另外可以使用积分图像加快求取5x5邻域灰度平均值的速度。  从上面可知，在31x31的邻域内共有(31-5+1)x(31-5+1)=729个这样的子窗口，那么取点对的方法共有M=265356种，我们就要在这Ｍ种方法中选取256种取法，选择的原则是这256种取法之间的相关性最小，选取方法如下  1）在300k特征点的每个31x31邻域内按M种方法取点对，比较点对大小，形成一个300kxM的二进制矩阵Q。矩阵的每一列代表300k个点按某种取法得到的二进制数。  2）对Q矩阵的每一列求取平均值，按照平均值到0.5的距离大小重新对Q矩阵的列向量排序，形成矩阵T。  3）将T的第一列向量放到R中。  4）取T的下一列向量和R中的所有列向量计算相关性，如果相关系数小于设定的阈值，则将T中的该列向量移至R中。  5）按照4）的方式不断进行操作，直到R中的向量数量为256。  通过这种方法就选取了这256种取点对的方法。   1. 代码   #include **<opencv2/core/core.hpp>**  #include **<opencv2/features2d/features2d.hpp>**  #include **<opencv2/highgui/highgui.hpp>**  **using namespace** std;  **using namespace** cv;  **int** main()  {  Mat rgbd1 = imread(**"/Users/apple/Desktop/13.jpg"**);  Mat rgbd2 = imread(**"/Users/apple/Desktop/14.png"**);  Ptr<ORB> orb = ORB::create(50);  vector<KeyPoint> Keypoints1,Keypoints2;  Mat descriptors1,descriptors2;  orb->detectAndCompute(rgbd1, Mat(), Keypoints1, descriptors1);  orb->detectAndCompute(rgbd1, Mat(), Keypoints2, descriptors2);  Mat ShowKeypoints1, ShowKeypoints2;  drawKeypoints(rgbd1,Keypoints1,ShowKeypoints1);  drawKeypoints(rgbd2, Keypoints2, ShowKeypoints2);  *//imshow*(*"Keypoints1"*, *ShowKeypoints1*);  *//imshow*(*"Keypoints2"*, *ShowKeypoints2*);  *//waitKey*(*0*);  *//Matching*  vector<DMatch> matches;  Ptr<DescriptorMatcher> matcher =DescriptorMatcher::create(**"BruteForce"**);  matcher->match(descriptors1, descriptors2, matches);  cout << **"find out total "** << matches.size() << **" matches"** << endl;  Mat ShowMatches;  drawMatches(rgbd1,Keypoints1,rgbd2,Keypoints2,matches,ShowMatches);  imshow(**"matches"**, ShowMatches);  imwrite(**"/Users/apple/Desktop/ORB.jpg"**,ShowMatches);  waitKey(0);  **return** 0;  }   1. 结果   ORB.jpg | | |
| 实验过程中遇到和解决的问题：  （记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）  解决opencv3的sift等特征匹配的代码问题 | | |
| 结论分析与体会： 通过本次实验，了解了较好的特征提取与匹配算法，sift为最经典的算法，surf要更快，orb据说则快sift接近100倍 | | |