Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой_____

Преподаватель _____
(подпись)
«_____» _____ 2018 г.

Отчет
по лабораторной работе №_5_

по дисциплине «Интеллектуальные технологии обработки изображений»

ЛР 09.04.04.20.000 О

Студент группы 8ПИ-61 А.Г. Шевелёва
(И. О., Фамилия)

Преподаватель старший преподаватель М.Г. Казаков
должность, ученое звание (И. О., Фамилия)

Барнаул 2018

Постановка задачи:

• Реализовать относительную инвариантность вычисления дескрипторов относительно вращения изображений на основе подхода SIFT.

• Реализовать этап оценки ориентации интересной точки и поворота сетки, в которой вычисляются гистограммы градиентов.

• Оценить полученный алгоритм с точки зрения реакции на соответствующие искажения изображений, сравнить с полученным в четвертой работе.

Решение:

Алгоритм.

• Строим широкую гистограмму распределения ориентаций градиентов в области (например, 36 корзин)

• При взвешивании магнитуд используем веса по Гауссу

• Выбираем пиковое значение

• Если есть другие пики – добавляем их тоже, считаем как разные точки интереса

• При вычислении окрестности учитываем ориентацию точки, при повороте изображения получим одинаковые результаты

Рисунок 1 — Результат работы алгоритма

Исходный код программы:

Descriptors.cpp

```
vector<Point> Descriptors::meth1(vector<Point> points){
   vector<Point> points1;
   const int localBinCount = 36;

//    unique_ptr<double []> localBin;
//    localBin = make_unique<double []>(localBinCount);
//double localBin[localBinCount];
   vector<double> localBin;

   for(int index = 0; index < points.size(); index++) {
     Point point = points.at(index);
      double localBinSize = 360.0 / localBinCount;
       int radius = 8;


      for(int i = 0; i < localBinCount; i++)
         if(index == 0)
            localBin.push_back(0);
         else
            localBin.at(i) = 0;

      for(int y = -radius; y < radius; y++){
         for(int x = -radius; x < radius; x++){

            //В пределах ?
            if(getDistanceAroundPoints((double)x, (double)y, 0.0, 0.0) < sqrt(pow(radius,2) + pow(radius,2))){

               double localPfi =  fiImage->getImagePixel(point.x + x, point.y + y, border);
               int binNumber = (localPfi / localBinSize + 0.5);

               if(binNumber >= localBinCount)
                  binNumber = 0;
               else
                  if(binNumber < 0)
                     binNumber = localBinCount - 1;

               double localBinCenter = (double)binNumber * localBinSize + localBinSize / 2.0;

               int relatedBin;
               if(localPfi < localBinCenter)
                  relatedBin = binNumber - 1;
               else
                  relatedBin = binNumber + 1;

               double thisCenterDistance = abs(localBinCenter - localPfi);
               double relatedCenterDistance = localBinSize - thisCenterDistance;

               //отдать в метод
               if(relatedBin >= localBinCount)
                  relatedBin = 0;
               else
                  if(relatedBin < 0)
                     relatedBin = localBinCount -1;
               //

               localBin.at(binNumber) += lImage->getImagePixel(point.x + x, point.y + y, border) * (1 -
thisCenterDistance / localBinSize);
               localBin.at(relatedBin) += lImage->getImagePixel(point.x + x, point.y + y, border) * (1 -
relatedCenterDistance / localBinSize);
```

```
            }
        }
    }

    double firstMaxValue = -1;
    int firstMaxValueIndex = -1;
    double secondMaxValue = -1;
    int secondMaxValueIndex = -1;

    //Ищем первую и вторую максимальную
    for(int i = 0; i < localBinCount; i++){
        if(localBin.at(i) > firstMaxValue){
            secondMaxValue = firstMaxValue;
            secondMaxValueIndex = firstMaxValueIndex;

            firstMaxValue = localBin.at(i);
            firstMaxValueIndex = i;
        } else {
            if(localBin.at(i) > secondMaxValue){
                secondMaxValue = localBin.at(i);
                secondMaxValueIndex = i;
            }
        }
    }

    //добавили первую
    Point firstPoint = point;
    firstPoint.angle = firstMaxValueIndex * localBinSize;
    points1.push_back(firstPoint);

    //если вторая >= 0.8 от первой, то добваляем то же
    if(secondMaxValue >= (firstMaxValue * 0.8)){
        Point secondPoint = point;
        secondPoint.angle = secondMaxValueIndex * localBinSize;
        points1.push_back(secondPoint);
    }
  }
  return points1;
}
```

Main.cpp

```
void MainWindow::on_pushButton_Descriptor_2_clicked()
{

  Border border = Border::Reflection;
  int pointsCount = 100;

  Image *image1 = displayHelper->loadImg();
  Descriptors *descriptors1 = new Descriptors(image1, border);
  vector<Descriptor> vectorDescriptors1;
  InterestPoints *harrisPoints1 = new InterestPoints(image1);
  harrisPoints1->harris(0.3, border);
  harrisPoints1->reduceNumberOfPoints(pointsCount);
  //vector<Point> points1 = descriptors1->orientPoints(harrisPoints1->getPoints());
  vector<Point> points1 = descriptors1->meth1(harrisPoints1->getPoints());

  for(int i = 0; i < points1.size(); i++){
      vectorDescriptors1.push_back(descriptors1->getDescriptorForPoint(points1.at(i)));
  }

  Image *image2 = displayHelper->loadImg();
  Descriptors *descriptors2 = new Descriptors(image2, border);
  vector<Descriptor> vectorDescriptors2;
```

```
InterestPoints *harrisPoints2 = new InterestPoints(image2);
harrisPoints2->harris(0.3, border);
harrisPoints2->reduceNumberOfPoints(pointsCount);
vector<Point> points2 = descriptors2->meth1(harrisPoints2->getPoints());

for(int i = 0; i < points2.size(); i++){
   vectorDescriptors2.push_back(descriptors2->getDescriptorForPoint(points2.at(i)));
}

QImage qImage1 = image1->getImage().toImage();
QImage qImage2 = image2->getImage().toImage();
int height;
if(qImage1.height() > qImage2.height())
   height = qImage1.height();
else
   height = qImage2.height();

QImage qImage( qImage1.width() + qImage2.width(), height, qImage1.format());

QPainter painter;
painter.begin(&qImage);
QColor color = QColor(qRgb(0,255,0));

painter.drawImage(QPoint(0,0), qImage1);
painter.drawImage(QPoint(qImage1.width(),0), qImage2);

for(int i = 0; i < points1.size(); i++){
   painter.setPen(color);
   painter.drawEllipse(points1.at(i).x - 8, points1.at(i).y - 8, 16, 16);
   painter.drawLine(
            points1.at(i).x,
            points1.at(i).y,
            points1.at(i).x + (15 * cos(points1.at(i).angle * M_PI / 180.0)),
            points1.at(i).y + (15 * sin(points1.at(i).angle * M_PI / 180.0)) );
}

for(int i = 0; i < points2.size(); i++){
   painter.setPen(color);
   painter.drawEllipse(points2.at(i).x - 8 + qImage1.width(), points2.at(i).y - 8, 16, 16);
   painter.drawLine(
            points2.at(i).x + qImage1.width(),
            points2.at(i).y,
            points2.at(i).x + (15 * cos(points2.at(i).angle * M_PI / 180.0)) + qImage1.width(),
            points2.at(i).y + (15 * sin(points2.at(i).angle * M_PI / 180.0)) );
}

//Поиск
int findCount = 0;
for(int i = 0; i < vectorDescriptors1.size(); i++){
   double firstMinValue = 10000;
   int firstMinValueIndex = 10000;
   double secondMinValue = 10000;
   int secondMinValueIndex = 10000;

   for(int j = 0; j < vectorDescriptors2.size(); j++){
      double dist = Descriptors::getDescriptorsDistance(vectorDescriptors1.at(i), vectorDescriptors2.at(j));
      if(dist < firstMinValue){
         secondMinValue = firstMinValue;
         secondMinValueIndex = firstMinValueIndex;

         firstMinValue = dist;
         firstMinValueIndex = j;
      } else {
```

```
            if(dist < secondMinValue){
                secondMinValue = dist;
                secondMinValueIndex = j;
            }
        }
    }

    if(firstMinValue / secondMinValue < 0.8){
        findCount++;
        QPen pen(QColor(rand()%255 ,rand()%255, rand()%255));
        painter.setPen(pen);

        painter.drawEllipse(points1.at(i).x - 8, points1.at(i).y - 8, 16, 16);
        painter.drawLine(
                points1.at(i).x,
                points1.at(i).y,
                points1.at(i).x + (15 * cos(points1.at(i).angle * M_PI / 180.0)),
                points1.at(i).y + (15 * sin(points1.at(i).angle * M_PI / 180.0)) );

        painter.drawEllipse(points2.at(firstMinValueIndex).x - 8 + qImage1.width(), points2.at(firstMinValueIndex).y -
8, 16, 16);
        painter.drawLine(
                points2.at(firstMinValueIndex).x + qImage1.width(),
                points2.at(firstMinValueIndex).y,
                points2.at(firstMinValueIndex).x + (15 * cos(points2.at(firstMinValueIndex).angle * M_PI / 180.0)) +
qImage1.width(),
                points2.at(firstMinValueIndex).y + (15 * sin(points2.at(firstMinValueIndex).angle * M_PI / 180.0)) );

        painter.drawLine(
                QPoint(vectorDescriptors1.at(i).point.x,
                    vectorDescriptors1.at(i).point.y),
                QPoint(vectorDescriptors2.at(firstMinValueIndex).point.x + qImage1.width(),
                    vectorDescriptors2.at(firstMinValueIndex).point.y));
    }
  }

  painter.end();

  showImageAtSecondScreen(QPixmap::fromImage(qImage));
  displayHelper->saveQPixmap(QPixmap::fromImage(qImage), filePath + "Descriptor2");

}
```