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 №_4_

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 вычисление дескрипторов окрестностей заданных точек путем вычисления градиентов в каждой точки изображения и разбиения окрестности на сетку.

• Реализовать вычисление гистограмм градиентов в ячейках сетки и нормализацию полученных дескрипторов.

• Реализовать визуализацию результатов поиска ближайших дескрипторов в двух изображениях.

Решение:

Дескрипторы — это область вокруг точки интереса.

Свойства:

• Для схожих регионов дескрипторы должны быть похожи

• Для различающихся регионов дескрипторы должны быть непохожи

Использование дескрипторов:

• Искать ближайшие между разными изображениями (задача сшивания)

Представление дескрипторов:

• n-мерный вектор

• Автоматически получаем функции расстояния и аппарат для поиска ближайший в сложных случаях

Алгоритм:

• Применяем к изображению фильтр Собеля, автоматически получаем величину градиента L и направление градиента Pfi.

• Имеем L и Pfi в каждой точке из окрестности

• Представляем 2π в виде N (N = 8) корзин

• Для каждой точки:

  • Выбираем 2 смежные корзины исходя из Pfi.

  • Пропорционально распределяем L между ними

  • Прибавляем полученные значения в выбранные корзины

• Нормализуем полученный вектор из N корзин

  • (приводим его к единичной длине)

Сопоставление дескрипторов происходит через Эвклидово расстояние

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_n - p_n)^2}$$

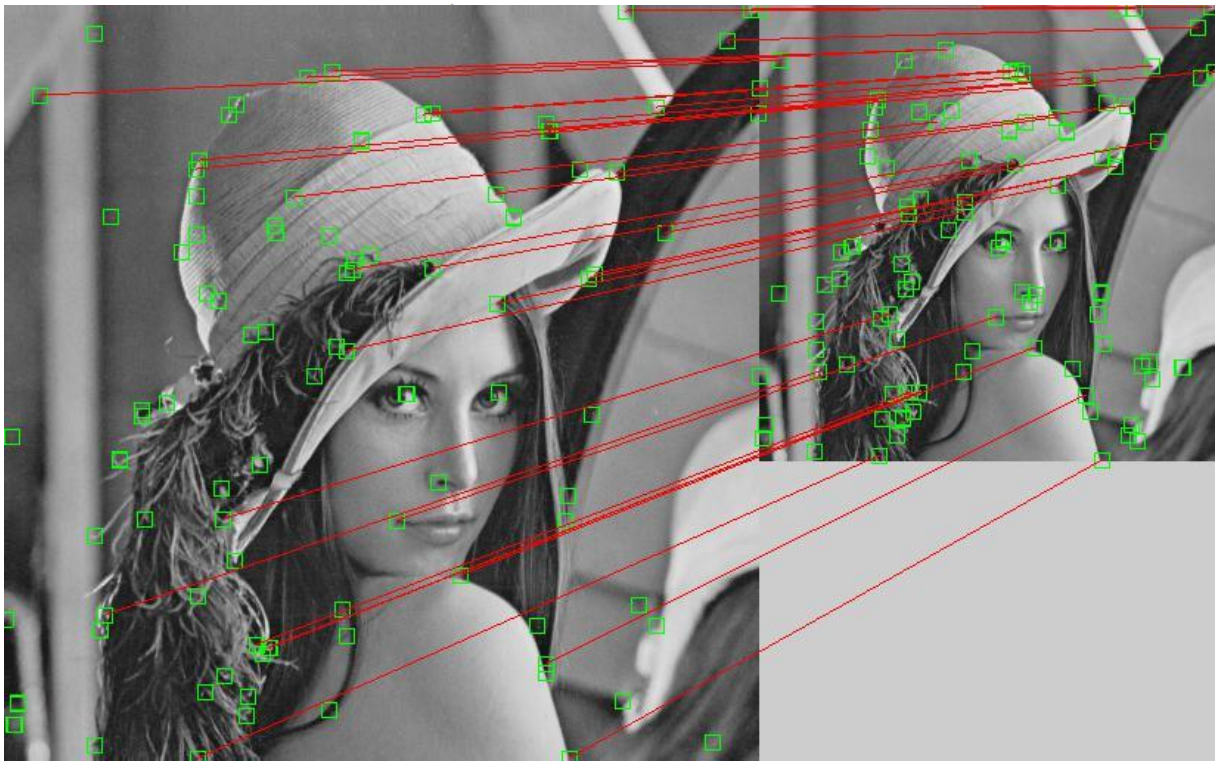

Рисунок 1 — Результат работы алгоритма

Исходный код программы:

Descriptors.h

```
struct Descriptor{
    Point point;
    double histograms[histogramsCount * binCount];
};

class Descriptors
{
private:
    Image *lImage;
    Image *fiImage;
    Border border;
    const int partsCount = 4;
    const double descriptorRadius = 8.0;
public:
    Descriptors(Image *image, Border inpBorder);
    Descriptor getDescriptorForPoint(Point point);
    static double getDescriptorsDistance(Descriptor descriptor1, Descriptor descriptor2);
    std::vector<Point> orientPoints(std::vector<Point> points);
vector<Point> meth1(vector<Point> points);
private:
    double getDistanceAroundPoints(double x0, double y0, double x1, double y1);
    double getHistValue(const double histograms[], int histogramNumber, int basketNumber);
    void Descriptors::addHistValue(double histograms[], int histogramNumber, int basketNumber, double value);
```

```
    void Descriptors::setHistValue(double histograms[], int histogramNumber, int basketNumber, double value);
    void descriptorRationing(double histograms[]);
};
```

Descriptors.cpp

```
Descriptors::Descriptors(Image *image, Border inpBorder)
{
    int height = image->getHeight();
    int width = image->getWidth();
    border = inpBorder;
    ImageModification *imageModifSobX = new ImageModification(Modification::SobelX);
    Image *sobelXImg = image->modificateImage(imageModifSobX, border);

    ImageModification *imageModifSobY = new ImageModification(Modification::SobelY);
    Image *sobelYImg = image->modificateImage(imageModifSobY, border);

    lImage = Image::getSobelXY(sobelXImg, sobelYImg);
    //lImage->imageMatrixRationing();  нужно ли нормирование
    fiImage = new Image(width, height);
    for( int y = 0; y < height; y++)
        for(int x = 0; x < width; x++)
            fiImage->setPixel(x, y, atan2(sobelYImg->getImagePixel(x, y, border),
                                          sobelXImg->getImagePixel(x, y, border)) * 180.0 / M_PI + 180.0);
}

Descriptor Descriptors::getDescriptorForPoint(Point point){
    Descriptor descriptor;
    descriptor.point = point;
    for(int i = 0; i < histogramsCount * binCount; i++)
        descriptor.histograms[i] = 0;

    double binSize = 360.0 / binCount;
    double radius = descriptorRadius;

    for(int i = -radius; i < radius; i++)
        for(int j = -radius; j < radius; j++){

            double cosAngle = cos((360.0 - point.angle) * M_PI / 180.0);
            double sinAngle = sin((360.0 - point.angle) * M_PI / 180.0);

            //Поворачиваем
            int angledY = i * cosAngle - j * sinAngle + 0.5;
            int angledX = i * sinAngle + j * cosAngle + 0.5;

            //В пределах дескриптора?
            if(getDistanceAroundPoints((double)angledX, (double)angledY, 0.0, 0.0) <
                    sqrt(pow(radius, 2) + pow(radius, 2))){
                if(angledX < -radius)
                    angledX = 0;
                else {
                    if(angledX >= radius)
                        angledX = radius -1;
                    else
                        angledX += radius;
                }
            //angledX += radius;

                if(angledY < -radius)
                    angledY = 0;
                else {
                    if(angledY >= radius)
```

```
                angledY = radius -1;
            else
                angledY += radius;
        }
    //angledY += radius;

        //Направление Фи
        double localFi = fiImage->getImagePixel(point.x + j, point.y + i, border) - point.angle;

        //За границей?
        if(localFi < 0)
            localFi =+ 360;
        if(localFi > 360)
            localFi =- 360;

        int x = angledX / ((radius * 2) / partsCount) - 0.5;
        int y = angledY / ((radius * 2) / partsCount) - 0.5;

        //Номер гистограммы
        int histogramNumber = x * partsCount + y;
        //Номер корзины
        int binNumber = (localFi / binSize + 0.5);

        //Раскидываем по корзинам
        double localBinCenter = (double)binNumber * binSize + binSize / 2.0;

        int binRelatedNumber;
        if(localFi < localBinCenter)
            binRelatedNumber = binNumber - 1;
        else
            binRelatedNumber = binNumber + 1;

        double thisCenterDistance = abs(localBinCenter - localFi);
        double relatedCenterDistance = binSize - thisCenterDistance;

        addHistValue(descriptor.histograms, histogramNumber, binNumber,
            lImage->getImagePixel(point.x + j, point.y + i, border) * (1 - thisCenterDistance / binSize));
        addHistValue(descriptor.histograms, histogramNumber, binRelatedNumber,
            lImage->getImagePixel(point.x + j, point.y + i, border) * (1 - relatedCenterDistance / binSize));
    }
  }
  descriptorRationing(descriptor.histograms);
  return descriptor;
}

double Descriptors::getDistanceAroundPoints(double x0, double y0, double x1, double y1){
  return sqrt(pow((x1 - x0),2) + pow((y1 - y0),2));
}

void Descriptors::addHistValue(double histograms[], int histogramNumber, int basketNumber, double value) {
  if(basketNumber >= binCount)
    basketNumber = 0;
  if(basketNumber < 0)
    basketNumber = binCount - 1;
  histograms[basketNumber * histogramsCount + histogramNumber] += value;
}
```

Main.cpp

```
void MainWindow::on_pushButton_Descriptor_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);

for(int i = 0; i < harrisPoints1->getPoints().size(); i++){
   vectorDescriptors1.push_back(descriptors1->getDescriptorForPoint(harrisPoints1->getPoints().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);

for(int i = 0; i < harrisPoints2->getPoints().size(); i++){
   vectorDescriptors2.push_back(descriptors2->getDescriptorForPoint(harrisPoints2->getPoints().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);
painter.drawImage(QPoint(0,0), qImage1);
painter.drawImage(QPoint(qImage1.width(),0), qImage2);
QColor color = QColor(qRgb(0,255,0));

for(int i = 0; i < harrisPoints1->getPoints().size(); i++){
   Point point = harrisPoints1->getPoints().at(i);
   painter.setPen(color);
   painter.drawRect(point.x - 5, point.y - 5, 10, 10);
}

for(int i = 0; i < harrisPoints2->getPoints().size(); i++){
   Point point = harrisPoints2->getPoints().at(i);
   painter.setPen(color);
   painter.drawRect(point.x - 5 + qImage1.width(), point.y - 5, 10, 10);
}

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){
            QPen pen(QColor(255, 0, 0));
            painter.setPen(pen);
            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 + "Descriptor");
}
```