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 №_3_

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 операторы Моравека и Харриса для поиска интересных точек в изображении.

• Реализовать фильтрацию интересных точек методом Adaptive Non-Maximum Suppression для заданного количества необходимых точек.

• Оценить повторяемость результата при некоторых искажениях оригинального изображения – сдвиг, поворот, шум, контрастность и яркость.

Точки интереса.

Точки интереса (интересные точки) должны обладать следующими свойствами:

• Инварианты к изменению освещения:

• Яркость/контрастность

• Экспозиция

• Инварианты к изменению ракурса

• На разных изображений одного объекта получим эту же точку

• Возможность ее идентифицировать и сопоставить

• Инварианты к искажениям

• Сферическая аберрация

• Шумы

• Сжатие

Детекторы интересных точек.

Оператор Моравека.

• Края не обладают уникальностью, их нужно избегать

• Уровень контраста точки – минимум суммы квадратов разностей окна вокруг точки и сдвинутого окна вокруг точки

• Если точка стоит на границе – уровень контраста будет минимальным: какой-нибудь сдвиг окна попадет на направление границы

• Угловой точкой может быть та, контрастность который максимальна в окрестности + пороговое значение контрастности

Алгоритм.

$$C(x,y,d) = \sum_u \sum_v [I(x+u, y+v) - I\left(x+u+d_x, y+v+d_y\right)]^2$$
$$S(x,y) = \min_d C(x,y,d)$$
$$\forall p: S(x,y) > S\left(x+p_x, y+p_y\right)$$
$$S(x,y) > T$$

, где

I - изображение

(x,y) – рассматриваемая точка

u,v – координаты в окне

d – вектор сдвига окна

C – значение контрастности в точке при заданном векторе сдвига

S – значение оператора

p – окрестность в которой определяется локальный максимум

T – пороговое значение

Рисунок 1 — Результат работы алгоритма

Моравек - недостатки

• Не обладает изотропностью

• Изотропен только при краях под 45х градусов

• Высокая вычислительная нагрузка

Оператор Харриса.

Алгоритм.

- Обозначим ошибку сдвига как $E(u,v)$:
- $E(u,v) = \sum_{(x,y)\in W}[I(x+u,y+v) - I(x,y)]^2$

Представим $I(x,y)$ в виде ряда Тейлора:

$$I(x+u,y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + .. (\text{высшие порядки})$$

Если (u,v) малые – достаточно первых производных:

$$I(x+u,y+v) \approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

$$\approx I(x,y) + [I_x \quad I_y]\begin{bmatrix}u\end{bmatrix}$$

Обозначим ошибку сдвига как $E(u,v)$:

$$E(u,v) = \sum_{(x,y)\in W}[I(x+u,y+v) - I(x,y)]^2$$

$$\approx \sum_{(x,y)\in W}[I(x,y) + I_x u + I_y v - I(x,y)]^2$$

$$\approx \sum_{(x,y)\in W}[I_x u + I_y v]^2$$

Обозначим ошибку сдвига как $E(u,v)$:

$$E(u,v) \approx \sum_{(x,y)\in W}[I_x u + I_y v]^2$$

$$\approx \sum_{(x,y)\in W}[I_x^2 u^2 + 2I_x I_y uv + I_y^2 v^2]$$

$$\approx Au^2 + 2Buv + Cv^2$$

$$A = \sum_{(x,y)\in W} I_x^2 \quad B = \sum_{(x,y)\in W} I_x I_y \quad C = \sum_{(x,y)\in W} I_y^2$$

Поверхность $E(u,v)$ локально аппроксимируется квадратичной формой

$$E(u,v) \approx Au^2 + 2Buv + Cv^2$$

$$\approx [u \quad v]\underbrace{\begin{bmatrix}A & B\\ B & C\end{bmatrix}}_{\text{H}}\begin{bmatrix}u\\ v\end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$
$$B = \sum_{(x,y)\in W} I_x I_y$$
$$C = \sum_{(x,y)\in W} I_y^2$$

Оператор Харриса

$$f = \det(H) - k\,trace(H)^2$$

- $\det(H) = AB - C^2$
- $trace(H) = A + B$
- например: k=0,06, T=4

Рисунок 2 — Результат работы алгоритма

Adaptive non-maximum suppression — используется для отфильтровывания избыточных точек интереса.

Алгоритм.

Найдем такой минимальный r для которого:

• Существуют такие точки, что в окрестности радиусом r все остальные точки имеют меньший f

• Число этих точек равно нужному нам числу

• Устанавливаем r=0, отфильтрованных точек нет

• Увеличиваем r, если появляются точки не удовлетворяющие условию=> убираем их

• Для устойчивости лучше добавлять коэффициент, например 0.9

• Повторяем, пока число оставшихся точек больше нужному нам

Рис 3 — Харрис с ограничением в 100 точек

Исходный код программы.

InterestPoints.h

```
struct Point{
    int x;
    int y;
    double s;
    double angle = 0.0;
};

class InterestPoints
{
private:
    Image *image;
public:
    vector<Point> vectorPoints;
public:
    InterestPoints(Image *origImage);
    void moravek(double thresholdValue, Border border);
    void harris(double thresholdValue, Border border);
    void reduceNumberOfPoints(int countPoint);
    QPixmap markPointsAtImage();  //нанести точки интереса на изображение
    vector<Point> getPoints(){return vectorPoints;}
private:
    void searchPoints(Image *imageMoravek, double t, Border border);
    bool pointAtVicinity(Point point1, Point point2, int radius);
};
```

InterestPoints.cpp

```
InterestPoints::InterestPoints(Image* origImage)
{
    image = origImage;
}

void InterestPoints::moravek(double thresholdValue, Border border){
    int width = image->getWidth();
    int height = image->getHeight();
    int roundPixel = 2;
    Image *imageMoravek = new Image(width, height);
    vectorPoints.clear();

    for(int x = 0; x < width; x++)
        for(int y = 0; y < height; y++){
            vector<double> errors;
            for(int dx = - 1; dx <= 1; dx++)
                for(int dy = -1; dy <= 1; dy++){
                    if(dx !=0 && dy!=0){ //сдвиг по диагонали
                        double windError = 0;
                        for(int u = -roundPixel; u <= roundPixel; u++)
                            for(int v = -roundPixel; v <= roundPixel; v++){
                                double c = pow(image->getImagePixel(x + u, y + v, border) -
                                        image->getImagePixel(x + u + dx, y + v + dy, border), 2);
                                windError += c;
                            }
                        errors.push_back(windError);
                    }
                }
            double s = *min_element(errors.begin(), errors.end());
            imageMoravek->setPixel(x, y, s);
        }
    searchPoints(imageMoravek, thresholdValue, border);
}

void InterestPoints::harris(double thresholdValue, Border border){
    int width = image->getWidth();
    int height = image->getHeight();
    int roundPixel = 2;
    Image *imageHarris = new Image(width, height);
    vectorPoints.clear();

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

    for (int x = 0; x < width; x++)
        for (int y = 0; y < height; y++){
            double A = 0;
            double B = 0;
            double C = 0;
            for(int u = -roundPixel; u <= roundPixel; u++)
                for(int v = -roundPixel; v <= roundPixel; v++){
                    double iX = sobelXImg->getImagePixel(x + u, y + v, border);
                    double iY = sobelYImg->getImagePixel(x + u, y + v, border);
                    A += pow(iX, 2);
                    B += iX * iY;
                    C += pow(iY, 2);
                }
            double discr = sqrt((A - C) * (A - C) + 4 * pow(B, 2));
            double l1 = abs((A + C + discr)/2);
```

```
            double l2 = abs((A + C - discr)/2);
            imageHarris->setPixel(x, y, min(l1, l2));
        }

    searchPoints(imageHarris, thresholdValue, border);
}

void InterestPoints::searchPoints(Image* imageMoravek, double t, Border border){
    int width = image->getWidth();
    int height = image->getHeight();
    int roundPixel = 2;

    for(int x = 0; x < width; x++)
        for(int y = 0; y < height; y++){
            bool localMax;
            double sXY = imageMoravek->getImagePixel(x, y, border);
            for(int u = -roundPixel; u <= roundPixel; u++)
                for(int v = -roundPixel; v <= roundPixel; v++){
                    if(u !=0 && v!=0){ //чтобы рассматривать значения s вокруг точки
                        if(sXY > imageMoravek->getImagePixel(x + u, y + v, border) && sXY > t)
                            localMax = true;
                        else{
                            localMax = false;
                            goto breakIsLocalMax;
                        }
                    }
                }
            breakIsLocalMax: if(localMax){
                Point point;
                point.x = x;
                point.y = y;
                point.s = sXY;

                vectorPoints.push_back(point);
            }
        }
}

void InterestPoints::reduceNumberOfPoints(int countPoint){
    double coef = 0.9;
    int radiusCurrent = 0;
    int radiusMax = sqrt(pow(image->getHeight(),2) + pow(image->getWidth(), 2));

    while(vectorPoints.size() > countPoint && radiusCurrent < radiusMax){
        for(int i = 0; i < vectorPoints.size(); i++)
            for(int j = i + 1; j < vectorPoints.size(); j++){ // j = i+1 тк может возникнуть казус, что размерность вектора
меньше чем индекс первой точки
                if(i != j)
                if(pointAtVicinity(vectorPoints[i], vectorPoints[j], radiusCurrent) &&
                        vectorPoints[i].s > vectorPoints[j].s * coef){
                    if(vectorPoints.size() <= countPoint)
                        goto breakAll;
                    vectorPoints.erase(vectorPoints.begin() + j);
                    j--;
                }
            }
        breakAll: radiusCurrent++;
    }

}

bool InterestPoints::pointAtVicinity(Point point1, Point point2, int radius){ //радиус <= расстоянию мд точками
    int distance = sqrt(pow(point1.x - point2.x,2) + pow(point1.y - point2.y, 2));
```

```
    return distance <= radius;
}


QPixmap InterestPoints::markPointsAtImage(){
    QImage resultImage = image->getImage().toImage();
    QPainter painter(&resultImage);
    painter.setPen(qRgb(255,255,0));
    for(Point point : vectorPoints) {
        painter.drawEllipse(point.x, point.y, 2, 2);
    }
   return QPixmap::fromImage(resultImage);
}
```

Main.cpp

```
void MainWindow::on_pushButton_Moravek_clicked()
{
    InterestPoints *moravekPoints = new InterestPoints(image);
    moravekPoints->moravek(0.1, Border::Reflection);
    moravekPoints->reduceNumberOfPoints(500);
    QPixmap moravekQPixmap = moravekPoints->markPointsAtImage();
    showImageAtSecondScreen(moravekQPixmap);
    displayHelper->saveQPixmap(moravekQPixmap, filePath + "moravekImg");
}

void MainWindow::on_pushButton_Harris_clicked()
{
    InterestPoints *harrisPoints = new InterestPoints(image);
    harrisPoints->harris(0.3, Border::Reflection);
    harrisPoints->reduceNumberOfPoints(100);
    QPixmap harrisQPixmap = harrisPoints->markPointsAtImage();
    showImageAtSecondScreen(harrisQPixmap);
    displayHelper->saveQPixmap(harrisQPixmap, filePath + "harrisImg");
}
```