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)

«\_\_\_\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2015 г.

Отчет по лабораторной работе №8

Оценка трансформации через схему RANSAC

«Интеллектуальные технологии обработки изображений»

ИЗ 231000.3.000 О

Студент группы 8ПИ-41 А.Ю. Смирнова

(И.О. Фамилия)

Преподаватель старший преподаватель М.Г. Казаков

(должность, ученое звание) (И.О. Фамилия)

Барнаул 2015

**Цель работы**

Освоить схему RANSAC для поиска модели трансформации между изображениями.

**Формулировка задачи**

* Даны два изображения, одно является сдвигом, смещением, вращением, изменением ракурса, либо все вместе.
* С помощью извлеченных наборов дескрипторов из обоих изображений и их соответствий, найти модель трансформации между изображениями, представленной в виде матрицы выбранного преобразования.
* Продемонстрировать полученные результаты на примере сшивания исходных изображений исходя из полученной матрицы.

**Исходный код:**

models.h

#ifndef MODELS\_H

#define MODELS\_H

#include "image.h"

#include "descriptor.h"

#include <gsl/gsl\_linalg.h>

#include <gsl/gsl\_blas.h>

class Models

{

vector<pair<shared\_ptr<Descriptor>, shared\_ptr<Descriptor>>> matches;

public:

vector<pair<Point, Point>> best;

Models(vector<shared\_ptr<Descriptor>> desc1, vector<shared\_ptr<Descriptor>> desc2);

float \*RanSaC(int iterCount, float eps);

~Models();

};

#endif

models.cpp

#include "models.h"

Models::Models(vector<shared\_ptr<Descriptor> > desc1, vector<shared\_ptr<Descriptor> > desc2)

{

matches = vector<pair<shared\_ptr<Descriptor>, shared\_ptr<Descriptor>>>();

best = vector<pair<Point, Point>>(8);

int closestNum;

for(uint i=0; i<desc1.size(); i++)

{

closestNum = desc1[i]->findClosest(desc2);

if(closestNum != -1)

{

matches.push\_back(make\_pair(desc1[i], desc2[closestNum]));

}

}

qDebug() << matches.size();

}

float\* Models::RanSaC(int iterCount, float eps)

{

if(matches.size() < 4 || iterCount == 0) return NULL;

int iter = 0;

mt19937 mt\_rand(time(0));

uniform\_int\_distribution<int> rnd(0, matches.size() - 1);

int baseMatches[4];

bool anyOverlap;

int overlap;

gsl\_matrix \*A = gsl\_matrix\_alloc(8,9);

gsl\_matrix \*AT = gsl\_matrix\_alloc(9,8);

gsl\_matrix \*multA = gsl\_matrix\_alloc(9,9);

gsl\_matrix \*V = gsl\_matrix\_alloc(9, 9);

gsl\_vector \*singular = gsl\_vector\_alloc(9);

gsl\_matrix\_set\_zero(A);

float h[9];

int inliners;

float xInit, yInit, xExpect, yExpect, xCur, yCur;

float lastH;

int bestInliners = 0;

float bestModel[9];

while(iter < iterCount)

{

//choose four points

baseMatches[0] = rnd(mt\_rand);

baseMatches[1] = rnd(mt\_rand);

baseMatches[2] = rnd(mt\_rand);

baseMatches[3] = rnd(mt\_rand);

gsl\_matrix\_set\_zero(A);

//we want them to be different

while(1)

{

anyOverlap = false;

for(int i=0; i<4; i++)

for(int j=i+1; j<4; j++)

if(baseMatches[i] == baseMatches[j])

{

anyOverlap = true;

overlap = j;

goto exit;

}

exit: if(anyOverlap)

{

baseMatches[overlap] = rnd(mt\_rand);

}

else break;

}

//find model

//set matrix A

for(int i=0; i<4; i++)

{

xInit = matches[baseMatches[i]].first->point.x;

yInit = matches[baseMatches[i]].first->point.y;

xExpect = matches[baseMatches[i]].second->point.x;

yExpect = matches[baseMatches[i]].second->point.y;

gsl\_matrix\_set(A, i\*2, 0, xInit);

gsl\_matrix\_set(A, i\*2, 1, yInit);

gsl\_matrix\_set(A, i\*2, 2, 1);

gsl\_matrix\_set(A, i\*2, 6, -xExpect\*xInit);

gsl\_matrix\_set(A, i\*2, 7, -xExpect\*yInit);

gsl\_matrix\_set(A, i\*2, 8, -xExpect);

gsl\_matrix\_set(A, i\*2 + 1, 3, xInit);

gsl\_matrix\_set(A, i\*2 + 1, 4, yInit);

gsl\_matrix\_set(A, i\*2 + 1, 5, 1);

gsl\_matrix\_set(A, i\*2 + 1, 6, -yExpect\*xInit);

gsl\_matrix\_set(A, i\*2 + 1, 7, -yExpect\*yInit);

gsl\_matrix\_set(A, i\*2 + 1, 8, -yExpect);

}

gsl\_matrix\_transpose\_memcpy(AT, A);

gsl\_blas\_dgemm(CblasNoTrans, CblasNoTrans, 1, AT, A, 0, multA);

gsl\_linalg\_SV\_decomp\_jacobi(multA, V, singular);

//h[8] = 1;

lastH = gsl\_matrix\_get(V, 8, 8);

for(int i=0; i<9; i++)

{

h[i] = gsl\_matrix\_get(V, i, 8)/lastH;

}

//find inliners count

inliners = 0;

for(uint i=0; i<matches.size(); i++)

{

xInit = matches[i].first->point.x;

yInit = matches[i].first->point.y;

xExpect = matches[i].second->point.x;

yExpect = matches[i].second->point.y;

xCur = (h[0]\*xInit + h[1]\*yInit + h[2]) / (h[6]\*xInit + h[7]\*yInit + h[8]);

yCur = (h[3]\*xInit + h[4]\*yInit + h[5]) / (h[6]\*xInit + h[7]\*yInit + h[8]);

if(sqrt(pow(xExpect - xCur,2)+pow(yExpect - yCur,2)) < eps)

{

inliners++;

}

}

if(inliners > bestInliners)

{

for(int i=0; i<9; i++)

{

bestModel[i] = h[i];

}

for(int i=0; i<4; i++)

{

xInit = matches[baseMatches[i]].first->point.x;

yInit = matches[baseMatches[i]].first->point.y;

xCur = (h[0]\*xInit + h[1]\*yInit + h[2]) / (h[6]\*xInit + h[7]\*yInit + h[8]);

yCur = (h[3]\*xInit + h[4]\*yInit + h[5]) / (h[6]\*xInit + h[7]\*yInit + h[8]);

best[i] = make\_pair(Point(xInit, yInit, 0, 0), Point(xCur,yCur, 0, 0));

}

bestInliners = inliners;

}

iter++;

}

gsl\_matrix\_free(A);

gsl\_matrix\_free(AT);

gsl\_matrix\_free(multA);

gsl\_matrix\_free(V);

gsl\_vector\_free(singular);

qDebug() << "Model:";

for(int i=0; i<9; i++)

{

qDebug() << bestModel[i];

}

qDebug() <<"Inliers: " << bestInliners;

return bestModel;

}

Models::~Models()

{

}

mainwindow.cpp

QImage MainWindow::drawBestPoints(const Models &models, const Image &img1, const Image& img2)

{

QImage result = QImage(img1.getWidth() + img2.getWidth() + 1, max(img1.getHeight(), img2.getHeight()), QImage::Format\_RGB32);

result.fill(0);

QPainter painter(&result);

painter.setPen(QPen(QColor(Qt::red)));

painter.drawImage(0,0,img1.toQImage());

painter.drawImage(img1.getWidth()+1, 0, img2.toQImage());

for(uint i=0; i<models.best.size(); i++)

{

if(i%2 == 0) painter.setPen(QPen(QColor(Qt::red)));

else painter.setPen(QPen(QColor(Qt::green)));

painter.drawRect(models.best[i].first.x - 1, models.best[i].first.y - 1, 3, 3);

painter.drawEllipse(QPoint(models.best[i].first.x, models.best[i].first.y), models.best[i].first.scale\*8/1.6, models.best[i].first.scale\*8/1.6);

painter.drawRect(models.best[i].second.x + img1.getWidth(), models.best[i].second.y -1, 3, 3);

painter.drawEllipse(QPoint(models.best[i].second.x + img1.getWidth() + 1, models.best[i].second.y), models.best[i].second.scale\*8/1.6, models.best[i].second.scale\*8/1.6);

painter.drawLine(QPoint(models.best[i].first.x, models.best[i].first.y), QPoint(models.best[i].second.x + img1.getWidth() + 1, models.best[i].second.y));

}

painter.end();

return result;

}

QImage MainWindow::createPanorama(const Image &img1, const Image& img2, float \*h)

{

QImage result = QImage(1500,1500, QImage::Format\_RGB32);

result.fill(0);

QPainter painter(&result);

int delta = 50;

painter.drawImage(delta,delta,img1.toQImage());

int xInit, yInit, xCur, yCur;

for(int i=0; i<img2.getWidth(); i++)

{

for(int j=0; j<img2.getHeight(); j++)

{

xInit = i;

yInit = j;

xCur = round((h[0]\*xInit + h[1]\*yInit + h[2]) / (h[6]\*xInit + h[7]\*yInit + h[8])) + delta;

yCur = round((h[3]\*xInit + h[4]\*yInit + h[5]) / (h[6]\*xInit + h[7]\*yInit + h[8])) + delta;

if(xCur >= 0 && xCur <1000 && yCur >= 0 && yCur <1000)

{

result.setPixel(xCur, yCur, qRgb(img2.getPixel(j, i),img2.getPixel(j, i),img2.getPixel(j, i)));

//result.setPixel(xCur, yCur, img2.getPixel(j, i));

}

}

}

painter.end();

return result;

}

**Результаты работы программы:**

Исходные изображения:



Результат:



Исходные изображения:





Результат:

