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 №2

Пространство масштаба

«Компьютерное зрение»

ИЗ 231000.3.000 О

Студент группы 8ПИ-41 А.Ю. Смирнова

(И.О. Фамилия)

Преподаватель М.Г. Казаков

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

Барнаул 2015

Задание:

* Из заданного изображения построить гауссову пирамиду
  + Устанавливается количество октав
  + Устанавливается число уровней в октаве
* Реализовать функцию
  + Поиск ближайшего изображение
  + Преобразование координат
* Реализовать отображение результатов
  + на каждом масштабе в октаве
  + эффективная для каждого масштаба

Решение:

В ходе решения поставленной задачи были разработаны следующие классы:

* Pyramid – пирамида.
* PyramidLevel – уровень пирамида.

Исходный код:

pyramid.h

#ifndef PYRAMID\_H

#define PYRAMID\_H

#include "image.h"

#include "filtermanager.h"

#include "maskfactory.h"

#include "pyramidlevel.h"

#include <QDir>

class Pyramid

{

int octaveNum;

int levelsNum;

vector<unique\_ptr<PyramidLevel>> images;

public:

Pyramid(int octaves, int levels);

bool saveToFolder(const QString &folderName);

static shared\_ptr<Pyramid> build(const Image &image, int octaveNum, int levelNum, float sigma0 = 1, float sigmaInit = 0.5);

float findPixel(int i, int j, float sigma);

~Pyramid();

};

#endif // PYRAMID\_H

pyramid.cpp

#include "pyramid.h"

Pyramid::Pyramid(int octaves, int levels)

{

octaveNum = octaves;

levelsNum = levels;

}

bool Pyramid::saveToFolder(const QString &folderName)

{

if(!QDir(folderName).exists())

{

QDir().mkdir(folderName);

}

QString fileName;

bool result = true;

for(int i=0; i<images.size(); i++)

{

fileName = "Octave" + QString::number(images[i]->getOctave()) + "Level" + QString::number(images[i]->getLevel()) + "Sigma" + QString::number(images[i]->getSigma());

result = result && images[i]->getImage()->toFile(folderName + "/" + fileName + ".jpg");

}

return result;

}

shared\_ptr<Pyramid> Pyramid::build(const Image &image, int octaveNum, int levelNum, float sigma0, float sigmaInit)

{

if(octaveNum > 0 && levelNum > 0)

{

shared\_ptr<Pyramid> result = make\_shared<Pyramid>(octaveNum, levelNum);

result->images.push\_back(make\_unique<PyramidLevel>(Image::fromQImage(image.toQImage()), 0, -1, sigmaInit));

shared\_ptr<Image> next = FilterManager::SeparatedFilter(image, \*MaskFactory::GaussSeparated(sqrt(sigma0\*sigma0 - sigmaInit\*sigmaInit)), true, EdgeMode::MIRROR);

float k = pow(2, 1./levelNum);

float curSigma = sigma0;

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

{

for(int j=0; j<levelNum; j++)

{

result->images.push\_back(make\_unique<PyramidLevel>(next, i, j, curSigma));

curSigma = curSigma \* k;

next = FilterManager::SeparatedFilter(\*next, \*MaskFactory::GaussSeparated(k), false, EdgeMode::MIRROR);

}

result->images.push\_back(make\_unique<PyramidLevel>(next, i, levelNum, curSigma));

next = next->compress(2);

}

return result;

}

else

{

qFatal("Wrong level or octave number");

return nullptr;

}

}

float Pyramid::findPixel(int i, int j, float sigma)

{

if(sigma <= images[0]->getSigma())

{

return images[0]->getImage()->getPixel(i,j);

}

int scale;

if(sigma >= images.back()->getSigma())

{

scale = 2\*images.back()->getOctave();

if(scale == 0) scale = 1;

return images.back()->getImage()->getPixel(i/scale, j/scale);

}

int lowNum = 0;

for(int k=0; k<images.size() - 1; k++)

{

if(sigma <= images[k+1]->getSigma() && sigma >= images[k]->getSigma())

{

lowNum = k;

break;

}

}

int octave = images[lowNum]->getOctave();

scale = 2\*octave;

if(scale == 0) scale = 1;

float lowPixel = images[lowNum]->getImage()->getPixel(i/scale, j/scale);

float highPixel = images[lowNum+1]->getImage()->getPixel(i/scale, j/scale);

float lowSigma = images[lowNum]->getSigma();

float hightSigma = images[lowNum + 1]->getSigma();

qDebug() << "Low pixel = " << lowPixel;

qDebug() << "High pixel = " << highPixel;

qDebug() << "Low sigma = " << lowSigma;

qDebug() << "High sigma = " << hightSigma;

return lowPixel + (highPixel - lowPixel)\*(sigma - lowSigma)/(hightSigma - lowSigma);

}

Pyramid::~Pyramid()

{

}

pyramidlevel.h

#ifndef PYRAMIDLEVEL\_H

#define PYRAMIDLEVEL\_H

#include "image.h"

class PyramidLevel

{

shared\_ptr<Image> image;

int levelNum;

int octaveNum;

float sigma;

public:

PyramidLevel(shared\_ptr<Image> image, int octave, int level, float sigma);

shared\_ptr<Image> getImage();

int getLevel();

int getOctave();

float getSigma();

~PyramidLevel();

};

#endif // PYRAMIDLEVEL\_H

pyramidlevel.cpp

#include "pyramidlevel.h"

PyramidLevel::PyramidLevel(shared\_ptr<Image> image, int octave, int level, float sigma)

{

this->image = image;

this->levelNum = level;

this->octaveNum = octave;

this->sigma = sigma;

}

shared\_ptr<Image> PyramidLevel::getImage()

{

return image;

}

int PyramidLevel::getLevel()

{

return levelNum;

}

int PyramidLevel::getOctave()

{

return octaveNum;

}

float PyramidLevel::getSigma()

{

return sigma;

}

PyramidLevel::~PyramidLevel()

{

}

Примеры использования:



Рисунок 1. Исходное изображение.



Рисунок 2. Изображение второго уровня второй октавы пирамиды с σ0 = 0.8

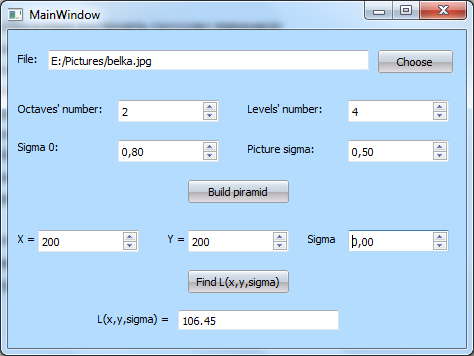


Рисунок 3. Вычисление L(x,y,σ).