|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | **ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**  **НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**  **«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**  Факультет компьютерных наук  Департамент программной инженерии | | | | | |
| СОГЛАСОВАНО  Преподаватель департамента  программной инженерии факультета компьютерных наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.А. Сибирцева  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2015 г. | | |  | | УТВЕРЖДАЮ  Академический руководитель  образовательной программы  «Программная инженерия»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2015 г. |
| ПРОГРАММА ПОИСКА ОБЪЕКТОВ НА ИЗОБРАЖЕНИИ С  ИСПОЛЬЗОВАНИЕМ ХАРАКТЕРИСТИЧЕСКИХ ТОЧЕК | | | | | |
| Подп. и дата | |  | | **ЛИСТ УТВЕРЖДЕНИЯ**  RU.17701729.503200--01 81 01-1-ЛУ | | | | | |
| Инв. № дубл. | |  | |  | | | | | |
| Взам. инв. № | |  | |  | |  | | Исполнитель: студент группы 201 ПИ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Ложков А.Г. /  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2015 г. | |
| Подп. и дата | |  | |  | | | | | |
| Инв. № подл. | |  | | **5** | | | | | |
|  | |  | | УТВЕРЖДЕНО  RU.17701729.503200--01 81 01-1-ЛУ  **ПРОГРАММА ПОИСКА ОБЪЕКТОВ НА ИЗОБРАЖЕНИИ С**  ИСПОЛЬЗОВАНИЕМ ХАРАКТЕРИСТИЧЕСКИХ ТОЧЕК  Пояснительная записка  RU.17701729.503200--01 81  Листов 30  5 | | | | | |
| Подп. и дата | |  | |
| Инв. № дубл. | |  | |
| Взам. инв. № | |  | |
| Подп. и дата | |  | |
| Инв. № подл. | | RU.17701729.503200--01 81 | |

СОДЕРЖАНИЕ

1. Введение 3

1.1 Наименование программы 3

1.3 Основания для разработки 3

2. Назначение и область применения 4

2.1 Назначение программы 4

2.2 Область применения программы 4

3. Требования к программе 4

3.1 Постановка задачи на разработку программы 5

3.2 Описание используемых математических методов 5

3.3 Описание алгоритма и функционирования программы 9

3.3.1 Описание алгоритма программы 9

3.4 Описание и обоснование выбора метода организации входных данных 15

3.5 Описание и обоснование выбора состава технических средств 15

3.6 Описание и обоснование выбора состава программных средств 15

4. Технико-экономические показатели 16

4.1 Предполагаемая потребность 16

4.2 Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами 16

5. Источники, использованные при разработке 17

6. Приложение 1. Описание и функциональное назначение классов 18

7. Приложение 2. Описание и функциональное назначение методов, полей и свойств 19

8. Приложение 3. Диаграммы классов 24

9. Приложение 4. Диаграмма вариантов использования 28

# ВВЕДЕНИЕ

## Наименование программы

Наименование программы - «Программа поиска объектов на изображении с использованием характеристических точек».

Краткое наименование программы – «SIFT Object Detector».

## Основания для разработки

Основанием для разработки является приказ Национального исследовательского университета «Высшая школа экономики» № 6.18.1-02/1912-11 от 19.12.2014.

# НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

## Назначение программы

Программа иллюстрирует работу алгоритма SIFT для поиска объекта на изображении.

Данная программа позволяет загружать изображение произвольной сцены (фотографии нескольких предметов, фона и т.п.) и локализовать на нем объекты, представленные их загружаемыми изображениями.

## Область применения программы

Программа может быть использована для изучения принципа работы алгоритма SIFT в контексте распознавания объектов на изображении в рамках изучения дисциплины “Введение в компьютерное зрение”.

# ТРЕБОВАНИЯ К ПРОГРАММЕ

## Постановка задачи на разработку программы

Цель разработки: предоставить конечному пользователю программу, способную при помощи сгенерированных алгоритмом SIFT характеристических точек найти положение объекта на сцене путем сопоставления его точек и точек сцены.

Исследование аналогичных свободно предоставляемых программ показало, что все программы, иллюстрирующие работу алгоритма SIFT для поиска объектов и сопоставления изображений, не имеют графического интерфейса. Многие программы не предоставляют контроль над параметрами алгоритма или их частью.

С принятием во внимание всех вышеперечисленных факторов была разработана «Программа поиска объектов на изображении с использованием характеристических точек».

## Описание используемых математических методов

Для нахождения характеристических точек используется алгоритм SIFT (Scale-Invariant Feature Transform). Дескрипторы точек, получаемые этим алгоритмом, инвариантны относительно смещения, масштабирования и поворота изображения, а так же в меньшей степени относительно изменений яркости и аффинных преобразований [1, 2].

* + 1. **Нахождение и уточнение точек-кандидатов**

Для обеспечения инвариантности относительно масштабирования строятся пирамида гауссианов и пирамида разностей гауссианов [3].

Под гауссианом понимается изображение, обработанное фильтром гаусса с определенным значением :

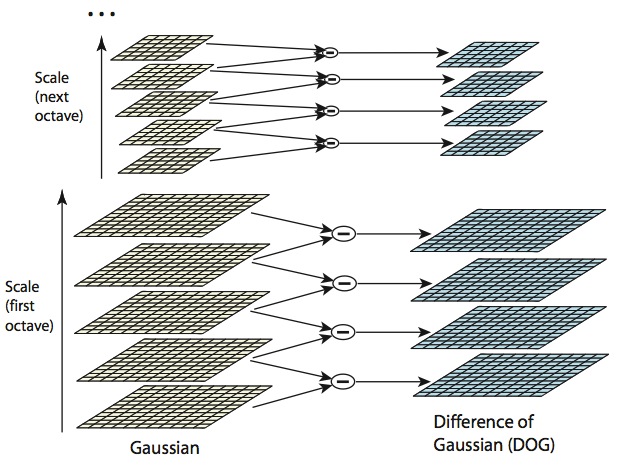


Здесь L – размытое изображение, G – ядро Гаусса ширины 5, I – исходное изображение.



Разность гауссианов получается вычитанием одного гауссиана пирамиды из соседнего в пределах одной октавы пирамиды с отличной :

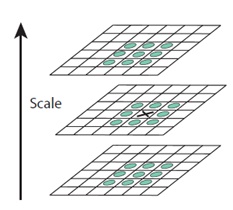




*Рис. 1. Представление пирамид гауссианов и разностей гауссианов*

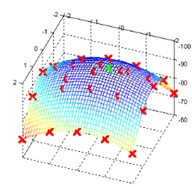
Пирамида гауссианов (масшатбируемое пространство) строится путем размытия исходного изображения фильтром гаусса с сигмой  и каждого следующего с сигмой . После получения N изображений таким способом, последнее изображение октавы уменьшается вдвое для получения первого изображения следующей октавы и процесс размытия продолжается.

Точки-кандидаты являются экстремумами пирамиды разностей (экстремум – точка, значение которой является максимумом или минимумом среди соседних 26 точек пирамиды).



*Рис. 2. Нахождение экстремума*

Следующим шагом является аппроксимация полученных экстремумов на сабпиксельном уровне при помощи многочлена Тейлора второго порядка в этих точках на пирамиде разностей гауссианов (D):



*Рис. 3. Сабпиксельный экстремум функции D*



Экстремумы многочлена находятся путем его дифференцирования и приравнивания к нулю.



Здесь  - градиент, - матрица Гессе, - вектор направления шага аппроксимации (значения сдвига по x, y, или z в пределах пирамиды).

* + 1. **Фильтрация точек**

Точки с малой контрастностью (со значением пикселя в пирамиде D меньше определенного порога) автоматически исключаются [4].

Также каждая точка проверяется на принадлежность границе объекта на изображении. В таких точках градиент в одном направлении (т.е. вдоль границы) имеет больший изгиб, чем перпендикулярный ему градиент.

Изгиб градиента определяется матрицей Гессе, ее следом и определителем:

****

****

Если взять за r коэффициент отношения большего градиента к меньшему (), то можно получить следующее отношение:



Таким образом, точка исключается, если , где thr – определенный порог отношения.

* + 1. **Определение ориентации**

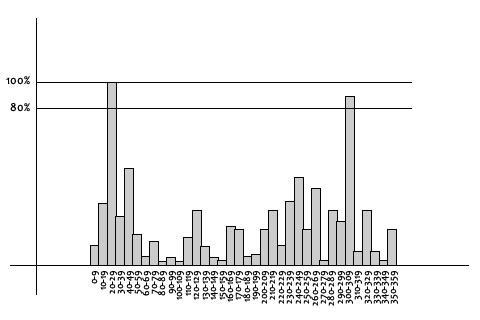
Определение ориентации характеристических точек на основе свойств изображения позволяет сделать ее дескриптор ориентированным и, как следствие, инвариантным к поворотам.

Величина и ориентация градиента в точке вычисляются по следующим формулам:



В окрестности точки (окне) размером  вычисляются значения градиентов, собираемые в гистограмму из 36 значений, представляющие значения градиентов в промежутках по 10 градусов (итого – 360 градусов). Каждый градиент увеличивает значение гистограммы на .

Максимальное значение гистограммы указывает на главную ориентацию. Все значения, большие чем 0.8\*max, также сохраняются и рассматриваются как возможные ориентации.



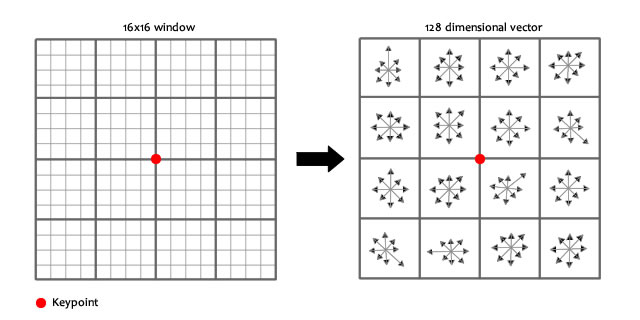
*Рис. 4. Представление гистограммы ориентаций*

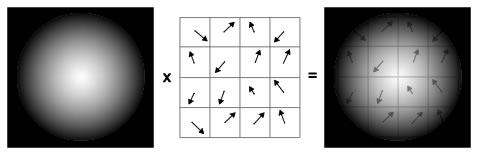
* + 1. **Построение дескрипторов точек**

Дескриптором точки является описание ее окрестности, устойчивое к искажениям изображения, изменению положения объекта и различному шуму.

Вычисляется окно 16х16 пикселей вокруг целевой точки, для каждого пикселя вычисляется значение градиента. Затем окно делится на 16 окон 4х4, для каждого из которых вычисляется гистограмма 8 направлений (0-44 градусов, 45-89 и так далее). Направления добавляются в диаграмму согласно весам, полученным из функции Гаусса, примененной на окнах 4х4.

Дескриптором будет являться гистограмма из 128 значений, собранная из 16 полученных гистограмм длины 8.





*Рис. 5. Представление дескриптора и вклада величин градиентов в его окна*

Итоговая гистограмма нормируется для обретения устойчивости к изменениям яркости изображения.

* + 1. **Сопоставление дескрипторов**

Для нахождения совпадений между двумя изображениями необходимо вычислить расстояния между всеми парами дескрипторов этих изображений. Чаще всего применяется квадрат расстояния Евклида:



* + 1. **Фильтрация совпадений алгоритмом RANSAC**

Алгоритм RANSAC стремится найти лучшую матрицу трансформации из системы координат точек объекта в систему координат сцены.[6] Далее точки, соответствующие модели этой матрицы, будут называться инлаерами.

Для этого на протяжении k итераций случайным образом делается выборка из четырех пар совпадающих точек, для которых строится модель преобразования.



Здесь  - вероятность того, что хотя бы одна точка в построенной модели не будет инлаером (модель будет неверной),  - вероятность того, что за k итераций не будет выбрано 4 инлаера (модель не будет построена).

При успешном построении матрицы трансформации по четырем точкам, все точки объекта переводятся этой матрицей в систему координат сцены.



После перевода координат точек, расчитывается кумулятивное расстояние между новыми координатами и координатами совпадающих точек сцены. Если это расстояние (ошибка) меньше ошибки на предыдущей итерации алгоритма и меньше определенного порога, определяющего максимальную ошибку для корректной модели, то модель принимается за эталонную.

После выполнения итераций алгоритма, получается список точек-инлаеров и матрицы транформации координат этих точек в точки сцены.

## Описание алгоритма и функционирования программы

### Описание алгоритма программы

Программа реализует классический алгоритм SIFT для нахождения характеристических точекс применением дополнительной оптимизации алгоритмом RANSAC. Ниже приведено его описание на языке псевдокода:

Sift(nScales, sigma, contrastThr, curvatureThr) {

// число октав:

nOctaves = log(min(imageWidth, imageHeight)) / log(2);

// пирамида гауссианов:

gaussPyramid = GaussianPyramid(image, nOctaves, nIntervals, sigma);

// пирамида разностей гауссианов:

dogPyramid = new DOGPyramid(gaussPyramid, nOctaves, nIntervals);

// массив характеристических точек

features = detectFeatures(contrastThr, curvThr);

// вычислить направления дескрипторов

getFeatureOrientations();

// вычислить дескрипторы точек

computeDescriptors(4, 8);

return features;

}

Алгоритм поиска точек-кандидатов на языке Java:

**private** List<Feature> detectFeatures(**float** contrastThr, **float** curvThr) {  
 List<Feature> features = **new** ArrayList<Feature>();  
 Feature feature;  
  
 **float** initialContrastThr = 0.5f \* contrastThr / nIntervals;  
  
 **for** (**int** octave = 0; octave < nOctaves; octave++) {  
 **for** (**int** interv = 1; interv < nIntervals + 1; interv++) {  
 **for** (**int** row = BORDER; row < dogPyramid.getSlice(octave, 0).getHeight() - BORDER; row++) {  
 **for** (**int** col = BORDER; col < dogPyramid.getSlice(octave, 0).getWidth() - BORDER; col++) {  
 **if** (Math.abs(dogPyramid.getSlice(octave, interv).getPix(row, col)) > initialContrastThr) {  
 **if** (isExtremum(octave, interv, row, col)) {  
 feature = approximateExtremum(octave, interv, row, col, contrastThr);  
 **if** (feature != **null**) {  
 **if** (!isOnEdge(dogPyramid.getSlice(feature.getOctave(), feature.getInterval()),  
 feature.getRow(), feature.getCol(), curvThr)) {  
 features.add(feature);  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
  
 **return** features;  
}

Нахождение экстремума из точки с сабпиксельной точностью (Java):

**private** Feature approximateExtremum(**int** octave, **int** interval, **int** row, **int** col, **double** contrastThr) {  
 **int** step = 0;  
 **double**[] shifts = **new double**[3];  
  
 **while** (step < MAX\_INTERPOLATION\_STEPS) {  
 shifts = approxStep(octave, interval, row, col);  
 *// if no need for shifting to another point, then successfully interpolated* **if** (Math.abs(shifts[0]) < 0.5 && Math.abs(shifts[1]) < 0.5 && Math.abs(shifts[2]) < 0.5) {  
 **break**;  
 }  
  
 col += Math.round(shifts[2]);  
 row += Math.round(shifts[1]);  
 interval += Math.round(shifts[0]);  
  
 **if** (interval < 1 || interval > nIntervals ||  
 col < BORDER || col > dogPyramid.getSlice(octave, 0).getWidth() - BORDER ||  
 row < BORDER || row > dogPyramid.getSlice(octave, 0).getHeight() - BORDER) {  
 **return null**;  
 }  
  
 step++;  
 }  
  
 **if** (step >= MAX\_INTERPOLATION\_STEPS) {  
 **return null**;  
 }  
  
 **double** newContr = approxContrast(octave, interval, row, col, shifts);  
 **if** (Math.abs(newContr) < contrastThr / nIntervals) {  
 **return null**;  
 }  
  
 Feature feature = **new** Feature(row, col, octave, interval);  
 feature.setExactX((col + shifts[2]) \* Math.pow(2.0, octave));  
 feature.setExactY((row + shifts[1]) \* Math.pow(2.0, octave));  
 feature.setRow(row);  
 feature.setCol(col);  
 feature.setOctave(octave);  
 feature.setInterval(interval);  
 feature.setSubinterval(shifts[0]);  
  
 **return** feature;  
}

**private double**[] approxStep(**int** octave, **int** interval, **int** row, **int** col) {  
 **double**[] shifts;  
  
 **double**[] dD = firstDeriv(octave, interval, row, col);  
 **double**[][] ddD = secondDeriv(octave, interval, row, col);  
 **double**[][] ddDinv = Matrix.invert3x3(ddD);  
  
 **double**[][] dDtransp = {{dD[0]},{dD[1]},{dD[2]}};  
 **double**[][] X = Matrix.multiply(ddDinv, dDtransp);  
  
 shifts = **new double**[]{X[2][0], X[1][0], X[0][0]};  
 **return** shifts;  
}

Определение, находится ли точка на границе объекта (Java):

**private boolean** isOnEdge(ImgArray img, **int** row, **int** col, **double** curvThr) {  
 **double** d, dxx, dyy, dxy;  
  
 d = img.getPix(row, col);  
 dxx = img.getPix(row, col + 1) + img.getPix(row, col - 1) - 2.0 \* d;  
 dyy = img.getPix(row+1, col) + img.getPix(row - 1, col) - 2.0 \* d;  
 dxy = (img.getPix(row + 1, col + 1) - img.getPix(row + 1, col - 1) -  
 img.getPix(row - 1, col + 1) + img.getPix(row - 1, col - 1)) / 4.0;  
  
 **double** tr = dxx + dyy;  
 **double** det = dxx \* dyy - dxy \* dxy;  
  
 **if** (det <= 0) {  
 **return true**;  
 }  
  
 **if** (tr \* tr / det < (curvThr + 1.0) \* (curvThr + 1.0) / curvThr){  
 **return false**;  
 }  
  
 **return true**;  
}

Определение направлений дескрипторов (Java):

**private void** getFeatureOrientations() {  
 **double**[] hist;  
 **double** omax;  
 List<Feature> newFeatures = **new** ArrayList<Feature>();  
  
 **for**(Feature feat : features) {  
 hist = orientationHist(gaussPyramid.getSlice(feat.getOctave(), feat.getInterval()),  
 feat.getRow(), feat.getCol(), N\_ORIENT\_HIST\_BINS,  
 (**int**) Math.round(ORIENT\_RADIUS \* feat.getScaleOctave()),  
 ORIENT\_SIGMA\_FACTOR \* feat.getScaleOctave());  
 **for**(**int** j = 0; j<ORIENT\_SMOOTHING\_PASSES; j++){  
 hist = smoothOrientHist(hist, N\_ORIENT\_HIST\_BINS);  
 }  
  
 omax = dominantOrient(hist, N\_ORIENT\_HIST\_BINS);  
 newFeatures.addAll(addPeakOrientFeatures(feat, hist, N\_ORIENT\_HIST\_BINS,  
 omax \* ADD\_ORIENT\_PEAK\_RATIO));  
  
 }  
 features.addAll(newFeatures);  
}

Вычисление дескрипторов точек (Java):

**private void** computeDescriptors(**int** d, **int** n) {  
 **double**[][][] hist;  
  
 **for**(Feature feat : features){  
 hist = descrHist(gaussPyramid.getSlice(feat.getOctave(), feat.getInterval()), feat.getRow(),  
 feat.getCol(), feat.getOrientation(), feat.getScaleOctave(), d, n);  
 histToDescr(hist, d, n, feat);  
 }  
  
}  
  
**private double**[][][] descrHist(ImgArray img, **int** row, **int** col, **double** ori, **double** scale, **int** d, **int** n) {  
 **double**[][][] hist = **new double**[d][d][n];  
  
 **double** PI2 = Math.PI \* 2.0;  
 **double** cos\_t = Math.cos(ori);  
 **double** sin\_t = Math.sin(ori);  
  
 **double** binsPerRad = n / PI2;  
 **double** exponDenom = d\*d\*0.5;  
 **double** histWidth = DESCR\_SCALE\_FACTOR \* scale;  
 **int** radius = (**int**)(histWidth \* Math.sqrt(2) \* (d+1.0) \* 0.5 + 0.5);  
  
 **double** row\_rot, col\_rot;  
 **double** rowBinCoord, colBinCoord, oriBinCoord;  
 **double** w;  
  
 **for**(**int** i = -radius; i<=radius; i++){  
 **for**(**int** j = -radius; j<=radius; j++){  
 col\_rot = ( j \* cos\_t - i \* sin\_t ) / histWidth;  
 row\_rot = ( j \* sin\_t + i \* cos\_t ) / histWidth;  
 rowBinCoord = row\_rot + d / 2 - 0.5;  
 colBinCoord = col\_rot + d / 2 - 0.5;  
  
 **if**(rowBinCoord>-1.0 && rowBinCoord < d && colBinCoord>-1.0 && colBinCoord < d){  
 Gradient grad = getGradientMagnOrient(img, row + i, col + j);  
 **double** grad\_ori = grad.getOrientation();  
 **if**(grad.isCorrect()){  
 grad\_ori -= ori;  
 **while**( grad\_ori < 0.0 )  
 grad\_ori += PI2;  
 **while**( grad\_ori >= PI2 )  
 grad\_ori -= PI2;  
  
 oriBinCoord = grad\_ori \* binsPerRad;  
 w = Math.exp(-(col\_rot \* col\_rot + row\_rot \* row\_rot) / exponDenom);  
 hist = interpHistEntry(hist, rowBinCoord, colBinCoord, oriBinCoord, grad.getMagnitude() \* w, d, n );  
 }  
 }  
 }  
 }  
  
 **return** hist;  
}

Алгоритм фильтрации совпадающих точек RANSAC на языке псевдокода:

Ransac(nPicks, nIter, distThr, minPoints, points1, points2) {

// nPicks – число случайно выбираемых точек

// nIter – число итераций

// distThr – порог расстояния между совпадающими точками

// minPoints – минимальное число точек для успешной трансформации

bestModel = null;

lowestError = infinity;

for(i = 0 to nIter) {

scenePoints = pickRandom(points1);

objectPoints = pickRandom(points2);

proposedModel = bestTransform(objectPoints -> scenePoints);

consensus = 0;

cumulativeError = 0;

for(j = 0 to nPoints) {

error = sqDistance(points2\*proposedModel, points1);

if (error < distThr) {

consensus++;

cumulativeError += error;

}

}

if(consensus > minPoints && cumulativeError < lowestError) {

bestModel = proposedModel;

lowestError = cumulativeError;

}

}

}

## Описание метода организации входных данных

В качестве форматов входных данных (изображений) были выбраны растровые графические форматы BMP, JPEG и PNG.

Преимущество данных форматов состоит в их широкой поддержке различными графическими редакторами и камерами, что позволяет

## Описание и обоснование выбора состава технических средств

Рекомендуется персональный компьютер, оснащенный следующими компонентами:

1) Процессор Intel Pentium - 4 / AMD Athlon 3200+ или совместимый с частотой 1.4 ГГц и более;

2) 1 ГБ RAM или более (требование операционной системы);

3) 10 МБ свободного места на устройстве хранения информации, требуется для хранения файлов программы и демонстрационных файлов;

4) Монитор и видеоадаптер с разрешением 800x600 или более высоким для реализации всех возможностей интерфейса программы;

5) Клавиатура и мышь или совместимые указывающие устройства.

## Описание и обоснование выбора состава программных средств

Программа предназначена для работы под управлением операционной системы Win-dows версии XP SP3, Vista, 7 или 8, которые поддерживает необходимый для работы програм-мы Java Runtime Environment 8, требующий Windows Installer 3.1 или более поздние версию.

В качестве средства разработки была выбрана среда IntelliJ Idea 13 и язык програм-мирования Java ввиду их пригодности для выполнения поставленной задачи.

# ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

## Предполагаемая потребность

Задача наглядной иллюстрации алгоритма нахождения характеристических точек и распознавания объектов по ним востребована в сфере образования. Программу могут использовать преподаватели и ученики, нуждающиеся демонстрации процесса решения задачи сопоставления изображений. Эта задача требует решения в таких областях как распознавание объектов, склейка изображений, 3D реконструкция, распознавание жестов и т. п.

## Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами

В процессе поиска в сети Интерент не было обнаружено отечественных или зарубежных приложений с графическим интерфейсом для наглядного отображения результатов поиска объекта на изображении с использованием характеристических точек алгоритма SIFT.

Преимущества данной разработки:

1. использование графического интерфейса с возможностью удобной загрузки изображений;
2. возможность менять параметры алгоритма нахождения характеристических точек;
3. возможность отображать и скрывать характеристические точки, связи между совпадающими точками и контуры найденного объекта;
4. использование классического алгоритма SIFT, что улучшает понимание принципа его работы;
5. приложение распростаняется бесплатно

# ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ

1. *Lowe, D. G.* Object Recognition from Local Scale-Invariant Features // The Proceedings of the Seventh IEEE International Conference on Computer Vision (Volume 2). IEEE, 1999. - С. 1150 - 1157.
2. *Lowe, D.G.* Distinctive Image Features from Scale-Invariant Keypoints // International Journal of Computer Vision. - 2004. - №60. - С. 91-110.
3. SIFT: Introduction [Электронный ресурс] // AI Shack URL: http://www.aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/ (дата обращения: 08.05.2015).
4. Scale Invariant Feature Transform [Электронный ресурс] // Scholarpedia URL: http://www.scholarpedia.org/article/Scale\_Invariant\_Feature\_Transform (дата обращения: 08.05.2015).
5. *Szeliski, R.* Computer vision: algorithms and applications. Springer Science & Business Media, 2010. - 957 с.
6. *Fischler, M. A., Bolles, R.C.* Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography // Communications of the ACM. - 1981. - №24. - С. 381-395.

**ПРИЛОЖЕНИЕ 1**

**ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ**

|  |  |
| --- | --- |
| **Класс** | **Назначение** |
| *MainForm.java* | Представляет главное окно приложения. Содержит обработчики событий и взаимодействует с классами Sift и RansacTransform. |
| *Sift.java* | Содержит методы, необходимые для получения дескрипторов характеристических точек изображения алгоритмом SIFT |
| *RansacTransform.java* | Содержит методы, необходимые для фильтрации совпадений характеристических точек алгоритмом RANSAC |
| *Filter.java* | Содержит методы для применения фильтра Гаусса на изображении |
| *Scaling.java* | Содержит методы для билинейного масштабирования изображения |
| *ImgArray* | Содержит массив значений [0; 1], представляющий информацию об обрабатываемом изображении |
| *GaussianPyramid.java* | Структура для построения и хранения пирамиды гауссианов |
| *DOGPyramid.java* | Структура для построения и хранения пирамиды разностей гауссианов |
| *Feature.java* | Объект характеристической точки, содержит всю необходимую информацию для построения ее дескриптора |
| *Gradient.java* | Объект градиента, содержащий информацию о его величине и ориентации |
| *ImageObject.java* | Объект обрабатываемого изображения, содержащий список найденных характеристических точек, а также связей с точками парного изображения |
| *Matrix.java* | Вспомогательный класс матричных методов |
| *Point.java* | Упрощенный класс точки в двумерном пространстве |
| *RansacTransform.java* | Содержит методы для фильтрации связей характеристических точек алгоритмом RANSAC и получения матрицы перехода для координат объекта |

**ПРИЛОЖЕНИЕ 2**

**ОПИСАНИЕ И ФУНКЦОНАЛЬНОЕ НАЗНАЧЕНИЕ МЕТОДОВ, ПОЛЕЙ И СВОЙСТВ**

*Sift.java*

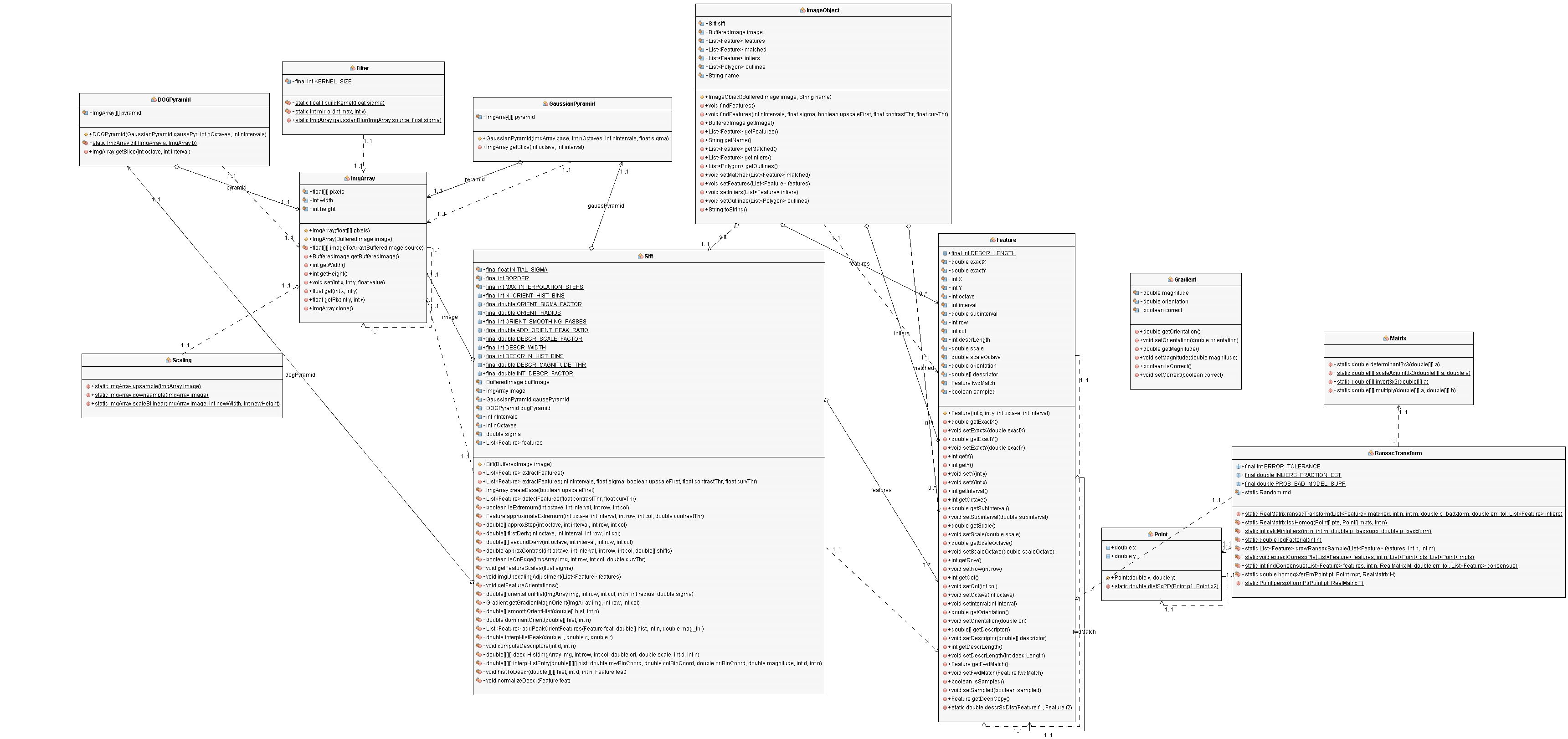
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поля** | | | | | |
| *Имя* | *Мод. Доступа* | *Тип* | | *Назначение* | |
| INITIAL\_SIGMA | private static final | float | | Предполагаемая степень размытия по Гауссу для начального изображения | |
| BORDER | private static final | int | | Ширина границ изображения для ограничения поиска точек | |
| MAX\_INTERPOLATION\_STEPS | private static final | int | | Максимальное число шагов интерполяции экстремума точки | |
| N\_ORIENT\_HIST\_BINS | private static final | int | | Число сегментов гистограммы ориентации точки | |
| ORIENT\_SIGMA\_FACTOR | private static final | double | | Сигма для определения ориентации точки | |
| ORIENT\_RADIUS | private static final | double | | Радиус окна для определения ориентации | |
| ORIENT\_SMOOTHING\_PASSES | private static final | int | | Число проходов сглаживания гистограммы ориентации | |
| ADD\_ORIENT\_PEAK\_RATIO | private static final | double | | Порог принятия дополнительных ориентаций | |
| DESCR\_SCALE\_FACTOR | private static final | double | | Определяет размер одной гистограммы дескриптора | |
| DESCR\_WIDTH | private static final | int | | Ширина дескриптора | |
| DESCR\_N\_HIST\_BINS | private static final | int | | Число сегментов в гистограммах дескриптора | |
| DESCR\_MAGNITUDE\_THR | private static final | double | | Порог величины вектора дескриптора | |
| INT\_DESCR\_FACTOR | private static final | double | | Параметр, используемый для конвертации значений дескрипторы в тип Integer | |
| buffImage | private | BufferedImage | | Исходное изображение | |
| image | private | ImgArray | | Массив черно-белых значений исходного изображения | |
| gaussPyramid | private | GaussianPyramid | | Пирамида гауссианов | |
| dogPyramid | private | DOGPyramid | | Пирамида разностей гауссианов | |
| nIntervals | private | int | | Число гауссианов в одной октаве пирамиды | |
| nOctaves | private | int | | Число октав пирамиды гауссианов | |
| sigma | private | double | | Начальное значение сигмы | |
| features | private | List<Feature> | | Список характеристических точек | |
| **Методы** | | | | | |
| *Имя* | *Мод. Доступа* | *Тип* | *Аргументы* | | *Назначение* |
| Sift | public | констр. |  | | Конструктор объекта алгоритма SIFT |
| extractFeatures | private | List<Feature> |  | | Главный метод алгоритма, возвращает список найденных им характеристических точек с параметрами по умолчанию |
| extractFeatures | private | List<Feature> | int nIntervals, float sigma, boolean upscaleFirst, float contrastThr, float curvThr | | Главный метод алгоритма, возвращает список найденных им характеристических точек |
| createBase | private | ImgArray | boolean upscaleFirst | | Производит начальную обработку изображения фильтром Гаусса и при необходимости увеличивает его вдвое |
| detectFeatures | private | List<Feature> | float contrastThr, float curvThr | | Находит на изображении точные позиции характеристических точек и фильтрует их |
| isExtremum | private | boolean | int octave, int interval, int row, int col | | Определяет, является ли точка экстремумом пирамиды разностей гауссианов |
| approximateExtremum | private | Feature | int octave, int interval, int row, int col, double contrastThr | | Получает уточненную позицию точки с сабпиксельной точностью |
| approxStep | private | double[] | int octave, int interval, int row, int col | | Производит один шаг нахождения экстремума точки и возвращает ее смещение |
| firstDeriv | private | double[] | int octave, int interval, int row, int col | | Вычисляет первую производную пирамиды разностей гауссианов в точке |
| secondDeriv | private | double[][] | int octave, int interval, int row, int col | | Вычисляет вторую производную пирамиды разностей гауссианов в точке |
| approxContrast | private | double | int octave, int interval, int row, int col, double[] shifts | | Получает интерполированное значение цвета изображения () |
| isOnEdge | private | boolean | ImgArray img, int row, int col, double curvThr | | Определяет, расположена ли точка на границе объекта |
| getFeatureScales | private | void | float sigma | | Определяет характеристический масштаб для каждой найденной точки |
| imgUpscalingAdjustment | private | void | List<Feature> features | | Уточняет координаты точек если было произведено увеличение изображения на первом шаге алгоритма |
| getFeatureOrientations | private | void |  | | Определяет доминантные ориентации точек |
| orientationHist | private | double[] | ImgArray img, int row, int col, int n, int radius, double sigma | | Получает гистограмму ориентаций точки |
| dominantOrient | private | double | double[] hist, int n | | Получает значение доминирующей ориентации из гистограммы |
| addPeakOrientFeatures | private | List<Feature> | Feature feat, double[] hist, int n, double mag\_thr | | Добавляет точки с ориентациями выше пороговой доли от доминирующей ориентации |
| computeDescriptors | private | void | int d, int n | | Строит дескрипторы для всех найденных точек |
| descrHist | private | double[][][] | ImgArray img, int row, int col, double ori, double scale, int d, int n | | Строит массив гистограмм дескриптора |
| interpHistEntry | private | double[][][] | double[][][] hist, double rowBinCoord, double colBinCoord, double oriBinCoord, double magnitude, int d, int n | | Интерполирует положение значения гистограммы в дескрипторе |
| histToDescr | private | void | double[][][] hist, int d, int n, Feature feat | | Переводит двумерный массив дескриптора в вектор |
| normalizeDescr | private | void | Feature feat | | Нормализует дескриптор |

*AntColonyLib.Node*

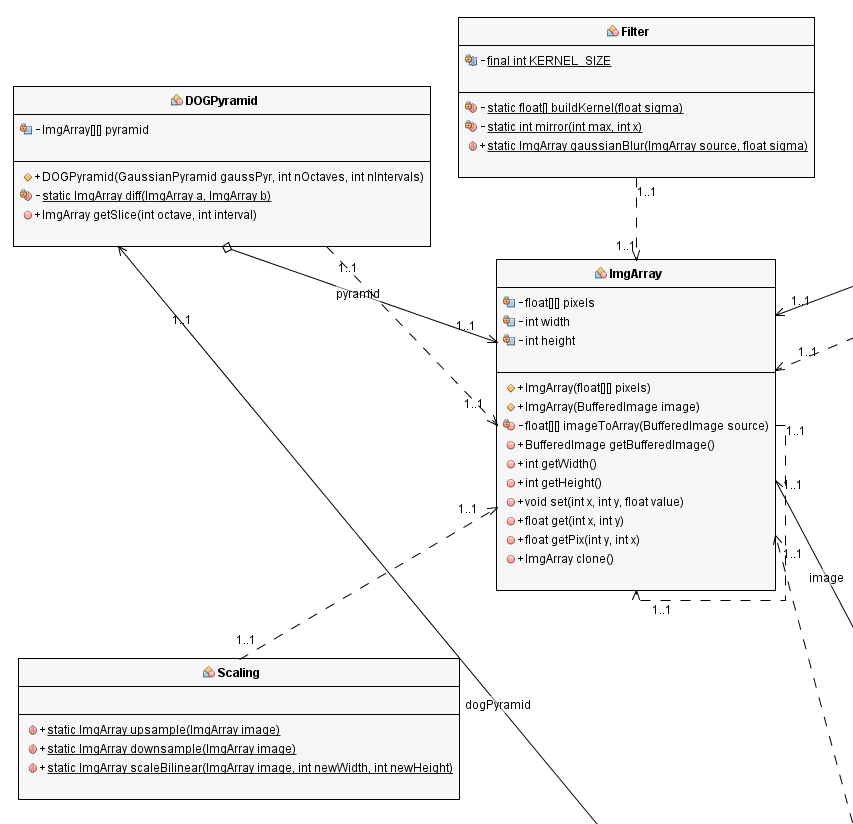
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поля** | | | | | |
| *Имя* | *Мод. Доступа* | *Тип* | | *Назначение* | |
| ERROR\_TOLERANCE | private static final | int | | Допустимый уровень ошибки в пикселях | |
| INLIERS\_FRACTION\_EST | private static final | double | | Оценка доли | |
| PROB\_BAD\_MODEL\_SUPP | private static final | double | | Вероятность получения неверной матрицы трансформации | |
| **Методы** | | | | | |
| *Имя* | *Мод. Доступа* | *Тип* | *Аргументы* | | *Назначение* |
| ransacTransform | public static | RealMatrix | List<Feature> matched, int n, int m, double badModelProb, double errTol, List<Feature> inliers | | Метод получения точек-инлаеров и матрицы трансормации алгоритмом RANSAC |
| buildModel | private static | RealMatrix | Point[] pts, Point[] mpts, int n | | Возвращает 3х3 матрицу транформации из точек объекта в точки сцены |
| calcMinInliers | private static | int | int n, int m, double badModelProb, double badTransfProb | | Вычисляет минимальное число инлаеров, которое будет гарантировать вероятность плохой модели меньше, чем badTransfProb |
| getRansacSample | private static | List<Feature> | List<Feature> features, int n, int m | | Берет случайную выборку из совпавших точек сцены |
| extractCorrespPts | private static | void | List<Feature> features, int n, List<Point> pts, List<Point> mpts | | Получает координаты точек сцены, соответствующие точкам объекта |
| findConsensus | private static | int | List<Feature> features, int n, RealMatrix M, double errTol, List<Feature> consensus | | При помощи функции ошибки вычисляет консенсус (число точек, удовлетворяющих модели) |
| modelTransformErr | private static | double | Point pt, Point mpt, RealMatrix H | | Функция ошибки матрицы трансформации из точек объекта в точки сцены |
| transformPoint | private static | Point | Point pt, RealMatrix T | | Переводит координаты точки в новую систему координат при помощи матрицы трансформации Т |

**ПРИЛОЖЕНИЕ 3**

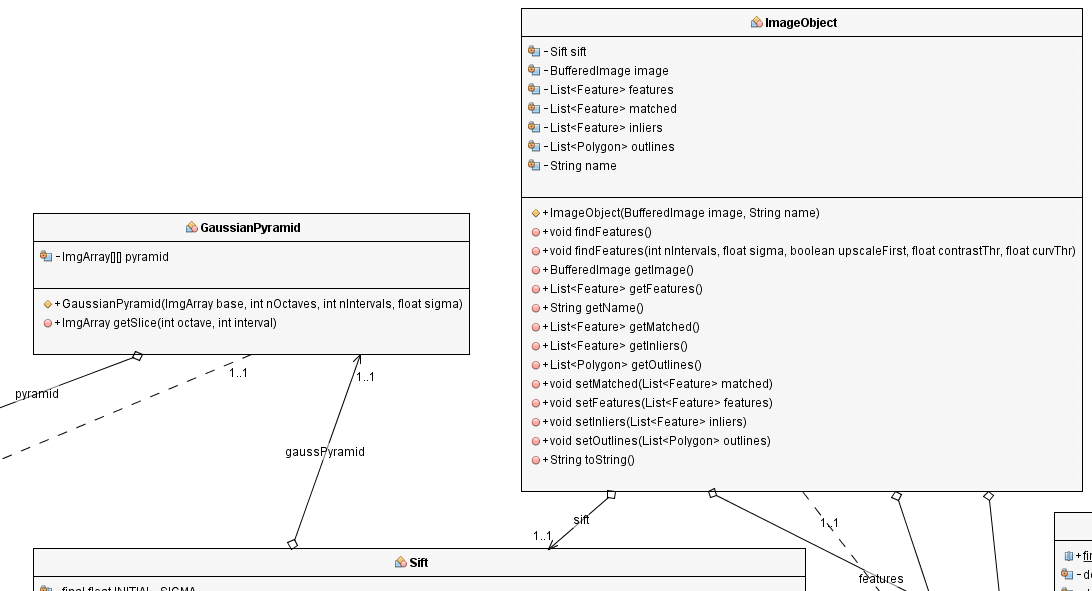
**ДИАГРАММЫ КЛАССОВ**



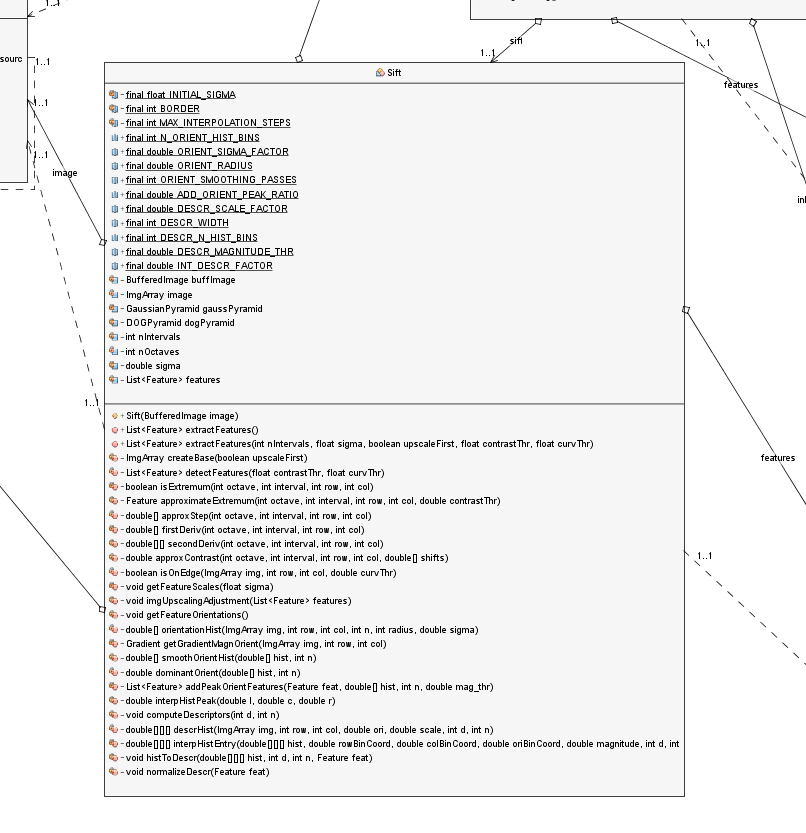
*Рис. 1. Общий вид диаграммы классов*



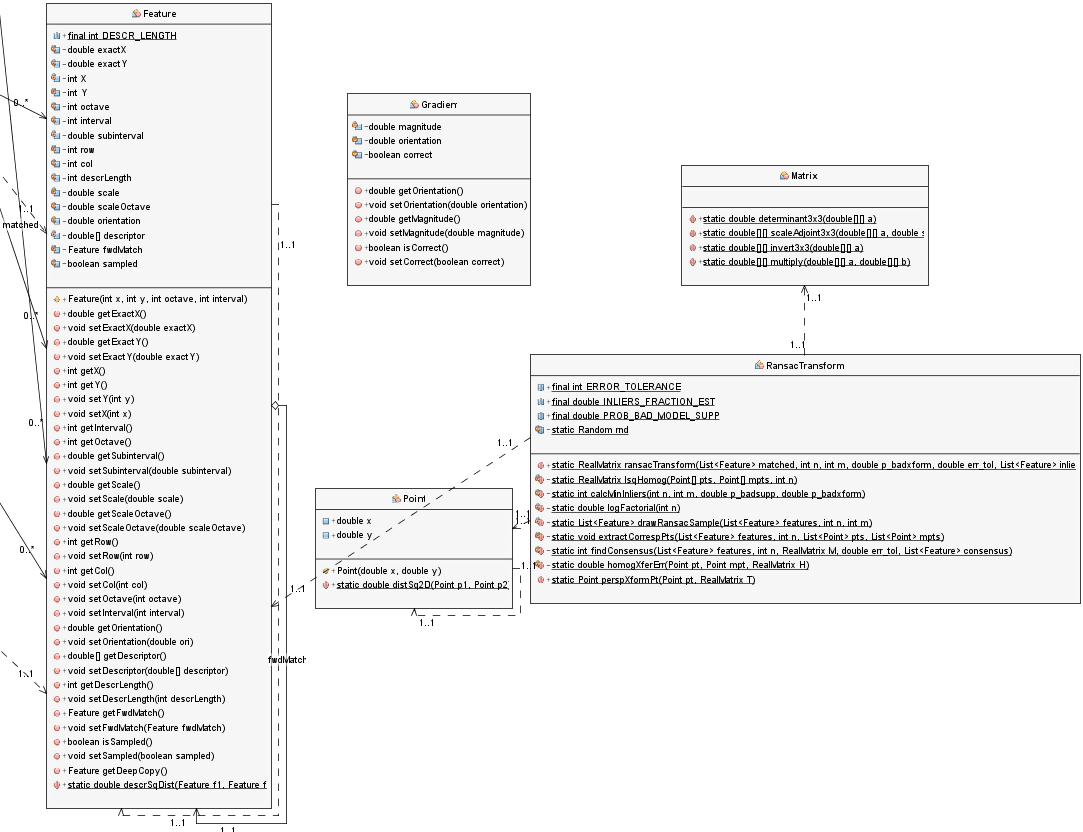
*Рис. 2. Диаграммы классов DOGPyramid, Filter, ImgArray, Scaling*



*Рис. 3. Диаграммы классов GaussianPyramid, ImageObject*



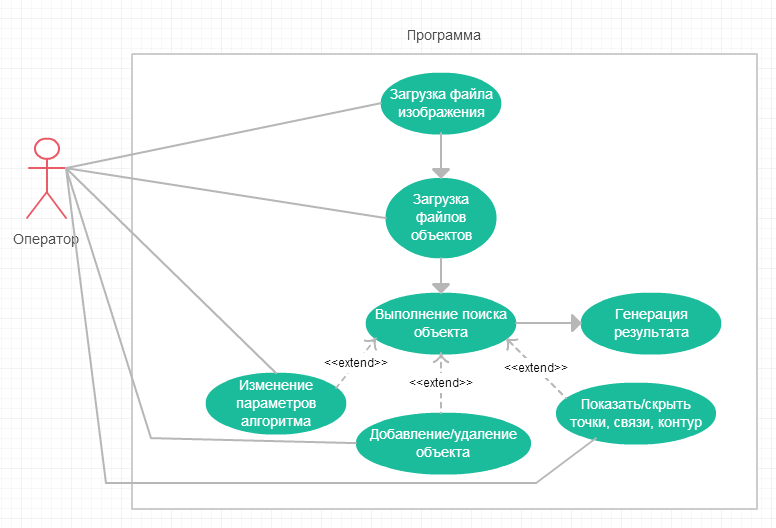
*Рис. 3. Диаграмма класса Sift*



*Рис. 4. Диаграммы классов Feature, Point, Matrix, RansacTransform, Gradient*

**ПРИЛОЖЕНИЕ 4**

**ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ**



*Рис. 4. Диаграмма вариантов использования*

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |