Нахождение объектов на изображении с использованием особенностей

Описанный в предыдущем параграфе подход к определению особенностей можно применять для нахождения объектов на изображении. Есть пример использования в OpenCV – find_obj.cpp. Есть объект (Рис. 1) и сцена (Рис. 2).

Рис. 1. Объект распознавания

Рис. 2. Сцена с искомым объектом

Ниже представлен текст программы примера из OpenCV с комментариями (Листинг 1).

```
#include <cv.h>
#include <highgui.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>

#include <iostream>
#include <vector>

using namespace std;

IplImage *image = 0;

  // Сравнение двух особенностей
double
compareSURFDescriptors( const float* d1, const float* d2, double best, int
length )
{
  double total_cost = 0;
  assert( length % 4 == 0 );
  for( int i = 0; i < length; i += 4 )
  {
    double t0 = d1[i] - d2[i];
    double t1 = d1[i+1] - d2[i+1];
    double t2 = d1[i+2] - d2[i+2];
    double t3 = d1[i+3] - d2[i+3];
    total_cost += t0*t0 + t1*t1 + t2*t2 + t3*t3;
    if( total_cost > best )
      break;
  }
  return total_cost;
}
  // Сравнивает одну особенность объекта со всеми особенностями сцены
int
naiveNearestNeighbor( const float* vec, int laplacian,
```

```
          const CvSeq* model_keypoints,
          const CvSeq* model_descriptors )
{
  int length = (int)(model_descriptors->elem_size/sizeof(float));
  int i, neighbor = -1;
  double d, dist1 = 1e6, dist2 = 1e6;
  CvSeqReader reader, kreader;
  // Начальная особенность сцены
  cvStartReadSeq( model_keypoints, &kreader, 0 );
  cvStartReadSeq( model_descriptors, &reader, 0 );

  // Перебор всех особенностей сцены
  for( i = 0; i < model_descriptors->total; i++ )
  {
    const CvSURFPoint* kp = (const CvSURFPoint*)kreader.ptr;
    const float* mvec = (const float*)reader.ptr;
    CV_NEXT_SEQ_ELEM( kreader.seq->elem_size, kreader );
    CV_NEXT_SEQ_ELEM( reader.seq->elem_size, reader );
    // Для ускорения сначала сравнивается лапласиан особенностей
    if( laplacian != kp->laplacian )
      continue;
    // Сравнение особенностей
    d = compareSURFDescriptors( vec, mvec, dist2, length );
    if( d < dist1 )
    {
      // Найдена лучшее совпадение особенностей
      dist2 = dist1;
      dist1 = d;
      neighbor = i;
    }
    else if ( d < dist2 )
      dist2 = d;
  }
  if ( dist1 < 0.6*dist2 )
    return neighbor;
  return -1;
```

```
}


  // Функция ищет совпадающие пары
void
findPairs( const CvSeq* objectKeypoints, const CvSeq* objectDescriptors,
     const CvSeq* imageKeypoints, const CvSeq* imageDescriptors, vector<int>&
ptpairs )
{
  int i;
  CvSeqReader reader, kreader;
  // Установка начальной особенности объекта распознавания
  cvStartReadSeq( objectKeypoints, &kreader );
  cvStartReadSeq( objectDescriptors, &reader );
  ptpairs.clear();

  // Перебор всех особенностетей объекта
  for( i = 0; i < objectDescriptors->total; i++ )
  {
    const CvSURFPoint* kp = (const CvSURFPoint*)kreader.ptr;
    const float* descriptor = (const float*)reader.ptr;
    CV_NEXT_SEQ_ELEM( kreader.seq->elem_size, kreader );
    CV_NEXT_SEQ_ELEM( reader.seq->elem_size, reader );
    // Сравнение текущей особенности со всеми особенностями из сцены
    int nearest_neighbor = naiveNearestNeighbor( descriptor, kp->laplacian,
imageKeypoints, imageDescriptors );
    if( nearest_neighbor >= 0 )
    {
      // Нашлось совпадение особенностей
      ptpairs.push_back(i);
      ptpairs.push_back(nearest_neighbor);
    }
  }
}


/* Грубое нахождение местоположения объекта */
int
locatePlanarObject( const CvSeq* objectKeypoints, const CvSeq*
```

```
objectDescriptors,
         const CvSeq* imageKeypoints, const CvSeq* imageDescriptors,
         const CvPoint src_corners[4], CvPoint dst_corners[4] )
{
  double h[9];
  CvMat _h = cvMat(3, 3, CV_64F, h);
  vector<int> ptpairs;
  vector<CvPoint2D32f> pt1, pt2;
  CvMat _pt1, _pt2;
  int i, n;
  // Ищем пары особенностей на обеих картинках, которые соответствуют
  // друг другу
  findPairs( objectKeypoints, objectDescriptors, imageKeypoints,
imageDescriptors, ptpairs );
  n = ptpairs.size()/2;
  // Если пар мало, значит надо выходить - объект не найден
  if( n < 4 )
    return 0;
  // Выделяем память
  pt1.resize(n);
  pt2.resize(n);
  // Считываем координаты «особых»точек
  for( i = 0; i < n; i++ )
  {
    pt1[i] = ((CvSURFPoint*)cvGetSeqElem(objectKeypoints,ptpairs[i*2]))->pt;
    pt2[i] = ((CvSURFPoint*)cvGetSeqElem(imageKeypoints,ptpairs[i*2+1]))->pt;
  }

  // По полученным векторам создаём матриц
  _pt1 = cvMat(1, n, CV_32FC2, &pt1[0] );
  _pt2 = cvMat(1, n, CV_32FC2, &pt2[0] );
  // Находим трансформацию между исходным изображением и с тем, которое
  // ищем
  if( !cvFindHomography( &_pt1, &_pt2, &_h, CV_RANSAC, 5 ))
    return 0;
  // По полученному значению трансформации (в матрицу _h) находим
```

```
  // координаты четырёхугольника, характеризующего объект
  for( i = 0; i < 4; i++ )
  {
    double x = src_corners[i].x, y = src_corners[i].y;
    double Z = 1./(h[6]*x + h[7]*y + h[8]);
    double X = (h[0]*x + h[1]*y + h[2])*Z;
    double Y = (h[3]*x + h[4]*y + h[5])*Z;
    dst_corners[i] = cvPoint(cvRound(X), cvRound(Y));
  }

  return 1;
}

int main(int argc, char** argv)
{
  // Инициализация параметров
  const char* object_filename = argc == 3 ? argv[1] : "box.png";
  const char* scene_filename = argc == 3 ? argv[2] : "box_in_scene.png";

  CvMemStorage* storage = cvCreateMemStorage(0);

  cvNamedWindow("Object", 1);
  cvNamedWindow("Object Correspond", 1);

  static CvScalar colors[] =
  {
    {{0,0,255}},
    {{0,128,255}},
    {{0,255,255}},
    {{0,255,0}},
    {{255,128,0}},
    {{255,255,0}},
    {{255,0,0}},
    {{255,0,255}},
    {{255,255,255}}
  };
```

```
  // Загрузка изображений
  IplImage* object = cvLoadImage( object_filename, CV_LOAD_IMAGE_GRAYSCALE );
  IplImage* image = cvLoadImage( scene_filename, CV_LOAD_IMAGE_GRAYSCALE );
  if( !object || !image )
  {
    fprintf( stderr, "Can not load %s and/or %s\n"
      "Usage: find_obj [<object_filename> <scene_filename>]\n",
      object_filename, scene_filename );
    exit(-1);
  }
  // Перевод в градации серого
  IplImage* object_color = cvCreateImage(cvGetSize(object), 8, 3);
  cvCvtColor( object, object_color, CV_GRAY2BGR );

  CvSeq *objectKeypoints = 0, *objectDescriptors = 0;
  CvSeq *imageKeypoints = 0, *imageDescriptors = 0;
  int i;
  // Инициализация структуры CvSURFParams с размером дескрипторов в 128
  // элементов
  CvSURFParams params = cvSURFParams(500, 1);

  // Засекаем время
  double tt = (double)cvGetTickCount();
  // Ищем особенности объекта распознавания
  cvExtractSURF( object, 0, &objectKeypoints, &objectDescriptors, storage,
params );
  printf("Object Descriptors: %d\n", objectDescriptors->total);
  // Ищем особенности сцены
  cvExtractSURF( image, 0, &imageKeypoints, &imageDescriptors, storage,
params );
  printf("Image Descriptors: %d\n", imageDescriptors->total);
  // Сколько потребовалось времени (У меня 167 милли секунд)
  tt = (double)cvGetTickCount() - tt;
  printf( "Extraction time = %gms\n", tt/(cvGetTickFrequency()*1000.));
  // Устанавливаем границы изображений, внутри которых будут сравниваться
```

```
  //особенности
  CvPoint src_corners[4] = {{0,0}, {object->width,0}, {object->width, object->height}, {0, object->height}};
  CvPoint dst_corners[4];
  // Создание дополнительного изображение (в нём будет сцена и объект)
  // Запустите пример и поймёте о чём речь
  IplImage* correspond = cvCreateImage( cvSize(image->width, object->height+image->height), 8, 1 );
  cvSetImageROI( correspond, cvRect( 0, 0, object->width, object->height ) );
  cvCopy( object, correspond );
  cvSetImageROI( correspond, cvRect( 0, object->height, correspond->width, correspond->height ) );
  cvCopy( image, correspond );
  cvResetImageROI( correspond );
  // Вызываем функцию, находящую объект на экране
  if( locatePlanarObject( objectKeypoints, objectDescriptors, imageKeypoints,
    imageDescriptors, src_corners, dst_corners ))
  {
     // Обводим нужный четырёхугольник

    for( i = 0; i < 4; i++ )
    {
      CvPoint r1 = dst_corners[i%4];
      CvPoint r2 = dst_corners[(i+1)%4];
      cvLine( correspond, cvPoint(r1.x, r1.y+object->height ),
        cvPoint(r2.x, r2.y+object->height ), colors[8] );
    }
  }
  // Если в этом месте вывести результат на экран, то получиться то, что
  // показано на рисунке 23.3.

  vector<int> ptpairs;
  // Снова ищутся все совпадающие пары особенностей в обеих картинках
  findPairs( objectKeypoints, objectDescriptors, imageKeypoints, imageDescriptors, ptpairs );

  // Между парами особенностей на рисунке проводятся прямые
```

```
  for( i = 0; i < (int)ptpairs.size(); i += 2 )
  {
    CvSURFPoint* r1 = (CvSURFPoint*)cvGetSeqElem( objectKeypoints,
ptpairs[i] );
    CvSURFPoint* r2 = (CvSURFPoint*)cvGetSeqElem( imageKeypoints,
ptpairs[i+1] );
    cvLine( correspond, cvPointFrom32f(r1->pt),
      cvPoint(cvRound(r2->pt.x), cvRound(r2->pt.y+object->height)),
colors[8] );
  }

  // Результат можно посмотреть на риснуке 23.4.
  cvShowImage( "Object Correspond", correspond );

  // Выделяем особенностио окружностями (Рис. 23.5)
  for( i = 0; i < objectKeypoints->total; i++ )
  {
    CvSURFPoint* r = (CvSURFPoint*)cvGetSeqElem( objectKeypoints, i );
    CvPoint center;
    int radius;
    center.x = cvRound(r->pt.x);
    center.y = cvRound(r->pt.y);
    radius = cvRound(r->size*1.2/9.*2);
    cvCircle( object_color, center, radius, colors[0], 1, 8, 0 );
  }
  cvShowImage( "Object", object_color );

  cvWaitKey(0);

  cvDestroyWindow("Object");
  cvDestroyWindow("Object SURF");
  cvDestroyWindow("Object Correspond");

  return 0;
}
```

Листинг 1. Поиск объектов на сцене

Рис. 3. Выделен искомый объект

Рис. 4. Выделен искомый объект

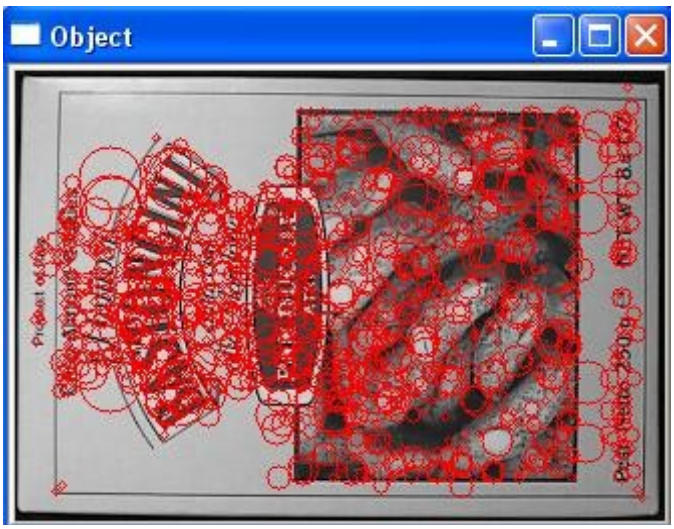

Рис. 5. Все особенности объекта