Praktikum 12
Object Detection

1. Face Detection (Offline)

Program berikut ini mengaplikasikan face detection dengan menggunakan Haar Classifier secara offline.

```
#include "cv.h"
#include "cxcore.h"
#include "highgui.h"

int main()
{
        IplImage* img;
        img = cvLoadImage( "face.jpg" );
        CvMemStorage* storage = cvCreateMemStorage(0);
        // Note that you must copy C:\Program
Files\OpenCV\data\haarcascades\haarcascade_frontalface_alt2.xml
        // to your working directory
        CvHaarClassifierCascade* cascade = (CvHaarClassifierCascade*)cvLoad(
"haarcascade_frontalface_alt2.xml" );
        double scale = 1.3;

        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}} };

        // Detect objects
        cvClearMemStorage( storage );
        CvSeq* objects = cvHaarDetectObjects( img, cascade, storage, 1.1, 4, 0, cvSize(
40, 50 ));

        CvRect* r;
        // Loop through objects and draw boxes
        for( int i = 0; i < (objects ? objects->total : 0 ); i++ ){
                r = ( CvRect* )cvGetSeqElem( objects, i );
                cvRectangle( img, cvPoint( r->x, r->y ), cvPoint( r->x + r->width, r->y
+ r->height ),
                        colors[i%8]);
        }

        cvNamedWindow( "Output" );
        cvShowImage( "Output", img );
        cvWaitKey();

        cvReleaseImage( &img );

        return 0;
}
```

Petunjuk praktikum:

- Copykan file **`C:\Program Files\OpenCV\data\haarcascades\haarcascade_frontalface_alt2.xml`** pada direktory kerja.
- Jalankan program diatas, kemudian coba gunakan beberapa gambar yang terdiri dari banyak wajah.
- Amati perbedaan tingkat keberhasilan deteksi wajah. Jelaskan mengapa tidak semua wajah dapat terdeteksi?

2. Face Detection (Online)

Program berikut ini mengaplikasikan face detection dengan menggunakan Haar Classifier secara online.

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

CvHaarClassifierCascade *cascade;
CvMemStorage            *storage;

void detectFaces( IplImage *img );

int main( int argc, char** argv )
{

        CvCapture *capture;
        IplImage  *frame;
        char       key = 0;
        char      *filename = "haarcascade_frontalface_alt2.xml";

        /* load the classifier note that I put the file in the same directory with this
code */
        cascade = ( CvHaarClassifierCascade* )cvLoad( filename, 0, 0, 0 );

        /* setup memory buffer; needed by the face detector */
        storage = cvCreateMemStorage( 0 );

        /* initialize camera */
        capture = cvCaptureFromCAM( 0 );

        /* always check */
        assert( cascade && storage && capture );

        /* create a window */
        cvNamedWindow( "video", 1 );

        while( key != 'q' ) {

                /* get a frame */
                frame = cvQueryFrame( capture );

                /* always check */
                if( !frame ) break;

                /* 'fix' frame */
                //cvFlip( frame, frame, -1 );
                //frame->origin = 0;

                /* detect faces and display video */
                detectFaces( frame );

                /* quit if user press 'q' */
                key = cvWaitKey( 10 );
        }

        /* free memory */
        cvReleaseCapture( &capture );
        cvDestroyWindow( "video" );
        cvReleaseHaarClassifierCascade( &cascade );
        cvReleaseMemStorage( &storage );

        return 0;
}

void detectFaces( IplImage *img )
{
        int i;
        /* detect faces */
        CvSeq *faces = cvHaarDetectObjects(
                img,
                cascade,
```

```
                storage,
                1.1,
                3,
                0
                /*CV_HAAR_DO_CANNY_PRUNNING*/,
                cvSize( 40, 40 )
        );

        /* for each face found, draw a red box */

        for( i = 0 ; i < ( faces ? faces->total : 0 ) ; i++ ) {
                CvRect *r = ( CvRect* )cvGetSeqElem( faces, i );
                cvRectangle( img,
                cvPoint( r->x, r->y ),
                cvPoint( r->x + r->width, r->y + r->height ),
                CV_RGB( 255, 0, 0 ), 1, 8, 0 );
        }

        /* display video */
        cvShowImage( "video", img );
}
```

Petunjuk praktikum:

- Copykan file **`C:\Program Files\OpenCV\data\haarcascades\haarcascade_frontalface_alt2.xml`** pada direktory kerja.
- Jalankan program diatas, kemudian coba gunakan beberapa cascade yang lain yang ada pada folder diatas.
- Jelaskan perbedaan penggunaan beberapa cascade.

3. Find Object (Offline)

Program berikut ini melakukan pencarian objek dengan cara mencocokkan fitur pada gambar asal dan mencari fitur tersebut pada gambar yang lainnya secara offline.

```
/*
 * A Demo to OpenCV Implementation of SURF
 * Further Information Refer to "SURF: Speed-Up Robust Feature"
 * Author: Liu Liu
 * liuliu.1987+opencv@gmail.com
 */

#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;

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;
}

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;
    cvStartReadSeq( model_keypoints, &kreader, 0 );
    cvStartReadSeq( model_descriptors, &reader, 0 );

    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 );
        if( laplacian != kp->laplacian )
            continue;
        d = compareSURFDescriptors( vec, mvec, dist2, length );
        if( d < dist1 )
        {
            dist2 = dist1;
            dist1 = d;
            neighbor = i;
        }
        else if ( d < dist2 )
            dist2 = d;
    }
    if ( dist1 < 0.6*dist2 )
        return neighbor;
    return -1;
}

void
findPairs( const CvSeq* objectKeypoints, const CvSeq* objectDescriptors,
           const CvSeq* imageKeypoints, const CvSeq* imageDescriptors, vector<int>&
ptpairs )
{
    int i;
    CvSeqReader reader, kreader;
    cvStartReadSeq( objectKeypoints, &kreader );
    cvStartReadSeq( objectDescriptors, &reader );
    ptpairs.clear();

    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 );
        int nearest_neighbor = naiveNearestNeighbor( descriptor, kp->laplacian,
imageKeypoints, imageDescriptors );
        if( nearest_neighbor >= 0 )
        {
            ptpairs.push_back(i);
            ptpairs.push_back(nearest_neighbor);
        }
    }
}

/* a rough implementation for object location */
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;

    findPairs( objectKeypoints, objectDescriptors, imageKeypoints, imageDescriptors,
ptpairs );
    n = ptpairs.size()/2;
    if( n < 4 )
        return 0;

    pt1.resize(n);
    pt2.resize(n);
    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;
    }

    _pt1 = cvMat(1, n, CV_32FC2, &pt1[0] );
    _pt2 = cvMat(1, n, CV_32FC2, &pt2[0] );
    if( !cvFindHomography( &_pt1, &_pt2, &_h, CV_RANSAC, 5 ))
        return 0;

    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)
{
    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}}
    };

    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);
    }
    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;
    CvSURFParams params = cvSURFParams(500, 1);
```

```
    double tt = (double)cvGetTickCount();
    cvExtractSURF( object, 0, &objectKeypoints, &objectDescriptors, storage, params );
    printf("Object Descriptors: %d\n", objectDescriptors->total);
    cvExtractSURF( image, 0, &imageKeypoints, &imageDescriptors, storage, params );
    printf("Image Descriptors: %d\n", imageDescriptors->total);
    tt = (double)cvGetTickCount() - tt;
    printf( "Extraction time = %gms\n", tt/(cvGetTickFrequency()*1000.));
    CvPoint src_corners[4] = {{0,0}, {object->width,0}, {object->width, object-
>height}, {0, object->height}};
    CvPoint dst_corners[4];
    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 );
    if( locatePlanarObject( objectKeypoints, objectDescriptors, imageKeypoints,
        imageDescriptors, src_corners, dst_corners ))
    {
        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] );
        }
    }
    vector<int> ptpairs;
    findPairs( objectKeypoints, objectDescriptors, imageKeypoints, imageDescriptors,
ptpairs );
    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] );
    }

    cvShowImage( "Object Correspond", correspond );
    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;
}
```

Petunjuk praktikum:

- Ambil sebuah gambar dari objek secara utuh, kemudian letakkan objek tersebut bersama dengan objek yang lainnya sehingga objek tersebut tertutupi. Kemudian jalankan program di atas.

- Berikut ini adalah contoh objek yang ada dalam Sample OpenCV, cobalah dengan menggunakan objek lainnya.

- Jelaskan algoritma SURF (Speed-Up Robust Feature) pada percobaan diatas!