Praktikum 8
Feature Extraction

1. Sobel Edge Detection

Program berikut ini menerapkan sobel edge detection untuk deteksi tepi pada gambar.

```
#include <cv.h>                                    //main OpenCV header
#include <highgui.h>                            //GUI header

int main()
{
        // Declare a new IplImage pointer
        IplImage* gray;
        IplImage* dst;

        // Load an image
        gray = cvLoadImage("image.jpg",0);

        // Create a new window & display the image
        cvNamedWindow("Grayscale", 1);
        cvMoveWindow("Grayscale", 300, 100);
        cvNamedWindow("Destination", 1);
        cvMoveWindow("Destination", 500, 100);

        // Fuction of processing image
        dst = cvCreateImage( cvSize(gray->width, gray->height), IPL_DEPTH_8U, 1 );

        // Fuction of processing image
        cvSobel(gray, dst, 1, 1, 5);

        cvShowImage("Grayscale", gray);
        cvShowImage("Destination", dst);

        // Wait for key to close the window
        cvWaitKey(0);
        cvDestroyWindow( "Grayscale" );
        cvReleaseImage( &gray );
        cvDestroyWindow( "Destination" );
        cvReleaseImage( &dst );
        return 0;
}
```

Petunjuk praktikum:

- Jelaskan konsep deteksi tepi menggunakan metode Sobel pada program di atas.
- Jelaskan fungsi berikut ini beserta dengan parameter yang ada di dalamnya.
    - `cvSobel()`

2. Canny Edge Detection with Trackbar

Program berikut ini menerapkan canny edge detection untuk deteksi tepi pada gambar grayscale.

```
#include "cv.h"                                    //main OpenCV header
#include "highgui.h"                            //GUI header

int high_switch_value = 0;
int highInt = 0;
int low_switch_value = 0;
int lowInt = 0;

void switch_callback_h( int position ){
        highInt = position;
}
void switch_callback_l( int position ){
        lowInt = position;
}

int main( int argc, char** argv )
{
        const char* name = "Edge Detection Window";

        // Kernel size
        int N = 7;

        // Set up images
        IplImage* img = cvLoadImage( "image.jpg", 0 );
        IplImage* img_b = cvCreateImage( cvSize(img->width+N-1,img->height+N-1), img-
>depth, img->nChannels );
        IplImage* out = cvCreateImage(cvGetSize(img_b),IPL_DEPTH_8U,img_b->nChannels);

        // Add convolution boarders
        CvPoint offset = cvPoint((N-1)/2,(N-1)/2);
        cvCopyMakeBorder(img, img_b, offset, IPL_BORDER_REPLICATE, cvScalarAll(0));

        // Make window
        cvNamedWindow( name, 1 );

        // Edge Detection Variables
        int aperature_size = N;
        double lowThresh = 20;
        double highThresh = 40;

        // Create trackbars
        cvCreateTrackbar( "High", name, &high_switch_value, 4, switch_callback_h );
        cvCreateTrackbar( "Low", name, &low_switch_value, 4, switch_callback_l );

        while( 1 ) {
                switch( highInt ){
                        case 0:
                                highThresh = 200;
                                break;
                        case 1:
                                highThresh = 400;
                                break;
                        case 2:
                                highThresh = 600;
                                break;
                        case 3:
                                highThresh = 800;
                                break;
                        case 4:
                                highThresh = 1000;
                                break;
                }
                switch( lowInt ){
                        case 0:
                                lowThresh = 0;
                                break;
                        case 1:
                                lowThresh = 100;
```

```
                        break;
                case 2:
                        lowThresh = 200;
                        break;
                case 3:
                        lowThresh = 400;
                        break;
                case 4:
                        lowThresh = 600;
                        break;
        }

        // Edge Detection
        cvCanny( img_b, out, lowThresh*N*N, highThresh*N*N, aperature_size );

        cvShowImage(name, out);

        if( cvWaitKey( 15 ) == 27 )
                break;
    }

    // Release
    cvReleaseImage( &img );
    cvReleaseImage( &img_b );
    cvReleaseImage( &out );
    cvDestroyWindow( name );

    return 0;
}
```

Petunjuk praktikum:

- Jelaskan konsep deteksi tepi menggunakan metode Canny pada program di atas.
- Jelaskan fungsi berikut ini beserta dengan parameter yang ada di dalamnya.
    - `cvCanny()`

3. Line Detection using HoughLines

Program berikut ini menerapkan HoughLines untuk deteksi garis pada gambar.

```
/* This is a standalone program. Pass an image name as a first parameter of the
program. Switch between standard and probabilistic Hough transform by changing "#if 1"
to "#if 0" and back */

#include <cv.h>
#include <highgui.h>
#include <math.h>

int main(int argc, char** argv)
{
    const char* filename = argc >= 2 ? argv[1] : "image.jpg";
    IplImage* src = cvLoadImage( filename, 0 );
    IplImage* dst;
    IplImage* color_dst;
    CvMemStorage* storage = cvCreateMemStorage(0);
    CvSeq* lines = 0;
    int i;

    if( !src )
        return -1;

    dst = cvCreateImage( cvGetSize(src), 8, 1 );
```

```
    color_dst = cvCreateImage( cvGetSize(src), 8, 3 );

    cvCanny( src, dst, 50, 200, 3 );
    cvCvtColor( dst, color_dst, CV_GRAY2BGR );
#if 0
    lines = cvHoughLines2( dst, storage, CV_HOUGH_STANDARD, 1, CV_PI/180, 100, 0, 0 );

    for( i = 0; i < MIN(lines->total,100); i++ )
    {
        float* line = (float*)cvGetSeqElem(lines,i);
        float rho = line[0];
        float theta = line[1];
        CvPoint pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a*rho, y0 = b*rho;
        pt1.x = cvRound(x0 + 1000*(-b));
        pt1.y = cvRound(y0 + 1000*(a));
        pt2.x = cvRound(x0 - 1000*(-b));
        pt2.y = cvRound(y0 - 1000*(a));
        cvLine( color_dst, pt1, pt2, CV_RGB(255,0,0), 3, CV_AA, 0 );
    }
#else
    lines = cvHoughLines2( dst, storage, CV_HOUGH_PROBABILISTIC, 1, CV_PI/180, 50, 50,
10 );
    for( i = 0; i < lines->total; i++ )
    {
        CvPoint* line = (CvPoint*)cvGetSeqElem(lines,i);
        cvLine( color_dst, line[0], line[1], CV_RGB(255,0,0), 3, CV_AA, 0 );
    }
#endif
    cvNamedWindow( "Source", 1 );
    cvShowImage( "Source", src );

    cvNamedWindow( "Hough", 1 );
    cvShowImage( "Hough", color_dst );

    cvWaitKey(0);

    return 0;
}
```

Petunjuk praktikum:

- Ubah **#if 0** dengan **#if 1** , kemudian amati perbedaan deteksi garis yang dihasilkan.
- Jelaskan konsep deteksi garis menggunakan HoughLines pada program di atas.
- Jelaskan fungsi berikut ini beserta dengan parameter yang ada di dalamnya.
    - cvHoughLines2()

4. Circle Detection

Program berikut ini menerapkan Circle Detection untuk deteksi lingkaran pada gambar.

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

int main(int argc, char** argv)
{
    IplImage* img;
    img=cvLoadImage("circle3.jpg", 1);

        IplImage* gray = cvCreateImage( cvGetSize(img), 8, 1 );
    CvMemStorage* storage = cvCreateMemStorage(0);
    cvCvtColor( img, gray, CV_BGR2GRAY );
    cvSmooth( gray, gray, CV_GAUSSIAN, 9, 9 ); // smooth it, otherwise a lot of false
circles may be detected
    CvSeq* circles = cvHoughCircles( gray, storage, CV_HOUGH_GRADIENT, 2, gray-
>height/4, 200, 100 );

        int i;
    for( i = 0; i < circles->total; i++ )
    {
         float* p = (float*)cvGetSeqElem( circles, i );
         cvCircle( img, cvPoint(cvRound(p[0]),cvRound(p[1])), 3, CV_RGB(0,255,0), -1,
8, 0 );
         cvCircle( img, cvPoint(cvRound(p[0]),cvRound(p[1])), cvRound(p[2]),
CV_RGB(255,0,0), 3, 8, 0 );
    }

        cvNamedWindow( "circles", 1 );
    cvShowImage( "circles", img );

    cvWaitKey();
        return 0;
}
```

Petunjuk praktikum:

- Jelaskan konsep circle detection (deteksi lingkaran) pada program di atas.
- Jelaskan fungsi berikut ini beserta dengan parameter yang ada di dalamnya.
    - `cvHoughCircles()`
    - `cvCircle()`

5. Square Detection

Program berikut ini menerapkan Square Detection untuk deteksi kotak/persegi pada gambar.

```
// The full "Square Detector" program.
// It loads several images subsequentally and tries to find squares in
// each image

#include <cv.h>
#include <highgui.h>
#include <stdio.h>
#include <math.h>
#include <string.h>

int thresh = 50;
IplImage* img = 0;
IplImage* img0 = 0;
CvMemStorage* storage = 0;
const char* wndname = "Square Detection Demo";

// helper function:
// finds a cosine of angle between vectors
// from pt0->pt1 and from pt0->pt2
double angle( CvPoint* pt1, CvPoint* pt2, CvPoint* pt0 )
{
    double dx1 = pt1->x - pt0->x;
    double dy1 = pt1->y - pt0->y;
    double dx2 = pt2->x - pt0->x;
    double dy2 = pt2->y - pt0->y;
    return (dx1*dx2 + dy1*dy2)/sqrt((dx1*dx1 + dy1*dy1)*(dx2*dx2 + dy2*dy2) + 1e-10);
}

// returns sequence of squares detected on the image.
// the sequence is stored in the specified memory storage
CvSeq* findSquares4( IplImage* img, CvMemStorage* storage )
{
    CvSeq* contours;
    int i, c, l, N = 11;
    CvSize sz = cvSize( img->width & -2, img->height & -2 );
    IplImage* timg = cvCloneImage( img ); // make a copy of input image
    IplImage* gray = cvCreateImage( sz, 8, 1 );
    IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 );
    IplImage* tgray;
    CvSeq* result;
    double s, t;
    // create empty sequence that will contain points -
    // 4 points per square (the square's vertices)
    CvSeq* squares = cvCreateSeq( 0, sizeof(CvSeq), sizeof(CvPoint), storage );

    // select the maximum ROI in the image
    // with the width and height divisible by 2
    cvSetImageROI( timg, cvRect( 0, 0, sz.width, sz.height ));

    // down-scale and upscale the image to filter out the noise
    cvPyrDown( timg, pyr, 7 );
    cvPyrUp( pyr, timg, 7 );
    tgray = cvCreateImage( sz, 8, 1 );

    // find squares in every color plane of the image
    for( c = 0; c < 3; c++ )
    {
        // extract the c-th color plane
        cvSetImageCOI( timg, c+1 );
        cvCopy( timg, tgray, 0 );

        // try several threshold levels
        for( l = 0; l < N; l++ )
        {
            // hack: use Canny instead of zero threshold level.
            // Canny helps to catch squares with gradient shading
            if( l == 0 )
            {
```

```
                // apply Canny. Take the upper threshold from slider
                // and set the lower to 0 (which forces edges merging)
                cvCanny( tgray, gray, 0, thresh, 5 );
                // dilate canny output to remove potential
                // holes between edge segments
                cvDilate( gray, gray, 0, 1 );
            }
            else
            {
                // apply threshold if l!=0:
                //     tgray(x,y) = gray(x,y) < (l+1)*255/N ? 255 : 0
                cvThreshold( tgray, gray, (l+1)*255/N, 255, CV_THRESH_BINARY );
            }

            // find contours and store them all as a list
            cvFindContours( gray, storage, &contours, sizeof(CvContour),
                CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );

            // test each contour
            while( contours )
            {
                // approximate contour with accuracy proportional
                // to the contour perimeter
                result = cvApproxPoly( contours, sizeof(CvContour), storage,
                    CV_POLY_APPROX_DP, cvContourPerimeter(contours)*0.02, 0 );
                // square contours should have 4 vertices after approximation
                // relatively large area (to filter out noisy contours)
                // and be convex.
                // Note: absolute value of an area is used because
                // area may be positive or negative - in accordance with the
                // contour orientation
                if( result->total == 4 &&
                    fabs(cvContourArea(result,CV_WHOLE_SEQ)) > 1000 &&
                    cvCheckContourConvexity(result) )
                {
                    s = 0;

                    for( i = 0; i < 5; i++ )
                    {
                        // find minimum angle between joint
                        // edges (maximum of cosine)
                        if( i >= 2 )
                        {
                            t = fabs(angle(
                            (CvPoint*)cvGetSeqElem( result, i ),
                            (CvPoint*)cvGetSeqElem( result, i-2 ),
                            (CvPoint*)cvGetSeqElem( result, i-1 )));
                            s = s > t ? s : t;
                        }
                    }

                    // if cosines of all angles are small
                    // (all angles are ~90 degree) then write quandrange
                    // vertices to resultant sequence
                    if( s < 0.3 )
                        for( i = 0; i < 4; i++ )
                            cvSeqPush( squares,
                                (CvPoint*)cvGetSeqElem( result, i ));
                }

                // take the next contour
                contours = contours->h_next;
            }
        }
    }

    // release all the temporary images
    cvReleaseImage( &gray );
    cvReleaseImage( &pyr );
    cvReleaseImage( &tgray );
    cvReleaseImage( &timg );

    return squares;
}
// the function draws all the squares in the image
void drawSquares( IplImage* img, CvSeq* squares )
```

```
{
    CvSeqReader reader;
    IplImage* cpy = cvCloneImage( img );
    int i;

    // initialize reader of the sequence
    cvStartReadSeq( squares, &reader, 0 );

    // read 4 sequence elements at a time (all vertices of a square)
    for( i = 0; i < squares->total; i += 4 )
    {
        CvPoint pt[4], *rect = pt;
        int count = 4;

        // read 4 vertices
        CV_READ_SEQ_ELEM( pt[0], reader );
        CV_READ_SEQ_ELEM( pt[1], reader );
        CV_READ_SEQ_ELEM( pt[2], reader );
        CV_READ_SEQ_ELEM( pt[3], reader );

        // draw the square as a closed polyline
        cvPolyLine( cpy, &rect, &count, 1, 1, CV_RGB(0,255,0), 3, CV_AA, 0 );
    }

    // show the resultant image
    cvShowImage( wndname, cpy );
    cvReleaseImage( &cpy );
}

char* names[] = { "pic1.png", "pic2.png", "pic3.png",
                  "pic4.png", "pic5.png", "pic6.png", 0 };

int main(int argc, char** argv)
{
    int i, c;
    // create memory storage that will contain all the dynamic data
    storage = cvCreateMemStorage(0);

    for( i = 0; names[i] != 0; i++ )
    {
        // load i-th image
        img0 = cvLoadImage( names[i], 1 );
        if( !img0 )
        {
            printf("Couldn't load %s\n", names[i] );
            continue;
        }
        img = cvCloneImage( img0 );

        // create window and a trackbar (slider) with parent "image" and set callback
        // (the slider regulates upper threshold, passed to Canny edge detector)
        cvNamedWindow( wndname, 1 );

        // find and draw the squares
        drawSquares( img, findSquares4( img, storage ) );

        // wait for key.
        // Also the function cvWaitKey takes care of event processing
        c = cvWaitKey(0);
        // release both images
        cvReleaseImage( &img );
        cvReleaseImage( &img0 );
        // clear memory storage - reset free space position
        cvClearMemStorage( storage );
        if( (char)c == 27 )
            break;
    }

    cvDestroyWindow( wndname );

    return 0;
}
```

Petunjuk praktikum:

- Jelaskan konsep square detection (deteksi persegi) pada pada program di atas.
- Jelaskan fungsi berikut ini beserta dengan parameter yang ada di dalamnya.
    - `findSquares4()`

Tugas:

Buatlah program untuk corner detection dengan menggunakan fungsi di bawah ini:

- `cvPreCornerDetect()`
- `cvCornerEigenValsAndVecs()`
- `cvCornerHarris()`
- `cvFindCornerSubPix()`
- `cvGoodFeaturesToTrack()`

Buatlah sebuah program untuk mengambil fitur yang robust pada sebuah gambar menggunakan fungsi Extracts Speeded Up Robust Features berikut ini:

- `cvExtractSURF()`