IMAGE PROCESSING APPLICATIONS USING C++ AND MATLAB

Enrollment. No. - 071019,071020,071056

Name of Student - SURAMYA, DRIPTO, JAGAT

Name of supervisor(s) - MR. SUNIL BHOOSHAN

May – 2011

Submitted in partial fulfillment of the Degree of

Bachelor of Technology

B. Tech

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Table Of Contents

CERTIFICATE

This is to certify that the work titled "IMAGE PROCESSING USING MATLAB AND C++" submitted by "SURAMYA TYAGI, DRIPTO GANGULY, and JAGATPRATAP SINGH." in partial fulfillment for the award of degree of E.C.E B. Tech, of Jaypee University of Information Technology; Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor .....

Name of Supervisor Prof. S.V. Bhooshan

Designation HOD (ECE)

Date 24/5/2011

ACKNOWLEDGEMENT

The completion of any project brings with it a sense of satisfaction, but it is never complete without thanking those people who made it possible and whose constant support has crowned our efforts with success.

I would like to thank our guide, **DR. S.V BHOOSHAN** Dept. of Electronics and Communication for his expert guidance, encouragement and valuable suggestions at every step.

We are extremely happy to acknowledge and express our sincere gratitude to our parents for their constant support and encouragement and last but not the least, friends and well wishers for their help and cooperation and solutions to problems during the course of the project.

Name of Student DRIPTO. SURAMYA, JAGAT PRATAP

Date 24/5/2011

Signature of the student

SUMMARY

In 7th semester we worked on pictures which were still there we found the edges of the pictures with the help on in build tools of matlab.In project part 1 we found out the histogram of images,we applied filter operations on images ,cropping,resizing,histogram equalization,morphological operation ,threshold operation,brightness,contrast..These all operations work on edges of an image. In the current semester we worked on motion pictures .We worked on image detection,face recognition and motion tracking.We basically worked on building a GUI so that we can perform all these tasks in one single software.

Signature of Student

Signature of Supervisor

Name DRIPTO, SURAMYA, JAGAT

THEORY

Histogram

They are frequency distributions that describe the frequency of the intensity values that occur in an image .

Histogram Equalization

A frequent task is to adjust two different images in such a way that their resulting intensity distributions are similar, for example to use them in a print publication or to make them easier to compare.The goal of histogram equalization is to find and apply a point operation such that the histogram of the modified image approximates a uniform distribution.

Threshold Operation

Thresholding an image is a special type of quantization that separates the pixel values in two classes, depending upon a given threshold value ath that is usually constant. The threshold function fthreshold (a) maps all pixels to one of two fixed intensity values a0 or a1.

fthreshold (a) =

a0 for a < ath

a1 for a >ath

Sobel Operators

These operators use linear filters that extend over 3 adjacent lines and columns to counteract the noise sensitivity of simple one line/column gradient operators .

Canny operation

Canny methods finds edges by looking local maxima of the gradient of image. The gradient is calculated using the derivative of gaussian filter .The method uses two thresholds to detect strong and weak edges and include the weak edges in output only if they are connected to strong edges this method is therefore less likely then the others to be fooled by noise and more likely to detect true weak edges .

Morphological operation

It includes erosion and dilation simultaneously

Erosion and Dilation

A dilation is the operation that corresponds to our intuitive concept of growing. The quasi-inverse of dilation is the erosion operation.

CROPPING

In cropping when we want to extract a rectangular portion of an image we use imcrop function.Using this we can specify the crop region.

Resizing

In resizing we use the imresize function when you resize an image you specify the image to be resized and the magnification factor .

CODE IN MATLAB

```
function varargout = im_proc(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @im_proc_OpeningFcn,
...
                   'gui_OutputFcn',  @im_proc_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end

function im_proc_OpeningFcn(hObject, eventdata,
handles, varargin)
```

```
handles.output =h.object;

guidata(hObject, handles);

function varargout = im_proc_OutputFcn(hObject,
eventdata, handles)

varargout{1} = handles.output;

function bro_Callback(hObject, eventdata, handles)

global path

[FileName,PathName] = uigetfile('*.jpg','Select the
Image file')

if FileName==0

errordlg('Select File properly')

path_name=strcat(PathName,FileName);

 path=path_name;

end

function popupmenu2_Callback(hObject, eventdata,
handles)

function popupmenu2_CreateFcn(hObject, eventdata,
handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))

   set(hObject,'BackgroundColor','white');

end

function pro_Callback(hObject, eventdata, handles)
```

```
global path

sel= get(handles.popupmenu2,'Value');
if isempty(path)
else
    imm1=imread(path);
    I=rgb2gray(imm1);
    axes(handles.axes1)
imshow(I);



if sel==2
   axes(handles.axes3)
   imhist(I);
else if sel==3

B = histeq(I, 64);
axes(handles.axes3)
imshow(B);
else if sel==4
bw = im2bw(I);
```

```
eroded = bwmorph(bw, 'erode');
dilated = bwmorph(eroded, 'dilate');
border = bwperim(dilated);
axes(handles.axes3)
imshow(border)

    else if sel ==5

eroded = imerode(I, ones(3));
axes(handles.axes3)
imshow(eroded)
        else if sel==6

dilated = bwmorph(I, 'dilate');
axes(handles.axes3)
imshow(dilated);
            else if sel==7
                axes(handles.axes1)
                imshow(imm1);
bright=input('Enter the value of bright(1-21) = ');
```

```
%bright gets its selected value in pop up menu or drop
down box

% bright =1 for -100%, bright =2 for -90% similarly
bright =21   % for 100%

 bright= ((bright)-11)*10

% calibrating bright accordingly, -100 for 100%, -90 for
90%

 image=imm1;

  %imread return data of image in matrix form


 new_image=image+ (bright);

% it sets new brightness features as accordingly for dark
black   % image (R,G,B)=(0,0,0) and for bright image
i.e. white

% (R,G,B)= (255,255,255)

else if sel== 8

    figure

new_image=imcrop(I);

axes(handles.axes3)

imshow(new_image);

                else if sel==9

mrows =input('Enter the value rows = ');

ncols=input('Enter the value cols = ');
```

```
B = imresize(I, [mrows ncols]);
axes(handles.axes3)
imshow(B);

else if sel==10
n=input('Enter the method(1=canny,2=sobel) ');
th=input('Enter the lower range, th = ');
if n==1
BW2 = edge(I,'canny',th);
axes(handles.axes3)
imshow(BW2);
else if n==2
axes(handles.axes3)
BW1=edge(I,'sobel',th);


imshow(BW1);
 else
 disp('wrong selection');
  end
  end
```

```
  else if sel==11



d=im2bw(I ,graythresh(I));
axes(handles.axes3)
 imshow(d);
  else


  end
  end
  end
  end
   end
   end
    end
    end
   end
end
          disp(' select option ');
   end
```

```
function exit_Callback(hObject, eventdata, handles)

clc

clear all

close all

function axes1_CreateFcn(hObject, eventdata, handles)
```

Screen shots of our GUI

Brightness

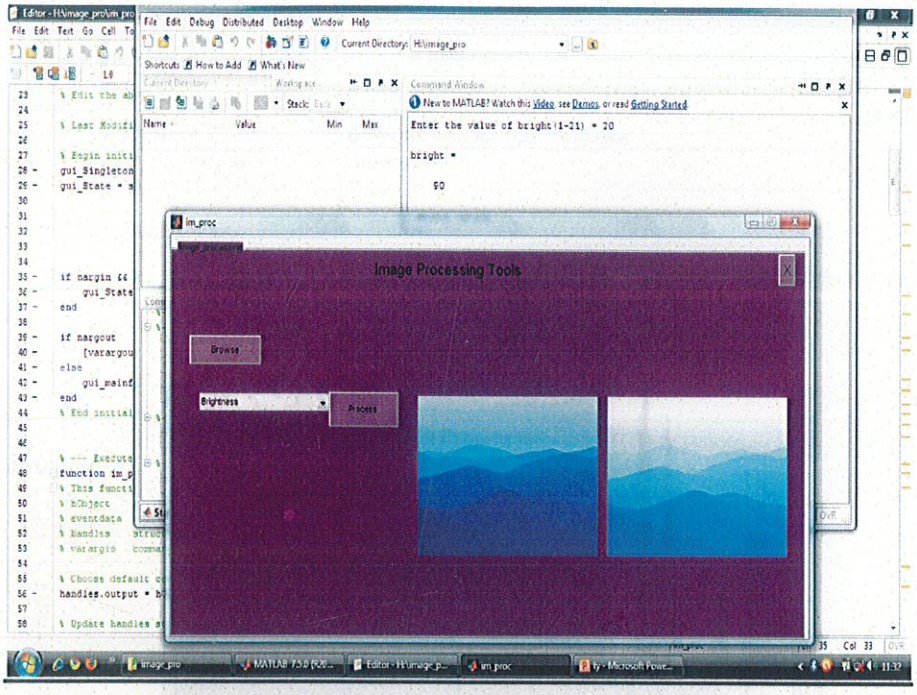
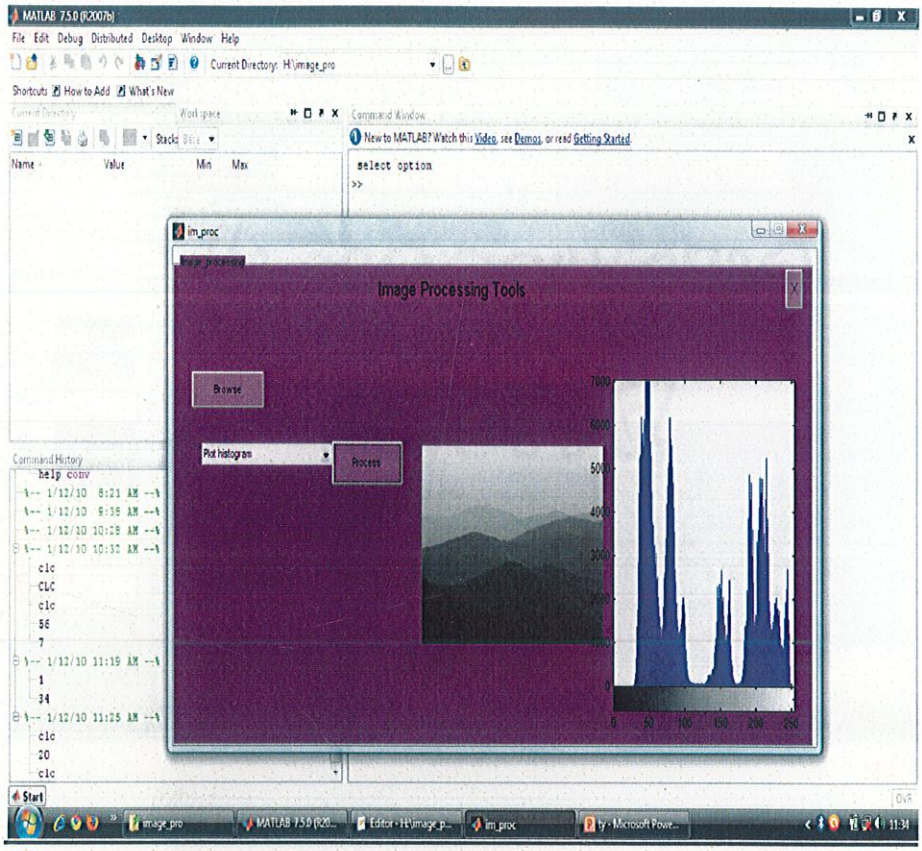
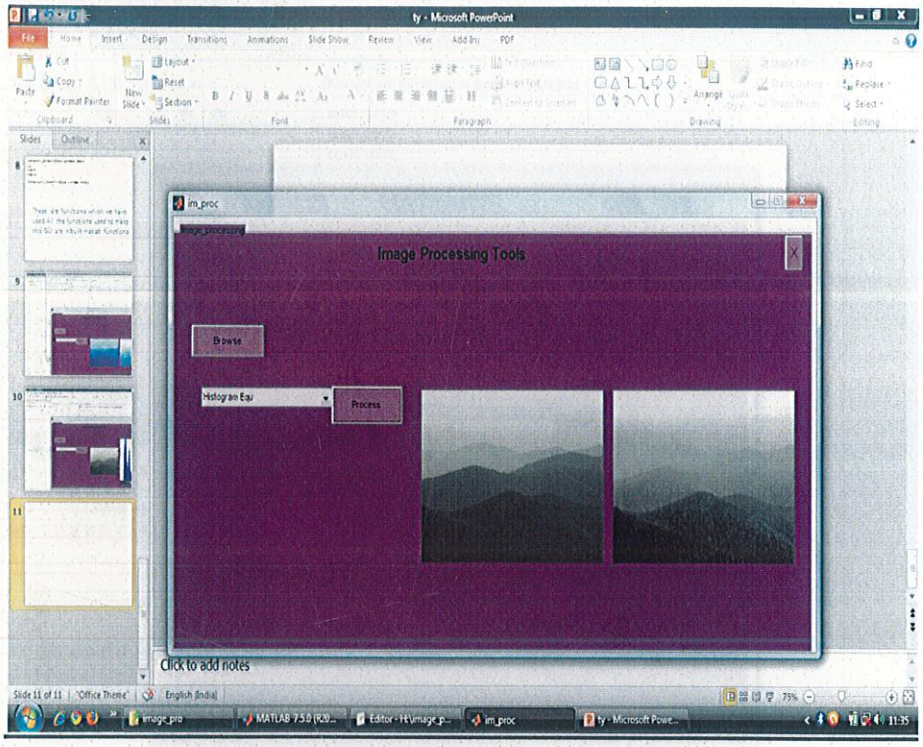


Image Processing Tools
Browse
Brightness
Process
Enter the value of bright (1-21) = 20
bright =
90

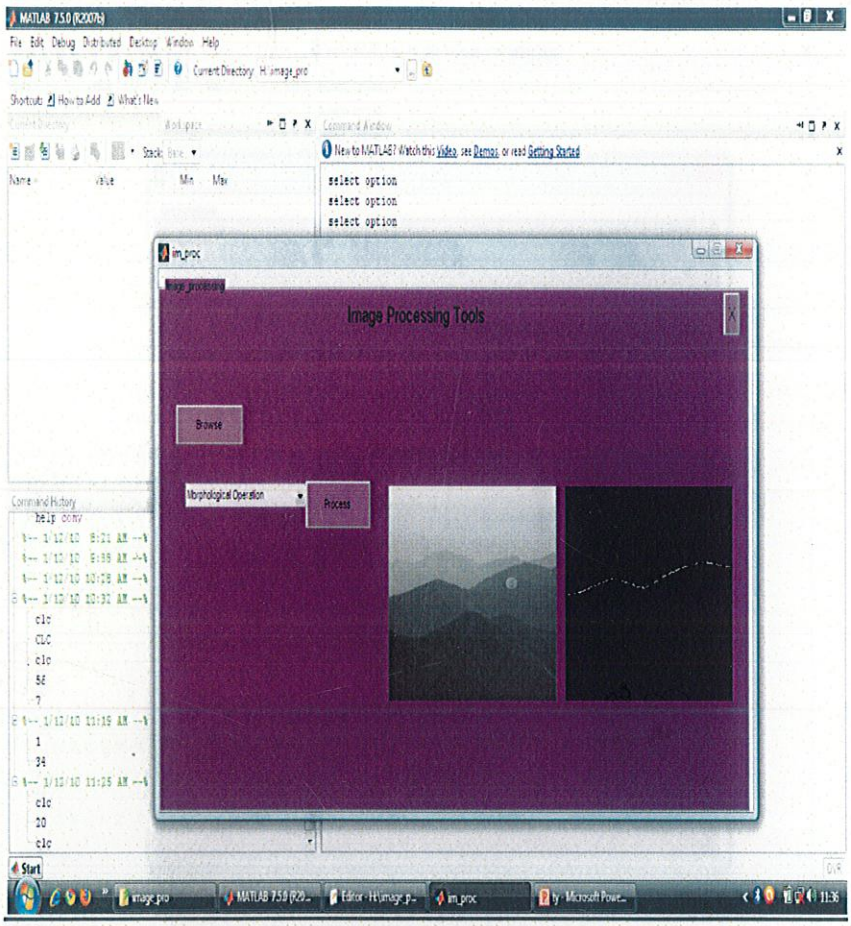
PLOT HISTOGRAM

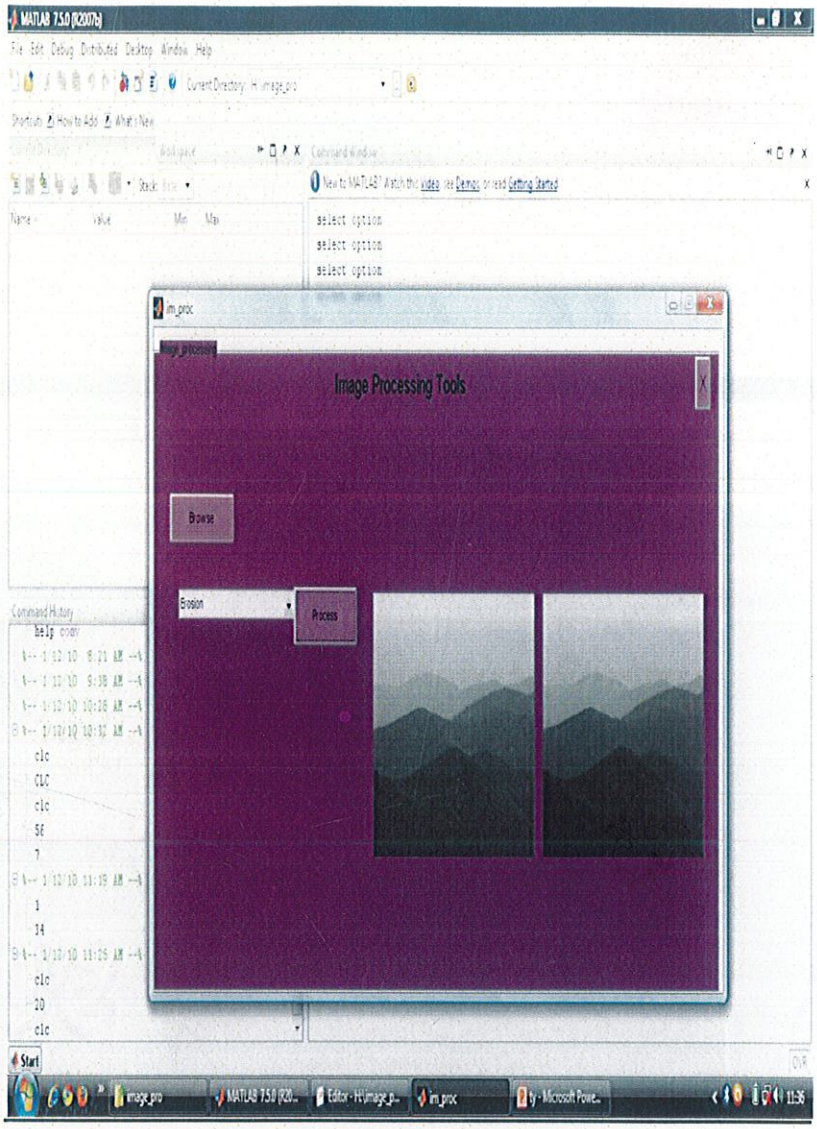


HISTOGRAM EQUALISATION

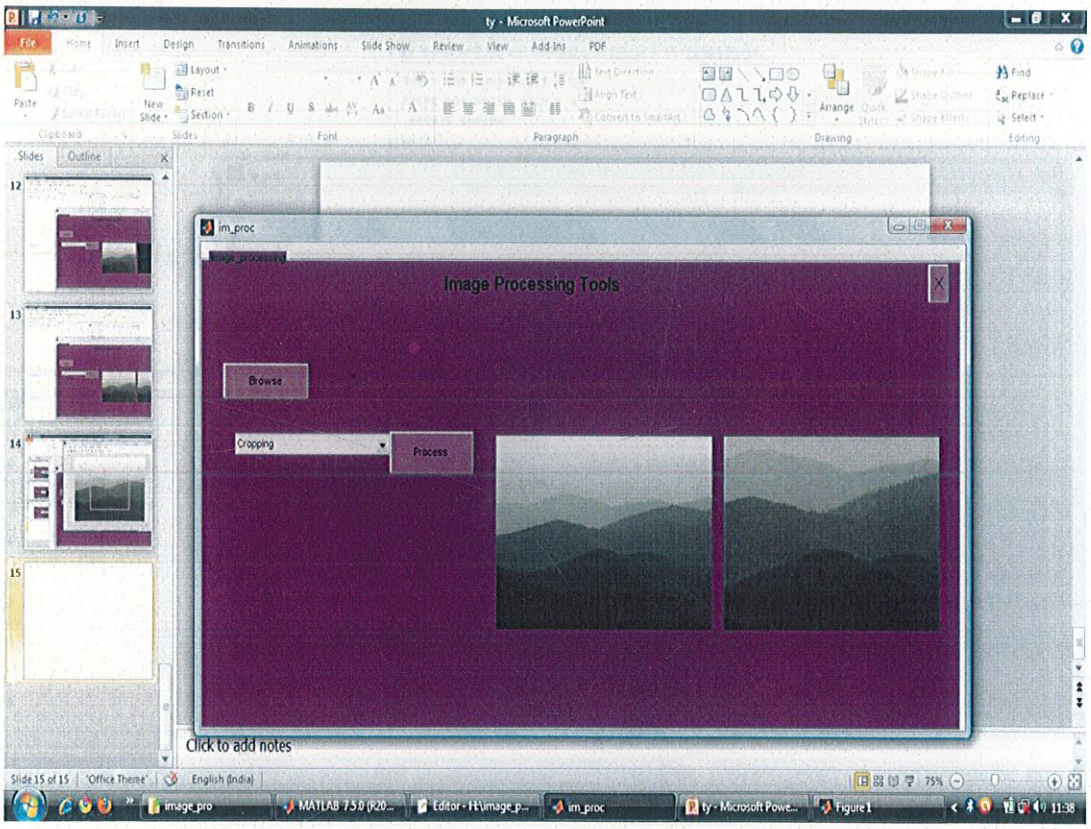


MORPHOLOGICAL OPERATION

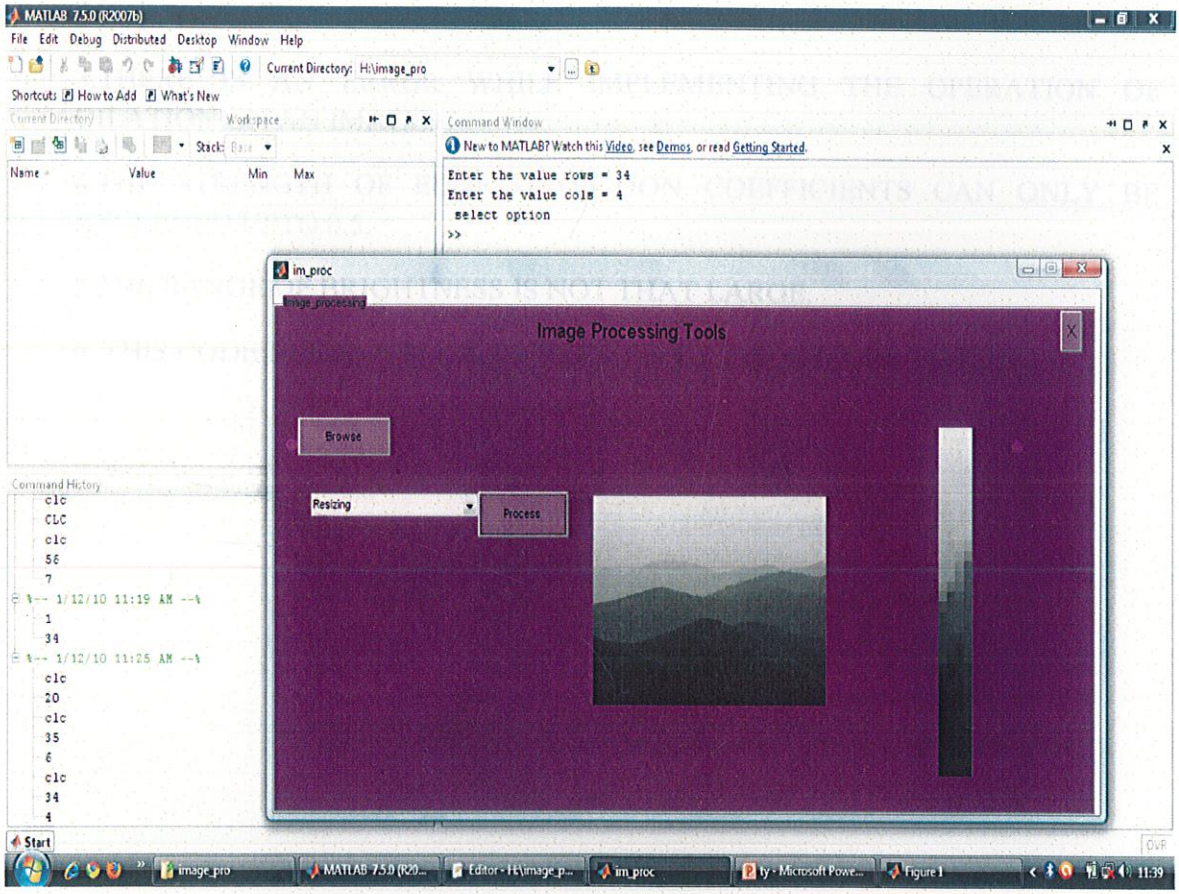



MATLAB 7.5.0 (R2007b)
File Edit Debug Distributed Desktop Window Help
Current Directory: H:\image_pro
Shortcuts How to Add What's New
Workspace
Command Window
New to MATLAB? Watch this video, see Demos, or read Getting Started
Name Value Min Max
select option
select option
select option
Command History
clc
CLC
clc
SE
7
1
34
clc
20
clc
im_proc
Image_processing
Image Processing Tools
Browse
Erosion
Process
Start
image_pro
MATLAB 7.5.0 (R20...
Editor - H:\image_p...
im_proc
ty - Microsoft Powe...
11:36

CROPPING


ty - Microsoft PowerPoint
im_proc
Image Processing Tools
Browse
Cropping
Process
Click to add notes

RESIZING


MATLAB 7.5.0 (R2007b)
File Edit Debug Distributed Desktop Window Help
Current Directory: H:\image_pro
Shortcuts How to Add What's New
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
Enter the value rows = 34
Enter the value cols = 4
select option
>>
im_proc
Image Processing Tools
Browse
Resizing
Process
Command History
clc
CLC
clc
56
7
%-- 1/12/10 11:19 AM --%
1
34
%-- 1/12/10 11:25 AM --%
clc
20
clc
35
6
clc
34
4
Start
image_pro
MATLAB 7.5.0 (R20...
Editor - H:\image_p...
im_proc
ty - Microsoft Powe...
Figure 1
11:39

LIMITATIONS

ON RUNNING THE CODE ON MATLAB WE ARE FACING THE FOLLOWING LIMITATIONS AS OF NOW

1.THROUGH THIS CODE WE CAN ONLY IMPORT COLOURED IMAGES(i.e. RGB IMAGE)

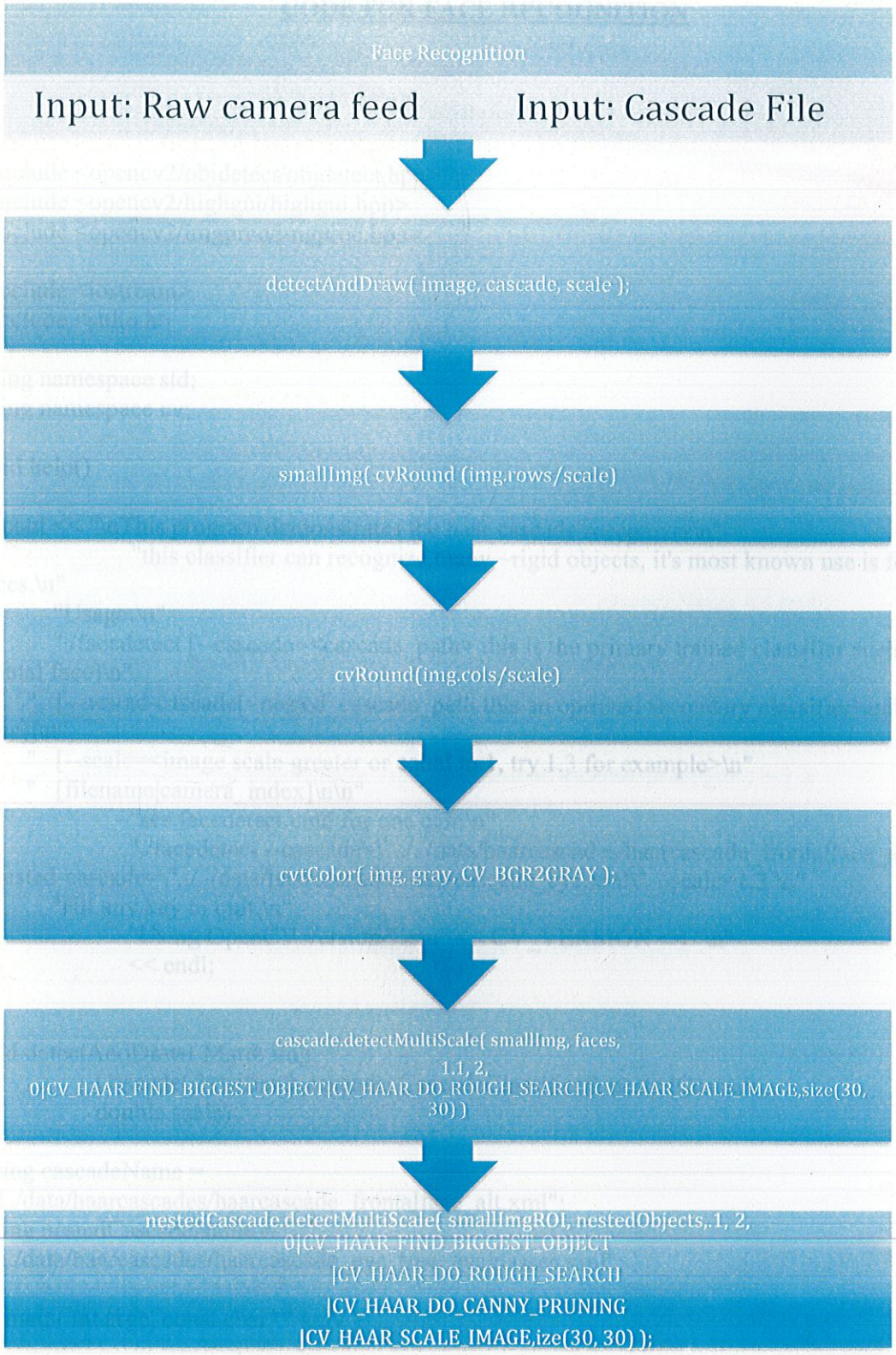
2.THIS PROGRAM TAKES IN AN COLOURED RGB IMAGE AND PROCESSES ON THE GRAY SCALE VERSION OF THE SAME.

3.THERE IS AN ERROR WHILE IMPLEMENTING THE OPERATION OF DILATION ON AN IMAGE.

4.THE STRENGTH OF EDGE DETECTION COEFFICIENTS CAN ONLY BE EXCEEDED UPTO 0.5.

5.THE RANGE OF BRIGHTNESS IS NOT THAT LARGE.

6. THIS CODE NEEDS A MATLAB VERSION OF 7.0 OR MORE TO RUN.


Face Recognition
Input: Raw camera feed
Input: Cascade File
detectAndDraw(image, cascade, scale);
smallImg(cvRound (img.rows/scale)
cvRound(img.cols/scale)
cvtColor(img, gray, CV_BGR2GRAY);
cascade.detectMultiScale(smallImg, faces,
1.1, 2,
0|CV_HAAR_FIND_BIGGEST_OBJECT|CV_HAAR_DO_ROUGH_SEARCH|CV_HAAR_SCALE_IMAGE,size(30,
30))
nestedCascade.detectMultiScale(smallImgROI, nestedObjects,.1, 2,
0|CV_HAAR_FIND_BIGGEST_OBJECT
|CV_HAAR_DO_ROUGH_SEARCH
|CV_HAAR_DO_CANNY_PRUNING
|CV_HAAR_SCALE_IMAGE,ize(30, 30));

CODE FOR FACE RECOGNITION

```
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

void help()
{
   cout << "\nThis program demonstrates the haar cascade recognizer\n"
                   "this classifier can recognize many ~rigid objects, it's most known use is for
faces.\n"
          "Usage:\n"
          "./facedetect [--cascade=<cascade_path> this is the primary trained classifier such as
frontal face]\n"
        "   [--nested-cascade[=nested_cascade_path this an optional secondary classifier such as
eyes]]\n"
        "   [--scale=<image scale greater or equal to 1, try 1.3 for example>\n"
        "   [filename|camera_index]\n\n"
                   "see facedetect.cmd for one call:\n"
                   "./facedetect --cascade=\"../../data/haarcascades/haarcascade_frontalface_alt.xml\"
--nested-cascade=\"../../data/haarcascades/haarcascade_eye.xml\" --scale=1.3 \n"
          "Hit any key to quit.\n"
                   "Using OpenCV version %s\n" << CV_VERSION << "\n"
                   << endl;
}

void detectAndDraw( Mat& img,
                CascadeClassifier& cascade, CascadeClassifier& nestedCascade,
                double scale);

String cascadeName =
"../../data/haarcascades/haarcascade_frontalface_alt.xml";
String nestedCascadeName =
"../../data/haarcascades/haarcascade_eye_tree_eyeglasses.xml";

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

```
    CvCapture* capture = 0;
    Mat frame, frameCopy, image;
    const String scaleOpt = "--scale=";
    size_t scaleOptLen = scaleOpt.length();
    const String cascadeOpt = "--cascade=";
    size_t cascadeOptLen = cascadeOpt.length();
    const String nestedCascadeOpt = "--nested-cascade";
    size_t nestedCascadeOptLen = nestedCascadeOpt.length();
    String inputName;

    help();
    CascadeClassifier cascade, nestedCascade;
    double scale = 1;

    for( int i = 1; i < argc; i++ )
    {
        cout << "Processing " << i << " " <<  argv[i] << endl;
      if( cascadeOpt.compare( 0, cascadeOptLen, argv[i], cascadeOptLen ) == 0 )
      {
        cascadeName.assign( argv[i] + cascadeOptLen );
        cout << "  from which we have cascadeName= " << cascadeName << endl;
      }
      else if( nestedCascadeOpt.compare( 0, nestedCascadeOptLen, argv[i],
nestedCascadeOptLen ) == 0 )
      {
        if( argv[i][nestedCascadeOpt.length()] == '=' )
          nestedCascadeName.assign( argv[i] + nestedCascadeOpt.length() + 1 );
        if( !nestedCascade.load( nestedCascadeName ) )
          cerr << "WARNING: Could not load classifier cascade for nested objects" << endl;
      }
      else if( scaleOpt.compare( 0, scaleOptLen, argv[i], scaleOptLen ) == 0 )
      {
        if( !sscanf( argv[i] + scaleOpt.length(), "%lf", &scale ) || scale < 1 )
          scale = 1;
        cout << " from which we read scale = " << scale << endl;
      }
      else if( argv[i][0] == '-' )
      {
        cerr << "WARNING: Unknown option %s" << argv[i] << endl;
      }
      else
        inputName.assign( argv[i] );
    }

    if( !cascade.load( cascadeName ) )
    {
```

```
    cerr << "ERROR: Could not load classifier cascade" << endl;
    cerr << "Usage: facedetect [--cascade=<cascade_path>]\n"
        "   [--nested-cascade[=nested_cascade_path]]\n"
        "   [--scale[=<image scale>\n"
        "   [filename|camera_index]\n" << endl ;
    return -1;
}

if( inputName.empty() || (isdigit(inputName.c_str()[0]) && inputName.c_str()[1] == '\0') )
{
    capture = cvCaptureFromCAM( inputName.empty() ? 0 : inputName.c_str()[0] - '0' );
    int c = inputName.empty() ? 0 : inputName.c_str()[0] - '0' ;
    if(!capture) cout << "Capture from CAM " <<  c << " didn't work" << endl;
}
else if( inputName.size() )
{
    image = imread( inputName, 1 );
    if( image.empty() )
    {
        capture = cvCaptureFromAVI( inputName.c_str() );
        if(!capture) cout << "Capture from AVI didn't work" << endl;
    }
}
else
{
    image = imread( "lena.jpg", 1 );
    if(image.empty()) cout << "Couldn't read lena.jpg" << endl;
}

cvNamedWindow( "result", 1 );

if( capture )
{
      cout << "In capture ..." << endl;
    for(;;)
    {
        IplImage* iplImg = cvQueryFrame( capture );
        frame = iplImg;
        if( frame.empty() )
            break;
        if( iplImg->origin == IPL_ORIGIN_TL )
            frame.copyTo( frameCopy );
        else
            flip( frame, frameCopy, 0 );

        detectAndDraw( frameCopy, cascade, nestedCascade, scale );
```

```
            if( waitKey( 10 ) >= 0 )
                goto _cleanup_;
        }

        waitKey(0);
_cleanup_:
        cvReleaseCapture( &capture );
    }
    else
    {
        cout << "In image read" << endl;
        if( !image.empty() )
        {
            detectAndDraw( image, cascade, nestedCascade, scale );
            waitKey(0);
        }
        else if( !inputName.empty() )
        {
            /* assume it is a text file containing the
            list of the image filenames to be processed - one per line */
            FILE* f = fopen( inputName.c_str(), "rt" );
            if( f )
            {
                char buf[1000+1];
                while( fgets( buf, 1000, f ) )
                {
                    int len = (int)strlen(buf), c;
                    while( len > 0 && isspace(buf[len-1]) )
                        len--;
                    buf[len] = '\0';
                    cout << "file " << buf << endl;
                    image = imread( buf, 1 );
                    if( !image.empty() )
                    {
                        detectAndDraw( image, cascade, nestedCascade, scale );
                        c = waitKey(0);
                        if( c == 27 || c == 'q' || c == 'Q' )
                            break;
                    }
                    else
                    {
                        cerr << "Aw snap, couldn't read image " << buf << endl;
                    }
                }
                fclose(f);
```

```
        }
      }
    }

    cvDestroyWindow("result");

    return 0;
}

void detectAndDraw( Mat& img,
                    CascadeClassifier& cascade, CascadeClassifier& nestedCascade,
                    double scale)
{
    int i = 0;
    double t = 0;
    vector<Rect> faces;
    const static Scalar colors[] =  { CV_RGB(0,0,255),
        CV_RGB(0,128,255),
        CV_RGB(0,255,255),
        CV_RGB(0,255,0),
        CV_RGB(255,128,0),
        CV_RGB(255,255,0),
        CV_RGB(255,0,0),
        CV_RGB(255,0,255)} ;
    Mat gray, smallImg( cvRound (img.rows/scale), cvRound(img.cols/scale), CV_8UC1 );

    cvtColor( img, gray, CV_BGR2GRAY );
    resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
    equalizeHist( smallImg, smallImg );

    t = (double)cvGetTickCount();
    cascade.detectMultiScale( smallImg, faces,
        1.1, 2, 0
        //|CV_HAAR_FIND_BIGGEST_OBJECT
        //|CV_HAAR_DO_ROUGH_SEARCH
        |CV_HAAR_SCALE_IMAGE
        ,
        Size(30, 30) );
    t = (double)cvGetTickCount() - t;
    printf( "detection time = %g ms\n", t/((double)cvGetTickFrequency()*1000.) );
    for( vector<Rect>::const_iterator r = faces.begin(); r != faces.end(); r++, i++ )
    {
        Mat smallImgROI;
        vector<Rect> nestedObjects;
        Point center;
        Scalar color = colors[i%8];
```

```
        int radius;
        center.x = cvRound((r->x + r->width*0.5)*scale);
        center.y = cvRound((r->y + r->height*0.5)*scale);
        radius = cvRound((r->width + r->height)*0.25*scale);
        circle( img, center, radius, color, 3, 8, 0 );
        if( nestedCascade.empty() )
            continue;
        smallImgROI = smallImg(*r);
        nestedCascade.detectMultiScale( smallImgROI, nestedObjects,
            1.1, 2, 0
            //|CV_HAAR_FIND_BIGGEST_OBJECT
            //|CV_HAAR_DO_ROUGH_SEARCH
            //|CV_HAAR_DO_CANNY_PRUNING
            |CV_HAAR_SCALE_IMAGE
            ,
            Size(30, 30) );
        for( vector<Rect>::const_iterator nr = nestedObjects.begin(); nr != nestedObjects.end();
nr++ )
        {
            center.x = cvRound((r->x + nr->x + nr->width*0.5)*scale);
            center.y = cvRound((r->y + nr->y + nr->height*0.5)*scale);
            radius = cvRound((nr->width + nr->height)*0.25*scale);
            circle( img, center, radius, color, 3, 8, 0 );
        }
    }
    cv::imshow( "result", img );
}
```

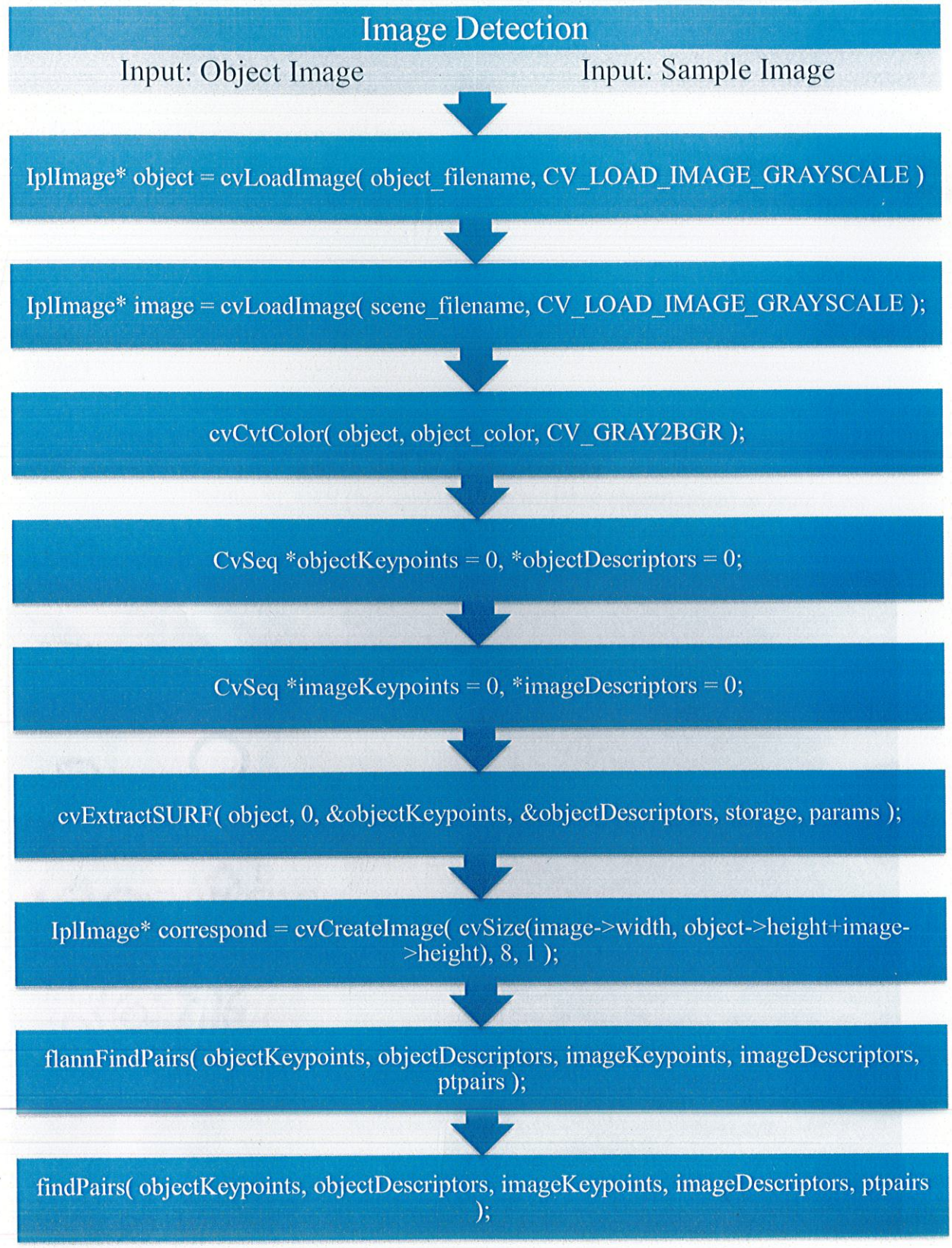
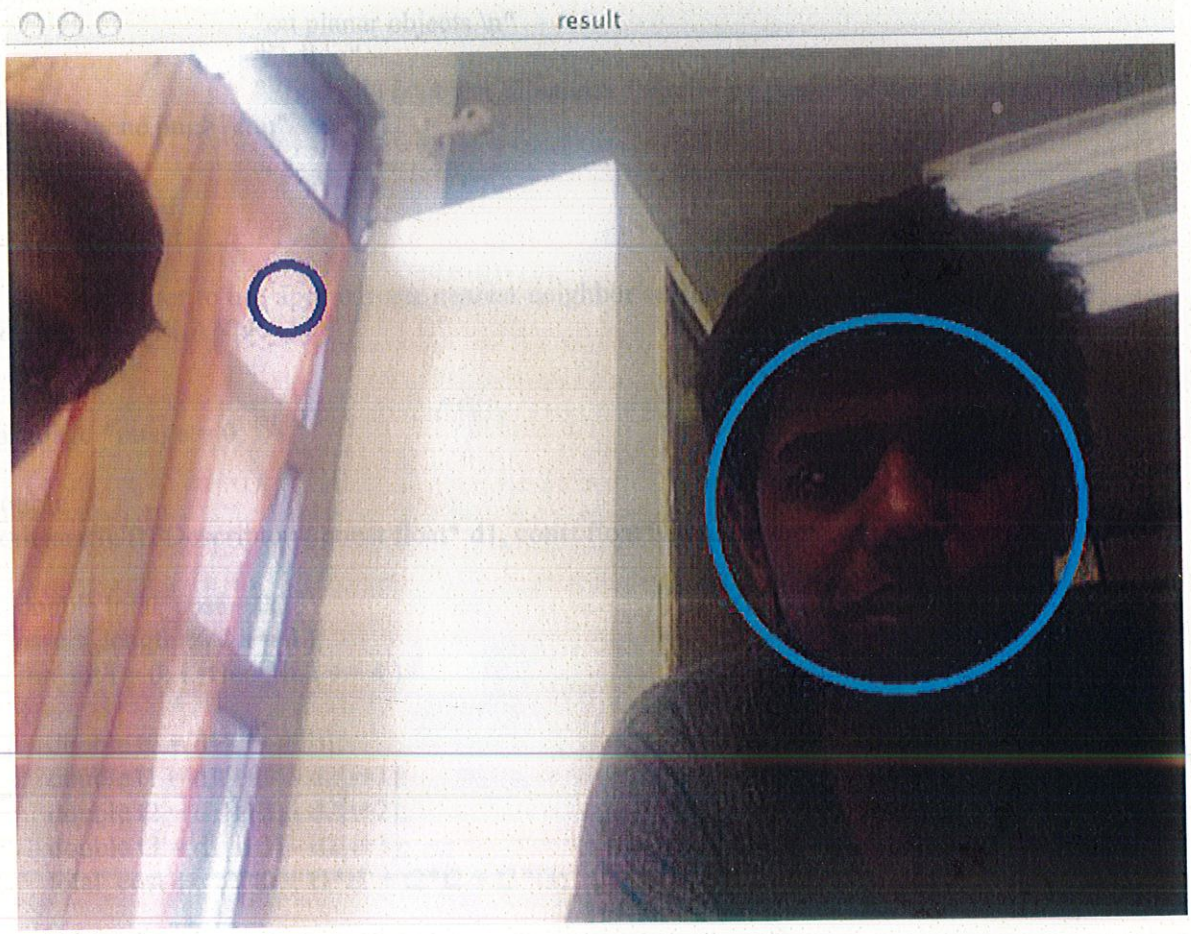


Image Detection
Input: Object Image
Input: Sample Image
IplImage* object = cvLoadImage(object_filename, CV_LOAD_IMAGE_GRAYSCALE)
IplImage* image = cvLoadImage(scene_filename, CV_LOAD_IMAGE_GRAYSCALE);
cvCvtColor(object, object_color, CV_GRAY2BGR);
CvSeq *objectKeypoints = 0, *objectDescriptors = 0;
CvSeq *imageKeypoints = 0, *imageDescriptors = 0;
cvExtractSURF(object, 0, &objectKeypoints, &objectDescriptors, storage, params);
IplImage* correspond = cvCreateImage(cvSize(image->width, object->height+image->height), 8, 1);
flannFindPairs(objectKeypoints, objectDescriptors, imageKeypoints, imageDescriptors, ptpairs);
findPairs(objectKeypoints, objectDescriptors, imageKeypoints, imageDescriptors, ptpairs);

Screen shot for image detection



CODE FOR OBJECT DETECTION

```
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <opencv2/imgproc/imgproc_c.h>

#include <iostream>
#include <vector>

using namespace std;
void help()
{
        printf(
                        "This program demonstrated the use of the SURF Detector and Descriptor
using\n"
                        "either FLANN (fast approx nearst neighbor classification) or brute force
matching\n"
                        "on planar objects.\n"
                        "Call:\n"
                        "./find_obj [<object_filename default box.png> <scene_filename default
box_in_scene.png>]\n\n"
                        );

}

// define whether to use approximate nearest-neighbor search
#define USE_FLANN


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);
        }
    }
}


void
flannFindPairs( const CvSeq*, const CvSeq* objectDescriptors,
        const CvSeq*, const CvSeq* imageDescriptors, vector<int>& ptpairs )
{
        int length = (int)(objectDescriptors->elem_size/sizeof(float));

    cv::Mat m_object(objectDescriptors->total, length, CV_32F);
        cv::Mat m_image(imageDescriptors->total, length, CV_32F);


        // copy descriptors
    CvSeqReader obj_reader;
        float* obj_ptr = m_object.ptr<float>(0);
    cvStartReadSeq( objectDescriptors, &obj_reader );
    for(int i = 0; i < objectDescriptors->total; i++ )
    {
        const float* descriptor = (const float*)obj_reader.ptr;
        CV_NEXT_SEQ_ELEM( obj_reader.seq->elem_size, obj_reader );
        memcpy(obj_ptr, descriptor, length*sizeof(float));
        obj_ptr += length;
    }
    CvSeqReader img_reader;
        float* img_ptr = m_image.ptr<float>(0);
```

```
    cvStartReadSeq( imageDescriptors, &img_reader );
    for(int i = 0; i < imageDescriptors->total; i++ )
    {
        const float* descriptor = (const float*)img_reader.ptr;
        CV_NEXT_SEQ_ELEM( img_reader.seq->elem_size, img_reader );
        memcpy(img_ptr, descriptor, length*sizeof(float));
        img_ptr += length;
    }

    // find nearest neighbors using FLANN
    cv::Mat m_indices(objectDescriptors->total, 2, CV_32S);
    cv::Mat m_dists(objectDescriptors->total, 2, CV_32F);
    cv::flann::Index flann_index(m_image, cv::flann::KDTreeIndexParams(4));  // using 4
randomized kdtrees
    flann_index.knnSearch(m_object, m_indices, m_dists, 2, cv::flann::SearchParams(64) ); //
maximum number of leafs checked

    int* indices_ptr = m_indices.ptr<int>(0);
    float* dists_ptr = m_dists.ptr<float>(0);
    for (int i=0;i<m_indices.rows;++i) {
        if (dists_ptr[2*i]<0.6*dists_ptr[2*i+1]) {
            ptpairs.push_back(i);
            ptpairs.push_back(indices_ptr[2*i]);
        }
    }
}


/* 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;

#ifdef USE_FLANN
    flannFindPairs( objectKeypoints, objectDescriptors, imageKeypoints, imageDescriptors,
ptpairs );
#else
    findPairs( objectKeypoints, objectDescriptors, imageKeypoints, imageDescriptors, ptpairs );
```

```
#endif

    n = (int)(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);
    help();
    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 );

#ifdef USE_FLANN
    printf("Using approximate nearest neighbor search\n");
```

```
#endif

    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;
#ifdef USE_FLANN
    flannFindPairs( objectKeypoints, objectDescriptors, imageKeypoints, imageDescriptors,
ptpairs );
#else
    findPairs( objectKeypoints, objectDescriptors, imageKeypoints, imageDescriptors, ptpairs );
#endif
    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;
}
```

Flow chart for motion tracking

SCREEN SHOT FOR MOTION TRACKING

CODE FOR MOTION TRACKING

```
#include "opencv2/video/tracking.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <time.h>
#include <stdio.h>

void help()
{
        printf(
                        "\nThis program demonstrated the use of motion templates -- basically
using the gradients\n"
                        "of thresholded layers of decaying frame differencing. New movements
are stamped on top with floating system\n"
                        "time code and motions too old are thresholded away. This is the 'motion
history file'. The program reads from the camera of your choice or from\n"
                        "a file. Gradients of motion history are used to detect direction of motoin
etc\n"
                        "Call:\n"
                        "./motempl [camera number 0-n or file name, default is camera 0]\n"
                        );
}
// various tracking parameters (in seconds)
const double MHI_DURATION = 1;
const double MAX_TIME_DELTA = 0.5;
const double MIN_TIME_DELTA = 0.05;
// number of cyclic frame buffer used for motion detection
// (should, probably, depend on FPS)
const int N = 4;

// ring image buffer
IplImage **buf = 0;
int last = 0;

// temporary images
IplImage *mhi = 0; // MHI
IplImage *orient = 0; // orientation
IplImage *mask = 0; // valid orientation mask
IplImage *segmask = 0; // motion segmentation map
CvMemStorage* storage = 0; // temporary storage

// parameters:
//  img - input video frame
//  dst - resultant motion picture
```

```
// args - optional parameters
void  update_mhi( IplImage* img, IplImage* dst, int diff_threshold )
{
   double timestamp = (double)clock()/CLOCKS_PER_SEC; // get current time in seconds
   CvSize size = cvSize(img->width,img->height); // get current frame size
   int i, idx1 = last, idx2;
   IplImage* silh;
   CvSeq* seq;
   CvRect comp_rect;
   double count;
   double angle;
   CvPoint center;
   double magnitude;
   CvScalar color;

   // allocate images at the beginning or
   // reallocate them if the frame size is changed
   if( !mhi || mhi->width != size.width || mhi->height != size.height ) {
      if( buf == 0 ) {
         buf = (IplImage**)malloc(N*sizeof(buf[0]));
         memset( buf, 0, N*sizeof(buf[0]));
      }

      for( i = 0; i < N; i++ ) {
         cvReleaseImage( &buf[i] );
         buf[i] = cvCreateImage( size, IPL_DEPTH_8U, 1 );
         cvZero( buf[i] );
      }
      cvReleaseImage( &mhi );
      cvReleaseImage( &orient );
      cvReleaseImage( &segmask );
      cvReleaseImage( &mask );

      mhi = cvCreateImage( size, IPL_DEPTH_32F, 1 );
      cvZero( mhi ); // clear MHI at the beginning
      orient = cvCreateImage( size, IPL_DEPTH_32F, 1 );
      segmask = cvCreateImage( size, IPL_DEPTH_32F, 1 );
      mask = cvCreateImage( size, IPL_DEPTH_8U, 1 );
   }

   cvCvtColor( img, buf[last], CV_BGR2GRAY ); // convert frame to grayscale

   idx2 = (last + 1) % N; // index of (last - (N-1))th frame
   last = idx2;

   silh = buf[idx2];
```

```
cvAbsDiff( buf[idx1], buf[idx2], silh ); // get difference between frames

cvThreshold( silh, silh, diff_threshold, 1, CV_THRESH_BINARY ); // and threshold it
cvUpdateMotionHistory( silh, mhi, timestamp, MHI_DURATION ); // update MHI

// convert MHI to blue 8u image
cvCvtScale( mhi, mask, 255./MHI_DURATION,
        (MHI_DURATION - timestamp)*255./MHI_DURATION );
cvZero( dst );
cvMerge( mask, 0, 0, 0, dst );

// calculate motion gradient orientation and valid orientation mask
cvCalcMotionGradient( mhi, mask, orient, MAX_TIME_DELTA, MIN_TIME_DELTA, 3 );

if( !storage )
   storage = cvCreateMemStorage(0);
else
   cvClearMemStorage(storage);

// segment motion: get sequence of motion components
// segmask is marked motion components map. It is not used further
seq = cvSegmentMotion( mhi, segmask, storage, timestamp, MAX_TIME_DELTA );

// iterate through the motion components,
// One more iteration (i == -1) corresponds to the whole image (global motion)
for( i = -1; i < seq->total; i++ ) {

   if( i < 0 ) { // case of the whole image
      comp_rect = cvRect( 0, 0, size.width, size.height );
      color = CV_RGB(255,255,255);
      magnitude = 100;
   }
   else { // i-th motion component
      comp_rect = ((CvConnectedComp*)cvGetSeqElem( seq, i ))->rect;
      if( comp_rect.width + comp_rect.height < 100 ) // reject very small components
         continue;
      color = CV_RGB(255,0,0);
      magnitude = 30;
   }

   // select component ROI
   cvSetImageROI( silh, comp_rect );
   cvSetImageROI( mhi, comp_rect );
   cvSetImageROI( orient, comp_rect );
   cvSetImageROI( mask, comp_rect );
```

```
        // calculate orientation
        angle = cvCalcGlobalOrientation( orient, mask, mhi, timestamp, MHI_DURATION);
        angle = 360.0 - angle;  // adjust for images with top-left origin

        count = cvNorm( silh, 0, CV_L1, 0 ); // calculate number of points within silhouette ROI

        cvResetImageROI( mhi );
        cvResetImageROI( orient );
        cvResetImageROI( mask );
        cvResetImageROI( silh );

        // check for the case of little motion
        if( count < comp_rect.width*comp_rect.height * 0.05 )
          continue;

        // draw a clock with arrow indicating the direction
        center = cvPoint( (comp_rect.x + comp_rect.width/2),
                          (comp_rect.y + comp_rect.height/2) );

        cvCircle( dst, center, cvRound(magnitude*1.2), color, 3, CV_AA, 0 );
        cvLine( dst, center, cvPoint( cvRound( center.x + magnitude*cos(angle*CV_PI/180)),
              cvRound( center.y - magnitude*sin(angle*CV_PI/180))), color, 3, CV_AA, 0 );
    }
}


int main(int argc, char** argv)
{
    IplImage* motion = 0;
    CvCapture* capture = 0;
    help();
    if( argc == 1 || (argc == 2 && strlen(argv[1]) == 1 && isdigit(argv[1][0])))
        capture = cvCaptureFromCAM( argc == 2 ? argv[1][0] - '0' : 0 );
    else if( argc == 2 )
        capture = cvCaptureFromFile( argv[1] );

    if( capture )
    {
        cvNamedWindow( "Motion", 1 );

        for(;;)
        {
            IplImage* image = cvQueryFrame( capture );
            if( !image )
                break;
```

```
            if( !motion )
            {
               motion = cvCreateImage( cvSize(image->width,image->height), 8, 3 );
               cvZero( motion );
               motion->origin = image->origin;
            }

            update_mhi( image, motion, 30 );
            cvShowImage( "Motion", motion );

            if( cvWaitKey(10) >= 0 )
               break;
         }
         cvReleaseCapture( &capture );
         cvDestroyWindow( "Motion" );
      }

      return 0;
}

#ifdef _EiC
main(1,"motempl.c");
#endif
```