

**FACULTY OF INFORMATION TECHNOLOGIES**

**DEPARTMENT OF INFORMATION SYSTEMS MANAGEMENT**

**MACHINE VISION**

*Report*

**Laboratory work #4**

|  |  |
| --- | --- |
|  | **Done by: Sailaubekova Zhansaya** |
|  | **Checked by: Auezova Anel** |

**Almaty 2015**

//

// main.cpp

// opencv2

//

// Created by Zhansaya on 22.09.15.

// Copyright (c) 2015 Zhansaya. All rights reserved.

//

#include "opencv2/opencv.hpp"

#include <string>

using namespace cv;

using namespace std;

class WatershedSegmenter{

private:

cv::Mat markers;

public:

void setMarkers(cv::Mat& markerImage)

{

markerImage.convertTo(markers, CV\_32S);

}

cv::Mat process(cv::Mat &image)

{

cv::watershed(image, markers);

markers.convertTo(markers,CV\_8U);

return markers;

}

};

int main(int argc, char\* argv[])

{

cv::Mat image = cv::imread("/Users/Zhansaya/Desktop/fruits.jpg");

cv::Mat binary;// = cv::imread(argv[2], 0);

cv::cvtColor(image, binary, COLOR\_BGR2GRAY);

cv::threshold(binary, binary, 100, 255, THRESH\_BINARY);

imshow("originalimage", image);

imshow("originalbinary", binary);

// Eliminate noise and smaller objects

cv::Mat fg;

cv::erode(binary,fg,cv::Mat(),cv::Point(-1,-1),2);

imshow("fg", fg);

// Identify image pixels without objects

cv::Mat bg;

cv::dilate(binary,bg,cv::Mat(),cv::Point(-1,-1),3);

cv::threshold(bg,bg,1, 128,cv::THRESH\_BINARY\_INV);

imshow("bg", bg);

// Create markers image

cv::Mat markers(binary.size(),CV\_8U,cv::Scalar(0));

markers= fg+bg;

imshow("markers", markers);

// Create watershed segmentation object

WatershedSegmenter segmenter;

segmenter.setMarkers(markers);

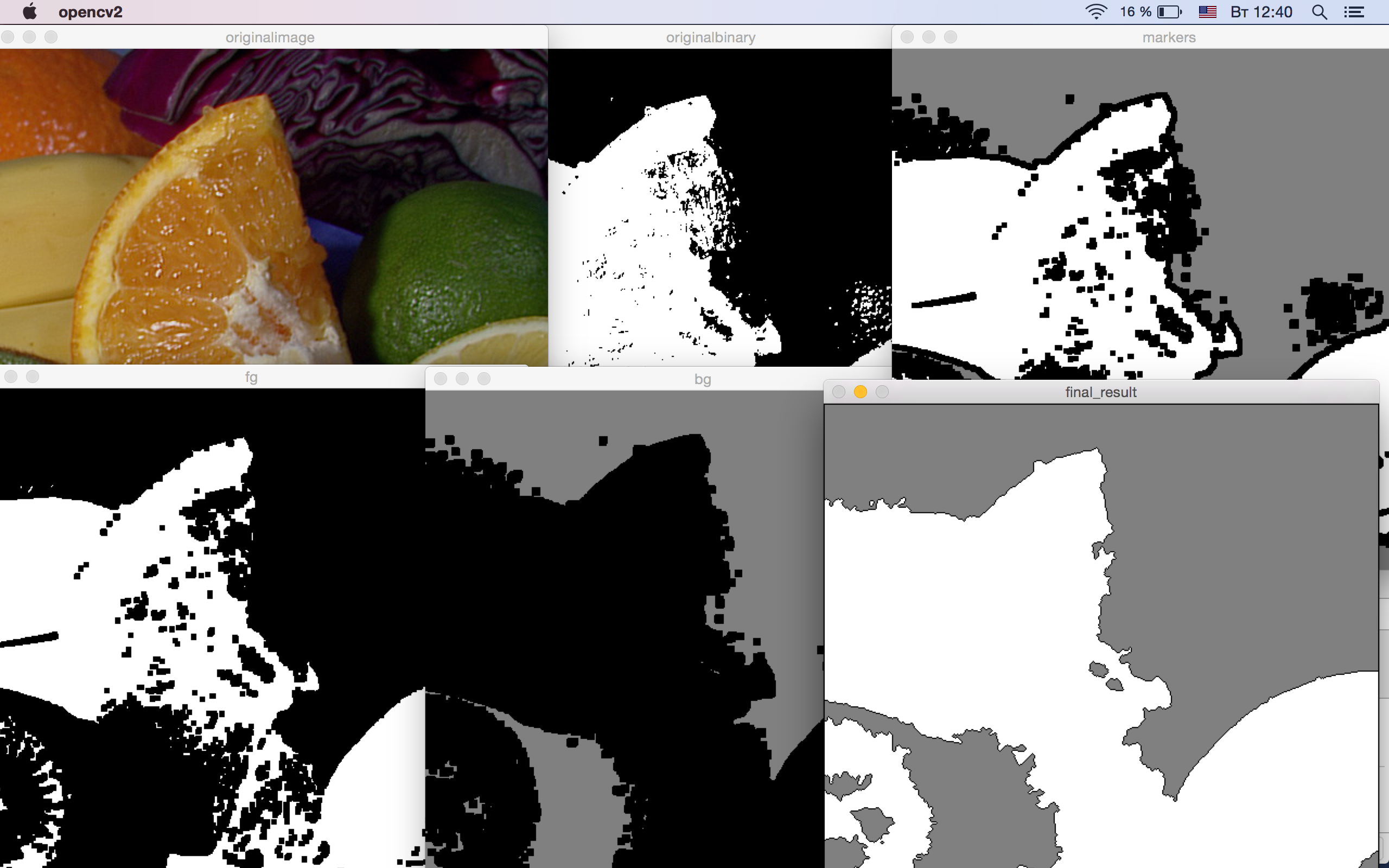
cv::Mat result = segmenter.process(image);

result.convertTo(result,CV\_8U);

imshow("final\_result", result);

cv::waitKey(0);

return 0;

}

#include "opencv2/opencv.hpp"

#include <iostream>

using namespace cv;

using namespace std;

int main( )

{

Mat image;

image= cv::imread("/Users/Zhansaya/Desktop/fruits.jpg");

// define bounding rectangle

cv::Rect rectangle(10,100,380,180);

cv::Mat result; // segmentation (4 possible values)

cv::Mat bgModel,fgModel; // the models (internally used)

// GrabCut segmentation

cv::grabCut(image, // input image

result, // segmentation result

rectangle, // rectangle containing foreground

bgModel,fgModel, // models

5, // number of iterations

cv::GC\_INIT\_WITH\_RECT); // use rectangle

// Get the pixels marked as likely foreground

cv::compare(result,cv::GC\_PR\_FGD,result,cv::CMP\_EQ);

// Generate output image

cv::Mat foreground(image.size(),CV\_8UC3,

cv::Scalar(255,255,255));

image.copyTo(foreground,// bg pixels are not copied

result);

// checking first bit with bitwise-and

result= result&1; // will be 1 if FG

// draw rectangle on original image

cv::rectangle(image, rectangle, cv::Scalar(255,255,255),5);

cv::namedWindow("Image");

cv::imshow("Image",image);

// display result

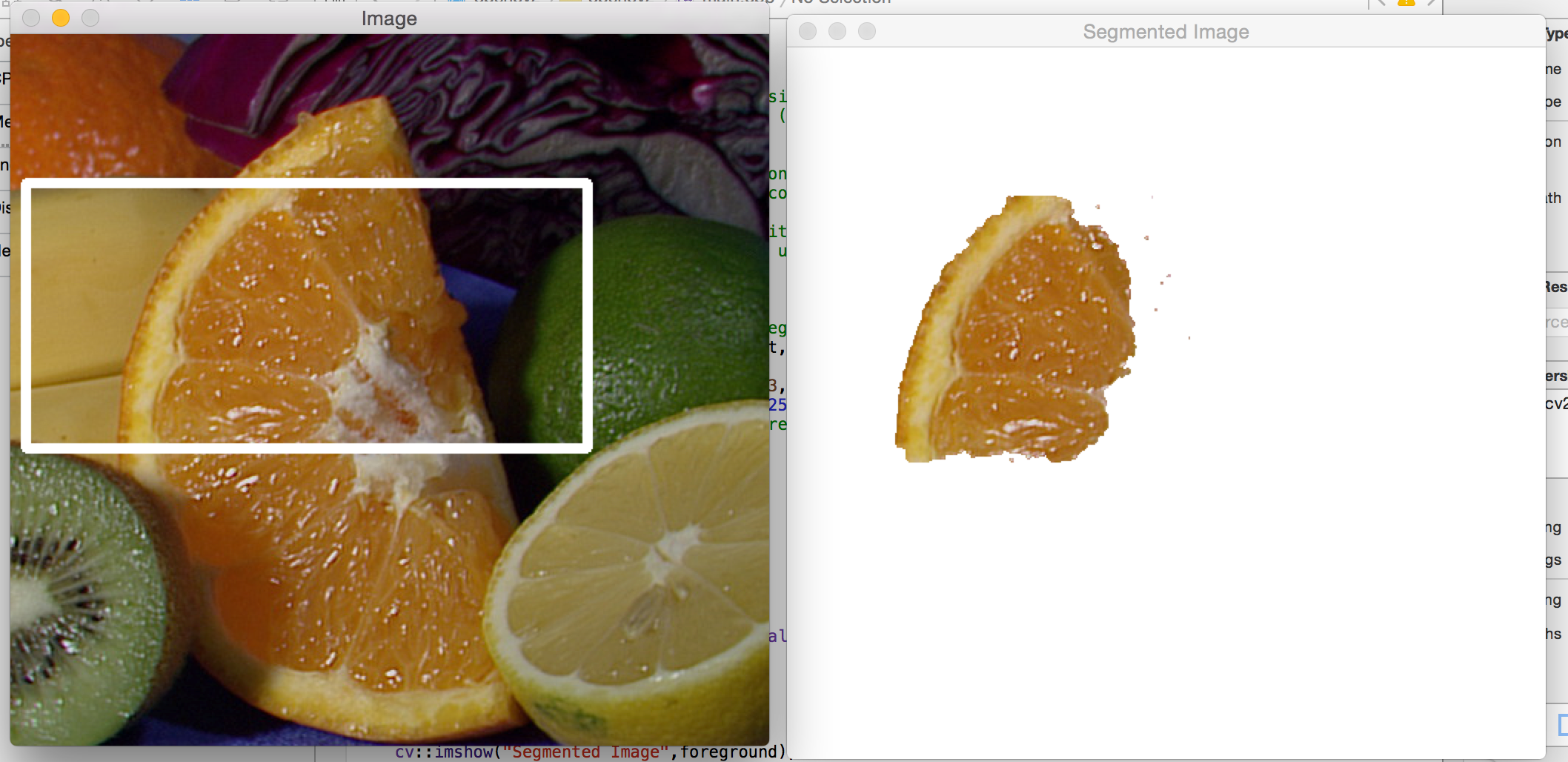
cv::namedWindow("Segmented Image");

cv::imshow("Segmented Image",foreground);

waitKey();

return 0;

}



#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

using namespace cv;

using namespace std;

Mat src; Mat hsv; Mat hue;

int bins = 25;

void Hist\_and\_Backproj(int, void\* )

{

MatND hist;

int histSize = MAX( bins, 2 );

float hue\_range[] = { 0, 180 };

const float\* ranges = { hue\_range };

calcHist( &hue, 1, 0, Mat(), hist, 1, &histSize, &ranges, true, false );

normalize( hist, hist, 0, 255, NORM\_MINMAX, -1, Mat() );

MatND backproj;

calcBackProject( &hue, 1, 0, hist, backproj, &ranges, 1, true );

imshow( "BackProj", backproj );

}

int main(int argc, char\* argv[])

{

src = imread("/Users/Zhansaya/Desktop/fruits.jpg");

cvtColor( src, hsv, COLOR\_BGR2HSV );

hue.create( hsv.size(), hsv.depth() );

int ch[] = { 0, 0 };

mixChannels( &hsv, 1, &hue, 1, ch, 1 );

string window\_image = "Source image";

namedWindow( window\_image, WINDOW\_AUTOSIZE );

Hist\_and\_Backproj(0, 0);

imshow( window\_image, src );

waitKey(0);

return 0;

}

