----------resim okuma yazma-----------

int main()

{

Mat Orgim = imread("lena.png");

Mat Grayim;

cvtColor(Orgim, Grayim, COLOR\_BGR2GRAY);

imshow("Orgim", Orgim);

imshow("Grayim", Grayim);

waitKey(0);

return(0);

}

-----------resim boyutlarını yazma-----

int main()

{

Mat resim = imread("lena.png");

int satir=resim.rows;

int sutun = resim.cols;

cout << "satir:" << satir << " ";

cout << "sutun:" << sutun << " ";

namedWindow("resimismi", WINDOW\_AUTOSIZE);

imshow("resimismi", resim);

waitKey(0);

destroyWindow("resimismi");

return(0);

}

--------resimi yeniden boyutlandırma----

int main()

{

Mat resim = imread("lena.png");

Mat newsize;

resize(resim, newsize, Size(resim.rows/2, resim.cols/2));

namedWindow("resim", WINDOW\_AUTOSIZE);

namedWindow("newresim", WINDOW\_AUTOSIZE);

imshow("resim", resim);

imshow("newresim", newsize);

waitKey(0);

destroyAllWindows;

return(0);

}

----rgb ve grayscale resim oluşturmak ---

int main()

{

Mat resim(240, 240, CV\_8UC3, Scalar(255, 0, 0)); //3 kanalli 8 bitlik bir resim olurşturdur BGR değerleri scalar olarak yazıldı.

Mat Grayresim(240, 240, CV\_8UC1, Scalar(123));

imshow("resim", resim);

imshow("resim2", Grayresim);

waitKey(0);

destroyAllWindows;

return(0);}

---------binary okuma,yazma--------------

int main(){

Mat Orgim = imread("lena.png");

Mat Grayim;

Mat binary;

cvtColor(Orgim, Grayim, COLOR\_BGR2GRAY);

threshold(Grayim, binary, 130, 255, THRESH\_BINARY);

imshow("Orgim", Orgim);

imshow("Grayim", Grayim);

imshow("Binary", binary);

waitKey(0);

return(0);

}

-------------resmi tersine çevirmek------

int main()

{

Mat Orgim = imread("lena.png");

Mat tersbit;

bitwise\_not(Orgim, tersbit);

imshow("Orgim", Orgim);

imshow("Tersbit", tersbit);

waitKey(0);

return(0);

}

-----------------split ve merge fonksiyonları------------------

int main()

{

Mat Orgim = imread("lena.png");

imshow("İlk Resim", Orgim);

Mat RgbKanal[3];

split(Orgim, RgbKanal); //rgb değerlenine göre kanallara ayırdık

imshow("R", RgbKanal[0]);

imshow("G", RgbKanal[1]);

imshow("B", RgbKanal[2]);

vector<Mat>kanallar;//vektör açtık

kanallar.push\_back(RgbKanal[0]); // kanalları çağırdık vektöre attık

kanallar.push\_back(RgbKanal[1]);

kanallar.push\_back(RgbKanal[2]);

Mat image;

merge(kanallar, image); // vektördeki kanalları tekrar birleştirdik

imshow("son", image);

waitKey(0);

return(0);}

----------- rgb pixel okuma--------------

int main()

{

Mat Orgim = imread("lena.png");

imshow("İlk Resim", Orgim);

mavi = Orgim.at<Vec3b>(100, 120)[0];

yesil = Orgim.at<Vec3b>(100, 120)[1];

kirmizi = Orgim.at<Vec3b>(100, 120)[2];

cout << "Mavi:" << mavi << " ";

cout << "Yesil:" << yesil << " ";

cout << "Kirmizi:" << kirmizi << endl;

waitKey(0);

return(0);

}

------------ grayscale pixel okuma-------

int main()

{

Mat Orgim = imread("lena.png");

resize(Orgim, Orgim, Size(500, 500));

cvtColor(Orgim, Gri, COLOR\_BGR2GRAY);

yogunluk = Gri.at<uchar>(100, 100);

cout << "yogunluk :" << yogunluk << " ";

namedWindow("image", WINDOW\_AUTOSIZE);

imshow("image", Gri);

waitKey(0);

return(0);

}

----------contras ayarı yapma------------

int yogunluk;

Mat Orgim = imread("lena.png");

Mat newhigh, newlow;

Orgim.convertTo(newhigh, -1, 1.3, 0);

Orgim.convertTo(newlow, -1, 0.3, 0);

namedWindow("orginal", WINDOW\_AUTOSIZE);

namedWindow("highcontrast", WINDOW\_AUTOSIZE);

namedWindow("lowcontrast", WINDOW\_AUTOSIZE);

imshow("orginal", Orgim);

imshow("highcontrast", newhigh);

imshow("lowcontrast", newlow);

waitKey(0);

return(0);

----------manual contrast ---------------

Mat son;

void convert(Mat resim, Mat cikis, int derinlik, float alfa) {

if (derinlik == -1) { //derinlik -1 ise rgb resim 1 ise grayscale resim için

for (int k = 0; k < resim.rows; k++) {

for (int l = 0; l < resim.cols; l++) {

if (resim.at<Vec3b>(k, l)[0] \* alfa>= 255)

resim.at<Vec3b>(k, l)[0] = 255;

else

resim.at<Vec3b>(k, l)[0] = resim.at<Vec3b>(k, l)[0] \* alfa;

if (resim.at<Vec3b>(k, l)[1] \* alfa >= 255)

resim.at<Vec3b>(k, l)[1] = 255;

else

resim.at<Vec3b>(k, l)[1] =resim.at<Vec3b>(k, l)[1] \* alfa;

if (resim.at<Vec3b>(k, l)[2] \* alfa >= 255)

resim.at<Vec3b>(k, l)[2] = 255;

else

resim.at<Vec3b>(k, l)[2] = resim.at<Vec3b>(k, l)[2] \* alfa;

}

}

}

if (derinlik == 1) {

cvtColor(resim,resim, COLOR\_BGR2GRAY);

for (int i = 0; i < resim.rows; i++)

{

for (int j = 0; j < resim.cols; j++)

{

if (resim.at<uchar>(i, j) \*alfa >= 255) resim.at<uchar>(i, j) = 255;

else resim.at<uchar>(i, j) = resim.at<uchar>(i, j) \* alfa;

}

}

}

son = resim;

}

int main()

{

Mat resim = imread("lena.png");

imshow("resim", resim);

convert(resim, son, -1,1.2 );

imshow("output", son);

convert(resim, son, 1, 0.2);

imshow("output2", son);

waitKey(0);

return(0);

}

----------trackbar oluşturma-------------

Int trackval;

int main()

{

namedWindow("Trackbar", WINDOW\_AUTOSIZE);

createTrackbar("Name", "Trackbar", &trackval, 100);

/\*while (true)

{

cout << trackval << endl;

if (waitKey(30) == 27)break;

}\*/

}

-------resme yazı oluşturma-------------

int main(){

Mat resim(500, 500, CV\_8UC3, Scalar(255, 0, 0));

putText(resim, "yazi", Point(100, 100), FONT\_HERSHEY\_PLAIN, 5.0,Scalar(0, 255, 0), 1);

imshow("face", resim);

waitKey(0);

destroyAllWindows();

}

----------resme çizgi oluşturma---------

int main(){

Mat orgim = imread("lena.png");

Mat chanim=orgim.clone();

line(chanim, Point(50, 50), Point(150, 100), Scalar(0, 255, 0), 2);

imshow("firsim", orgim);

imshow("secim", chanim);

waitKey(0);

destroyAllWindows();

}

--------line kütüphanesi oluşturma-------

void Line(Mat resim, Point Pt1, Point Pt2, Scalar scalar, int kalinlik) {

for (int i = Pt1.x; i <= Pt2.x; i++) {

for (int j = Pt1.y; j <= Pt2.y; j++) {

if (Pt1.x > Pt2.x && Pt1.y<=Pt2.y) {

resim.at<Vec3b>(j, i)[0] = scalar[0];

resim.at<Vec3b>(j, i)[1] = scalar[1];

resim.at<Vec3b>(j, i)[2] = scalar[2];

}

if( Pt1.y > Pt2.y && Pt1.x<=Pt2.y)

resim.at<Vec3b>(i, j)[0] = scalar[0];

resim.at<Vec3b>(i, j)[1] = scalar[1];

resim.at<Vec3b>(i, j)[2] = scalar[2];

}

}

}

int main(){

Mat resim(512, 512, CV\_8UC3, Scalar(255, 0, 0));

Line(resim, Point1, Point2, Scalar(0, 0, 255), 1);

imshow("firsim", resim);

waitKey(0);

destroyAllWindows();

}

--------------RESME DAİRE ÇİZDİRMEK------

int main(){

Mat resim(500, 500, CV\_8UC3, Scalar(50, 50, 50));

circle(resim, Point(100, 100), 100, Scalar(0, 0, 255), 3, LINE\_AA, 0);

imshow("firsim", resim);

waitKey(0);

destroyAllWindows();

}

-----------mause hareketlerini pixelleri gösterme-----------

Mat img;

void fonksiyon(int event, int x, int y, int flag, void\* userdata) {

if (event == EVENT\_MOUSEMOVE)

{

int blue = img.at<Vec3b>(y, x)[0];

int green = img.at<Vec3b>(y, x)[1];

int red = img.at<Vec3b>(y, x)[2];

cout << "blue:" << blue << " ";

cout << "green:" << green << " ";

cout << "red:" << red << " ";

cout << "sutun:" << x << " " << "satir:" << y << endl;

}

}

int main(){

img = imread("lena.png");

namedWindow("resim", WINDOW\_AUTOSIZE);

setMouseCallback("resim", fonksiyon, NULL);

imshow("resim", img);

waitKey(0);

destroyAllWindows();

}

-----resme mause ile cizgi çekme--------

Mat img;

int sayac;

Point pt1, pt2;

void fonksiyon(int event, int x, int y, int flag, void\* userdata) {

if (event == EVENT\_LBUTTONDOWN)

{

sayac++;

if (sayac == 1) {

pt1.x = x;

pt1.y = y;

}

if (sayac == 2) {

pt2.x = x;

pt2.y = y;

line(img, pt1, pt2, Scalar(0, 0, 255), 3, 8, 0);

imshow("resim", img);

sayac = 0;

}

}

}

int main(){

img = imread("lena.png");

namedWindow("resim", WINDOW\_AUTOSIZE);

setMouseCallback("resim", fonksiyon, NULL);

imshow("resim", img);

waitKey(0);

destroyAllWindows();

}

------------mouse ile resim kesme------

void fonksiyon(int event, int x, int y, int flags, void\* userdata) {

if (event == EVENT\_LBUTTONDOWN){

pt1.x = x;

pt1.y = y;

}

if (event == EVENT\_MOUSEMOVE && flags == EVENT\_FLAG\_LBUTTON) {

pt2.x = x;

pt2.y = y;

Mat gecici = Mat::zeros(img.size(), CV\_8UC3);

gecici = gecici + img;

rectangle(gecici, Point(pt1.x, pt1.y), Point(pt2.x, pt2.y), Scalar(0, 255, 0), 4, 8, 0);

imshow("kırp", gecici);

}

if (event = EVENT\_LBUTTONUP) {

Rect crop;

crop.x = pt1.x;

crop.y = pt1.y;

crop.width = pt2.x - pt1.x;

crop.height = pt2.y - pt1.y;

Mat sonresim = img(crop);

imshow("sonresim", sonresim);

}

}

int main(){

img = imread("lena.png");

namedWindow("resim", WINDOW\_AUTOSIZE);

setMouseCallback("resim", fonksiyon, NULL);

imshow("resim", img);

waitKey(0);

return 0;

}

---------------blur filtresi-------------

int main(){

img = imread("lena.png");

Mat cikis;

blur(img, cikis, Size(3, 3));

imshow("ilk", img);

imshow("son", cikis);

waitKey(0);

return 0;

}

----kendi yazdığımız blur filtresi-------

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/opencv.hpp>

#include <opencv2/highgui/highgui\_c.h>

using namespace cv;

float filter[7][7] = {{ 1,1,1,1,1,1,1 },{ 1,1,1,1,1,1,1 },{ 1,1,1,1,1,1,1 },{ 1,1,1,1,1,1,1 },{ 1,1,1,1,1,1,1 },{ 1,1,1,1,1,1,1 },{ 1,1,1,1,1,1,1 }};

int filterDimension = 7;

double toplam, toplam1, toplam2;

double k, l;

Mat img;

Mat img1;

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

img = imread("lena.png");

Mat img2(img.rows, img.cols, CV\_8UC3, Scalar(0, 0, 0));

namedWindow("Resim", CV\_WINDOW\_AUTOSIZE);

namedWindow("HazirFonk", CV\_WINDOW\_AUTOSIZE);

namedWindow("KendiFonk", CV\_WINDOW\_AUTOSIZE);

l = filterDimension \* filterDimension;

k = 1.0 / l;

for (int y = 0; y < img.rows - filterDimension; y++) {

for (int x = 0; x < img.cols - filterDimension; x++) {

for (int t = 0; t < filterDimension; t++) {

for (int f = 0; f < filterDimension; f++) {

toplam = toplam + img.at<Vec3b>(y + t, x + f)[0] \* filter[t][f] \* k;

toplam1 = toplam1 + img.at<Vec3b>(y + t, x + f)[1] \* filter[t][f] \* k;

toplam2 = toplam2 + img.at<Vec3b>(y + t, x + f)[2] \* filter[t][f] \* k;

}

}

img2.at<Vec3b>(y, x)[0] = toplam;

img2.at<Vec3b>(y, x)[1] = toplam1;

img2.at<Vec3b>(y, x)[2] = toplam2;

toplam = 0;

toplam1 = 0;

toplam2 = 0;

}

}

blur(img, img1, Size(7, 7), Point(-1, -1));

imshow("Resim", img);

imshow("HazirFonk", img1);

imshow("KendiFonk", img2);

waitKey(0);

return 0;

}

---------------median filtresi-----------

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/opencv.hpp>

#include <opencv2/highgui/highgui\_c.h>

using namespace cv;

Mat orgImg;

Mat dst;

int trackVal = 1;

void track(int, void \*) {

// Burada sadece tek say˝lar˝ almak zorunday˝z Á¸nk¸ 2 2 boyutunda bir filtrenin orta deeri olamaz.

if (trackVal % 2 == 1) {

medianBlur(orgImg, dst, trackval);

printf("%d", trackVal);

imshow("Haz˝rFonk", dst);

}

}

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

{

orgImg = imread("noise.jpg");

namedWindow("Trackbar Penceresi", CV\_WINDOW\_AUTOSIZE);

createTrackbar("Name", "Trackbar Penceresi", &trackVal, 100,track);

imshow("Resim", orgImg);

waitKey(0);

return 0;

}

----------------gaussian blur-----------

int main()

{

img = imread("lena.jpg");

GaussianBlur(img, dst, Size(5, 5), 1, 4);

imshow("Resim", img);

imshow("son",dst);

waitKey(0);

return 0;

}

----------------blateral blur-----------

int main()

{

img = imread("lena.jpg");

bilateralFilter(img, dst, 15, 100, 100);

imshow("Resim", img);

imshow("son",dst);

waitKey(0);

return 0;

}

----------------kenar keskinleştirme------------------

int main()

{

orgImg = imread("lena.png");

namedWindow("Trackbar Penceresi", CV\_WINDOW\_AUTOSIZE);

namedWindow("Resim", CV\_WINDOW\_AUTOSIZE);

namedWindow("Haz˝rFonk", CV\_WINDOW\_AUTOSIZE);

createTrackbar("Name", "Trackbar Penceresi", &trackVal, 100);

while (true) {

GaussianBlur(orgImg, dst, Size(3, 3), 11);

addWeighted(orgImg, trackVal / 7.0, dst, -0.5, 0, dst);

imshow("Resim", orgImg);

imshow("Haz˝rFonk", dst);

waitKey(100);

}

return 0;

}

---------gaussian kenar bulma------------

int main()

{

Mat img = imread("lena.png");

resize(img, img, Size(500, 500));

cvtColor(img, img2, CV\_RGB2GRAY);

GaussianBlur(img, img, Size(3, 3), 7);

GaussianBlur(img2, img2, Size(3, 3), 7);

Laplacian(img, img3, CV\_16S, 3);

Laplacian(img2, img4, CV\_16S, 3);

convertScaleAbs(img3, img3);

convertScaleAbs(img4, img4);

imshow("orjinal", img);

imshow("gray", img2);

imshow("orjinal2", img3);

imshow("gray2", img4);

waitKey(0);

return 0;

}

-----------canny kenar bulma----------

int main()

{

Mat img = imread("lena.png");

resize(img, img, Size(500, 500));

cvtColor(img, img2, CV\_RGB2GRAY);

imshow("orjinal", img);

Canny(img2, img3, 0, 100);

imshow("gray", img2);

imshow("canny", img3);

waitKey(0);

return 0;

}

----------------canny kenar bulma el yazımı-------------

void track(int,void\*) {

Mat bosresim(img.rows, img.cols, CV\_8UC1,Scalar(0));

for (int i = 0; i < img2.cols - 1; i++) {

for (int j = 0; j < img2.rows - 1; j++) {

int a = img2.at<uchar>(j + 1, i) - img2.at<uchar>(j, i);

int b = img2.at<uchar>(j , i) - img2.at<uchar>(j+1, i);

if (a > trackVal || b > trackVal) {

bosresim.at<uchar>(j, i) = 255;

}

}

}

for (int i = 0; i < img2.cols - 1; i++) {

for (int j = 0; j < img2.rows - 1; j++) {

int a = img2.at<uchar>(j , i+1) - img2.at<uchar>(j, i);

int b = img2.at<uchar>(j, i) - img2.at<uchar>(j , i+1);

if (a > trackVal || b > trackVal) {

bosresim.at<uchar>(j, i) = 255;

}

}

}

imshow("son", bosresim);

}

int main()

{

img = imread("opencv.png");

cvtColor(img, img2, CV\_BGR2GRAY);

namedWindow("trackbar", WINDOW\_AUTOSIZE);

createTrackbar("türev", "trackbar", &trackVal, 255, track);

imshow("gray", img2);

waitKey(0);

return 0;

}

----pixellerin olasılık dağılımı---------

int dizi[255];

int main()

{

img = imread("lena.png");

cvtColor(img, img2, CV\_BGR2GRAY);

for (int i = 0; i < img2.cols; i++) {

for (int j = 0; j < img2.rows; j++) {

dizi[img2.at<uchar>(j, i)]++;

}

}

for (int l = 0; l < 255; l++) {

printf("dizi %d : %d \n",l,dizi[l]);

}

imshow("gray", img2);

waitKey(0);

return 0;

}

--pixel olasılık dağılım grafiği ve quantization eşitleme--

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/opencv.hpp>

#include <opencv2/highgui/highgui\_c.h>

using namespace cv;

Mat img;

Mat img2;

Mat img3;

Mat img4;

Mat bosresim;

int dizi[255];

int dizi2[255];

int main(){

img = imread("lena.png");

cvtColor(img, img2, CV\_BGR2GRAY);

equalizeHist(img2, img3);

Mat graph2(512, 512, CV\_8UC3, Scalar(255, 255, 255));

for (int i = 0; i < img3.cols; i++) {

for (int j = 0; j < img3.rows; j++) {

dizi2[img3.at<uchar>(j, i)]++;

}

}

int max2 = dizi2[0];

for (int i = 1; i < 256; i++) {

if (dizi2[i] > max2)max2 = dizi2[i];

}

for (int i = 1; i < 256; i++) {

dizi2[i] = dizi2[i] \* 512 / max2;

}

for (int i = 0; i < 256; i++) {

line(graph2, Point(i \* 2, 512), Point(i \* 2, 512 - dizi2[i]),Scalar(i,0,255-i) ,2);

}

///////////////////7

Mat graph(512, 512, CV\_8UC3, Scalar(255, 255, 255));

for (int i = 0; i < img2.cols; i++) {

for (int j = 0; j < img2.rows; j++) {

dizi[img2.at<uchar>(j, i)]++;

}

}

int max = dizi[0];

for (int i = 1; i < 256; i++) {

if (dizi[i] > max)max = dizi[i];

}

for (int i = 1; i < 256; i++) {

dizi[i] = dizi[i] \* 512 / max;

}

for (int i = 0; i < 256; i++) {

line(graph, Point(i\*2 , 512), Point(i\*2 , 512 - dizi[i]), Scalar(i, 0, 255 - i), 2);

}

imshow("gray", img2);

imshow("equalized", img3);

imshow("grafik", graph);

imshow("equgrafik", graph2);

waitKey(0);

return 0;

}

-------------------eorsion aşındırma işlemi--------------

int main()

{

img = imread("lena.png");

erode(img, img2, getStructuringElement(MORPH\_RECT, Size(7, 7)));

imshow("orginal", img);

imshow("asindirilmis", img2);

waitKey(0);

return 0;

}

------genisletme islemleri diliate-------

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/opencv.hpp>

#include <opencv2/highgui/highgui\_c.h>

using namespace cv;

Mat img;

Mat img2;

Mat img3;

Mat img4;

Mat bosresim;

int main()

{

img = imread("blackwhite.png");

erode(img, img2, getStructuringElement(MORPH\_RECT, Size(7, 7)));

dilate(img, img3, getStructuringElement(MORPH\_RECT, Size(7, 7)));

imshow("orginal", img);

imshow("asindirilmis", img2);

imshow("genisletilmis", img3);

waitKey(0);

return 0;

}

--------open close işlemleri-------------

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/opencv.hpp>

#include <opencv2/highgui/highgui\_c.h>

using namespace cv;

Mat img;

Mat img2;

Mat img3;

Mat img4;

Mat bosresim;

int main()

{

img = imread("blackwhite.png");

imshow("orginal", img);

Mat element = getStructuringElement(MORPH\_RECT, Size(5, 5));

morphologyEx(img, img2, MORPH\_OPEN,element,Point(-1,-1),4);

morphologyEx(img, img3, MORPH\_CLOSE, element, Point(-1, -1), 4);

imshow("open", img2);

imshow("close", img3);

waitKey(0);

return 0;

}

------tophat ve gradiend işlemleri-------

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/opencv.hpp>

#include <opencv2/highgui/highgui\_c.h>

using namespace cv;

Mat img;

Mat img2;

Mat img3;

Mat img4;

Mat bosresim;

int main()

{

img = imread("blackwhite.png");

img2 = imread("lena.png");

imshow("orginal", img);

imshow("orginal2", img2);

Mat element = getStructuringElement(MORPH\_RECT, Size(5, 5));

morphologyEx(img, img3, MORPH\_TOPHAT, element, Point(-1, -1), 2);

morphologyEx(img2, img4, MORPH\_GRADIENT, element, Point(-1, -1), 3);

imshow("tophat", img3);

imshow("gradient", img4);

waitKey(0);

return 0;

}

----------yuvarlak bulma----------------

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/core/core.hpp"

#include <iostream>

#include <opencv2\imgproc\types\_c.h>

using namespace std;

using namespace cv;

Mat orgImg;

Mat src\_gray;

int main()

{

orgImg = imread("opencv.png");

cvtColor(orgImg, src\_gray, CV\_BGR2GRAY);

GaussianBlur(src\_gray, src\_gray, Size(3, 3), 2, 2);

vector<Vec3f> circles;

HoughCircles(src\_gray, circles, CV\_HOUGH\_GRADIENT, 1, src\_gray.rows / 8, 10,50);

for (size\_t i = 0; i < circles.size(); i++)

{

Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));

int radius = cvRound(circles[i][2]);

circle(orgImg, center, 3, Scalar(0, 255, 0), -1, 8, 0);

circle(orgImg, center, radius, Scalar(0, 0, 0), 3, 8, 0);

}

imshow("Hough Circle Transform Demo", orgImg);

waitKey(0);

return 0;

}

-----------line bulma---------------------------------------

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/core/core.hpp"

#include <iostream>

using namespace std;

using namespace cv;

Mat orgImg;

Mat dst,cdst;

int main()

{

orgImg = imread("bulmaca.jpg");

Canny(orgImg, dst, 50, 200, 3);

cvtColor(dst, cdst, CV\_GRAY2BGR);

vector<Vec2f> lines;

HoughLines(dst, lines, 1, CV\_PI / 180, 150);

for (size\_t i = 0; i < lines.size(); i++)

{

float rho = lines[i][0], theta = lines[i][1];

Point 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));

line(cdst, pt1, pt2, Scalar(0, 0, 255), 3, CV\_AA);

}

imshow("Orjinal Resim", cdst);

waitKey(0);

return 0;

}

-----------------köşe bulmaca-----------------------

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <opencv2\imgproc\types\_c.h>

using namespace cv;

using namespace std;

int thresh = 200;

Mat src, gray;

int main()

{

src = imread("bulmaca.jpg");

resize(src, src, Size(512, 512));

cvtColor(src, gray, CV\_BGR2GRAY);

Mat dst, dst\_norm, dst\_norm\_scaled;

dst = Mat::zeros(src.size(), CV\_32FC1);

cornerHarris(gray, dst, 7, 5, 0.05, BORDER\_DEFAULT);

normalize(dst, dst\_norm, 0, 255, NORM\_MINMAX, CV\_32FC1, Mat());

convertScaleAbs(dst\_norm, dst\_norm\_scaled);

for (int j = 0; j < dst\_norm.rows; j++)

{

for (int i = 0; i < dst\_norm.cols; i++)

{

if ((int)dst\_norm.at<float>(j, i) > thresh)

{

circle(dst\_norm\_scaled, Point(i, j), 5, Scalar(0), 2, 8, 0);

}

}

}

imshow("corners\_window", dst\_norm\_scaled);

waitKey(0);

return(0);

}

-----------eş resim bulma----------------

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

#include <stdio.h>

using namespace std;

using namespace cv;

Mat img; Mat templ; Mat result;

char\* image\_window = "Source Image";

char\* result\_window = "Result window";

int match\_method;

int max\_Trackbar = 5;

void MatchingMethod(int, void\*);

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

img = imread("ronaldo.jpg", 1);

templ = imread("smallronaldo.jpg", 1);

char\* trackbar\_label = "Method: \n 0: SQDIFF \n 1: SQDIFF NORMED \n 2: TM CCORR \n 3: TM CCORR NORMED \n 4: TM COEFF \n 5: TM COEFF NORMED";

createTrackbar(trackbar\_label, image\_window, &match\_method, max\_Trackbar, MatchingMethod);

MatchingMethod(0, 0);

waitKey(0);

return 0;

}

void MatchingMethod(int, void\*)

{

Mat img\_display;

img.copyTo(img\_display);

int result\_cols = img.cols - templ.cols + 1;

int result\_rows = img.rows - templ.rows + 1;

result.create(result\_rows, result\_cols, CV\_32FC1);

matchTemplate(img, templ, result, match\_method);

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

double minVal; double maxVal; Point minLoc; Point maxLoc;

Point matchLoc;

minMaxLoc(result, &minVal, &maxVal, &minLoc, &maxLoc, Mat());

if (match\_method == CV\_TM\_SQDIFF || match\_method == CV\_TM\_SQDIFF\_NORMED){

matchLoc = minLoc;

}

else{

matchLoc = maxLoc;

}

rectangle(img\_display, matchLoc, Point(matchLoc.x + templ.cols, matchLoc.y + templ.rows), Scalar::all(0), 2, 8, 0);

rectangle(result, matchLoc, Point(matchLoc.x + templ.cols, matchLoc.y + templ.rows), Scalar::all(0), 2, 8, 0);

imshow(image\_window, img\_display);

imshow("resim", templ);

return;

}