#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

#include <opencv2\imgproc\imgproc.hpp>

#include <windows.h>

#include <tchar.h>

using namespace cv;

using namespace std;

int minx, miny, maxx, maxy, avgx, avgy, rad;

void putBlack(Mat img, int y, int x)

{

img.at<Vec3b>(y, x)[0] = saturate\_cast<uchar>(0);

img.at<Vec3b>(y, x)[1] = saturate\_cast<uchar>(0);

img.at<Vec3b>(y, x)[2] = saturate\_cast<uchar>(0);

}

void putWhite(Mat img, int y, int x)

{

img.at<Vec3b>(y, x)[0] = saturate\_cast<uchar>(255);

img.at<Vec3b>(y, x)[1] = saturate\_cast<uchar>(255);

img.at<Vec3b>(y, x)[2] = saturate\_cast<uchar>(255);

}

cv::Mat imageBrighten(Mat image)

{

// Increasing the brightness by 50.

Mat imgBright = Mat::zeros(image.size(), image.type());

for (int y = 0; y < image.rows; y++)

{

for (int x = 0; x < image.cols; x++)

{

for (int c = 0; c < 3; c++)

{

imgBright.at<Vec3b>(y, x)[c] = saturate\_cast<uchar>((image.at<Vec3b>(y, x)[c]) + 50);

}

}

}

return imgBright;

}

cv::Mat cvt2HSV(Mat image)

{

//Converting into HSV color space

Mat imgHSV = Mat::zeros(image.size(), image.type());

double r, g, b, h, s, v, max, min, delta;

for (int y = 0; y < image.rows; y++)

{

for (int x = 0; x < image.cols; x++)

{

b = (double)image.at<Vec3b>(y, x)[0] / 255;

g = (double)image.at<Vec3b>(y, x)[1] / 255;

r = (double)image.at<Vec3b>(y, x)[2] / 255;

min = r < g ? r : g;

min = min < b ? min : b;

max = r > g ? r : g;

max = max > b ? max : b;

//cout << max << " " << min;

v = max \* 255.0;

imgHSV.at<Vec3b>(y, x)[2] = saturate\_cast<uchar>(v);

delta = max - min;

if (delta == 0)

{

h = 0;

}

else

{

if (r == max)

h = 60 \* (g - b) / delta;

else

{

if (g == max)

h = 60 \* (((b - r) / delta) + 2);

else

h = 60 \* (((r - g) / delta) + 4);

}

}

if (h < 0)

{

h = h + 360;

}

//cout << h;

h = h / 2;

imgHSV.at<Vec3b>(y, x)[0] = saturate\_cast<uchar>(h);

if (max != 0.0) {

s = (delta / max);

}

else {

s = 0.0;

}

s = s \* 255.0;

imgHSV.at<Vec3b>(y, x)[1] = saturate\_cast<uchar>(s);

}

}

return imgHSV;

}

cv::Mat ODRGB(Mat image)

{

//Object detection in RGB Domain - imgMaskRBG

Mat imgMaskRGB = Mat::zeros(image.size(), image.type());

int b, g, r;

for (int y = 0; y < imgMaskRGB.rows; y++)

{

for (int x = 0; x < imgMaskRGB.cols; x++)

{

b = image.at<Vec3b>(y, x)[0];

g = image.at<Vec3b>(y, x)[1];

r = image.at<Vec3b>(y, x)[2];

if (r > 140)

{

if (g < 120)

putWhite(imgMaskRGB, y, x);

else

putBlack(imgMaskRGB, y, x);

}

else

{

putBlack(imgMaskRGB, y, x);

}

}

}

for (int i = 0; i < 10; i++)

{

medianBlur(imgMaskRGB, imgMaskRGB, 5);

}

minx = imgMaskRGB.rows, miny = minx = imgMaskRGB.cols;

maxx = 0, maxy = 0;

for (int y = 0; y < imgMaskRGB.rows; y++)

{

for (int x = 0; x < imgMaskRGB.cols; x++)

{

if (imgMaskRGB.at<Vec3b>(y, x)[2])

{

if (minx > x)

minx = x;

if (maxx < x)

maxx = x;

}

}

}

for (int y = 0; y < imgMaskRGB.cols; y++)

{

for (int x = 0; x < imgMaskRGB.rows; x++)

{

if (imgMaskRGB.at<Vec3b>(x, y)[2])

{

if (miny > x)

miny = x;

if (maxy < x)

maxy = x;

}

}

}

avgx = (minx + maxx) / 2;

avgy = (miny + maxy) / 2;

rad = ((maxx - minx) / 2) >((maxy - miny) / 2) ? ((maxx - minx) / 2) : ((maxy - miny) / 2);

circle(imgMaskRGB, Point(avgx, avgy), rad, Scalar(0, 0, 255), 3, LINE\_8);

return imgMaskRGB;

}

cv::Mat ODHSV(Mat image)

{

//Object detection in HSV Domain - imgMaskHSV

Mat imgMaskHSV = Mat::zeros(image.size(), image.type());

int h, s, v;

for (int y = 0; y < imgMaskHSV.rows; y++)

{

for (int x = 0; x < imgMaskHSV.cols; x++)

{

h = image.at<Vec3b>(y, x)[0];

s = image.at<Vec3b>(y, x)[1];

v = image.at<Vec3b>(y, x)[2];

if ((h > 0 && h < 12) || (h > 170 && h < 180))

{

if (s > 70 && v > 50)

putWhite(imgMaskHSV, y, x);

else

putBlack(imgMaskHSV, y, x);

}

else

{

putBlack(imgMaskHSV, y, x);

}

}

}

for (int i = 0; i < 10; i++)

{

medianBlur(imgMaskHSV, imgMaskHSV, 5);

}

minx = imgMaskHSV.rows, miny = minx = imgMaskHSV.cols;

maxx = 0, maxy = 0;

for (int y = 0; y < imgMaskHSV.rows; y++)

{

for (int x = 0; x < imgMaskHSV.cols; x++)

{

if (imgMaskHSV.at<Vec3b>(y, x)[2])

{

if (minx > x)

minx = x;

if (maxx < x)

maxx = x;

}

}

}

for (int y = 0; y < imgMaskHSV.cols; y++)

{

for (int x = 0; x < imgMaskHSV.rows; x++)

{

if (imgMaskHSV.at<Vec3b>(x, y)[2])

{

if (miny > x)

miny = x;

if (maxy < x)

maxy = x;

}

}

}

avgx = (minx + maxx) / 2;

avgy = (miny + maxy) / 2;

rad = ((maxx - minx) / 2) >((maxy - miny) / 2) ? ((maxx - minx) / 2) : ((maxy - miny) / 2);

circle(imgMaskHSV, Point(avgx, avgy), rad, Scalar(0, 0, 255), 3, LINE\_8);

return imgMaskHSV;

}

cv::Mat ChangingColorsHSV(Mat image)

{

//ChangingColors

Mat imgCC = Mat::zeros(image.size(), image.type());

cvtColor(image, imgCC, CV\_BGR2HSV);

for (int y = 0; y < imgCC.rows; y++)

{

for (int x = 0; x < imgCC.cols; x++)

{

Vec3b color = imgCC.at<Vec3b>(Point(x, y));

if (color[0] > 0 && color[0] < 20)

{

imgCC.at<Vec3b>(Point(x, y))[0] = color[0] + 45;

}

else if (color[0] > 160)

{

imgCC.at<Vec3b>(Point(x, y))[0] = color[0] / 4;

}

else if (color[0] > 30 && color[0] < 80)

{

imgCC.at<Vec3b>(Point(x, y))[0] = color[0] \* 0.05;

}

}

}

cvtColor(imgCC, imgCC, CV\_HSV2BGR);

return imgCC;

}

cv::Mat ChangingColorsRGB(Mat image)

{

Mat imgCC = Mat::zeros(image.size(), image.type());

for (int y = 0; y < imgCC.rows; y++)

{

for (int x = 0; x < imgCC.cols; x++)

{

imgCC.at<Vec3b>(Point(x, y))[0] = image.at<Vec3b>(Point(x, y))[0];

imgCC.at<Vec3b>(Point(x, y))[1] = image.at<Vec3b>(Point(x, y))[2];

imgCC.at<Vec3b>(Point(x, y))[2] = image.at<Vec3b>(Point(x, y))[1];

}

}

return imgCC;

}

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

{

Mat image = imread(argv[1], IMREAD\_COLOR); // Read the file

if (!image.data) // Check for invalid input

{

cout << "Could not open or find the image" << std::endl;

return -1;

}

namedWindow("Original Image", WINDOW\_AUTOSIZE); // Create a window for display.

imshow("Original Image", image);

Mat imgBright = imageBrighten(image);

namedWindow("Brightened Image", WINDOW\_AUTOSIZE);

imshow("Brightened Image", imgBright);

Mat convertedOGimage = cvt2HSV(image);

Mat convertedBrightenedimage = cvt2HSV(imgBright);

namedWindow("convertedOGimage", WINDOW\_AUTOSIZE);

imshow("convertedOGimage", convertedOGimage);

namedWindow("convertedBrightenedimage", WINDOW\_AUTOSIZE);

imshow("convertedBrightenedimage", convertedBrightenedimage);

Mat imgMaskRGB = ODRGB(image);

namedWindow("Object detection in RGB Domain", WINDOW\_AUTOSIZE);

imshow("Object detection in RGB Domain", imgMaskRGB);

Mat imgMaskHSV = ODHSV(convertedOGimage);

namedWindow("Object detection in HSV Domain", WINDOW\_AUTOSIZE);

imshow("Object detection in HSV Domain", imgMaskHSV);

Mat imgCCHSV = ChangingColorsHSV(image);

namedWindow("CCHSV", WINDOW\_AUTOSIZE);

imshow("CCHSV", imgCCHSV);

Mat imgCCRGB = ChangingColorsRGB(image);

namedWindow("CCRGB", WINDOW\_AUTOSIZE);

imshow("CCRGB", imgCCRGB);

CreateDirectory(\_T("OutputImages"), 0);

vector<int> compression\_params;

compression\_params.push\_back(CV\_IMWRITE\_JPEG\_QUALITY);

compression\_params.push\_back(100);

cv::imwrite("OutputImages/OriginalInputImage.jpg", image, compression\_params);

cv::imwrite("OutputImages/BrightenedImage.jpg", imgBright, compression\_params);

cv::imwrite("OutputImages/convertedOGimage.jpg", convertedOGimage, compression\_params);

cv::imwrite("OutputImages/convertedBrightenedimage.jpg", convertedBrightenedimage, compression\_params);

cv::imwrite("OutputImages/ObjectDetectionInRGBDomain.jpg", imgMaskRGB, compression\_params);

cv::imwrite("OutputImages/ObjectDetectionInHSVDomain.jpg", imgMaskHSV, compression\_params);

cv::imwrite("OutputImages/InvertedColorHSV.jpg", imgCCHSV, compression\_params);

cv::imwrite("OutputImages/InvertedColorRGB.jpg", imgCCRGB, compression\_params);

waitKey(0); // Wait for a keystroke in the window

return 0;

}