Laboratorium 8

Na koniec zajęć wysyłamy pliki z wykonanymi ćwiczeniami na adres **dydaktyka.sk@gmail.com**. Pliki NALEŻY nazywać w następujący sposób: **imię_nazwisko_nrindexu_zadanieX.cpp** a w tytule maila podać: **Imię i Nazwisko**. Proszę nie opuszczać sali laboratoryjnej zanim prowadzący nie potwierdzi otrzymania maila z zadaniami. W przypadku nie otrzymania maila przez prowadzącego z jakichkolwiek przyczyn obecność na zajęciach nie zostanie uznana. Zadania punktowane wykonujemy na zajęciach lub w domu i wysyłamy najpóźniej do 10min PRZED następnymi zajęciami (tydzień na wykonanie zadania).

Wczytywanie zdjęć, skala szarości, zapisywanie

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

using namespace cv;

int main()
{
char* imageName = "lena.jpg";

Mat image;
image = imread( imageName, 1 );


Mat gray_image;
cvtColor( image, gray_image, CV_BGR2GRAY );

imwrite( "../../images/Gray_Image.jpg", gray_image );

namedWindow( imageName, CV_WINDOW_AUTOSIZE );
namedWindow( "Gray image", CV_WINDOW_AUTOSIZE );

imshow( imageName, image );
imshow( "Gray image", gray_image );

waitKey(0);

return 0;
}
```

Wyjaśnienie

1. Zaczynamy ładowanie obrazu za pomocą imread, znajdującego się w ścieżce zawartej w zmiennej imageName. Na przykład załóżmy że ładujesz obraz RGB.
2. Teraz będziemy konwertować nasz obraz z BGR do skali szarości. OpenCV ma naprawdę ładną funkcję do tego rodzaju przekształcenia:

cvtColor (obraz, gray_image, CV_BGR2GRAY);

Jak widać, cvtColor ma jako argumenty:

- źródło obrazu (image),
- obraz docelowy (gray_image), w którym będziemy zapisywać obraz przerobiony,
- dodatkowy parametr, który wskazuje, jakiego rodzaju transformacje będą wykonywane. W tym przypadku możemy użyć CV_BGR2GRAY (funkcja imread ma domyślną kolejność kanałów w przypadku obrazów kolorowych: BGR).

3. Tak więc teraz mamy nasz nowy gray_image i chcemy zapisać go na dysku (w przeciwnym razie będzie utracony po zakończeniu programu). Aby zapisać go będziemy używać funkcji analogicznej do imread: imwrite

imwrite (".../../ images/Gray_Image.jpg ", gray_image);

który zapisze nasz gray_image jako Gray_Image.jpg w folderze obrazów zlokalizowanym dwa poziomy wyżej niż obecna lokalizacja.

4. Wreszcie sprawdźmy zdjęcia. Tworzymy dwa okna i używamy ich do pokazania oryginalnego obrazu, jak również nowego:

```
namedWindow (imageName, CV_WINDOW_AUTOSIZE);
namedWindow ("szary obrazek", CV_WINDOW_AUTOSIZE);
imshow (imageName, obraz);
imshow ("Szary obrazek", gray_image);
```

5. Dodajemy wywołanie funkcji waitKey(0) aby program czekał na naciśnięcia klawisza przez użytkownika.

Wykrywanie twarzy

Wykrywanie twarzy w obrazach jest banalne przy zastosowaniu biblioteki OpenCV. Cały kod sprowadza się do wykonania dwóch instrukcji: załadowania klasyfikatora i uruchomienia go. Oczywiście, wyniki detekcji trzeba jeszcze jakoś pokazać, na przykład rysując prostokąt w miejscu wykrytej twarzy. Poniżej najprostszy przykład:

```
#include <opencv2\opencv.hpp>

using namespace std;
using namespace cv;

void main()
{
    CascadeClassifier kl_twarz;

    //załadowanie gotowego klasyfikatora; plik dostępny pod adr. http://goo.gl/qyA2a0
    //na wszelki wypadek ze sprawdzeniem czy się udało
    if(!kl twarz.load("C:\\OpenCV248\\data\\haarcascades\\haarcascade frontalface default
.xml"))
        cout << "Nie moge zaladowac klasyfikatora" << endl;

    //wektor prostokątów oznaczających obszar znalezionych twarzy
    vector<Rect> twarze;

    Mat obraz=imread("lena.jpg");
    kl_twarz.detectMultiScale( obraz, twarze);

    //narysowanie wszystkich znalezionych prostokątów
    for( int i = 0; i < twarze.size(); i++ )
        rectangle(obraz,twarze[i],CV_RGB(255,0,0),2);

    imshow("Detekcja twarzy",obraz);
    waitKey();
}
```

Jeżeli detektor wykrył dwie twarze, podczas gdy faktycznie jest tylko jedna. Można temu zaradzić na kilka sposobów modyfikując parametry metody detectMultiScale.

void CascadeClassifier::detectMultiScale(const Mat& **image**, vector<Rect>& **objects**, double **scaleFactor**=1.1, int**minNeighbors**=3, int **flags**=0, Size **minSize**=Size(), Size **maxSize**=Size())

1. Parametr **minNeighbors** reguluje czułość algorytmu. Im większy, tym większa szansa, że fałszywe twarze nie zostaną wykryte. Nie należy go jednak za bardzo zwiększać, gdyż wówczas algorytm może przeoczyć niezbyt wyraźne czy lekko przesłonięte twarze. W naszym przypadku wystarczy zwiększyć go do wartości 4.
2. Inny sposób to ograniczenie rozmiaru poszukiwanych obiektów poprzez ustawienie **minSize** na Size(40,40). Spowoduje to nieposzukiwanie obiektów mniejszych niż 40 na 40 pikseli.
3. Można jeszcze ustawić **flags** na CV_HAAR_FIND_BIGGEST_OBJECT, co spowoduje że zapamiętany zostanie jedynie największy ze wszystkich obiektów (to raczej się nie sprawdzi, gdy na obrazie jest faktycznie więcej niż jedna twarz).

Poza wyszukiwaniem twarzy, możemy też znaleźć, korzystając z klasyfikatorów dostarczonych wraz z biblioteką, inne części twarzy. Oto przykładowy kod znajdujący twarze i parę oczu:

```
kl_oczy.load("C:\\OpenCV248\\data\\haarcascades\\haarcascade_eye_tree_eyeglasses.xml");
//ten plik dostępny pod adr. http://goo.gl/ABjb4Z
kl_oczy.detectMultiScale( obraz, oczy);
for( int i = 0; i < oczy.size(); i++ )
  ellipse(obraz,Point(oczy[i].x+oczy[i].width/2,oczy[i].y+oczy[i].height/2),Size(oczy[i].
width/2,oczy[i].height/2),360,0,360,CV_RGB(255,255,0),2);
```

Możliwa modyfikacja klasyfikatora to np.:

```
kl_oczy.detectMultiScale( obraz, oczy,1.1,3);
```

ZADANIA PUNKTOWANE (1 zadanie):

Zadanie do oddania do następnych zajęć (3 pkt) Załaduj dowolne zdjęcie zbiorowe dowolnej klasy, przekształć je do skali szarości a następnie wykonaj procedurę wyszukiwania twarzy i oczu. Obrysuj oczy zielonymi okręgami a twarze czerwonym prostokątem. Dopasuj parametry klasyfikatora w taki sposób aby nie wykrywał błędnie obiektów nie będących oczyma lub twarzami.