KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Instrukcja współfinansowana przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego
w projekcie

*„Innowacyjna dydaktyka bez ograniczeń
– zintegrowany rozwój Politechniki Łódzkiej – zarządzanie Uczelnią,
nowoczesna oferta edukacyjna i wzmacniania zdolności
do zatrudniania osób niepełnosprawnych”*

Instrukcja jest dystrybuowana bezpłatnie.

Instrukcja do laboratorium

Hubert Nowak

Programy komputerowe do przetwarzania i analizy obrazów oraz wideo I

Zadanie nr 13 – Studia podyplomowe „**Przetwarzanie i analiza obrazów biomedycznych**”

Politechnika Łódzka
Instytut Elektroniki

90-924 Łódź, ul. Żeromskiego 116,
tel. 042 631 28 83
www.kapitalludzki.p.lodz.pl

Spis treści

Spis treści 2
Wprowadzenie 2
Instalacja i konfiguracja środowiska 3
Przygotowanie projektu aplikacji dla VC++ 3
Uwagi do konfiguracji Visual Studio Express Edition 5
Konfiguracja środowiska BC++ i przygotowanie bibliotek (*.lib) 6
Projekt aplikacji dla BC++ 6
Podstawowe funkcje przetwarzania obrazu. 8
Zawartość skrypty do ćwiczeń. 8
Wczytywanie i wyświetlanie obrazu 9
Kopiowanie obrazu (i odbicie obrazu) 10
Tworzenie nowej zmiennej dla obrazu i skalowanie obrazu 11
Korekcja jasności 12
Korekcja jasności obrazów barwnych 13
Manipulacja przestrzeniami barw 14
Filtracja 16
Segmentacja obrazu - Progowanie 19
Interakcja z użytkownikiem 19
Użycie suwaków (Trackbar) do zmiany parametrów przetwarzania 20
Przetwarzanie sekwencji wideo. 21
Odczyt i przetwarzanie obrazów z sekwencji wideo. 21

Wprowadzenie

Biblioteka Intel OpenCV została stworzona jako narzędzie do tworzenia aplikacji z dziedziny Computer Vision działających w czasie rzeczywistym. Jednym z głównych elementów jest zbiór zmiennych i funkcji do przetwarzania obrazów. Biblioteka jest bezpłatna i można ją pobrać pod ze strony internetowej:
http://sourceforge.net/projects/opencvlibrary/
Najnowsza dostępna wersja to 1.0. W przypadku programowania w środowisku Borland C++ Builder lepiej jest wykorzystywać wersję Beta 5, ponieważ w innych wersjach (w tym finalnych jak wersja 1.0) obsługa tego środowiska nie jest pełna. Środowisko Visual C++ jest w pełni kompatybilne ze wszystkimi wersjami. Po zainstalowaniu OpenCV dostępne są szablony projektów dla Visual Studio wykorzystujących OpenCV. Szczegóły, pomoc oraz instrukcje używania biblioteki w różnych środowiskach programistycznych można znaleźć na stronie internetowej:
http://opencvlibrary.sourceforge.net

Instalacja i konfiguracja środowiska

Bibliotekę można pobrać z Internetu w postaci spakowanego archiwum, kodów źródłowych lub najprościej w postaci pliku instalatora dla systemu Windows. Dla aktualnej wersji 1.0 plik instalatora to OpenCV_1.0.exe i tej właśnie wersji dotyczyć będą zawarte tu instrukcje (dla środowiska Borland C++ Builder zalecane jest stosowanie wersji beta5 - plik instalatora to OpenCV_b5a.exe). Domyślnie biblioteka instalowana jest w folderze "C:\Program Files\OpenCV" w którym tworzona jest odpowiednia struktura katalogów z których najistotniejsze to:

C:\Program Files\OpenCV\bin - biblioteki dynamiczne (*.dll)
...\cxcore - kody źródłowe dla modułu cxcore, podstawowe typy danych i funkcje.
...\cv - j.w. dla cv, funkcje przetwarzania obrazu
...\cvaux - j.w. dla cvaux, funkcje zaawansowane i eksperymentalne (3D, PCA, HMM)
...\otherlibs_graphics - funkcje obsługi różnych formatów plików graficznych
...\cvcam - kody źródłowe dla modułu cvcam do obsługi video i kamer
...\highgui - kody źródłowe dla modułu highgui czyli graficznego interfejsu użytkownika
...\lib - biblioteki importu (*.lib)
...\docs - dokumentacja biblioteki OpenCV

Przydatna może okazać się także zawartość folderów:

..._make - szablony projektu dla VC++ i konfiguracja innych kompilatorów.
...\sample - przykładowe aplikacje, wraz z kodami źródłowymi

Biblioteki importu i biblioteki dynamiczne zostały wygenerowane w środowisku Visual C++. W tym właśnie środowisku, po instalacji, biblioteka może być od razu wykorzystywana (dostarczony jest nawet szablon projektu dla VC++ w wersji 6). Ponieważ biblioteki importu dla VC++ (Visual Studio C++) i BC++ (Borland C++) mają inny format to należy w pierwszej kolejności wygenerować odpowiednie dla BC++ biblioteki.

Przygotowanie projektu aplikacji dla VC++

Po zainstalowaniu OpenCV dostępne są szablony projektów aplikacji w Visual C++ .NET 2003. W tym właśnie środowisku zostały skompilowane biblioteki importu oraz dołączane dynamicznie biblioteki (*.dll). Uruchomienie projektu w tym środowisku nie nastręcza więc kłopotów - wystarczy użyć gotowego projektu. W przypadku innych wersji konieczne jest ręczne wygenerowanie i skonfigurowanie projektu oraz samego środowiska. Poniższy opis dotyczy wersji Visual C++ 2005 i może być także wykorzystany do konfiguracji innych wersji środowiska VC++. W nowszych wersjach konfiguracja i tworzenie projektu przebiega analogicznie.

Opis przygotowania projektu aplikacji w Visual Studio 6.0 znajduje się także pod tym adresem:
http://opencvlibrary.sourceforge.net/VisualC%2B%2B

Konfiguracja środowiska:
1. Po uruchomieniu Visual C++ należy z menu głównego wybrać *Tools* > *Options*
2. Z listy po lewej stronie wybrać *Projects* > *VC++ Directories*.
3. W polu oznaczonym *Show Directories for* zaznaczyć *Library files*.
4. kliknąć ikonę *New Line*, po czym w nowej linii podać ścieżkę do katalogu bibliotek OpenCV w tym przypadku "C:\Program Files\OpenCV\lib"

5. Zmienić zaznaczenie w polu *Show Directories for* na *Include files* i tak samo jak w punkcie 4 dodać ścieżki do katalogów z plikami nagłówkowymi:

"C:\Program Files\OpenCV\cv\include"
"C:\Program Files\OpenCV\cxcore\include"
"C:\Program Files\OpenCV\otherlibs\highgui"
"C:\Program Files\OpenCV\cvaux\include"
"C:\Program Files\OpenCV\otherlibs\cvcam\include"

6. Zmienić zaznaczenie w polu *Show Directories for* na *Source files* i tak samo jak w punkcie 4 dodać ścieżki do katalogów z plikami wykonywalnymi:

"C:\Program Files\OpenCV\cv\src"
"C:\Program Files\OpenCV\cxcore\src"
"C:\Program Files\OpenCV\cvaux\src"
"C:\Program Files\OpenCV\otherlibs\highgui"
"C:\Program Files\OpenCV\otherlibs\cvcam\src\windows"

W tym momencie kończy się konfiguracja środowiska Visual Studio do korzystania z bibliotek OpenCV. Dalsze punkty dotyczą już projektu programu.

7. Można teraz stworzyć projekt aplikacji wybierając z menu głównego kolejno *File* > *New* > *Project*.

8. W otwartym oknie należy wybrać typ aplikacji, np. *Win32 Console Application* i w polu *Name* podać nazwę projektu oraz zaznaczyć opcję: *Create directory for solution*.

9. W kolejnym oknie należy przejść do zakładki *Application Setings* i sprawdzić czy zaznaczona jest opcja *Console Application* i ewentualnie *Precompiled header* (nie jest konieczna ale jej zaznaczenie skraca nieco czas kolejnych kompilacji programu), po czy wcisnąć OK.

Automatycznie wygenerowany zostanie projekt programu. Struktura projektu ze wszystkimi automatycznie wygenerowanymi plikami jest wyświetlona w oknie z lewej strony ekranu (Solution Explorer). W polu edycji otwarty zostanie plik z główną funkcją programu _tmain(...):

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{
      return 0;
}
```

10. Wygenerowany kod aplikacji można teraz skompilować wybierając z menu głównego kolejno
Build > *Build Solution*.

11. Aby korzystać z biblioteki OpenCV należy do aplikacji dołączyć pliki nagłówkowe. Najwygodniej jest dołączyć je w pliku stdafx.h (w przeciwnym wypadku mogą się pojawić błędy kompilacji) wpisując następujący fragment kodu:

```
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
#include <cvaux.h>
#include <cvcam.h>
```

12. Aby uniknąć możliwych błędów linkera należy wybrać z menu kolejno *Project* > *Properties*, następnie wybrać z listy pole *Linker* > *Input*, gdzie w polu *Additional Dependencies* wpisać nazwy plików: cxcore.lib cv.lib highgui.lib cvaux.lib cvcam.lib.

Zamiast tego można w miejscu dopisania plików nagłówkowych (pkt. 11) dodać następujący kod:

```
#pragma comment (lib, "cv.lib")
#pragma comment (lib, "cxcore.lib")
#pragma comment (lib, "cvaux.lib")
#pragma comment (lib, "cvcam.lib")
#pragma comment (lib, "highgui.lib").
```

Można teraz wewnątrz funkcji _tmain(...) wpisać kod wykorzystujący OpenCV a następnie skompilować program.
Kolejny projekt nie wymaga już konfiguracji a jedynie dołączenia plików nagłówkowych do kodu i powinien kompilować się bez błędów.
W przypadku Visual C++ 6, lub Visual C++ .Net aplikacja powinna się kompilować i uruchamiać bez problemów.
Przykładowy program wykorzystujący bibliotekę OpenCV do wyświetlenia obrazu z pliku (tu pliki nagłówkowe dodano bezpośrednio w głównym pliku z kodem programu):

```
#include "stdafx.h"
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
#include <cvaux.h>
#include <cvcam.h>

int _tmain(int argc, _TCHAR* argv[])
{
      IplImage *img= cvLoadImage("E:\\lena.bmp",0);
    cvNamedWindow("Image", 1);
    cvShowImage("Image",img);
    cvWaitKey(0);
    cvReleaseImage(&img);
    return 0;
}
```

Podany tu przykład programu, przy prawidłowo skonfigurowanym środowisku i projekcie powinien się kompilować i uruchamiać bez problemu.

Uwagi do konfiguracji Visual Studio Express Edition

W wersji Express Edition (np. Visual C++ 2005 Express Edition) dostępnych bezpłatnie także do zastosowań komercyjnych, konieczne jest zainstalowanie Microsoft Windows platform SDK, który można zainstalować z Internetu bądź pobrać obraz ISO płyty DVD. Adres skąd można pobrać SDK:
http://www.microsoft.com/downloads/details.aspx?FamilyId=0BAF2B35-C656-4969-ACE8-E4C0C0716ADB&displaylang=en
Podczas instalacji wystarczy wybrać komponenty dla C++. Po zainstalowaniu SDK konieczne jest jeszcze skonfigurowanie środowiska Visual C++. W tym celu należy:
1. Wybrać z menu głównego *Tools* > *Options*.
2. Z listy po lewej stronie okna dialogowego wybrać *Project and Solutions* > *VC++ Directories*.
3. W polu *Show Directory* for: zaznaczyć *Include files* a w polu poniżej dodać wiersz ze ścieżką do folderu z plikami nagłówkowymi: C:\Program Files\Microsoft

SDKs\Windows\v6.1\Include (przy założeniu że biblioteka zainstalowana jest w domyślnej lokalizacji).
4. W polu *Show Directory* for: zaznaczyć *Library files* a w polu poniżej dodać wiersz ze ścieżką do folderu z bibliotekami importu: C:\Program Files\Microsoft SDKs\Windows\v6.1\Lib.
Po skonfigurowaniu IDE, można skompilować projekt (Build > Build Solution F7) i uruchomić go (Debug > Start Without Debugging Ctrl+F5).

Konfiguracja środowiska BC++ i przygotowanie bibliotek (*.lib)

Aby korzystać z biblioteki konieczne jest wygenerowanie bibliotek importu (*.lib). Można to zrobić poprzez kompilacje całej biblioteki z kodów źródłowych (metoda może być użyta także dla innych kompilatorów), konwersję z bibliotek dynamicznych (za pomocą programu implib.exe, dostarczanego razem ze środowiskiem BC++) lub poprzez konwersję bibliotek dla VC++ (za pomocą programu coff2omf.exe, będącego częścią środowiska BC++).
Biblioteki importu znajdują się w katalogu "..\OpenCV\lib". Konwersję wygodnie jest wykonać za pomocą pliku wsadowego umieszczonego w katalogu w którym znajdują się biblioteki, którego zawartość pokazano poniżej.

```
md bcblib
coff2omf -v cv.lib bcblib\cv097.lib
coff2omf -v cvaux.lib bcblib\cvaux097.lib
coff2omf -v cvcam.lib bcblib\cvcam097.lib
coff2omf -v cxcore.lib bcblib\cxcore097.lib
coff2omf -v highgui.lib bcblib\highgui097.lib
coff2omf -v trs.lib bcblib\trs.lib
coff2omf -v cxt.lib bcblib\cxts001.lib
```

Nowe biblioteki zostaną zapisane do nowo utworzonego katalogu "..\OpenCV\lib\bcblib", który po wygenerowaniu bibliotek wygodnie jest przenieść bezpośrednio do katalogu "..\OpenCV". Można także wykonać odpowiednią sekwencję operacji wpisując komendy w wierszu poleceń (należy wtedy najpierw przejść do odpowiedniego katalogu – w tym przypadku "..\OpenCV\lib\").

Projekt aplikacji dla BC++

Ponieważ OpenCV zawiera funkcje do obsługi interfejsu użytkownika nie ma konieczności korzystania z komponentów oferowanych przez Buildera dzięki wystarczające jest wygenerowanie aplikacji konsolowej. Aby wygenerować projekt takiej aplikacji należy po uruchomieniu środowiska Borland C++ Builder wykonać następujące kroki:
1. Z menu głównego wybrać kolejno *File>New>Other...*
2. W wyświetlonym oknie wybrać *Console Wizard* a w kolejnym oknie wystarczy jeśli będzie zaznaczona opcja *C++* oraz *Console Application*.
3. Po wciśnięciu OK, wyświetli się okno dialogowe zapisu plików z kodem (domyślnie Unit1.cpp) oraz projektu (domyślnie Project1.bpr), które najlepiej zapisać w osobnym katalogu utworzonym specjalnie dla projektowanej aplikacji.
4. We właściwościach projektu *Project > Options* w zakładce *Directories/Conditionals* należy dodać ścieżki do katalogów z plikami nagłówkowymi do katalogu z bibliotekami dla BC++.

a. W polu *Include Path* dodać wpisy:
C:\Program Files\OpenCV\cv\include
C:\Program Files\OpenCV\cvaux\include

C:\Program Files\OpenCV\cxcore\include
C:\Program Files\OpenCV\otherlibs\highgui
C:\Program Files\OpenCV\otherlibs\cvcam\include

b. W polu *Library Path* należy dodać:
C:\Program Files\OpenCV\bcblib

5. Do głównego pliku programu należy dodać potrzebne pliki nagłówkowe:

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

6. Następnie należy wskazać lokalizację bibliotek. W tym celu należy z menu głównego wybrać: *Project* > *Add to Project*.
7. W oknie wyboru plików zmienić filtr rozszerzeń na *.lib i przejść do katalogu gdzie znajdują się wygenerowane wcześniej biblioteki "..\OpenCV\bcblib" i zaznaczyć odpowiednie biblioteki (odpowiadające dołączonym plikom nagłówkowym czyli w tym przypadku highgui.lib, cv.lib i cxcore.lib).
8. Ze względu na możliwą nie pełną kompatybilność z kompilatorem firmy Borland należy w we właściwościach Projektu, *Project* > *Options*, w zakładce *Advanced Compiler* zaznaczyć w polu *Source* opcję *MFC Compatibility*.
9. Tak skonfigurowany projekt należy zapisać (może się on przydać jako szablon projektu wykorzystującego OpenCV).
10. W pliku głównym programu można teraz zacząć pisać właściwy kod programu (np. taki jak w przykładzie poniżej).

Jeśli aplikacja ma wykorzystywać systemowe komponenty graficznego interfejsu użytkownika bądź komponenty środowiska BC++ Builder to do projektu aplikacji generowanego przez IDE (środowisko programistyczne) trzeba dodać odpowiednie elementy wykonując kroki od 4 do 10.

Przykład kodu wyświetlającego obraz wczytany z pliku. Tekst jaśniejszy został wygenerowany automatycznie, ciemniejszy należy wpisać samodzielnie.

```
//---------------------------------------------------------------
#pragma hdrstop
#include <highgui.h>
#include <cxcore.h>
#include <cv.h>
//---------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
    IplImage *img= cvLoadImage("E:\\lena.bmp",0);
    cvNamedWindow("Image", 1);
    cvShowImage("Image",img);
    cvWaitKey(0);
    cvReleaseImage(&img);
    return 0;
}
//---------------------------------------------------------------
--
```

Uwaga: Alternatywny szablon aplikacji w BCB wykorzystującego OpenCV dostępny jest na stronie:
http://opencvlibrary.sourceforge.net/C%2B%2BBuilder

Podstawowe funkcje przetwarzania obrazu.

OpenCV jest biblioteką funkcji w języku C++. Dlatego podstawy programowania są przydatne w tworzeniu programów z wykorzystaniem biblioteki. Nie jest jednak konieczna zaawansowana wiedza na temat tego języka. OpenCV posiada własny zestaw funkcji do budowy prostego GUI i komunikacji z użytkownikiem oraz operacji na plikach.
W zakresie przetwarzania obrazów najistotniejszym typem danych jest IplImage, który służy do przechowywania informacji o obrazie oraz pozwala wykonywać wszelkie operacje na nim. IplImage pochodzi z biblioteki Intel Image Processing Library (obecnie Intel Performance Primitives) gdzie zdefiniowany jest jako struktura:

```
typedef struct _IplImage
  {
    int  nSize;         /* sizeof(IplImage) */
    int  nChannels;     /*  kanały barw, większość funkcji OpenCV obsługuje 1,2,3 lub 4
                        kanały */
    int  depth;         /* liczba bitów na piksel: IPL_DEPTH_8U, IPL_DEPTH_8S,
                        IPL_DEPTH_16U,
                        IPL_DEPTH_16S, IPL_DEPTH_32S, IPL_DEPTH_32F and
                        IPL_DEPTH_64F */
    int  dataOrder;     /* 0 – przeplot kanałów barwnych, 1 – osobne kanały barw.
                         cvCreateImage tworzy tylko kanały z przeplotem */
    int  origin;        /* 0 – początek układu współrzędnych w lewym górnym rogu,
                        1 – w prawym dolnym rogu */
    int  width;         /* szerokość obrazu w pikselach */
    int  height;        /* wysokość obrazu w pikselach */
    struct _IplROI *roi;/* fragment obrazu do przetwarzania */
    struct _IplImage *maskROI; /* musi być równe NULL w OpenCV */
    int  imageSize;     /* rozmiar obrazu w bajtach  (=image->height*image->widthStep)*/
    char *imageData;  /* wskaźnik do danych obrazowych */
    int  widthStep;   /* Rozmiar wiersza obrazu w bajtach */
    ...
  }
  IplImage;
```

Zawartość skrypty do ćwiczeń.

Pokazano tu jedynie fragment struktury ponieważ OpenCV nie wykorzystuje wszystkich jej elementów.
Pokazywane w dalszej części skryptu programy są w większości przypadków kompletnym kodem wykonującym opisywane zadanie. Pominięto jedynie dołączane pliki nagłówkowe które omówiono w rozdziale o konfiguracji środowiska i tworzeniu projektu programu. W przykładowych programach elementy biblioteki OpenCV które pojawiają się w skrypcie po raz pierwszy są napisane wytłuszczoną czcionką. W dalszych częściach skryptu kod aplikacji jest ograniczony jedynie do istotnych dla jego działania elementów i może nie zawierać fragmentów powtarzających się we wcześniejszych ćwiczeniach, np. wczytywania pliku lub tworzenia i wyświetlania okna z obrazem. Za każdym razem jest to zaznaczone w opisie ćwiczenia lub komentarzach w kodzie programu. Dłuższy kod programu może być także dzielony na fragmenty, co także zaznaczono w opisie.

Pełny kod każdego z opisywanych tu ćwiczeń jest dostępny w wersji elektronicznej w formie projektu dla VC++ . Nazwa projektu dla każdego z przykładów jest podana na końcu każdego z ćwiczeń. Na końcu każdego z ćwiczeń zamieszczono także polecenia do samodzielnego wykonania. Wykonanie zadanych ćwiczeń w większości przypadków polega na zmodyfikowaniu kodu programu z danego ćwiczenia przy użyciu informacji z opisu do zrealizowanych wcześniej przykładów.

Wczytywanie i wyświetlanie obrazu

Podstawowe operacje to oczywiście wczytywanie i wyświetlanie obrazu:

```
int _tmain(int argc, _TCHAR* argv[])
{
    IplImage *img= cvLoadImage("E:\\lena.bmp",0);
    cvNamedWindow("Image", 1);
    cvShowImage("Image",img);
    cvWaitKey(0);
    cvReleaseImage(&img);
    return 0;
}
```

Program którego kod pokazano wyżej jest aplikacją konsolową. Jego uruchomienie spowoduje wyświetlenie konsoli systemowej. Funkcje interfejsu, tworzenie i obsługa okien jest realizowana przez funkcje OpenCV (oczywiście można też inaczej).

```
IplImage *img= cvLoadImage("lena.bmp",0);
```

Ciąg instrukcji który tworzy odpowiednią zmienną typu IplImage (dokładniej wskaźnik do niej) i wczytuje obraz z pliku. Funkcja cvLoadImage jako pierwszy argument pobiera łańcuch znaków z nazwą pliku (lub pełną ścieżką i nazwą jeśli plik nie jest tam gdzie program), drugi argument wymusza wczytanie obrazu jako monochromatyczny: 0, lub wczytuje jako obraz barwny: 1, albo też w oryginalnym formacie: -1.

```
cvNamedWindow("Image", 1);
```

Funkcja zdefiniowana w pliku highgui.h zawierającym m.in. funkcje do obsługi interfejsu graficznego. Pierwszy argument to łańcuch znaków reprezentujący nazwę okna (to co podano jako argument dla funkcji cvNamedWindow). Drugi argument decyduje czy okno ma być dopasowane do rozmiaru obrazu – 1, czy obraz dopasowany do domyślnego rozmiaru okna – 0;

```
cvShowImage("Image",img);
```

Wyświetla obraz we wskazanym oknie. Pierwszy argument to łańcuch znaków z nazwą okna, natomiast drugi to nazwa zmiennej z obrazem.

```
cvWaitKey(0);
```

Zatrzymuje wykonywanie programu i czeka na wciśnięcie klawisza. Jeśli argument jest równy 0 to czeka w nieskończoność. Jeśli większy od zera to oznacza on liczbę milisekund oczekiwania na klawisz.

Jak można się domyślić wszystkie funkcje z biblioteki mają przedrostek cv, natomiast wszystkie typy zmiennych mają przedrostek Cv (tu jeszcze się nie pojawiły), zdefiniowane stałe zaczynają się od CV_.

Szczegóły działania tych funkcji jak i pozostałych funkcji z biblioteki highgui.h znajdują się na stronie:

http://opencvlibrary.sourceforge.net/HighGui

Kod programu zawiera projekt opcv_1

Do zrobienia:

- Wczytać obraz kolorowy z różnymi argumentami decydującymi o trybie wczytywania i wyświetlić go także modyfikując parametry okna.

Kopiowanie obrazu (i odbicie obrazu)

Klonowanie jest czasem przydatną operacja gdy potrzebne są kopie zmiennych przetwarzanych, lub gdy potrzebujemy zmienną o odpowiednich parametrach (liczba kanałów, rozmiar, głębia kolorów,...), np. po to aby podać ją jako argument dla funkcji.

```
...
int _tmain(int argc, _TCHAR* argv[])
{
    IplImage *img= cvLoadImage("E:\\lena.bmp",0);
    cvNamedWindow("Image", 1);
    cvShowImage("Image",img);

    IplImage *img2 = cvCloneImage(img); // tworzy nową zmienną i
    wpisuje do                                    // niej kopię
    img
    cvFlip(img,img2,1); // odbicie w poziomie

    cvNamedWindow("Fliped Image",1); // tworzę okno
    cvShowImage("Fliped Image", img2); // wyswietlam obraz w oknie
    cvWaitKey(0);
    cvReleaseImage(&img);
    cvReleaseImage(&img2);
    return 0;
}
```

Nowymi funkcjami są tu polecenia **cvCloneImage** i **cvFlip**.

```
IplImage *img2 = cvCloneImage(img);
```

Tworzy kopię img2 zmiennej img.

```
cvFlip(img,img2,1);
```

Odbija obraz określony pierwszym argumentem (tu img) i zapisuje go do zmiennej określonej drugim argumentem (tu img2). Ponieważ funkcja cvFlip wymaga podania jako argumentów zmiennej źródłowej i docelowej oraz parametry tych zmiennych muszą się zgadzać to użycie funkcji cvClone upraszcza całe zadanie. Operacje można wykonać także na tej samej zmiennej wtedy podstawiamy tą samą zmienną jako źródło i wynik lub jako drugi argument podajemy NULL.

```
cvFlip(img,img,1);
```

lub

```
cvFlip(img,NULL,1);
```

Ostatnim argumentem jest kierunek odbicia gdzie 0 oznacza odbicie w pionie, 1 – w poziomie.

Funkcja cvFlip jak i pozostałe funkcje przetwarzania obrazu zdefiniowano w cv.h a szczegółowy ich opis znajduje się na stronie:
http://opencvlibrary.sourceforge.net/CvReference

W tym przykładzie nie ma to wielkiego znaczenia ponieważ zakończenie programu usunie z pamięci wszystkie zmienne. W większych programach konieczne jest kontrolowanie istnienia w pamięci zmiennych i usuwanie ich gdy nie są już potrzebne. Służy do tego funkcja void

cvReleaseImage (IplImage** image). Której argumentem jest wskaźnik do zmiennej przechowującej obraz np.:

```
cvReleaseImage(&img2);
```

spowoduje wyczyszczenie zawartości zmiennej o nazwie img2.

Kod programu zawiera projekt opcv_2

Do zrobienia:

- Odbicie w pionie.

Tworzenie nowej zmiennej dla obrazu i skalowanie obrazu

W przypadku gdy w wyniku przetwarzania otrzymujemy obraz o innych atrybutach, np. podczas skalowania gdy obraz wynikowy ma inne rozmiary nie można wykorzystać kopiowania zmiennej aby uzyskać zmienną dla obrazu wynikowego. W takim przypadku należy stworzyć nową zmienną typu IplImage.
Odpowiedni przykład tworzenia zmiennej i wykorzystania jej do skalowania pokazano w przykładowym kodzie.

```
int _tmain(int argc, _TCHAR* argv[])
{
    IplImage *img= cvLoadImage("E:\\lena.bmp",0);
    cvNamedWindow("Image", 1);
    cvShowImage("Image",img);
    // tworzy pusty obraz
    IplImage *bigimg =
cvCreateImage(cvSize(512,512),IPL_DEPTH_8U,1);
    cvZero(bigimg); // czyszczenie pamięci zajętej przez zmienną
    cvResize(img,bigimg,CV_INTER_CUBIC);

    cvNamedWindow("Resized Image",1); // tworzę okno
    cvShowImage("Resized Image", bigimg); // wyswietlam obraz w
oknie
    cvWaitKey(0);
    cvReleaseImage(&img);
    cvReleaseImage(&img2);
    return 0;
}
```

W tym przypadku konieczne jest stworzenie nowej zmiennej do przechowywania obrazu. Do stworzenia zmiennej, wskaźnika do zmiennej typu IplImage służy funkcja: cvCreateImage o następującej składni:

```
IplImage* cvCreateImage(CvSize size, int depth, int channels );
```

Pierwsza zmienna to rozmiar obrazu. Drugi argument określa sposób kodowania kolorów w obrazie i głębi bitową – można tu podać wartość liczbową (można ją znaleźć w dokumentacji) lub wpisać odpowiednią predefiniowaną stałą:

IPL_DEPTH_8U - unsigned 8-bit integers
IPL_DEPTH_8S - signed 8-bit integers
IPL_DEPTH_16U - unsigned 16-bit integers
IPL_DEPTH_16S - signed 16-bit integers
IPL_DEPTH_32S - signed 32-bit integers
IPL_DEPTH_32F - single precision floating-point numbers
IPL_DEPTH_64F - double precision floating-point numbers

Ostatni argument to liczba kanałów barw w obrazie – może mieć wartości od 1 do 4. Rozmiar obrazu musi być zmienną typu CvSize. Można ją stworzyć za pomocą funkcji cvSize(int szerokość, int wysokość).
Stworzony obraz zostanie później wypełniony wartościami. Czasem zachodzi potrzeba stworzenia pustego obrazu, np. po to aby rysować na nim. W tym celu należy wyczyścić zawartość obrazu za pomocą wykorzystanej tu funkcji `void cvSetZero(CvArr* arr)` której argumentem jest wskaźnik do zerowanej tablicy typu CvArr. Można tu podać jako argument także wskaźnik do zmiennej typu IplImage. Do zmiany rozmiarów obrazu wykorzystano tu funkcję cvResize o następującej składni:

```
void cvResize(const CvArr* src, CvArr* dst, int interpolation);
```

Pierwszy argument to zmienna z obrazem źródłowym. Drugi argument to zmienna z obrazem docelowym. Mogą to być zarówno macierze (jak w definicji funkcji) jak i zmienne typu IplImage (jak w przykładzie). Ostatni argument określa sposób interpolacji. Dostępne są cztery rodzaje interpolacji:
CV_INTER_NN - najbliższego sąsiedztwa,
CV_INTER_LINEAR – dwuliniowa (domyślna)
CV_INTER_AREA – interpolacja na podstawie sąsiednich bloków pikseli.
CV_INTER_CUBIC - sześcienna.

Kod programu zawiera projekt opcv_3

Do zrobienia:

- Pokazać skalowanie z innymi niż CV_INTER_CUBIC argumentem i porównać wyniki

Korekcja jasności

OpenCV posiada zestaw funkcji arytmetycznych które mogą być wykonywane na macierzach, także na macierzach reprezentujących obrazy. Korekcję jasności można wykonać poprzez liniową transformację poziomów jasności (jak w przykładzie poniżej) lub poprzez wyrównywanie histogramu (funkcja cvEqualizeHist). W przykładzie poniżej obraz źródłowy jest argumentem x w wyrażeniu y = a*x+b, gdzie y jest obrazem wynikowym, a – współczynnikiem skalującym, b – określa przesunięcie poziomu zerowego.
Korekcja poziomów jasności (liniowa bądź nie liniowa) powoduje zmianę jasności i kontrastu obrazu.

```
int _tmain(int argc, _TCHAR* argv[])
{
   IplImage *img= cvLoadImage("E:\\lena.bmp",0);
   cvNamedWindow("Image", 1);
   cvShowImage("Image",img);
   IplImage *im_scal = cvCreateImage(cvSize(img->width,img->height),
                                                    IPL_DEPTH_8U,1);
   cvConvertScaleAbs(img, im_scal, 2, 0);
   cvNamedWindow("Scaled image a=2 b=0", 1);
   cvShowImage("Scaled image a=2 b=0", im_scal);
    cvWaitKey(0);
   cvConvertScaleAbs(img, im_scal, 0.5, 0);
   cvNamedWindow("Scaled image a=0.5 b=0", 1);
   cvShowImage("Scaled image a=0.5 b=0", im_scal);
    cvWaitKey(0);
   cvConvertScale (img, im_scal, 1, 128);
```

```
    cvNamedWindow("Scaled image a=1 b=128", 1);
    cvShowImage("Scaled image a=1 b=128", im_scal);
    cvWaitKey(0);
    cvEqualizeHist(img, im_scal);
    cvNamedWindow("Equalized histogram", 1);
    cvShowImage("Equalized histogram", im_scal);
    cvWaitKey(0);
    return 0;
}
```

Funkcja cvConvertScaleAbs wykonuje przekształcenie opisane powyżej dając w wyniku obraz monochromatyczny z 8 bitową głębią kolorów – lub ściślej w tym przypadku, poziomów jasności obrazu monochromatycznego (IPL_DEPTH_8U) i ma następujący format:

```
cvConvertScaleAbs(const CvArr* src, CvArr* dst, double
scale=1, double shift=0);
```

Argumenty `src` i `dst` oznaczają odpowiednio obraz źródłowy i docelowy. Argument scale to oczywiście współczynnik skalujący („a" w równaniu funkcji), natomiast shift to składnik stały - przesunięcie (b w równaniu funkcji).
Można do tego wykorzystać także funkcję cvConvertScale która nie zmienia typu macierzy wynikowej na reprezentację za pomocą 8 bitów (dzieki czemu można zmienną na obraz wynikowy stworzyć przez kopiowanie zmiennej oryginalnej), lub za pomocą funkcji cvLUT która wykorzystuje Look-Up-Table do wykonania transformacji.
Ostatnia z użytych funkcji cvEqualizeHist wykonuje operację wyrównywania histogramu. Funkcja ta `cvEqualizeHist(const CvArr* src, CvArr* dst)` przyjmuje jako argument wskaźnik do obrazu źródłowego src i wynikowego dst.

Kod programu zawiera projekt opcv_4

Do zrobienia:

- Jak przy użyciu funkcji cvConvertScaleAbs zrobić negatyw obrazu.
- Dobrać argumenty funkcji tak aby uzyskać zwiększenie kontrastu obrazu.
- Zastąpić funkcję cvConvertScaleAbs funkcją cvConvertScale i porównać wynik.

Korekcja jasności obrazów barwnych

Korekcja jasności i kontrastu obrazów barwnych może być wykonana w ten sam sposób. Dla obrazów barwnych konieczne jest aby obraz wynikowy miał tą samą liczbę kanałów barwnych. W przypadku obrazów barwnych nie można także bezpośrednio wykorzystać funkcji wyrównywania histogramu ponieważ obsługuje ona tylko obrazy z jednym kanałem (monochromatyczne).

```
int _tmain(int argc, _TCHAR* argv[])
{
    IplImage *img= cvLoadImage("E:\\lena_k.bmp",1);
    cvNamedWindow("Image", 1);
    cvShowImage("Image",img);
    IplImage *im_scal = cvCreateImage(cvSize(img->width,img->height),
                                      IPL_DEPTH_8U,img->nChannel);
    cvConvertScaleAbs(img, im_scal, 2, 0);
    cvNamedWindow("Scaled image a=2 b=0", 1);
    cvShowImage("Scaled image a=2 b=0", im_scal);
    cvWaitKey(0);
```

```
    cvConvertScaleAbs(img, im_scal, 0.5, 0);
    cvNamedWindow("Scaled image a=0.5 b=0", 1);
    cvShowImage("Scaled image a=0.5 b=0", im_scal);
    cvWaitKey(0);
    cvConvertScale (img, im_scal, 1, 128);
    cvNamedWindow("Scaled image a=1 b=128", 1);
    cvShowImage("Scaled image a=1 b=128", im_scal);
    cvWaitKey(0);
    return 0;
}
```

Aby zachować zgodną liczbę kanałów w zmiennej źródłowej i docelowej wartość ta została odczytana ze zmiennej źródłowej img->nChannels i podstawiona do funkcji cvCreateImage.

Kod programu zawiera projekt opcv_5

Do zrobienia:

- Jak przy użyciu funkcji cvConvertScaleAbs zrobić negatyw obrazu.
- Zastąpić funkcję cvConvertScaleAbs funkcją cvConvertScale i porównać wynik.

Manipulacja przestrzeniami barw

Obrazy barwne są kodowane za pomocą trzech składowych np. RGB (Red Green Blue), HLS (Hue, Lightness, Saturation) czyli barwa, jasność, nasycenie, czy stosowanym w kodowaniu plików JPEG standardzie YCrCb. OpenCV pozwala na rozkładanie obrazu na składowe i przekształcanie między wymienionymi tu standardami i jeszcze kilkoma innymi formatami. Poniższy przykład programu pokazuje rozkład obrazu kolorowego na składowe RGB i ich wyświetlenie w osobnych oknach. Poniższy kod pokazuje zarówno wyświetlanie składowych jak i konwersję przestrzeni barw dlatego został rozbity na dwie części:

```
int _tmain(int argc, _TCHAR* argv[])
{
    IplImage *imgc= cvLoadImage("E:\\lena_k.bmp",1);
    cvNamedWindow("Image", 1);
    cvShowImage("Image",imgc);

    // tworzenie obrazów dla poszczególnych kanałów.
    IplImage *kanal1 =
cvCreateImage(cvSize(256,256),IPL_DEPTH_8U,1);
    IplImage *kanal2 = cvCreateImage(cvSize(256,256),IPL_DEPTH_8U,1);
    IplImage *kanal3 = cvCreateImage(cvSize(256,256),IPL_DEPTH_8U,1);

    cvCvtPixToPlane(imgc,kanal1,kanal2,kanal3,0);

    cvNamedWindow("Kanal B",1);
    cvShowImage("Kanal B",kanal1);
    cvNamedWindow("Kanal G",1);
    cvShowImage("Kanal G",kanal2);
    cvNamedWindow("Kanal R",1);
    cvShowImage("Kanal R",kanal3);
    cvWaitKey(0);
    cvDestroyWindow("Kanal B");
    cvDestroyWindow("Kanal G");
    cvDestroyWindow("Kanal R");
    // cdn...
```

Należy zwrócić uwagę że w openCV zamieniona jest kolejność kanałów barw podstawowych na BGR (niebieski, zielony, czerwony). Przy wczytywaniu obrazów kolorowych należy pamiętać aby podać odpowiedni parametr określający tryb otwarcia w funkcji cvLoadImage. W tym przypadku drugi parametr powinien być równy 1. Obrazy reprezentujące poszczególne kanały są monochromatyczne z 8 bitową głębią kolorów dlatego przed rozdzieleniem obrazu na składowe należy przygotować odpowiednie zmienne korzystając z funkcji cvCreateImage. Należy przy tym pamiętać aby wielkość obrazów była taka sama jak obrazu źródłowego (w przykładzie podano ją od razu liczbami). Do wydzielenia obrazów poszczególnych kanałów barwnych służy funkcja `cvCvtPixToPlane(const CvArr* src, CvArr* dst0, CvArr* dst1, CvArr* dst2, CvArr* dst3)`, w której argument src jest obrazem źródłowym a argumenty dst0, dst1,... są wskaźnikami do zmiennych reprezentujących kolejne kanały obrazu wydzielone w wyniku działania funkcji. Poszczególne obrazy są następnie wyświetlone w analogiczny sposób jak w poprzednich przykładach.
Dla zachowania przejrzystości aplikacji przed następnym krokiem usunięto okna wyświetlające obrazy kanałów za pomocą funkcji cvDestroyWindow(const char* name), która jako argument wymaga podania nazwy okna.
Jeśli chcemy przekształcić obraz do innej przestrzeni barw np. YCrCb należy wykorzystać funkcję cvCvtColor, której wykorzystanie ilustruje dalszy fragment kodu programu:

```
//...cd
   IplImage *img2 = cvCloneImage(imgc);
   cvCvtColor(imgc,img2,CV_RGB2YCrCb); //konwersja do YCrCb
   cvNamedWindow("kolory",1);
   cvShowImage("kolory",img2);
   cvCvtPixToPlane(img2,kanal1,kanal2,kanal3,0);
   cvNamedWindow("Kanal Y",1);
   cvShowImage("Kanal Y",kanal1);
   cvNamedWindow("Kanal Cr",1);
   cvShowImage("Kanal Cr",kanal2);
   cvNamedWindow("Kanal Cb",1);
   cvShowImage("Kanal Cb",kanal3);
   cvWaitKey(0);
   return 0;
}
```

Funkcja `cvCvtColor( const CvArr* src, CvArr* dst, int code )` jako argumenty wymaga podania wskaźnika do obrazu źródłowego, wskaźnika do obrazu wynikowego o tej samej co źródłowy strukturze (tu wykorzystano do jego stworzenia funkcje kopiowania zmiennej obrazowej) oraz kod reprezentujący rodzaj konwersji. Kod konwersji może przyjmować wartości które tworzy się w następujący sposób:
CV_<przestrzeń_wejściowa>2<przestrzeń_wyjściowa>, gdzie <przestrzeń_wejściowa> i <przestrzeń_wyjściowa> mogą przyjmować wartości: RGB, BGR, XYZ, YCrCb, HSV, HLS, Lab, Luv, BayerBG, BayerGB,... Na przykład konwersja między z przestrzeni RGB do HSV będzie miała kod CV_RGB2HSV i analogicznie konwersja odwrotna będzie miała kod CV_HSV2RGB.

Kod programu zawiera projekt opcv_6

Do zrobienia:

- Wykonać konwersję do przestrzeni HLS, wyświetlić poszczególne kanały, wykonać np. progowanie jednego lub więcej kanałów i pokazać obraz wynikowy.

Filtracja

Filtracja w dziedzinie przestrzennej może być wykonana za pomocą zdefiniowanej funkcji cvSmooth, która realizuje filtry „wygładzające" - dolnoprzepustowe, filtru górno przepustowego typu laplacea cvLaplace, filtrów do detekcji krawędzi cvSobel. Może być także przeprowadzona za pomocą funkcji cvFilter2D, która pozwala na użycie dowolnej zdefiniowanej maski filtru. W większości przypadków wymienione funkcje mogą być stosowane

W pierwszym fragmencie kodu programu pokazano przykłady filtracji dolno i górno przepustowej za pomocą pojedynczych funkcji cvSmooth i cvLaplace.

```
int _tmain(int argc, _TCHAR* argv[])
{
   IplImage *img= cvLoadImage("E:\\lena_k.bmp",1);
   cvNamedWindow("Image", 1);
   cvShowImage("Image",img);

   IplImage *imgf = cvCloneImage(img);
   cvSmooth(img, imgf, CV_BLUR, 5, 5);

   cvNamedWindow("Rozmywanie", 1);
   cvShowImage("Rozmywanie",imgf);

   cvSmooth(img, imgf, CV_GAUSSIAN, 5,5);
   cvNamedWindow("Rozmywanie Gaussa", 1);
   cvShowImage("Rozmywanie Gaussa",imgf);

   cvWaitKey(0);
// usunąć niepotrzebne okna funkcjami cvDestroyWindow(...)
// cdn...
```

Funkcja filtrująca dolnoprzepustowo (rozmywanie) `cvSmooth( const CvArr* src, CvArr* dst, int smoothtype=CV_GAUSSIAN, int param1=3, int param2=0, double param3=0 )` wymaga jako argument podania wskaźników do obrazu źródłowego src i wynikowego dst, następnie typu wykonywanej filtracji smoothtype oraz parametrów tej filtracji określonych trzema kolejnymi wartościami param1, param2 i param3. Argument smoothtype może przyjmować wartości:

CV_BLUR filtr rozmywający o wielkość param1 x param2;

CV_GAUSSIAN filtr rozmywający Gaussa o rozmiarach param1 x param2 lub jeśli param1 i 2 są równe 0, to filtr definiowany jest na podstawie parametru sigma funkcji gaussowskiej określonej przez param3. Jeśli param 2 i 3 są równe 0 to filtr ma rozmiary param1 x param2.

CV_MEDIAN filtr medianowy o masce filtru param1 x param1

CV_BILATERAL filtr rozmywający gaussa zachowujący krawędzie o rozmiarach 3x3 piksele z dodatkowymi parametrami filtru określanymi przez param1 i param2.

W przypadku filtrów górno przepustowych należy zwrócić uwagę na głębie bitową przetwarzanego obrazu i odpowiednie skalowanie wartości wynikowego obrazu przed jego wyświetleniem, co pokazano w dalszej części kodu programu:

```
// cd...

   IplImage *imgf_16s = cvCreateImage(cvSize(img->width, img->height),
                                      IPL_DEPTH_16S,1);
```

```
    IplImage *imgf_8u = cvCreateImage(cvSize(img->width,
                                      img->height),IPL_DEPTH_8U,1);
    cvLaplace(img, imgf_16s, 1);
    cvConvertScaleAbs(imgf_16s,imgf_8u,1,0);

    cvNamedWindow("Filtracja Laplacea", 1);
    cvShowImage("Filtracja Laplacea",imgf_8u);

    cvSobel(img,imgf_16s,1,0,3);
    cvConvertScaleAbs(imgf_16s,imgf_8u,1,0);
    cvNamedWindow("Filtracja Sobela - pion", 1);
    cvShowImage("Filtracja Sobela - pion",imgf_8u);

    cvSobel(img,imgf_16s,0,1,3);
    cvConvertScaleAbs(imgf_16s,imgf_8u,1,0);
    cvNamedWindow("Filtracja Sobela - poziom", 1);
    cvShowImage("Filtracja Sobela - poziom",imgf_8u);

    cvWaitKey(0);

    return 0;
}
```

Ze względu na możliwe wyjście poza zakres dopuszczalności dla 8 bitowej głębi barw w wyniku zastosowania funkcji cvLaplace czy cvSobel, wymagają one zmiennej dla obrazu wynikowego o głębi bitowej typu IPL_DEPTH_16S (16 bitowe liczby ze znakiem - tu zmienna imgf_16s). Można potem obraz wynikowy przekształcić z powrotem do IPL_DEPTH_8U (8 bitowe liczby bez znaku - tu zmienna imgf_8u), za pomocą funkcji cvConvertScaleAbs z parametrami takimi jak w przykładowym kodzie.
Funkcja cvLaplace(const CvArr* src, CvArr* dst, int aperture_size=3) wyznacza laplasjan obrazu czyli dokonuje filtracji obrazu funkcją o charakterystyce będącej odwróconą krzywą gausowską. Jako argumenty wymaga podania wskaźnika do zmiennej źródłowej src i wynikowej dst oraz rozmiaru filtru (domyślnie jest to wartość 3).
Kolejna funkcja `cvSobel( const CvArr* src, CvArr* dst, int xorder, int yorder, int aperture_size=3 )`, jest wykorzystywana do wyznaczania krawędzi. Jako argumenty wymaga podania wskaźnika do zmiennej źródłowej src i wynikowej dst oraz parametrów xorder i yorder które określają które kierunki krawędzi zostaną uwypuklone w obrazie (dla xorder =0 yorder=1 będą to krawędzie poziome, jeżeli zamienimy wartości będą to krawędzie pionowe), oraz rozmiaru filtru (domyślnie jest to wartość 3). Należy przy tym pamiętać że rozmiar filtru musi być większy niż wartości xorder i yorder.

Kod programu zawiera projekt opcv_7

Funkcja cvFilter2D pozwala wykonać filtrację obrazu przy użyciu dowolnie zdefiniowanej maski filtru przestrzennego. Definicję filtru uśredniającego 3x3 oraz filtracje pokazuje poniższy kod:

```
int _tmain(int argc, _TCHAR* argv[])
{
    IplImage *img = cvLoadImage("E:\\lena_k.bmp",1);
    IplImage *im_filter = cvCloneImage(img);
    cvNamedWindow("Orginal image", 1);
```

```
    cvShowImage("Orginal image", img);

    CvMat* filtr = cvCreateMat(3,3,CV_32FC1);
    cvSet2D(filtr,0,0,cvRealScalar(0.11));
    cvSet2D(filtr,0,1,cvRealScalar(0.11));
    cvSet2D(filtr,0,2,cvRealScalar(0.11));
    cvSet2D(filtr,1,0,cvRealScalar(0.11));
    cvSet2D(filtr,1,1,cvRealScalar(0.11));
    cvSet2D(filtr,1,2,cvRealScalar(0.11));
    cvSet2D(filtr,2,0,cvRealScalar(0.11));
    cvSet2D(filtr,2,1,cvRealScalar(0.11));
    cvSet2D(filtr,2,2,cvRealScalar(0.11));
    cvFilter2D(img, im_filter, filtr, cvPoint(-1,-1));
    cvNamedWindow("Filtered image", 1);
    cvShowImage("Filtered image", im_filter);

    cvWaitKey(0);
    return 0;
}
```

Maskę filtru definiuje się w postaci macierzy prostokątnej (tu o rozmiarach 3x3). Służy do tego zmienna typu CvMat. Macierz tworzy się używając funkcji `CvMat* cvCreateMat( int rows, int cols, int type )`, która zwraca wskaźnik do macierzy. Jako argumenty należy podać liczbę wierszy – rows i kolumn – cols, oraz typ wartości przechowywanych w macierzy (tu CV_32FC1 – 32 bitowe liczby zmiennoprzecinkowe). Następnie należy ustawić wartość każdego z elementów macierzy przy użyciu funkcji `void cvSet2D(CvArr* arr, int idx0, int idx1, CvScalar value)`. Jako argumenty należy podać wskaźnik do macierzy której element będzie ustawiany arr, pozycje w macierzy poprzez określenie wiersza idx0 i kolumny idx1 oraz wartość typu CvScalar. Odpowiedni typ zmiennej został tu uzyskany przy użyciu funkcji cvRealScalar z argumentem będącym wymaganą wartością liczbową.
Przy użyciu sdefiniowanej macierzy można przefiltrować obraz używając funkcji `cvFilter2D( const CvArr* src, CvArr* dst, const CvMat* kernel, CvPoint anchor=cvPoint(-1,-1))`. Funkcja wymaga podania jako argumentu wskaźników do zmiennej źródłowej i docelowej src i dst (może to być ta sama zmienna), wskaźnika do macierzy definiującej maskę filtru i pozycję filtrowanego punktu obrazu względem maski filtru. Domyślna wartość dla maski 3x3 wskazuje na jej środkowy element. Pozycja ta jest określona jako wartości typu CvPoint. W tym przykładzie stworzona została za pomocą funkcji cvPoint z argumentami –1, –1 określającymi właśnie ten centralny punkt maski.

Kod programu zawiera projekt opcv_7a

Do zrobienia:

- Zaprojektować filtry górno-przepustowy i pokazać przykład jego działania z maską:

$$f1 = \begin{matrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{matrix} \qquad f2 = \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix}$$

Segmentacja obrazu - Progowanie

Progowanie jest najprostszą techniką segmentacji obrazu. W przypadku segmentacji na podstawie jasności polega ono na przypisaniu pikselom obrazu o jasności większej od wartości progowej maksymalnej jasności (kolor biały) natomiast pikselom ciemniejszym od wartości progowej jasności zerowej (kolor czarny). Funkcja progowania dostępna w OpenCV pozwala na inny sposób progowania obrazu w zależności o podanych argumentów funkcji. Progowanie może być wykonane na obrazach o monochromatycznych i barwnych.

```
int _tmain(int argc, _TCHAR* argv[])
{
   IplImage *img = cvLoadImage("E:\\lena.bmp",0);
   cvNamedWindow("Orginal image", 1);
   cvShowImage("Orginal image", img);
   IplImage *im_thres = cvCreateImage(cvSize(img->width, img-
>height),                                    IPL_DEPTH_8U,1);
   cvThreshold(img,im_thres, 128, 255, CV_THRESH_BINARY);
   cvNamedWindow("Thresholded image", 1);
   cvShowImage("Thresholded image", im_thres);
   cvWaitKey(0);
   return 0;
}
```

Do progowania obrazu służy w OpenCV funkcja `cvThreshold( const CvArr* src, CvArr* dst, double threshold, double max_value, int threshold_type )`, w której pierwszy i drugi argument to odpowiednio obraz źródłowy src i docelowy dst. Trzeci argument threshold to wartość progowa. Argument kolejny max_value to maksymalna jasność w obrazie. Przy określaniu tych dwóch wartości należy brać pod uwagę głębię bitową obrazu (dla IPL_DEPTH_8U wartość maksymalna to 255, natomiast dla IPL_DEPTH_16S to 65535). Ostatni argument określa sposób progowania. Możliwe jest przeprowadzenie tej operacji na pięć różnych sposobów które określane są odpowiednimi stałymi:

CV_THRESH_BINARY: dst(x,y) = max_value, jeśli src(x,y)>threshold jeśli nie to: dst(x,y) =0,
CV_THRESH_BINARY_INV: dst(x,y) = 0, if src(x,y)>threshold jeśli nie to: dst(x,y) =max_value,
CV_THRESH_TRUNC: dst(x,y) = threshold, jeśli src(x,y)>threshold jeśli nie to: dst(x,y) =src(x,y),
CV_THRESH_TOZERO: dst(x,y) = src(x,y), jeśli src(x,y)>threshold jeśli nie to: dst(x,y) =0,
CV_THRESH_TOZERO_INV: dst(x,y) = 0, jeśli src(x,y)>threshold jeśli nie to: dst(x,y) =src(x,y).

Oprócz progowania ze stałym progiem OpenCV zawiera także funkcję w której wartość progowa dobierana jest w zależności od lokalnych cech obrazu – cvAdaptiveThreshold.

Kod programu zawiera projekt opcv_8

Do zrobienia:

- Wykonać progowanie z innymi wartościami określającymi rodzaj operacji niż CV_THRESH_BINARY

Interakcja z użytkownikiem

OpenCV oprócz narzędzi do przetwarzania obrazów wyposażono w skromne narzędzie do kontroli procesu przetwarzania. Można oczywiście podawać parametry przetwarzania z konsoli tekstowej – raczej niepraktyczne w przypadku np. przetwarzania sygnału z kamery, lub wykorzystać kontrolki systemowe. W prostych zastosowaniach można wykorzystać „suwaki" wbudowane w bibliotekę OpenCV.

Użycie suwaków (Trackbar) do zmiany parametrów przetwarzania

Do obsługi suwaków zaimplementowano w OpenCV kilka funkcji i można z nich korzystać na różne sposoby Do obsługi suwaka służą funkcje cvCreateTrackbar, cvGetTrackbarPos i cvSetTrackbarPos. W każdym przypadku konieczne jest w pierwszym kroku stworzenie trackabara przy użyciu funkcji:
cvCreateTrackbar(const char* trackbar_name, const char* window_name, int* value, int count, CvTrackbarCallback on_change). Kolejne argumenty tej funkcji oznaczają:
trackbar_name – łańcuch znaków oznaczający nazwę suwaka
window_name – łańcuch znaków oznaczający nazwę okna w którym umieszczony jest suwak
value – wskaźnik do liczby typu int określającej pozycję suwaka oraz definiującej początkową jego pozycję,
count – maksymalna pozycja suwaka (minimum jest równe 0).
on_change – wskaźnik do funkcji która będzie uruchamiana przy każdej zmianie położenia suwaka. Musi ona zwracać argument typu void. Może być także równy NULL, wtedy żadna funkcja nie będzie wykonywana.
W poniższym przykładzie wykorzystano ostatni argument funkcji cvCreateTrackbar i cały proces przetwarzania zamieszczono nie w głównej funkcji programu a w funkcji treshold(...). Aby program działał prawidłowo na początku kodu aplikacji należy zadeklarować zmienne do przechowywania obrazu źródłowego (zmienna typy IplImage) i wartości progowej (w tym przypadku może to być zmienna typu int).

```
IplImage *img;
int value=128;

void treshold(int a)
{
   IplImage *im_thres = cvCreateImage(cvSize(img->width, img->height),
                                      IPL_DEPTH_8U,1);
   cvThreshold(img,im_thres, value, 255, CV_THRESH_BINARY);
   cvShowImage("Thresholded image", im_thres);
   cvReleaseImage(&im_thres);
}

int _tmain(int argc, _TCHAR* argv[])
{
   img= cvLoadImage("E:\\lena.bmp",0);
   cvNamedWindow("Image", 1);
   cvShowImage("Image",img);
   cvNamedWindow("Thresholded image", 1);

   cvCreateTrackbar("próg","Thresholded image",&value,255,treshold);

   cvWaitKey(0);
   cvReleaseImage(&img);
   return 0;
}
```

W tym przypadku suwak przypisano do okna w którym wyświetlany jest obraz po progowaniu dlatego też odpowiednie okno musi być stworzone funkcją cvNamedWindow przed stworzeniem suwaka. Można także stworzyć okno w którym wyświetlany będzie tylko suwak lub suwaki.

Można także wykorzystać funkcję cvGetTrackbarPos do odczytywania pozycji, bez definiowania funkcji wykonującej przetwarzanie. Trzeba wtedy przed wykonanie operacji odczytać położenie suwaka (może on być wtedy użyty jako przełącznik. Można także programowo (cvSetTrakckbarPos) ustawić pozycję i wykorzystać suwak jako wskaźnik postępu jakiegoś procesu.

Kod programu zawiera projekt opcv_14

Do zrobienia:

- Zmodyfikować aplikację tak aby suwaki były w osobnym oknie.
- Zastosować suwaki do innego rodzaju operacji, np. skalowania jasności, lub powiększenia.

Przetwarzanie sekwencji wideo.

W openCV zaimplementowano możliwość przechwytywania obrazu z pliku z sekwencją wideo jak i z kamery. Użytkownik nie musi znać szczegółów dotyczących sposobu kompresji i kodowania filmów ponieważ funkcje OpenCV wykorzystują zainstalowane w systemie sterowniki.

Odczyt i przetwarzanie obrazów z sekwencji wideo.

OpenCV zawiera zestaw funkcji potrzebnych do obsługi przetwarzania wideo, czyli otwierania i zamykania plików oraz przechwytywania strumienia danych oraz pojedynczych klatek sekwencji. Przetwarzanie sekwencji wideo polega przetwarzaniu kolejno pobieranych z sekwencji klatek. W uproszczeniu należy w tym celu stworzyć zmienną zawierającą strumień danych z pliku wideo, następnie pobrać klatkę z obrazu z sekwencji, skopiować ją do zmiennej przechowującej obraz na której będzie można wykonać przetwarzani (tu będzie to zmienna typu IplImage). Uzyskany w ten sposób obraz można teraz przetwarzać lub wyświetlać.
W przykładzie poniżej pokazano kod aplikacji który tylko wyświetla sekwencję wideo aby pokazać kolejność działań przy obsłudze plików wideo.

```
int _tmain(int argc, _TCHAR* argv[])
{
// tworzę strumień video z pliku
   CvCapture* capture = cvCaptureFromAVI("E:\\wideo1.avi");
   IplImage* img = 0;
   cvNamedWindow("seq", 1);

   while(cvGrabFrame(capture))
   {
      img=cvRetrieveFrame(capture);     // kopiuję klatkę do obrazu
      cvShowImage("seq",img);           // wyświetlanie
      cvWaitKey(100);       // czekam 100ms przed następną klatką
   }
   cvWaitKey(0);    // czekam na wciśniecie klawisza aby
zakończyć program
   cvReleaseCapture(&capture);
   return 0;
}
```

W pierwszym kroku tworzony jest wskaźnik do strumienia wideo z pliku. Odpowiednia zmienna typu CvCapture tworzona jest funkcją `cvCaptureFromFile( const char* filename )`, która jako argument wymaga podania ścieżki do pliku z filmem. Aby odtworzyć cały film należy w pętli przechwytywać kolejne klatki sekwencji. W warunku pętli while sprawdzane jest czy funkcja cvGrabFrame zwraca wartość inną niż zero i jeśli tak jest to wykonywany ciąg instrukcji wewnątrz pętli. Funkcja `cvGrabFrame( CvCapture* capture )` zwraca wartość różną od zera jeśli uda się pobrać klatkę ze strumienia do, którego wskaźnik jest podany jako argument funkcji. Klatka ta nie jest od razu dostępna ze względu na możliwości uzyskania jak najszybszego przechwytywania obrazu. Wewnątrz pętli, obraz z przechwyconej klatki jest zapisywany do zmiennej typu IplImage (tu zmienna img), która jest następnie wyświetlana w utworzonym wcześniej oknie. Kopiowanie obrazu ze strumienia do zmiennej typu IplImage wykonuje się przy użyciu funkcji `cvRetrieveFrame( CvCapture* capture )`, której argumentem jest wskaźnik do strumienia wideo a zwracana jest ostatnio przechwycona klatka tego strumienia.
Aby aplikacja zdążyła wyświetlić obraz przed przejściem do następnego cyklu w pętli konieczne jest dodanie opóźnienia przy użyciu funkcji cvWaitKey. W tym przypadku jako argument podano wartość 100 co oznacza 100ms oczekiwania na wciśnięcie klawisza i przejście do kolejnego wykonania instrukcji z pętli.
Po zakończeniu korzystania ze strumienia wideo należy go zamknąć funkcją `cvReleaseCapture( CvCapture** capture )`, która jako argumentu wymaga podania wskaźnika do strumienia wideo.
Aby przetwarzać kolejne klatki sekwencji należy wewnątrz pętli dodać operacje wykonujące przetwarzanie obrazu:

```
//...
while(cvGrabFrame(capture))
{
   img=cvRetrieveFrame(capture);   // kopiuję klatkę do obrazu

   cvConvertScale(img, img, 2.5, 0);    // skalowanie jasności

   cvShowImage("seq",img);          // wyświetlanie
   cvWaitKey(10);
}
//...
```

W pokazanym tu fragmencie, do pętli w której pobierane są kolejne klatki filmu, pomiędzy pobieranie obrazu a wyświetlanie go, wstawiono funkcję skalowania jasności i dopiero jej wynik został wyświetlony. W podobny sposób można wykonać inne operacje. Jeżeli operacje przetwarzania obrazu wymagają wykorzystania dodatkowych zmiennych to powinny one być utworzone przed pętlą.

Kod programu zawiera projekt opcv_15

Do zrobienia:

- Zmodyfikować program do odtwarzania sekwencji wideo tak aby przetwarzał on kolejne klatki wykonując dowolną operację filtracji.