ApacsCCLib

Prowadzący:
dr inż. Krzysztof Krawiec

Autorzy:
Piotr Garbacik piotr.garbacik@gmail.com
Adrian Włodarczak punish3r@interia.pl

1. Kompilacja biblioteki.

Przed pierwsza kompilacja należy wejść do katalogu *ApacsCCLib* i wpisac '*make dirs'* w celu utworzenia niezbędnych folderów (jeżeli komuś potrzebne dodatkowe rzeczy, jak np. *viewer,* należy zrobić to samo w odpowiednim katalogu - w tym przypadku *NiftiViewer*). Następnie w głównym folderze projektu należy wpisać *'make apacscclib'* aby zbudować bibliotekę (lub *'make'* aby zbudować cały projekt).

Oczywiście wcześniej należy posiadać zainstalowane i skompilowane biblioteki *Nifti, OpenCV* (*viewer* wymaga *GTK* aby wyświetlać okno), *Zlib* (aby otwierać skompresowane pliki *nifti*).

2. Biblioteka *ApacsCCLib*.

Aby korzystać z biblioteki należy dołączyć do projektu nagłówek *FxApacsCCLib.h*. Wszystkie elementy zawierają się w przestrzeni nazw *APACS_NAMESPACE*, także albo używamy dyrektywy '*using namespace APACS_NAMESPACE'* (lub lepiej deklaracji), albo kwalifikujemy bezpośrednio, np. '*APACS_NAMESPACE::CxNiftiImage'*. Biblioteka definiuje różne typy i symbole, dla zapewnienia maksymalnej przenaszalności. Warto zapoznać się z nagłówkiem *FxVcWin32Ver.h* (lub *FxGnuGVer.h* - są tam te same symbole, ale inaczej zmapowane).

3. Klasa *CxNiftiImage*.

Klasa ta obudowuje *nifti_image* z biblioteki *Nifti*, dzięki czemu pozbywamy się problemu ręcznego zarządzania pamięcią. Klasa obsługuje kopiowanie oraz przypisywanie. Dzięki przeciążonym operatorom '*operator nifti_image**' oraz '*operator const nifti_image**' klasa *CxNiftiImage* może zostać użyta wszędzie tam, gdzie wymagany jest wskaźnik na strukturę *nifti_image*.* Jeżeli potrzebujemy odwołać się do pola w/w struktury poprzez obiekt klasy *CxNiftiImage* należy do tego celu wykorzystać przeciążony operator ->.

Aby otworzyć obraz *Nifti* należy przekazać ścieżkę do pliku do konstruktora *CxNiftiImage* lub posłużyć się funkcją *Open*. W razie niepowodzenia obie metody rzucają wyjątek klasy *std::runtime_error*. Istnieje także konstruktor przyjmujący za parametr wskaźnik na strukturę *nifti_image*.* Należy pamiętać, ze obiekt klasy *CxNiftiImage* staje się wtedy właścicielem obrazu, więc nie należy go zwalniać ręcznie za pomocą funkcji *nifti_image_free*.

Funkcja *ToCvImage* konwertuje obraz *Nifti* do formatu wykorzystywanego przez *OpenCV.* Obsługiwane są obrazy 3D i 4D. Parametr *InAxis* określa oś wzdłuż której wykonamy „cięcie", a parametr *InDepth* z której warstwy wyciąć obrazek. *InVolume* wykorzystywane jest w obrazach 4D do określenia woluminu czasowego. Wartości parametrów *InDepth* oraz *InVolume* są automatycznie przycinane aby nie przekroczyć zakresu (np. przy próbie wycięcia 37 warstwy przy dostępnych 25). W razie niepowodzenia funkcja rzuca wyjątek klasy *std::runtime_error*.

4. Klasa *CxCCShapeProperties*.

Zakres funkcjonalności tej klasy polega na przetwarzaniu obrazu 2d wygenerowanego przy użyciu wtyczki *CC Segmentation* w celu uzyskania jego właściwości. Aktualnie dostępne cechy:

- Długość konturu,
- Pole powierzchni,
- Spatial moments (10 wartości),
- Central moments (7 wartości),
- Central normalized moments (7 wartości),
- Hu invariants (7 wartości),

Własności obliczane są raz, przy tworzeniu instancji klasy. Pobieranie poszczególnych cech nie wymaga zatem ponownych obliczeń. Użytkownik może także określić jednostkę - osobno dla osi x oraz y - jakiej odpowiada jeden piksel obrazu. Przykładowo - podanie 3.0 oznacza, że 1 piksel obrazu wejściowego odpowiada 3 jednostkom rzeczywistym.

5. Klasa *CxParameters*.

W zamierzeniu klasa ta ma na celu ujednolicenie i usprawnienie zarządzania parametrami odbieranymi od i przekazywanymi do *ScriptRunnera*. *ScriptRunner* przekazuje do wtyczki parametry postaci *Klucz=Wartość* oddzielone znakami końca linii (*'\n'*). Klasa *CxParameters* parsuje taki wejściowy ciąg, poczym umożliwia dostęp do poszczególnych par za pomocą metod *Get* i *Set*. Ponieważ metody te są szablonowe, w praktyce możliwe jest pobieranie dowolnych typów wartości, o ile istnieje ich konwersja do łańcucha tekstowego. Dla typu *bool*, oprócz wartości 0 dla fałszu oraz dowolnej innej wartości dla prawdy, możliwe jest podawanie ciągów *True* oraz *False* (w tym wypadku wielkość liter nie ma znaczenia).

Aby przekazać parametry do *ScriptRunnera* dostępne są dwa rozwiązania:

- Wysłać je bezpośrednio do strumienia wyjściowego (*std::cout << OutputParams*),
- Powiązać je z odpowiednim urządzeniem wyjściowym (*CxOutputDevice*) a następnie ustawić *SetDumpOnDestroy* na *True*. Powiązanie parametrów z urządzeniem wyjścia możliwe jest także na etapie tworzenia obiektu.

W przypadku tej drugiej opcji globalnie dostępne są dwa urządzenia wyjścia:

- CxOutputDeviceNull GNullDev,
- CxOutputDeviceStdOut GStdOutDev,

Pierwsze po prostu „pochłania" strumień, drugie - wysyła go na standardowe wyjście. Funkcjonalność ta ma na celu zwolnienia użytkownika z pamiętania o przesłaniu parametrów do strumienia lub ułatwienia całej tej operacji jeżeli np. ścieżka przebiegu programu jest trudna do przewidzenia.

Uwaga: wielkość liter w nazwach parametrów ma znaczenie!

6. Kompilacja własnego programu.

Należy pamiętać o dołączeniu biblioteki *ApacsCCLib* (*'-l apacscclib.a'*) oraz bibliotek *Nifti* (*'-lniftiio –lznz'*), *Zlib* (*'-lz'*) i *OpenCV ('-lcv -lhighgui'*). Najlepiej przejrzeć Makefile jednego z projektów.

Warto także zapoznać się z implementacjami *NiftiViewer* oraz *CCMeasurement* jako przykładami wykorzystania biblioteki.