Strona tytułowa

POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY
INSTYTUT STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ

PRACA MAGISTERSKA
na kierunku ELEKTROTECHNIKA

Mariusz SZABŁOWSKI
Nr imm. 182172

Rok. akad. 2005/2006
Warszawa, 27.10.2005

ANALIZA PORÓWNAWCZA WYBRANYCH BIBLIOTEK DO PRZETWARZANIA OBRAZÓW

Zakres pracy:

1. *Wprowadzenie*
2. *Wybrane elementy przetwarzania obrazów*
3. *Opis i analiza wybranych bibliotek*
4. *Wykonanie testów i analiza wyników*
5. *Podsumowanie i wnioski*

Podpis i pieczątka
Kierownika Zakładu Dydaktycznego

Opiekun naukowy:
Dr inż. Witold Czajewski

Termin wykonania: 2006
Praca wykonana i zaliczona pozostaje własnością Instytutu i nie będzie zwrócona wykonawcy

Pragnę złożyć serdeczne podziękowania
Panu dr inż. Witoldowi Czajewskiemu za pomoc
i cenne uwagi podczas realizacji tej pracy.

1. Wprowadzenie

Niniejsza praca ma zapełnić lukę, jaka panuje na rynku publikacji poruszających zagadnienie programowania z wykorzystaniem darmowych bibliotek do przetwarzania obrazów, ma być pomocna przy wyborze oprogramowania, jak również programowaniu aplikacji przetwarzania obrazu i wizji komputerowej. W pracy zawarto opis mający na celu porównanie wyselekcjonowanych przez autora bibliotek, które uznano za godne uwagi. Dołączono również przykłady realizacji poszczególnych zagadnień w danych bibliotekach. Praca zawiera także porównanie możliwości bibliotek oraz testy opisane w taki sposób, aby czytelnik mógł wybrać bibliotekę odpowiednią dla postawionych zadań.

Przetwarzanie obrazów jest dziedziną od lat zagłębianą przez rzesze naukowców, studentów na całym świecie. W tym czasie powstał szereg bibliotek. Jedne są bardziej rozbudowane, dedykowane dla bardziej wymagających użytkowników, inne prostsze dla osób mniej wymagających lub też bardziej wyspecjalizowane w jednej konkretniej dziedzinie. Biblioteki te są również napisane w różnych językach programowania, pracują pod różnymi systemami operacyjnymi oraz wymagają różnych kompilatorów. Bibliotek takich w Internecie można znaleźć bardzo dużą liczbę. Wybór bibliotek opisanych w pracy został oparty na kilku przesłankach. Przede wszystkim biblioteki te powinny być darmowe o otwartym kodzie źródłowym, co pozwala korzystać z biblioteki każdej osobie bez konieczności wykupowania licencji, umożliwia również dodawanie własnego kodu źródłowego do biblioteki, a także prześledzenie sposobu realizacji zaimplementowanych już w bibliotece algorytmów. Podstawową wadą darmowych bibliotek jest brak kompletnej dokumentacji, tutoriali, przykładowych kodów źródłowych pozwalających szybko zrozumieć system pracy z daną biblioteką. Często też prace nad darmowymi bibliotekami zostają przerwane przez ich twórców, a nowe wersje ukazują się w dużym odstępie czasu. Biblioteki komercyjne natomiast są bardzo dobrze udokumentowane i zawierają zawsze aktualne narzędzia, jednakże fakt, że ich kod jest zamknięty lub wymagają nieraz dokupienia specjalnych urządzeń, na których będą one pracować (np. kart grafiki), często eliminuje je z prac badawczych wykonywanych na uczelniach wyższych. Kolejnym wymaganiem postawionym bibliotekom był język programowania, w jakim zostały one napisane. Skupiono się wyłącznie na bibliotekach napisanych w C/C++. Nasuwa się więc pytanie, dlaczego

C/C++ a nie inne języki? Przede wszystkim w językach tych stworzono najwięcej narzędzi do przetwarzania obrazu oraz charakteryzują one się większą wydajnością od innych języków. Następnym wymaganiem była specjalizacja bibliotek; powinny być ogólnego przeznaczenia a nie realizować wąskie zagadnienie wykorzystywane w danej dziedzinie naukowej, na przykład takiej jak medycyna czy geografia. Ponieważ na uczelniach takich jak Politechnika Warszawska wykonywane są różne prace projektowe wymagające szerokiej gamy opracowanych algorytmów, dlatego też zawężono poszukiwania do bibliotek ogólnego przeznaczenia. Podczas wyszukiwania bibliotek zwrócono również uwagę na to, żeby w jak największym stopniu były one niezależne systemowo, a przede wszystkim działały pod systemami Windows oraz Linux (dwoma najpopularniejszymi systemami operacyjny na świecie). Do porównania i analizy wybrano następujące biblioteki: OPENCV, IPL98, LTI-LIB, CIMG. W krótki sposób opisano również jedną bibliotekę komercyjną IPP, która może współpracować z biblioteką OPENCV.

1.1. Układ pracy

Niniejsza praca magisterska składa się z części informatycznej (w postaci programów testowych służących do porównania biblotek oraz przykładowych programów pokazujących realizację poszczególnych zagadnień w danych bibliotekach) oraz części opisowej (intrukcje instalacji bibliotek, opis funcjonalności bibliotek, sposobu pracy z bibliotekami, wybranych zagadnień przetwarzania obrazów).

Rozdział 2 wprowadza w tematykę przetwarzania obrazów. Zakres opisanych zagadnień przetwarzania obrazów został dostosowany do możliwości wyselekcjonowanych bibliotek.

Rozdział 3 opisuje i analizuje wybrane biblioteki pod względem funkcjonalności (struktury danych, funkcje do przetwarzania obrazów), dostępnej dokumentacji, sposobu instalacji, dostępności pod danymi systemami operacyjnymi, kompilatorami. Rozdział ten jest szczególnie pomocny przy programowaniu aplikacji przetwarzania obrazu i wizji komputerowej.

Rozdział 4 przedstawia zaproponowane przez autora testy, opisuje ich realizację oraz podsumowuje wyniki tych testów. Testy składały się z dwóch części. Pierwsza z nich to testy poszczególnych, wspólnych funkcji bibliotek i porównanie ich w szczególności pod względem wydajności. Druga część to napisanie aplikacji do

rozpoznawania znaków drogowych i porównanie na jej podstawie funkcjonalności, łatwości implementacji poszczególnych funkcji, przejrzystości kodu tworzonej aplikacji, wydajności w poszczególnych bibliotekach.

1.2. Różne podejścia do porównywania

Co to jest biblioteka? Biblioteka jest zbiorem funkcji, które zostały stworzone, aby można było z nich korzystać w wielu programach [32]. Ma na celu ułatwić programowanie w danej dziedzinie (w tym przypadku dziedziną jest przetwarzanie obrazów).

Większość z bibliotek nie jest kompatybilna ze sobą, dlatego też użytkownik musi dokonywać wyboru między nimi. Istnieje kilka kryteriów porównania bibliotek w celu wybrania najbardziej odpowiedniej postawionym zagadnieniom. Podstawowymi kryteriami decydującymi o wyborze bibliotek jest ich dostępność pod danymi systemami operacyjnymi oraz kompilatorami. Bardziej znaczącymi kryteriami wpływającymi na wybór są funkcjonalność (liczba dostępnych funkcji), możliwości rozwojowe-potencjał (ang. capabilities) biblioteki oraz szybkość wykonywania poszczególnych operacji.

W większości przypadków użytkownicy wybierają biblioteki, które dostarczają najwięcej funkcji, dostępnych bezpośrednio bez wykonywania jakiejkolwiek pracy. Np. takimi funkcjami są: rotacja albo kopiowanie obrazu, czytanie obrazu w formacie JPEG, konwersja miedzy kilkoma przestrzeniami kolorów. Funkcje takie muszą być również wydajne. Na funkcjonalność biblioteki składają się nie tylko dostarczone funkcje, ale także typy danych, które są bezpośrednio dostępne, takie jak struktury do reprezentacji obrazów w pamięci. Ponieważ najważniejszym czynnikiem doboru biblioteki przez użytkowania jest jej maksymalna funkcjonalność, jednym z podejść sprostania wymaganiom użytkownika, jest dostarczenie jak największej liczby funkcji oraz typów danych, których użytkownik może potrzebować przy projektowaniu aplikacji przetwarzania obrazu i wizji komputerowej. Związku z czym biblioteki zwykle mają zaimplementowane:

- obsługę kilku kanałów obrazów
- wbudowane wszystkie typy danych (bool, short, int, float, double . . .) oraz wektory złożone z tych typów
- jak największą liczbę funkcji

Bywają sytuacje, kiedy użytkownik potrzebuje dostosować bibliotekę do szczególnego przypadku użycia, a jak wiadomo wszystkie funkcjonalności nie mogą być dodane do każdej biblioteki, ponieważ czas potrzebny do zbudowania takiej biblioteki byłby zbyt duży, a nawet gdyby było to możliwe, biblioteka taka byłaby zbyt duża, co również stanowiłoby problem.

Możliwości rozwojowe, potencjał sprawiają, że biblioteka staje się łatwiejsza lub trudniejsza w użyciu. Jeżeli dana funkcja nie jest dostępna, w większości przypadków nowa funkcja może zostać dodana, jednakże w niektórych przypadkach może to być ograniczone przez brak możliwości rozwojowych biblioteki. Dlatego też należy zwrócić uwagę na potencjał bibliotek. Użytkownik może chcieć użyć własnych typów w istniejących już w bibliotece narzędziach lub użyć istniejących typów w realizacji nowego algorytmu, dlatego też biblioteka powinna mu dostarczać dużej elastyczności działania. Potencjał biblioteki łączy się z ograniczeniami narzuconymi podczas jej projektowania. Niektóre biblioteki dostarczają łatwych i eleganckich sposobów do ich rozbudowy bez specjalistycznej wiedzy technicznej o nich. Zaprojektowanie bardzo dobrej biblioteki jest trudnym zadaniem i często równa się z podejmowaniem wybory między jej możliwościami rozwoju a:

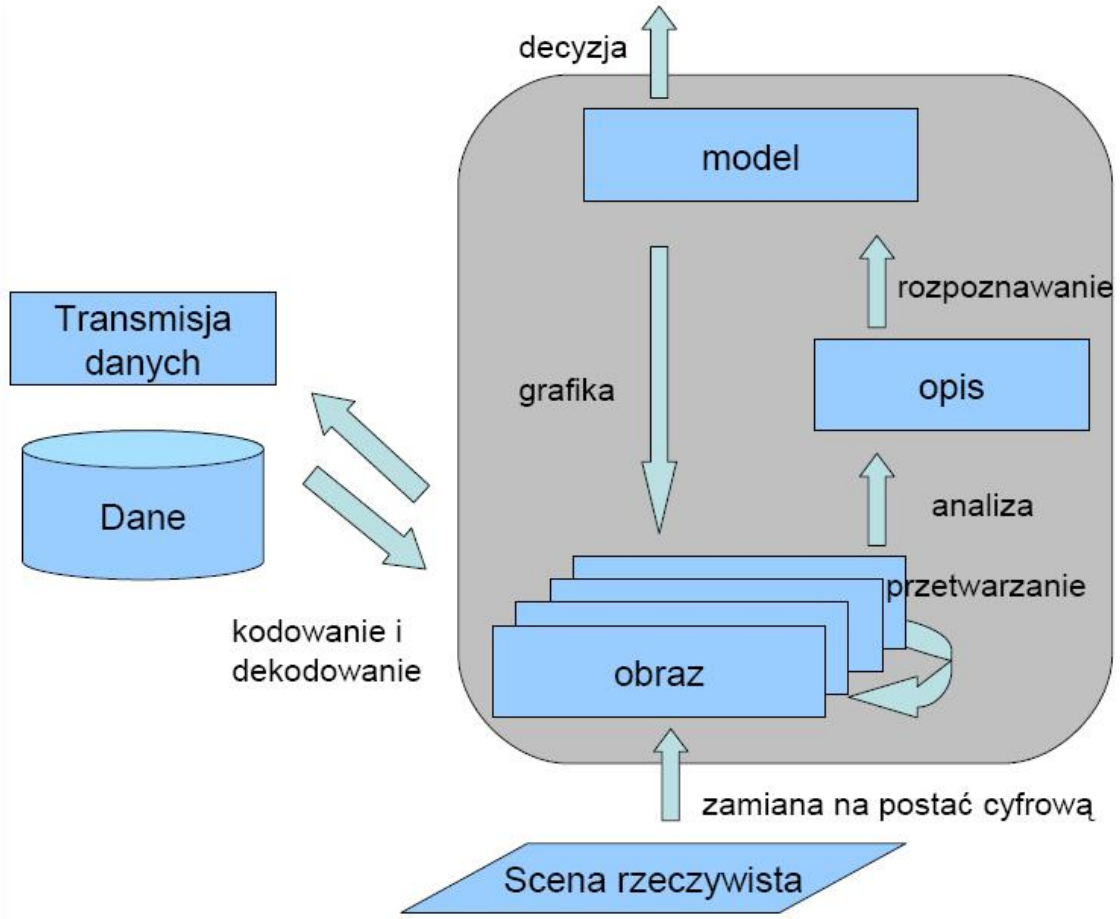
- złożonością jądra biblioteki (łatwiej jest napisać bibliotekę, która pracuje na obrazach jednokanałowych niż obrazach n-kanałowych)
- złożonością dodawania nowych algorytmów, struktur danych
- wydajnością
- środowiskiem programistycznym

2. Wybrane elementy przetwarzania obrazów

Przetwarzanie obrazu i widzenie komputerowe zyskują w ostatnich latach coraz większą popularność, znajdują zastosowanie w coraz większej ilości dziedzin (Rysunek 1). Stosowane są w automatyce, robotyce, w procesach przemysłowych do ich nadzorowania, w sterowaniu ruchem ulicznym, w medycynie, w metalurgii, w astronomii, w astrofizyce, w procesach artystycznych takich jak tworzenie teledysków oraz wielu innych dziedzinach.

Rysunek 1. Dziedziny zastosowań przetwarzania obrazów.

Przetwarzanie obrazu jest dziedziną nauki, której zadaniem jest wyodrębnienie pewnych cech obrazu, na podstawie których można coś w miarę jednoznacznie stwierdzić na temat zawartości danego obrazu np.: dokonać lokalizacji pewnych obiektów, wyznaczyć ich cechy, zlikwidować szum, rozjaśnić obraz, dokonać segmentacji itp. Za tymi drobnymi, wydawałoby się operacjami, kryje się potężny aparat matematyczny w postaci ogromnej ilości mniej lub bardziej skomplikowanych wzorów popartych szeroko teorią.



Rysunek 2. Formy przetwarzania obrazów cyfrowych.

2.1. Filtracja

Jednym z zasadniczych celów przetwarzania obrazu jest polepszenie jego jakości, dlatego też podstawową i jedną z najważniejszych operacji przetwarzania obrazu jest filtracja [40]. Często zdarza się, że odbierany przez system wizyjny obraz jest zniekształcony. Przyczyn może być wiele np. złe oświetlenie obrazu, wadliwie nastawiona ostrość lub dodatkowy szum. Odpowiednio wykonana filtracja jest krokiem milowym na drodze dążącej do wyodrębnienia istotnych cech, informacji z obrazu, do których normalnie nie ma dostępu. Ogólnie filtracja polega na wykorzystaniu informacji z otoczenia danego piksela. Dysponując wiedzą o bezpośrednich sąsiadach można z pewnym prawdopodobieństwem stwierdzić czy dany piksel jest rzeczywistym obrazem, czy też powstał w wyniku działania szumu. Promień badania sąsiedztwa piksela może być różnych rozmiarów, związany jest z definicją maski. Gdy rozważany jest filtr,

którego promień przetwarzania jest równy jeden powstanie macierz (tzw. maska) o wymiarze 3x3, której piksel środkowy jest badanym pikselem (Rysunek 3).

$$\begin{bmatrix} otoczenie & otoczenie & otoczenie \\ otoczenie & badany\ piksel & otoczenie \\ otoczenie & otoczenie & otoczenie \end{bmatrix}$$

Rysunek 3. Przykład maski do filtra dowolnego typu.

Liczba wierszy i kolumn maski jest najczęściej nieparzysta, aby istniał element centralny. Maska ta może zawierać współczynniki, przez które należy przemnożyć wartość jasności danego piksela w celu zwiększenia ich ważności (Rysunek 4). Maska nie musi być kwadratowa. Może też mieć kształt prostokąta, krzyża, rombu, okręgu. Jednakże zmniejszanie liczby pikseli w otoczeniu badanego piksela jest równoznaczne ze zmniejszeniem informacji, na podstawie której wnioskuje dany filtr, co w konsekwencji osłabia jego działanie.

$$\begin{bmatrix} A*otoczenie & B*otoczenie & C*otoczenie \\ D*otoczenie & E*badany\ piksel & F*otoczenie \\ G*otoczenie & H*otoczenie & I*otoczenie \end{bmatrix}$$

Rysunek 4. Przykład maski dla filtra dowolnego typu, gdzie: A, B, C, D, E, F, G, H, I – dowolne współczynniki.

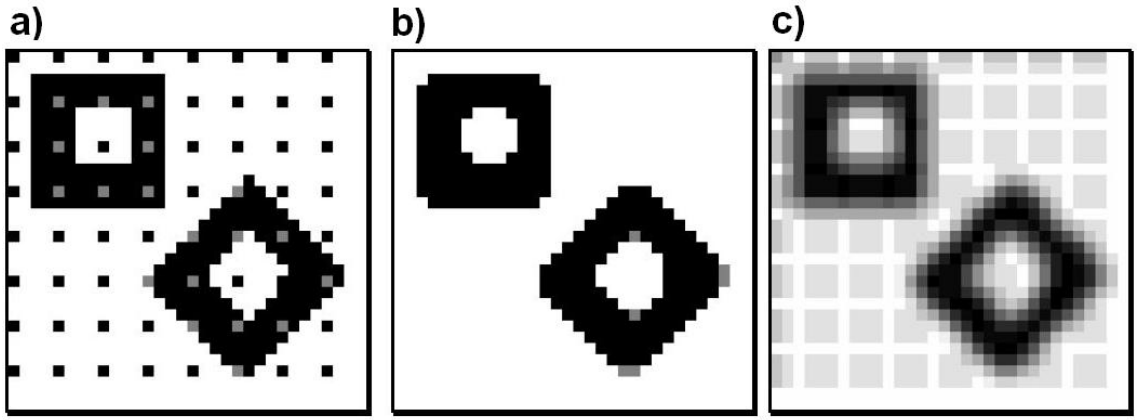
2.1.1. Filtry konwolucyjne

Stosowanie operacji konwolucji zwanej także splotem funkcji jest jedną z najpopularniejszych metod filtracji obrazu [40]. Konwolucją dwóch ciągłych funkcji *f* i *g* nazywana jest operacja:

$$f * g = \int_{\infty}^{\infty} \mathrm{f}\,(\mathrm{x} - \mathrm{t})\,\mathrm{g}\,(\mathrm{t})\,\mathrm{dt}$$

W przypadku funkcji dyskretnych (np. o obrazie można powiedzieć, że jest funkcją dyskretną), konwolucję można prosto opisać w sposób algorytmiczny. Ponadto konwolucja posiada właściwości matematyczne zbliżone do operacji mnożenia, co umożliwi dalsze uproszczenie niektórych operacji. W przypadku filtracji obrazów, pierwszą funkcją jest obraz, natomiast drugą funkcję określa się mianem jądra konwolucji i od jej postaci zależy jak będzie wyglądał obraz po filtracji. W przypadku

dyskretnego splotu jądro (maska filtra) jest kwadratową tablicą zawierającą współczynniki filtra. Dla zobrazowania operacji opisano konwolucję obrazu monochromatycznego, gdzie każdy piksel jest zapamiętany na jednym bajcie a jego wartość odpowiada jasności. Na filtrowany piksel nakładana jest maska filtru (jądro splotu) w taki sposób, żeby element centralny pokrywał się z filtrowanym pikselem. Operacja filtrowania polega na pomnożeniu współczynników filtru przez wartość znajdujących się pod nimi pikseli a następnie dodaniu wyników poszczególnych mnożeń (Rysunek 5). Ponieważ wynik może przekraczać 255, czyli maksymalną dopuszczalną wartość dla piksela, wynik należy podzielić przez współczynnik normalizujący. Współczynnik taki możne być łatwo policzony, jest to po prostu suma wszystkich współczynników filtra. Jeżeli suma współczynników filtra wynosi 0, wtedy współczynnik normalizujący ma wartość 1.Wynik operacji jest zapisywany do nowego obrazu, jest to bardzo istotne, ponieważ przy zapisie do tego samego obrazu, filtrując kolejne piksele używane byłyby do obliczeń piksele przefiltrowane, co dałoby nieprzewidywalne rezultaty.



Rysunek 5. Sztuczny obraz (a), po filtracji medianowej (b) i po filtracji konwolucyjnej (c).

2.1.2. Filtry medianowe

Filtracja medianowa [40] jest transformacją nieliniową. Działanie tego filtru polega na umieszczeniu wszystkich punktów znajdujących się pod maską na posortowanej liście. Następnie z listy wybierany jest element środkowy i on jest zapisywany do nowego obrazu. Zaletą tego filtru jest dobre usuwanie szumu przy zachowaniu ostrości krawędzi (Rysunek 7), bardzo skutecznie zwalcza wszelkie lokalne szumy (Rysunek 6), nie powodując ich „rozmywania" na większym obszarze, co jest

niestety przypadłością wszystkich filtrów konwolucyjnych (Rysunek 5). Eliminuje on także zamazywanie drobnych detali obrazu, które występują w metodach wygładzania. Filtry medianowe nie powodują zniekształceń funkcji skokowej i rosnącej oraz funkcji pulsacyjnej.

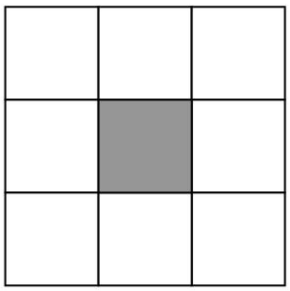
Rysunek 6. Usuwanie zakłóceń filtrem medianowym (wysokości zaznaczonych na rysunku słupków symbolizują wartości stopnia jasności przetwarzanych pikseli).

Rysunek 7. Wpływ filtru medianowego na brzegi obiektu.

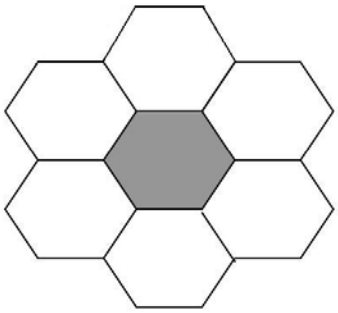
2.2. Morfologia

Wraz z rozwojem i znaczną popularyzacją przetwarzania obrazów w różnych dziedzinach życia, przekształcenia morfologiczne [20] stały się jednymi z najważniejszych operacji w komputerowych systemach analizy obrazu. Odpowiednio kombinowane w zestawy pozwalają na najbardziej złożone operacje, związane z analizą kształtu elementów obrazu i ich wzajemnego położenia.

Fundamentalnym pojęciem przekształceń morfologicznych jest tzw. „element strukturalny obrazu”(Rysunek 8 i Rysunek 9). Jest to pewien wycinek obrazu z wyróżnionym jednym punktem nazywanym „punktem centralnym”.



Rysunek 8. Element strukturalny - koło o promieniu jednostkowym, na siatce kwadratowej.



Rysunek 9. Element strukturalny – koło o promieniu jednostkowym, na siatce heksagonalnej.

Każde przekształcenie morfologiczne można określić następującym algorytmem: Element strukturalny jest przemieszczany po całym obrazie i dla każdego punktu obrazu wykonywana jest analiza punktów obrazu i elementu strukturalnego, przy założeniu, że badany punkt obrazu jest punktem centralnym elementu strukturalnego. Następnie w każdym punkcie obrazu następuje sprawdzenie, czy rzeczywista konfiguracja pikseli obrazu w otoczeniu tego punktu zgodna jest z wzorcowym elementem strukturalnym. W przypadku wykrycia zgodności wzorca pikseli obrazu i szablonu elementu strukturalnego, następuje wykonanie pewnej operacji na badanym punkcie (np. zmiana jasności) według funkcji przynależności związanej z danym elementem strukturalnym. Wynik przekształcenia przedstawiony jest w postaci nowego obrazu.

Opierając się na podanych powyżej definicjach i właściwościach można zdefiniować wiele przekształceń morfologicznych. Operacje morfologiczne posiadają pewną ważną cechę odróżniającą je od wszystkich innych przekształceń i filtracji obrazów. Mianowicie przekształcają tylko tę część punktów obrazu, których otoczenie jest zgodne z elementem strukturalnym.

2.2.1. Erozja

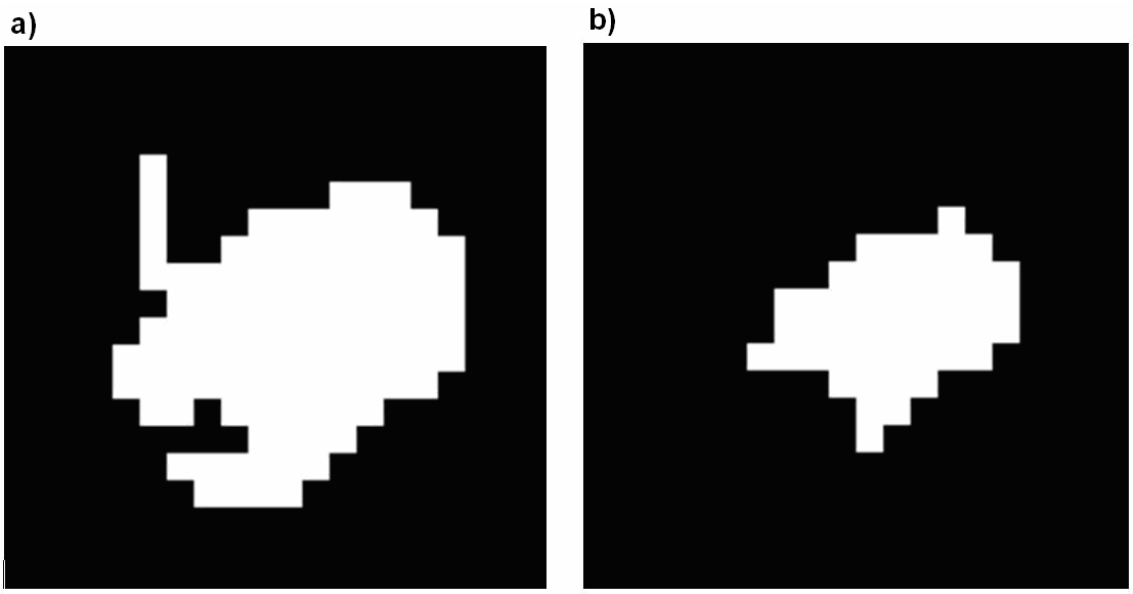
Erozja jest podstawowym przekształceniem morfologicznym. Erozja obrazu A przez element strukturalny B jest zdefiniowana jako przecięcie przesunięć zbioru A o wszystkie elementy b, przy czym b € B

$$A \ominus B = \bigcap_{b \in B} A_{-b}$$

Praktycznie erozje wyznacza się, przemieszczając element strukturalny B po wszystkich elementach obrazu A. Wartość elementu, dla którego jest wyznaczana erozja, jest

wówczas iloczynem logicznym elementu strukturalnego B i części obrazu A, który jest nim przysłonięty. Erozje obrazów z gradacją poziomów szarości można traktować jako poszukiwanie lokalnego minimum, przy czym obszar poszukiwań jest wyznaczony przez wielkość i kształt elementu strukturalnego.

Erozja jest fundamentalnym przekształceniem morfologicznym cechującym się niezmienniczością, monotonicznością oraz antyekstensywnością (gdy początek układu współrzędnych należy do zbioru B). Erozja dokonuje generalizacji obrazu. Odizolowane, drobne obszary zostają usunięte. Brzegi wyróżnionych obszarów zostają wygładzone, ich powierzchnie zostają zmniejszone. Często większe obszary zostają podzielone na mniejsze. Ogólnie spada jasność (nasycenie) obrazu. Na rysunku (Rysunek 10) pokazano przykłady działania operatora erozji dla przykładowego obiektu (duże piksele).



Rysunek 10. Przebieg erozji prostego obrazu, obraz oryginalny (a), obraz po erozji elementem 3x3 (b).

2.2.2. Dylacja

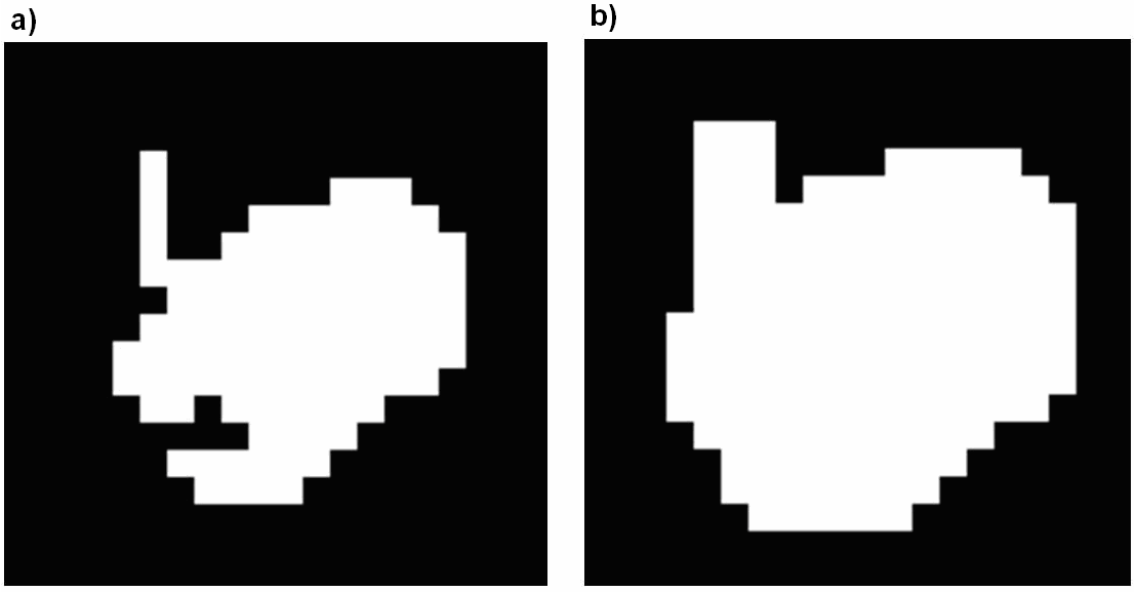
Dylacja obrazu A przez element strukturalny B jest zdefiniowana jako suma przesunięć zbioru A o wszystkie elementy b, przy czym b € B

$$A \oplus B = \bigcup_{b \in B} A_b$$

Praktycznie dylację wyznacza się, przemieszczając element strukturalny B po wszystkich elementach obrazu A. Wartość elementu, dla którego jest wyznaczana dylacja, jest wówczas suma logiczna elementu strukturalnego B i części obrazu A, który

jest nim przysłonięty. Dylację obrazów z gradacja poziomów szarości można traktować jako poszukiwanie lokalnego maksimum, przy czym obszar poszukiwań jest wyznaczony przez wielkość i kształt elementu strukturalnego.

Dylacja jako operacja fundamentalna morfologii matematycznej ma wiele różnych właściwości, m.in. ekstensywność, monotoniczność oraz niezmienniczość względem przesunięcia. Dodatkowo dylacja posiada następujące własności: zamyka małe otwory i wąskie „zatoki", posiada zdolność do łączenia obiektów, które położone są blisko siebie, powoduje zwiększanie powierzchni obiektów jaśniejszych. Na rysunku (Rysunek 11) pokazano przykłady działania operatora dylacji dla przykładowego obiektu (duże piksele).



Rysunek 11. Przebieg dylacji prostego obrazu, obraz oryginalny (a), obraz po dylacji elementem 3x3 (b).

2.2.3. Złożone operacje morfologiczne

Gradient morfologiczny jest operacja, która powstaje jako różnica wyniku operacji dylacji i erozji:

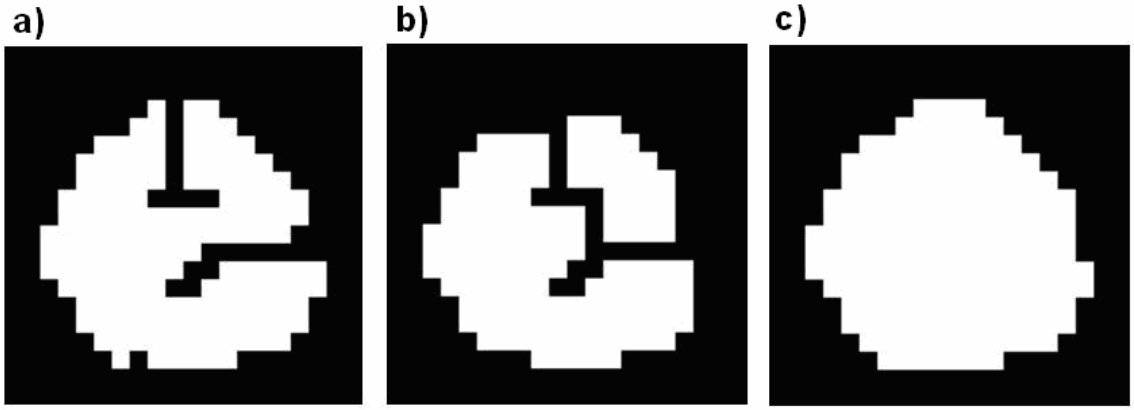
$$\gamma = (A \oplus B) - (A \ominus B)$$

Operacja ta jest stosowana głównie do wykrywania krawędzi występujących w obrazie.

Otwarcie dla klasy obrazów reprezentowanych zbiorami jest operacja powstającą przez złożenie najpierw operacji erozji, a następnie dylacji:

$$A \circ B = [A \Theta B] \oplus B$$

Operacja otwarcia może być wykorzystywana do realizacji prostej segmentacji oraz eliminacji zakłóceń w postaci pojedynczych pikseli. Otwarcie usuwa drobne obiekty i drobne szczegóły, nie zmieniając zasadniczej wielkości figury, może rozłączyć niektóre obiekty z przewężeniami (Rysunek 12).



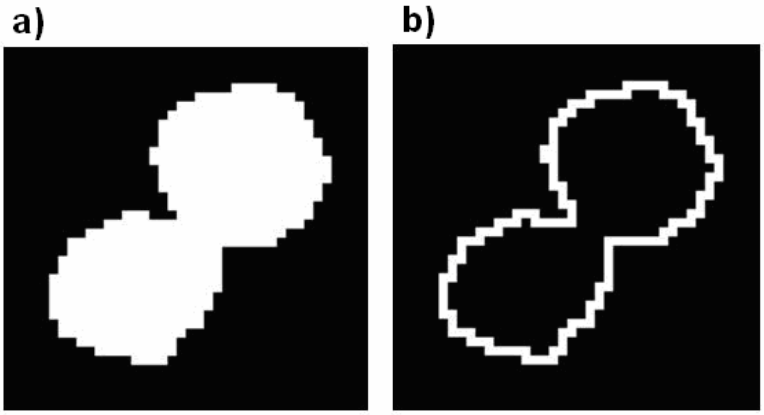
Rysunek 12. Przebieg operacji otwarcia i zamknięcia prostego obrazu, obraz oryginalny (a), obraz po otwarciu (b), obraz po zamknięciu elementem 5x5 (c).

Zamknięcie jest operacja, która powstaje przez złożenie operacji dylacji, a następnie erozji:

$$A \bullet B = [A \oplus B] \Theta B$$

Zamknięcie wypełnia wąskie wcięcia i zatoki oraz drobne otwory wewnątrz obiektu, nie zmieniając wielkości zasadniczej części, może połączyć leżące blisko siebie obiekty (Rysunek 12).

Ścienianie jest operacją mającą na celu wyznaczenie brzegów określonych obszarów. Do realizacji tego przekształcenia można zastosować wiele elementów strukturalnych. Wynikiem ścieniania jest zawsze obraz binarny, obrazem wejściowym jest zazwyczaj również obraz binarny, chociaż nie jest to konieczne. Pewną cechą ścieniania jest fakt, że figura po wykonaniu tej operacji zawiera się w figurze wyjściowej (Rysunek 13).



Rysunek 13. Przykład działania operacji ścieniania dla przykładowego obiektu (duże piksele) , obraz oryginalny (a), obraz po operacji ścieniania (b) .

2.3. Wykrywanie krawędzi

Wykrywanie krawędzi jest bardzo ważnym elementem procesu analizy obrazu [14]. Otrzymanie obrazu w formie wyróżnionych krawędzi jest często wystarczające, a jednocześnie wygodne do przeprowadzenia logicznej interpretacji obrazu. Techniki wykrywania krawędzi mają na celu znalezienie lokalnych nieciągłości w pewnych atrybutach obrazu, które odzwierciedlają granice obiektów znajdujących się na scenie. Nieciągłości takie, a więc i krawędzie, powstają także w wyniku wystąpienia zmian właściwości i powierzchni obiektów, zmian w oświetleniu, cieni itp. Idealna, jednowymiarowa krawędź może więc być zdefiniowana jako nieciągła, skokowa zmiana pewnych atrybutów obrazu. W niżej opisanych metodach atrybutem wykrywania krawędzi jest intensywność. Najczęściej stosowanymi metodami detekcji krawędzi są metody wykorzystujące operatory gradientowe aproksymujące pierwszą i drugą pochodną funkcji jasności obrazu. Działają one na zasadzie splotu kilku sąsiednich pikseli z daną maską w wyniku czego powstaje krawędź.

Detektory bazujące na pierwszej pochodnej funkcji obrazowej

Poniżej przedstawiono maski splotu dla najczęściej stosowanych w przetwarzaniu obrazów operatorów gradientowych (Rysunek 14, Rysunek 15).

a)

1	2	1
0	0	0
-1	-2	-1

dla krawędzi o nachyleniu 0° do poziomu

b)

1	0	-1
2	0	-2
1	0	-1

dla krawędzi o nachyleniu 90°

c)

0	1	2
-1	0	1
-2	-1	0

dla krawędzi o nachyleniu 45°

Rysunek 14. Maski operatora Sobela – a) pozioma, b) pionowa i c) ukośna.

a)

1	1	1
0	0	0
-1	-1	-1

dla krawędzi poziomych

b)

1	0	-1
1	0	-1
1	0	-1

dla krawędzi pionowych

c)

0	1	1
-1	0	1
-1	-1	0

dla krawędzi ukośnych

Rysunek 15. Maski operatora Prewitta – a) pozioma, b) pionowa, c) ukośna.

Detektory bazujące na drugiej pochodnej funkcji obrazowej

Do wydobycia krawędzi z obrazu można wykorzystać także operator Laplace'a (laplasjan), aproksymujący drugą pochodną funkcji jasności. W cyfrowej wersji laplasjan dla dowolnego punktu ma postać:

$$L(x,y) = \left[f(x+1,y) + f(x-1,y) + f(x,y-1) + 4f(x,y) \right]$$

i może być zaimplementowany z pomocą maski (Rysunek 16). Wynik zastosowania operatora jest niezależny od kierunku krawędzi.

0	1	0
1	-4	1
0	1	0

Rysunek 16. Maska operatora Laplace'a.

2.4. Histogram

O tym, jak licznie występują w obrazie punkty o różnych poziomach jasności, można dowiedzieć się z histogramu jasności obrazu [14]. Tak więc histogram przedstawia rozkład (w formie graficznej lub tablicy) częstości występowania w obrazie cyfrowym poszczególnych poziomów jasności. Niech J_0, J_1,.., J_{L-1} oznaczają możliwe wartości jasności pikseli obrazu, gdzie L oznacza liczbę dostępnych poziomów intensywności. Wówczas rzędne histogramu $h(J_0)$, $h(J_1)$, ..., $h(J_{L-1})$ przedstawiają udział w obrazie pikseli o kolejnych stopniach jasności, tzn.

$h(J_i) = n_i, \quad i = 0,1,2,... L\text{-}1$

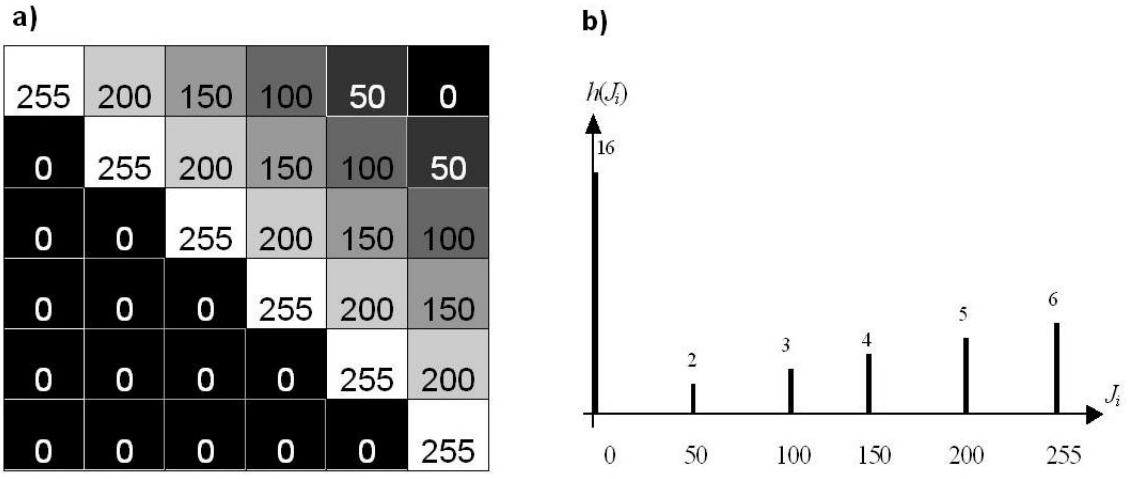
Obliczanie składowych histogramu odbywa się według wzoru:

$$n_i = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} g_i(x,y), \qquad i = 0,1,...,L\text{-}1$$

gdzie: M – rozmiar obrazu w kierunku x,

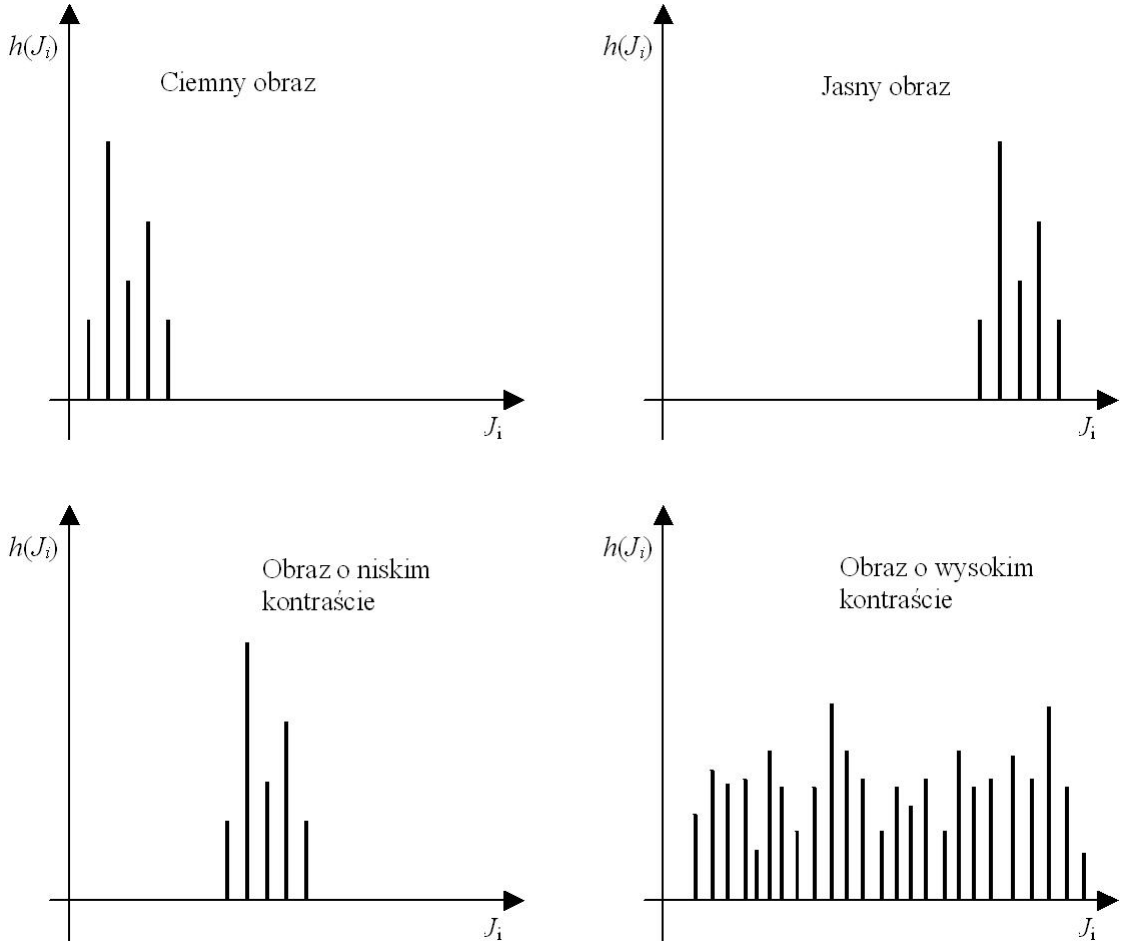
N – rozmiar obrazu w kierunku y,

$$g_i(x,y) = \begin{cases} 1 & \text{gdy } J(x,y) = i \\ 0 & \text{w przeciwnym razie.} \end{cases}$$



Rysunek 17. Przykładowy obraz (a), histogram ilościowy obrazu (b).

Na rysunku (Rysunek 17) pokazano przykładowy obraz (liczby w kratkach oznaczają poziomy jasności pikseli) i jego histogram. Analizując histogram można uzyskać wiele użytecznych informacji na temat rozważanego obrazu. Na przykład można zauważyć, że *h(Ji)* może być zerowa dla pewnych wartości *i*. Może to wystąpić w obrazach, w których niektóre poziomy szarości usunięto, albo kiedy dynamika obrazu jest mała i zakres dostępnych szarości nie jest poprawnie wykorzystany. Objawia się to tym, że składowe histogramu mogą być skupione na początku, pośrodku lub na końcu osi szarości (Rysunek 18). Na ogół taka sytuacja jest nieprawidłowa i można temu zaradzić dokonując odpowiedniej transformacji zakresu jasności lub modyfikacji histogramu. Metody modyfikowania histogramu należą raczej do prostych procedur korekcji obrazu. Stosowane są w przypadkach wymagających zwiększenia kontrastu, przyciemnienia obrazów prześwietlonych lub rozjaśnienia obrazów niedoświetlonych.



Rysunek 18. Przykłady histogramów obrazów o różnej jakości.

Wyrównywanie histogramu

Wyrównywanie histogramu (ang. histogram equalization) ma na celu takie dobranie wartości, aby wykres był możliwie "płaski". W praktyce wyrównywanie histogramu sprowadza się do wykonania przekształcenia obrazu przy pomocy odpowiednio przygotowanej tablicy LUT. Operacja wyrównywania histogramu pozwala na uwypuklenie tych szczegółów w obrazie, które z uwagi na niewielki kontrast są mało widoczne. Na rysunku (Rysunek 19) przedstawiono wynik operacji wyrównywania histogramu dla obrazu kolorowego.

Rysunek 19. Obraz oryginalny (a), obraz wynikowy po operacji wyrównania histogramu (b).

Rozciąganie histogramu

Rozciąganie histogramu (ang. histogram stretching) wykonuje się wówczas, gdy nie pokrywa on całego zakresu wartości składowych obrazu. Operacja ta prowadzi do takiej konwersji zakresu wartości składowych, aby histogram obejmował wszystkie wartości składowych. Czyli jeżeli zakres składowej jest równy 0-255, a najmniejsza wartość w obrazie wynosi 4, największa natomiast wynosi na przykład 198, to po operacji rozciągnięcia wartości tej składowej będą w pełnym zakresie 0-255. Czyli teraz najmniejsza wartość w obrazie wynosi 0 a największa 255. Operacje rozciągania histogramu można przeprowadzić odpowiednio dobierając jasność i kontrast obrazu. W praktyce rozciąganie histogramu sprowadza się do wykonania przekształcenia obrazu przy pomocy odpowiednio przygotowanej tablicy LUT.

2.5. Progowanie i segmentacja

Jednym z podstawowych zadań analizy obrazu jest segmentacja [14] [21] [40], czyli podział obrazu na obszary spełniające pewne kryterium jednorodności. Mówiąc inaczej, pojęcie obszaru jest używane do określenia spójnej grupy punktów obrazu mających dodatkowo pewną wspólną cechę, która nie występuje poza najbliższym sąsiedztwem. Takimi cechami mogą być np. kolor, poziom jasności, faktura. Obszary uzyskane w wyniku stosowania segmentacji mają zazwyczaj swoje odpowiedniki w rzeczywistości. Najczęściej segmentacja polega na oddzieleniu poszczególnych obiektów składających się na obraz.

Segmentację możemy podzielić na:

- segmentację przez podział obszaru,
- segmentację przez rozrost obszaru.

Segmentacja przez podział obszaru zwana też iteracyjną (rekursywną) polega na stopniowym podziale dużych obszarów na mniejsze, w których piksele mają odpowiednią własność (kolor, jasność), znacznie różniącą się od własności pikseli w innych obszarach.

W **segmentacji przez rozrost obszaru** piksele są testowane na to, czy mają określony stopień podobieństwa i jeżeli spełniają warunki, są dołączane do obszaru.

Segmentacja dostarcza zawsze krawędzi zamkniętych (granice obszarów). Z tego też powodu rezultaty segmentacji chętnie są wykorzystywane przez procesy wizji wysokiego poziomu, dokonujące np. rozpoznawania obiektów. Ciągłość krawędzi jest niewątpliwie atutem segmentacji w porównaniu z detektorami krawędzi, które nie zapewniają ciągłości wykrytych krawędzi.

Jedną z najprostszych metod segmentacji przez podział obszaru jest progowanie. Polega ono na porównywaniu wartości każdego punktu obrazu z zadaną wartością progową. Odpowiedni wybór wartości progowej umożliwia wyodrębnienie obszarów określonego typu. Powstało kilka wersji metody wyodrębniania obszarów przez progowanie. Metoda najprostsza stosuje jedną wartość progu i jest realizowana według zależności:

$$J_w(x,y) = \begin{cases} 1; & J(x,y) > t \\ 0; & J(x,y) \le t \end{cases}$$

gdzie t - próg binaryzacji.

Operacja ta oznacza transformację obrazu źródłowego w odcieniach szarości w obraz wyjściowy binarny. Punkty, dla których $J(x,y)>t$ są punktami obiektu, zaś pozostałe punkty obrazu nazywane są tłem.

N rysunku (Rysunek 20) pokazano przykładowy oraz po segmentacji oraz obraz po progowaniu.

Rysunek 20. Obraz oryginalny (a), obraz po segmentacji (b), obraz po progowaniu (c).

2.6. Momenty

Metody momentowe [24] [14] [40] służą do opisu własności figur. W procesie charakteryzowania figur przydatne są zwłaszcza momenty pierwszego i drugiego rzędu. Momenty pierwszego rzędu określają położenie środka ciężkości a momenty drugiego rzędu (p=0,q=2 lub p=2,q=0) są miarą bezwładności danego obiektu.

Moment rzędu p funkcji ciągłej jednej zmiennej jest zdefiniowany następująco:

$$m_p = \int_X x^p \ f(x)dx$$

Moment rzędu p, q funkcji ciągłej dwu zmiennych jest zdefiniowany następująco:

$$m_{p,q} = \int_X \int_Y x^p y^q \ f(x,y)dxdy$$

Definiuje się również momenty centralne, niezależne od współrzędnych obiektu:

$$\mu_{p,q} = \int_X \int_Y (x-\bar{x})^p \ (y-\bar{y})^q \ f(x,y)\ dxdy$$

gdzie $(\bar{x},\bar{y})$ to współrzędne środka obiektu, a funkcja f przyjmuje wartości z przedziału [0,1]. Współrzędne środka można obliczyć ze wzoru:

$$\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}$$

W dwuwymiarowej przestrzeni dyskretnej wzory na momenty p,q i momenty centralne są następujące:

$$m_{p,q} = \sum_X \sum_Y x^p y^q \ f(x,y)$$

$$\mu_{p,q} = \sum_X \sum_Y (x-\bar{x})^p \ (y-\bar{y})^q \ f(x,y)$$

Momenty takie możemy znormalizować, dzieląc je przez odpowiednią potęgę μ_{00}. Otrzymuje się niezmienne względem przesunięcia i skalowalne momenty centralne znormalizowane wyrażające się wzorem:

$$\eta_{p,q} = \frac{\mu_{p,q}}{\mu_{00}^{\gamma}}, \quad \gamma = \frac{p+q}{2} + 1$$

Za pomocą znormalizowanych momentów można wyznaczyć niezmienniki momentowe Hu:

$h_1 = \eta_{20} + \eta_{02}$

$h_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$

$h_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$

$h_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$

$h_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$

$h_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$

$h_7 = (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$

itd. ...

Niezmienniki momentowe są cechami o fundamentalnym znaczeniu przy rozpoznawaniu obiektów płaskich, bo umożliwiają rozpoznanie obiektów przy ich różnej lokalizacji w polu widzenia (w tym ruchome obiekty), przy różnych powiększeniach układu (także ruchy kamer w górę i w dół) oraz przy różnych kątach obrotów obiektów.

3. Opis i analiza wybranych bibliotek

3.1. OPENCV

3.1.1. Przedstawienie

OPENCV [36], pełna nazwa to Open source Computer Vision library została rozwinięta przez Intel Corporation w centrum badawczym Intela w Niżnym Nowogrodzie w Rosji i udostępniona w roku 2000. Jest to darmowa, bardzo duża biblioteka zarówno do użytku edukacyjnego jak również komercyjnego, zawierająca około pięciuset wysoko rozwiniętych algorytmów do przetwarzania obrazów oraz widzenia komputerowego, które są zaprojektowane do tworzenia począwszy od podstawowych aplikacji („zabawek") do potężnych, bardzo zaawansowanych aplikacji wizji cyfrowej. Umożliwia prace zarówno grupom nie posiadającym specjalistycznej wiedzy, poprzez możliwość stworzenia funkcji takich jak np. wykrywanie twarzy w kilku liniach kodu, jak również grupom posiadającym tę wiedzę, umożliwiając przejście do pracy na wyższym poziomie bez zbędnego poświęcania czasu na budowę standardowych funkcji. Zaimplementowana została w C/C++, o „otwartym" kodzie źródłowym [34], co umożliwia dodanie własnego kodu do biblioteki oraz prześledzenie sposobu realizacji poszczególnych zagadnień. Biblioteka OPENCV cechuje się wysoką jakością, funkcjonalnością oraz wydajnością kodu, dzięki czemu możliwe jest jej zastosowanie w szerokiej gamie aplikacji, także w czasie rzeczywistym na komputerze klasy PC.

3.1.2. Wersje oraz dokumentacja

Od momentu udostępnienia pierwszej wersji biblioteki „alpha3" w roku 2000 do tej pory ukazało się siedem wersji biblioteki. Każda z nich niesie ze sobą nowe rozwiązania oraz udoskonalania wersji poprzednich. I tak roku 2001 ukazały się wersje „beta 1", „beta 2", „beta 2_1", w roku 2002 „beta 3", w roku 2003 „beta 3_1", w roku 2004 „beta 4", aż do dnia 19 października 2006 roku, kiedy to ukazała się ostatnia wersja pod nazwą „1.0". Jak widać twórcy tego oprogramowania wykazują się dosyć dobrą aktywnością. Całość oprogramowania wraz z dokumentacją można ściągnąć z

oficjalnej strony internetowej [34], gdzie znajdują się aktualna oraz poprzednie wersje OPENCV oraz dokumentacja. Dokumentacja ta nie jest niestety aktualna, podręcznik użytkownika (manual) dostępny wraz z biblioteką pochodzi z 2001 roku, natomiast tutorial, którego część dotyczy IPP jest z 2003 roku.

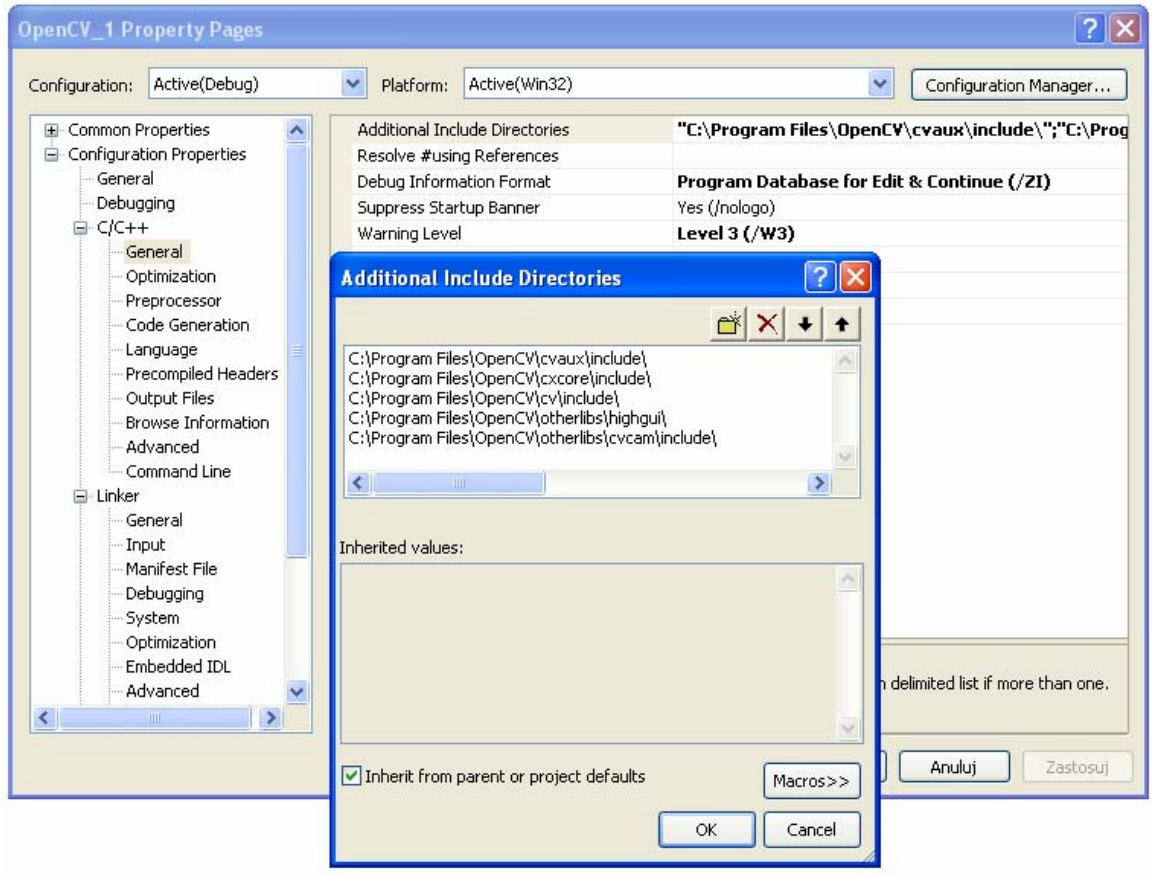
3.1.3. Systemy operacyjne, kompilatory

OPENCV przeznaczona jest do pracy na systemach operacyjnych Windows i Linux pod kompilatorami Visual C++ (działa zarówno na wersji najnowszej Visual Studio .NET 2005, jak również na starszych wersjach Visual Studio .NET 2003 oraz Visual Studio 6.0), Borland oraz GCC.

3.1.4. Instalacja

Aby zainstalować bibliotekę w systemie Windows należy ściągnąć ze strony internetowej [34] najnowszą wersję biblioteki, rozpakować ją, a następnie otworzyć plik instalacyjny biblioteki. Podczas instalacji wybrać docelową lokalizację, gdzie ma zostać zainstalowana biblioteka (sugerowane jest, aby była nią C:\Program Files\OpenCV), zaznaczyć również opcje dodania lokalizacji biblioteki (C:\Program Files\OpenCV\bin) do zmiennej systemowej PATH, wskazuje to systemowi operacyjnemu Windows, gdzie znajdzie biblioteki dynamiczne dll OPENCV. Teraz można przystąpić do tworzenia projektu. Po utworzeniu własnego – nowego - pustego projektu należy zmienić jego ustawienia. W tym celu należy kliknąć prawym przyciskiem myszy na nazwę projektu i wybrać pole *Properties*, następnie wybrać *Configuration Properties* oraz wykonać następujące czynności:
W polu *Additional Include Directories* dodać następujące ścieżki dostępu (Rysunek 21):

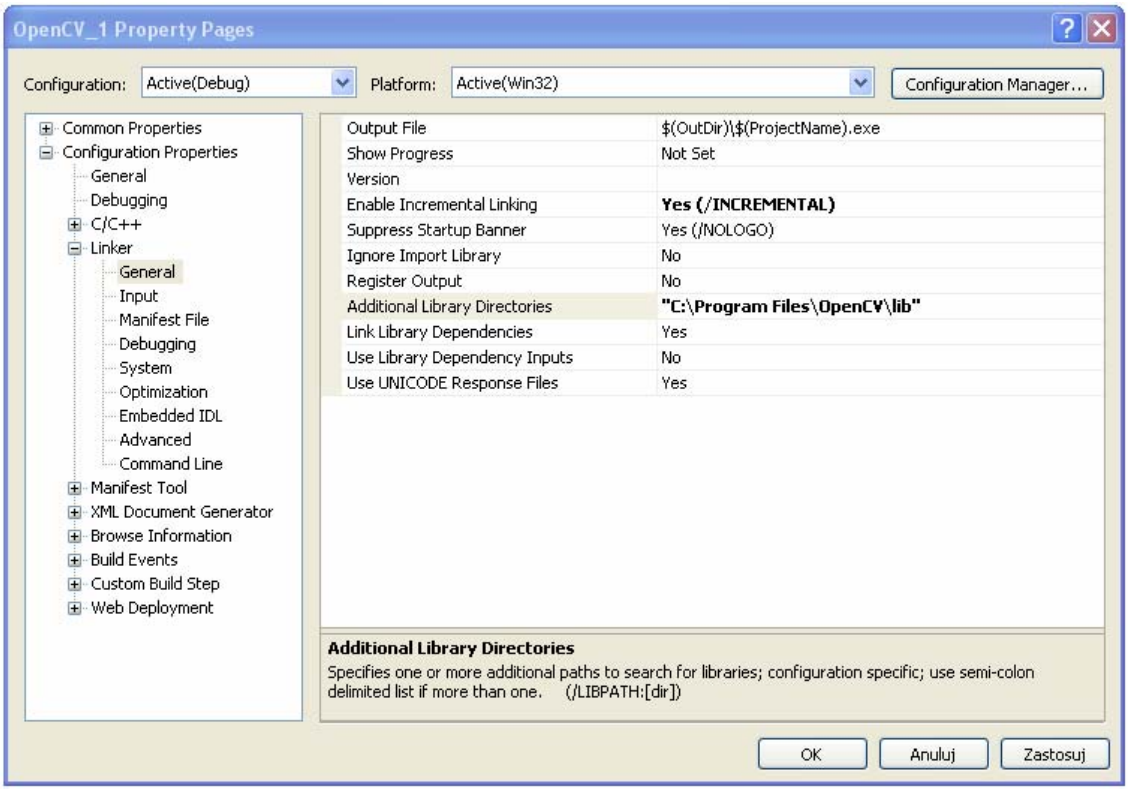
- C:\Program Files\OpenCV\cvaux\include\
- C:\Program Files\OpenCV\cxcore\include\
- C:\Program Files\OpenCV\cv\include\
- C:\Program Files\OpenCV\otherlibs\highgui\
- C:\Program Files\OpenCV\otherlibs\cvcam\include\



Rysunek 21. Widok obrazujący dodawanie ścieżek dostępu do pola *Additional Include Directories*.

W polu *Additional Library Directories* dodać ścieżkę dostępu (Rysunek 22):

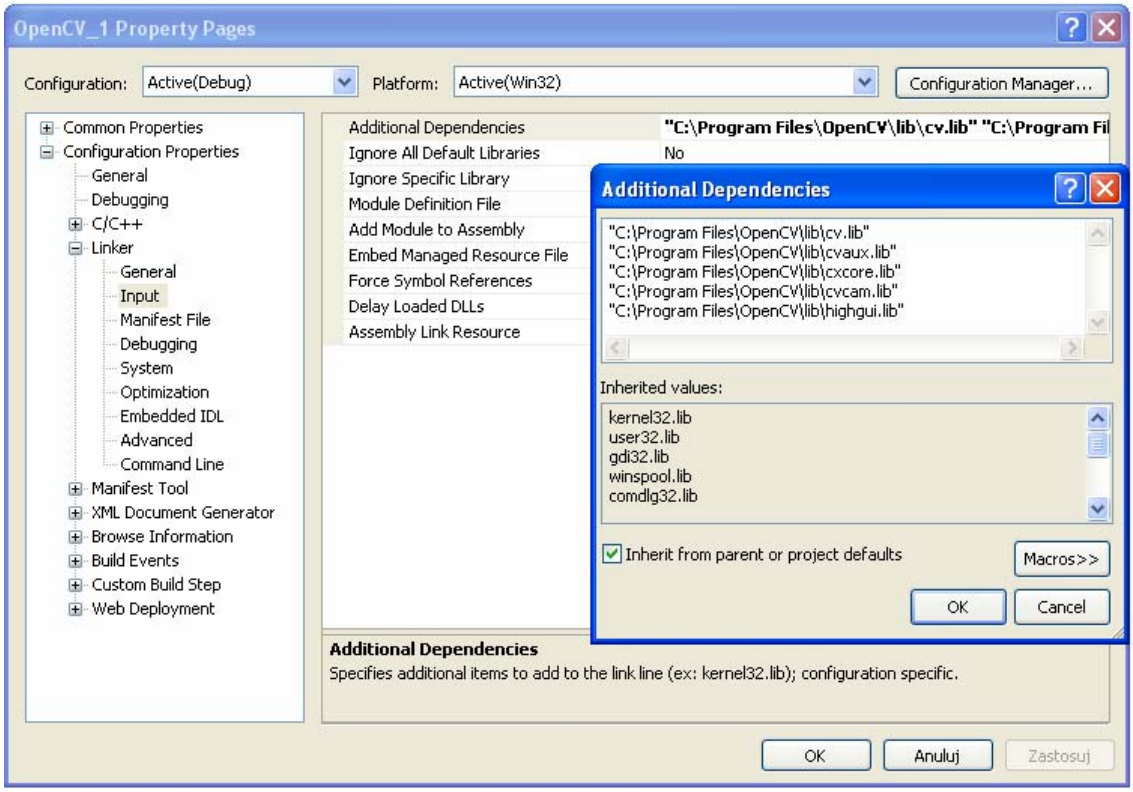
- C:\Program Files\OpenCV\lib



Rysunek 22. Widok obrazujący dodannie ścieżki dostępu do pola *Additional Library Directories*.

W polu *Additional Dependencies* dodać następujące ścieżki dostępu (należy umieścić je w cudzysłowach, Rysunek 23):

- "C:\Program Files\OpenCV\lib\cv.lib"
- "C:\Program Files\OpenCV\lib\cvaux.lib"
- "C:\Program Files\OpenCV\lib\cxcore.lib"
- "C:\Program Files\OpenCV\lib\cvcam.lib"
- "C:\Program Files\OpenCV\lib\highgui.lib"



Rysunek 23. Widok obrazujący dodawanie ścieżek dostępu do pola *Additional Dependencies*.

Na koniec należy dodać do pliku źródłowych własnego projektu następujący kod:

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

Po wykonaniu tych czynności uzyskuje się dostęp do funkcji zawartych w OPENCV.

3.1.5. Struktury danych

Typy danych w OPENCV dzielą się na podstawowe typy danych, na których operuje OPENCV oraz pomocnicze typy danych, dzięki którym praca w OPENCV staje się prostsza i bardziej jednolita.. Do podstawowych typów danych zaliczamy: IplImage, CvMat, CvMatND, CvSeq, CvGraph, CvHistogram. Natomiast do pomocniczych typów danych zaliczają się: CvPoint, CvSize, CvTermCriteria, IplConvKernel, CvMoments itp.

3.1.5.1. Podstawowe struktury danych

Obrazy w OPENCV przechowywane są w formacie IplImage, format ten pochodzi z Intel® Image Processing Library (IPL).

Format IplImage

- `int  nSize;`
- `int  ID;`
- `int  nChannels;`
- `int  alphaChannel;  // ignorowane przez OPENCV`
- `int  depth;`
- `char colorModel[4];  // ignorowane przez OPENCV`
- `char channelSeq[4];  // ignorowane przez OPENCV`
- `int  width;`
- `int  height;`
- `int  origin;`
- `int  dataOrder;`
- `int  align;  // ignorowane przez OPENCV`
- `struct _IplROI *roi;`
- `struct _IplImage *maskROI;  // musi być NULL w OPENCV`
- `void  *imageId;  // musi być NULL`
- `struct _IplTileInfo *tileInfo;  // musi być NULL`
- `int  imageSize;`
- `char *imageData;`
- `int  widthStep;`
- `int  BorderMode[4];  // ignorowane przez OPENCV`
- `int  BorderConst[4];  // ignorowane przez OPENCV`
- `char *imageDataOrigin;`

Parametr **nSize** wskazuje rozmiar IplImage, **ID** wskazuje wersję. Parametr **nChannels** oznacza liczbę kanałów obrazu. Obrazy monochromatyczne zawierają jeden kanał, natomiast obrazy kolorowe przeważnie trzy lub cztery kanały. Pole **width** i **height**

zawiera odpowiednio szerokość i wysokość obrazu w pikselach, głębia pikseli (liczba bitów na piksel) zawarta jest w parametrze **depth** i może ona przyjmować następujące wartości:

- IPL_DEPTH_8U, 8-bitowa liczba całkowita bez znaku
- IPL_DEPTH_8S, 8-bitowa liczba całkowita ze znakiem
- IPL_DEPTH_16U, 8-bitowa liczba całkowita bez znaku
- IPL_DEPTH_16S, 16-bitowa liczba całkowita ze znakiem
- IPL_DEPTH_32S, 32-bitowa liczba całkowita ze znakiem
- IPL_DEPTH_32F, 32-bitowa liczba zmiennoprzecinkowa o pojedynczej precyzji
- IPL_DEPTH_64F, 64-bitowa liczba zmiennoprzecinkowa o podwójnej precyzji

Parametr **origin** wskazuje, czy jako pierwszy w pamięci zostanie umieszczony wiersz górny (origin = IPL_ORIGIN_TL) pikseli obrazu, czy wiersz dolny (origin = IPL_ORIGIN_BL) pikseli obrazu. Parametr **dataOrder** wskazuje jak są umieszczane poszczególne kanały w obrazach kolorowych, czy są umieszczone na przemian czy oddzielnie. Parametr **widthStep** zawiera liczbę bitów pomiędzy punktami w tej samej kolumnie a kolejnymi rzędami. Parametr **imageSize** zawiera rozmiar danych obrazu w bajtach, **imageData** zawiera wskaźnik do pierwszego rzędy danych obrazu. Należy przy tym zapamiętać, że w przestrzeń RGB w OPENCV kolory są umieszczone w kolejności odwrotnej, czyli w kolejności BGR. Struktura IplImage zawiera również pole **roi** (ang. Region of Interest - obszar zainteresowania), będące wskaźnikiem do struktury **_IplROI**, zawierającej prostokątny obszar obrazu, który będzie podlegał przekształceniom oraz kanał zainteresowania (ang. channel of interest – COI), który również będzie podlegał przekształceniom.

Macierze reprezentowane są w strukturach CvMat, CvMatND oraz CvSparseMat, gdzie CvMat jest strukturą dla macierzy jednowymiarowych, CvMatND dla wielowymiarowych zagęszczonych macierzy, natomiast CvSparseMat dla wielowymiarowych rzadkich macierzy. Na typ elementów macierzy we wszystkich trzech strukturach składa się głębia elementów oraz liczba kanałów macierzy (od jednego do czterech). Struktura CvMat jest uogólnieniem macierzy w normalnym tego słowa znaczeniu, oznacza to iż może również przechowywać dane z innych formatów,

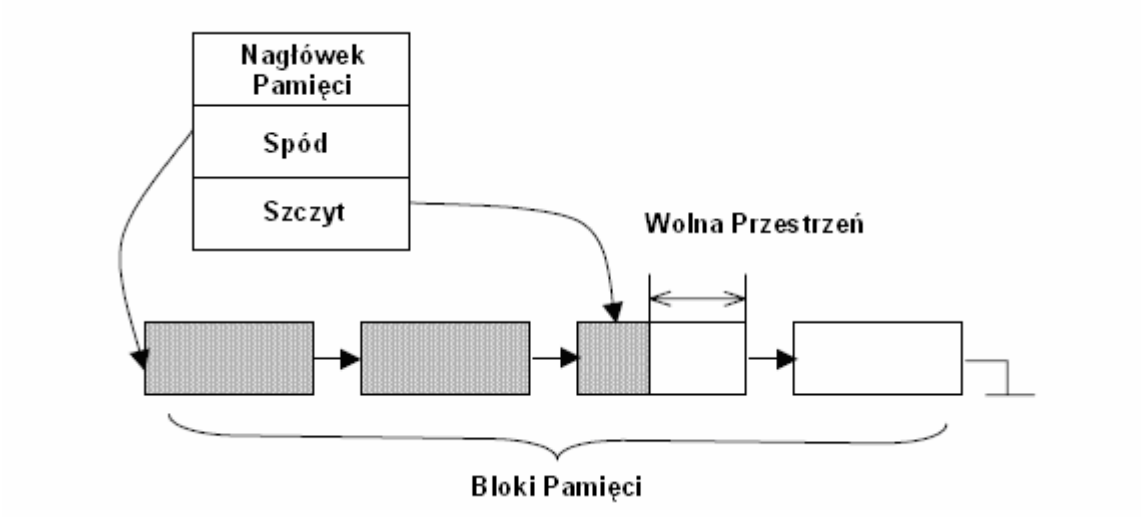
w tym IplImage, dlatego też prawie wszystkie funkcje OPENCV, które akceptują format CvMat zaakceptują również format IplImage i vice versa. Wszystkie podstawowe operacje dla macierzy i obrazów są wykonywalne na strukturze CvMat. Aby zaznaczyć, że dana funkcja działa na kilku różnych typach (np. IplImage, CvMat, CvSeq) wprowadzono pomocniczy typ CvArr.

3.1.5.2. Dynamiczne struktury danych

Dynamiczne struktury danych to proste i złożone struktury danych, którym pamięć jest przydzielana i zwalniana w trakcie wykonywania programu.

CvMemStorage (ang. Memory storage - magazyn pamięci) jest strukturą niskiego poziomu dostarczającą przestrzeń do przechowywania dla dynamicznie rozrastających się struktur danych takich jak sekwencje (ang. sequences), kontury (ang. contours), grafy (ang.graphs) itp. Magazyn (ang. storage) składa się z nagłówka oraz podwójnie połączonej listy bloków pamięci. Lista jest potraktowana jako stos, nagłówek zawiera wskaźnik do bloku, który nie jest całkowicie zajęty oraz liczbę wolnych bajtów w bloku. Spód (ang. bottom) jest początkiem listy bloków, natomiast szczyt (ang. top) jest aktualnie użytym blokiem, lecz niekoniecznie musi być ostatnim blokiem. Bloki te są tych samych rozmiarów (Rysunek 24).

Nowy bufor pamięci może zostać zaalokowany bezpośrednio przez funkcję cvMemStorageAlloc(…) lub też w sposób pośredni przez funkcje wyższego rzędu takie jak np. cvSeqPush(…), cvGraphAddEdge(…). Do utworzenia magazynu pamięci służy cvCreateMemStorage(…), natomiast cvCreateChildMemStorage(…) tworzy potomny magazyn pamięci od wzorca. Dla struktury tej dostarczono też funkcje do zwalniania (cvReleaseMemStorage(…)), czyszczenia (cvClearMemStorage(…)) magazynu pamięci oraz zapisywania (cvSaveMemStoragePos(…)) i przywracania (cvRestoreMemStoragePos(…)) aktualnej pozycji do magazynu pamięci.

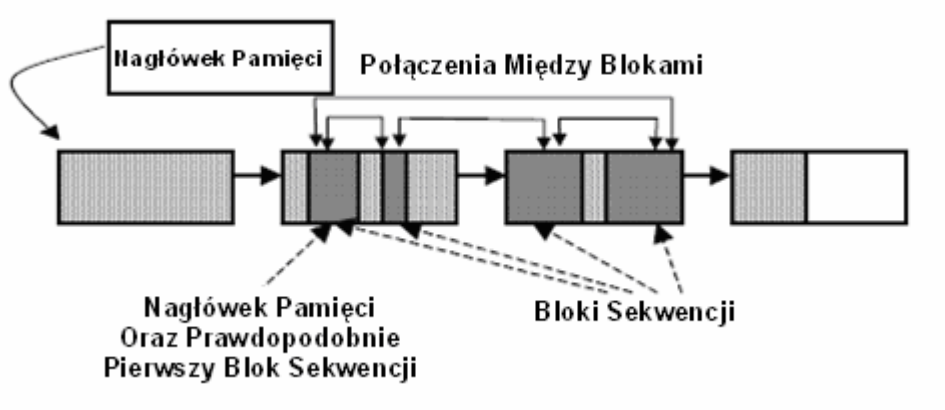


Rysunek 24. Organizacja magazynu pamięci.

Sekwencja jest zmiennej wielkości tablicą dowolnego typu elementów umieszczoną w magazynie pamięci. Sekwencję można podzielić na dwa typy: sekwencje zagęszczoną, gdzie nie ma przerw między elementami sekwencji oraz sekwencje rzadką (luźną), gdzie między elementami sekwencji występują przerwy.

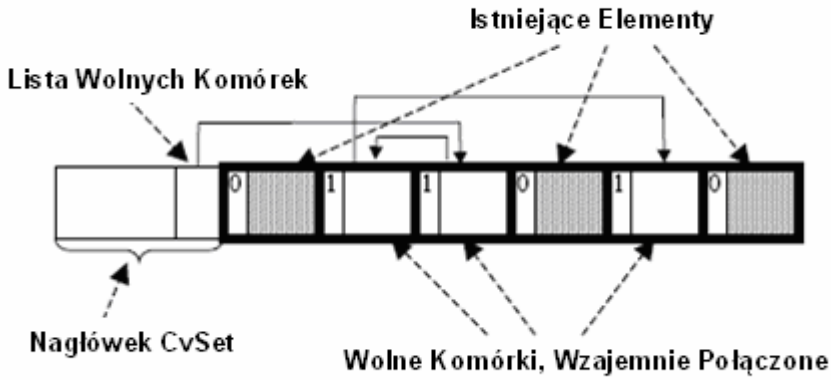
Dla dynamicznych struktur danych podstawową strukturą jest **CvSeq** (Rysunek 25), jest to typowa struktura dla sekwencji zagęszczonych. Struktura ta jest nieciągła. Dane sekwencji mogą zostać podzielone na kilka ciągłych bloków, zwanych blokami sekwencji, które mogą zostać umieszczone w różnych blokach pamięci. Bloki sekwencji są połączone w okrągłe podwójnie związane listy do magazynowania dużych sekwencji lub trzymają kilka małych sekwencji w pojedynczym bloku pamięci. Implementacja sekwencji dostarcza szybkich funkcji do dodawania, usuwania elementów z przodu oraz tyłu sekwencji. Dostępne są również funkcje do wstawiania oraz usuwania elementów ze środka sekwencji, jednakże są one wolniejsze, co jest spowodowane tym, iż jeżeli element jest wstawiany lub usuwany ze środka sekwencji to najbliższe elementy końca są przesuwane, ale dzięki czemu nie ma przerw w sekwencji.

Funkcja cvCreateSeq(…) tworzy nową sekwencję, cvSeqPush(…) dodaje elementy do końca sekwencji, cvSeqPop(…) usuwa elementy z końca sekwencji, cvSeqPushFront(…) dodaje elementy do początku sekwencji, cvSeqPopFront(…) usuwa elementy z początku sekwencji, cvSeqInsert(…) wstawia elementy do środka sekwencji, cvSeqRemove(…) usuwa elementy ze środka sekwencji, natomiast do usuwania wszystkich elementów z sekwencji służy cvClearSeq(…). Dostępnych jest jeszcze kilka funkcji do pracy z sekwencjami.



Rysunek 25. Struktura sekwencji CvSeq.

CvSet (Rysunek 26) jest podstawową strukturą dla rzadkich (luźnych), nieuporządkowanych struktur danych OPENCV. Służy do reprezentacji grafów (CvGraph), rzadkich wielowymiarowych tablic (CvSparseMat) itp. Jest ona implementowana jako podklasa sekwencji CvSeq, używa elementów sekwencji jako komórek oraz zawiera listę wolnych komórek. CvSet wygląda jak lista niezawierająca żadnych związków miedzy elementami struktury. Zarówno istniejące komórki, jak i wolne komórki są elementami sekwencji. Inaczej mówiąc CvSet jest sekwencją zawierającą dodatkowo listę wolnych komórek. Specjalny bit określa, czy dany element struktury CvSet istnieje czy też nie. W poniższym rysunku bity oznaczone przez „1" wskazują na wolne komórki, natomiast bity oznaczone „0" wskazują na zajęte komórki. Poszczególne elementy struktury CvSet nie są połączone, co umożliwia szybsze wykonywanie operacji dodawania oraz usuwania elementów ze środka sekwencji.
Do utworzenia pustej struktury CvSet służy funkcja cvCreateSet(…), cvSetAdd(…) alokuje nową komórkę oraz kopiuje dane do niej, cvGetSetElem(…) znajduje element o danym indeksie, cvSetRemove(…) usuwa element o danym indeksie, natomiast cvClearSet(…) usuwa wszystkie elementy ze struktury CvSet.



Rysunek 26. Struktura CvSet.

CvGraph jest podstawową strukturą dla grafów użytych w OPENCV. Struktura ta dziedziczy od CvSet. Definiowana jest przez dwa zbiory: zbiór wierzchołków i zbiór krawędzi określający powiązania pomiędzy poszczególnymi wierzchołkami. Graf może być zorientowany lub też niezorientowany.
Funkcja cvCreateGraph(…) tworzy pusty graf, cvGraphAddVtx(…) dodaje wierzchołek do grafu, cvGraphRemoveVtx(…) usuwa wierzchołek z grafu, cvGraphAddEdge(…) dodaje krawędź do grafu, cvGraphRemoveEdge(…) usuwa krawędz z grafu.

CvTreeNodeIterator jest strukturą zorganizowaną w drzewo. Drzewo to struktura dynamiczna, w której każdy element (węzeł) ma określoną liczbę dowiązań niższego rzędu (lub równorzędnych w wielokierunkowej odmianie drzewa). Określona jest więc hierarchia węzłów typu rodzic-potomkowie.
Funkcja cvInitTreeNodeIterator(…) inicjuje iterator drzewa, cvInsertNodeIntoTree(…) dodaje węzeł do drzewa, natomiast cvRemoveNodeFromTree(…) usuwa węzeł z drzewa.

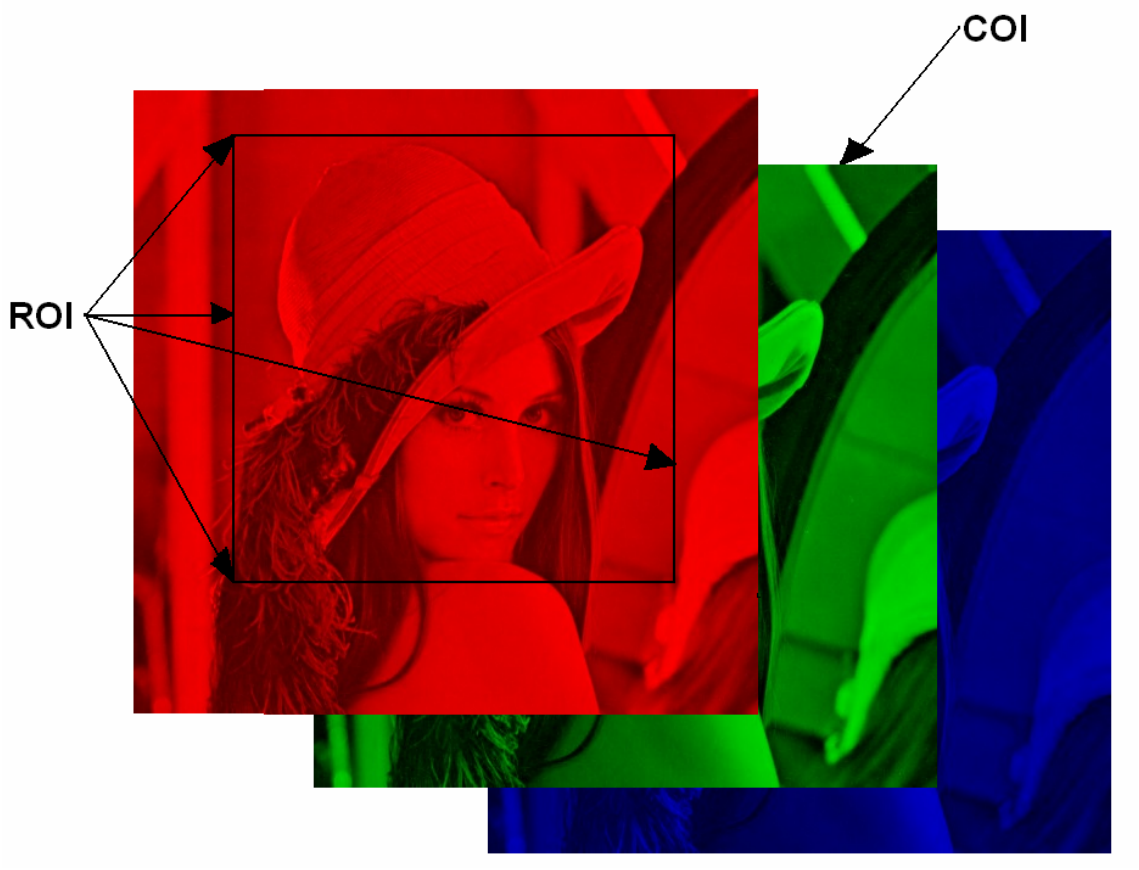
3.1.6. Praca z obrazami

3.1.6.1. Alokacja i zwalnianie obrazu

Podstawową funkcją w bibliotece, z którą można się zetknąć praktycznie przy każdym projekcie jest cvCreateImage(…).Funkcja ta tworzy nagłówek oraz rezerwuje pamięć dla obrazu. Parametrami tej funkcji są wysokość i szerokość w liczbie pikseli, głębia elementów obrazu oraz liczba kanałów. Działanie tej funkcji można rozbić na dwie inne. Mianowicie najpierw utworzyć nagłówek funkcją cvCreateImageHeader(…) a następnie zaalokować pamięć cvCreateData(…). Natomiast funkcją do usuwania nagłówka i zwalniania przydzielonej pamięci jest cvReleaseImage(…), którą odpowiednio można zastąpić funkcjami cvReleaseData(…) i cvReleaseImageHeader(…). Jak widać powyżej w bibliotece tej występują funkcje realizujące szeroki zakres prac jak również funkcje realizujące wąski zakres prac.

3.1.6.2. Wybór obszaru, kanału zainteresowania obrazu oraz kopiowanie obrazu

W OPENCV istnieje możliwość wybrania interesującego obszaru prostokątnego obrazu (ROI) lub interesującego kanału obrazu (COI) lub wybranie obu jednocześnie i przetwarzanie zaznaczonej części (Rysunek 27). Funkcja cvSetImageROI(…) ustawia obszar zainteresowania, cvGetImageROI(…) zwraca obszar zainteresowania obrazu, natomiast cvResetImageROI(…) zwalnia obraz ROI. Do wyboru kanału zainteresowania służy funkcja cvSetImageCOI(…), natomiast cvGetImageCOI(…) zwraca kanał zainteresowania.Za kopiowanie całej zawartości obrazu do innego obrazu odpowiada funkcja cvCloneImage (…).



Rysunek 27. Widok obrazujący wybór ROI i COI.

3.1.6.3. Odczyt i zapis obrazu do pliku

Obrazy mogą być bezpośrednio załadowane z pliku funkcją cvLoadImage(…), funkcja ta zwraca wskaźnik do załadowanego obrazu. Istnieje możliwość wyboru liczby kanałów, w jakich obraz zostanie wczytany. Aktualnie funkcja obsługuje następujące formaty plików:

- Windows bitmaps - BMP, DIB;
- JPEG files - JPEG, JPG, JPE;
- Portable Network Graphics - PNG;
- Portable image format - PBM, PGM, PPM;
- Sun rasters - SR, RAS;
- TIFF files - TIFF, TIF.

Do zapisania obrazu w pliku służy funkcja cvSaveImage(…), obsługuje ona te same formaty plików co cvLoadImage(…), jednakże należy pamiętać iż tylko 8-bitowe obrazy o jednym lub trzech kanałach (w kolejności kanałów BGR) mogą zostać zapisane przy pomocy tej funkcji. Jeżeli natomiast głębia pikseli w bitach, kolejność kanałów jest inna należy poddać obraz przekształceniom funkcjami cvCvtScale(…)

oraz cvCvtColor(…) i następnie zapisać przy pomocy tej funkcji lub użyć uniwersalnej funkcji cvSave(…) zapisującej obraz w formatach XML i YAML.

3.1.6.4. Dostęp do pikseli obrazu

Istnieje kilka sposobów dostępu do pikseli. Aby to zobrazować posłużono się przykładami. Założono, że wymagany jest dostęp do *k-tego* kanału obrazu, *i-tego* wiersza oraz *j-tej* kolumny, gdzie *i* jest z zakresu [0, height-1], *j* jest z zakresu [0, width-1], natomiast *k* jest z zakresu [0, channel-1]:

- pośredni dostęp do pikseli (uogólniony, mało wydajny, do wszystkich typów obrazów)

```
IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);
CvScalar s;
s=cvGet2D(img,i,j);  // pobranie wartości piksela (i,j)
printf("B=%f, G=%f, R=%f\n",s.val[0],s.val[1],s.val[2]);
s.val[0]=111;
s.val[1]=122;
s.val[2]=133;
cvSet2D(img,i,j,s);  // ustawienie wartości piksela (i,j)
```

- bezpośredni (wydajny, ale podatny na błędy)

```
IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,3);
CvScalar s;
s.val[0]= ((uchar *)(temp->imageData + i*temp-
>widthStep))[j*temp->nChannels + 0]; //pobranie wartości
                          piksela, kanał B
s.val[1]= ((uchar *)(temp->imageData + i*temp-
>widthStep))[j*temp->nChannels + 1]; //pobranie wartości
                          piksela, kanał G
s.val[2]= ((uchar *)(temp->imageData + i*temp-
>widthStep))[j*temp->nChannels + 2]; //pobranie wartości
                          piksela, kanał R
s.val[0]=111;
s.val[1]=122;
```

```
s.val[2]=133;
((uchar *)(img->imageData + i*img->widthStep))[j*img->nChannels
+ 0]= s.val[0]; // kanał B, ustawienie wartości piksela
((uchar *)(img->imageData + i*img->widthStep))[j*img->nChannels
+ 1]= s.val[1]; // kanał G, ustawienie wartości piksela
((uchar *)(img->imageData + i*img->widthStep))[j*img->nChannels
+ 2]= s.val[2]; // kanał R, ustawienie wartości piksela
```

- bezpośredni dostęp przy użyciu wskaźnika (bardzo wydajny)

```
IplImage* img = cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,3);
CvScalar s;
int height = img->height;
int width = img->width;
int step = img->widthStep/sizeof(uchar);
int channels = img->nChannels;
uchar* data = (uchar *)img->imageData;
s.val[k] = data[i*step+j*channels+k]; // pobranie wartości
                            piksela, kanał k
s.val[k] = 111;
data[i*step+j*channels+k] = s.val[k]; // ustawienie wartości
                            piksela, kanał k
```

3.1.7. Operacje na macierzach

Bardzo licznie reprezentowane są funkcje do operacji na macierzach. Realizują następujące zagadnienia: dodawanie dwóch macierzy (cvAdd(…)), odejmowanie jednej macierzy od drugiej (cvSub(…)), mnożenie dwóch macierzy (cvMul(…)), dzielenie dwóch macierzy (cvDiv(…)), obliczanie koniunkcji dwóch macierzy (cvAnd(…)), obliczanie koniunkcji macierzy i skalara (cvAddS(…), obliczanie transpozycji macierzy (cvTranspose(…)), obliczanie wyznacznika macierzy (cvDet(…)), obliczanie macierzy odwrotnej lub pseudo-odwrotnej (cvInvert(…), rozwiązywanie równania A*x=b (cvSolve(…)), obliczanie wartości własnych i wektorów własnych macierzy (cvEigenVV(…)), obliczanie dekompozycji SVD (cvSVD(…)).

3.1.8. Konwersja przestrzeni kolorów

W bibliotece tej operacji konwersji z jednej przestrzeni kolorów do innej dokonuje się za pomocą funkcji cvCvtColor(…)[12]. Argumentami tej funkcji są obraz źródłowy, obraz docelowy oraz kod określający między jakimi przestrzeniami kolorów wykonywana jest konwersja. Obrazy źródłowy i docelowy mogą być obrazami o głębi 8-bitowej, 16-bitowej oraz 32-bitowej float. Jednakże przy wykonywaniu danej konwersji oba obrazy muszą być o tej samej głębi pikseli w bitach. Dostępne są następujące przestrzenie kolorów do konwersji: RGB, Gray, HSV, YCrCb, XYZ, Lab, Luv, HLS, Bayer.

3.1.9. Filtracja

Do filtracji w obrazów w OPENCV służą funkcje cvSmooth(…) i cvFilter2D(…).

cvSmooth(…) – funkcja umożliwia realizację filtrów: medianowego, gaussa, rozmywającego prostego, rozmywającego bez skalowania oraz obustronnego (Rysunek 28). Zastosowanie danego filtru definiuje parametr smoothtype. Każdy z filtrów posiada własne cechy oraz ograniczenia. Filtr rozmywający bez skalowania działa tylko na pojedynczych kanałach obrazu o głębi od 8-bitów do 16-bitów oraz 32-bitowej float. Prosty rozmywający i gaussa działają na jednym lub trzech kanałach o głębi 8-bitowej i 32-bitowej float obrazu. Medianowy oraz obustronny działają na jednym lub trzech kanałach 8-bitowych.

cvFilter2D(…) - służy do implementowania dowolnych liniowych filtrów. Działa na zasadzie konwolucji obrazu z jądrem w postaci pojedynczego kanału float macierzy.

cvCopyMakeBorder(…) - kopiuje obraz w postaci tablicy 2D do wnętrza innego obrazu w postaci innej tablicy i zaznacza granice wyszczególnionego typu dookoła skopiowanego obszaru.

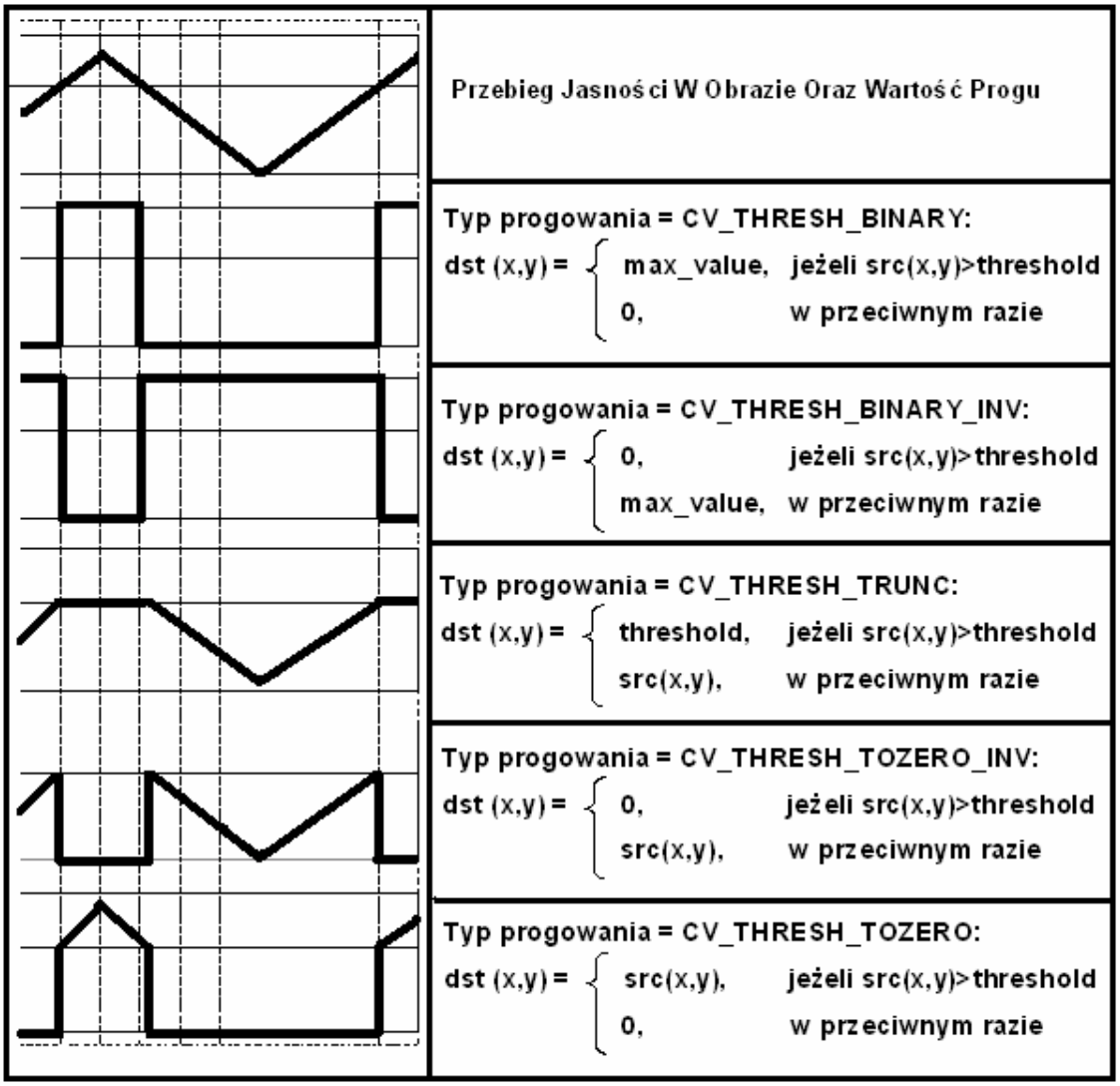
Rysunek 28. Obraz oryginalny z szumem Salt and Pepper (a), obraz po filtracji medianowej 3x3 (b), obraz po filtracji rozmywającej – blur (c), obraz po filtracji filtrem gaussa (d).

3.1.10. Progowanie

W OPENCV dostępne są dwie funkcje do progowania: cvThreshold(…) oraz cvAdaptiveThreshold(…).

cvThreshold(…) dokonuje progowania ze stałą wartością progu dla całego obrazu. Parametrami funkcji są obraz źródłowy, docelowy, wartość progu, maksymalna wartość pikseli przy zastosowaniu typów progowania CV_THRESH_BINARY i CV_THRESH_BINARY_INV oraz typ progowania. Wymagane jest, aby obrazy źródłowy i docelowy były obrazami tego samego typu, o pojedynczym kanale o głębi

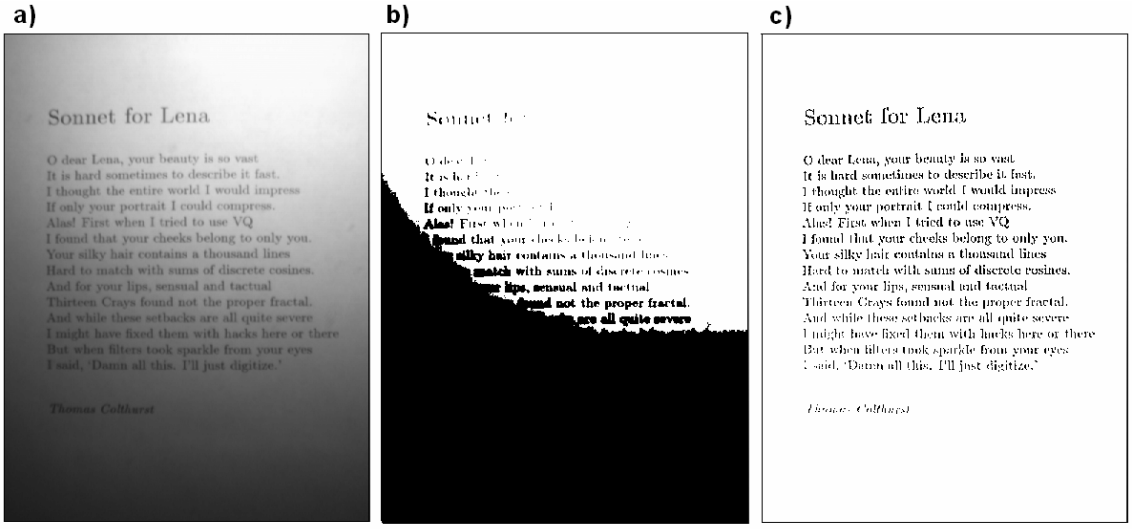
8-bitowej lub 32-bitowej. Rysunek 29 w sposób wizualny przedstawia różne typy progowania [14].



Rysunek 29. Wizualne przedstawienie różnych typów progowania. Oznaczenia: *src* - obraz źródłowy, *dst* - obraz docelowy, *x,y* - współrzędne pikseli, *max_value* - maksymalna wartość pikseli, *threshold* - próg.

cvAdaptiveThreshold(…) - funkcja ta dokonuje progowania adaptacyjnego. W każdym kroku pewne piksele są klasyfikowane na podstawie ich wartości oraz progu obliczonego na podstawie wartości pikseli z pewnego zdefiniowanego otoczenia. Pozostałe, niesklasyfikowane piksele przechodzą do następnego etapu. Funkcja kończy działanie, gdy wszystkie piksele obrazu zostaną sklasyfikowane. W funkcji tej dostępne są dwa algorytmy progowania adaptacyjnego CV_ADAPTIVE_THRESH_MEAN_C oraz CV_ADAPTIVE_THRESH_GAUSSIAN_C, które dokonują progowania typu CV_THRESH_BINARY oraz CV_THRESH_BINARY_INV.

Rysunek 30 ilustruje różnice między progowaniem binarnym a progowaniem adaptacyjnym.



Rysunek 30. Obraz oryginalny (a), obraz po progowaniu binarnym z progiem 120 (b), obraz po progowaniu adaptacyjnym (c).

3.1.11. Morfologia

Morfologia [20] w OPENCV jest reprezentowana dość licznie. OPENCV zawiera następujące funkcje do wykonywania operacji morfologicznych na obrazach:

- cvErode(…) – wykonuje erozję
- cvDilate(…) – wykonuje dylację
- cvMorphologyEx(…) – wykonuje otwarcie, zamkniecie, gradient morfologiczny, operacje "top hat" i "black hat"

Aby móc wykonać operacje na danym obrazie należy utworzyć element strukturalny, natomiast jeżeli nie zostanie on zdefiniowany, domyślnie użyty będzie element o kształcie prostokątnym i rozmiarze 3x3. Do tworzenia elementu strukturalnego o różnych rozmiarach i kształtach (np:. prostokątnym, eliptycznym, krzyżowym, dowolnym) służy funkcja cvCreateStructuringElementEx(…), natomiast do jego zwolnienia cvReleaseStructuringElement(…). Przy każdej operacji morfologicznej podaje się krotność wykonywania funkcji, sprawia to, że funkcja może zostać wykonana n - krotnie bez potrzeby umieszczania jej w dodatkowej pętli. Wszystkie operacje można wykonywać na obrazach kolorowych, wielokanałowych, gdzie każdy kanał przetwarzany jest niezależnie.

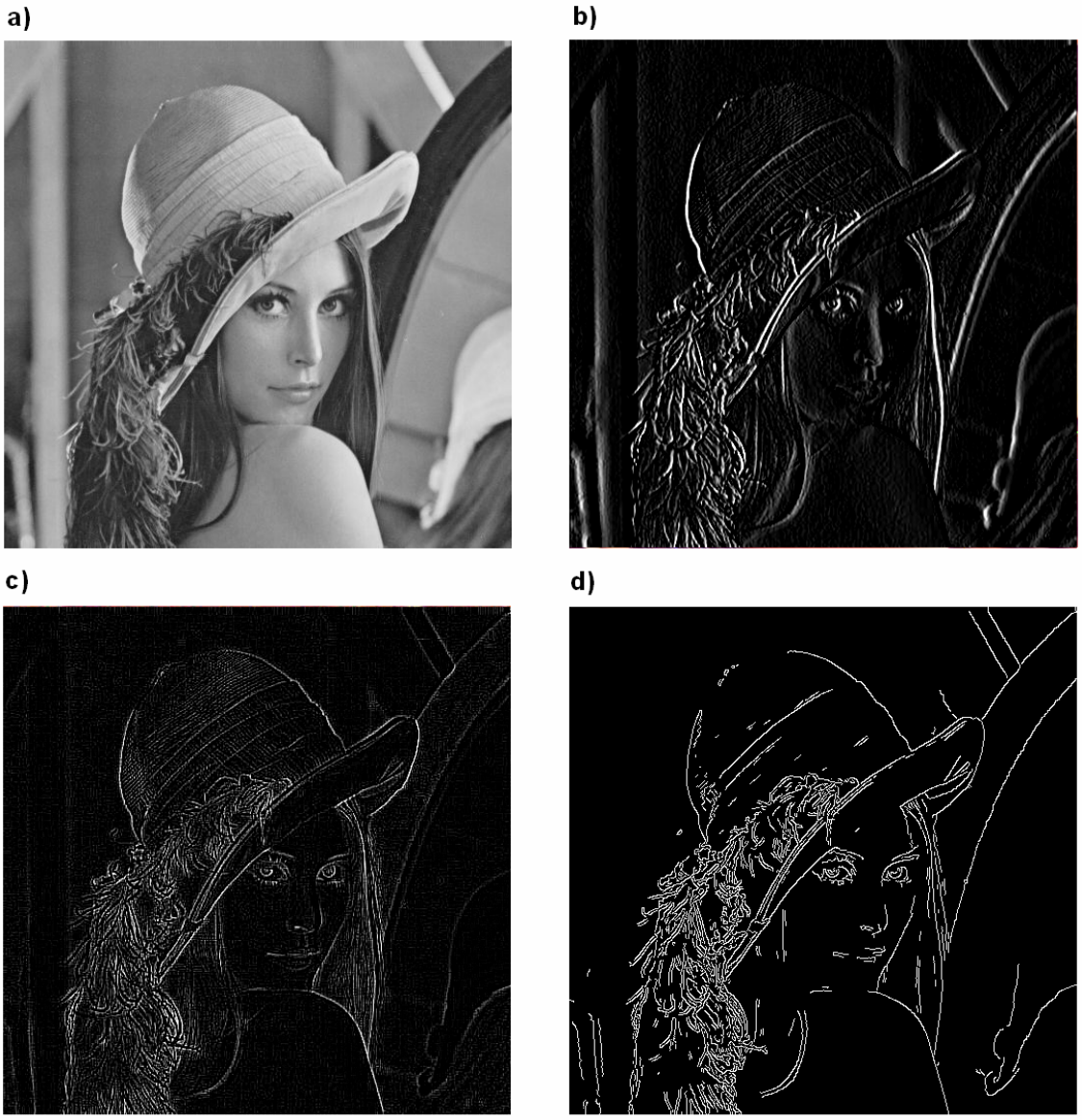
3.1.12. Wykrywanie krawędzi oraz rogów

Wszystkie funkcje do wykrywania krawędzi mogą posiadać maski o następujących rozmiarach: 1x1, 3x3, 5x5, 7x7.

Funkcja cvSobel(…) służy do obliczania pierwszej, drugiej, trzeciej lub mieszanych pochodnych obrazu przez konwolucję obrazu z odpowiednią maską Sobela. W funkcji tej możliwe jest także zastosowanie jako maski filtru Scharra. Wymagane jest, aby obraz docelowy był 16-bitowy, przy 8-bitowym obrazie źródłowym, ma to zapobiegać wylewaniu powodowanemu przez większy rozmiar obrazu docelowego niż źródłowego. Funkcja może również pracować na 32-bitowych obrazach typu float. Operator Sobela jest połączeniem różniczkowania z filtracją Gaussa.

Do obliczania laplasjanu obrazu źródłowego przez sumowanie pochodnych cząstkowych drugiego rzędu po zmiennych x i y służy funkcja cvLaplace(…), posiada ona te same ograniczenia odnośnie obrazu źródłowego i docelowego, co cvSobel(…).

Funkcja cvCanny(…)zawiera zaimplementowany algorytm Canny'ego do wykrywania krawędzi [5]. W algorytmie tym zastosowano progowanie z histerezą o dwóch wartościach progu (górnym i dolnym progiem histerezy) mające na celu poprawienie efektywności wykrywania krawędzi. Wartości tych progów są parametrami tej funkcji. Jako obraz docelowy otrzymywany jest obraz krawędziowy.
Rysunek 31 przedstawia obrazy wynikowe funkcji Sobel'a, Lapalce'a i Canny obrazu monochromatycznego.



Rysunek 31. Obraz oryginalny monochromatyczny (a), obraz po użyciu funkcji Sobel'a (b), obraz po użyciu funkcji Laplace'a (c), obraz po użyciu funkcji Canny'ego (c).

cvPreCornerDetect(…) to funkcja do wykrywa rogów w obrazie źródłowym, które mogą być zdefiniowane jako obszar, gdzie poziome krzywe przemnożone przez wartość gradientu podniesionego do trzeciej potęgi obejmują maksima lokalne następującej funkcji:

$$f(x) = D_x^2 D_{yy} + D_y^2 D_{xx} - 2 D_x D_y D_{xy}$$

gdzie D_x – pierwsza pochodna po x, D_{xy} - pierwsza pochodna cząstkowa po x i y, D_{yy} - druga pochodna po y.

cvCornerEigenValsAndVecs(…) oblicza wartości własne i wektory własne bloków obrazu w celu wykrycia rogów. Dla każdego piksela rozważane jest sąsiedztwo P(s) o rozmiarach blok ^{x} blok. Obliczana jest macierz kowariancji:

$$M = \begin{bmatrix} \sum S(p)\,(dI/dx)^2 & \sum S(p)(dI/dx * dI/dy) \\ \sum S(p)(dI/dx * dI/dy) & \sum S(p)(dI/dx)^2 \end{bmatrix}$$

Następnie znajdowane są wartości i wektory własne, które są gromadzone w obrazie docelowym w formie (λ_1, λ_2, x_1, y_1, x_2, y_2), gdzie:

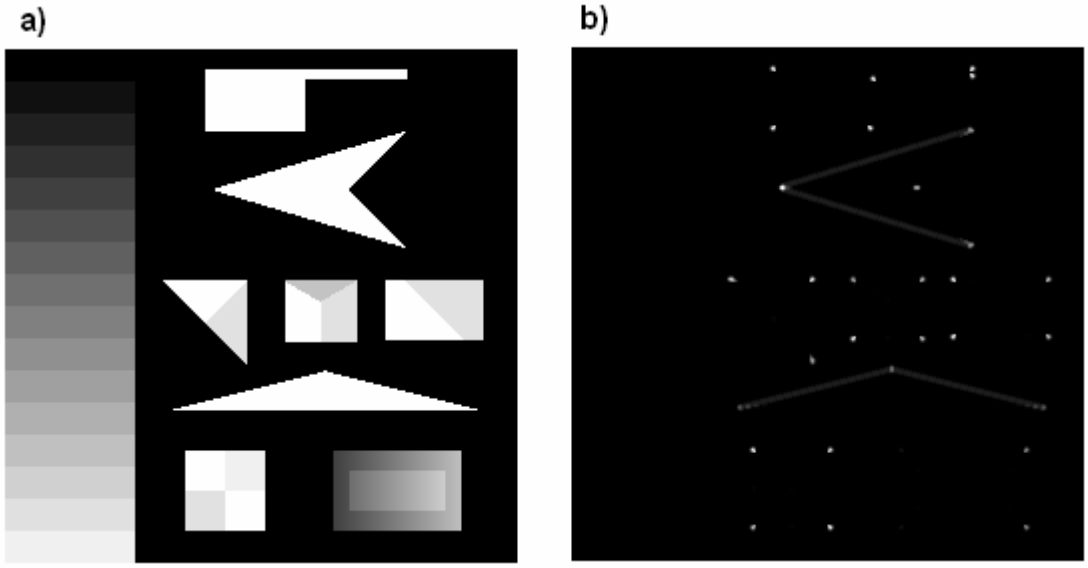
λ_1, λ_2 - wartości własne macierzy *M*

(x_1, y_1) - wektory własne odpowiadające λ_1

(x_2, y_2) - wektory własne odpowiadające λ_2

W funkcji tej do obliczania pochodnych użyty jest operator Sobela. Obraz docelowy, w którym gromadzone są wyniki, powinien być 6 razy większy niż obraz źródłowy.

cvCornerMinEigenVal(…) jest bardzo podobną funkcją do poprzednio opisanej funkcji cvCornerEigenValsAndVecs(…) z tą różnicą, iż oblicza i gromadzi w obrazie docelowym jedynie minimalną wartość własną pochodnej macierzy kowariancji dla każdego piksela tj. $min(\lambda_1, \lambda_2)$ oraz w funkcji tej obraz docelowy i źródłowy są tych samych rozmiarów. Rysunek 32 obrazuje operację wykrywania rogów funkcją cvCornerMinEigenVal(…).



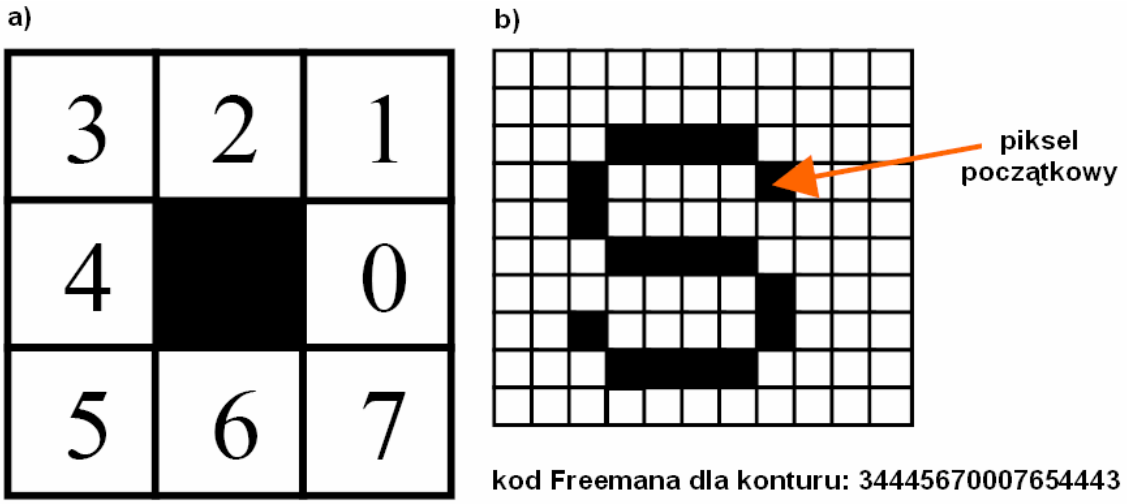
Rysunek 32. Obraz oryginalny (a), obraz po operacji wykrywania rogów funkcją cvCornerMinEigenVal(…) (b).

cvCornerHarris(…) jest to funkcja oparta na algorytmie Harrisa do wykrywania rogów, podobna do poprzednich funkcji cvCornerMinEigenVal(…) oraz cvCornerEigenValsAndVecs(…). Dla każdego piksela oblicza gradient macierzy kowariancji M o rozmiarach 2x2 przemnożony przez sąsiedztwo piksela o rozmiarach blok ^{x} blok. Następnie w obrazie docelowego gromadzone są $C = det(M) - k*trace(M)^2$, gdzie C - róg, k -współczynnik Harrisa definiowany jako parametr i domyślnie ustawiony na 0.04. Rogi można znaleźć jako lokalne maksima obrazu docelowego.

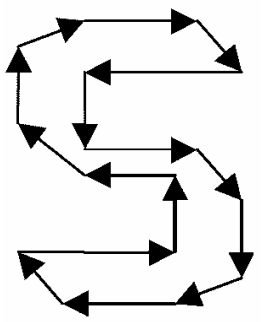
cvGoodFeaturesToTrack(…) znajduje rogi z dużymi wartościami własnymi w obrazie, najpierw jest obliczana najmniejsza wartość własna dla każdego piksela obrazu źródłowego przy pomocy funkcji cvCornerMinEigenVal(…) i umieszczana w tymczasowym obrazie *eig_image*, następnie z obrazu gradientowego usuwane są punkty nie będące lokalnymi maksimami gradientu. Następnie odrzucane są rogi z minimalnymi wartościami własnymi mniejszymi niż *quality_level•max(eig_image(x,y))*, gdzie *quality_level* precyzuje minimalną jakość rogów obrazu. Ostatecznie funkcja zapewnia, że wszystkie ze znalezionych rogów są wystarczająco oddalone jeden od drugiego rozpatrując rogi i sprawdzając czy odległość pomiędzy ostatnio rozważaną cechą a cechą rozważaną wcześniej jest większa od minimalnej odległości pomiędzy zwróconymi rogami (parametr *min_distance*). Funkcja ta usuwa cechy będące zbyt blisko mocniejszych cech.

3.1.13. Kontury

W OPENCV zaimplementowano funkcje do znajdowania konturów w obrazie binarnym. Do reprezentacji konturów biblioteka używa dwóch metod: łańcucha kodów (kod Freemana (Rysunek 33)) oraz reprezentacji za pomocą wielokątów w postaci sekwencji punktów, wierzchołków linii łamanej (Rysunek 34).



Rysunek 33. Oznaczenie sąsiednich pikseli w kodzie Freemana (a), przykład kodowania konturu kodem Freemana (b).



Rysunek 34. Reprezentacja za pomocą wielokątów konturów w postaci linii łamanych i wierzchołków.

Aby znaleźć kontur w obrazie można:

- posłużyć się funkcją cvFindContours(…), która znajduje wszystkie kontury z obrazu binarnego oraz zwraca liczbę znalezionych konturów.

- dokonać inicjalizacji skanera konturów cvStartFindContours(…), a następnie funkcją cvFindNextContour(…) wyznaczyć kolejne kontury. Funkcja cvSubstituteContour(…) zastępuje kontur znaleziony podczas poprzedniego wywołania funkcji cvFindNextContour(…) przez nowy kontur. Natomiast funkcja cvEndFindContours(…) kończy znajdowanie konturów oraz zwraca wskaźnik do pierwszego konturu na najwyższym poziomie.

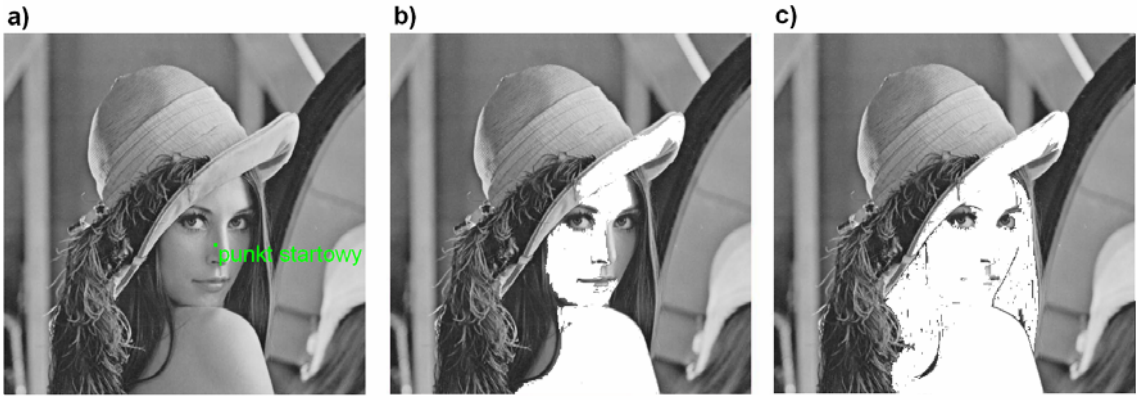
Parametr *mode* dostępny w funkcjach cvFindNextContour(…) oraz cvStartFindContours(…) definiuje rodzaj zastosowanego algorytmu do znajdowania konturów:

- CV_RETR_EXTERNAL - pierwszy algorytm znajduję wyłącznie krańcowe, najbardziej zewnętrzne kontury

- CV_RETR_LIST - drugi algorytm znajduje wszystkie kontury
- CV_RETR_CCOMP - trzeci algorytm zwraca wszystkie znalezione kontury w dwu-poziomowej strukturze. Na szczycie tej struktury znajdują się najbardziej zewnętrzne kontury komponentów złożonych z pikseli o wartościach równych „1”. Każdy nagłówek konturu zawiera odwołanie do listy dziur (połączone komponenty, gdzie wartości pikseli wynoszą „0”) zawartych w danym konturze.

CV_RETR_TREE - znajduje wszystkie kontury oraz zwraca hierarchiczne drzewo, gdzie wszystkie kontury zawierają listę konturów otoczonych przez dany kontur bezpośrednio.

cvFloodFill(…) dokonuje wypełniania zamkniętych, połączonych obszarów (komponentów) wybranym kolorem. Algorytm flood fill rozpoczyna swoje działanie od punktu początkowego o określonym kolorze i dokonuje wypełniania obszaru do momentu aż natrafi na granice obrazu lub nie będzie mógł znaleźć żadnego nowego piksela do wypełnienia z powodu zbyt dużej różnicy między wartościami jasności pikseli początkowego i wypełnianego w porównaniu do maksymalnej dozwolonej różnicy jasności. Rysunek 35 ilustruje działanie funkcji cvFloodFill(...).



Rysunek 35. Obraz oryginalny z zaznaczonym punktem startowym (a), obraz po przekształceniach funkcją cvFloodFill (...) z tolerancją +/-4 (b), obraz po przekształceniach funkcją cvFloodFill (...) z tolerancją +/-7 (c).

3.1.14. Histogram

Aby móc gromadzić wszystkie typy (1D, 2D, nD) histogramu, w OPENCV zawarto specjalną strukturę CvHistogram. Tworzenia histogramu dokonuje się za pomocą funkcji cvCreateHist(…). Parametr *dims* definiuje liczbę wymiarów histogramu, *sizes* określa rozmiar histogramu, Parametr *type* służy do określenia formy reprezentacji histogramu, może on zostać zgromadzony albo w zwartej (ang. dense)

formie jako wielowymiarowa zwarta tablica CvMatND lub rzadkiej (ang. sparse) formie jako wielowymiarowa rzadka tablica CvSparseMat. Zalecane jest, aby dla histogramów jedno i dwuwymiarowych używać zwartej formy, natomiast dla histogramu trzy i więcej wymiarowego formy rzadkiej. Parametr *ranges* definiuje zakres histogramu, jeżeli będzie wynosił 0 to musi on później zostać określony przez funkcję cvSetHistBinRanges(…), chociaż funkcje cvCalcHist(…) i cvCalcBackProject(…) mogą pracować na 8-bitowych obrazach bez ustawionego zakresu, przyjmują wówczas zakres od 0 do 255. Do zwalniania (usuwania) histogramu służy cvReleaseHist(…), natomiast do czyszczenia służy cvClearHist(…).

Do biblioteki dostarczono również bardzo duży zasób funkcji operujących na histogramie:

cvCalcHist(…) - oblicza histogram jednego lub kilku obrazów jednokanałowych

cvGetMinMaxHistValue(…) znajduje minimalną i maksymalną wartość (minimalną i maksymalną liczbę wystąpień piksela) histogramu oraz ich pozycje,

cvNormalizeHist(…) - dokonuje normalizacji histogramu,

cvEqualizeHist(…) – wyrównuje histogram

cvQueryHistValue_*D(…) - zwraca wartość histogramu

cvThreshHist(…) – dokonuje progowania histogramu

cvCompareHist(…) dokonuje porównania dwóch histogramów, aktualnie dostępne są cztery metody porównania

Do biblioteki tej dostarczono także funkcje do obliczania tzw. projekcji wstecznej (ang. back projection). Funkcją cvCalcBackProject(…) oblicza projekcję wsteczną z histogramu, natomiast cvCalcBackProjectPatch(…) dokonuje obliczenia projekcji wstecznej przez porównanie histogramu obrazu źródłowego z dostarczonym histogramem.

3.1.15. Momenty

W OPENCV dostarczono funkcje do obliczania momentów centralnych, przestrzennych, centralnych znormalizowanych oraz niezmienników momentowych. Funkcja cvMoments(…) dokonuje obliczania wszystkich momentów o maksymalnym rzędzie równym 3 i zapisuje je w strukturze CvMoments oraz zwraca wskaźnik do tej struktury. Obraz źródłowy musi być jednokanałowy lub trzykanałowy z wybranym

kanałem zainteresowania (COI). Następnie momenty te mogą zostać użyte przez następujące funkcje:

- cvGetSpatialMoment(…) – do wyznaczenia momentu przestrzennego
- cvGetCentralMoment(…) – do wyznaczenia momentu centralnego
- cvGetNormalizedCentralMoment – do wyznaczenia momentu centralnego znormalizowanego
- cvGetHuMoments(…) – do wyznaczenia siedmiu niezmienników momentowych.

3.1.16. Porównywanie obszarów obrazu

Funkcja cvMatchTemplate(…) dokonuje porównywania obszarów obrazu na wypadek ich pokrywania się. Parametrami tej funkcji są obraz źródłowy 8-bitowy lub 32-bitowy float, szukany szablon, tablica gdzie zgromadzone zostaną wyniki oraz jedna z sześciu metod porównania. Po zakończeniu działania funkcji cvMatchTemplate(…), przy pomocy funkcji cvMinMaxLoc(…) można znaleźć najlepsze dopasowanie.

Natomiast funkcja cvMatchShapes(…) dokonuje porównywania dwóch kształtów na podstawie niezmienników momentowych Hu, aktualnie dostępne są trzy metody porównania:

- method=CV_CONTOUR_MATCH_I1:
  $I_1(A,B)=\sum_{i=1..7} abs(1/m^A_i - 1/m^B_i)$

- method=CV_CONTOUR_MATCH_I2:
  $I_2(A,B)=\sum_{i=1..7} abs(m^A_i - m^B_i)$

- method=CV_CONTOUR_MATCH_I3:
  $I_3(A,B)=\sum_{i=1..7} abs(m^A_i - m^B_i)/abs(m^A_i)$

gdzie:

A - pierwszy kształt, B - drugi kształt,

$m^A_i=sign(h^A_i)\cdot log(h^A_i)$,

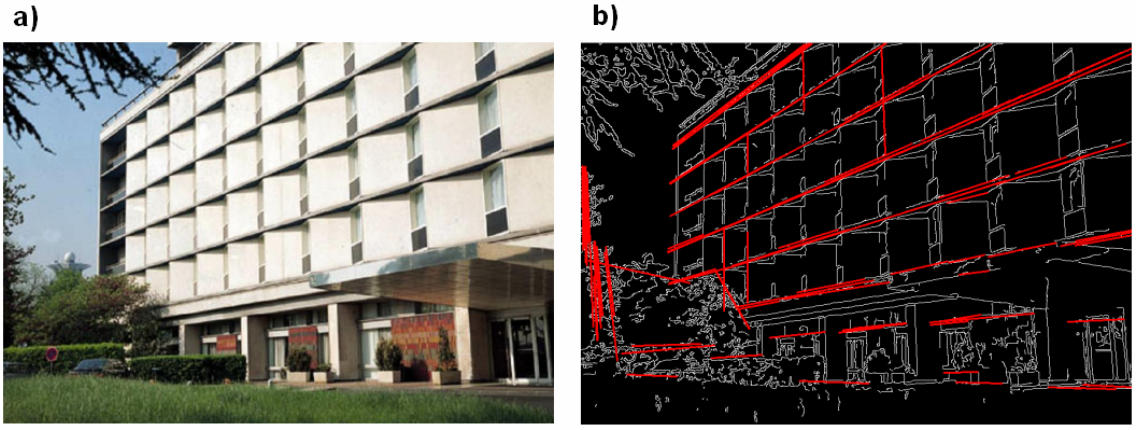
$m^B_i=sign(h^B_i)\cdot log(h^B_i)$,

h^A_i, h^B_i – niezmienniki momentowe Hu odpowiednio kształtu A i B.

3.1.17. Transformata Hough'a

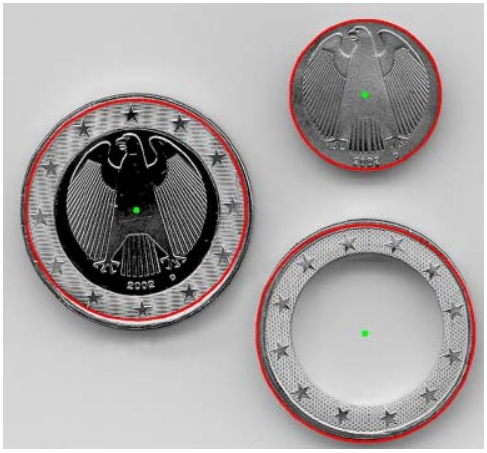
W OPENCV użyto transformaty Hough'a do stworzenia funkcji do wykrywania linii oraz okręgów z obrazu.

Funkcja cvHoughLines2(…) implementuje trzy wersje transformaty Hough'a do wykrywania linii (Rysunek 36). Działa ona na obrazach binarnych 8-bitowych. Parametr method zawarty w tej funkcji definiuje zastosowanie odpowiednie metody:
CV_HOUGH_STANDARD – klasyczna transformata Hough'a, każdy linia reprezentowana jest przez dwie liczby (ρ,T) typu float (dwie liczby zmiennoprzecinkowe o podwójnej precyzji), gdzie ρ jest odległością pomiędzy punktem (0,0) a linią, natomiast T to kąt pomiędzy osią x a normalną (prostopadłą) do linii.
CV_HOUGH_PROBABILISTIC – probabilistyczna transformata Hough'a, bardziej skuteczna w przypadku, gdy obraz zawiera kilka długich liniowych segmentów. Każdy segment reprezentowany jest przez punk startowy i końcowy.
CV_HOUGH_MULTI_SCALE – multi-scale wariant transformaty klasycznej Hough'a, linie są podobnie kodowane jak w CV_HOUGH_STANDARD.



Rysunek 36. Obraz oryginalny (a), obraz po detekcji linii funkcją cvHoughLines2(...) (b).

Funkcja cvHoughCircles(…) znajduje okręgi z obrazu 8-bitowego w odcieniach szarości przy pomocy zmodyfikowanej transformaty Hough'a (Rysunek 37). Do tej pory została zaimplementowana tylko jedna metoda CV_HOUGH_GRADIENT, opisana w [46].



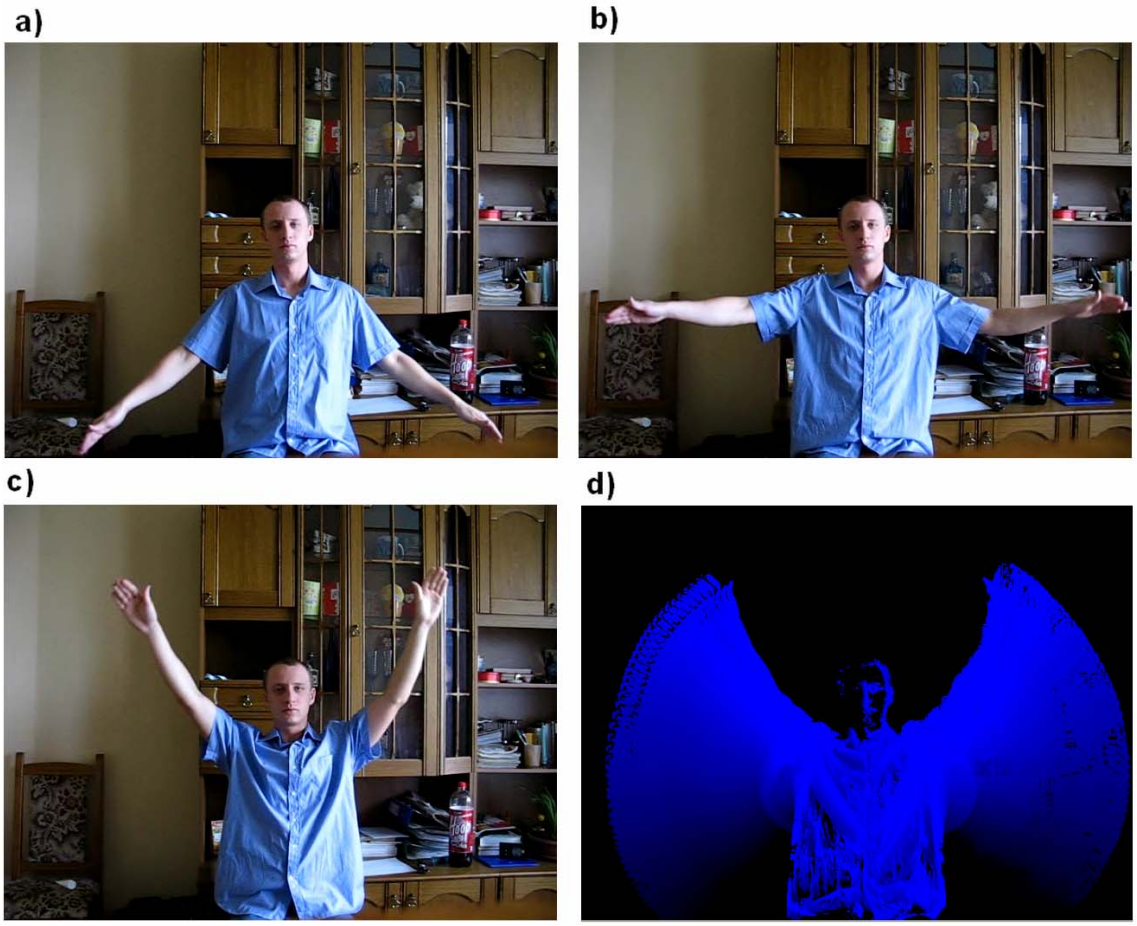
Rysunek 37. Obraz z zaznaczonymi na czerwono wykrytymi okręgami przy pomocy funkcji cvHoughCircles(...).

3.1.18. Analiza ruchu, śledzenie obiektów

W tym rozdziale termin „tło” będzie rozumiany jako zbiór pikseli obrazu bez ruchu, które nie należą do żadnych obiektów będących w ruchu. Najprostszy model tła przyjmuje, że jasność piksela tła zmienia się niezależnie, zgodnie z normalnym rozkładem. Cechy charakterystyczne tła mogą być obliczone poprzez zebranie kilkudziesięciu klatek funkcją cvAcc(...), która działa na zasadzie dodawania obrazu wejściowego do akumulatora, poprzez spotęgowanie klatek funkcją cvSquareAcc(...), która dodaje kwadrat obrazu wejściowego do akumulatora, poprzez wymnożenie klatek funkcją cvMultiplyAcc(...), która dodaje iloczyn klatek wejściowych do akumulatora oraz poprzez obliczenie sumy ważonej obrazu wejściowego oraz akumulatora funkcją cvRunningAvg(…), przez co akumulator staje się bieżącą średnią sekwencji klatek.

3.1.18.1. Szablony ruchu

W punkcie tym zostały opisane funkcje do generowania obrazów szablonów ruchu, które mogą zostać użyte do określenia, gdzie wystąpił ruch, jak to się zdarzyło oraz w jakim kierunku wystąpił. Algorytmy są oparte na publikacjach [7], [3] oraz [8]. Funkcje te operują na obrazach produkowanych przez operacje odejmowania tła (ang. background subtraction). Dla wszystkich omówionych tu funkcji obraz wejściowy oraz wyjściowy musi być jednokanałowy w odcieniach szarości. Jednym z takich szablonów jest MHI (ang. Motion History Image - obraz historii ruchu), zwarty szablon przedstawiającym ruch w postaci obrazu (Rysunek 38).



Rysunek 38. Klatka-0 sekwencji obrazów(a), klatka-20 sekwencji obrazów (b), klatka-45 sekwencji obrazów (c), obraz historii ruchu (MHI) dla poruszającej się sylwetki z sekwencji obrazów (c).

Funkcja cvUpdateMotionHistory(…) służy do aktualizacji obrazu historii ruchu. Argument silhouette określa maskę z wartościami różnymi od 0, gdzie wystąpił ruch. Argument mhi określa obraz historii ruchu, który jest aktualizowany, timestamp określa aktualny czas w milisekundach, natomiast argument duration określa maksymalny czas trwania ruchu, który zostanie wykryty. Funkcja ta aktualizuje mhi w następujący sposób:

$$mhi(x,y) = \begin{cases} \text{jeżeli silhouette}(x,y)=0 \text{ i } mhi(x,y)<\text{timestamp-duration} \\ mhi(x,y) \quad \text{w przeciwnym razie} \end{cases}$$

Funkcja cvCalcMotionGradient(..) oblicza gradient obrazu historii ruchu. Argumentami jej są obraz historii ruchu(mhi), mask – obraz w postaci maski z zaznaczonymi pikselami, gdzie nachylenie obrazu historii ruchu jest poprawiane, orientation – obraz orientacji nachylenia ruchu, argumenty delta1 i delta2 oraz aperture_size.

Funkcja cvCalcGlobalOrientation(…) oblicza ogólny kierunek ruchu wybranego regionu oraz zwraca kąt pomiędzy 0 a 360 stopniami.

Funkcja cvSegmentMotion(…) znajduje wszystkie segmenty ruchu oraz zaznacza je w obrazie seg_mask każdy z indywidualnymi wartościami. Zwraca również sekwencje struktur CvConnectedComp, jedną dla każdego komponentu ruchu.

3.1.18.2. Śledzenie obiektów

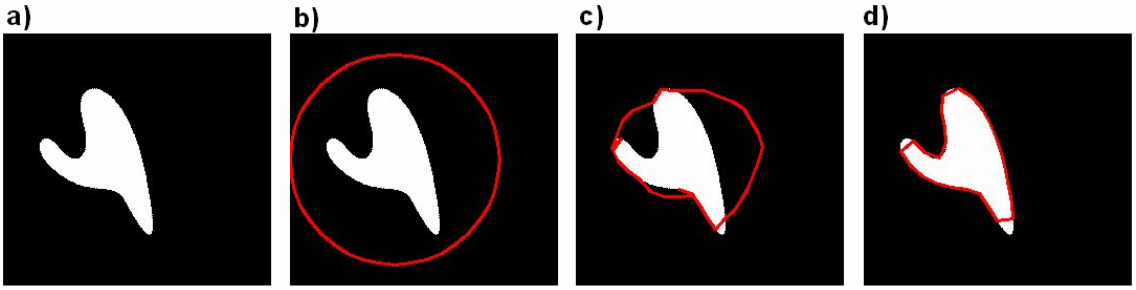
Do śledzenia obiektów w OPENCV zaimplementowano następujące funkcje:
cvMeanShift() – funkcja wykonuje kolejne iteracje po to by znaleźć środek obiektu w obrazie po zastosowaniu projekcji (rzutowania) wstecznej oraz początkową pozycję przeszukiwanego okna. Iteracje wykonywane są do momentu aż środek okna przeszukiwanego przesuwa się poniżej zadanej wartości lub funkcja wykonała maksymalną liczbę iteracji. Funkcja ta zwraca liczbę wykonanych iteracji.

cvCamShift(…) – w funkcji tej został zaimplementowany algorytm CAMSHIFT do śledzenia obiektów [4]. Najpierw funkcja znajduje środek obiektu przy użyciu cvMeanShift(), a następnie oblicza rozmiar i orientację obiektu.

cvSnakeImage(…) – funkcja ta służy do aktualizacji węża (zbiór uszeregowanych punktów opisujących kontur, na który oddziałują różne siły dążący do zminimalizowania jego całkowitej energii). Wyodrębniane kształtów jest po prostu minimalizacją tej całkowitej energii (Rysunek 39):

$$E = E_{int} + E_{ext}$$

gdzie E_{int} jest wewnętrzną energią ukształtowaną przez konfiguracje węża, natomiast E_{ext} jest zewnętrzną energią uformowaną przez zewnętrzne ograniczenia nałożone na węża (na przykład człowieka nadzorującego cały proces). Funkcja oparta jest na algorytmie [17].



Rysunek 39. Obraz oryginalny (a), wyodrębniony kontur po 1 iteracji metodą aktywnego konturu (b), wyodrębniony kontur po 14 iteracjach metodą aktywnego konturu (c), wyodrębniony kontur po 30 iteracjach (osiągnięto minimum) metodą aktywnego konturu (d).

3.1.18.3. Przepływ optyczny

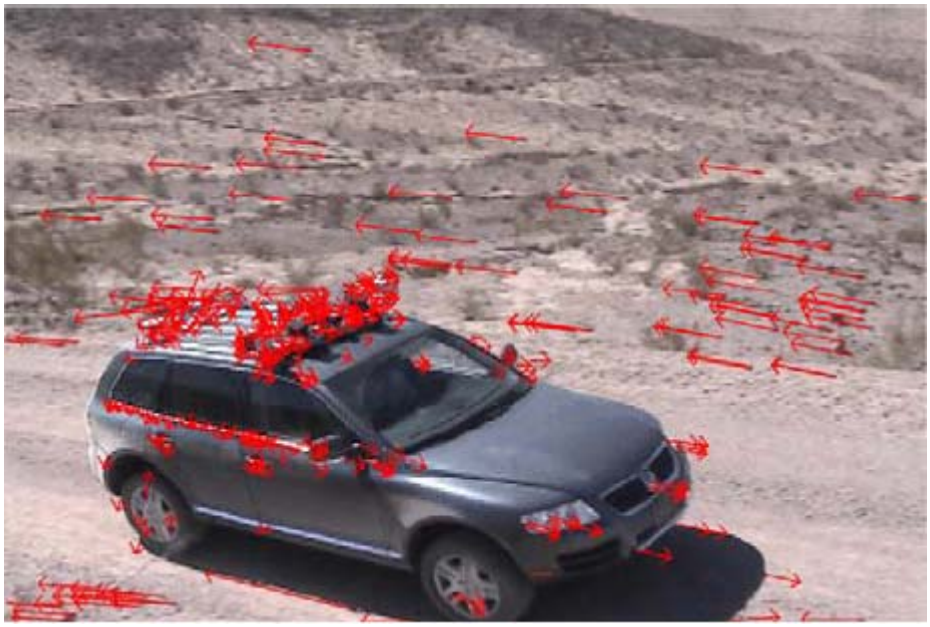
Do śledzenia obiektów zaimplementowano również cztery funkcje liczące przepływ optyczny dla dwóch obrazów:

cvCalcOpticalFlowHS(…) - oblicza przepływ optyczny przy użyciu algorytmu Horn – Schunck [16], w którym są wykonywane obliczenia z wykorzystaniem pochodnych wyższego rzędu.

cvCalcOpticalFlowLK(…) - oblicza przepływ optyczny przy użyciu algorytmu Lucas – Kanade [19] polegającym na łączeniu przylegających pikseli w grupy przy założeniu, że mają jednakową prędkość

cvCalcOpticalFlowBM(…) - oblicza przepływ optyczny przy użyciu metody dopasowania blokowego. Dla każdego bloku z obrazu poprzedniego funkcja ta próbuje znaleźć podobny blok w obrazie następnym w jakimś sąsiedztwie oryginalnego bloku.

cvCalcOpticalFlowPyrLK(…) - oblicza przepływ optyczny przy użyciu rzadkiej, iteracyjnej wersji algorytmu Lucas – Kanade w piramidach [2]. Rysunek 40 ilustruje przepływ optyczny wynikający z ruchu obiektu uzyskany przy pomocy tejże funkcji.



Rysunek 40. Obraz ilustruje przepływ optyczny wynikający z ruchu obiektu uzyskany przy pomocy funkcji cvCalcOpticalFlowPyrLK(…).

3.1.18.4. Estymacja

W punkcie tym opisano grupę funkcji służącą do estymacji stochastycznych modeli stanu. OPENCV dostarcza dwóch estymatorów: filtr Kalmana [44] [29] oraz kondensacje.

Do przechowywania stanu filtru Kalmana w OPENCV używana jest struktura CvKalman. Do utworzenia tej struktury służy funkcja cvCreateKalman(…), funkcje cvKalmanPredict(…) i cvKalmanCorrect(…) służą do aktualizacji modelu stanu, natomiast funkcja cvReleaseKalman zwalniania struktury CvKalman.

Do trzymania stanu warunkowego propagowania gęstości (ang. CONditional DENSity propagATION) [30] [29] służy struktura CvConDensation. Funkcja cvCreateConDensation(…) tworzy strukturę CvConDensation oraz zwraca wskaźnik do niej, cvReleaseConDensation(…) służy do zwalniania tej struktury, natomiast cvConDensInitSampleSet(…) wypełnia przykładowe tablice wartościami o określonym zasięgu, natomiast cvConDensUpdateByTime(…) dokonuje estymacji następnego stochastycznego modelu stanu na podstawie aktualnego stanu.

3.2. IPP

INTEL IPP jest komercyjna biblioteką wydaną pod licencją INTEL, pełna nazwa to INTEL Integrated Performance Primitives. IPP oferuje programistom szeroki zakres funkcji niskiego poziomu, które zostały specjalnie zoptymalizowane pod procesory firmy INTEL, funkcje te korzystają z zalet tych procesorów. IPP jest bardzo dobrą biblioteką do przetwarzania obrazów, sygnałów oraz dźwięku z wysoce wydajnymi funkcjami. Godne uwagi jest to, iż OPENCV jest tak stworzona, że może współpracować z IPP i może być zoptymalizowana przez załadowanie komercyjnej biblioteki IPP. OPENCV automatycznie wykrywa oraz używa IPP jak tylko jest ona zainstalowana, przy czym nie wymagana jest zmian ani nawet rekompilacja kodu źródłowego użytkownika. Mechanizm zamiany w zoptymalizowany kod IPP jest prosty. Używa on wskaźników funkcji oraz dynamicznej biblioteki - załadowane programy wspomagające, które dostarczone przez system operacyjny (OS). Dla każdej funkcji, którą może użyć OPENCV występuje wskaźnik do funkcji, który pierwotnie ustawiony jest na NULL i który jest przydzielany do właściwego adresu, kiedy odpowiednie komponenty IPP są wykryte i załadowane. Aktualnie OPENCV może używać przeszło 300 funkcji IPP. Funkcje OPENCV są zaimplementowane na dwa sposoby: z lub bez odwołań do funkcji IPP. W pierwszym przypadku użytkownik musi mieć zainstalowaną bibliotekę IPP, aby skorzystać z jej możliwości, w drugim przypadku wszystkie funkcjonalności są zawarte w wersji OPENCV.

Poprzednikiem IPP jest IPL (Intel Image Processing Library). W IPP podstawową struktura danych do przechowywania obrazu nie jest już IplImage, ponieważ w zoptymalizowanych aplikacjach odczyt i zapis pól IplImage jest zadaniem czasochłonnym. Podstawową strukturą IPP jest "ipp image", który jest wskaźnikiem do specjalnego typu danych. Wskaźnik ten jest tylko adresem w pamięci i wymaga zarządzania jego rozmiarem oraz właściwościami poza strukturą. Powoduje to, iż programista jest bardziej odpowiedzialny za operowanie własnymi obrazami, ale z drugiej strony wewnętrzne funkcje IPP operują tylko wskaźnikiem, co pociąga za sobą oszczędność czasu potrzebną do zoptymalizowania aplikacji. Należy zwrócić uwagę, iż IPP nie posiada graficznego interfejsu użytkownika.

Poniżej znajduje się Tabela 1 z przybliżonymi opisami możliwości zwiększenia wydajności na procesorze INTEL, które użytkownik może uzyskać korzystając z IPP.

Na szeroki zakres wzrostu wydajności w tabeli tej wpływają różne typy potencjalnych obrazów. Obrazy typu Byte szybciej są przetwarzane niż obrazy typu Integer, które są z kolei szybciej przetwarzane od obrazów typu Float. Innym powodem różnic czasowych są wyniki przy zastosowaniu różnych rozmiarów jądra. Operacje z małymi jądrami są szybsze niż z dużymi jądrami.

Tabela 1. Tabela obrazująca wzrost wydajności poszczególnych funkcji przy zastosowaniu kombinacji bibliotek OPENCV z IPP.

Funkcja	**Wzrost wydajności przy zastosowaniu kombinacji bibliotek OPENCV z IPP**
Morfologia	~ 3 - 7
Filtr medianowy	~ 2,1 - 18
Konwolucja	~ 2 - 8
Konwersja kolorów	~ 1 - 3
Momenty	~ 1,5 - 3
Wykrywanie rogów	~ 1,8
Wykrywanie krawędzi	~ 2 - 3
Operacje matematyczne	~ 3 - 10
Porównywanie obszarów	~ 1,5 - 2

Porównując OPENCV z IPP, IPP jest szybsza, ale działa wyłącznie na procesorach INTEL. IPP użyta w kombinacji z OPENCV jest bardzo dobrym narzędziem do przetwarzania obrazów i wizji komputerowej. Są one uzupełniającymi się narzędziami do budowy wydajnych i skutecznych aplikacji przetwarzania obrazu i wizji komputerowej.

3.3. IPL98

3.3.1. Przedstawienie

IPL98 [37], pełna nazwa brzmi Image Processing Libary została stworzona w Instytucie Technologii Produkcji imienia Mc-Kinneya Mollera na Uniwersytecie Południowej Dani przez René Dencker Eriksena. Biblioteka z założenia ma być łatwa w użyciu, niezależna od platformy sprzętowej jak również programowej, przyjazna przy wprowadzaniu nowych kodów i algorytmów oraz szybka w działaniu. Aby spełniać wymagania dotyczące bycia łatwym i szybkim, biblioteka często posiada więcej niż jeden sposób realizacji poszczególnych zagadnień, jak również dostarcza elastycznego wyboru co do realizacji tegoż zagadnienia np. posiada kilka różnych dróg dostępu do pikseli w obrazie.

3.3.2. Wersje oraz dokumentacja

Aktualna wersja 2.20 biblioteki pochodzi z 5 stycznia 2006 roku. Początki istnienia IPL98 datuje się na sierpień 1998 roku, kiedy ukazała się pierwsza wersja 0.1. Do tej pory ukazało się osiemnaście wersji biblioteki, co świadczy o aktywności osób tworzących.

Na stronie poświęconej bibliotece [37] znajduje się dokumentacja. Wśród niej można znaleźć tutorial z sierpnia 2003 roku (niestety do wersji 2.14, choć można się nim posługiwać również przy nowszych wersjach) zawierający opis struktury biblioteki, instalacji oraz kompilacji, algorytmów i wiele innych przydatnych informacji. Na stronie domowej biblioteki znajdziemy także historię (jak powstawały kolejne wersje i co nowego zawierały), manual z wykazem i opisem wszystkich klas, funkcji oraz zmiennych. Natomiast wszystkie wersje IPL98, można znaleźć na stronie internetowej www.Sourceforge.net [38].

3.3.3. Systemy operacyjne, kompilatory

IPL98 z założenia jest niezależna systemowo, co nie zawsze sprawdza się w praktyce. Sukcesywnie została przetestowana pod Windowsem oraz Linuxem. Od wersji 2.20 biblioteka jest przeznaczona do użytku pod kompilatorami GCC 3.2 (lub

nowszy) oraz Visual C++ .Net 2003 (lub nowszy), jednakże nadal działa pod Visual C++ 6.0 i Borland Builder C++ 6.0.

3.3.4. Instalacja

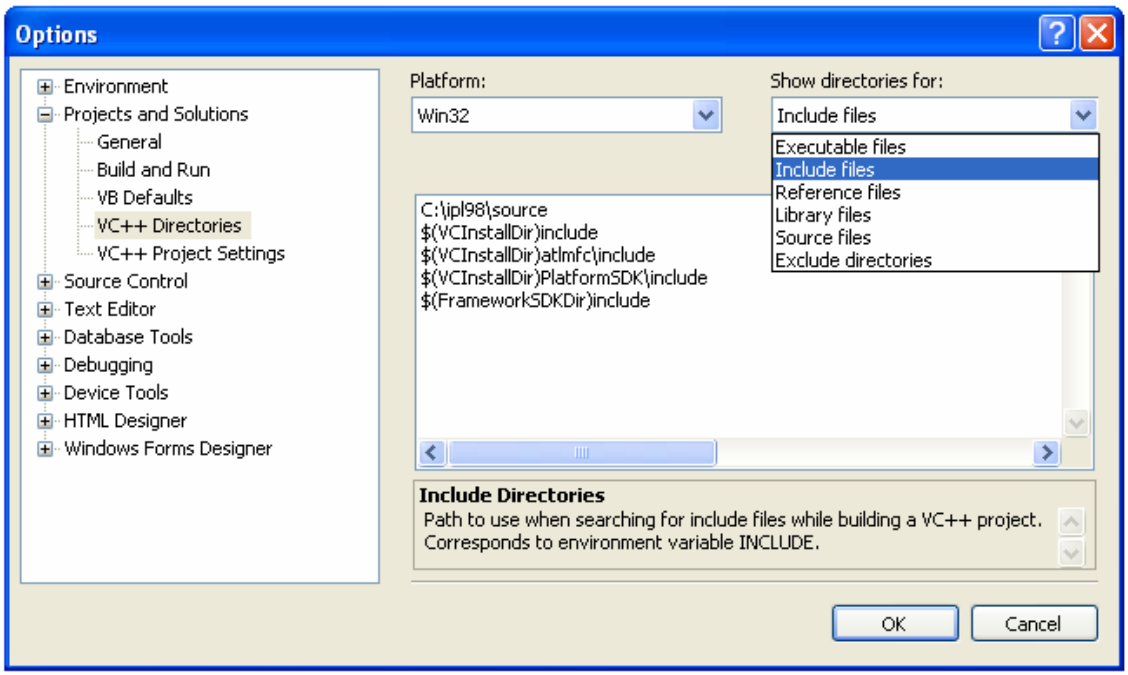
Biblioteka IPL98 może być użyta jako biblioteka wyłącznie dla języka C, jak również dla języka C++. W obydwu przypadkach należy ściągnąć ze strony internetowej [38] najnowszą wersje biblioteki, którą należy rozpakować oraz umieścić na dysku lokalnym C:\.

Aby w sposób łatwy, lecz mało elegancki móc skorzystać z części biblioteki przeznaczonej wyłącznie do programowania w C należy do utworzonego wcześniej własnego projektu dołączyć pliki /ipl98/ipl98.h oraz /ipl98/ipl98.c, zawierające wszystkie inne niezbędne pliki. Następnie zawrzeć plik /ipl98/ipl98.h w plikach źródłowych własnego projektu, gdzie biblioteka będzie użyta. Jednakże sposób ten sprawia, że kompilacja trwa zbyt długo, aby ją przyśpieszyć należy zbudować bibliotekę albo dołączyć do projektu wyłącznie pliki niezbędne przy tworzeniu projektu.

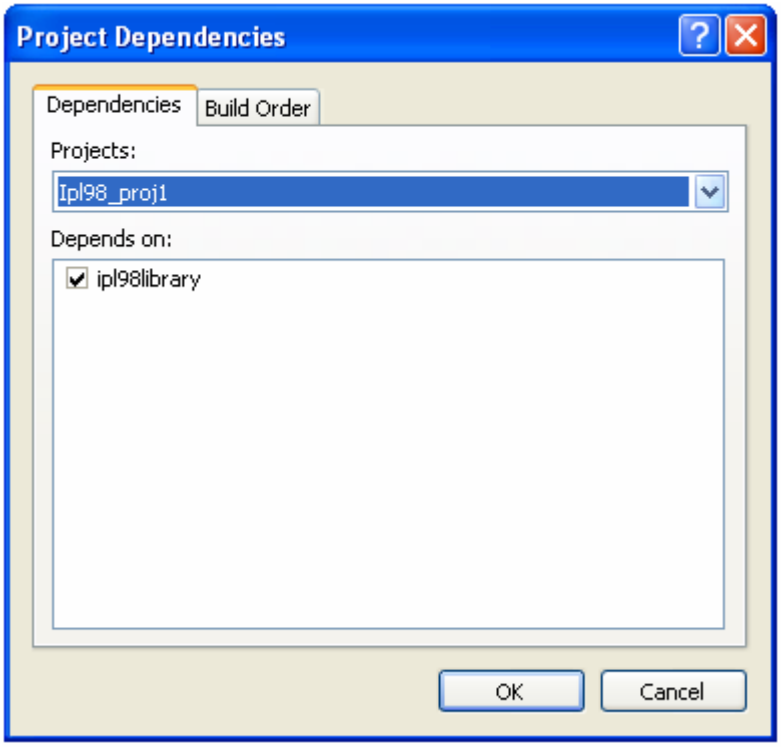
W akapicie tym zostały opisane czynności, jakie trzeba wykonać, aby skompilować oraz używać bibliotekę IPL98 na kompilatorze MS Visual C++ 2005.NET programując w C++. Instrukcja tam może być przydatna podczas kompilacji biblioteki na innych kompilatorach, szczególnie dla Visual C++ 6.0, gdzie proces ten przebiega bardzo podobnie. Od wersji 2.0 proces instalacji biblioteki uległ zmianie, aktualna wersja biblioteki zawiera kod źródłowy (ang. source code) oraz pliki projektu budującego bibliotekę (lib) generowane przez projekt odpowiedzialny za zbudowanie biblioteki na podstawie plików źródłowych. Zmiany te wpłynęły na skrócenie czasu kompilacji. Pierwszą czynnością, jaką należy wykonać jest ustawienie ścieżki dostępu do plików źródłowych (ang. source files) oraz plików projektu budującego bibliotekę (lib). W tym celu należy otworzyć okno dialogowe *Options* (Rysunek 41) wybierając zakładkę *Tools->Options*, kliknąć zakładkę *Project and Solutions-> VC++ Directories* i w polu *Show directories for* wybrać *Include files* i dodać ścieżkę dostępu c:/ipl98/source. Następnie wybrać *Library files* i dodać ścieżkę dostępu c:/ipl98/lib oraz wybrać *Source files* i dodać ścieżkę c:/ipl98/source. Teraz można przystąpić do tworzenia projektu. Po utworzeniu własnego – nowego - pustego projektu należy dołączyć do przestrzeni roboczej jeden z dwóch projektów odpowiedzialnych za zbudowanie plików biblioteki. Umieszczone są one w c:\ipl98\projects\visualc.

Następnie ustawić zależności pomiędzy dwoma projektami wybierając zakładkę *Project-> Project Dependencie*s (Rysunek 42). Zaletą Visual C++ jest możliwość ustawienia zależności pomiędzy dwoma różnymi projektami w jednej przestrzeni roboczej, sprawia to, że kompilator dokonuje tworzenia wersji wykonywalnej z kodu źródłowego automatycznie i w odpowiedniej kolejności z różnych projektów. Na koniec należy zapewnić własnemu projektowi status aktywnego projektu klikając prawym przyciskiem myszy na nazwę tego projektu i wybierając *Set as StartUp Project*.

Do biblioteki dostarczone są również przykładowe projekty, ulokowane na c:\ipl98\examples. Aktualnie znajdują się tam dwie przestrzenie robocze zawierające każda po dwa projekty, jeden jest przykładowym projektem a drugi projektem budującym pliki biblioteki.



Rysunek 41. Widok okna dialogowego *Options* podczas ustawiania ścieżek dostępu do plików źródłowych oraz plików projektów budujących bibliotekę.



Rysunek 42. Widok okna dialogowego *Project Dependencies* podczas ustawiania zależności między projektami.

3.3.4.1. Błędy przy instalacji

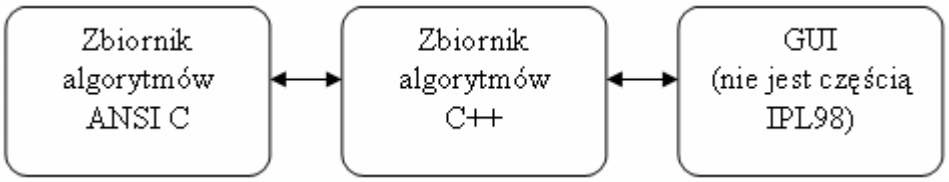
W wyniku niedokończonych prac przystosowujących bibliotekę do wersji .NET po zainstalowaniu biblioteki otrzymuje się szereg błędów i ostrzeżeń. Do pracy została dołączona wersja biblioteki, z której usunięto błędy i która kompiluje się prawidłowo pod wersją MS Visual C++ 2005.

Często też przyczyną błędów podczas rozpoczynania pracy w Visual C++ jest niekompatybilność bibliotek run-time. Oznacza to, że nie można połączyć dwóch części kodu programu z różnych bibliotek run-time. Dopóki jeden projekt korzysta z jednej biblioteki run-time błędy te nie powstają, natomiast w momencie, kiedy używane są pliki biblioteki (lib) sytuacja ulega zmianie. Wtedy należy zapewnić, aby podczas kompilacji własnego kodu była używana ta sama biblioteka run-time co podczas kompilacji plików biblioteki (lib). W tym celu należy wybrać właściwości projektu (ang. Properties) zakładkę *C/C++*, wybrać *Code Generation*, a następnie w polu *Use run-time library* wybrać odpowiednią z czterech kombinacji. Dla aktualnej wersji biblioteki Ipl98 2.20 należy wybrać *Multi-threaded Debug DLL (/MDd)*. W punkcie tym został opisany typowy błąd dla kompilatora Visual C++, jednakże wiedza tu zawarta może być przydatna przy pracy z innymi kompilatorami.

3.3.5. Struktura

W tym rozdziale zostanie wyjaśniona podstawowa struktura danych w IPL98, zawierająca relacje pomiędzy częścią biblioteki napisaną w C i częścią napisaną w C++, definicje podstawowych typów oraz umiejscowienie plików.

Jądro biblioteki zawierające większość podstawowych i niezbędnych metod zostało zakodowane w ANSI C. Kod w C jest zorientowany jak to tylko możliwe obiektowo z wszystkimi atrybutami klas opakowanymi w jedną strukturę oraz wszystkimi funkcjami operującymi na tym kodzie. Część napisana w ANSI/ISO C++ opakowuje te metody i dodaje do nich nowe funkcjonalności. Zarówno dla kodu w C jak i w C++ biblioteka została podzielona na dwie logiczne części: zbiornik klas oraz zbiornik algorytmów (Rysunek 43). Zbiornik klas zawiera kod potrzebny do tworzenia lub załadowania obrazu w pamięci RAM, dostępu do pikseli oraz innych informacji dotyczących obrazu np. głębia w bitach. Zbiornik algorytmów zawiera kilka klas, pogrupowanych względem dziedzin przetwarzania obrazów. Żeby uniezależnić bibliotekę od platformy sprzętowej i programistycznej interfejs GUI nie jest dołączony do biblioteki.



Rysunek 43. Relacje pomiędzy częściami C i C++ biblioteki IPL98.

3.3.5.1. Podstawowe typy danych

W IPL98 dostępne są następujące typy danych reprezentujące wartości każdego piksela obrazu (Tabela 2):

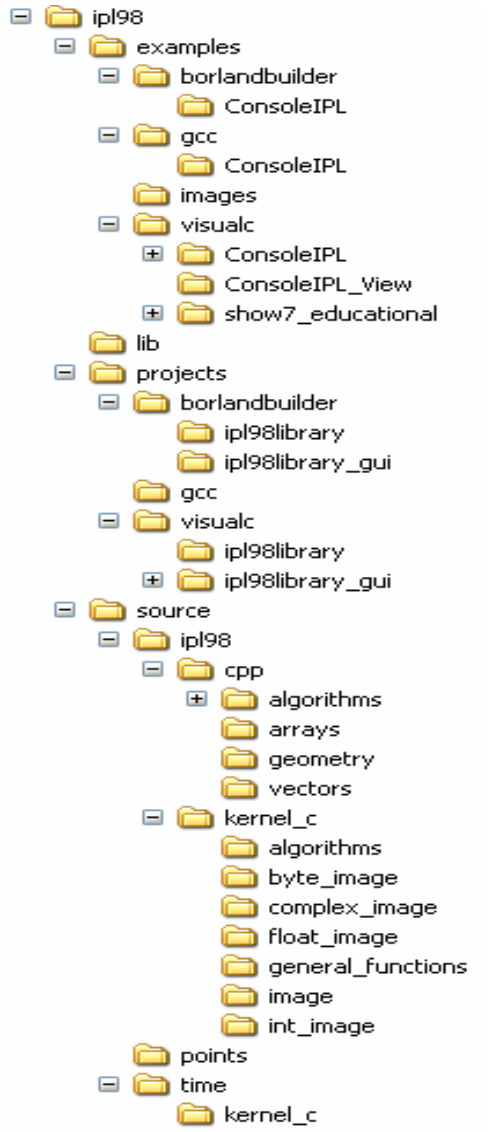
Tabela 2. Typy danych reprezentujące wartości pikseli obrazu.

Nazwa typu	Opis
UINT8	**8-bitowa liczba całkowita bez znaku**
INT8	**8-bitowa liczba całkowita ze znakiem**
UINT16	**16-bitowa liczba całkowita bez znaku**
INT16	**16-bitowa liczba całkowita ze znakiem**
UINT32	**32-bitowa liczba całkowita bez znaku**
INT32	**32-bitowa liczba całkowita ze znakiem**
FLOAT32	**32-bitowa liczba zmiennoprzecinkowa o pojedynczej precyzji**

3.3.5.2. Układ folderów

Budowa biblioteki przedstawiona jest na Rysunek 44. Cały kod źródłowy biblioteki znajduje się w folderze *source*, w folderze *projects* znajdują się projekty, które odpowiadają za zbudowanie biblioteki na podstawie kodu źródłowego oraz generują pliki biblioteki, które są umieszczane w folderze *lib*. Dla kompilatorów Visual C++ oraz Borland C++ Builder dostępne są po dwa różne projekty budujące bibliotekę, pierwszy nazywa się ipl98library, natomiast drugi ipl98library_gui. Dla kompilatora GCC dostarczony jest plik makefile.

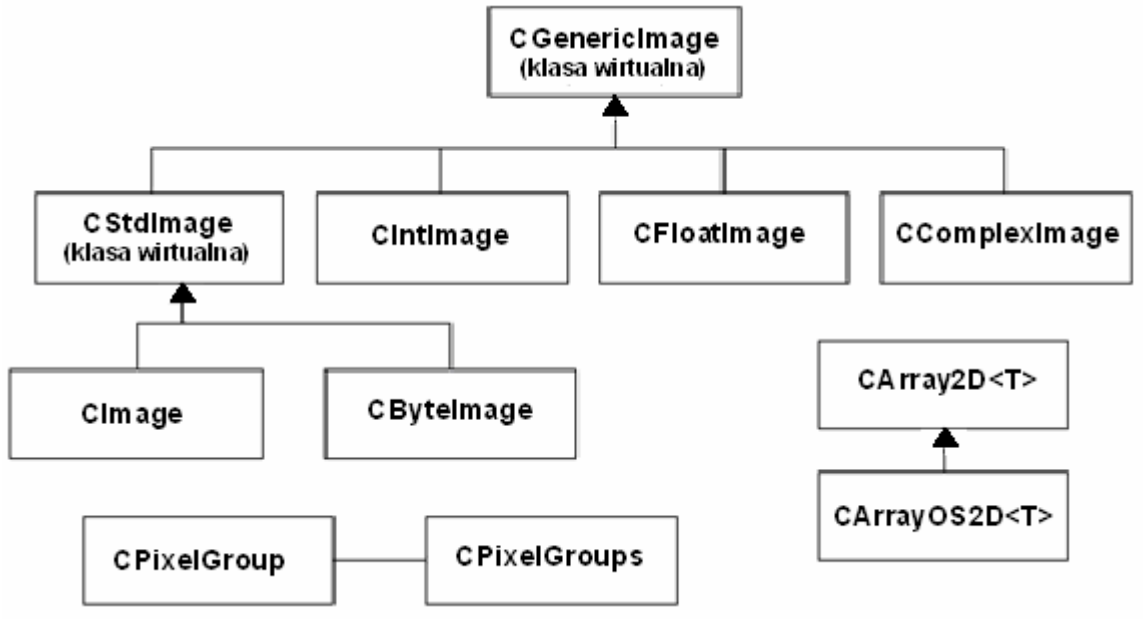
W folderze *examples* znajdują się przykładowe projekty dla trzech kompilatorów, ukazujące jak prawidłowo korzystać z biblioteki. Aktualnie dostępne są projekty dla kompilatorów Visual C++, Borland C++ Builder oraz GCC. Niestety projekty dla Visual C++ nie są jeszcze uaktualnione do działania pod Visual C++.NET. Do każdego projektu dołączony jest plik *readme.txt*, zawierający opis użycia danego projektu.



Rysunek 44. Układ folderów biblioteki IPL98.

3.3.5.3. Sposoby przechowywania obrazu w pamięci

Zbiornik klas (Rysunek 45), który jest jądrem biblioteki IPL98 zawiera informacje o obrazie i daje użytkownikowi łatwy i szybki dostęp do danych w nim zawartych.



Rysunek 45. Zbiornik klas IPL98.

W IPL98 istnieją dwa sposoby przechowywania danych obrazu w pamięci RAM. Mianowicie poprzez klasę CPixelGroup oraz poprzez klasy wywodzące się z wirtualnej klasy CGenericImage, odwzorowujące obrazy w postaci wewnętrznych macierzy.

Opis przechowywania obrazu w klasach odwzorowujących obraz w postaci wewnętrznych macierzy

CStdImage, która występuje jako argument w większości algorytmów operujących na obrazach zawartych w bibliotece również jest wirtualną klasą, a pochodzą od niej klasy CImage i CByteImage. Najczęściej używaną klasą obrazu jest CImage zawierająca tzw. obrazy normalne, które aktualnie mogą zostać wyświetlone na ekranie. Klasa ta może przechowywać trzy różne typy pikseli (1, 8, 24-bitowe), gdzie liczba bitów użyta do reprezentowania pikseli zwana jest głębią w bitach lub liczbą bitów na piksel. Bardziej wyspecjalizowaną klasą jest CByteImage, która jest bardziej wydajna, lecz nadal zawiera normalne dane obrazu. Klasa ta może reprezentować tylko piksele w 8-bitach. Zaletą tej klasy jest szybkość działania, umożliwia ona szybszy dostęp do pikseli. Natomiast klasy CIntImage (do reprezentacji każdego piksela użyta jest 16-bitowa liczba całkowita ze znakiem), CFloatImage (do reprezentacji każdego piksela użyta jest 32-bitowa liczba zmiennoprzecinkowa o pojedynczej precyzji) i CComplex-Image (do reprezentacji każdego piksela użyta jest liczba zespolona, składająca się z 32-bitowej liczby zmiennoprzecinkowej o pojedynczej precyzji zarówno dla składowej rzeczywistej, jak i urojonej) są specjalnymi klasami

zawierającymi różne dane obrazów i wartości poszczególnych pikseli w tych klasach, nie mogą być bezpośrednio interpretowane jako kolor. Klasy te przydatne są podczas wewnętrznego przetwarzania obrazu np. podczas odejmowania jednego obrazu od drugiego, kiedy wartości mogą być ujemne i wtedy wyniki należy zapisać w klasie CIntImage. CFloatImage może być przydatna np. podczas transformacji pikseli, kiedy wartości po przekształceniach nie są liczbami całkowitymi (Tabela 3).
Klasę CArray2D<T> różni od poprzednich klas to, iż jest ona szablonem, w którym można użyć własnego typu <T> zawierającego dane w tej klasie.

Tabela 3. Typy pikseli dla poszczególnych klas.

Nazwa klasy	Typ pikseli
CImage	**1,8 lub 16-bitowa liczba całkowita bez znaku**
CByteImage	**8-bitowa liczba całkowita bez znaku**
CIntImage	**16-bitowa liczba całkowita ze znakiem**
CFloatImage	**32-bitowa liczba zmiennoprzecinkowa o pojedynczej precyzji**
CComplexImage	**2* 32-bitowa liczba zmiennoprzecinkowa o pojedynczej precyzji**

Przechowywanie informacji o obrazie w CPixelGroup

Innym sposobem gromadzenia informacji o obrazie jest klasa CPixelGroup. Zawiera ona listę pozycji pikseli oraz opcjonalnie może zawierać kolor dla każdej pozycji. Klasa ta zapewnia szybszy dostęp do powiązanych pikseli. Do zalet tej klasy można zaliczyć również to, iż w pamięci gromadzone są wyłącznie interesujące użytkownika wartości pikseli. Wadą jest natomiast to, że grupa pikseli nie może być bezpośrednio wyświetlona, najpierw musi ulec konwersji do kasy CImage lub CByteImage.

Nową, pustą CPixelGroup tworzymy się za pomocą domyślnego konstruktora CPixelGroup(). Jeżeli programista zna całkowitą liczbę pozycji, jaka będzie przechowywana w grupie, szybsze będzie zainicjalizowanie grupy pikseli konstruktorem CPixelGroup(InitialSize) - sprawi to, że pamięć zostanie zaalokowana w jednym fragmencie, co zaoszczędzi czas. Dodawanie i usuwanie pozycji może zostać wykonane na dwa sposoby. Wolniejsze metody to InsertPosition(...) - dodawanie pozycji oraz RemovePositionSlow(...) - usuwanie pozycji, w obydwu tych metodach

zachowane jest porządek pozycji (pikseli). Szybszymi metodami są: AddPosition(...) - nowe pozycje zawsze dodaje na końcu oraz RemovePosition(...) - usuwa pozycję o danej współrzędnej poprzez przeniesienie ostatniej pozycji w tej grupie do tej współrzędnej.
Dodawanie kolorów do grupy pikseli realizuje funkcja AddColors(const CStdImage& Img), której argumentem jest obraz źródłowy, Pobiera ona kolor pikseli z odpowiednich pozycji obrazu źródłowego oraz wstawia go do grupy pikseli. Kolor może zostać również dodany ręcznie funkcją AllocColors() wykonaną po funkcji AddPositionColor(...) lub InsertPositionColor(...). Może również w każdej chwili zostać usunięty przez funkcję RemoveColors(). Do kopiowania grupy pikseli do obrazu dostępne są dwie metody: CopyToImage(…) oraz AddToImage(...).

CPixelGroups - Zbiornik dla kilku grup pikseli

Często podczas prac z różnymi projektami zmuszeni jesteśmy do pracy z kilkoma CPixelGroup, dlatego też została w IPL98 utworzona klasa CPixelGroups, która przechowuje obiekty CPixelGroup. W wielu przypadkach klasa ta pracuje tak samo jak klasa CPixelGroup. Domyślnie pamięć jest alokowana do przechowywania 20 obiektów CPixelGroup oraz dostosowuje swój rozmiar analogicznie jak CPixelGroup dostosowuje liczbę pozycji. Konstruktor CPixelGroups(InitialSize) umożliwia zainicjalizowanie rozmiaru w celu oszczędności czasu. Do dodawania grupy pikseli służy funkcja InsertGroupInGroups(...), która jest wolna, ale zachowuje porządek grup oraz szybsza AddGroup(...). Do usuwania grupy pikseli służy RemoveGroupSlow(...), która jest wolniejszą funkcją niż RemoveGroup(...), ale zachowuje porządek grup. Do pobrania grupy o odpowiedniej pozycji służy GetGroupWithPos(…).

3.3.6. Praca z obrazami

3.3.6.1. Ustawienie granicy wokół obrazu

We wszystkich klasach obrazu dostępne jest ustawienie granicy (ang. border) wokół obrazu, co przekłada się to na łatwiejsze wykonywanie pewnych procedur manipulacyjnych na obrazie. Np. kiedy używana jest maska filtru dla całego obrazu, a granica ustawiona jest do połowy rozmiaru maski filtru oraz posiada ten sam kolor co

tło, w takim przypadku nie trzeba martwić się o uzyskiwanie dostępu do pikseli znajdujących się na zewnątrz obrazu, kiedy maska jest blisko krawędzi.

3.3.6.2. Odczyt i zapis obrazu do plików

Bezpośrednio załadowane ze standardowych plików graficznych (BMP oraz PGM) oraz zapisane w tych plikach mogą być tylko dwie klasy obrazu: CImage i CByteImage. Natomiast dwie pozostałe klasy CIntImage oraz CFloatImage nie mogą zostać zapisane w tych dwóch standardowych formatach graficznych plików. Wartości poszczególnych pikseli obrazów przechowywane w tych dwóch klasach mogą zostać zapisane w pliku TXT.

3.3.6.3. Operacje na obrazach

Do kopiowania zawartości z jednego obrazu do innego służy operator przypisania „=". Może on wystąpić pomiędzy obrazami o tej samej głębi w bitach, możliwe też jest kopiowanie pomiędzy klasami CImage oraz CByteImage dopóki dana głębia w bitach jest dopuszczalna w obrazie docelowym.

Do zmieniania głębi obrazów zarówno dla klasy CImage jak i CByteImage służy funkcja CopyConvert(…). Funkcja ta pobiera dwa parametry, pierwszym jest docelowa głębia, natomiast drugim obraz źródłowy, który może być zarówno klasy CImage jak i CByteImage. W przypadku, gdy wykonywana jest konwersja do mniejszej głębi niektóre informacje o obrazie mogą zostać utracone.

Funkcja CopySubImage(…) kopiuje określony obszar z obrazu źródłowego do obrazu docelowego, jednakże działa ona wolniej, a jest to spowodowane tym, iż wewnętrzna kopia obrazu musi zostać zaalokowana.

3.3.6.4. Operacje na pikselach

Podstawową czynnością podczas przetwarzania obrazów jest uzyskanie dostępu do odpowiednich pikseli w obrazie oraz pobranie wartości kolorów tych pikseli. Aby wyjaśnić to zagadnienie posłużono się klasą CImage. Wartości poszczególnych pikseli zgromadzone w obrazie posiadają dwa sposoby interpretacji: dla głębi 24–bitowej

zwracana jest wartość składająca się z 3 bajtów zawierających składowe RGB, natomiast dla 1-bitowej oraz 8-bitowej głębi obrazów wartości pikseli zawierają indeksy w palecie. Dla głębi 1-bitowej zwracane wartości to 0 lub 1, a dla 8-bitowej wartości z zakresu 0-255 (również dla klasy CByteImage). Kiedy tworzony jest obraz o głębi 1-bitowy lub 8-bitowy budowana jest domyślna paleta. Dla 1-bitowej indeks 0 odpowiada w palecie (R,G,B)=(0,0,0), czyli jest kolorem czarnym, natomiast indeks 1 odpowiada RGB=(255,255,255), czyli jest to kolor biały. Natomiast dla obrazów monochromatycznych o głębi 8-bitowej indeks równa się każdej wartości składowej R, G, B w palecie np. indeks 65 odpowiada składowym (R,G,B)=(65,65,65). Tak więc w przypadku obrazów monochromatycznych indeks może być traktowany bezpośrednio jako kolor.

Paleta

Paleta dostępna jest tylko dla klasy wirtualnej CStdImage poprzez atrybut m_Pal typu CPalette. Zbiór klas CPalette dostarcza metod potrzebnych do operowania na palecie. Wartość koloru zgromadzona w palecie jest zawsze typu UINT32 i dostępna jest za pomocą GetColor(…), a wartości poszczególnych składowych R, G, B koloru dostępne są za pomocą GetRedVal(...), GetGreenVal(...) i GetBlueVal(...) i są typu UINT8.

Dostęp do pikseli

Domyślnie w układzie współrzędnych początek układu współrzędnych (parametr origo) jest ustawione w lewym górnym rogu i można go zmienić za pomocą metody SetOrigo(x,y) dostępnej dla klas CImage, CIntImage i CFloatImage. Do pobierania minimalnej i maksymalnej współrzędnej x i y służą odpowiednio GetMinX(), GetMinY(),GetMaxX() oraz GetMaxY().

Bardzo przydatną funkcją w IPL98 jest możliwość wybrania prostokątnego, aktywnego obszaru w obrazie, który będzie podlegał przekształceniom, zwanego ROI (ang. Region of Interest - obszar zainteresowania). Godne zauważenia jest, iż w obrazie zawierającym ROI metody GetMinX(), GetMinY(),GetMaxX() oraz GetMaxY() zwrócą granice zgodne z ROI.

Dostępnych jest kilka sposobów odczytywania oraz ustawiania wartości pikseli. Najbardziej powszechnym jest SetPixel(x,y,Color) i GetPixel(x,y), a funkcje te zwracają błąd, gdy pozycje piksela (x,y) wychodzi poza zakres obrazu. Dostępne są

również szybsze funkcje do ustawiania oraz czytania wartości pikseli SetPixelFast(x,y,Color) i GetPixelFast(x,y), jednakże funkcje te nie sprawdzają czy pozycje pikseli nie sięgają poza zakres obrazu.

Dostęp do pikseli obrazu można uzyskać poprzez:

- Wersja wolniejsza

```
CImage Img(640,480,8);      //utworzenie obrazu
UINT32 s;
Img1.Flush(123);            //ustawienie wartości 123 wszystkim
                            pikselom
s=Img.GetPixel(i,j);        //pobranie wartości piksela
Img.SetPixel(i,j,s);        //ustawienie wartości piksela
```

- Wersja szybsza

```
CImage Img(640,480,8);      //utworzenie obrazu
UINT32 s;
Img1.Flush(123);            //ustawienie wartości 123 wszystkim
                            pikselom
s=Img.GetPixelFast(i,j);    //pobranie wartości piksela
Img.SetPixelFast(i,j,s);    //ustawienie wartości piksela
```

Możliwy jest również dostęp do pikseli za pomocą iteratorów.

Dostęp do pikseli za pomocą iteratorów

Aby zoptymalizować dostęp do pikseli w IPL98 dostarczono iteratory dla wszystkich klas obrazów.

W programowaniu obiektowym iteratorem nazywamy obiekt pozwalający na sekwencyjny dostęp do wszystkich elementów lub części zawartych w innym obiekcie i można go rozumieć jako rodzaj wskaźnika udostępniającego dwie podstawowe operacje: odwołanie się do konkretnego elementu w kolekcji (dostęp do elementu) oraz modyfikację samego iteratora tak, by wskazywał na kolejny element (sekwencyjne przeglądanie elementów). Podstawowym celem iteratora jest pozwolić użytkownikowi przetworzyć każdy element w kolekcji bez konieczności zagłębiania się w jej wewnętrzną strukturę.

Pierwszą czynnością podczas pracy w IPL98 z wykorzystaniem iteratorów jest określenie typu potrzebnego iteratora. Iteratory te mogą wskazywać na wiersz pikseli w

obrazie lub tez na kolumnę pikseli w obrazie, przy czym iteratory wskazujące na wiersz są szybsze, co uwarunkowane jest sposobem przechowywania obrazów w pamięci i należy je stosować zawsze, kiedy jest to możliwe. Iteratory mogą być iteratorami do stałej (pozwalają na zmianę kolekcji) lub do zmiennej (nie pozwalają na zmianę kolekcji). Natomiast przy wyborze interatora dla klasy CImage dodatkowo trzeba wziąć pod uwagę głębię pikseli.

3.3.7. Filtracja

W punkcie tym przedstawione zostały funkcje biblioteki IPL98 związane z wykonywaniem konwolucji (splotu) oraz filtracji obrazów [14] [40]. Wszystkie te funkcje są zawarte w klasie CMakOperation.

Operacja konwolucji realizowana jest przez funkcję Convolve(…) oraz ConvolveFast(…). Funkcja Convolve(…) dokonuje konowlucji obrazu klas CStdImage, CIntImage o głębi 8-bitowej oraz klasy CFloatImage z maskami klasy CFloatImage. Natomiast funkcja ConvolveFast (…), wykonuje konwolucje obrazu klas CStdImage, CIntImage o głębi 8-bitowej z maską rozmiarów 3x3 klasy CFloatImage i działa bardzo szybko, gdy elementy maski mają wartość 0 lub 1. Przy pomocy tych funkcji można implementować dowolne filtry liniowe.

Do filtracji obrazów zaimplementowane zostały funkcje realizujące filtr rozmywający oraz medianowy z maskami na stałe zdefiniowanymi w bibliotece.

MedianFilterFourConnected3x3(…) - realizuje filtr medianowy z maską czterospójną, działa na obrazach klasy CStdImage o głębi 8-bitowej oraz klasy CIntImage.
MedianFilterDiagonalConnected3x3(…)- realizuje filtr medianowy z maską w postaci krzyżowej, działa na obrazach klasy CStdImage o głębi 8-bitowej oraz klasy CIntImage.

BlurFourConnected3x3(…)- realizuje filtr rozmywający z maską czterospójną, działa na obrazach klasy CStdImage o głębi 8-bitowej

BlurEightConnected3x3(…) – realizuje filtr rozmywający z maską ośmiospójną, działa na obrazach klasy CStdImage o głębi 8-bitowej oraz CIntImage

3.3.8. Morfologia

Klasą, w której są zawarte operacje morfologiczne [20] [14] jest klasa CMorphology. Zawiera ona dwie podstawowe operacje morfologiczne erozje i dylację oraz trzy wersje szkieletyzacji.

Dla każdej z operacji erozji i dylacji są dostępne po dwie funkcje, wolniejsza pracująca na szerszym zakresie obrazów oraz szybsza pracująca na węższym zakresie obrazów. Dilate(...) wykonuje dylację na obrazach klasy CStdImage o głębi 1-bitowej lub 8-bitowej z elementem strukturalnym w postaci obrazu klasy CStdImage, gdzie aktywnym pozycjami są wartości pikseli różne od zera. Erode(…) wykonuje erozję na obrazach klasy CStdImage o głębi 1-bitowej lub 8-bitowej z elementem strukturalnym w postaci obrazu klasy CStdImage, gdzie aktywnym pozycjami są wartości pikseli różne od zera. Funkcjami, które odznaczają się szybszym działaniem od poprzednich są DilateFast(...) oraz ErodeFast(…) które odpowiednio wykonują dylację oraz erozję na obrazie klasy CStdImage o głębi 1-bitowej z elementem strukturalnym wyłącznie rozmiaru 3x3 o wartości elementów z zakresu od 0 do 7.

Do pocieniania i szkieletyzacji dostępne są trzy funkcje:

Thinning(...)- redukuje wszystkie czarne komponenty (składniki) w obrazie binarnym do gałęzi o szerokości jednego piksela zachowując następujące ograniczenia: nie usuwa punktów na końcu, nie przerywa połączonych regionów (przylądków) oraz nie powoduje nadmiernej erozji regionów.

Skeletonizing(...) - przekształca wszystkie czarne komponenty obrazu binarnego do szkieletu definiowanego przez średnią transformacje osi (medial axis transform-MAT). Funkcja ta nie zawsze działa poprawnie, w wielu przypadkach zawiera dużo małych gałązek, które muszą być później usunięte.

SkeletonZhou(...) - szkielet otrzymany po zastosowaniu tej funkcji, daje rezultaty pomiędzy wynikami działania funkcji Thinning(...) i Skeletonizing(...), jednakże jest bliższy algorytmowi MAT niż funkcji Thinning(...). Wykazuje się on większą odpornością na szum na krawędziach. t.j. zawiera mniej małych zbędnych gałęzi pochodzących od szumu na krawędziach [22].

3.3.9. Wykrywanie krawędzi

Do wykrywania krawędzi w obrazie służy MaxGradientFast(…) obliczająca maksymalną różnicę dwóch sąsiednich pikseli we wszystkich czterech kierunkach, działa na obrazach klasy CStdImage o głębi 8-bitowej oraz CIntImage (Rysunek 46).

Rysunek 46. Oryginalny monochromatyczny (a), obraz po użyciu funkcji MaxGradientFast(…) do wykrywania krawędzi.

3.3.10. Histogram

W klasie CLocalOperation zawarte są funkcje związane z histogramem. DrawHistogram(...) oblicza histogram obrazu źródłowego i rysuje go w obrazie docelowym. Natomiast EqualizeHistogram(...) dokonuje wyrównywania histogramu obrazu źródłowego i zapisuje obraz po operacji wyrównywania w obrazie docelowym.

3.3.11. Transformacje

Kilka prostych transformacji obrazu zostało zawartych bezpośrednio w klasach obrazu. Natomiast bardziej złożone metody zostały umieszczone w CCoordinateTransform.

Transformacje zawarte bezpośrednio w klasach obrazu:

FlipHorisontal() – odbicie obrazu względem środkowej poziomej osi x

FlipVertical() – odbicie obrazu względem pionowej, środkowej osi y

Invert() – inwersja obrazu, działa na wszystkich głębiach pikseli

Rotate90(int steps) – obrót obrazu o 90 stopni, jako parametr podawana jest liczba obrotów

W klasie CCoordinateTransform zebrane są algorytmy do transformacji pikseli z jednego układu współrzędnych do drugiego, nowego układu współrzędnych:
ScaleAuto(...)- dokonuje skalowania obrazu używając jako parametrów współczynników skalowania oraz automatycznie reguluje rozmiary nowego obrazu.
ScaleFixed(...)- dokonuje skalowania obrazu oraz zmienia rozmiary obrazu do nowych rozmiarów podawanych jako parametr
ScaleAndRotateAuto(...) – dokonuje skalowania, rotacji obrazu oraz automatycznie zmienia rozmiary obrazu zapobiegając przez to utracie niektórych pikseli
ScaleAndRotateFixed(...) – dokonuje skalowania, rotacji obrazu oraz zmienia rozmiary obrazu do nowych rozmiarów podawanych jako parametr

3.3.12. Transformata Hough'a

Podstawową klasą, z której wywodzą się wszystkie inne klasy transformaty Hough'a jest CHoughBase. Klasa ta dostarcza typy i metody wykorzystywane w innych technikach transformacji. Do wykrywania linii zostały zaimplementowane:

- probabilistyczna, losowa transformata Hough'a w klasie CHoughProbRHTLine
- losowa transformata Hogh'a w klasie CHoughRHTLine
- standardowa transformata Hough'a w klasie HoughSHTLine
- naturalnej wielkości dwuwymiarowa przestrzeń Houhg'a w klasie CHoughSpace2D

Natomiast w klasie CHoughRHTCircle zaimplementowana została standardowa losowa transformata Hough'a do wykrywania okręgów.

3.3.13. Progowanie i segmentacja

Funkcja CopyConvertThreshold(...) dokonuje progowania obrazu zadaną wartością progu oraz dokonuje konwersji z obrazu 8-bitowego do obrazu 1-bitowego.

W bibliotece tej dostępne są dwa algorytmy do segmentacji: algorytm blob oraz śledzenie konturu. Metody te można znaleźć w klasie CSegmentate. Podczas segmentacji piksele łączone są w grupy i przechowywane w CPixelGroup.

- Algorytm blob, skrót pochodzi od Binary Large OBjects.

DeriveBlobs(CStdImage & Source, CPixelGroups & PixelGroups, COLORTYPE Color, UINT8 Threshold, CONNECTIVITY Connected, unsigned int MinSize = 1)

W wyniku działania tego algorytmu segmentacji ulegają piksele obrazu nie należące do tła. Piksele te gromadzone są w grupy. Jednym z parametrów jest obraz źródłowy (*Source*) o głębi 1-bitowej lub 8-bitowej, z którego wywodzą się bloby. Każdy blob przechowywany jest jako CPixelGroup w CPixelGroups. Parametr *Color* definiuje, czy białe czy czarne piksele będą rozpatrywana w pierwszym planie. Parametr *Threshold* jest używany do progowania, gdy obraz źródłowy jest w odcieniach szarości. *Connected* definiuje siatkę spójności bloba, która może być cztero-(*FOURCONNECTED*) lub ośmio-spójna (*EIGHTCONNECTED*). Ostatni parametrem jest *MinSize* określający minimalną liczbę pikseli, z których może się składać blob.

- Dostępne są dwie metody pracujące na zasadzie łączenia pikseli należących do konturu:

FindAndFollowLowContour(CStdImage& Source, CPoint2D<int> Start, DIRECTION SearchDirection, CONNECTIVITY Connected, CPixelGroup& PixelGroup, ROTATION& RotationDirection, COLORTYPE& SurroundedColor)

FindAndFollowHighContour(CStdImage& Source, CPoint2D<int> Start, DIRECTION SearchDirection, CONNECTIVITY Connected, CPixelGroup& PixelGroup, ROTATION& RotationDirection, COLORTYPE& SurroundedColor)

W obydwu funkcjach przeszukiwanie rozpoczyna się od punktu startowego – parametr *Start*, następnym parametrem jest kierunek przeszukiwania (ang. *SearchDirection*), może być jednym z następujących *NORTH* (północ), *EAST* (wschód), *SOUTH* (południe) lub *WEST* (zachód). Dostępne są dwie wartości parametru *Connected EIGHTCONNECTED* lub *FOURCONNECTED*. Informacja o otaczającym kolorze znalezionej krawędzi zwraca parametr *SurroundedColor,* natomiast parametr *RotationDirection* zwraca kierunek, w którym został znaleziony kontur. Funkcja kontynuuje przeszukiwanie do moment, w którym znajdzie pierwszy kontur i umieści go w obiekcie CPixelGroup.

3.3.14. Momenty

Do tej pory w IPL98 z algorytmów służących do wyznaczania cech obrazów zostały zaimplementowane tylko metody momentowe, które można znaleźć w klasie CFeature. Wszystkie funkcje działają na obiektach CPixelGroup i służą do obliczania prostych momentów binarnych, monochromatycznych, momentów centralnych, momentów centralnych znormalizowanych oraz niezmienników momentowych.

DeriveMomentsGrayTone(...) – służy do obliczania momentów zwykłych monochromatycznych, dla każdego piksela obliczane są wagi, które stanowią różnice pomiędzy oryginalna wartością danego piksela a wartością tła (ang.: Background) podawanego jako parametr funkcji. Rezultaty umieszczane są w atrybucie m_Moments. Struktura atrybutu zawiera człon m_{pg} gdzie suma *p* i *g* jest rzędem momentu. Funkcja ta działa bardzo szybko.

DeriveMomentsBinary(...) – służy do obliczania momentów binarnych, działa podobnie jak DeriveMomentsGrayTone(...) z tą różnicą iż wartość każdego piksela wynosi 1.

DeriveCentralMoments() - wyznacza centralne momenty dwuwymiarowe rzędu (p+g) na podstawie momentów wcześniej obliczonych za pomocą DeriveMomentsGrayTone(...) lub DeriveMomentsBinary(...).

DeriveCentralNormMoments() - wyznacza centralne momenty znormalizowane dwuwymiarowe rzędu (p+g) na podstawie momentów wcześniej obliczonych za pomocą DeriveMomentsGrayTone(...) lub DeriveMomentsBinary(...).

DeriveHuFromCentralMom() - służy do wyznaczania wszystkich 13 niezmienników momentowych obliczonych z momentów centralnych obliczonych przez DeriveCentralMoments()

DeriveHuFromCentralNormMom() - służy do wyznaczania wszystkich 13 niezmienników momentowych obliczonych z momentów centralnych znormalizowanych obliczonych przez DeriveCentralNormMoments()

3.4. LTI-LIB

3.4.1. Przedstawienie

LTI-LIB [31] jest łatwą w użyciu obiektowo zorientowaną biblioteką z algorytmami wykorzystywanymi często w przetwarzaniu obrazów oraz widzeniu komputerowym, została stworzona na Politechnice Aachen w 1999 roku jako część wielu projektów i badań poświęconych widzeniu komputerowemu. Głównym jej celem jest dostarczenie narzędzi niezależnych od systemu operacyjnego, dlatego tez zbudowano ją w sposób (jak tylko to było możliwe) zbliżony do standardu ANSI C++.

3.4.2. Wersje oraz dokumentacja

LTI-LIB zamieszczona jest na stronie SourceForge [31]. Do 25 listopada 2005 roku biblioteka ta była regularnie, w krótkim odstępnie czasu uaktualniana, jednakże od ostatniej aktualizacji minął już ponad rok. Nie świadczy to, że prace na biblioteka zostały przerwane, tylko wynika z przeciągających się prac nad publikacja wersji przeznaczonej pod Visual Studio C++ 2005. Na stronie biblioteki można znaleźć przejrzysty manul oraz "Przewodnik programisty biblioteki LTI-LIB" z roku 2003.

3.4.3. Systemy operacyjne, kompilatory

Z założenia biblioteka LTI-LIB ma być niezależna systemowo. Sukcesami zakończyły się testy na platformach Windows oraz Linux, natomiast na innych platformach biblioteki nie testowano. Przeznaczona jest do pracy pod kompilatory:

- MS Visual C++ 6.0 z Service Packiem 5.0
- MS Visual C++ .NET 2003 (wersja .NET 2002 nie została sprawdzona)
- GCC 2.95.3 (lub nowszą wersją)

Niestety biblioteka ta nie daje się skompilować pod MS Visual Studio 2005.NET, jednakże aktualnie trwają prace nad wersją pod ten kompilator, która powinna się niedługo ukazać. Do obu platform dodatkowo potrzebny jest PERL, a do wyświetlania należy również zainstalować bibliotekę GTK.

3.4.4. Instalacja

Aby zainstalować bibliotekę na systemie Windows należy wcześniej zainstalować PERL'a i bibliotekę GTK+ lub podczas ściągania ze strony najnowszej wersji biblioteki wybrać wersję instalacyjną zawierającą PERL i GTK+. Następnie należy ściągniętą wersję biblioteki z Internetu rozpakować i otworzyć plik instalacyjny biblioteki. Podczas instalacji wybrać docelową lokalizację, gdzie ma zostać zainstalowana biblioteka (sugerowane jest, aby była nią C:\ltilib).
Aby rozpocząć pracę z biblioteką trzeba stworzyć projekt używający biblioteki - może on zostać utworzony ręcznie lub można do jego stworzenia użyć skryptu PERL'a. Zawsze możliwa jest praca z biblioteką na dwa sposoby: pierwszy używać jej jako biblioteki kodów źródłowych (przeznaczona do dokonywania zmian w istniejących kodach źródłowych biblioteki) oraz używać jej jako biblioteki statycznej (przeznaczona do pracy z zaimplementowanymi funkcjami).

Automatyczne tworzenie projektu przez PERL'a :

- w ltilib\win znajdują się foldery console_src, console_lib i tester. Każdy zawiera skrypt PERL'a generujący projekt konsoli z domyślnymi ustawieniami. Jeżeli pliki projektu juz istnieją, skrypty te dokonają jedynie ich uaktualnienia. Należy przejść do katalogu ltilib\win\buildLib (gdy tworzony jest projekt pod MS Visual Studio 6.0) lub do katalogu ltilib\win\buildLib_net (gdy tworzony jest projekt pod MS Visual Studio 2003.NET) i uruchomić plik wsadowy odpowiednio buildLib.bat lub buildLib_net.bat. Po tak wykonanych czynnościach biblioteka gotowa jest do pracy.

Ręczne tworzenie projektu biblioteki z kodami źródłowymi (dodawane są wszystkie kody źródłowe LTI-LIB bezpośrednio w projekcie), należy:

- dodać wszystkie źródła LTI-LIB do widoku pliku własnej przestrzeni roboczej projektu
- wskazać pod ustawieniami projektu C/C++->Preprocessor dla wszystkich konfiguracji ścieżki dostępu do katalogów zawierających wszystkie pliki nagłówkowe lti biblioteki
- aktywować C/C++->Language->Enable Run Time Type Info

Ręczne tworzenie projektu biblioteki statycznej (kompilacja LTI-LIB do biblioteki statycznej), należy:

- wywołać bulidLib.bat lub buildLib_net.bat, to spowoduje zbudowanie biblioteki w ltilib\lib oraz zgromadzi i umiejscowi wszystkie pliki nagłówkowe w ltilib\lib\headerFiles
- pod ustawieniami projektu Link->General dodać ltilib.lib dla konfiguracji debug oraz rltilib.lib dla konfiguracji release
- pod ustawieniami projektu C/C++Preprocessor wskazać ścieżkę do katalogu ltilib\lib\headerFiles dla wszystkich konfiguracji
- aktywować C/C++->Language->Enable Run Time Type Info

3.4.5. Struktura

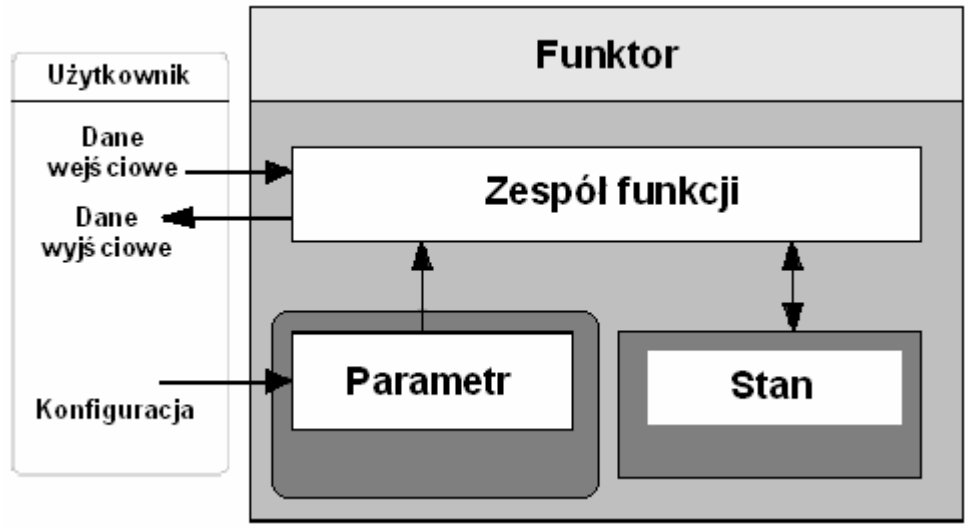
LTI-LIB jest łatwa w użyciu ze względu na spójny interfejs programistyczny dla wszystkich klas. Osiągnięcie to częściowo zawdzięcza się użyciu w bibliotece PERL'a - skrypt ltiGenerator. Skrypt ten w oparciu na niewielu podstawowych danych dostarczonych przez programistę (takich jak nazwa klasy oraz klasa macierzysta) buduje pliki szablonu zawierające wszystkie standardowe definicje. Następnie należy tylko zaimplementować zespół funkcji (ang. functionality).

3.4.5.1. Funktory, parametry, stany

Większość algorytmów wymaga parametrów tj. zdefiniowanych przez użytkownika wartości, które modyfikują działanie algorytmu. Np. nazwa pliku jest parametrem klasy do ładowania obrazu lub rozmiar jądra filtru jest parametrem algorytmu konwolucji. W bibliotece LTI-LIB wszystkie algorytmy są zawarte w tak zwanych funktor-klasach. Zawierają one zawsze klasę zwaną **parameters**, która może być jawnie zadeklarowana lub może dziedziczyć z klasy macierzystej. Oznacza to, że kiedy używana jest biblioteka LTI-LIB, nie trzeba wywoływać jakiś funkcji lub metod klas z dużą liczbą zawiłych argumentów. Domyślny obiekt parametrów zwykle jest zgromadzony w klasie funktor a od użytkownika wymagane jest jedynie zdefiniowanie danych wejściowych oraz wyjściowych, gdzie rezultaty zostaną zapisane. Oczywiście

użytkownik może zmieniać te parametry by dostosować funkcjonalnie funktory do własnych potrzeb.

Parametry funktora muszą odróżniać się od jego stanu, który składa się ze wszystkich atrybutów klas, które są obliczone podczas wykonywania algorytmu, ale nie są bezpośrednio dostarczone lub wymagane prze użytkownika. Np. dla obrazów historii ruchu (lti::temporalTemplates) ostatni prezentowany obraz musi być trzymany by wykonać następną iterację. Koncepcja ta pokazana jest na Rysunek 47.



Rysunek 47. Architektura funktora. Użytkownik może zmieniać działanie funktora poprzez parametry. Funktor może również posiadać stan, który ostatecznie może być zapisany podobnie jak parametry.

Istnieje kilka powodów, dla których w bibliotece tej wykonano niezależne klasy parametrów. Mianowicie można utworzyć kilka przykładów z różnym kompletem wartości i zmieniać funkcjonalności własnego funktora poprzez funkcję setParameters(). Można również załadować lub zapisać własne obiekty parametrów w pliku lub też udostępnić je w graficznym interfejsie użytkownika, gdzie będą mogły być zmieniane.

Parametry zawierają wartości bezpośrednio sprecyzowane przez użytkownika i nie powinny one być modyfikowane przez funktor. Jeżeli programista uważa, że w jego algorytmie musi zostać zmieniony parametr oznacza to, że parametr ten powinien zostać skopiowany gdzieś gdzie rozpoczyna się algorytm jako zmienna lokalna lub zmienna statyczna (atrybut klasy funktora).

Dostęp do funkcjonalności (zespołu funkcji) funktora uzyskuje się zawsze przez jego metody apply(). Oczekują one danych wejściowych (zwykle stałe odniesienie do obiektów jak macierze lub obrazy) oraz odwołania do wyjściowych obiektów (odniesienie do zbiorników, gdzie rezultaty zostaną zapisane). Funktory dostarczają

również tak zwanych skrótów metod, które upraszczają użycie określonych zespołów funkcji. Np. do załadowania pliku obrazu program lądujący dostarcza skrót load, który oczekuje nazwy pliku oraz obrazu gdzie zostaną zapisane rezultaty. Niewymagane jest w tym przypadku tworzenie obiektu parametrów, ustawianie w nich nazwy pliku, zapisanie tych parametrów a na koniec wywołanie metody apply():

```
lti::image img;             //obraz
lti::loadBMP loader;        // funktor do załadowania  obrazów w
                            formacie BMP
lti::loadBMP::parameters loaderParam;     //parametry dla loader
loaderParam.filename = "lena.bmp";         //plik do załadowania
loader.setParameters(loaderParam);         //ustawienie
                                           parametrów do
                                           załadownia obrazu
loader.apply(img);                         //załadowanie obrazu
```

Łatwiej i bardziej komfortowo jest użyć następującego skrótu:

```
lti::image img;             //obraz
lti::loadBMP loader;        // funktor do załadowania  obrazów w
                            formacie BMP
loader.load("lena.bmp",img);          //załadowanie obrazu
```

Należy również zapamiętać, że parametry nigdy nie powinny być zmieniane w metodzie apply().

Oprócz parametrów, funktor może mieć również stan, gdzie są gromadzone informacje nieistotne dla użytkownika, ale konieczne dla późniejszych obliczeń. Obiekt lti::principalComponents jest przykładem funktora z rozdzielonym stanem i parametrami. Można tu znaleźć parametr autoDim, który wskazuje, że inny parametr resultDim powinien być wykryty automatycznie. W metodzie apply() ta ostatnia wartość nie jest zmieniona. Obliczenie macierzy transformacji jest częścią stanu funktora, który może być później użyty do transformacji innych wektorów. Macierzy tej użytkownik nie może podać bezpośrednio, ale może ją załadować lub zapisać z inną częścią stanu funktora (zrobione jest to wtedy, kiedy załadowywany lub zapisywany jest

cały funktor). Wszystkie funktory z stanem związanym dla późniejszych obliczeń mogą zostać zapisane lub załadowane tj. przeładowują metody read i write.

3.4.5.2. Klasy wizualizacji

Nie wszystko w bibliotece do przetwarzania obrazu i widzenia komputerowego może być rozpatrywane jako funktor. Przykładami na to są tak zwane obiekty do rysowania i wizualizacji. Rysowanie obiektów nie wykonuje jednego algorytmu. Dostarczone są różne narzędzia do rysowania prostych geometrycznych konstrukcji na obrazach lub innych wyjściowych nośnikach. Aby użyć obiektu rysującego trzeba dostarczyć mu tło rysunku tj. należy sprecyzować obraz, gdzie ma być narysowany. Dokonane jest to metodą use(). Następnie można wybrać kolor, który chce się użyć metodą setColor(). Wtedy wszystkie linie, punkty, okręgi będą namalowane przy użyciu tego koloru.
Obiekty przeglądarki (lti::viewer) dostarczają prostych sposobów do wyświetlania danych nie modyfikując ich.

3.4.5.3. Klasyfikatory

Innymi ważnymi klasami obiektu, które nie pasują do modelu funktora są klasyfikatory (ang. classifiers). Dostarczają one metod do uczenia z danych oraz do używania nauczonych informacji do analizy nowych danych. Dla kierowanych i niekierowanych klasyfikatorów istnieją odmienne interfejsy. Obydwa typy mogą zostać sklasyfikowane do przypadku klasyfikatorów, które uczą pojedynczy wektor (podobnie jak sieci neuronowe) oraz sekwencji klasyfikatorów, które także rozpatrują aspekty czasu (podobnie jak ukryte modele Markowa).
W LTI-LIB wszystkie klasyfikatory dostarczają wyników przy użyciu tych samych struktur danych lti::classifier::outputVector, by przetwarzanie ich wyników nie zależało od użycia określonego klasyfikatora.

3.4.6. Struktury danych

Wszystkie obiekty w LTI-LIB występują w przestrzeni nazw lti. Podstawową klasą w LTI-LIB jest object. Klasa lti::mathObject jest macierzystą klasą dla wielu globalnych klas jak wektory oraz macierze. Większość algorytmów operuje na tym typie danych. Obiekty lti::functor implementują algorytmy, obiekty lti::exception dziedziczą od std::exception, które są wrzucane w przypadku wystapienia błędu krytycznego.

LTI-LIB dostarcza struktur danych niezbędnych do gromadzenia różnych rodzajów danych wymaganych w widzeniu komputerowym i aplikacjach matematycznych. Występują różne poziomy struktur danych, które można podzielić na:

- podstawowe typy
- prymitywy geometryczne
- prymitywy piksela
- typy ogólne
    - macierze i wektory
    - obrazy i kanały
    - jadra filtru
    - kontury, obszary, wielokąty
    - drzewa
    - sekwencje
    - histogram
    - piramidy

3.4.6.1. Podstawowe typy

Ponieważ standard C++ nie definiuje dokładnych rozmiarów wewnętrznych typów, wyszczególniono podstawowe typy, które normalizują rozmiary niezależnie od systemu i procesora, gdzie biblioteka będzie kompilowana (Tabela 4).

Tabela 4. Typy danych dla poszczególnych klas.

Nazwa typu	Opis
lti::ubyte	**8-bitowa liczba całkowita bez znaku**
lti::byte	**8-bitowa liczba całkowita ze znakiem**
lti::uint16	**16-bitowa liczba całkowita bez znaku**
lti::int16	**16-bitowa liczba całkowita ze znakiem**
lti::uint32	**32-bitowa liczba całkowita bez znaku**
lti::int32	**32-bitowa liczba całkowita ze znakiem**
lti::sreal	**liczba zmiennoprzecinkowa o pojedynczej precyzji**
lti::dreal	**liczba zmiennoprzecinkowa o podwójnej precyzji**

3.4.6.2. Prymitywy piksela

Możliwe jest manipulowanie kanałami obrazów wyciągniętymi z różnych przestrzeni kolorów. Jednakże gromadzenie i wyświetlanie danych obrazów kolorowych zawsze dokonywane jest w przestrzeni RGB, dlatego też dostarczono dwa typy do pracy z tą przestrzenią:

lti::rgbPixel jest podstawowym typem 4-bajtowym przestrzeni RGB, służy do reprezentacji kolorowego piksela w przestrzeni RGB, składa się z czterech kanałów: czerwonego, zielonego, niebieskiego i czwartego tzw. "alfa" będącego dopełnieniem do 4 bajtów.

lti::trgbPixel jest typem szablonu złożonym trzech kanałów (czerwony, zielony, niebieski) i służy do reprezentacji typów innych niż ubyte.

Wiele kolorów w LTI-LIB jest zdefiniowane jako stałe przypadki lti::rgbPixel, są to lti::Black (kolor czarny) , lti::White (kolor biały), lti::Red (kolor czerwony), lti::Green (kolor zielony), lti::Blue (kolor niebieski), lti::Cyan (kolor niebiesko-zielony), lti::Magenta (kolor fuksji) oraz lti::Yellow (kolor żółty).

3.4.6.3. Macierze i wektory

Algorytmy LTI-LIB operują na dwóch typach pojemników: 1-wymiarowych (lti::vector) lub 2-wymiarowych (lti::matrix). Obydwa z nich są zaimplementowane jako szablony C++.

lti::vector jest klasą szablonu do gromadzenia i manipulowania n-wymiarowymi wektorami, więc można w niej wyszczególnić, jaki typ powinny posiadać elementy. Dostarcza dużo prostych metod dla typowych operacji wektorowych, iteratorów wpływających na szybszy dostęp. Jednakże klasa ta nie jest została zaimplementowana by zastępować klasę std::vector.
lti::matrix jest klasą szablonu do gromadzenia i manipulowania macierzami o dowolnym rozmiarze, podobna jest do wektorów. Jako metody tej klasy dostarczonych jest dużo prostych operacji macierzowych. Dostępne są również iteratory zwiększające szybkość operacji.

3.4.6.4. Obraz

Obrazy w LTI-LIB reprezentowane są w klasach lti::image, lti::channel8 oraz lti::channel. Klasa lti::image jest klasą dziedziczącą od lti::matrix<rgbPixel> i użyta jest do reprezentacji obrazów kolorowych, tzn. jest jedyną klasą dla obrazów RGB oraz zdefiniowana jest jako macierz pikseli kolorowych (lti::rgbPixel). Wiersze macierzy reprezentują poziome linie w obrazie, natomiast kolumny pionowe. Wiersz z indeksem zero będzie wierszem na szczycie, kolumna z indeksem zero będzie kolumną z lewej strony. Współrzędna „x" oznacza pozycje w osi poziomej, natomiast współrzędna „y" oznacza pozycje w osi pionowej. Innymi słowy współrzędna „y" oznacza wiersze, a współrzędna „x" kolumny macierzy. Informacje te istotne są przy uzyskiwaniu dostępu do pikseli. Obrazy monochromatyczne reprezentowane są w klasach lti::channel i lti::channel8, które zdefiniowane są jako macierze odpowiednio o elementach typu float oraz integer. Obie te klasy różnią się miedzy sobą typem wartości oraz rozpatrywanym zasięgiem wartości ich elementów. lti::channel dopuszcza wartości float między 0.0f (kolor czarny) a 1.0f (kolor biały), natomiast lti::channel8 dopuszcza wartości integer miedzy 0 (kolor czarny) a 255 (kolor biały). Możliwe jest rzutowanie między różnymi kanałami (monochromatyczne obrazy) przy użyciu metod castFrom().

3.4.6.5. Kontury, obszary, wielokąty

Struktury danych do reprezentacji wektorów, macierzy, obrazów nie są wystarczające by sprostać wszystkim wymaganiom algorytmów przetwarzania obrazu.

Potrzebne są również inne typy reprezentujące obszary obrazu, kontury, wielokąty. Klasą z której dziedziczą inne opisane klasy w tym punkcie jest lti::tpointList, klasa ta pozwala gromadzić listy tpoints<T> (lista punktów ze współrzędnymi typu integer). Możliwy jest dostęp do elementów tpointList przy pomocy iteratorów. Następne cztery klasy dziedziczą od lti::tpointList. Klasa lti::borderPoints jest listą granicznych punktów reprezentujących kontur w obrazie, wszystkie algorytmy zapewniają, że dwa przyległe punkty w liście są sąsiadami w obrazie. Klasa lti::ioPoints reprezentuje kontury w postaci punktów wejściowych i wyjściowych dla każdej linii. Klasa lti::areaPoints reprezentuje kontur w postaci listy wszystkich punktów należących do konturu. Klasa lti::polygonPoints reprezentuje wierzchołki wielokąta, zawiera metody do aproksymacji dowolnego konturu (reprezentowanego przez obiekt lti::borderPoints) przez wielokąty, Uzyskać również można wypukłą otoczkę (ang. convex hull) listy punktów używając odpowiednich metod tej klasy.

3.4.6.6. Drzewa

Klasa pojemnika dla uporządkowanych n-drzew zaimplementowana jest w lti::tree jako szablon. Typ szablonu jest typem elementów danych zawartych w każdym węźle drzewa, niezależnie czy jest to liść czy też nie. Jedynym stawianym wymaganiem odnośnie typu T jest to, iż nie powinien on zależeć od dynamicznej alokacji wewnętrznych danych. Drzewo tworzone jest zawszę jako puste, można jednak sprecyzować stopień tj. maksymalną liczbę potomków oczekiwaną w punkcie węzła.

3.4.6.7. Sekwencje

lti::sequence użyta jest do reprezentacji sekwencji innych obiektów, takich jak np. obrazy. Może być użyta jako rodzaj wektora innych obiektów. Poszczególne elementy sekwencji są indeksowane miedzy 0 a n-1.

3.4.6.8. Histogram

Histogram jest zbiorem klas użytych do wygenerowania n-wymiarowych histogramów. Podstawową klasą jest lti::thistogram, jest to klasa macierzysta dla różnych reprezentacji wielowymiarowych histogramów. Aby uzyskać większą efektowność dla histogramów jedno i dwu wymiarowych zalecane jest używanie klas lti::histogram1D oraz lti::histogram2D

3.4.7. Praca z obrazami

3.4.7.1. Dostęp do pikseli obrazu

Istnieje kilka sposobów dostępu do pikseli. Aby to zobrazować posłużono się przykładami.

Dla obrazu monochromatycznego można uzyskać dostęp do pikseli w następujący sposób:

- poprzez podanie współrzędnych

```
for(int y = 0; y < img.size().y; y++)
    for(int x = 0; x < img.size().x; x++)
      img[y][x] *= 0.75;
```

- poprzez podanie współrzędnych w sposób

```
for(int y = 0; y < img.size().y; y++)
    for(int x = 0; x < img.size().x; x++)
      img.at(y,x) *= 0.75;
```

- poprzez iteratory

```
lti::image::iterator it;
  for(it = img.begin(); it != img.end(); it++)
    *it *= 0.75;
```

Dla obrazu kolorowego dostęp do pikseli otrzymuje się w następujący sposób:

- ```
  for(int y = 0; y < img.size().y; y++)
    for(int x = 0; x < img.size().x; x++)
      img.at(y,x) = lti::rgbPixel(255,0,0);
  ```

- ```
  lti::rgbPixel cp(128,128,255);
  for(int y = 0; y < img.size().y/2; y++)
    for(int x = 0; x < img.size().x/2; x++)
      img.at(y,x) = cp;
  ```

3.4.7.2. Odczyt i zapis obrazu do pliku

Do załadowania obrazu pliku w LTI-LIB służy funktor lti::loadImage, natomiast funktor lti::saveImage służy do zapisania obrazu w pliku. Obrazy mogą być ładowane i zapisywane w następujących formatach plików:

- BMP
- PNG
- JPEG

3.4.8. Konwersja przestrzeni kolorów

W LTI-LIB algorytmy pracują jedynie na obrazach w przestrzeni kolorów RGB, obrazy te jednakże można podzielić na kanały różnych przestrzeni kolorów. Można również trzy kanały w określonej przestrzeni kolorów złączyć w obraz. Do wydzielenia obrazu kolorowego można użyć funktorów wywodzących się z klasy lti::splitImage, natomiast do złączenia kanałów funktorów wywodzących się z klasy lti::mergeImage. Dostępne są następujące przestrzenie kolorów podlegające opisanym przekształceniom: RGB, HSV, HSI, HLST, Torgl, XYZ, xyY, YIQ, OCPT.
Można również utworzyć własną liniową transformację przestrzeni kolorów z lti::linearMixer.

3.4.9. Filtracja

W LTI-LIB zaimplementowano szereg funktorów wykonujących filtrację obrazu.

lti::convolution jest klasyczną operacja filtrującą, działa na zasadzie konwolucji obrazu z jądrem w postaci wektora lub macierzy.

lti::squareFilter jest zoptymalizowaną wersją konwolucji z prostokątnym jądrem

lti::ogdFilter dokonuje filtracji obrazu z ukierunkowanymi pochodnymi Gaussa (ang. oriented gaussian derivatives OGD)

lti::correlation użyty jest do korelacji małych obszarów w większych obrazach, w klasycznej formie zbliżony jest on do konwolucji, jednakże w LTI-LIB funktor ten oferuje dodatkowe tryby działania

lti::downsampling funktor jest używany do próbkowania w dół obrazu, który wykonuje konwolucję jądra z obrazem

lti::upsampling funktor używany do próbkowania w górę obrazu

lti::medianFiltrer klasyczny filtr medianowy wykonujący filtrację na kanale obrazu

lti::kNearestNeighFilter oparty jest na statystycznym algorytmie k-najbliższych sąsiadów, gdzie każdy piksel jest wyznaczony jako najczęściej występujący piksel z sąsiedztwa.

3.4.10. Morfologia

W LTI_LIB zaimplementowano następujące klasyczne i niekonwencjonalne operatory morfologiczne:

Funktor lti::dilation ten implementuje dylację, natomiast funktor lti::erosion erozję. Przez parametr dokonuje się wyboru trybu operacji morfologicznej, czy ma być binarna (parametr binary) czy monochromatyczna (parametr gray scale), należy również podać element strukturalny reprezentowany jądro filtru liniowego.

lti::skeleton dokonuje szkieletyzacji kanału wejściowego, gdzie piksele o wartości 0 traktowane są jako tło, natomiast piksele różne od 0 jako obiekty.

lti::maximumFilter działa na zasadzie przypisywania każdemu pikselowi maksymalnej wartości z danego sąsiedztwa podanego jako obszar przez parametr kernelSize.

3.4.11. Wykrywanie krawędzi i rogów

Detektory krawędzi w LTI-LIB dziedziczą od macierzystej abstrakcyjnej lti::edgeDetector, znajdują krawędzie w obrazach monochromatycznych. Rezultatem działania detektora krawędzi jest zwykle kanał obrazu (lti::channel8) zawierający tylko dwie wartości wskazujące na krawędź oraz co do krawędzi nie należy.

Funktor lti::susanEdge oparty jest na algorytmie Susan, jest to szybka metoda do wykrywania binarnych krawędzi w obrazie lub kanale. Jako wynik otrzymuje się czarno-białą mapę, gdzie białe piksele (255) odznaczają krawędź, a czarne piksele (0) odznaczają obszar tła. Wrażliwość algorytmu kontrolowana jest przez parametr threshold, przyjmujący wartości od 0 do 255, gdzie przy małym progu dostarczana jest duża ilość krawędzi, natomiast przy dużym progu wykrywane są ostre krawędzie (Rysunek 48).

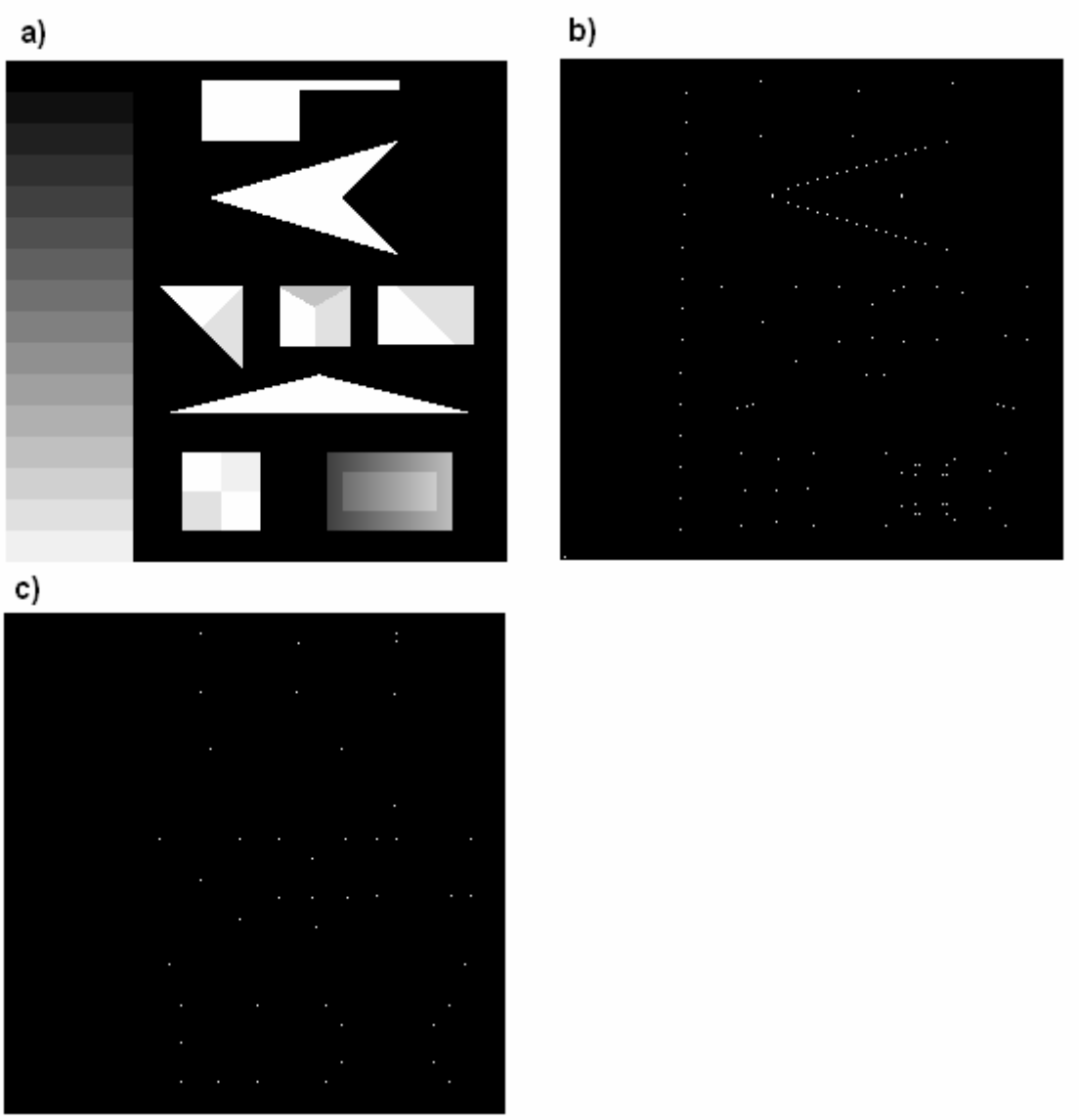
lti::cannyEdges jest standardowym algorytmem Canny'ego do wykrywania krawędzi przeznaczony do wykrywania "optymalnych" krawędzi. Parametrami tego funktora są wariancja i rozmiar jądra Gaussa użytego do filtracji obrazu, maksymalny próg, powyżej którego będą rozpatrywane znalezione krawędzie oraz minimalny próg.

lti::colorEdgesGS, funktor ten implementuje algorytm wykrywania krawędzi koloru, przeznaczony jest wyłącznie dla obrazów kolorowych, jeżeli zostanie użyty w obrazach monochromatycznych wyniki nie będą prawidłowe [13].

lti::classicEdgeDetector przy pomocy tego funktora zaimplementować można proste, standardowe wykrywanie krawędzi takim operatorami jak Sobel, Prewitt itp.

Rysunek 48. Oryginalny monochromatyczny (a), obraz po użyciu funktora lti::susanEdge z progiem 20 do wykrywania krawędzi (b).

Do wykrywania rogów w LTI-LIB zaimplementowano dwa algorytmy. Funktor lti::susanCorners znajduje rogi algorytmem Susan niskiego poziomu przetwarzania obrazu. Natomiast funktor lti::harrisCorners znajduje rogi algorytmem Harrisa do wykrywania rogów [15]. Rysunek 49 przedstawia przykładowe obrazy po zastosowani powyższych funktorów do wykrywania rogów.



Rysunek 49. Obraz oryginalny (a), obraz po operacji wykrywania rogów funktorem lti::harrisCorners (b), obraz po operacji wykrywania rogów funktorem lti::susanCorners (c).

3.4.12. Histogram

W LTI-LIB znajdują się następujące funktory do pracy z histogramem:

lti::histogramRGBL funktor ten oblicza cztery niezależne histogramy, jeden dla każdej składowej R, G, B oraz luminancji L zdefiniowanej jako L = (min(R,G,B) + max(R,G,B))/2

lti::histogramEqualization dokonuje klasycznego wyrównywania histogramu poprawiającego kontrast w obrazie

lti::chromaticityHistogram tworzy histogram chrominancji wektora cech.

3.4.13. Transformata Hough'a

W LTI-LIB zaimplementowano dwie wersje transformaty Hough'a: lti::orientedHLTransform oraz lti::gHoughTransform.

lti::gHoughTransform jest uogólnioną transformatą Hough'a do wykrywania dowolnych kształtów, wyszczególnionych wcześniej przez funkcję use(). Przekształca ona kanał obrazu w przestrzeń czterowymiarową parametrów. Wynikami działania tego algorytmu są szukane wierzchołki (piki) przestrzeni parametrów. Następnie znalezione obiekty mogą być narysowane z powrotem w obrazie przez funkcję draw().

lti::orientedHLTransform dokonuje szybkiej liniowej transformaty Hougha w celu wykrycia linii w obrazie źródłowym.

3.4.14. Techniki analizy intensywności

Aby wyeliminować wpływ oświetlenia można użyć funktorów wywodzących się z klasy lti::colorNormalizationBase:

lti::grayWorldNormalization jest prostą metodą, która oblicza dane drugiego rzędu (ang. second order statistics) każdego kanału koloru (RGB) by wyprodukować kanoniczne obrazy niezależne od koloru oświetlenia.

lti::comprehensiveColourNormalization jest metoda zaproponowana przez Finlayson, Schiele i Crowley [11] do eliminacji zależności od geometrii oświetlenia oraz koloru oświetlenia, normalizuje chromatywności i intensywność każdego kanału.
Rysunek 50 przedstawia wyniki eliminacji wpływu oświetlenia funktorami lti::grayWorldNormalization oraz lti::comprehensiveColourNormalization.

Rysunek 50. Obraz oryginalny (a), obraz po przetworzeniu funktorem lti::grayWorldNormalization (b), obraz po przetworzeniu funktorem lti::comprehensiveColourNormalization (c).

3.4.15. Progowanie, segmentacja, lokalizacja obszarów, kontury

Progowania obrazu monochromatycznego dokonuje funktor lti::thresholding. Najbardziej rozbudowanym zagadnieniem przetwarzania i rozpoznawania obrazów w LTI-LIB są algorytmy segmentacji. W bibliotece tej funktory segmentacji można podzielić na typy algorytmów, pierwszy "kompletne algorytmy", drugi "funkcyjne bloki", z którym można zbudować własne podejście do segmentacji.

Do algorytmów kompletnych zalicza się:

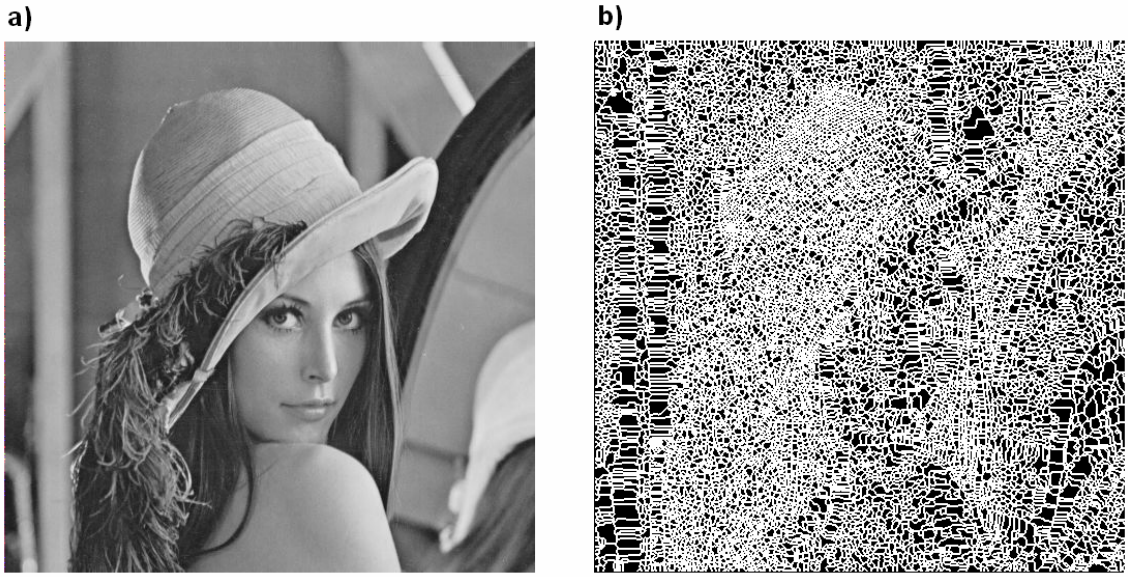
segmentację progową (ang. threshold segmentation). Użyta do oddzielenia tła od obiektu. Należy podać maskę z wartościami pikseli, która będzie rozpoznawana jako obiekt, reszta natomiast będzie traktowana jako tło. Dla tego typu segmentacji dostępne są dwa funktory. lti::thresholdSegmentation wykonuje segmentację progową na obrazie monochromatycznym. lti::optimalThresholding wykonuje segmentację progową pojedynczego kanału z wielokrotnie obliczaną wartością optymalną progu [25].

segmentację działu wodnego (ang. watershed segmentation), jest to algorytm bardzo rozlegle używany. Możliwe jest wybranie kilku typów implementacji modyfikując parametry funktora lti::watershedSegmentation. Segmentacja działu wodnego jest operatorem morfologicznym użytym do segmentacji obrazów monochromatycznych, bazującym na przeglądaniu obrazu monochromatycznego jako mapy topograficznej [42]

[10]. Rysunek 51 przedstawia obraz wynikowy po zastosowaniu funktora lti::watershedSegmentation.

segmentację obrazu algorytmem mean shift (funktor lti::meanShiftSegmentation), oparta jest o oryginalny algorytm D. Comaniciu [6] z niewielkimi zmianami zwiększającymi funkcjonalność oraz konfigurowalność.

funktor lti::regionGrowing, użyty jest on do segmentacji obrazu z regularnym tłem, działa na zasadzie rozrostu obszaru.



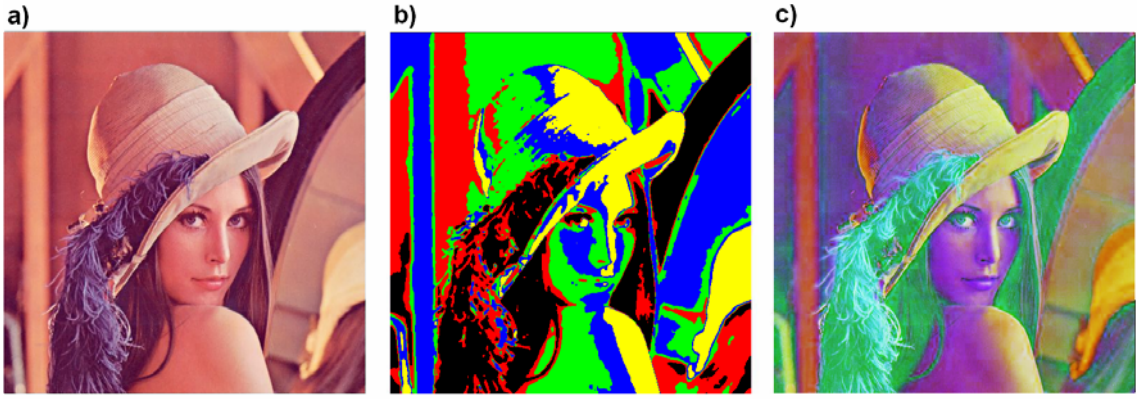
Rysunek 51. Obraz oryginalny (a), obraz po segmentacji funktorem lti::watershedSegmentation (b).

Do bloków funkcyjnych, które mogą być wykorzystane jako części do bardziej złożonych algorytmów segmentacji zalicza się:

segmentację wstępną, funktor lti::csPresegmentation próbuje znaleźć piksele, które należące do tła oraz te które do niego nie należą, działa na zasadzie kwantyzacji kolorów, usuwa jednorodne tło z innych obiektów, pracuje przy założeniu, że granice obrazu zawierają głownie piksele tła.

lti::kMeansSegmentation oparte jest na połączeniu kwantyzacji koloru i filtracji zachowującej krawędź, bardzo podobny jest do segmentacji wstępnej z tą różnicą iż nie jest rozpoznawane tło.

lti::whiteningSegmentation próbuje powiększyć obszary koloru w przestrzeni kolorów do odrębnych obiektów, które posiadają podobne kolory.
Rysunek 52 przedstawia wyniki segmentacji funktorami lti::kMeansSegmentation oraz lti::whiteningSegmentation.



Rysunek 52. Obraz oryginalny (a), obraz po segmentacji funktorem lti::kMeansSegmentation (b), obraz po segmentacji funktorem lti::whiteningSegmentation (c)

Innymi narzędziami związanymi z segmentacją są:

lti::regionMerge używa macierzy podobieństwa (lti::similarityMatrix) oraz wartości progu podanej jako parametr by zdecydować czy dwa obiekty w masce (zwróconej także przez lti::similarityMatrix lub lti::objectsFromMask) powinny być połączone czy też nie. Rezultatem działania tego funktora jest nowa maska z mniejszą liczbą obiektów niż oryginalna.

lti::boundingBox wydobywa część obrazu by rozważyć reprezentację konturu interesującego obiektu, przez co usuwa nieistotne tło.

lti::objectsFromMask, funktor ten zasługuje na specjalną uwagę, pracuje na kanałach 8-bitowych lub macierzach, działa na zasadzie wydobywania z obrazu połączonych komponentów i opisuje je w postaci topologicznej "obiektów" i "dziur".

Algorytmy do lokalizacji określonych obszarów obrazu:

lti::activeShapeModel jest podstawową klasą dla funktorów, pracujących nad statyczną analizą modeli rozkładu punktów (ang. Point Distribution Model - PDM) mających na

celu uzyskanie modelu aktywnego kształtu (ang. active shape models - ASM) służącego do dopasowania różnych kształtów.

lti::blobEM estymuje pozycję M pokrywającą bloby przez zastosowanie algorytmu EM oraz estymację parametrów modelu mieszanego Gaussa dopasowującego bloby [1].

lti::backgroundModel może zostać użyty do wykrycia poruszających się obiektów w sekwencji obrazów ze względnie stałym tłem [45].

lti::axLocalRegions funktor używa algorytmu opisanego w [18] do detekcji powiązanych małych regionów w obrazie, które mogą być użyte do wydzielania deskryptorów lokalnych

lti::locationSelector dzieli listę lokalizacji na kilka mniejszych list zależnych wartości danej maski decyzyjnej. Istnieją dwa tryby pracy dla tego funktora. Pierwszy usuwa wszystkie lokalizacje z danej listy, która jest oznaczona by mogła zostać usunięta z maski. Drugi tryb rozdziela listę lokalizacji w zespół mniejszych list, tak żeby wszystkie lokalizacje w każdej podliście miały tą samą etykietę. Tak może być wygenerowana podlista lokalizacji dla obiektu.

3.4.16. Momenty

Z metod momentowych w LTI-Lib występuje tylko jeden funktor lti::huMoments obliczający siedem niezmienników momentowych dla obszarów obrazu monochromatycznego.

3.4.17. Śledzenie obiektów, przepływ optyczny

Do śledzenia obiektów w LTI-LIB zaimplementowano następujące funktory:

lti::kalmanFilter oraz lti::kalmanTracker używają mechanizmu przewidywania filtru Kalmana do śledzenia części obrazu lub punktu [44]. Filtr Kalmana może zostać użyty do estymacji stanu dynamicznego z zaszumionych, mierzonych danych. Funktor

lti::kalmanTracker wykorzystuje filtr Kalmana do śledzenia punktu 2D przy użyciu systemowego stanu wektora, zawierającego współrzędne *x* i *y* jak również prędkości *x* i *y* oraz opcjonalnie przyśpieszenie *x* i *y* tego punktu. Ten stan wektorów odpowiada założeniu stałej prędkości oraz stałego przyśpieszenia. Ma on następującą strukturę *(x,vx,y,vy)* lub *(x,vx,y,vy,ax,ay)*. Zmierzone współrzędne *x* i *y* są podawane do metody apply(), która zwraca estymowane współrzędne *x* i *y* w następnym kroku.

lti::lkTracker jest piramidalną implementacją śledzenia punktu algorytmu Lucas-Kanade [2]

lti::camshiftTracker, w funktor tym został zaimplementowany algorytm CAMSHIFT do śledzenia obiektów [4]

lti::meanshiftTracker, w funktor tym został zaimplementowany algorytm meanshift do śledzenia obiektów [6]

Do śledzenia obiektów zaimplementowano również funktory liczące przepływ optyczny dla dwóch obrazów:

lti::opticalFlowHS oblicza przepływ optyczny przy użyciu algorytmu Horn-Schunck [16], w którym są wykonywane obliczenia z wykorzystaniem pochodnych wyższego rzędu

lti::opticalFlowLK oblicza przepływ optyczny przy użyciu algorytmu Lucas-Kanade [19] polegającym na łączeniu przylegających pikseli w grupy przy założeniu, że mają jednakową prędkość

lti::temporalTemplate nie jest dokładnie funktorem obliczającym przepływ optyczny, ale czymś podobnym do niego, wydobywa on obrazy historii ruchu z sekwencji kanałów [9]

3.5. CIMG

3.5.1. Przedstawienie

CIMG [28] jest mała biblioteką napisana w C++ przeznaczoną dla studentów oraz badaczy zajmujących się przetwarzaniem obrazów oraz widzeniem komputerowym. Została zapoczątkowana w roku 1999, jako praca dyplomowa doktorska na Uniwersytecie De Nice-Sophia Antipolis przez Davida Tschumperlé. Jest mała, prosta w użyciu i relatywnie wydajna. Definiuje grupę użytecznych klas oraz dużo prostych procedur do ładowania, zapisywania, wyświetlania oraz przetwarzania różnych typów obrazów, a wszystko to zawarte jest w 13000 liniach kodu.

3.5.2. Wersje oraz dokumentacja

Od 2003 roku jest ona udostępniana na Sourceforge [33]. Regularnie, w krótkim odstępie czasu ukazują się nowe, uaktualnione wersje, ostatnia (1.2.2) pochodzi z 4 lipca 2007 roku.

Na stronie biblioteki można znaleźć przejrzystą dokumentację tworzoną przez Doxygen, krótki tutorial stanowiący wprowadzenie do biblioteki. Również na stronie Sourceforge dostępne jest forum poświęcone pracy w bibliotece.

3.5.3. Systemy operacyjne, kompilatory

Biblioteka ta jest bardzo przenośna i może być używana na wielu systemach. Sukcesami zakończyły się testy na następujących platformach: Windows, Linux, Solaris, MacOS X, FreeBSD. Zaleca się, aby używać jednego z poniższych kompilatorów (jednakże na inny kompilatorach biblioteka również powinna kompilować się bez problemów):

Visual Studio C++.NET (od wersji 1.1.7 biblioteka przeznaczona jest pod .NET, jednakże nadal kompiluje się bezproblemowo w Visual C++ 6.0), GCC, Borland Bcc, Intel ICL, Dev-Cpp, DMC.

3.5.4. Struktura

CIMG jest biblioteką bardzo prostą do zrozumienia oraz bardzo przyjazną dla użytkownika, użyte są w niej jedynie standardowe biblioteki systemu, przez to uniknięto złożonych zależności oraz problemów ze zgodnością biblioteki. Jedynym wymaganiem jest użycie dosyć nowego kompilatora C++. Biblioteka ta składa się z pojedynczego pliku nagłówkowego CImg.h dostarczającego zbiór szablonów klas C++, które mogą zostać użyte przez programistę we własnym kodzie źródłowym do załadowania, zapisywania, przetwarzania oraz wyświetlania obrazów lub listy obrazów. A po zainstalowaniu ImageMagick oraz XMedcon w bibliotece dodatkowo dostępnych jest kilka zaawansowanych narzędzi.

Ponieważ w pliku nagłówkowym zawarte są wszystkie klasy oraz funkcje toteż:

- nie jest wymagana prekompilacja biblioteki, kompilacja funkcji CIMG jest wykonywana w tym samym momencie co kompilacja własnego kodu źródłowego
- nie muszą być obsługiwane złożone zależności: wystarczy zawrzeć plik CImg.h i można już pracować z zestawem narzędzi do przetwarzania obrazów w C++
- kompilacja dokonywana jest na bieżąco, tylko zespół funkcji CIMG użytych w programie użytkownika jest kompilowany oraz pojawia się w wersji wykonywalnej programu. To sprawia, że kod jest zwięzły bez jakichkolwiek nieużytecznych rzeczy.
- składniki klas oraz funkcje są wbudowane (inline) co przekłada się na lepszą wydajność podczas wykonywania programu.

W bibliotece występują dwie przestrzenie nazw:

- wszystkie klasy oraz funkcje są zdefiniowane w przestrzeni nazw cimg_library. Ta przestrzeń nazw obejmuje zespół funkcji oraz unika jakichkolwiek kolizji nazw, które mogłyby się zdarzyć z innymi dołączeniami (includes).
- przestrzeń nazw cimg_library::cimg definuje zbiór funkcji i zmiennych niskiego poziomu używanych przez bibliotekę. Zgromadzone funkcje w tej przestrzeni nazw mogą zostać bezpiecznie użyte w programie użytkownika jednakże należy pamiętać żeby nie używać cimg_library::cimg jako domyślnej przestrzeni nazw

dopóki zawarte są funkcje, których nazwy są zdefiniowane w standardowej bibliotece C/C++

Przestrzeń nazw cimg_library zawiera klasy:

- klasa cimg_library::CImg reprezentuje obraz w czterech wymiarach, zawierających piksele typu T (parametr szablonu), jest to właściwie główna klasa biblioteki.
- klasa cimg_library::CImgList reprezentuje listę obrazów klasy cimg_library::CImg<T>, może zostać użyta do np. do gromadzenia różnych klatek sekwencji obrazów.
- klasa cimg_library::CImgDisplay służy do wyświetlania obrazów lub listy obrazów w oknie graficznym. Kod w tej klasie jest wysoce systemowo zależny, zmienne środowiskowe są automatycznie ustalane przez bibliotekę.
- klasa cimg_library::CImgStats reprezentuje dane obrazu, używa się jej do obliczania minimum, maksimum, średniej wartości pikseli obrazów, jak również lokalizacje minimalnej i maksymalnej wartości piksela.
- klasa cimg_library::CImgException jest używana przez bibliotekę do sygnalizacji wystąpieniu błędów lub ostrzeżeń

Ustawianie zmiennych systemowych

CIMG jest biblioteką działającą na wielu platformach. Implikuje to istnienie pewnych zmiennych systemowych, które muszą być poprawnie zdefiniowane dla systemu, na którym biblioteka ta będzie uruchamiana. W większości przypadków biblioteka definiuje zmienne systemowe automatycznie, dzieje się tak dla najpopularniejszych systemów (Windows, Linux), w innych przypadkach ustawienia tych zmiennych systemowych (cimg_OS, cimg_display_type, cimg_debug, cimg_convert_path, cimg_temporary_path, cimg_plugin, cimglist_plugin, cimgdisplay_plugin, cimgstats_plugin) należy dokonać ręcznie przed zawarciem pliku nagłówkowego CImg.h we własnym kodzie źródłowym. Ustawienia wszystkich zmiennych mogą zostać sprawdzone przy użyciu funkcji cimg_library::cimg::info().

3.5.4.1. Reprezentacja obrazu

CImg jest klasą reprezentującą obraz w czterech wymiarach, każdy piksel należy do typu T. Jest to główna klasa biblioteki, deklaruje oraz buduje obraz, pozwala uzyskać dostęp do wartości pikseli oraz wykonywać różne przekształcenia na obrazach. Obraz CImg jest zdefiniowany jako przykład zbiornika CImg<T>, który zawiera regularną siatkę pikseli, gdzie każda wartość piksela jest typu <T>. Siatka ta może posiadać aż cztery wymiary: szerokość, wysokość, głębia oraz liczba kanałów. Zwykle trzy pierwsze wymiary są użyte by opisać współrzędne przestrzenne (x,y,z), podczas gdy liczba kanałów użyta jest jako wektor wymiarów opisujący np. kanały R,G,B przestrzeni kolorów. Aby użyć piątego wymiaru należy raczej skorzystać z listy obrazów CImgList<T> niż prostej klasy obrazów CImg<T>. Klasa CImg<T> jest w stanie reprezentować wolumetryczne obrazy oszacowanych wektorów pikseli, jak również obrazy o mniejszej liczbie wymiarów (sygnał skalarny 1D, obraz kolorowy 2D, ...). Większość funkcji klasy CImg<T> jest zaprojektowana by posługiwać się maksymalnym przypadkiem wymiarów (3+1). Jako typ T wartości piksela mogą być użyte podstawowe typy C++: unsigned char, char, short, unsigned int, int, unsigned long, long, float, double itp. Typowo szybkie wyświetlenie obrazu można uzyskać używając obrazów CImg<unsigned char>, podczas gdy złożone algorytmy przetwarzania obrazu powinny być kodowane przy użyciu obrazów CImg<float> lub CImg<double>. Domyślną wartością dla szablonu T jest float. Możliwe jest również użycie własnych typów szablonów, jednakże należy wtedy zdefiniować pełny komplet operatorów arytmetycznych i logicznych dla tej klasy.

Struktura CImg<T> zbudowana jest z pięciu pól:

- width - definiuje liczbę kolumn obrazu (rozmiar wzdłuż osi X)
- height - definiuje liczbę wierszy obrazu (rozmiar wzdłuż osi Y)
- depth - definiuje liczbę segmentów obrazu (rozmiar wzdłuż osi Z)
- dim - definiuje liczbę kanałów obrazu (rozmiar wzdłuż osi V)
- data - definiuje wskaźnik do danych piksela typu T

Można uzyskać dostęp do tych pól publicznie, jednakże zalecane jest by używać wyspecjalizowanych do tego funkcji: dimx(), dimy(), dimz(), dimv() oraz ptr(). Wymiary obrazu nie są ograniczone do określonego zasięgu, ograniczone są jedynie

przez ilość dostępnej pamięci. Wartość 1 danego wymiaru zwykle oznacza, że określony wymiar jest płaski, natomiast jeżeli jeden z wymiarów jest 0 lub wskaźnik danych jest pusty wtedy obraz rozpatrywany jest jako pusty. Pusty obraz nie powinien zawierać żadnych danych pikseli i przez to nie będzie przetwarzany przez funkcje klasy CImg. Dane pikseli są gromadzone w pamięci bez przeplotu.

Większość funkcji do przetwarzania klasy CImg<T> posiada dwie wersje: jedna przetwarza na bieżąco natomiast druga zwraca obraz (nazwa funkcji poprzedzona jest przedrostkiem get).

3.5.5. Praca z obrazami

3.5.5.1. Alokacja i zwalnianie obrazu

Deklaracja obrazu może zostać wykonana przy użyciu jednego z kilku dostępnych konstruktorów:

- budowanie obrazów o dowolnych wymiarach:

np. CImg<unsigned char> img(128,128,1,3); deklaruje obraz kolorowy 128x128x1x3, gdzie kolor jest zgromadzony jako obraz z trzema kanałami
Należy tu zapamiętać, iż piksele obrazów nie są automatycznie inicjonowane do wartości 0, by tego dokonać trzeba użyć funkcji fill() lub specyficznego konstruktora pobierającego pięć parametrów jak CImg<> img(128,128,128,3,0);

- budowanie obrazu z pliku

np. CImg<unsigned char> img("source.bmp"); wczytuje obraz kolorowy z pliku o nazwie "source.bmp"
Po za instalowaniu ImageMagick możliwy będzie dostęp do innych skompresowanych formatów obrazów (JPG, PNG,...)

- budowanie obrazu z tablicy stylu C:

np. CImg<unsigned char> img(data_buffer,256,256,1,3,false); buduje obraz kolorowy 256x256, gdzie kanały R,G,B następują jeden za drugim

3.5.5.2. Odczyt i zapis obrazu do pliku

Obrazy mogą być bezpośrednio załadowane oraz zapisane w następujących formatach plików: BMP (nieskompresowany), RAW, ASC (Ascii), HDR (Analyze 7.5), INR (Inrimage), PPM/PGM (Portable Pixmap), PAN (Pandore-5), DLM (Matlab ASCII).
Dodatkowo po zainstalowaniu ImageMagick biblioteka CIMG może ładować i zapisywać obrazy w formatach obsługiwanych przez ImageMagick: JPG, GIF, PNG, TIF.

3.5.5.3. Dostęp do pikseli obrazu

Dla zobrazowania sposobów dostępu do pikseli posłużono się następującymi przykładami:

- `img_1(80,70) = 1234.6; // dostęp do wartości piksela o współrzędnych (80,70)`
- `img_2(20,20,0) = 255; // dostęp do składowej czerwonej piksela o współrzędnych (20,20)`
- `img_3(20,20,1) = 255; //dostęp do składowej zielonej piksela o współrzędnych (20,20)`
- `img_4(10,20,30,2) = 31000; // dostęp do składowej niebieskiej piksela o współrzędnych (10,20,30)`

3.5.5.4. Użycie pętli przez obraz

Biblioteka CIMG dostarcza różne makra, które definiują przydatne iteracyjne pętle przez obraz. Zasadniczo, można ich użyć by zastąpić jedną lub kilka pętli for(...), jak również dostarczają one rozszerzenia do klasycznych pętli. Poniżej przedstawiono cztery różne kategorie, na jakie można podzielić wszystkie istniejące makra pętli:

- pętle przez bufor piksela - są podstawowymi pętlami powtarzającymi wskaźnik na buforze danych piksela obrazu.

- pętle przez wymiary obrazu - są najczęściej używanymi pętlami w programach przetwarzających obraz, pozwalają wykonać pętle przez obraz wzdłuż jednego lub kilku wymiarów.
- pętle przez wewnętrzne obszary lub brzegi, pozwalają wykonać pętle na brzegu obrazu lub wewnątrz obrazu po wyłączeniu brzegu.
- pętle wykorzystujące sąsiedztwo - wewnętrzna pętla obrazu, zwykle przydatna do uzyskania wartości sąsiedztwa aktualnego piksela w pętli. Biblioteka CIMG dostarcza bardzo inteligentnego i szybkiego mechanizmu do tego celu, z definicją kilku makr pętli, które pamiętają wartości sąsiedztwa pikseli. Użycie tych makr wysoce optymalizuje kod jak również upraszcza go.

3.5.6. Konwersja przestrzeni kolorów, progowanie

W CIMG bardzo dobrze opracowano funkcję do konwersji przestrzeni kolorów. Konwersji dokonuję się między przestrzenią RGB a inną przestrzenią, która może być jedną z następujących: HSV, YCbCr, YUV, Lab, XYZ, xyY, LUT, LUT8.

Do progowania obrazu służy threshold(…), gdzie jedynym parametrem jest wartość progu.

3.5.7. Filtracja

W CIMG dostępnych jest szereg funkcji filtrujących obrazy. Funkcje get_blur_median() oraz blur_median() dokonują filtracji medianowej obrazu oraz w niewielkim stopniu rozmywają go. get_blur_anisotropic(...) oraz blur_anisotropic(...) dokonuje filtracji rozmywającej anizotropowej, a funkcje blur(...) oraz get_blur(...) filtracji rozmywającej z filtrem Canny-Deriche. Istnieją również funkcje convolve(...) oraz get_convolve(...) służące do implementowania dowolnych liniowych filtrów, działają one na zasadzie konwolucji obrazu z maską w postaci innego obrazu. Natomiast funkcje deriche(…) oraz get_ deriche(…) realizują filtr Deriche. Dodatkowo w bibliotece tej zaimplementowano funkcje nosie(…) oraz get_noise(…) dodające do obrazu szum gaussowski, jednorodny, sól i pieprz.

3.5.8. Morfologia

Z operacji morfologicznych w CIMG zaimplementowano funkcje wykonujące erozje i dylację. Erozję obrazu elementem strukturalnym w postaci obrazu dokonują funkcje erode(...)(funkcja przetwarzającą na bieżąco) oraz get_erode(...)(funkcja zwracająca obraz). Dylację obrazu elementem strukturalnym w postaci obrazu dokonują funkcje dilate(...)(funkcja przetwarzającą na bieżąco) oraz get_dilate(...)(funkcja zwracająca obraz). Funkcje te nie posiadają ograniczeń, co do liczby przetwarzanych kanałów.

3.5.9. Funkcje do rysowania

Do rysowania, które odbywa się na przykładowym obrazie dostarczono szereg funkcji rysujących punkty, linie, strzałki, obrazy, prostokąty, trójkąty, elipsy, okręgi, osie, tekst itp. Parametry funkcji rysujących są zdefiniowane jako zmienne const i zwracają odniesienie do aktualnego przykładu, dlatego też funkcje rysujące mogą być potokowe. Rysowanie zwykle odbywa się na obrazach kolorowych 2D, ale również może zostać wykonane na obrazach 3D. Parametr color musi być zawsze podany, gdy używa się tych funkcji.

3.5.10. Inne

W bibliotece tej zaimplementowano również szereg innych funkcji wykonujących operacje matematyczne, logiczne, dokonujących transformacji obrazu, wykonujących operacje na macierzach, wyświetlających obrazy w oknie oraz funkcji do obsługi myszki komputerowej oraz klawiatury.

4. Testy bibliotek

W celu bezpośredniego porównania pracy bibliotek został wykonany odpowiedni program testów. Testy składały się z dwóch części. Pierwsza z nich to testy poszczególnych, wspólnych funkcji bibliotek i porównanie ich w szczególności pod względem wydajności. Aby uniezależnić wyniki od innych czynników niż sposób realizacji danego zagadnienia (np. algorytmu przetwarzania obrazów) do testów wybrano funkcje, które wykonujące te same czynności i dające jednakowe (ewentualnie podobne) wyniki we wszystkich bibliotekach. Druga część to napisanie aplikacji do rozpoznawania znaków drogowych i porównanie na jej podstawie funkcjonalności, łatwości implementacji poszczególnych funkcji, przejrzystości kodu tworzonej aplikacji, wydajności w poszczególnych bibliotekach. Zdecydowano się napisać taki program, ze względu na możliwość zaimplementowania w nim dużej ilości funkcji do przetwarzania obrazu.

Testy bibliotek wykonano na komputerze wyposażonym w procesor AMD Athlon XP „Barton" 2500+ o częstotliwości 1833 MHz oraz zawierającym 1 GB pamięci operacyjnej RAM. Na komputerze tym zainstalowany był system operacyjny Windows XP z dodatkiem Service Pack 2. Podczas wykonywania testów ustawiono priorytet procesowi aplikacji na priorytet „Czasu rzeczywistego". Biblioteki OPENCV, IPL98 oraz CIMG komplikowano na MS Visual Studio C++ 2005, natomiast bibliotekę LTI-LIB na kompilatorze MS Visual C++ 6.0, ponieważ nie daję się ona w pełni skompilować pod wersją MS Visual Studio C++ 2005. Aby sprawdzić czy zastosowanie innego kompilatora wpływa na czasy działania funkcji uruchomiono również biblioteki OPENCV, IPL98 oraz CIMG na kompilatorze MS Visual C++ 6.0. Jednakże okazało się, że wyniki działania poszczególnych funkcji dają na obydwu kompilatorach jednakowe rezultaty czasowe. Wynika stąd wniosek, że użycie kompilatorów w wersji MS Visual C++ 6.0 czy MS Visual Studio C++ 2005 nie wpływa na wyniki testów.

4.1. Test funkcji ładujących obraz

Podczas pracy z bibliotekami jedną z pierwszych funkcji wykorzystywanych w większości programów jest funkcja ładująca obraz z pliku. W teście tym we wszystkich bibliotekach dokonano ładowania obrazu kolorowego 24-bitowego oraz obrazu

binarnego 1-bitowego w różnych rozdzielczościach. Wszystkie obrazy były w formacie BMP.

Tabela 5. Tabela przedstawia czasy wykonywania funkcji do ładowania obrazu różnych typów obrazów.

		Biblioteka			
		OPENCV	**IPL98**	**LTI-LIB**	**CIMG**
Typ obrazu	**Obraz 1-bitowy, 640x480**	2,5 ms	0,3 ms	124 ms	6,9 ms
	Obraz 24-bitowy, 640x480	3,7 ms	4,5 ms	166 ms	8,5 ms
	Obraz 1-bitowy, 800x600	3,9 ms	0,4 ms	192ms	10,6 ms
	Obraz 24-bitowy, 800x600	5,7 ms	6,8 ms	258 ms	12,5 ms
	Obraz 1-bitowy, 1024x768	6,4 ms	0,5 ms	311 ms	16,8 ms
	Obraz 24-bitowy, 1024x768	9,3 ms	10,5 ms	425 ms	19,3 ms
	Obraz 1-bitowy, 1280x960	9,8 ms	0,6 ms	486 ms	27,7 ms
	Obraz 24-bitowy, 1280x960	14,5 ms	16,3 ms	661 ms	31,2 ms
	Obraz 1-bitowy, 1600x1200	15,2 ms	0,9 ms	765 ms	42,1 ms
	Obraz 24-bitowy, 1600x1200	22,6 ms	25,4 ms	1040 ms	46,1 ms

Na uwagę zasługują bardzo krótkie czasy trwania ładowania obrazu binarnego w bibliotece IPL98. Najlepsze czasy ładowania obrazu 24-bitowego osiągnięto w bibliotece OPENCV, nieznacznie dłuższe w IPL98, natomiast w CIMG były one około dwukrotnie większe, a w LTI-LIB około 40-50 krotnie większe w porównaniu z OPENCV. W IPL98 istnieje bardzo duża różnica między czasem ładowania obrazu 24-bitowego, a czasem ładowania obrazu 1-bitowego, w innych bibliotekach różnice te nie są już takie duże. Czas wykonywania funkcji ładujących rośnie proporcjonalnie do liczby pikseli w obrazie.

4.2. Test funkcji filtrujących

Do filtracji obrazów biblioteki dostarczają różnych funkcji, nie zawsze tych samych oraz nie zawsze działających identycznie. Wykonano testy działania poszczególnych funkcji na obrazie o rozdzielczości 640x480 oraz głębi 8 bitów na piksel. Tabela 6 obrazuje różnice czasów wykonywania funkcji filtrujących.

Tabela 6. Tabela przedstawia czasy wykonywania różnych funkcji filtrujących obraz.

		Biblioteka			
		OPENCV	**IPL98**	**LTI-LIB**	**CIMG**
Typ filtru	**medianowy z maską 3x3**	21,3 ms	23,1 ms	354 ms	106,3 ms
	konwolucja z maską 3x3	8,9 ms	21,3 ms (w. szybsza), 54,1 ms (w. woln.)	294 ms	121,1 ms
	rozmywający z maską 3x3	2,7 ms	6,5 ms	-	73,1 ms
	Gaussa z maską 3x3	4,3 ms	-	-	-
	obustronny	33,9 ms	-	-	-
	k najbliższych sąsiadów	-	-	5578 ms	-
	filtr deriche	-	-	-	37,5 ms

Najszybszym działaniem funkcji filtrujących odznacza się OPENCV, bardzo dobre czasy osiąga również IPL98, natomiast najwięcej czasu na operacje potrzebuje biblioteka LTI-LIB. Identyczne obrazy wynikowe po zastosowaniu filtracji otrzymano jedynie przy filtracji medianowej w bibliotekach OPENCV, LTI-LIB oraz CIMG, natomiast obraz wynikowy po filtracji medianowej w bibliotece IPL98 nieznacznie różni się od nich, co spowodowane jest rozpatrywaniem innego sąsiedztwa pikseli. W wyniku zastosowania różnych odmian filtrów i ich kombinacji ze sobą lub też rozpatrywania innego sąsiedztwa pikseli obrazy wynikowe po zastosowaniu pozostałych, odpowiednich funkcji filtrujących nieznacznie różnią się między sobą (maksymalna różnica wartości pikseli wynosi 6 w skali od 0 do 255) w każdej z bibliotek. Długość kodu źródłowego potrzebna do zaimplementowania filtracji jest podobnych wielkości we wszystkich bibliotekach. Najobszerniej filtracja rozbudowana jest w OPENCV, gdzie za pomocą szeregu argumentów przy każdej z funkcji można w pełen sposób modyfikować filtry, natomiast najuboższa pod tym względem jest biblioteka IPL98, gdzie na stałe mamy zdefiniowane rozmiary masek oraz sąsiedztwo pikseli. Na uwagę zasługuje również fakt, iż w bibliotekach OPENCV i CIMG można

dokonywać filtracji jednocześnie wszystkich trzech kanałów, natomiast w bibliotekach IPL98 oraz LTI-LIB najpierw należy wyodrębnić poszczególne kanały a następnie filtrować każdy oddzielnie. W bibliotekach OPENCV i CIMG na czas trwania filtracji zarówno dla obrazu kolorowego 24-bitowego jak i obrazu monochromatycznego 8-bitowego jest identyczny.

4.3. Test funkcji wykonujących operacje morfologiczne

Jedynymi wspólnymi operacjami morfologicznymi występującymi we wszystkich bibliotekach i dającymi jednakowe obrazy wynikowe są operacje erozji i dylacji. Testy operacji morfologicznych wykonano na obrazie o rozdzielczości 640x480 i głębi 8-bitowej zbudowanym wyłącznie z obszarów czarno-białych, element strukturalny o rozmiarach 3x3 i wartościach równych 1. Wybrano obraz złożony z elementów czarno-białych, ponieważ w bibliotece IPL98 została zaimplementowana wyłącznie morfologia binarna, tzn. przy wykonywaniu operacji morfologicznych aktywnymi pozycjami są piksele o wartościach różnych od zera.

Tabela 7. Tabela przedstawia czasy wykonywania operacji morfologicznych elementem strukturalnym o rozmiarach 3x3 i wartościach równych 1.

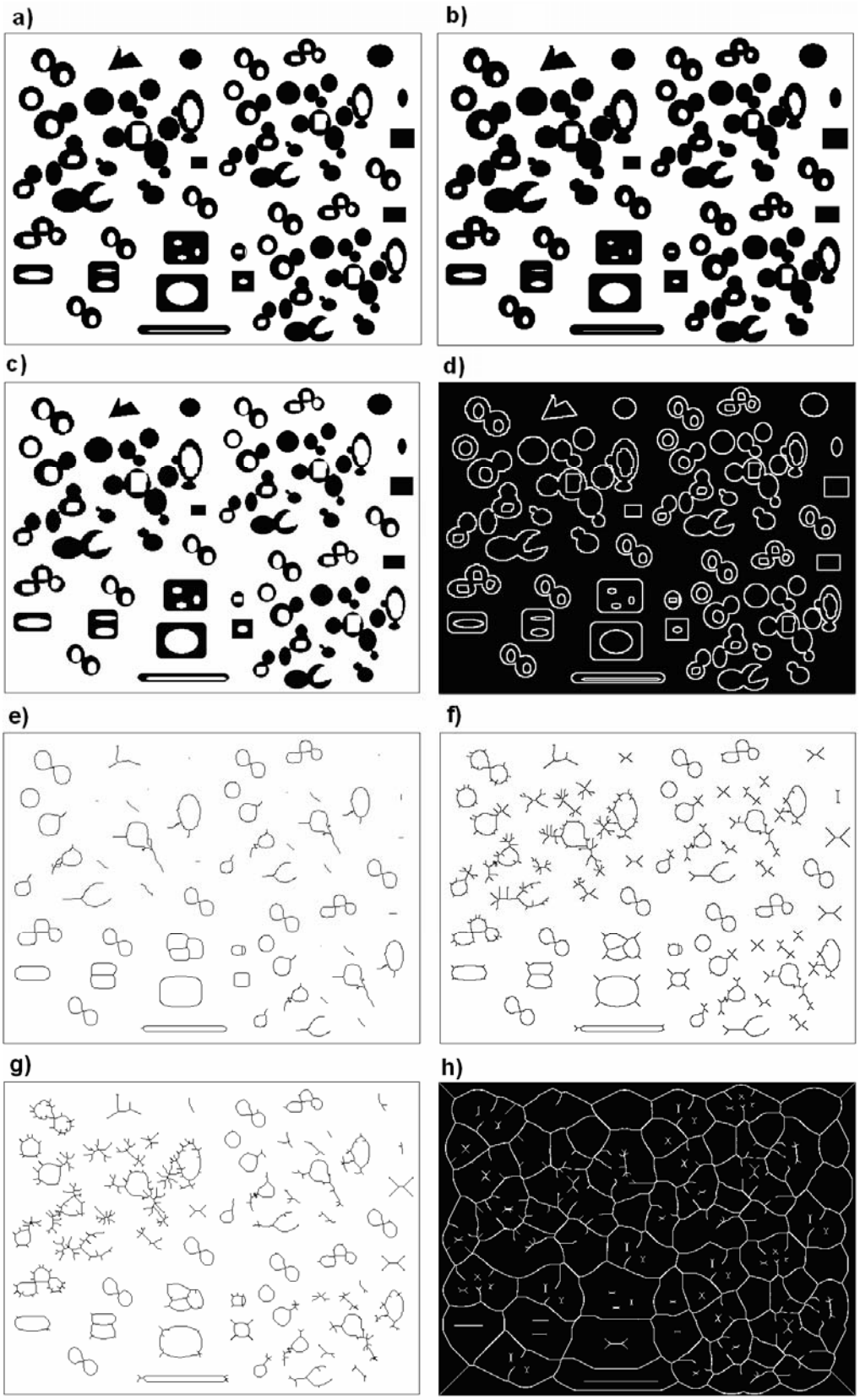
		Biblioteka			
		OPENCV	**IPL98**	**LTI-LIB**	**CIMG**
Operacja morfologiczna	**erozja**	5,6 ms	5,4 ms (wersja szybsza) 185 ms (wersja wolniejsza)	383 ms	183,9 ms
	dylacja	5,6 ms	4,8 ms (wersja szybsza) 226 ms (wersja wolniejsza)	416 ms	183,8 ms

Jak wynika z analizy (Tabela 7) najkrótsze czasy realizacji operacji erozji i dylacji uzyskano w bibliotece IPL98 w wersjach szybszych algorytmów morfologicznych pracujących na obrazach o głębi 1-bitowej, natomiast wersje działające na obrazach

8-bitowej dają czasy około 40-krotnie większe. Porównywalne czasy do wersji szybszych IPL98 uzyskano w bibliotece OPENCV, natomiast dużo większe w bibliotece CIMG a największe LTI-LIB. W bibliotekach OPENCV oraz CIMG można dokonywać operacji morfologicznych na obrazach kolorowych, w LTI LIB na obrazach monochromatycznych, a w IPL98 jak już wcześniej wspomniano wyłącznie na obrazach czarno-białych. We wszystkich bibliotekach z wyjątkiem CIMG zostały zaimplementowane również inne operacje morfologiczne:

- OPENCV: - gradient morfologiczny – czas realizacji 12,4 ms
  - otwarcie i zamknięcie – czas realizacji 10,8 ms
  - operacje „top hat' i „black hat" – czas realizacji 11,9 ms
  - funkcja odległości – 12,5 ms
- IPL98: - ścienianie – czas realizacji 203 ms
  - szkieletyzacja - czas realizacji 372 ms
  - wersja mieszana szkieletyzacji i ścieniania – czas realizacji 242 ms
- LTI-LIB: - szkieletyzacja – czas realizacji 44062 ms
  - funkcja odległości – czas realizacji 460 ms

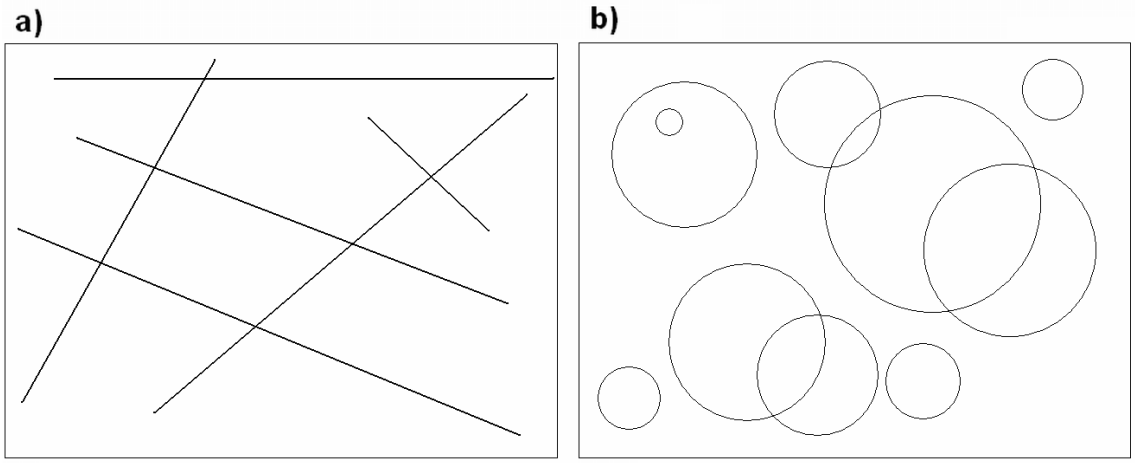
Rysunek 53 przedstawia przykładowe operacje morfologiczne otrzymane w różnych bibliotekach.



Rysunek 53. Operacje morfologiczne. Obraz oryginalny (a), erozja 3x3 (b), dylacja 3x3 (c), gradient morfologiczny (OPENCV) (d), ścienianie (IPL98) (e), szkieletyzacja (IPL98) (f), szkieletyzacja z podcinaniem (IPL98) (g), szkieletyzacja (LTI-LIB) (h).

4.4. Test transformaty Hough'a

W trzech (OPENCV,IPL98, LTI-LIB) spośród czterech badanych bibliotek wstępują funkcje transformaty Hough'a. Aby dokonać ich porównania napisano programy z wykorzystanie bibliotek OPECV, IPL98 i LTI-LIB, których celem było wykrycie i zaznaczenie wszystkich linii z obrazu 8-bitowego o rozdzielczości 640x480 (Rysunek 54) oraz napisano programy z wykorzystaniem bibliotekach OPENCV i IPL98 do wykrywania i zaznaczania okręgów z obrazu 8-bitowego o rozdzielczości 640x480 (Rysunek 54).



Rysunek 54. Obrazy wykorzystane do testów transformaty Hough'a.

Twórcy biblioteki LTI-LIB nie zaimplementowali funkcji umożliwiających wykrywanie okręgów przy pomocy transformaty Hough'a, natomiast twórcy biblioteki CIMG nie opracowali żadnych funkcji związanych z transformatą Hough'a.

W każdej z bibliotek funkcje wykrywające różnią się między sobą, dlatego też podczas testu zdecydowano się przyjąć za podstawowe kryterium dokładne wykrycie wszystkich linii przy tak ustawionych parametrach, które pozwolą to osiągnąć w jak najkrótszym czasie. W związku z powyższym, na czasy wykonania operacji w każdej z bibliotek oprócz sposobu implementacji algorytmu w bibliotece ma również wpływ rodzaj zastosowanego algorytmu.

Czasy osiągnięte w poszczególnych bibliotekach podczas wykrywania linii transformatą Hough'a :

- OPENCV - CV_HOUGH_STANDARD – czas realizacji 28 ms
  - CV_HOUGH_PROBABALISTIC – czas realizacji 28 ms
- IPL98 - ProbRHTLine – czas realizacji 880 ms

- RHTLine – czas realizacji 3800 ms
- SHTLine – czas realizacji 1900 ms

- LTI-LIB - OrientedHLTransform - czas realizacji 125 ms

Czasy osiągnięte w poszczególnych bibliotekach podczas wykrywania okręgów transformatą Hough'a :

- OPENCV - czas realizacji 58 ms
- IPL98 - czas realizacji 2300 ms

Najkrótsze czasy wykonywania funkcji wykrywających zarówno linie jak i okręgi osiągnięto w bibliotece OPENCV. Obrazy, które użyto do testów zawierały odpowiednio bardzo małą liczbę linii i okręgów oraz grubość tychże linii i okręgów była mała. Gdy zwiększy się liczbę linii, okręgów lub też ich grubość zwiększa się również czas potrzebny na ich wykrycie. Dlatego mając to na względzie czasy osiągnięte we wszystkich bibliotekach można uznać je za zbyt długie do zastosowań transformaty Hough'a w aplikacjach czasu rzeczywistego, również w bibliotece OPENCV, gdzie realizacja transformaty Hough'a jest najczęściej krytykowana przez jej użytkowników.

4.5. Test operacji wyrównywania histogramu

Wyrównywanie histogramu jest jedyną operacją na histogramie, która występuje we wszystkich badanych bibliotekach. Operacji wyrównywania histogramu dokonano na obrazie monochromatycznym o rozdzielczości 640x480 i głębi 8-bitowej oraz na obrazie kolorowym o rozdzielczości 640x480 i głębi 24-bitowej. W bibliotekach IPL98 i CIMG oprócz wyrównywania histogramu zaimplementowano jedynie funkcje do jego rysowania, natomiast w bibliotekach OPENCV i LTI-LIB dostarczono szeroki zakres funkcji do operowania na histogramie. W IPL98 i CIMG wyrównywanie histogramu można wykonywać bezpośrednio na trzech kanałach jednocześnie, natomiast w OPENCV i LTI-LIB operację to trzeba wykonywać na każdym kanale oddzielnie po wcześniejszym wyodrębnieniu każdego kanału. Podczas wyrównywania histogramu obrazu kolorowego do czasu wykonywania operacji w bibliotekach OPENCV i LTI-LIB został doliczony czas potrzebny na wydzielenie poszczególnych kanałów z

obrazu kolorowego, a także czas potrzebny na utworzenie obrazu z trzech kanałów po wykonaniu wyrównywania. W bibliotekach tych powoduje to większy niż 3-krotny wzrost czasu wyrównywania histogramu obrazu kolorowego do czasu wyrównywania histogramu obrazu monochromatycznego.

Tabela 8. Tabela przedstawia czasy wykonywania operacji wyrównywania histogramu.

		Biblioteka			
		OPENCV	**IPL98**	**LTI-LIB**	**CIMG**
Wyrównywanie histogramu	**obraz monochromatyczny**	1,8 ms	9,1 ms	210 ms	96,3 ms
	obraz kolorowy	10,1 ms	91,3 ms	860 ms	96,3 ms

Operacja wyrównywania histogramu zarówno dla obrazu monochromatycznego, jaki kolorowego została najszybciej wykonana w bibliotece OPENCV (Tabela 8). W bibliotece CIMG czasy dla obydwu operacji są jednakowe, wynika to z tego, iż obraz niezależnie od głębi (czy 8-bitowej czy też 24-bitowej) podlega tym samym przekształceniom, natomiast w IPL98 dziesięciokrotny wzrost czasu między obrazem monochromatycznym a kolorowym spowodowany jest tym, że funkcje do pobierania i zapisywania pikseli w obrazie 8-bitowym są znacznie szybsze od odpowiadających im funkcji dla obrazu 24 -bitowego. Brak możliwości wyrównywania histogramu na trzech kanałach obrazu wpłynął także na objętość kodu źródłowego w OPENCV i LTI LIB, która się zwiększyła w porównaniu do bibliotek IPL98 i CIMG.

4.6. Test operacji progowania obrazu

W teście tym dokonano progowania w każdej z bibliotek obrazu monochromatycznego 8-bitowego o rozdzielczości 640x480 z wartością progu równą 130. Operacja progowania jest jedną z najprostszych operacji, jakie są zaimplementowane w bibliotekach. W związku z powyższym wyniki testów obrazują,

w której z bibliotek dostarczono najlepszej struktury danych dla obrazów oraz najbardziej zoptymalizowano kod źródłowy.

Czasy osiągnięte w poszczególnych bibliotekach podczas operacji progowania obrazu:

- OPENCV - czas realizacji 1,1 ms
- IPL98 - czas realizacji 8,3 ms
- LTI-LIB - czas realizacji 67 ms
- CIMG - czas realizacji 11,4 ms

4.7. Test sekwencji operacji do przetwarzania obrazu

W celu sumarycznego porównania wydajności poszczególnych bibliotek wykonano test sekwencji składającej się z następujących operacji: ładowanie obrazu, filtracja medianowa, zamknięcie, progowanie. Test wykonano na obrazie monochromatycznym o rozdzielczości 640x480 pikseli, obraz złożony był wyłącznie z elementów czarno-białych.

Czasy jakie osiągnięto w poszczególnych bibliotekach:

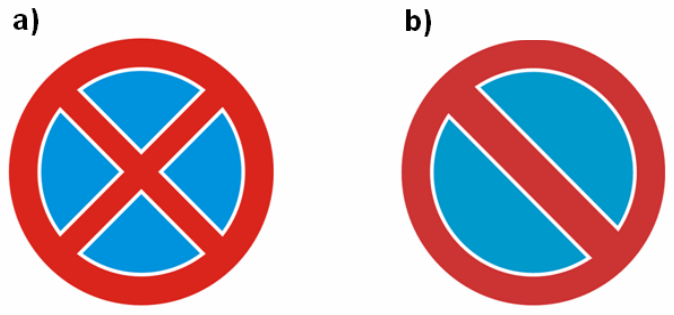
- OPENCV - czas realizacji 37,3 ms
- IPL98 - czas realizacji 47,9 ms
- LTI-LIB - czas realizacji 1386 ms
- CIMG - czas realizacji 493,8 ms

Podobnie jak we wcześniejszych testach najszybszą biblioteką okazała się OPENCV, a najwolniejszą LTI-LIB.

4.8. Programy do rozpoznawania znaków drogowych

Dynamiczny rozwój technik komputerowych pozwala na fizyczną realizację coraz bardziej skomplikowanych przedsięwzięć, co powoduje zainteresowanie automatyzacją procesów, które do tej pory były wykonywane częściowo lub całkowicie ręcznie. W ramach pracy zaproponowana została metoda rozpoznawania (klasyfikacji) pionowych znaków drogowych ze zdjęć - znaku zakazu zatrzymywania oraz znaku zakazu postoju (Rysunek 55). Napisane programy w bibliotekach OPENCV, LTI-LIB oraz IPL98 realizujące tą metodę miały posłużyć porównaniu bibliotek pod względem przejrzystości kodu źródłowego, łatwości implementacji poszczególnych funkcji,

funkcjonalności oraz wydajności. W programach tych została użyta szeroka gama funkcji zaimplementowanych w bibliotekach do przetwarzania obrazu.

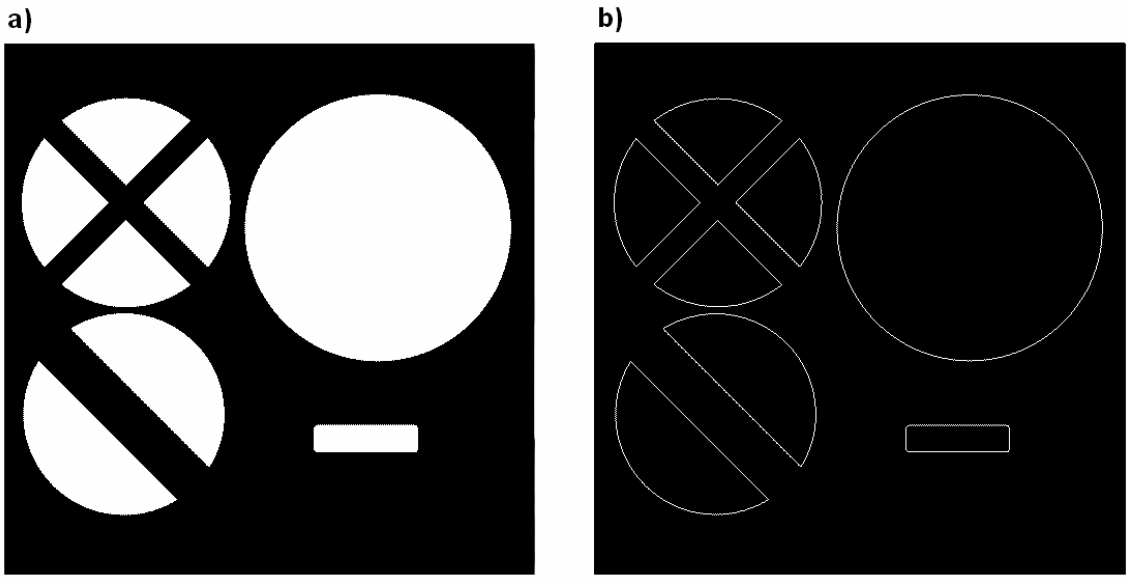


Rysunek 55. Znak zakazu zatrzymywania się (a), znak zakazu postoju (b).

Metoda rozpoznawania znaków drogowych składa z kilku etapów.

Etap 1. Wczytanie obrazu z wzorcowymi obiektami.

Funkcją odpowiedzialną za pierwszy etap, która uruchamiana jest jako pierwsza po starcie programu jest loading_sample().W funkcji tej wykonywane jest ładowanie obrazu o rozdzielczości 512x512 z wzorcami (Rysunek 56). Jest to sztucznie zbudowany obraz składający się z obiektów, które będą porównywane przy pomocy niezmienników momentowych z obiektami wyodrębnionymi z obrazu rozpoznawanego (Rysunek 57). Obraz ze wzorcami został tak zbudowany, aby we wszystkich bibliotekach można było z niego wydobyć obiekty wykorzystywane do późniejszego porównywania. Jeżeli obraz ze wzorcami jest głębi 24-bitowej w funkcji tej wykonywana jest konwersja obrazu RGB do obrazu monochromatycznego. Ze względu, że dostarczony obraz jest obrazem sztucznym i nie zawiera defektów zrezygnowano z wykonywania przetwarzania wstępnego w postaci filtracji medianowej obrazu, operacji zamknięcia oraz progowania. Następnie z obrazu wydzielane są kontury w przypadku bibliotek OPENCV i LTI-LIB oraz obszary w przypadku bibliotek IPL98, a następnie z tych obszarów wydzielane są kontury. Wydzielone kontury zapisywane są do tablicy sample_obj. Rysunek 56 przedstawia obraz ze wzorcami oraz obraz wynikowy po wykrywaniu konturów z obrazu ze wzorcami.



Rysunek 56. Obraz ze wzorcami (a), obraz po wykrywaniu konturów z obrazu ze wzorcami (b).

Podsumowanie etapu 1.

Biblioteki OPENCV oraz LTI-LIB zawierają funkcje do wykrywania wszystkich konturów z obrazu, natomiast IPL98 zawiera funkcje (FindAndFollowLowContour(...), FindAndFollowHighContour(...)), które dokonują poszukiwania konturu do momentu znalezienia pierwszego konturu a następnie przerywają swoje działanie, w związku z czym zdecydowano się wcześniej zastosować funkcję do znalezienia wszystkich obszarów w postaci blobów funkcją DeriveBlobs(...) a następnie poprzez zaznaczenie obszaru zainteresowania ROI wokół każdego znalezionego obszaru wykrycie konturu tego obszaru. Dodatkowe przekształcanie z obszarów pikseli do konturów wiąże z wydłużeniem działania funkcji loading_sample(), ale spowodowało skrócenie czasu potrzebnego na wyznaczenie niezmienników momentowych Hu, ponieważ czas obliczeniowy potrzebny do wyznaczenia niezmienników jest zależny od liczby pikseli, a także poprawiło wyniki porównywania.

Etap 2. Ładowanie obrazu źródłowego oraz jego wstępne przetwarzanie.

Następnie w programie wykonywana jest funkcja loading_image(), dokonuje ona ładowania obrazu, z którego będą rozpoznawane znaki drogowe (Rysunek 57). Wstępnie obraz ten jest odszumiany filtrem medianowym w celu usunięcia drobnych defektów obrazu oraz wykonywana jest na nim operacja zamknięcia, aby pozbyć się drobnych dziur w obrazie. Wynikiem działania funkcji jest obraz po wstępnym przetwarzaniu.

Rysunek 57. Przykładowe zdjęcie ze znakiem zakazu zatrzymywania.

Podsumowanie etapu 2.

Różnice w wykonaniu działania tego etapu w poszczególnych bibliotekach polegają na tym, iż w bibliotece OPENCV zostały zaimplementowane funkcje filtrujące działające na obrazach kolorowych, natomiast w bibliotekach LTI-LIB i IPL98 z obrazu kolorowego 24-bitowego wyodrębniane są poszczególne kanały przestrzeni RGB a następnie przetwarzane każdy kanał niezależnie, a po przetwarzaniu łączone w jeden obraz kolorowy, 24-bitowy.

Etap 3. Rozpoznawanie kolorów.

Po zakończeniu drugiego etapu uruchamiana jest funkcja find_color() do rozpoznawania kolorów z obrazu rozpoznawanego. W funkcji tej do wykrycia interesującego koloru posłużono się pojęciem odległości euklidesowej [32] oraz przestrzenią HSV w celu uniezależnienia wyników rozpoznawania od oświetlenia (składowa H jest nie zależna od jasności), gdzie każdy piksel jest klasyfikowany niezależnie na podstawie wartości składowych przestrzeni HSV. Obrazy wejściowe we wszystkich bibliotekach są w przestrzeni RGB, toteż piksele są przekształcane z przestrzeni RGB na przestrzeń HSV w następujący sposób:

$$\text{Jeżeli} \quad MAX = R_1 \quad \text{wtedy} \qquad H = \frac{G_1 - B_1}{MAX - MIN} 60$$

$$\text{Jeżeli} \quad MAX = G_1 \quad \text{wtedy} \qquad H = \frac{2 + (B_1 - R_1)}{MAX - MIN} 60$$

$$\text{Jeżeli } MAX = B_1 \quad \text{wtedy} \quad H = \frac{4 + (R_1 - G_1)}{MAX - MIN} 60$$

$$\text{Jeżeli } H < 0 \quad \text{wtedy} \quad H = H + 360$$

$$V = MAX$$

$$\text{Jeżeli } V \neq 0 \quad \text{wtedy} \quad S = \frac{MAX - MIN}{MAX} 255$$

gdzie:

MAX – wartość maksymalna ze składowych R,G,B piksela

MIN – wartość minimalna ze składowych R,G,B piksela

R_1 – wartość składowej R przestrzeni RGB piksela obrazu rozpoznawanego

G_1 – wartość składowej G przestrzeni RGB piksela obrazu rozpoznawanego

B_1 – wartość składowej B przestrzeni RGB piksela obrazu rozpoznawanego

H – wartość składowej H przestrzeni HSV piksela obrazu rozpoznawanego

S – wartość składowej S przestrzeni HSV piksela obrazu rozpoznawanego

V – wartość składowej V przestrzeni HSV piksela obrazu rozpoznawanego

Do rozpoznania poszczególnych kolorów zostały dobrane następujące reguły decyzyjne:

jeśli ($H < 30$ lub $H > 300$) to kolor czerwony

jeśli ($H > 150$ *i* $H < 300$) to kolor niebieski

jeśli ($H > 30$ *i* $H < 90$) to kolor żółty

gdzie reguły decyzyjne mają zakres dla H od 0 do 360, S od 0 do 255 i V od 0 do 255.

Dla każdego piksela obrazu obliczona jest odległość euklidesowa „D" między poszczególnymi składowymi przestrzeni RGB piksela obrazu rozpoznawanego a składowymi przestrzeni RGB koloru szukanego.

$$D = \sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2}$$

gdzie:

D – odległość Euklidesowa pomiędzy dwoma wektorami

R_1 – wartość składowej R przestrzeni RGB piksela obrazu rozpoznawanego

R_2 – wartość składowej R przestrzeni RGB szukanego koloru

G_1 – wartość składowej G przestrzeni RGB piksela obrazu rozpoznawanego

G_2 – wartość składowej G przestrzeni RGB szukanego koloru

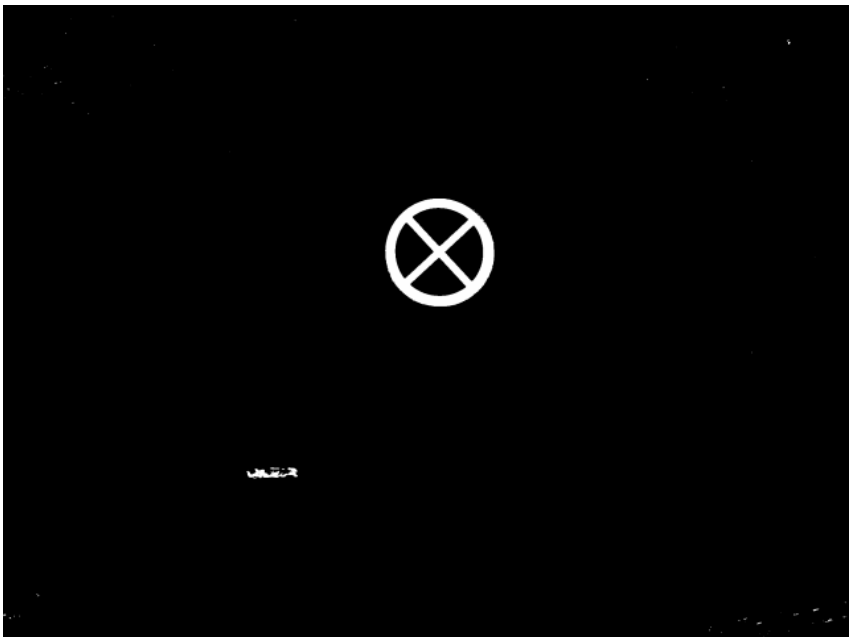
B_1 – wartość składowej B przestrzeni RGB piksela obrazu rozpoznawanego

B_2 – wartość składowej B przestrzeni RGB szukanego koloru

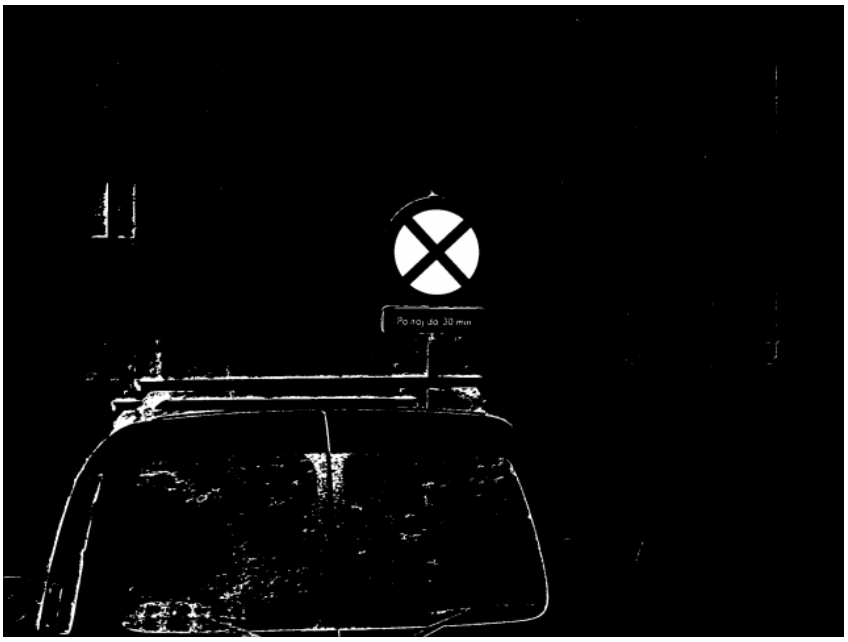
Następnie każdy piksel jest klasyfikowany, jeżeli wartość odległości euklidesowej jest poniżej zadanego progu oraz wartość składowej H zawiera się w przedziale charakterystycznym dla danego koloru, wtedy pikselowi przypisywana jest wartość 255, natomiast jeśli jest powyżej zadanego progu oraz wartość składowej H nie mieści się w przedziale charakterystycznym dla danego koloru pikselowi przypisywana jest wartość 0. Zadany próg tolerancji odległości euklidesowej jest wartością ustawianą manualnie i wyznaczany jest metodą prób i błędów. Metodą tą z obrazu źródłowego wydzielane są obrazy zawierające kolor czerwony oraz kolor niebieski (kolory występujące w znakach zakazu zatrzymania i postoju).

Podsumowanie etapu 3.

Etap ten we wszystkich bibliotekach jest wykonywany w sposób jednakowy. W bibliotece OPENCV zaimplementowane jest między innymi przekształcenie z przestrzeni RGB na HSV, jednakże w funkcji find_color() nie zostało wykorzystane to przez autora. Przy pomocy funkcji find_color() wykonywane jest rozpoznawanie koloru czerwonego i niebieskiego, które są odpowiednio zapisywane pod obrazami red_img (Rysunek 58) oraz blue_img (Rysunek 59).



Rysunek 58. Obraz po zastosowaniu funkcji do rozpoznawania elementów czerwonych.



Rysunek 59. Obraz po zastosowaniu funkcji do rozpoznawania elementów niebieskich.

Etap 4. Podział obrazów na kontury, przetwarzanie konturów, określenie przynależności do wzorca.

Następną uruchamianą funkcją jest find_contours(). W której na wstępie dokonywana jest ponownie filtracja medianowa oraz operacja zamknięcia w celu wyeliminowania defektów powstałych podczas klasyfikacji kolorów oraz wykonywane jest progowanie obrazów. Następnie z obrazów po klasyfikacji kolorów wydzielane są kontury w przypadku bibliotek OPENCV i LTI-LI oraz obszary w przypadku biblioteki IPL98, a następnie z tych obszarów wydzielane są kontury podobnie jak w etapie 1.

W funkcji processing_contours() następuje zapisanie konturów do odpowiednich tablic. Kontury i regiony z obrazu z czerwonego są zapisywane do tablicy forms_red natomiast kontury i regiony z obrazu niebieskiego zapisywane są do tablicy forms_blue. W polach tych struktur zapisywane są informacje potrzebne do późniejszego określenia czy dany kontur należy do znaku drogowego. W bibliotece OPENCV obliczany jest najmniejszy prostokąt otaczający kontur i zapisywany w polu box tych tablic. W bibliotece LTI-LIB obliczane są cechy konturów (momenty, niezmienniki momentowe, wymiary regionów), które są zapisywane są w polu features, natomiast w bibliotece IPL98 do informacji tych można bezpośrednio się odwołać przy pomocy funkcji operujących na grupie pikseli, z których jest zbudowany kontur. Następnie dla każdego konturu znajdywany jest najbardziej podobny odpowiednik z pośród konturów

wyodrębnionych z obrazu wzorcowego zapisanych w tablicy sample_obj. W bibliotece OPENCV posłużono się funkcją cvMatchShapes() porównującą kształty przy pomocy siedmiu niezmienników momentowych Hu oraz napisano dodatkową funkcję porównującą kontury na podstawie trzech niezmienników momentowych, natomiast w bibliotekach IPL98 oraz LTI-LIB napisano analogiczne funkcje do funkcji cvMatchShapes() oraz do funkcji porównującej kontury na podstawie trzech niezmienników momentowych. Funkcje te zwracają „odległość" między porównywanymi kształtami. Za wybór wzorca z minimalną odległością pomiędzy porównywanymi kształtami odpowiada funkcja compare().

Podsumowanie etapu 4.

Podobnie jak w etapie 1 podczas wykrywania konturów z obrazu wzorcowego również podczas wykrywania konturów z obrazów img_red oraz img_blue istnieją różnice w sposobie wykrywania konturów. Różnice te zostały dokładniej opisane w etapie 1. Otrzymane niezmienniki momentowe Hu z bibliotek OPENCV, LTI-LIB oraz IPL98 różnią się między sobą i w bibliotekach LTI oraz IPL98 nie dają precyzyjnych wyników przy porównywaniu zaimplementowaną funkcją na wzór funkcji cvMatchShapes z OPENCV. Znacznie lepsze wyniki uzyskano po zastosowaniu porównania na podstawie trzech niezmienników momentowych:

$$I_4(A,B)=\sum_{i=1..3}\left|m^A_i - m^B_i\right| / \left|m^A_i\right|$$

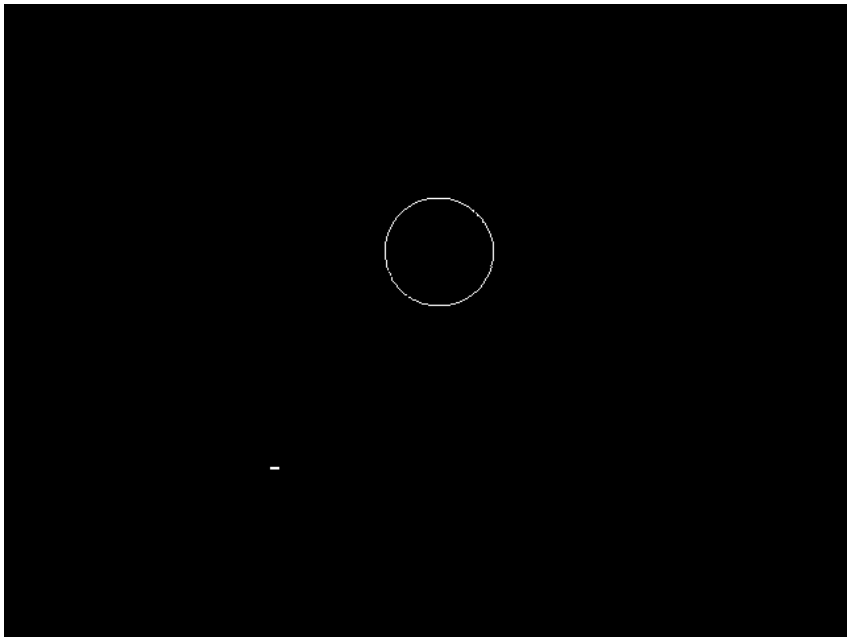
gdzie:

A - pierwszy kształt, B - drugi kształt,

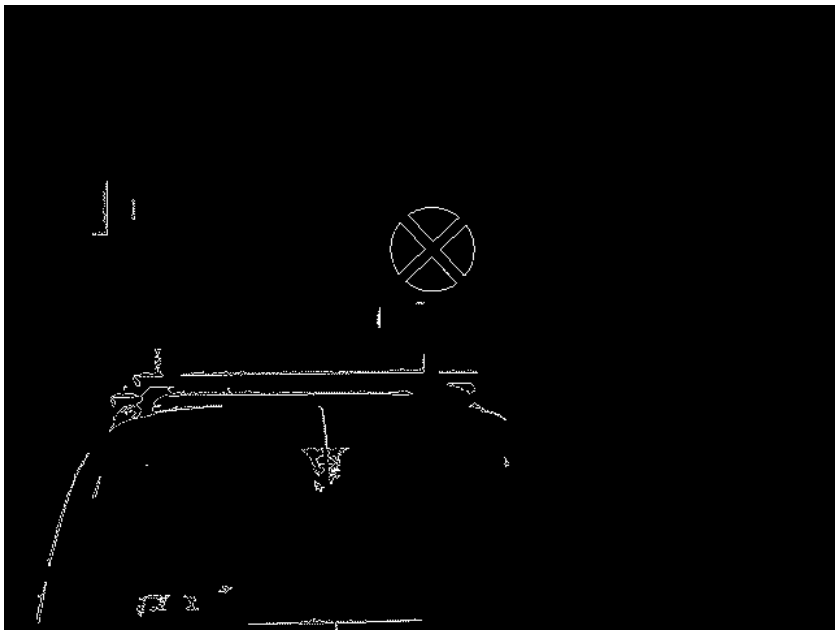
$m^A_i=\mathrm{sign}(h^A_i)\cdot\log(h^A_i)$,

$m^B_i=\mathrm{sign}(h^B_i)\cdot\log(h^B_i)$,

h^A_i, h^B_i – niezmienniki momentowe Hu odpowiednio kształtu A i B.



Rysunek 60. Obraz z wyodrębnionymi konturami z obrazu red_img (Rysunek 58).



Rysunek 61. Obraz z wyodrębnionymi konturami z obrazu blue_img (Rysunek 59).

Etap 5. Klasyfikacja przynależności konturów do znaku, wydzielenie znaków drogowych.

Następny etapem działania tego programu jest określenie czy dany kontur należy do znaku drogowego i rozpoznanie znaku.

Pierwszym warunkiem rozpoznania znaku na zdjęciu jest sprawdzenie czy kontury wydzielone z obrazu img_red, czyli zawierające potencjalnie obwiednie znaku mają

wysokość z pewną tolerancją równą szerokości. Jako graniczną tolerancję odstępstwa wysokości od szerokości konturu ustawiono +/- 10%. Sprawdzane jest też czy kontur składa się z przynajmniej 40 pikseli oraz czy został przyporządkowany do odpowiedniego wzorca. Jeśli kontur spełni te warunki, szukane są kontury niebieskie, które w całości są zawarte w wewnątrz konturu czerwonego. Sprawdzane jest również czy kontury niebieskie składają się z przynajmniej 20 pikseli oraz czy zostały przyporządkowane do wzorców ze znaku zakazu zatrzymania czy do wzorców ze znaku zakazu postoju. Jeżeli kontury niebieskie są w liczbie 4 i przyporządkowane zostały do wzorców ze znaku zakazu zatrzymania to zapisywane są do tablicy halt_sign, natomiast jeżeli kontury niebieskie są w liczbie 2 i przyporządkowane zostały do wzorców ze znaku zakazu postoju to zapisywane są do tablicy rank_sign. Zaś kontur czerwony zapisywany jest w tablicy sign. W ten sposób klasyfikowane wszystkie znalezione kontury. Wynikiem działania programu jest obraz z rozpoznanymi znakami drogowymi (Rysunek 62).

Rysunek 62. Obraz z rozpoznanym znakiem zakazu zatrzymania na tle obrazu źródłowego.

Podsumowanie

Największą funkcjonalnością podczas tworzenia powyższej aplikacji wykazała się biblioteka OPENCV. Zawiera ona najszerszą gamę algorytmów do przetwarzania obrazów oraz najlepiej zdefiniowane struktury danych, dzięki którym można w sposób elastyczny budować aplikację. Wysoką funkcjonalność wykazała również biblioteka LTI-LIB. Najmniejszą funkcjonalnością wykazała się biblioteka IPL98, której znaczna

cześć funkcji działa wyłącznie na obrazach binarnych, co znacznie utrudnia tworzenie aplikacji.

Długość kodu aplikacji w OPENCV wynosiła 500 linii kodu, w LTI-LIB 680 linii kodu, natomiast w IPL98 630 linii kodu. Kod ten po wstępnym zapoznaniu się obserwatora się z bibliotekami jest dla niego przejrzysty i nie powinien być trudny do zrozumienia. Pod tym względem OPENCV wykazała się największą kompaktowością dostarczając jednocześnie największą funkcjonalność.

Czas wykonywania aplikacji dla poszczególnych bibliotek:

- OPENCV - obraz o rozm. 640x480: 406 ms
  - obraz o rozm. 2272x1704: 4646 ms
- IPL98 - obraz o rozm. 640x480: 2062 ms
  - obraz o rozm. 2272x1704: 27742 ms
- LTI-LIB - obraz o rozm. 640x480: 5960 ms
  - obraz o rozm. 2272x1704: 68562 ms

W celu poprawy działania aplikacji należałoby dodatkowo zaproponować lepszy algorytm porównywania niezmienników momentowych dla każdej z bibliotek.

5. Podsumowanie i wnioski

5.1. Wnioski

Tabela 9. Zbiorcza tabela z cechami poszczególnych bibliotek.

	Biblioteka			
	OPENCV	**IPL98**	**LTI-LIB**	**CIMG**
Struktury danych	bardzo dobre	średnie	dobre	dobre
Funkcjonalność	bardzo dobra	średnia	dobra	podstawowa
Przejrzystość kodu źródłowego biblioteki	mała	dobra	dobra	dobra
Przejrzystość kodu tworzonej aplikacji	bardzo dobra	bardzo dobra	bardzo dobra	bardzo dobra
Wydajność	bardzo dobra	średnia	słaba	średnia
Niezależność systemowa	dobra	średnia	średnia	bardzo dobra
GUI	bardzo dobre	brak	średnie	słabe

Najbardziej kompletną biblioteką jest OPENCV, charakteryzuje się ona bardzo dobrą struktura danych, bardzo dużą liczbą różnorodnych zaimplementowanych algorytmów, na bazie których można budować potężne aplikacje przetwarzania obrazów i wizji komputerowej działające w czasie rzeczywistym. Jest ona bardzo szybka a dodatkowo może ją jeszcze bardziej zoptymalizować na procesorach firmy Intel doinstalowując do niej bibliotekę IPP. Instalacja IPP zmniejsza czas wykonywania praktycznie każdej funkcji (Tabela 1). Jako wadę biblioteki tej można uznać to, iż kod źródłowy tej biblioteki jest bardzo trudny do zrozumienia oraz to, iż bibliotekę tę jest

ciężko połączyć z innymi bibliotekami. Biblioteka ta jest wykorzystywana w wielu aplikacjach [27] [26].

Bardzo rozbudowaną biblioteką jest LTI-LIB, ilością opracowanych algorytmów prawie dorównuje bibliotece OPENCV, a przy tym jest ona napisana w sposób zrozumiały, jest kompatybilna z innymi bibliotekami. Największą jej wadą natomiast jest bardzo długi czas trwania algorytmów, nie jednokrotnie czterdzieści razy większy od biblioteki OPENCV.

Biblioteka CIMG jest dobrym wyborem dla studentów chcących tworzyć własne algorytmy a potrzebujących biblioteki z dobrze opracowaną podstawową strukturą danych oraz zawierającej funkcje do ładowania, zapisywania, wizualizacji danych oraz podstawowe funkcje do przetwarzania obrazów. Jest najbardziej niezależna systemowo i sprzętowo spośród wszystkich badanych bibliotek.

Biblioteka IPL98 charakteryzuje się bardzo szybkimi funkcjami operującymi na obrazach 1-bitowych osiągającymi czasy porównywalne z biblioteką OPENCV. Nie jest ona aż tak bardzo rozbudowana jak OPENCV i LTI-LIB, jednakże dostarcza wszystkich podstawowych funkcji do przetwarzania obrazu.

Wybór biblioteki należy uzależnić od postawionego do rozwiązania problemu. Jednakże najlepszym wyborem w większości przypadków będzie OPENCV.

5.2. Podsumowanie

Celem niniejsze pracy było przeanalizowanie, opisanie i przetestowanie bibliotek w taki sposób, aby wspomóc użytkownika przy wyborze oprogramowania a następnie przy programowaniu aplikacji przetwarzania obrazów i wizji komputerowej. Cel ten, zdaniem autora, został osiągnięty. Praca ta wymagała dużo poświęcenia oraz czasu, związane było to głownie z brakiem dokumentacji technicznej lub też ze słabą jej jakością, błędami występującymi w dostarczanym przez twórców oprogramowaniu. Rezultatem tejże pracy jest obszerna dokumentacja poszczególnych bibliotek, na którą składają się między innymi takie zagadnienia jak instrukcje instalacji bibliotek oraz opis i analiza struktur danych, funkcji, możliwości. Trudności przysporzyło również zaproponowanie oraz realizacja testów. Związane było to głownie z rozbieżnościami realizacji poszczególnych zagadnień w każdej z bibliotek, a jak wiadomo testy te miały być w jak największym stopniu uniezależnione od typu zastosowanego algorytmu.

Efektem pracy są programy do testów bibliotek, a wśród nich aplikacja do rozpoznawania znaków drogowych ze zdjęć, przykładowe kody obrazujące realizację poszczególnych zagadnień w danych bibliotekach oraz wersje instalacyjne bibliotek po usunięciu z nich błędów. Cała część informatyczna zawarta jest na płycie CD dostarczonej wraz z pracą dyplomową.

6. LITERATURA

[1] Bilmes J.: „A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models", Technical Report TR-97-021 by the International Computer Science Institute, Berkeley, 1998.

[2] Bouguet J.-Y.: „Pyramidal Implementation of the Lucas - Kanade Feature Tracker".

[3] Bradski G., Davis J.: „Motion Segmentation and Pose Recognition with Motion History Gradients." IEEE WACV'00, 2000.

[4] Bradski G.: „Computer vision face tracking as a component of a perceptual user interface." In Workshop on Applications of Computer Vision, pages 214–219, Princeton, NJ, Oct. 1998.

[5] Canny J.: „A Computational Approach to Edge Detection", IEEE Trans. on Pattern Analysis and Machine Intelligence, 8(6), pp.679-698 (1986).

[6] Comaniciu D., Meer P.: "Mean Shift: A Robust Approach toward Feature Space Analysis" appeared in IEEE Trans. Pattern Analysis Machine Intell. Vol. 24, No. 5, 603-619, 2002.

[7] Davis J., Bobick A.F.: „The Representation and Recognition of Action Using Temporal Templates." MIT Media Lab Technical Report 402, 1997.

[8] Davis J., Bradski G.: „Real-time Motion Template Gradients using Intel CVLib" IEEE ICCV Workshop on Frame-rate Vision, September 1999.

[9] Davis J.W., Bobick A.F.: „The Representation and Recognition of Action Using Temporal Templates", Proceedings Computer Vision and Pattern Recognition, 1997.

[10] De Smet P., Pires R.L.V.P.M.: "Implementation and analysis of an optimized rainfalling watershed algorithm". IS&T\SPIE's 12th Annual Symposium Electronic Imaging 2000, January 2000, pp. 759-766.

[11] Finlayson G.D., Schiele B., Crowley J. L.: "Comprehensive Colour Image Normalization", In ECCV'98 Fifth European Conference on Computer Vision, 1998.

[12] Ford A., Roberts A.: „Colour Space Conversions", http://www.poynton.com/PDFs/coloureq.pdf

[13] Gevers T., Stokman H.: "Classifying Color Edges in Video into Shadow-Geometry, Highlight or Material Transitions", IEEE Trans. on Multimedia, Vol. 5 No. 2, June 2003.

[14] Gonzalez R. C., Woods R. E.: „Digital Image Processing”, 2nd Edition, Prentice Hall, 2002.

[15] Harris C., Stephens M.: "A Combined Corner and Edge Detector", in Proc. 4th Alvey Vision Conference, pp. 147-151, 1988.

[16] Horn B.K.P., Schunck B.G.: „Determining Optical Flow”, Artificial Intelligence, 17, pp. 185-203, 1981.

[17] Kass M., Witkin A., Terzopoulos D.: „Snake: Active Contour Models,” International Journal of Computer Vision, vol. 1, pp. 321 - 331, 1988.

[18] Lowe D.G.: "Object Recognition from Local Scale-Invariant Features" Proc. of the International Conference on Computer Vision, Corfu, Greece, 1999.

[19] Lucas B., Kanade T.: „An Iterative Image Registration Technique with an Application to Stereo Vision.” Proc. of 7th International Joint Conference on Artificial Intelligence (IJCAI), pp. 674-679.

[20] Nieniewski M.: „Morfologia matematyczna w przetwarzaniu obrazów”, Akademicka Oficyna Wydawnicza PLJ, 1998.

[21] Pratt W.K.: „Digital Image Processing”, Wiley, 2001.

[22] Quek C., Zhou G. S. Ng. R. W.: „A novel single-pass thinning algorithm and an effective set of performance criteria.” Pattern Recognition Letters”, 16:1267–1275, 1995.

[23] Śluzek A.: „Komputerowa analiza obrazów”, Wydawnictwa Politechniki Warszawskiej, 1991.

[24] Śluzek A.: „Zastosowanie metod momentowych do identyfikacji obiektów w cyfrowych systemach wizyjnych”, Instytut Automatyki Politechniki Warszawskiej, 1989.

[25] Sonka M., Hlavac V., Boyle R.: „Image Processing, Analysis, and Machine Vision” Chapman and Hall Publishers, London, 1993.

[26] Strona internetowa http://swis.epfl.ch/research/swistrack/

[27] Strona internetowa http://www.grandchallenge.org/

[28] Strona internetowa: http://cimg.sourceforge.net/

[29] Strona internetowa: http://en.wikipedia.org/

[30] Strona internetowa: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/ISARD1/condensation.html

[31] Strona internetowa: http://ltilib.sourceforge.net/

[32] Strona internetowa: http://pl.wikipedia.org/

[33] Strona internetowa: http://sourceforge.net/projects/cimg/

[34] Strona internetowa: http://sourceforge.net/projects/opencvlibrary/

[35] Strona internetowa: http://www.cs.cmu.edu/~cil/vision.html

[36] Strona internetowa: http://www.intel.com/technology/computing/opencv/

[37] Strona internetowa: http://www.mip.sdu.dk/ipl98/

[38] Strona internetowa: https://sourceforge.net/projects/ipl98/

[39] Tadeusiewicz R., Flasiński M.: „Rozpoznawanie obrazów", PWN 1991.

[40] Tadeusiewicz R., Korohoda P.: „Komputerowa analiza i przetwarzanie obrazów", Wydawnictwo Fundacji Postępu Telekomunikacji, 1997.

[41] Tadeusiewicz R.: „Systemy wizyjne robotów przemysłowych", WNT 1992.

[42] Vincent L., Soille P.: "Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations". IEEE transactions on pattern analysis and machine intelligence, vol. 13, No. 6, June 1991, pp. 583-598.

[43] Watkins C.D., Sadun A., Marenka S.: „Nowoczesne metody przetwarzania obrazu", WNT 1995.

[44] Welch G., Bishop G.: „An Introduction To the Kalman Filter", Technical Report TR95-041, University of North Carolina at Chapel Hill, 1995.

[45] Wren Ch., Azarbayejani A., Darrell T., Pentland A.: „Pfinder: Real-Time Tracking of the Human Body" IEEE Transactions on Pattern Analysis and Machine Intelligence, 1997.

[46] Yuen H.K., Princen J., Illingworth J., Kittler J.: „Comparative study of Hough Transform methods for circle finding".