

**Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie**

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki
Katedra Automatyki



PRACA MAGISTERSKA

SEBASTIAN WOJAS

**Metody przetwarzania obrazów z
wykorzystaniem biblioteki OpenCV**

**PROMOTOR:
DR PAWEŁ ROTTER**

KRAKÓW 2010

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA
POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ
WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE I ŻE NIE KORZYSTAŁEM
ZE ŹRÓDEŁ INNYCH NIŻ WYMIENIONE W PRACY.

PODPIS

AGH University of Science and Technology

Faculty of Electrical Engineering, Automatics, Computer Science and
Electronics



MSc THESIS

SEBASTIAN WOJAS

Practical applications of image analysis using OpenCV

PROMOTOR:
DR PAWEŁ ROTTER

KRAKÓW 2010

PODZIĘKOWANIA

*Z GŁĘBI SERCA DZIĘKUJE SWOIM RODZICOM
DANUCIE I ANDRZEJOWI WOJAS ZA TRUD JAKI
WŁOŻYLI W MOJE WYCHOWANIE ORAZ ZA
MĄDROŚĆ, DZIĘKI KTÓREJ ŻYCIE STAŁO SIĘ
ŁATWIEJSZE.*

*PROMOTOROWI **DOKTOROWI PAWŁOWI
ROTTEROWI** ZA POMOC W PODJĘCIU
INTERESUJĄCEJ MNIE TEMATYKI PRACY ORAZ
CZAS POŚWIECONY NA DYSKUSJE NAD JEJ
ZAGADNIENIAMI.*

Spis Treści

1. Wstęp	8
1.1 Cele i założenia	8
1.2 Zawartość pracy	8
1.3 Wstęp do cyfrowego przetwarzania obrazów	9
1.4 Wstęp do biblioteki OpenCV	9
1.4.1 Rozwój projektu i zgodność	10
1.4.2 Pobieranie i instalacja	11
1.4.3 Dokumentacja	14
1.4.4 Architektura OpenCV	14
2. Podstawy OpenCV	15
2.1 Typy danych	15
2.1.1 Statyczne struktury danych	15
2.1.2 Dynamiczne struktury danych	18
2.1.3 Pomocnicze struktury danych	21
2.2 Praca z obrazami	21
2.2.1 Tworzenie, kopiowanie i usuwanie obrazów	21
2.2.2 Przestrzeń barw i ich konwersja	22
2.3 Tworzenie interfejsu użytkownika	25
2.3.1 Obsługa okien	25
2.3.2 Wczytywanie, wyświetlanie i zapisywanie obrazów	26
2.3.3 Odczytywanie i zapisywanie danych z kamery oraz plików wideo	27
2.3.4 Obsługa myszy	30
2.3.5 Dodawanie i wykorzystywanie suwaków	32
2.4 Operacje na macierzach i obrazach	33
2.4.1 Dostęp do pikseli obrazów	33
2.4.2 Dostęp do fragmentów obrazu	34
2.4.3 Obszar ROI i COI	35
2.4.4 Operacje macierzowe	36
2.4.5 Rysowanie kształtów	37
2.4.6 Wstawianie tekstu	39
3. Przetwarzanie obrazów	40
3.1 Progowanie	40

3.2	Przekształcenia kontekstowe i konwolucja.....	43
3.3	Filtry do detekcji krawędzi	44
3.3.1	Filtr Sobela.....	46
3.3.2	Filtr Laplace'a	48
3.3.3	Filtr Canny'go	49
3.4	Kontury.....	51
3.4.1	Kontury i ich interpretacja	51
3.4.2	Operacje na konturach.....	53
3.4.3	Kod łańcuchowy Freemana	54
3.4.4	Rysowanie konturów.....	55
3.5	Rozmycie	56
3.6	Transformata Hougha	59
3.6.1	Wykrywanie linii.....	59
3.6.2	Wykrywanie okręgów.....	62
3.7	Przekształcenia morfologiczne	65
3.8	Histogramy	70
3.9	Analiza ruchu.....	76
3.9.1	Metody śledzenia ruchu.....	78
3.9.2	Opis i implementacja wybranych metod	79
3.10	Śledzenie obiektów	84
4.	Realizacja praktyczna.....	87
4.1	Finger Drawing	87
4.1.1	Realizacja.....	87
4.1.2	Problemy	88
4.1.3	Opis działania programu	88
4.1.4	Obsługa programu.....	92
4.1.5	Możliwość rozwoju.....	94
4.2	Face Detect.....	95
4.2.1	Realizacja.....	95
4.2.2	Problemy	95
4.2.3	Opis działania programu	95
4.2.4	Możliwości rozwoju.....	100
4.3	Line Assist.....	101
4.3.1	Realizacja.....	101

4.3.2	Problemy	101
4.3.3	Opis działania programu	102
4.3.4	Możliwości rozwoju.....	104
5.	Podsumowanie.....	105
	Literatura	106
	Spis Rysunków	108
	Dodatek. Opis plików programów	111
	Program Finger Draw	111
	Program Face Detect.....	111
	Program Line Assist	111

1. Wstęp

1.1 Cele i założenia

Głównym celem tego opracowania jest ukazanie możliwości jakie daje biblioteka OpenCV w procesie tworzenia systemów detekcji i analizy obrazu. Na podstawie zagadnień zawartych w pracy zostały stworzone programy pokazujące funkcjonalność biblioteki pod względem analizy ruchu, detekcji obiektów oraz przetwarzania sekwencji wideo.

1.2 Zawartość pracy

Niniejsza praca składa się z pięciu rozdziałów.

Rozdział 1 opisuje główne cele pracy oraz jej tematykę. Stanowi również wstęp do cyfrowego przetwarzania obrazów (1.3). Prezentuje bibliotekę OpenCV (1.4) pod kątem historii jej rozwoju, wsparcia technicznego, architektury oraz sposobu instalacji na komputerze z zainstalowanym systemem Windows oraz oprogramowaniem Visual Studio C++ 2010 Express.

Rozdział 2 jest przedstawieniem podstawowych narzędzi biblioteki OpenCV, niezbędnych do analizy i przetwarzania obrazów. Zawiera między innymi opis typu danych (2.1), pracy z obrazami (2.2), tworzenia GUI (2.3) oraz podstawowych operacji na obrazach (2.4).

Rozdział 3 poświęcony jest w całości metodom przetwarzania obrazów, pod kątem teoretycznym oraz ich implementacji z wykorzystaniem funkcjonalności OpenCV. Opisuje metody progowania (3.1), detekcji krawędzi (3.3), wykrywania konturów (3.4) Zawiera operacje rozmycia (3.5), przekształcenia morfologiczne (3.6), wykorzystanie transformaty Hougha (3.7) oraz histogramów (3.8). Rozdział ten jest również swoistym wprowadzeniem do analizy obrazów pod kątem ruchu (3.9) oraz śledzenia obiektów (3.10). Większość z wymienionych zagadnień w celu lepszego zrozumienia zasady działania, oparta jest o realizacje programowe pokazujące ich możliwości.

Rozdział 4 jest częścią programową, wykorzystującą poznane we wcześniejszych rozdziałach techniki analizy obrazów do tworzenia rozbudowanych systemów wizyjnych. Przedstawia interaktywny interfejs *Finger Draw* wykorzystywany do rysowania obiektów geometrycznych (4.1). Opisuje również implementację systemu detekcji twarzy *Face Detect* z wykorzystaniem metody *Haar-Like* (4.2) oraz systemu analizy drogi *Line Assist* (4.3).

Rozdział 5 jest podsumowaniem zrealizowanych zagadnień pod kątem funkcjonalnym oraz rozwojowym.

W **Dodatk** znajduje się kod oprogramowania użyty do stworzenia systemów opisanych w rozdziale 4.

1.3 Wstęp do cyfrowego przetwarzania obrazów

Każdego dnia dokonujemy decyzji na podstawie analizy informacji odbieranych przez narządy zmysłów. Najwięcej danych, bo aż 90% dostarczają oczy, dlatego też głównie na podstawie tego, co widzimy podejmujemy odpowiednie działania. W celu zautomatyzowania stanowiska pracy zajmowanego przez człowieka, należy budować urządzenia bazujące na wizji maszynowej – automatyczne systemy wizyjne, będące elektronicznym odpowiednikiem wzroku.

Faktem jest, iż efektywność wykorzystywania wspomnianych systemów rośnie z dnia na dzień. Co więcej, coraz więcej gałęzi nauki czerpie z dużych możliwości jakie daje przetwarzanie obrazów. Znanymi od wielu lat dziedzinami wykorzystującymi analizę obrazu są:

- Medycyna – analiza obrazów biomedycznych (np. radiografia, rezonans magnetyczny, tomografia komputerowa, ultrasonografia, morfologia, mammografia cyfrowa. Za pomocą przetwarzania obrazów zaobserwowano m.in. 62% wzrost skuteczności diagnozy płodowych wad rozwojowych i ginekologii, 28% wzrost ilości informacji diagnostycznej przy badaniach wątroby i przewodu żółciowego),
- Astronomia,
- Robotyka,
- Radiologia,
- Metrologia,
- Sejsmologia,
- Metalurgia,
- Systemy zabezpieczeń i wiele innych.

1.4 Wstęp do biblioteki OpenCV

OpenCV (*Open Source Computer Vision library*) jest bezpłatną, open-source'ową biblioteką do użytku komercyjnego i edukacyjnego, bez konieczności udostępniania swoich projektów opartych o jej implementację. Została napisana w języku C i C++ i rozwijana jest za pomocą *wrapperów* w wielu językach programowania takich jak C#, Python, Ruby, Matlab czy Java. Posiada wielu użytkowników na całym świecie – doskonałym przykładem jest grupa tematyczna o OpenCV na portalu Yahoo, która aktualnie posiada ponad 48.000 użytkowników.

Biblioteka ta została stworzona na potrzeby aplikacji czasu rzeczywistego gdzie wydajność obliczeniowa programów jest bardzo istotna. Napisana jest w zoptymalizowanym języku C/C++ i wykorzystuje możliwości jakie dają popularne od kilku lat procesory wielordzeniowe.

Jednym z głównych celów biblioteki OpenCV jest dostarczenie narzędzia, które pozwoli tworzyć zarówno proste programy jak również zaawansowane projekty ludziom z różnym poziomem wiedzy na temat programowania i przetwarzania obrazów. Amatorzy za pomocą OpenCV i kamery internetowej mogą poznawać elementy przetwarzania obrazów w zakresie: wykrywania krawędzi, segmentacji czy filtracji obrazów bez większego wysiłku. Zaś sami specjaliści nie muszą poświęcać czasu na pisanie bazowych funkcji w zaawansowanych

projektach – nigdy więcej wymyślania koła! Możliwe jest to dzięki wbudowanym w bibliotekę ponad 500 funkcjom, które obejmują wiele obszarów komputerowej wizji takich jak robotyka, stereowizja, bezpieczeństwo, analiza obrazów medycznych czy kalibracja kamery. Wiele projektów wykorzystujących zdjęcia z lotu ptaka czy mapowanie ulic takie jak – „Google Maps”, czy „Google Street’s View”, wykorzystuje kalibrację kamery oraz metody „zszywania” obrazów, które są częścią OpenCV. Bezzałogowe samoloty, urządzenia do detekcji obiektów czy układy monitoringu również oparte są o wizję komputerową.

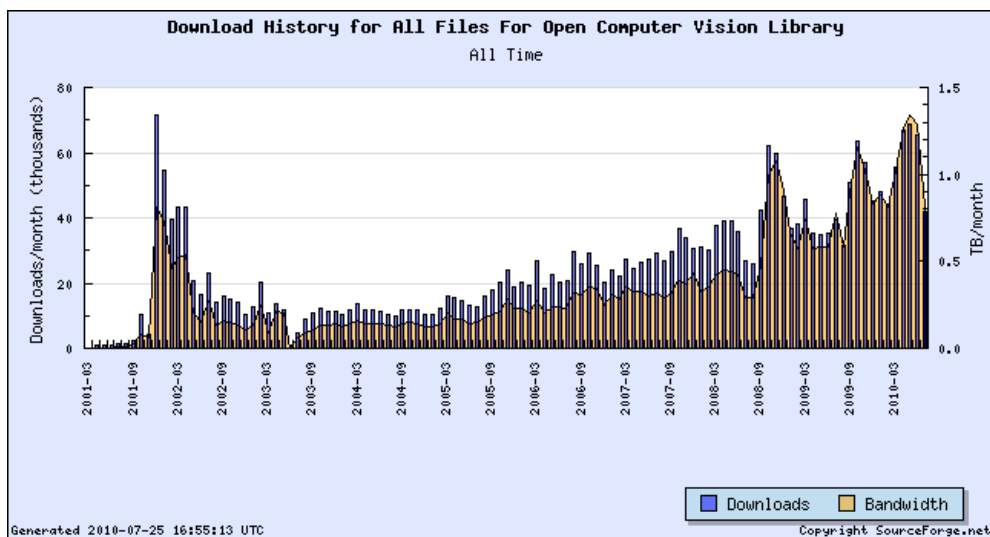
Inspiracją do zgłębienia tematyki OpenCV był artykuł „Innovations from a robot rally” w SCIENTIFIC AMERICAN[1] o wyścigu „Grand Challenge” przeznaczony dla autonomicznych pojazdów. Wyścig odbył się w październiku roku 2005, a organizatorem była amerykańska agencja DARPA, zajmująca się rozwojem technologii wojskowej. Samochód „Stanley” – Volkswagen Touareg przejechał samodzielnie trasę 212 km po pustyni Mojave oraz południu Las Vegas w czasie 6 godzin i 54 minut. Jedną z głównych ról przy projektowaniu systemu wizyjnego odegrała właśnie biblioteka OpenCV.

1.4.1 Rozwój projektu i zgodność

Początki OpenCV sięgają końca lat 90’. W tamtych latach korporacja Intel rozpoczęła kilka projektów opartych o prace w czasie rzeczywistym – takich jak śledzenie promieni oraz wyświetlanie obiektów 3D na ścianie. Jeden z autorów tych projektów wizytując w tym czasie amerykańskie uniwersytety zauważył, że kilka grup uniwersyteckich takich jak „MIT Media Lab” z MIT w Massachusetts posiada wiele rozbudowanych narzędzi do przetwarzania obrazów, które były propagowane i jednocześnie udoskonalane przez studentów. Właśnie przy współpracy tych grup oraz pomocy Intel’s Performance Library Team zostało stworzone jądro z zaimplementowanym kodem oraz specyfikacją algorytmów. Dane te zostały przesłane do Intel’s Russian Library Team w Niższym Nowogrodzie w Rosji – gdzie właśnie powstała biblioteka OpenCV.

Głównymi celami OpenCV jakie zostały założone w trakcie jej budowania było stworzenie darmowego, otwartego i zoptymalizowanego kodu dla podstawowych funkcji przetwarzania obrazów. Kod ten musiał być czytelny i łatwy do przenoszenia, gdyż był podstawą dla deweloperów, którzy mieli go rozwijać[W1].

Można by się zastanawiać dlaczego Intel stworzył bardzo pomocną bibliotekę dla deweloperów całkowicie za darmo. Idea była bardzo prosta – wykorzystanie tej biblioteki pomagało w projektowaniu aplikacji wymagających dużej mocy obliczeniowej, a tym samym szybkich procesorów. Sprzedaż komponentów firmy Intel była bardziej opłacalna niż wypuszczenie na rynek dodatkowego oprogramowania. Pomysł mógł być zrealizowany tylko przy dużej popularności tej biblioteki. Intel nie pomylił się z prognoząmi – do tej pory ze strony projektu[W2] biblioteka OpenCV została ściągnięta 2.858.950 razy, co stanowi 40.5 TB danych!



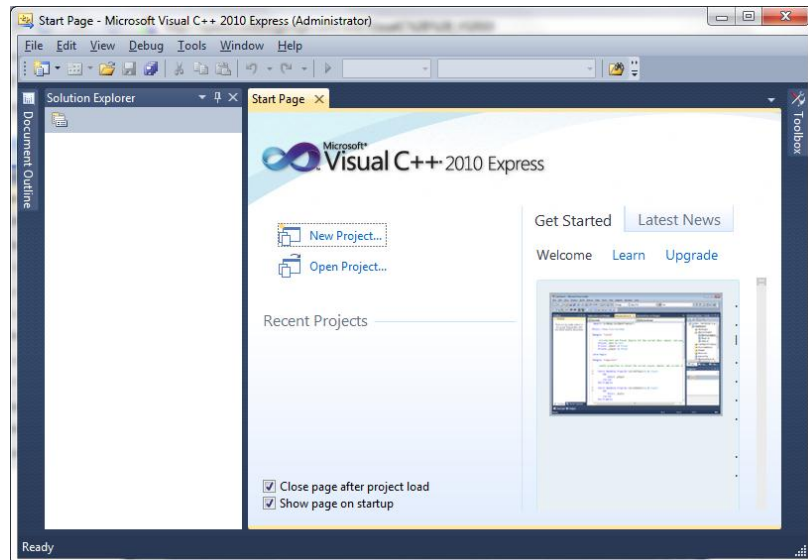
Rysunek 1 Statystyki pobrań biblioteki OpenCV ze strony projektu.

Pierwsza wersja „alpha” biblioteki została udostępniona w roku 2000. Następne, udoskonalone wersje to: „Beta 1” – 2001 (wsparcie dla Linuxa), „Beta 2” – 2002, „Beta 3” – 2003, „Beta 4” – 2004, „Beta 5” – 2005. Oficjalna wersja OpenCV o nazwie „1.0” powstała dopiero w październiku 2006 roku. Spora aktywność deweloperów w kierunku rozwijania projektu spowodowała powstawanie kolejnych wersji: „1.1pre1” – 2008, „2.0” – 2009. Od kwietnia 2010 roku dostępna jest wersja „2.1”. Z informacji zawartych na stronie projektu wynika, że wersja 2.2 ma pojawić się w październiku 2010 roku.

Biblioteka OpenCV przeznaczona jest do pracy z systemami Linux, MacOS oraz Windows pod kompilatorami Visual Studio 2005, 2008 (wraz z wersjami Express), MinGW – Windows oraz GCC 4.x – Linux i MacOS.

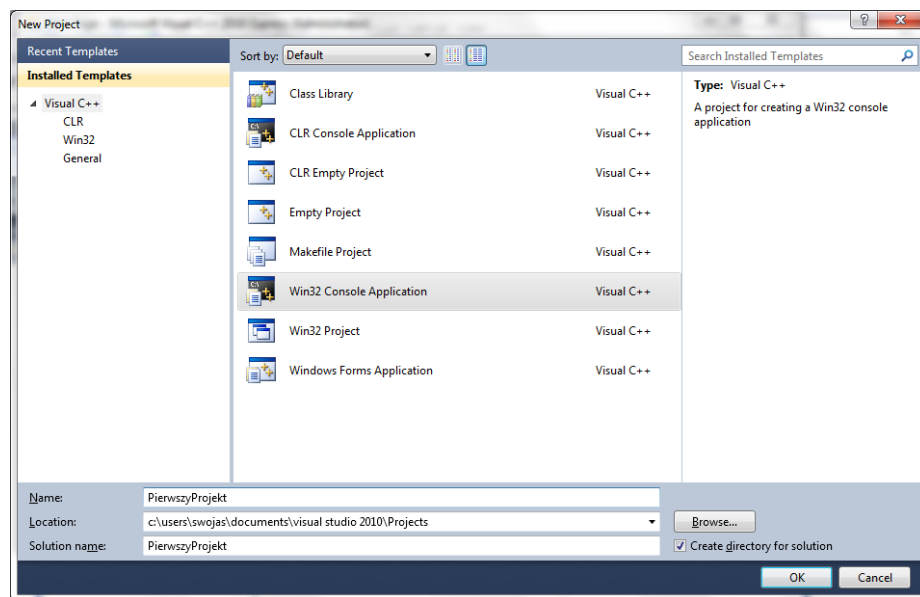
1.4.2 Pobieranie i instalacja

Aplikacje stworzone na potrzeby tej pracy projektowałem za pomocą aktualnej wersji OpenCV oraz środowiska Visual Studio C++ 2010 Express. Visual Studio Express jest wersją darmową, którą można pobrać ze strony[W3] w wersji webowej i później zainstalować z wykorzystaniem dostępu do Internetu. Jak już wspomniałem przy okazji statystyk, bibliotekę można pobrać ze strony projektu – najlepiej wersję wykonywalną (*OpenCV-2.1.0-win32-vs2008.exe*). Następnie instalujemy OpenCV otwierając ściągnięty plik. Wskazujemy miejsce docelowe instalacji biblioteki (domyślnie C:\OpenCV2.1) wraz z określeniem zmiennej systemowej PATH – informuje system, w którym miejscu znajdują się pliki dynamicznych bibliotek DLL (*Dinamic-Link Library*) OpenCV. Po zainstalowaniu możemy przejść do tworzenia nowego projektu. W tym celu otwieramy Visual Studio C++ Express z menu START. Następnie otwieramy nowy projekt:



Rysunek 2 Tworzenie nowego projektu w VS2010 Express

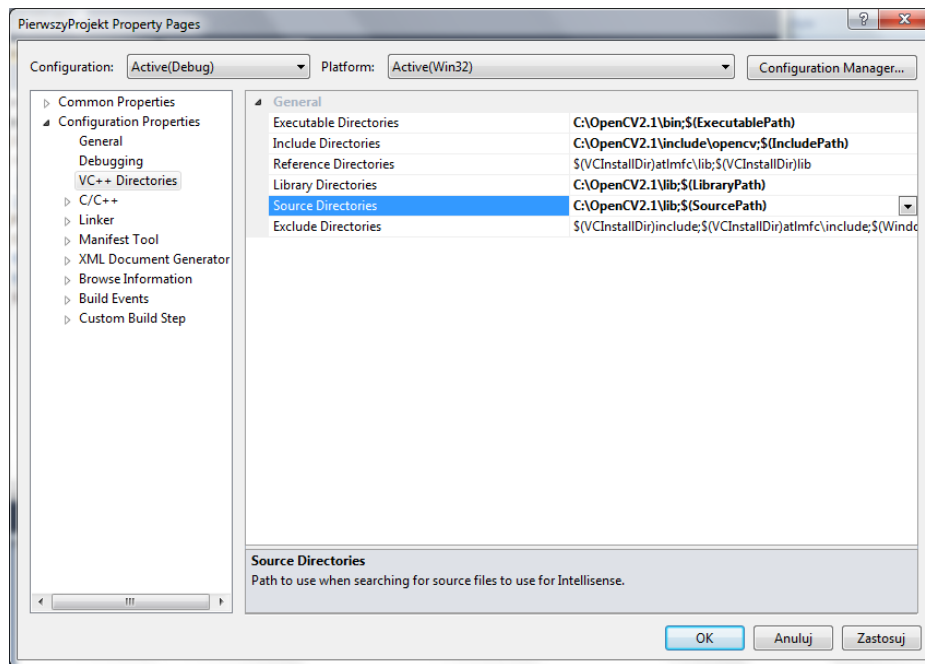
Klikamy na „Win32 Console Application” i nadajemy projektowi unikalną nazwę „PierwszyProjekt”.



Rysunek 3 Widok tworzenia nowego projektu

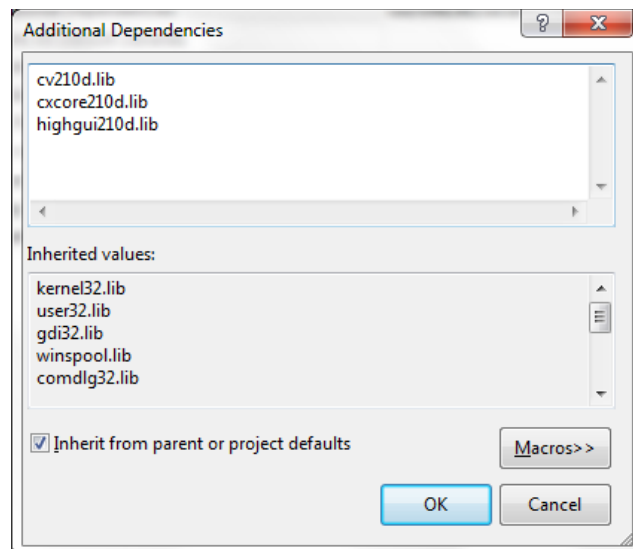
Następnie musimy zmienić ustawienia projektu *klikając Project-> Properties*, następnie *Configuration Properties -> VC++ Directories* dodając ścieżki dostępu dla:

- *Include Directories* dodajemy 'C:\OpenCV2.1\include\opencv',
- *Library Directories* dodajemy 'C:\OpenCV2.1\lib',
- *Source Directories* dodajemy 'C:\OpenCV2.1\src',
- *Executables Directories* dodajemy 'C:\OpenCV2.1\bin'.



Rysunek 4 Okno właściwości projektu

W pozycji *Linker->Input->Additional Dependencies* dodajemy 'cv210d.lib; cxcore210d.lib; highgui210d.lib'.



Rysunek 5 Okno dodawania dodatkowych ścieżek dostępu

Następnie zamykamy wszystkie okna i przechodzimy do widoku projektu. Należy pamiętać żeby na początku pliku projektu dodać cztery pliki nagłówkowe:

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

Dołączenie tych plików pozwala na dostęp do większości algorytmów zawartych w OpenCV.

1.4.3 Dokumentacja

Główna dokumentacja w formacie HTML oraz PDF znajduje się w podkatalogu *OpenCV2.1/doc*, który został stworzony przy instalacji biblioteki na naszym komputerze. Znajduje się tam opis biblioteki jak i podręcznik użytkownika opisujący całą zawartość biblioteki.

Bardzo pożytecznym źródłem wiedzy jest również strona Wiki OpenCV, kilkakrotnie przeze mnie przytoczona, na której znajdują się aktualne informacje. Wsparcie oraz pomoc przy projektowaniu, można znaleźć na wcześniej wspomnianym forum dyskusyjnym serwisu Yahoo.

1.4.4 Architektura OpenCV

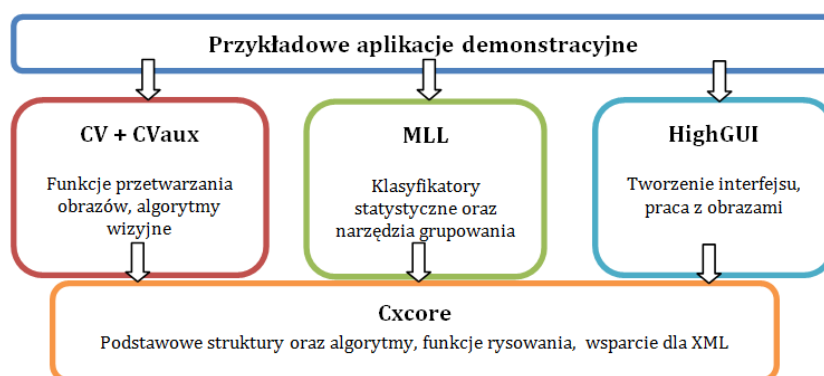
OpenCV składa się z pięciu podstawowych komponentów:

CV oraz CVaux – komponenty zawierające funkcję transformacji, filtracji oraz konwersji przestrzeni obrazów, funkcję analizy obrazów takie jak selekcja, operacje morfologiczne, detekcję krawędzi oraz obsługę histogramów, detekcję obiektów, kalibrację kamery, rekonstrukcję sceny 3D i inne,

MLL – Machine Learning Library, jak sama nazwa wskazuje zawiera funkcje tworzenia klasyfikatorów bazujących na statystyce odgrywających znaczącą rolę w uczeniu maszyn sposobu detekcji,

HighGUI – zawiera metody akwizycji i zwalniania obrazów, sekwencji wideo, narzędzia tworzenia interfejsu okienkowego, suwaków, obsługi myszy etc.

CxCore – Podstawowy komponent biblioteki zawiera operacje na tablicach, algebrę macierzy, funkcje matematyczne, transformacje Fouriera, wsparcie dla plików XML, narzędzia rysowania obiektów 2D i inne.



Rysunek 6 Architektura OpenCV

2. Podstawy OpenCV

2.1 Typy danych

Przed opisem typów danych musimy zdać sobie sprawę jak dane te są reprezentowane w pamięci komputera. Przyjęta konwencja matematycznego opisu obrazów zakłada, że obraz jest macierzą. Elementy macierzy, w związku z liniowym charakterem pamięci komputera są ciągiem liczb. OpenCV również przyjęło ten sposób reprezentacji. Obrazy poddawane algorytmom przetwarzania obrazów konwertowane są do skali szarości. Dlatego wartości macierzy przyjmują wartości od 0 do 255 dla skali szarości (reprezentacja odcieni szarości), lub od 0 do 1 dla postaci binarnej obrazu, gdzie zazwyczaj, 0 – kolor czarny, 1 – kolor biały). W przypadku obrazów kolorowych elementy macierzy przyjmują trzy wartości (w zależności od przestrzeni kolorów), wyjątkiem jest model przestrzeni RGBA, który posiada czwarty parametr – kanał „alpha”, pozwalający na uzyskanie efektu przeźroczystości.

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix},$$

Którą zapisujemy jako ciąg liczb $p_{11}, p_{12}, p_{13}, p_{21}, p_{22}, p_{23}, p_{31}, p_{32}, p_{33}$ – elementy macierzy zapisywane wierszami od lewej do prawej. Dostęp do elementów macierzy obrazuje następujący wzór matematyczny $p_{xy} = P[y \cdot w + x]$, gdzie w – szerokość obrazu (macierzy).

Należy również wspomnieć o pewnych konwencjach przyjętych w bibliotece OpenCV, ułatwi to przyswajanie informacji w dalszej części pracy. Twórcy biblioteki przyjęli określony sposób nazewnictwa, mianowicie:

- struktury oraz klasy zaczynają się od prefiksu „Cv” i są pisane metodą łączenia słów, gdzie każde pisane jest od dużej litery (tzw. „CamelCase”), np. `cvCreateImage` (tworzenie obrazu),
- funkcje, które współpracują z powyższymi klasami i strukturami zaczynają się od prefiksu „cv” i również są pisane systemem „CamelCase”.

2.1.1 Statyczne struktury danych

Biblioteka zawiera trzy podstawowe statyczne struktury danych. Są to `IplImage`, `CvMat` oraz `CvArr`. Kolejność wymienienia tych struktur nie jest przypadkowa. OpenCV pisana jest w C, ale ma charakter obiektowy, dlatego gdyby powyższe struktury były klasami, a nie klasycznymi strukturami, można byłoby mówić tutaj o schemacie dziedziczenia – `IplImage` dziedziczy od `CvMat`, `CvMat` zaś od `CvArr`. W programowaniu proceduralnym możemy powiedzieć, że struktury te są strukturami pochodnymi.

Struktura `CvMat` opisuje macierze dwuwymiarowe. Analizując temat macierzy w OpenCV należy zdać sobie sprawę z faktu, iż są one bardziej abstrakcyjne niż macierze, które wykorzystywane są w matematycznej algebrze liniowej. Elementy macierzy w OpenCV reprezentowane są przez dwa parametry opisujące głębokość elementu oraz jego przestrzeń

kolorów, tzn. że mogą przyjmować one kilka wartości. Ogólny schemat typu elementów macierzy wygląda następująco: CV_<bit_depth>(S/U/F)C<channel_num>, tzn. typ CV_32FC3 opisuje elementy będące trójkami liczb całkowitych bez znaku. (czyli piksele o głębi 32 bitów w przestrzeni np. RGB).

Konstrukcja struktury nie jest zbyt rozbudowana posiada takie parametry jak szerokość, wysokość, typ elementów macierzy, długość wiersza wyrażoną w bitach oraz wskaźnik do tablicy typów danych. Poniżej przedstawiona jest pełna definicja struktury:

```
typedef struct CvMat {
    int type; // typ elementów macierzy
    int step; // długość wiersza macierzy
    int* refcount;
    union {
        uchar* ptr;
        short* s;
        int* i;
        float* fl;
        double* db;
    } data; // wskaźniki do danych
    union {
        int rows;
        int height;
    };
    union {
        int cols; //ilość wierszy
        int width; //ilość kolumn
    };
} CvMat;
```

Macierze można tworzyć na kilka sposobów. Najpopularniejszym sposobem jest wykorzystanie funkcji cvCreateMat(). Pełna jej deklaracja wygląda następująco:

```
CvMat* cvCreateMat( int rows, int cols , int type)
Argumentami funkcji są wymiary macierzy oraz typ danych elementów w niej zawartych.
```

Wspomnianą funkcję można zastąpić dwiema bardziej bazowymi funkcjami cvCreateMatHeader() oraz cvCreateData(). Pierwsza z tych dwóch funkcji tworzy wyłącznie nagłówki macierzy, bez alokacji pamięci, druga zaś zajmuje się przydzieleniem pamięci na dane macierzy:

```
CvMat* mat = cvCreateMatHeader( int rows,int cols, int type );
cvCreateData( mat );
```

Wspomniane funkcje bazowe używane są również niezależnie, na przykład w momencie kiedy chcemy stworzyć macierz, ale nie chcemy jeszcze jej przydzielać pamięci, lub w momencie gdy pamięć jest przydzielona ale z innego źródła. Jednokładną kopię macierzy możemy utworzyć za pomocą funkcji cvCloneMat(CvMat*) Gdy macierz, która została stworzona nie jest już więcej potrzebna, wystarczy użyć funkcji cvReleaseMat(CvMat*). Wszystkie wspomniane funkcje zwracają wskaźnik do utworzonej macierzy.

Dostęp do elementów macierzy odbywa się również na kilka sposobów. W tej pracy zostanie umieszczony tylko podstawowy, najczęściej używany gdyż jest on najprostszy w implementacji i zupełnie wystarczający do szerokiej gamy zastosowań. Wykorzystuję on wbudowane w OpenCV makro `CV_MAT_ELEM()`, które ze wskazanej macierzy pobiera wartość interesującego nas elementu.

```
CvMat* mat = cvCreateMat (3, 3, CV_8UC1);  
int elem22 = CV_MAT_ELEM(*mat, int, 2, 2);
```

Wynikiem tej operacji będzie przypisanie do zmiennej `elem22` wartości wskazanego elementu macierzy.

Następnym podstawowym formatem jest `IplImage`, który wykorzystywany jest do obsługi obrazów. Format ten jest rozbudowaną wersją `CvMat`. Dzięki `IplImage` macierze interpretowane są jako obrazy. Struktura ta pochodzi z innej biblioteki stworzonej przez firmę Intel, a mianowicie IPL (Image Processing Library). Format ten jest quasi-standardem w przetwarzaniu obrazów, dlatego zostanie opisany bardzo dokładnie. Struktura `IplImage` wygląda następująco:

```
typedef struct _IplImage {  
    int nSize;  
    int ID;  
    int nChannels;  
    int alphaChannel;  
    int depth;  
    char colorModel[4];  
    char channelSeq[4];  
    int dataOrder;  
    int origin;  
    int align;  
    int width;  
    int height;  
    struct _IplROI *roi;  
    struct _IplImage *maskROI;  
    void *imageId;  
    struct _IplTileInfo *tileInfo;  
    int imageSize;  
    char *imageData;  
    int widthStep;  
    int BorderMode[4];  
    int BorderConst[4];  
    char *imageDataOrigin;  
} IplImage;
```

Poniżej przedstawiony jest opis parametrów:

- `nSize` – rozmiar obrazu,
- `ID` – wersja,
- `nChannels` – liczba kanałów, standardowo określa przestrzeń kolorów. Przyjmuje wartości od 0 (obrazy czarno-białe) do 4 (obrazy w przestrzeni RGB + alpha)
- `alphaChannel` – parametr ignorowany w OpenCV,

- `depth` – tzw. głębia, czyli ilość bitów na piksel. Przyjmuje następujące wartości:
 - `IPL_DEPTH_8U(8S)`, 8-bitów bez znaku (ze znakiem),
 - `IPL_DEPTH_16U(16S)`, 16-bitów bez znaku (ze znakiem),
 - `IPL_DEPTH_32S`, 32-bitów ze znakiem,
 - `IPL_DEPTH_32F`, 32-bitowa liczba zmiennoprzecinkowa,
 - `IPL_DEPTH_64F`, 64-bitowa liczba zmiennoprzecinkowa.
- `colorModel[4]`, `channelSeq[4]` – parametry ignorowane przez OpenCV,
- `dataOrder` – określa sposób rozmieszczenia kanałów przestrzeni obrazów, oddzielnie (wartość 1), lub razem (wartość 0),
- `origin` – wskazuje kierunek zapamiętywania wierszy pikseli obrazu, od górnego wiersza (wartość `IPL_ORIGIN_TL`) lub od dolnego wiersza (wartość `IPL_ORIGIN_BL`),
- `width` – szerokość obrazu,
- `height` – wysokość obrazu,
- `*roi` – wskaźnik na strukturę `Ip1ROI`¹. Struktura ta zawiera prostokątny obszar, który będzie podlegał przetwarzaniu oraz obszar `COI`², który również będzie przetwarzany,
- `*maskROI`, `*imageId`, `*tileInfo` – parametry nie używane w OpenCV,
- `imageSize` – rozmiar obrazu w bajtach,
- `*imageData` – wskaźnik na pierwszy wiersz w obrazie,
- `widthStep` – ilość bitów w jednej linii obrazu, (w `CvMat` ten parametr nazwany był `step`)
- `BorderMode[4]`, `BorderConst[4]` – parametry ignorowane w OpenCV,
- `*imageDataOrigin` – parametr mało używany w OpenCV.

Typ `CvArr` można traktować jak klasę abstrakcyjną, wykorzystywany jest tylko jako parametr funkcji, który pozwala na wykorzystanie tej funkcji przy pracy z obiektami innych typów (np. `CvMat` lub `Ip1Image`).

2.1.2 Dynamiczne struktury danych

Wszystkie wcześniej omawiane typy danych były strukturami statycznymi, tak więc należy teraz wspomnieć o jednej z podstawowych i bardzo często wykorzystywanych dynamicznych struktur danych, jaką jest `CvSeq`. Obiektem tego typu są sekwencje, które mogą być alokowane w bloku pamięci (ang. memory storage), co stanowi jedną z metod obsługi alokacji pamięci dla dynamicznych obiektów.

Bloki pamięci tworzą listę w taki sposób, iż pozwala to na alokację i dealokację pamięci dla ciągłych zestawów bloków. Wiele funkcji zaimplementowanych w OpenCV wymaga tego

¹ ROI – ang. Region of Interest – obszar zainteresowania

² COI – ang. Channel of interest - kanał zainteresowania

sposobu przydzielania pamięci, dlatego bloki pamięci są naprawdę często wykorzystywanymi obiektami biblioteki. Bufor pamięć alokowany jest za pomocą funkcji:

```
void* cvMemStorageAlloc( CvMemStorage* storage, size_t_size),
```

gdzie parametrami są: *storage – wskaźnik na blok pamięci oraz t_size, czyli rozmiar buforu, który nie może być większy niż wskazany blok pamięci. Sam blok pamięci tworzy jest za pomocą funkcji:

```
CvMemStorage* cvCreateMemStorage(int blocksize),
```

gdzie parametrem jest rozmiar bloku. Domyślnie wartość blocksize równa jest zero, w tym przypadku alokowane zostaje 64kB pamięci. Funkcja ta zwraca wskaźnik na początek stworzonego bloku. W momencie gdy przydział pamięci nie jest już potrzebny, zwalniamy go za pomocą funkcji:

```
void cvReleaseMemStorage(CvMemStorage** storage)
```

oraz czyścimy za pomocą

```
void cvClearMemStorage(CvMemStorage** storage)
```

gdzie parametrem jest wskaźnik na interesujący nas blok.

Bloki pamięci, w zależności od rozmiaru sekwencji są zapełniane na dwa sposoby. Jeżeli danych sekwencji jest więcej niż jest w stanie pomieścić jeden blok pamięci, to dzielona zostaje na kilka bloków, które nazywane są blokami sekwencji. W przypadku gdy sekwencje potrzebują mało pamięci, to w jednym bloku pamięci można zmieścić kilka takich sekwencji. Sekwencje zaimplementowane są w oparciu o kolejkę dwustronną (ang. deque, double-ended queue), dlatego posiada funkcję które bardzo szybko dodają lub usuwają elementy sekwencji na jej początku lub końcu. W przypadku działań na elementach wewnątrz sekwencji, funkcję je wykonywujące działają dość wolno, gdyż są to operacje o dużej złożoności obliczeniowej.

```
typedef struct CvSeq {
    int flags;
    int header_size;
    struct CvSeq* h_prev;
    struct CvSeq* h_next;
    struct CvSeq* v_prev;
    struct CvSeq* v_next;
    int total;
    int elem_size;
    char* block_max;
    char* ptr;
    int delta_elems;
    CvMemStorage* storage;
    CvSeqBlock* free_blocks;
    CvSeqBlock* first;
}
```

Najważniejszymi parametrami tej struktury są:

- `flags` – flaga określająca rodzaj tworzonej sekwencji,
- wskaźniki `h_prev`, `h_next`, `v_prev`, `v_next` – wskaźniki te nie służą do wskazywania elementów wewnątrz sekwencji, ale do łączenia różnych sekwencji pomiędzy sobą w bardziej złożone struktury takie jak listy, grafy czy drzewa,
- `total` – liczba punktów lub obiektów w sekwencji,
- `header_size` – rozmiar nagłówka sekwencji,
- `elem_size` – rozmiar pojedynczego elementu sekwencji,
- `*ptr` – wskaźnik na aktualne miejsce zapisu danych,
- `*storage` – wskaźnik na początek sekwencji, czyli jego miejsce w pamięci,
- `*free_first` – wskaźnik na pierwszy nie zajęty blok,
- `*first` – wskaźnik na pierwszy blok sekwencji.

Sekwencje możemy tworzyć za pomocą funkcji alokującej:

```
CvSeq* cvCreateSeq(int seq_flags,  
int header_size, int elem_size, CvMemStorage* storage),
```

Parametry tej funkcji zostały opisane powyżej. Jeżeli chcemy modyfikować sekwencję przez dodanie lub usunięcie elementów korzystamy z następujących funkcji³:

```
cvSeqPush() – dodawanie elementu do końca sekwencji,  
cvSeqPop() – usuwanie elementu z końca sekwencji,  
cvSeqPushFront() – dodawanie elementu na początku sekwencji,  
cvSeqPopFront() – usuwanie elementu z początku sekwencji,  
cvSeqInsert() – dodaje elementy do wnętrza sekwencji,  
cvSeqRemove() – usuwa elementy z wnętrza sekwencji.
```

Jeżeli chcemy usunąć sekwencję używamy funkcję `cvClearSeq()`. Należy jednak pamiętać, że w momencie usuwania sekwencji, nie zostanie zwolniona pamięć na nią przydzielona w bloku pamięci. Zaalokowana pamięć może być ponownie użyta tylko przez tą usuniętą sekwencję. Pamięć z bloku pamięci należy wyczyścić za pomocą wspomnianej wcześniej funkcji `cvClearMemStorage()`.

Istnieją jeszcze inne dynamiczne struktury danych takie jak: `CvSet`, `CvGraph`, `CvTreeNodeIterator` lecz nie będą one opisane, w związku z brakiem ich wykorzystania w moich projektach. Szczegółowe informacje na ich temat można znaleźć w dokumentacji OpenCV.

³ Nie zostaną opisane deklaracje funkcji, gdyż nie ma takiej potrzeby. Pełne deklaracje można znaleźć w dokumentacji OpenCV

2.1.3 Pomocnicze struktury danych

OpenCV zawiera kilka pomocniczych typów danych. Wykorzystywane są często do określania parametrów funkcji:

- `CvPoint` – podstawowy struktura określająca współrzędne punktu. Do ich określania wykorzystuje się funkcję `cvPoint()` gdy współrzędne są liczbami całkowitymi, oraz `cvPoint2D32f(x,y)` dla współrzędnych będących liczbami zmiennoprzecinkowymi. Dla punktów określanych w przestrzeni trójwymiarowej wykorzystuje się funkcję `cvPoint3D(x,y,z)` np.:

```
CvPoint pt = cvPoint(0,0); //cvPoint(int x, int y)
```

`CvSize` – struktura określająca wymiary obiektu, posiada dwa argumenty szerokość oraz wysokość. Parametry te określane są za pomocą funkcji `cvSize()` np.:

```
CvSize size = cvSize(80,40); //cvSize(int width, height)
```

Zmienna `size` typu `CvSize` przyjmuje parametry szerokość 80 pikseli, wysokość 40 pikseli.

- `CvRect` – struktura za pomocą, której tworzy się prostokąty, posiada cztery argumenty, współrzędną `x`, `y` – lewego górnego rogu oraz wysokość i szerokość. Parametry te określane są za pomocą funkcji `cvRect()` np.:

```
CvRect rect = cvRect(10,10,80,40); //cvRect(int x,y,width,height)
```

Zmienna `rect` typu `CvRect` jest prostokątem „zaczepionym” w punkcie `[10,10]` o wymiarach szerokość 80 pikseli, wysokość 40 pikseli,

- `CvScalar` – struktura ta pozwala przechowywać maksymalnie cztery wartości typu `double`. Jej parametrem jest wskaźnik do czteroelementowej tablicy elementów typu `double`. Parametry te określane są za pomocą funkcji tworzącej `cvScalar` np.:

```
CvScalar skalar = cvScalar(1,2.1,3.1,4); //cvScalar(double val[4])
```

Zmienna `skalar` typu `CvScalar` jest skalarą o wartościach `(1,2.1,3.1,4)`.

2.2 Praca z obrazami

2.2.1 Tworzenie, kopiowanie i usuwanie obrazów

Tworzenie obrazów odbywa się za pomocą jednej z najczęściej używanych funkcji z biblioteki OpenCV, a mianowicie `cvCreateImage()`. Podczas wywołania tej funkcji zostaje stworzony nagłówek i przydzielona zostaje pamięć na dane za pomocą wskaźnika `*imageData`:

```
IplImage* cvCreateImage( cvSize size, int depth, int channels),
```

gdzie parametrami tej funkcji są: rozmiary obrazu, podawane za pomocą zmiennej `size` (szerokość i wysokość), głębokość pikseli obrazu oraz wybór przestrzeni kolorów. Należy pamiętać,

że w przypadku przestrzeni RGB, trójki wartości piksela są podawane w kolejności (b0, g0, r0), (b1, g1, r1) itd., a nie w kolejności na jaką wskazywała by nazwa RGB.

Podobnie jak w przypadku macierzy, operację tworzenia obrazu możemy wykonać dwuetapowo. Najpierw tworzymy nagłówek obrazu za pomocą funkcji `cvCreateImageHeader()`, której parametry są takie same jak `cvCreateImage()`. Następnie alokujemy pamięć za pomocą `cvCreateData()`.

Gdy przestajemy używać obrazu używamy funkcji `cvReleaseImage()`, która usuwa i zwalnia pamięć po obrazie:

```
void cvReleaseImage(IplImage** img),
```

gdzie `*img` jest wskaźnikiem do wskaźnika i określa obraz, który chcemy zwolnić. Jeżeli nie zaalokowaliśmy pamięci na obraz a jedynie stworzyliśmy dla niego nagłówek to możemy go usunąć za pomocą funkcji `cvReleaseImageHeader()`, o parametrach takich jak powyższa funkcja `cvReleaseImage()`.

W momencie kiedy na potrzeby programu potrzebujemy wierną kopię obrazu, na którym pracujemy wykorzystujemy funkcję `cvCloneImage()`, która zwraca kopię obrazu:

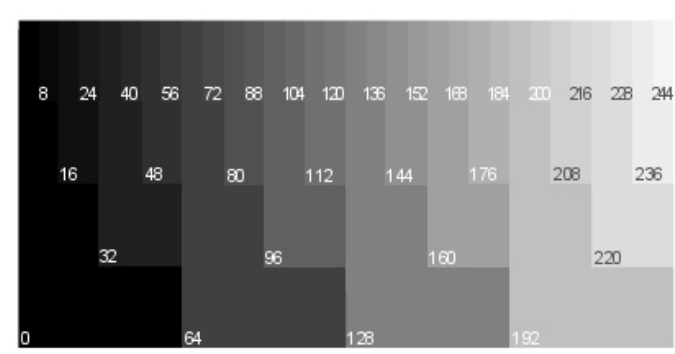
```
IplImage* cvCloneImage(const IplImage* img),
```

gdzie `*img` jest wskaźnikiem na kopiowany obraz.

2.2.2 Przestrzenie barw i ich konwersja

Przed omówieniem sposobów konwersji przestrzeni barw, zostaną przedstawione podstawowe jej rodzaje.

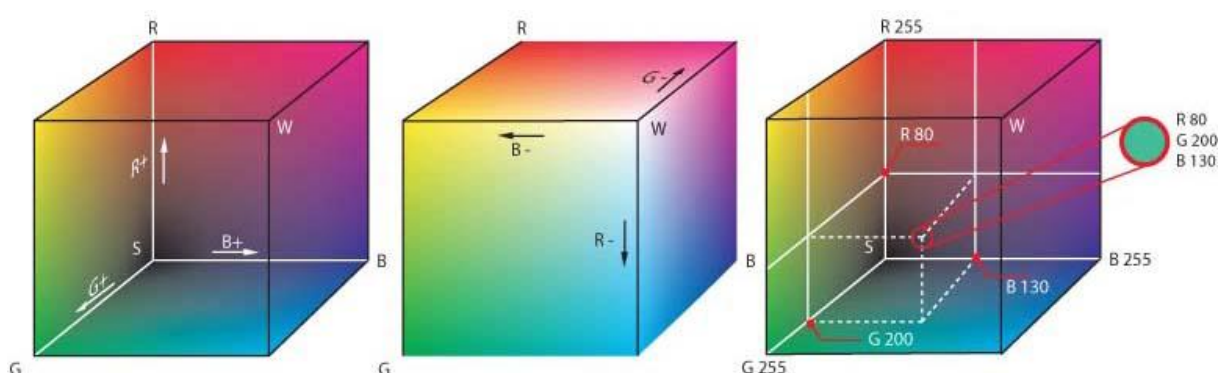
Podstawowy tryb koloru to skala szarości (ang. grey scale). Zawiera 256 tonów koloru, zaczynając od 0 – koloru czarnego, przez 254 odcienie szarości i kończąc na 255 – kolorze czarnym. Zawiera 8-bitową głębię kolorów. Obrazy w trybie skali szarości nazywane są monochromatycznymi.



Rysunek 7 Przekrój tonów skali szarości⁴

⁴ Źródło zdjęcia: [W4]

Nazwa modelu przestrzeni RGB powstał od pierwszych liter angielskich nazw kolorów, z których ta przestrzeń jest stworzona (Red-Green-Blue). Kolory te nie są przypadkowe i wynikają z możliwości odbiorczych oka ludzkiego. Wrażenie widzenia konkretnego koloru przez oko może powstać z odpowiedniego połączenia trzech wiązek światła o wspomnianych kolorach. Barwy pochodne powstają przez zastosowanie syntezy addytywnej barw podstawowych. Model RGB jest modelem teoretycznym. Jego odwzorowanie może być różne, zakres barw przestrzeni jest zależny od urządzenia. Najczęściej wykorzystuje się 24bitową reprezentację kolorów RGB, 8 bitów na każdy z kolorów podstawowych (zakres od 0-255). Przy wartości 0 wszystkich kolorów składowych modelu RGB piksel ma kolor czarny, zaś przy wartości 255 kolor biały [3].



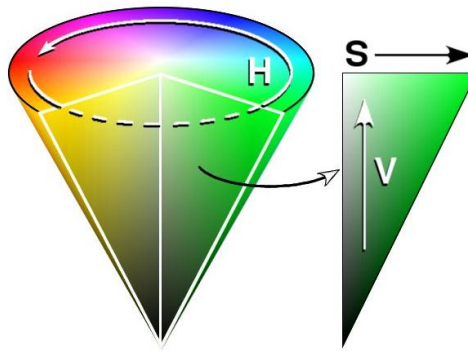
Rysunek 8 Model RGB przestrzeni kolorów.⁵

Pochodnym modelem koloru od RGB jest model RGBA (inaczej ARGB), który został wzbogacony o dodatkowy parametr alpha. Wartość alpha jest współczynnikiem pochłaniania światła, który odpowiada za przezroczystość piksela. Jeżeli jego wartość jest równa 0% to dany piksel jest całkowicie przezroczysty i przyjmuje kolor tła, które znajdują się pod nim. Dla wartości kanału alpha równej 100%, piksel przyjmuje taki kolor jak w standardowym modelu RGB.

Model HSV został stworzony z myślą jak najlepszego odzwierciedlenia sposobu widzenia ludzkiego oka. Model ten zakłada, że wszystkie barwy pochodzą od światła białego, gdzie część widma zostaje wchłonięta a część odbita od oświetlanych przedmiotów. Jego nazwa powstała od pierwszy liter angielskich nazw:

- barwy światła (ang. Hue), będącej kątem na kole barw podstawy stożka, z zakresem wartości od 0 do 360 stopni,
- nasycenia koloru (ang. Saturation), będącego promieniem koła barw podstawy stożka,
- mocy światła białego (ang. Value, znanej również jako Brightness), będącej wysokością stożka.

⁵ Źródło obrazu: [W5]



Rysunek 9 Model HSV przestrzeni kolorów.⁶

Środek podstawy (koła) odpowiada kolorowi białemu, wierzchołek zaś kolorowi czarnemu. Barwy pomiędzy tymi punktami stanowią tonacje przejścia pomiędzy kolorem białym i czarnym. Centrum barwy czerwonej odpowiada wartość kąta 0 lub 360 stopni, zielonej kąt 120 stopni, niebieskiej zaś 240 składowej Hue.

Istnieje jeszcze wiele innych modeli kolorów takie jak: YCrCb, XYZ, Lab, Luv, HLS, Bayer. Każdy z nich jest obsługiwane przez OpenCV.

Konwersja przestrzeni kolorów zostanie opisana w oparciu o podstawowe jej rodzaje, czyli: RGB->Grey (Grey->RGB) oraz RGB->HSV.

Pierwsza z konwersji jest stosunkowo prosta w implementacji. Wartość piksela przekonwertowanego z RGB do skali szarości obliczany jest na podstawie następującej operacji:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

W OpenCV do konwersji kolorów wykorzystywana jest funkcja `cvCvtColor()`:

```
void cvCvtColor( const CvArr* src, const Arr* dst, int code);
```

Pierwsze dwa argumenty to obraz źródłowy i przekonwertowany. Istotą tej funkcji jest trzeci argument `code` determinujący typ konwersji (instrukcja do biblioteki OpenCV zawiera opisy wszystkich możliwych rodzajów konwersji). W przypadku zamiany RGB na skalę szarości argument `code` powinien mieć wartość `CV_RGB2GREY`, w przypadku odwrotnym `CV_GREY2RGB`.

Konwersja z przestrzeni RGB do HSV stanowi trudniejsze zadanie. Wartości R, G oraz B muszą zostać przekonwertowane na typ zmiennoprzecinkowy na następnie przeskalowane do przedziału od 0 do 1. Należy pamiętać, że wartość Hue jest z zakresu od 0 do 360, Saturation i Value od 0 do 1. Poniżej został przedstawiony algorytm konwersji wartości RGB do HSV:

$$V = \min(R, G, B) \quad S = \begin{cases} \frac{V - \min(H, S, V)}{V}, & \text{gdy } V \neq 0 \\ 0, & \text{gdy } V = 0 \end{cases}$$

⁶ Źródło obrazu: [W6]

$$H = \begin{cases} \frac{60(G - B)}{S}, & \text{gdy } V = R \\ 120 + \frac{60(B - R)}{S}, & \text{gdy } V = G \\ 240 + \frac{60(R - G)}{S}, & \text{gdy } V = B \end{cases}$$

W przypadku tej konwersji przy zastosowaniu funkcji `cvCvtColor()`, parametr `code` powinien mieć wartość `CV_RGB2HSV`.

Wszystkie pozostałe dostępne typy konwersji przy zastosowaniu powyższej funkcji są wymienione w podręczniku użytkownika OpenCV, dlatego nie zostaną tutaj przetoczone.

2.3 Tworzenie interfejsu użytkownika

Funkcje OpenCV, które pozwalają na współdziałanie z systemem operacyjnym, systemem plików, czy elementami sprzętowymi takimi jak kamera, są zebrane w bibliotece HighGUI (High-Level Graphical User Interface). Biblioteka ta pozwala na tworzenie okien, oraz wyświetlanie na nich obrazów, wczytanych dzięki możliwościom HighGUI w zakresie obsługi plików graficznych (obrazów i sekwencji wideo). Wspomaga również obsługę myszy, którą można wykorzystywać do obsługi przełączników oraz suwaków.

Tworzenie interfejsu użytkownika możemy podzielić na trzy części: sprzętową, obsługę plików oraz interfejsu użytkownika.

Pierwszą z nich jest komunikacja z urządzeniami takimi jak kamera. Biblioteka HighGUI dostarcza narzędzi, które w prosty sposób pozwalają na komunikację z kamerą, co zazwyczaj było zmartwieniem programistów wykorzystujących obrazy z kamery.

Drugą część stanowi obsługa plików graficznych. Skupia się ona głównie na załadowaniu oraz zapisaniu obrazów. Udostępnione narzędzia pozwalają na pracę zarówno z obrazami jak i sekwencjami wideo. Wykorzystywane funkcje wspierają wiele formatów graficznych wraz z ich kodowaniem i dekodowaniem, gdy tego wymagają.

Ostatnia część skupia się na właściwym interfejsie użytkownika (GUI). Wbudowane funkcje pozwalają na otwieranie zdjęć w stworzonych przez nie oknach. Ułatwiają modyfikację parametrów obrazów za pomocą funkcji tworzących suwaki, przełączniki oraz wspomagających obsługę myszy.

W poniższych podrozdziałach zostaną opisane operacje, które pozwalają na stworzenie rozbudowanego interfejsu użytkownika.

2.3.1 Obsługa okien

Podstawą interfejsu użytkownika są okna, które tworzone są za pomocą funkcji `cvNamedWindow()`. Parametrami tej funkcji są: wymagana, unikatowa nazwa, która wyświetlana

jest jako tytuł oraz flaga, określająca możliwości związane ze zmianą rozmiaru okna. Sama flaga `CV_WINDOW_AUTORESIZE` jest ustawiona domyślnie, dzięki czemu okno dopasowuje się automatycznie do rozmiaru obrazu wczytanego w tym oknie. Jeżeli użytkownik ma mieć możliwość regulacji rozmiaru okna, to flaga musi być ustawiona na 0. Pełna specyfikacja omawianej funkcji wygląda następująco:

```
int cvNamedWindow(const char* name,
int flags = CV_WINDOW_AUTOSIZE
);
```

W momencie gdy nie potrzebujemy już okna to zamykamy je używając funkcji `cvDestroyWindow()`. Parametrem tej funkcji jest nazwa okna, które zostało wcześniej stworzone:

```
void cvDestroyWindow (const char* name);
```

Rozmiar okna możemy zmienić za pomocą funkcji `cvResizeWindow()`, która poza nazwą okna, potrzebuje parametry nowej szerokości oraz wysokości okna wyrażonej w pikselach:

```
void cvResizeWindow (const char* name, int width, int height);
```

Można również zmienić położenie okna funkcją `cvMoveWindow()`, której parametrami są standardowo nazwa okna oraz współrzędne lewego i górnego rogu okna wyrażone w pikselach:

```
void cvMoveWindow (const char* name, int x, int y);
```

2.3.2 Wczytywanie, wyświetlanie i zapisywanie obrazów.

Przed wyświetleniem obrazu w oknie, należy zapoznać się ze sposobem wczytania zdjęcia z pliku. Za ten proces odpowiedzialna jest funkcja `cvLoadImage()`:

```
IplImage* cvLoadImage (const char* filename, int iscolor = 1),
```

gdzie parametrami są: `filename` nazwa pliku zawierającego obraz oraz `iscolor`, który określa kolor obrazu dla następujących wartości:

`iscolor > 0` (lub flaga `CV_LOAD_IMAGE_COLOR`) – obraz traktowany jest jak kolorowy trzykanałowy,

`iscolor = 0` (lub flaga `CV_LOAD_IMAGE_GRAYSCALE`) – obraz jest w odcieniach szarości,

`iscolor < 0` (lub flaga `CV_LOAD_IMAGE_ANYCOLOR`) – obraz jest traktowany w zależności od ilości kanałów zapisanych w pliku.

Funkcja zwraca w wyniku wskaźnik na zaalokowany obraz. Jeżeli plik nie istnieje funkcja zwraca pusty wskaźnik (`NULL`). W chwili obecnej biblioteka OpenCV wspiera następujące formaty plików: bitmapy BMP, DIB, pliki JPEG – JPEG, JPG, JPE, pliki PNG, PBM, PGM, PPM, SR, RAS, TIFF oraz TIF. Jeżeli chcemy załadować plik o innym formacie, należy wcześniej

w programie graficznym przeprowadzić konwersję do formatu obsługiwanego lub użyć funkcji `cvCvtColor()`.

W momencie gdy plik został wczytany i przetworzony, można go zapisać używając funkcji `cvSaveImage()`. Zapisywany obraz musi mieć format jednokanałowy 8-bitowy lub trzykanałowy w przestrzeni RGB, z zachowaniem kolejności kolorów niebieski (B), zielony (G), czerwony (R). Funkcja ma następującą strukturę:

```
int cvSaveImage( const char* filename, const CvArr* image ),
```

gdzie `filename` jest nazwą zapisywanego pliku (jego rozszerzenie świadczy o formacie zapisu) oraz wskaźnik `*image` wskazuje obraz do zapisu. W przypadku poprawnego zapisu pliku funkcja zwróci liczbę całkowitą 1, w przypadku niepowodzenia 0.

W momencie gdy obraz jest już załadowany, można go wyświetlić używając funkcji `cvShowImage()`, która przyjmuje następujące argumenty:

```
void cvShowImage(const char* name, const CvArr* image,
```

gdzie wskaźnik `*name` jest oknem, na którym ma zostać wyświetlony obraz, który jest drugim parametrem – wskaźnikiem `*image`.

2.3.3 Odczytywanie i zapisywanie danych z kamery oraz plików wideo

Pierwsza część tego podrozdziału zostanie poświęcona obsłudze plików wideo, druga zaś funkcjom, które pozwalają na gromadzenie danych z kamery podłączonej do komputera.

Ładowanie plików wideo do OpenCV odbywa się za pomocą funkcji `cvCreateFileCapture()`, której parametrem jest interesujący programistę plik AVI lub MPG. Funkcja ta ma za zadanie załadowanie pliku i jego przygotowanie do odczytu. Poniżej deklaracja funkcji:

```
CvCapture* cvCreateFileCapture( const char* filename);
```

Przy poprawnym załadowaniu pliku funkcja zwraca wskaźnik do struktury `CvCapture` będącą analogią do `IplImage`, która zawiera informacje potrzebne do odczytania klatek z plików wideo lub kamery. Przy błędnym wczytaniu zostanie zwrócony wskaźnik `NULL`. Błąd może wynikać z braku pliku docelowego lub gdy plik wideo został skompresowany kodekiem, który nie jest obsługiwany przez bibliotekę.

Gdy plik został poprawnie załadowany można zacząć odczytywać z niego klatki. Można to zrobić na dwa sposoby.

Pierwszy z nich wykorzystuje funkcję `cvGrabFrame()`, która przyjmuje jako parametr wskaźnik na strukturę `CvCapture` i zwraca liczbę całkowitą 1 w przypadku powodzenia lub 0 w przypadku błędnego odczytania. Klatka zostaje zapisana do wewnętrznego bufora pamięci biblioteki, do której użytkownik nie ma dostępu, dzięki temu cała operacja pobrania ramki jest

bardzo szybka. Następnie korzystając z funkcji `cvRetrieveFrame()` ramka jest przetwarzana i zapisywana do również niedostępnego wewnętrznego bufora pamięci. Funkcja zwraca wskaźnik na `IplImage` wskazujący ten obraz. Należy pamiętać, że ta przetworzona ramka zostanie zastąpiona następną przy kolejnym wywołaniu funkcji `cvGrabFrame()`, dlatego jeżeli chce się korzystać z konkretnie tej ramki należy ją wcześniej skopiować w inne miejsce. Poniżej przedstawione są deklaracje opisanych funkcji:

```
int cvGrabFrame (CvCapture* capture);  
IplImage* cvRetrieveFrame (CvCapture* capture);
```

Drugi sposób jest zdecydowanie łatwiejszy. Wymaga on użycia funkcji `cvQueryFrame()`, która jest dwóch powyżej wymienionych funkcji i zwraca podobnie jak `cvRetrieveFrame()` wskaźnik na `IplImage`.

```
IplImage* cvQueryFrame (CvCapture* capture);
```

W momencie kiedy została zakończona praca z plikiem wideo należy zwolnić przydzieloną na niego pamięć za pomocą funkcji `cvReleaseCapture()`, której parametrem jest wskaźnik do wskaźnika `CvCapture`:

```
void cvReleaseCapture (CvCapture** capture);
```

Przechwytywanie obrazów z kamery jest niezbędne w programach, które mają działać w trybie rzeczywistym i na bieżąco mają analizować zachodzące zmiany na interesującym je obszarze. Proces ten odbywa się za pomocą funkcji `cvCreateCameraCapture()`:

```
CvCapture *capture = cvCreateCameraCapture (int index),
```

Jedynym argumentem funkcji jest `index`, będący numerem kamery w systemie operacyjnym. Parametr ten domyślnie ustawiony jest na wartość „0”, która oznacza, że w systemie istnieje tylko jedna kamera. Wartość ta jest inkrementowana, gdy do komputera podłączona jest więcej niż jedna kamera. Funkcja ta zwraca wskaźnik do struktury `CvCapture`. Klatki z sekwencji wideo nagrywanej przez kamerę zapisywane są za pomocą funkcji `cvQueryFrame()`, której jedynym parametrem jest wskaźnik na strukturę `CvCapture`. Funkcja ta nie musi oddzielnie alokować pamięci na sekwencję wideo, gdyż jest ona automatycznie przydzielona pamięci będącej częścią struktury `CvCapture`. Poniżej został zamieszczony kod, który obrazuje jak w poprawny sposób pobrać sekwencję wideo z kamery (w tym przypadku jedynej w systemie) oraz jak ją zapisać w pamięci:

```
CvCapture* cap = cvCreateCameraCapture( 0 );  
frame = cvQueryFrame( cap );
```

Powyżej zostały opisane sposoby odczytywania i przetwarzania obrazów. Ostatnia część tego podrozdziału będzie opisywać sposób w jaki zapisuje się pliki wideo. Podstawową rzeczą jest stworzenie „video writer’a” – jest to obiekt, który przechowuje wszystkie niezbędne parametry do zapisu sekwencji wideo:

```
CvVideoWriter* cvCreateVideoWriter(  
const char* filename,
```

```
int fourcc,
double fps,
CvSize frame_size,
int is_color = 1
);
```

Pierwszy parametr filename jest nazwą obiektu. Drugi fourcc odpowiada za określenie kodeka kompresji wideo, który zapisywany jest w unikalnej dla danego kodeka sekwencji czterobitowej (c0,c1,c2,c3). Następny parametr fps określa ilość klatek na sekundę w sekwencji wideo, frame_size długość zapisywanej sekwencji, is_color zaś określa czy sekwencja jest czarno-biała, czy kolorowa (domyślnie parametr ustawiony jest na 1, czyli sekwencję kolorową).

Następnie należy dokonać procesu zapisu sekwencji wykorzystując funkcję cvWriteFrame():

```
int cvWriteFrame(
CvVideoWriter* writer,
const IplImage* image
);
```

Jej argumentami jest tylko writer obiekt struktury CvVideoWriter oraz zapisywany obraz. Po dokonanych zapisie nie można zapomnieć o wywołaniu funkcji cvReleaseVideoWriter() w celu uwolnienia „video writera” oraz zamknięciu zapisywanego pliku wideo. Jeżeli wywołanie to zostanie pominięte to na końcu programu może dojść do przerwania zapisu i plik nie zostanie poprawnie zapisany.

Na koniec warto wspomnieć o przydatnych funkcjach cvSetCaptureProperties() i cvGetCaptureProperties(), które pozwalają na ustawienie i odczytanie wielu parametrów sekwencji wideo. Ich deklaracje wyglądają następująco:

```
int cvSetCaptureProperty(CvCapture* cap, int property_id, double value),
int cvGetCaptureProperty(CvCapture* cap, int property_id, double value),
```

Parametr *cap jest sekwencją wideo, a parametr property_id przyjmuje następujące wartości (w nawiasach są podane ich numeryczne odpowiedniki):

- CV_CAP_PROP_POS_MSEC (0) – pozycja w sekwencji wyrażona w milisekundach,
- CV_CAP_PROP_POS_FRAME (1) – pozycja w sekwencji wyrażona w klatkach,
- CV_CAP_PROP_POS_AVI_RATIO (2) – pozycja w sekwencji znormalizowana na przedziale od 0 do 1,
- CV_CAP_PROP_FRAME_WIDTH (3) – szerokość (rozdzielczość) sekwencji,
- CV_CAP_PROP_FRAME_HEIGHT (4) – wysokość (rozdzielczość) sekwencji,
- CV_CAP_PROP_FPS (5) – ilość klatek na 1 sekundę czasu,
- CV_CAP_PROP_FOURCC (6) – rodzaj kodeka zastosowanego do kompresji sekwencji,
- CV_CAP_PROP_FRAME_COUNT (7) – ograniczona ilość klatek w sekwencji.

Ostatni parametr value jest wartością ustawianą przez parametr property_id.

2.3.4 Obsługa myszy

Kolejną istotną częścią HighGUI jest obsługa myszy i klawiatury, które pozwalają na interakcję pomiędzy użytkownikiem a wyświetlanym w oknie obrazem. Komunikacja pomiędzy urządzeniami wskazującymi, a OpenCV nie jest prostym zadaniem gdyż oparta jest na standardowym mechanizmie wywołania zwrotnego. W celu ułatwienia czytelności, mechanizm zostanie opisany dla myszy.

Do detekcji zdarzenia pochodzącego od urządzenia należy najpierw napisać funkcję wywołania zwrotnego, która pozwoli OpenCV na reakcję po zdarzeniu od myszy (np. pojedyncze kliknięcie). Następnie należy zarejestrować wspomnianą funkcję, dzięki temu OpenCV będzie wiedziało, że powinna zostać wywołana przy określonym zdarzeniu ze strony myszy, we wskazanym oknie. OpenCV nie zawiera konkretnej funkcji, która reaguje na zdarzenia. Ta funkcja musi zostać napisana przez programistę, w taki sposób żeby zawierała odpowiedni zestaw parametrów i zwracała odpowiedni typ. Funkcja musi posiadać informację o rodzaju zdarzenia oraz miejscu jego zajścia. Następnie funkcja musi również wiedzieć, czy użytkownik w czasie zdarzenia wcisnął przycisk Alt, Shift lub Ctrl. Poniżej został przedstawiony prototyp funkcji spełniającej wszystkie wymagania:

```
void CvMouseCall (
    int event,
    int x, int y,
    int flags, void* param);
```

Przy zajściu zdarzenia OpenCV wypełnione zostaną argumenty funkcji. Pierwszy parametr event przyjmie jedną z poniższych wartości (ponownie, w nawiasach znajdują się odpowiedniki numeryczne):

- CV_EVENT_MOUSEMOVE (0)
- CV_EVENT_LBUTTONDOWN (1)
- CV_EVENT_RBUTTONDOWN (2)
- CV_EVENT_MBUTTONDOWN (3)
- CV_EVENT_LBUTTONUP (4)
- CV_EVENT_RBUTTONUP (5)
- CV_EVENT_MBUTTONUP (6)
- CV_EVENT_LBUTTONDBLCLK (7)
- CV_EVENT_RBUTTONDBLCLK (8)
- CV_EVENT_MBUTTONDBLCLK (9)

Powyższych wartości nie trzeba tłumaczyć, są to wszystkie ewentualności akcji ze strony myszy. Dwóm następnym parametrom x i y zostaną przypisane współrzędne piksela, w którym doszło do zdarzenia. Należy pamiętać, że są to współrzędne obrazu a nie okna, w którym ten obraz się znajduje jak piszę wiele z zachodnich książek o tematyce OpenCV.

Czwarty parametr jest flagą, która dostarcza informacji o przycisku, który został wciśnięty, czy był to lewy klawisz myszy, prawy, środkowy lub klawisz z klawiatury Ctrl, Shift lub Alt:

- CV_EVENT_FLAG_LBUTTON (1)
- CV_EVENT_FLAG_RBUTTON (2)

- CV_EVENT_FLAG_MBUTTON (4)
- CV_EVENT_FLAG_CTRLKEY (8)
- CV_EVENT_FLAG_SHIFTKEY (16)
- CV_EVENT_FLAG_ALTKEY (32)

Ostatni parametr jest przekazywany do funkcji podczas wywołania funkcji rejestrującej `cvSetMouseCallback()`, która zostanie opisana poniżej. Parametr ten pozwala na wykorzystanie funkcji obsługi zdarzeń dla wielu obrazów. Wystarczy tylko wcześniej rzutować wskaźnik typu `void` na wskaźnik na typ `IplImage`.

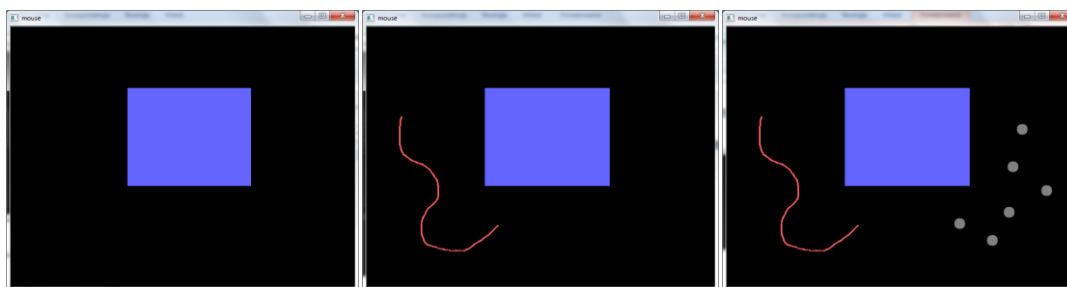
Gdy funkcja obsługi zdarzeń jest kompletna należy ją zarejestrować, aby mogła przechwycić zdarzenie. Odpowiedzialna za to jest powyżej wspomniana funkcja `cvSetMouseCallback()`:

```
void cvSetMouseCallback (
    const char* window_name,
    CvMouseCallback on_mouse,
    void* param = NULL),
```

gdzie parametr `window_name` wskazuje nazwę okna, dla którego funkcja jest rejestrowana, `on_mouse` jest funkcją rejestrowaną, a `param` jest wspomnianym parametrem przekazywanym do funkcji, która jest rejestrowana.

W celu ułatwienia zrozumienia zagadnienia obsługi myszy poniżej został przedstawiony program, który ma pokazać proste zastosowanie obsługi myszy.

Po włączeniu programu użytkownik może na czarnym obrazie rysować elementy graficzne. Po kliknięciu lewego klawisza myszy i jego przytrzymaniu rysowane są niebieskie prostokąty, przytrzymany, prawy przycisk myszy pozwala na rysowanie czerwonej linii, przyciśnięcie lewego przycisku myszki wraz z klawiszem SHIFT pozwala na rysowanie małych kół.



Rysunek 10 Przykłady zastosowania obsługi myszy w HighGUI

Obsługa klawiatury sprowadza się do wywołania funkcji `cvWaitKey()`. Jest to funkcja, którą często umieszcza się wewnątrz pętli `while`. Taki układ wykorzystuje się do zamknięcia programu za pomocą zadeklarowanego klawisza. Poniżej pokazany jest najprostszy sposób implementacji takiego układu:

```
while (1) {
    if (cvWaitKey(10) == 'q')
        break;
}
```

Stworzona została pętla nieskończona (argument pętli `while` jest zawsze prawdziwy). W jej wnętrzu warunkiem instrukcji `if` jest funkcja `cvWaitKey(10)` – liczba w nawiasie oznacza czas w milisekundach jaki funkcja ma oczekiwać na naciśnięcie klawisza. Jeżeli użytkownik wciśnie klawisz „q” nastąpi wyjście z pętli, które zazwyczaj kończy się zwolnieniem zasobów pamięci zaalokowanych na potrzeby programu, a następnie jego zakończeniem.

2.3.5 Dodawanie i wykorzystywanie suwaków

Niestety biblioteka HighGUI dostarcza tylko jeden element GUI, którym jest suwak. Brakuje natomiast podstawowego, wydawałoby się elementu jakim jest przycisk, czy przełącznik.

Funkcją odpowiedzialną za tworzenie suwaka jest `cvCreateTrackBar()`. Podobnie jak nowemu oknu, suwakowi przydzielana jest unikalna nazwa, przez którą w programie jest identyfikowany. Definicja `cvCreateTrackBar()` wygląda następująco:

```
int cvCreateTrackbar(  
    const char* trackbar_name,  
    const char* window_name,  
    int* value,  
    int count,  
    CvTrackbarCallback on_change);
```

Pierwszy parametr `trackbar_name` jest już wcześniej wspomnianą nazwą suwaka, `window_name` jest nazwą okna, w którym się znajduje (suwak zazwyczaj umieszczany jest na górze okna). Następny parametr `value` jest automatycznie ustawiany w zależności od pozycji, w której znajduje się suwak, `count` jest maksymalnym zakresem suwaka. Ostatni parametr jest funkcją obsługi zdarzenia, która zostaje wywoływana przy każdym jego przesunięciu. Suwak nie musi mieć włączonej obsługi zdarzeń, wystarczy ustawić wartość parametru na `NULL`, wówczas jedynym efektem działania suwaka będzie zmiana wartości `value`.

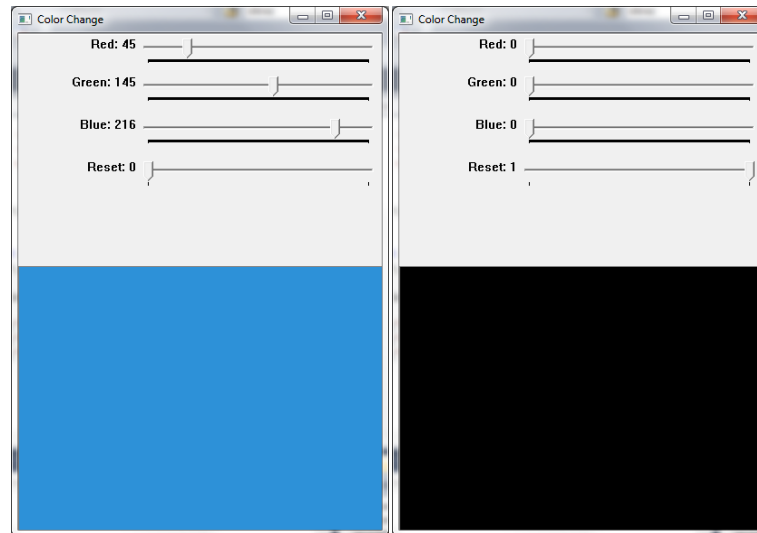
Należy wspomnieć jeszcze o dwóch przydatnych funkcjach związanych z obsługą suwaków. Pierwsza z nich `cvGetTrackBarPos()` służy do odczytywania wartości z suwaka. Wartości z suwaka mogą być używane np. do zmiany parametrów obrazu. Poniżej umieszczona zostaje definicja tej funkcji (jej parametrów nie trzeba tłumaczyć):

```
cvGetTrackBarPos(  
    const char* trackbar_name,  
    const char* window_name);
```

Druga funkcja `cvSetTrackBarPos()` zmienia położenie suwaka, a dokładnie jego wartość. Jej definicja wygląda następująco:

```
cvSetTrackBarPos(  
    const char* trackbar_name,  
    const char* window_name,  
    int pos);
```

Poniżej umieszczony został przykład zastosowania tych funkcji w programie, który zawiera cztery suwaki. Trzy z nich odpowiedzialne są za ustawianie parametrów związanych z przestrzenią kolorów RGB obrazu. Ostatni z nich resetuje zmiany do postaci (0,0,0). Działa na zasadzie przełącznika (ON/OFF). Program kończony jest po wciśnięciu klawisza „q”. Kod programu umieszczony został w załączniku nr 2.



Rysunek 11 Przykład zastosowania suwaków jako przycisków

Jak widać w HighGUI można poradzić sobie bez natywnego przełącznika. Wystarczy ustawić wartość count suwaka na 1.

Problem z brakiem standardowego przycisku można rozwiązać podobnie, jedyna różnica polega na takim skonstruowaniu funkcji obsługi zdarzeń, żeby pozycja suwaka po zmianie z 0 na 1 automatycznie wróciła na pozycję 0.

2.4 Operacje na macierzach i obrazach

2.4.1 Dostęp do pikseli obrazów

Do pikseli wszystkich typów obrazów można uzyskać dostęp dwiema metodami. Pierwsza z nich to metoda pośrednia, która wykorzystuje do tego celu funkcje z grupy `cvGet2D()` oraz `cvSet2D()`.

W przypadku kiedy mamy do czynienia z obrazami monochromatycznymi i jednokanałowymi wykorzystujemy funkcje `cvGetReal2D()` i `cvSetReal2D()`. Pierwsza z nich pobiera dane z piksela i zwraca je w postaci liczby double:

```
double cvGetReal2D(const CvArr** arr, int x0, int x1);
```

gdzie `arr` wskazany obraz, którego elementom nadajemy wartości, `x0` to współrzędna y, a `x1` współrzędna x obrazu.

Druga funkcja służy do zapisywania danych do piksela obrazu:

```
void cvGetReal2D(const CvArr** arr, int x0, int x1, double val);
```

gdzie dodatkowym parametrem jest `val`, będący wartością wpisywaną do piksela obrazu.

W przypadku obrazów kolorowych oraz wielokanałowych stosowane są analogiczne funkcje `cvSet2D()` i `cvGet2D()`. Ich definicje są niemal identyczne jak w przypadku obrazów czarno-białych, z tą różnicą, że operujemy na danych `CvScalar`, a nie `double`. Warto pamiętać, że wygodnym sposobem na zapisywanie wartości piksela z przestrzeni RGB jest wykorzystanie makra `CV_RGB` np.:

```
CvScalar pix_value = CV_RGB (r,g,b);,
```

Trójka `(r,g,b)` to liczby całkowite.

Dostęp pośredni do piksela obrazu jest sposobem mało wydajnym, dlatego najczęściej stosuje się dostęp bezpośredni, który realizowany jest za pomocą wskaźników. Problem ten zostanie opisany w oparciu o przykład, gdyż ten sposób najłatwiej będzie można zrozumieć zasadę jego działania:

```
double grey =0;
int r,g,b=0;
CvScalar pix_val = CV_RGB(b, g, r);
int height = img->height;
int width = img->width;
int step = img->widthStep/sizeof(uchar);
int channels = img->nChannels;
uchar* data = (uchar *)img->imageData;
// dla obrazu wielokanałowego
b= data [y*step + x*channels + 0]
g= data [y*step + x*channels + 1]
r= data [y*step + x*channels + 2]
// dla obrazu jednokanałowego
grey = data [y*step + x]
```

Dostęp do elementów obrazu uzyskany został za pomocą wskaźnika `*imageData`, zawartego w strukturze `IplImage`. Dane w nim przechowywane są jako typ `char`, dlatego należało wykonać rzutowanie na inny typ (w tym przypadku na typ `uchar` w celu uniknięcia problemów ze znakami). Długość linii obrazu jest określana przez parametr `step` (wyrażony w bajtach). Parametry `x` jest numerem kolumny obrazu, a `y` numerem wiersza.

2.4.2 Dostęp do fragmentów obrazu

Dostęp do fragmentów obrazu odbywa się dzięki funkcji `cvGetSubRect()`:

```
CvSize cvGetSubRect(const CvArr* arr,CvArr* submat,CvRect rect);
```

Wydziela ona interesujący obraz za pomocą prostokąta `rect`, i wypełnia obraz `submat` pikselami wyznaczonego obrazu. Oryginalny obraz jest wyznaczany za pomocą parametru `arr`.

2.4.3 Obszar ROI i COI

Obszar ROI został zaimplementowany w celu zwiększenia wydajności i szybkości działania programów. Pozwala on na ograniczenie przetwarzania wyłącznie do prostokątnego fragmentu obrazu.

Funkcją odpowiedzialną za określanie prostokątnego obszaru jest `cvSetImageROI()`:

```
void cvSetImageROI (IplImage** image, CvRect rect);,
```

gdzie `image` jest obrazem wyjściowym, `rect` zdefiniowanym prostokątnym obszarem.

Funkcja usuwającą obszar ROI jest `cvResetImageROI()`:

```
void cvResetImageROI (IplImage** image);,
```

której jedynym parametrem jest obszar, na którym ROI się znajdował.

W celu odnalezienia położenia ROI na obrazie można posłużyć się funkcją `cvGetImageROI()`:

```
CvRect cvGetImageROI (const IplImage* image);,
```

która zwraca obiekt typu `CvRect` z informacją o położeniu i rozmiarze ROI.

Drugi z obszarów COI jest zdecydowanie rzadziej używanym narzędziem. Pozwala on na wyodrębnienie interesującego nas kanału kolorów, który ma zostać poddany przetwarzaniu. Do pracy z COI wykorzystywane są analogicznie do ROI funkcje `cvSetImageCOI()`, `cvResetImageCOI()`, `cvGetImageCOI()`.

W celu lepszego zrozumienia zagadnienia regionu zainteresowania napisany został program, który na prostym przykładzie pokazuje jeden ze sposobów wykorzystania ROI.



Rysunek 12 Przykład zastosowania ROI na obrazie.

Na powyższym rzucie ekranu pokazane zostało działanie programu. Na obrazie głównym zostaje zaznaczony obszar ROI. Drugi obraz, który widnieje w prawym dolnym rogu, został wycięty z obrazu głównego objętego ROI i przekonwertowany z formatu RGB na skalę szarości.

2.4.4 Operacje macierzowe

Biblioteka OpenCV dostarcza wiele funkcji odpowiedzialnych za operacje na macierzach, które wykorzystywane są do przetwarzania obrazów. W tym podrozdziale zostaną opisane tylko te, które są najczęściej używane lub zostały wykorzystane w moich projektach.

Pierwszą grupę stanowią funkcję obliczające wartości bezwzględne:

- `cvAbs()` – oblicza wartość bezwzględną z macierzy,
- `cvAbsDiff()` – oblicza wartość bezwzględną z różnicy dwóch macierzy,
- `cvAbsDiffS()` – oblicza wartość bezwzględną z różnicy macierzy i skalar.

Następnie grupę stanowią funkcję dodające:

- `cvAdd()` - dodaje do siebie dwie macierze,
- `cvAddS()` – dodaje do siebie macierz i skalar,

Pozostałe funkcję zostaną opisane w postaci wyliczenia:

- `cvAvg()` - oblicza średnią wartość ze wszystkich elementów macierzy,
- `cvConvertScale()` – pozwala zmienić głębokość pikseli obrazu np. z 8 bitowego obrazu RGB na 16 bitowy lub przeprowadzić liniową transformację w oparciu o parametry `scale` i `Shift`,
- `cvCopy()` – kopiuje elementy z jednej macierzy do drugiej,
- `cvCvtColor()` – została opisana w podrozdziale 2.2.2,
- `cvDet()` – oblicza wyznacznik macierzy kwadratowej,
- `cvDiv()` – wykonuje dzielenie macierzy jednej przez drugą,
- `cvGetSubRect()` – została opisana w podrozdziale 2.4.2,
- `cvInRange()` – sprawdza, które elementy macierzy znajdują się w zdefiniowanym wcześniej przedziale. W wynikowej macierzy elementy z zakresu mają wartość maksymalną, zaś elementy z poza zakresu wartość 0,
- `cvInRange_()` – podobnie jak `cvInRange()`, tylko porównywane są elementy macierzy ze skalem,
- `cvMax()` – przy porównaniu odpowiadających sobie elementów z dwóch macierzy, do macierzy wynikowej zostaje wpisany piksel o większej wartości,
- `cvMin()` – podobnie jak `cvMax()`, tylko do macierzy wynikowej zostaje wpisany piksel o mniejszej wartości,
- `cvMinMaxLoc()` – znajdują wartość minimalną i maksymalną pikseli macierzy, zwraca ich współrzędne i zapisuje ich wartości do zmiennych,

- `cvNorm()` – wyznacza normę z jednej lub dwóch macierzy. Wszystkie dopuszczalne normy opisane są w dokumentacji biblioteki,
- `cvSplit()` – pozwala na rozdzielenie macierzy z pikselami wielokanałowymi (o 3 lub 4 wartościach), na macierze o pikselach jednokanałowych (o jednej wartości),
- `cvZero()` – ustawia wszystkie wartości elementów macierzy na 0.

Dokładna specyfikacja przedstawionych funkcji opisana jest w podręczniku użytkownika biblioteki OpenCV.

2.4.5 Rysowanie kształtów

Podczas pracy z obrazami często występuje potrzeba narysowania na nim pewnego obiektu. Biblioteka OpenCV dostarcza narzędzia niezbędne do narysowania linii, kwadratów, okręgów, elips i wielokształtów. Pozwala również na umieszczanie znaków tekstowych na obrazach.

OpenCV zawiera trzy algorytmy do rysowania linii. Dwa z nich to algorytmy Bresenhama a trzeci jest algorytmem antyaliasingowym, wykorzystującym filtr Gaussa [3]. Funkcja rysująca linię to `cvLine()`:

```
void cvLine( IplImage* img, CvPoint pt1, CvPoint pt2, int color, int
thickness, int connectivity = 8 );
```

gdzie `img` jest obrazem na którym będzie rysowana linia, `pt1` oraz `pt2` wyznacza początek i koniec linii, `color` określa jej kolor, parametr `thickness` wyznacza grubość linii (w pikselach). Ostatni z parametrów `connectivity` określa sposób rysowania linii. Wartość domyślna 8, pozwala na rysowanie idealnie prostej linii, wartość 4 linii lekko chropowatej i załamanej, ale za to rysowanej dużo szybciej.

Funkcja umożliwiająca narysowanie prostokąta to `cvRectangle()`:

```
void cvRectangle( IplImage* img, CvPoint pt1, CvPoint pt2, int color, int
thickness );
```

gdzie wszystkie parametry są takie same jak w `cvLine()`. Dodatkowo parametr `thickness` przyjmując wartość -1 (lub flagę `CV_FILLED`) wypełnia figurę kolorem określonym parametrem `color`.

`CvCircle()` jest funkcją odpowiedzialną za rysowanie okręgów. Jej deklaracja wygląda następująco:

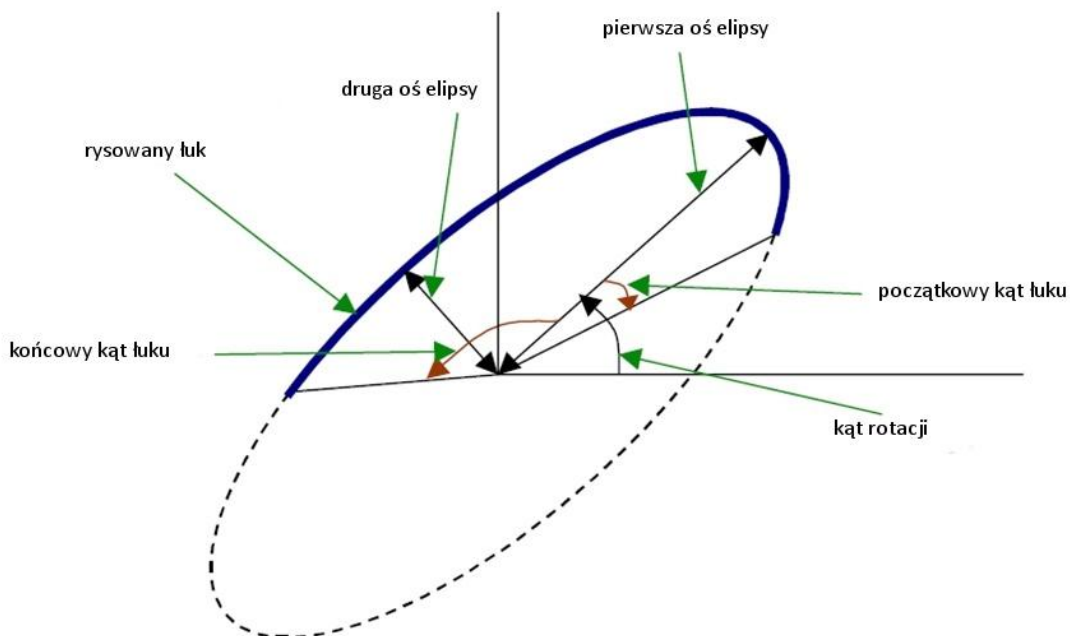
```
void cvCircle( IplImage* img, CvPoint center, int radius, int color, int
thickness=1 );
```

gdzie `center` jest punktem środka okręgu, a `radius` jego promieniem.

Funkcja `cvEllipse()` rysuje elipsy:

```
void cvEllipse( IplImage* img, CvPoint center, CvSize axes, double angle,
double startAngle, double endAngle, int color, int thickness=1 );
```

gdzie center jest środkiem elipsy, axes wymiary osi elipsy, angle jest kątem rotacji elipsy, startAngle jest początkowym kątem rysowania krzywej elipsy, endAngle kątem końcowym, ostatnie parametry tak samo jak w przypadku pozostałych funkcji rysujących. W celu lepszego zrozumienia parametrów elipsy zamieszczony został poniżej rysunek:



Rysunek 13 Sposób rysowania Elipsy w HighGUI⁷

Ostatnia grupa funkcji dotyczy rysowania wieloboków. Odpowiedzialne za nie są trzy funkcje, które różnią się sposobem wyznaczania punktów:

```
void cvFillPoly( IplImage* img, CvPoint** pts, int* npts, int contours,int
color );
void cvFillConvexPoly( IplImage* img, CvPoint* pts, int npts, int color );
void cvPolyLine( IplImage* img, CvPoint** pts, int* npts, int contours,
isClosed, int color, int thickness=1 );
```

Pierwsza z nich ma możliwość rysowania wielu wieloboków razem, ale jest stosunkowo wolna. CvFillConvexPoly() działa na tej samej zasadzie co cvFillPoly() ale może rysować tylko jeden wielobok w tym samym czasie. Dodatkowo musi to być wielobok wypukły. Sporym atutem tej funkcji jest szybkość działania. Ostatnia funkcja ma dokładnie takie same możliwości jak cvFillPoly() jednak przez to, że działa w inny sposób jest od niej zdecydowanie szybsza.

⁷Źródło obrazu (modyfikacja własna): [W7]

2.4.6 Wstawianie tekstu

Jak już zostało wspomniane OpenCV pozwala na umieszczanie tekstu na obrazach. Ograniczeniem nałożonym przez twórców biblioteki jest ilość czcionek. Przed dodaniem napisu do obrazu należy najpierw zainicjować czcionkę, którą litery będą pisane. Tworzy się najpierw obiekty typu `CvFont` funkcją `cvFont()`, a następnie za pomocą `cvInitFont()` ładuję się konkretną czcionkę:

```
cvInitFont( CvFont* font, int font_face, double hscale, double vscale, double
shear =0, int thickness = 1, int line_type = 8);
```

gdzie `font` jest wskaźnikiem do struktury czcionki, `font_face` jest rodzajem czcionki, `hscale` wysokością czcionki, `vscale` szerokością, `shear` kątem pochylenia czcionki, pozostałe parametry opisane zostały w podrozdziale 2.4.5.

Po inicjalizacji czcionka dodaje się funkcją `cvPutText()`:

```
cvPutText( CvArr* img, const char* text, CvPoint org, const CvFont* font,
CvScalar color);
```

gdzie `img` jest obrazem, na który ma być nakładany tekst, `text` ciągiem wpisywanego tekstu, `org` współrzędnymi lewego dolnego rogu tekstu, `font` czcionką która została wcześniej zainicjowana, `color` zaś jej kolorem.

3. Przetwarzanie obrazów

3.1 Progowanie

Progowanie jest jedną z najprostszych metod wydzielenia obszarów (segmentacji), spełniających pewne kryterium jednorodności [4]. Operacja ta realizowana jest na zasadzie porównywania każdego piksela obrazu z zadany progim (ang. threshold) i przypisaniem mu odpowiedniej wartości. W bibliotece OpenCV zaimplementowanych jest kilka rodzajów progowania:

- progowanie binarne (CV_THRESH_BINARY),
- odwrócone progowanie binarne (CV_THRESH_BINARY_INV),
- okrawające (CV_THRESH_TRUNC),
- progowanie „do zera” proste (CV_THRESH_ZERO),
- progowanie „do zera” odwrotne (CV_THRESH_ZERO_INV).

Operacja progowania wykonywana jest za pomocą funkcji `cvThreshold()`:

```
double cvThreshold(const CvArr* src, CvArr* dst,  
double threshold,  
double maxValue,  
int thresholdType );
```

gdzie parametr `threshold` jest wartością progu, `maxValue` maksymalną wartością przypisywaną do piksela, `thresholdType` jednym z wyliczonych wyżej sposobów progowania.

Dla progowania binarnego dla wartości piksela większej od wyznaczonego progu funkcja przypisuje 0, w przeciwnym wypadku wartość `maxValue`:

$$dst(x, y) = \begin{cases} maxValue & \text{if } src(x, y) > threshold \\ 0 & \text{otherwise} \end{cases}$$

gdzie x i y są współrzędnymi piksela obrazu wyjściowego `dst` i źródłowego `src`.

W przypadku odwróconego progowania binarnego piksel o wartości większej od progu ma przypisywaną wartość 0, zaś w przeciwnym wypadku wartość `maxValue`:

$$dst(x, y) = \begin{cases} 0 & \text{if } src(x, y) > threshold \\ maxValue & \text{otherwise} \end{cases}$$

W progowaniu okrawającym piksel o wartości większej od progu ma przypisywaną wartość tego progu, w przeciwnym wypadku jego wartość nie jest zmieniana.

$$dst(x, y) = \begin{cases} threshold & \text{if } src(x, y) > threshold \\ src(x, y) & \text{otherwise} \end{cases}$$

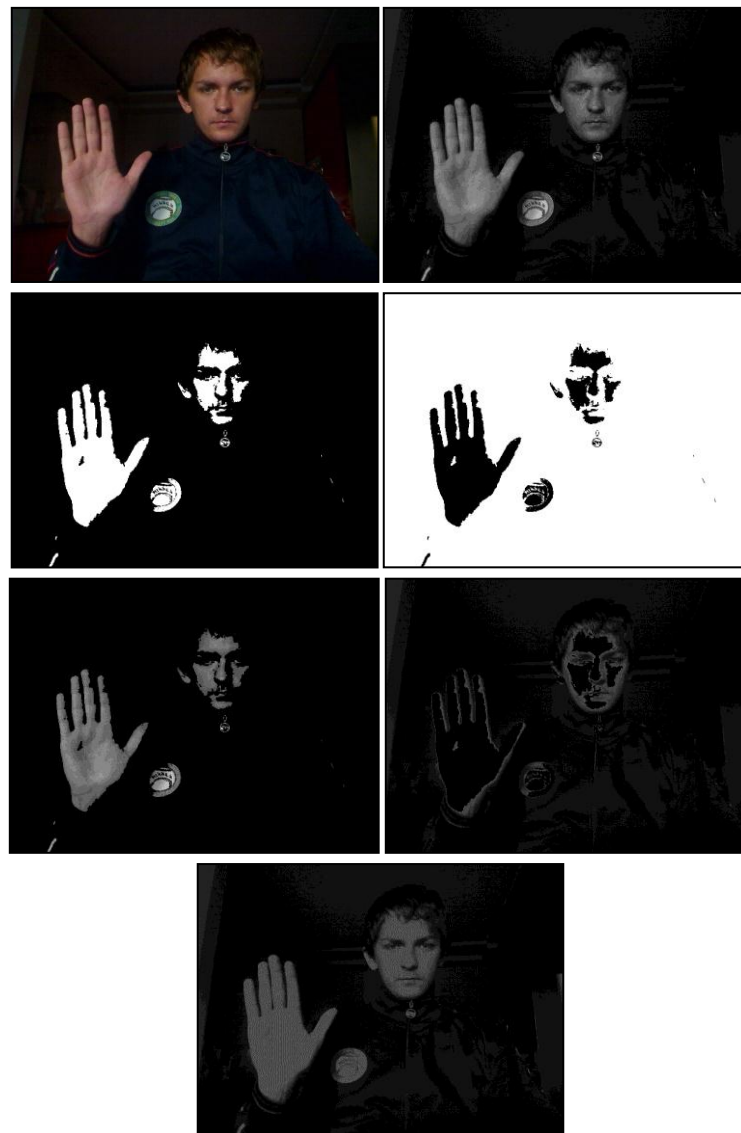
Progowanie do zera proste zakłada, że do piksela o wartości mniejszej od ustalonego progu przypisywana jest wartość 0, w przeciwnym wypadku wartość nie jest zmieniana.

$$dst(x, y) = \begin{cases} src(x, y) & \text{if } src(x, y) > threshold \\ 0 & \text{otherwise} \end{cases}$$

Funkcja ostatniego omawianego progowania do zera odwrotnego, przypisuje pikselowi o wartości powyżej progu wartość 0 lub w przeciwnym wypadku nie zostaje zmieniana.

$$dst(x, y) = \begin{cases} 0 & \text{if } src(x, y) > threshold \\ src(x, y) & \text{otherwise} \end{cases}$$

Przykłady realizacji omawianych rodzajów progowa przedstawiają poniższe rysunki.



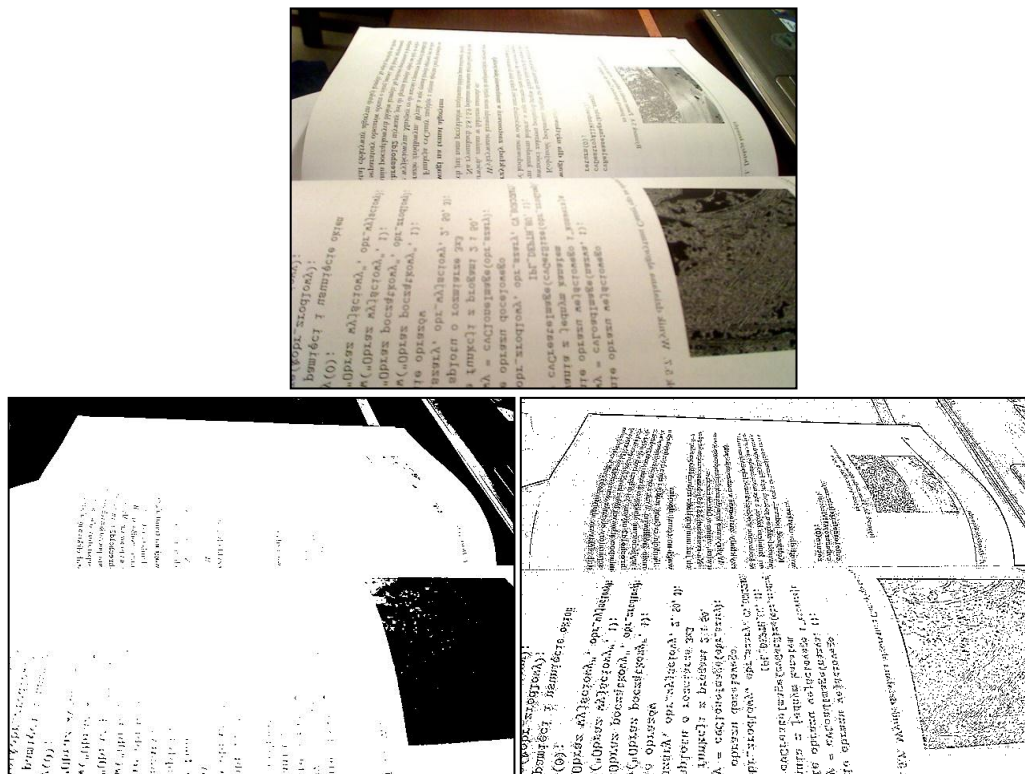
Rysunek 14 Progowanie binarne, binarne odwrócone, proste, proste odwrócone, okrawające

Biblioteka dodatkowo dostarcza bardzo pomocną zmodyfikowaną wersję progowania, tzw. progowanie adaptacyjne, w którym wartość progu jest jedną ze zmiennych. Algorytm ten jest realizowany funkcją `cvAdaptiveThreshold()`:

```
void cvAdaptiveThreshold(CvArr* src, CvArr* dst,
double max_val,
int adaptive_method = CV_ADAPTIVE_THRESH_MEAN_C
int threshold_type = CV_THRESH_BINARY,
int block_size = 3,
double param1 = 5
);
```

Progowanie adaptacyjne może być wykonywane na dwa sposoby określane przez parametr `threshold_type`. W obu przypadkach badane piksele klasyfikowane są na podstawie swojej wartości oraz zmiennego progu obliczonego na podstawie analizy pikseli otoczenia badanego punktu. Otoczenie to jest wyznaczane przez kwadratowy obszar o boku wartości `block_size`. Z sąsiedztwa badanego piksela obliczana jest średnia ważona wartość, od której odejmowana jest stała wartość `param1`.

Pierwszy sposób obliczania progu to `CV_ADAPTIVE_THRESH_MEAN_C`, w którym każdy piksel jednakowo wpływa na obliczenie wartości średniej ważonej. W drugim wypadku dla `CV_ADAPTIVE_THRESH_GAUSSIAN_C` waga pikseli obliczana jest z wykorzystaniem dwuwymiarowego rozkładu Gaussa na podstawie ich odległości od badanego piksela.



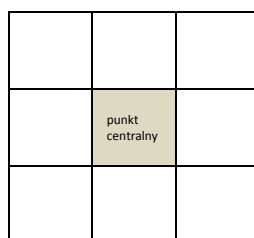
Rysunek 15 Przykład działania progowania adaptacyjnego.

Jak widać na powyższym porównaniu. Zdecydowanie lepszy wynik dało zastosowanie progowania adaptacyjnego. W obu przypadkach algorytmem progowania był `CV_THRESH_BINARY`. Progowanie adaptacyjne wywołane zostało progowaniem algorytmem uśredniającym `CV_ADAPTIVE_THRESH_MEAN_C`. Zastosowanie algorytmu z rozkładem Gaussa dało niewiele gorszy efekt od algorytmu uśredniającego.

3.2 Przekształcenia kontekstowe i konwolucja

Operacje kontekstowe są przekształceniami, które dla wyznaczenia wartości jednego piksela obrazu wynikowego dokonują obliczeń na wielu pikselach obrazu źródłowego. Polega to na wyznaczeniu wartości funkcji (filtru), której argumentami są wartości piksela o tym samym położeniu na obrazie źródłowym oraz wartości pikseli z jego otoczenia (obszaru maski)[17]. Operacje te pozwalają wydobyć z obrazu wiele informacji, umożliwiają m.in. usuwanie lub ograniczanie poziomu zakłóceń nałożonych na obraz (redukcja szumu), wygładzają chropowate krawędzie, rozmywają i wygładzają obraz, ujawniając szczegóły obiektów itp.[4]

Przekształcenia kontekstowe realizowane są za pomocą konwolucji (splotu funkcji) obrazu źródłowego z maską filtru. Maska splotu jest macierzą elementów zwanych współczynnikami maski, definiuje ona otoczenie analizowanego piksela. Zazwyczaj ma ona kształt kwadratu, z punktem centralnym (ang. anchor point) zazwyczaj znajdującym się w jej środku.



Rysunek 16 Maska splotu o rozmiarze 3 x 3.

Wartość konwolucji badanego piksela jest obliczana w następujący sposób. Najpierw punkt centralny maski umieszczany jest na analizowanym pikselu źródłowego obrazu, a pozostałe punkty maski nakładane są na piksele sąsiednie. Wartość badanego piksela obliczana jest poprzez sumowanie iloczynów współczynników maski z wartościami odpowiadającymi im pikseli na obrazie. Operacja ta jest powtarzana dla każdego piksela obrazu źródłowego. Proces może zostać opisany następującym równaniem matematycznym:

$$H(x, y) = (G \times I)(x, y) = \sum_{i=0}^{M_i-1} \sum_{j=0}^{M_j-1} I(x + i - a_i, y + j - a_j) \cdot G(i, k)$$

gdzie $I(x, y)$ jest obrazem źródłowym, $G(i, j)$ maską filtra oraz punkt centralny znajduje się w punkcie (a_i, a_j) .

W OpenCV operację tę wykonuje funkcja `cvFilter2D()`:

```
void cvFilter2D(  
    const CvArr* src,  
    CvArr* dst,  
    const CvMat* kernel,  
    CvPoint anchor = cvPoint(-1,-1)  
);
```

gdzie parametr `kernel` jest wskaźnikiem do macierzy maski, którą tworzy programista, wypełniając ją wartościami współczynników. W standardzie OpenCV 2.1 wartości te muszą być liczbami zmiennoprzecinkowymi 32bitowymi (`CV_32FC1`).

Istotnym problemem podczas wykonywania operacji kontekstowych (filtracji) są piksele brzegowe obrazu. Maski nałożona na analizowany brzegowy piksel nie zawiera pełnej ilości potrzebnych punktów do obliczenia jego nowej wartości. Funkcja `cvFilter2D()` radzi sobie z tym problemem. Jednak w momencie gdy programista chce stworzyć własny algorytm obliczania spłotu funkcji, musi sam rozwiązać ten problem. Biblioteka zawiera funkcję `cvCopyMakeBorder()`, która kopiuje obraz źródłowy do wnętrza innego obrazu, którego wysokość oraz szerokość jest powiększona o $(N-1)/2$ pikseli (przy założeniu, że maska ma rozmiar $N \times N$).

3.3 Filtry do detekcji krawędzi

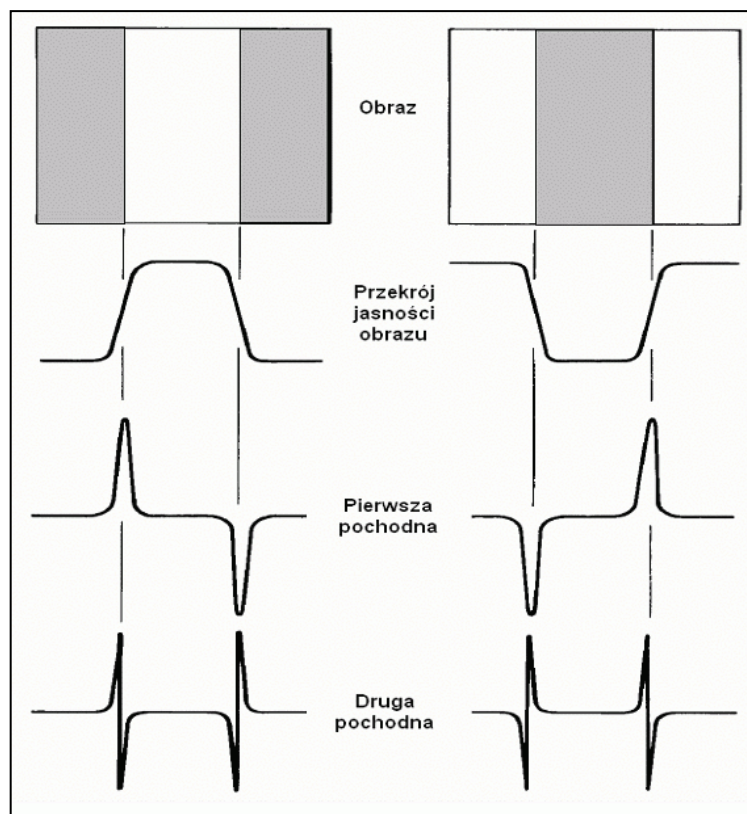
Detekcja krawędzi polega na takiej modyfikacji obrazu, aby pokazać krawędzie występujących na nim obiektów. Detekcja krawędzi odbywa się za pomocą obliczenia pierwszej i drugiej pochodnej funkcji opisującej zmiany jasności pikseli na obrazie.

Na obrazie dyskretnym wyznaczanie pierwszej pochodnej polega na policzeniu dyskretnego operatora gradientu, zaś drugiej pochodnej na policzeniu kierunku zmiany operatora gradientu.

Filtry krawędziowe oparte o pierwszą pochodną dają najlepsze rezultaty dla obrazów, na których obiekty charakteryzują się dużą różnicą poziomu jasności pikseli w stosunku do tła, natomiast dla obrazów, na których granica pomiędzy jasnością pikseli obiektów i tła jest bardziej rozmyta stosuje się metodę opartą o drugą pochodną.

Należy pamiętać, że stosowanie filtrów krawędziowych powoduje wyostrenie małych detali oraz powstaniu szumów impulsowych, co skutkuje pogorszeniem jakości obrazu oraz błędną detekcją nieistniejących brzegów.

W rozdziale tym opisane zostały następujące filtry do detekcji krawędzi wykorzystujące operatory Roberts'a, Sobela, Laplace'a, Canny'ego.



Rysunek 17 Obliczanie pierwszej i drugiej pochodnej obszarów o zmiennej jasności [5]

Operatory gradientów po zadanych współrzędnych dla obrazu $I(x, y)$:

- po współrzędnej x : $G_x = \frac{\partial I(x, y)}{\partial x}$
- po współrzędnej y : $G_y = \frac{\partial I(x, y)}{\partial y}$

W obrazach składających się z macierzy pikseli gradienty dyskretne wyznaczane są na podstawie różnicy wartości dwóch sąsiednich pikseli. Podstawowymi operatorami wykorzystywanymi do tego celu są operatory Roberts'a:

0	0	0
0	-1	1
0	0	0

0	0	0
0	-1	0
0	0	1

0	0	0
0	-1	0
0	1	0

0	0	0
0	-1	0
1	0	0

Są one jednak bardzo podatne na szumy i należy je stosować jedynie na obrazach o dobrej jakości lub na obrazach po filtracji usuwającej szumy wysokoczęstotliwościowe (np. filtrem Gaussa, medianowym).

3.3.1 Filtr Sobela

Bardziej odporny na zakłócenia jest operator Sobela. Funkcją realizującą filtr Sobela jest `cvSobel()`:

```
void cvSobel(const CvArr* src,CvArr* dst,
int xorder,
int yorder,
int apertureSize=3 );
```

Pozwala ona na obliczanie pierwszej, drugiej i trzeciej pochodnej obrazie przez jego splot z maskami: $dst(x,y) = \frac{\partial^{xorder+yorder}}{\partial x^{xorder} \partial y^{yorder}} src(x,y)$. Parametr `xorder` jest rzędem pochodnej w kierunku x, `yorder` w kierunku y, `aperture_size` rozmiarem maski Sobela. Wersja 2.1 OpenCV pozwala na używanie rozmiarów 1x1, 3x3, 5x5, 7x7. Standardowo funkcja ta jest wywoływana z parametrami:

- `xorder = 1, yorder = 0, aperture_size = 3`, wówczas maska ma postać:

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

- `xorder = 0, yorder = 1, aperture_size = 3`, wówczas maska ma postać:

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

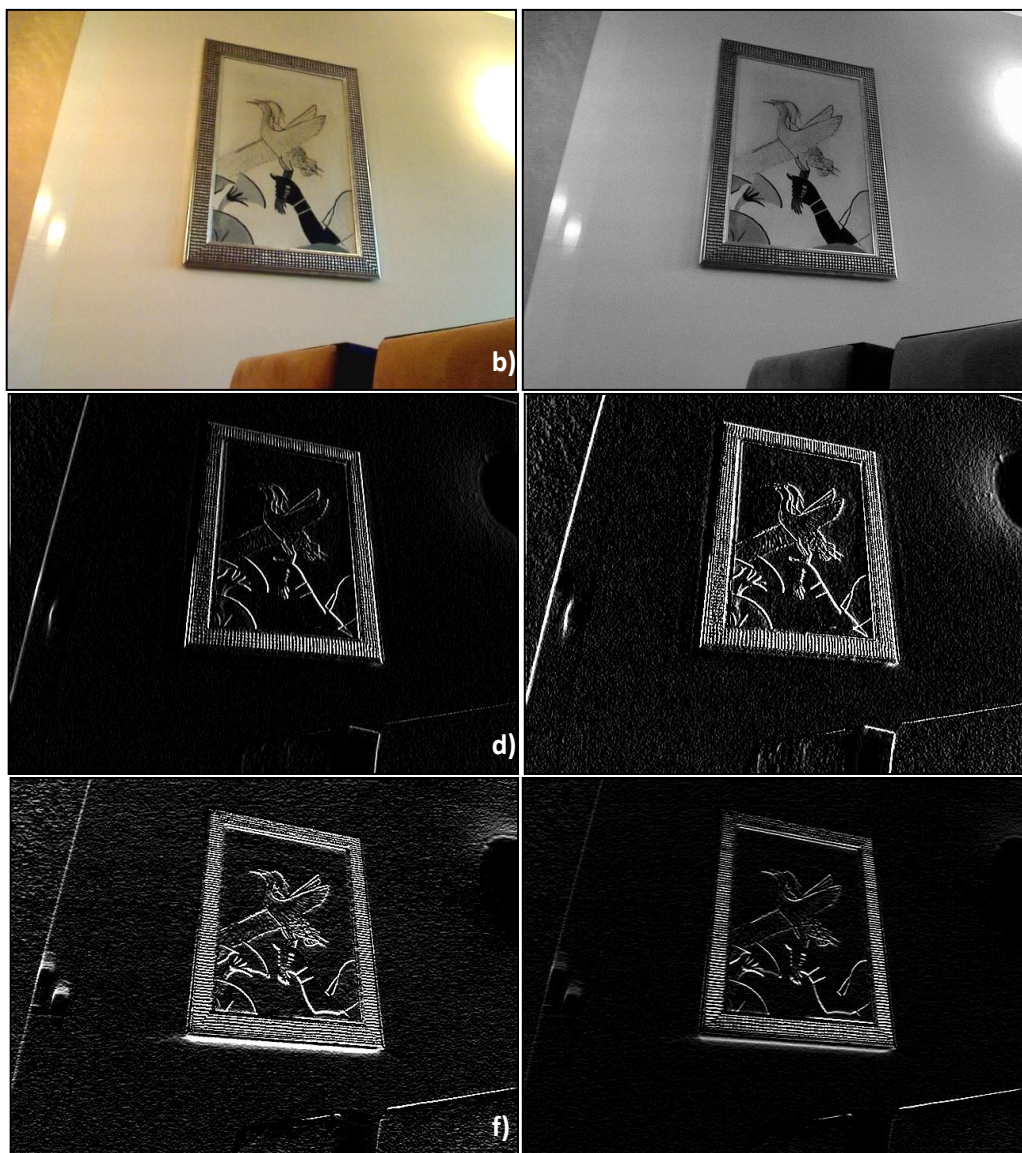
Operator Sobela dzięki połączeniu właściwości Gaussowskiego rozmycia oraz różniczkowania jest dość odporny na zakłócenia na obrazie w momencie gdy rozmiar maski jest odpowiednio duży. W przypadku zastosowywania maski 3x3 do wykrycia krawędzi pionowych lub poziomych operator Sobela radzi sobie dostatecznie dobrze.

Jeżeli potrzebna jest większa dokładność przy zachowaniu tej samej złożoności obliczeniowej można wykorzystać specjalną wartość parametru `aperture_size` funkcji `cvSobel()`, czyli `CV_SCHARR` (numerycznie „-1”). Jest to filtr o masce 3x3, który radzi sobie lepiej niż filtr Sobela o tych samych rozmiarach maski. Maski operatora Scharr’a mają następującą postać:

-3	0	3
-10	0	10
-3	0	3

3	10	3
0	0	0
-3	-10	-3

Poniżej przedstawione są efekty działania filtru Sobela z różnymi parametrami:



Rysunek 18 Zastosowanie różnych operatorów kierunkowych Sobela dla obrazu w skali szarości

Powyższe rysunki odpowiadają kolejno: a – obraz oryginalny, b – obraz w skali szarości, c – maska Sobela XY, d – maska Sobela XY 5x5, e – maska Sobela X, f – maska Sobela Y, g – maska Scharr’a X, h – maska Scharr’a Y. Z obserwacji wynika, że Sobel jest typowym filtrem kierunkowym. Krawędzie wykrywane są w płaszczyznach odpowiadającym wykorzystanym operatorom kierunkowym. Widać również, że zastosowanie maski Scharr’a, która przez większe współczynniki maski zwiększyła istotność pikseli sąsiednich w obliczaniu punktu centralnego i dzięki temu krawędzie są bardziej wyeksponowane.

Krawędzie mogą być wykrywane w pełnym otoczeniu punktu obrazu za pomocą operatora kombinowanego obliczanego na podstawie amplitudy gradientu, obliczanej z pionowego i poziomego gradientu zgodnie z formułą Euklidesową: $|G| = \sqrt{G_x^2 + G_y^2}$. Stosuje

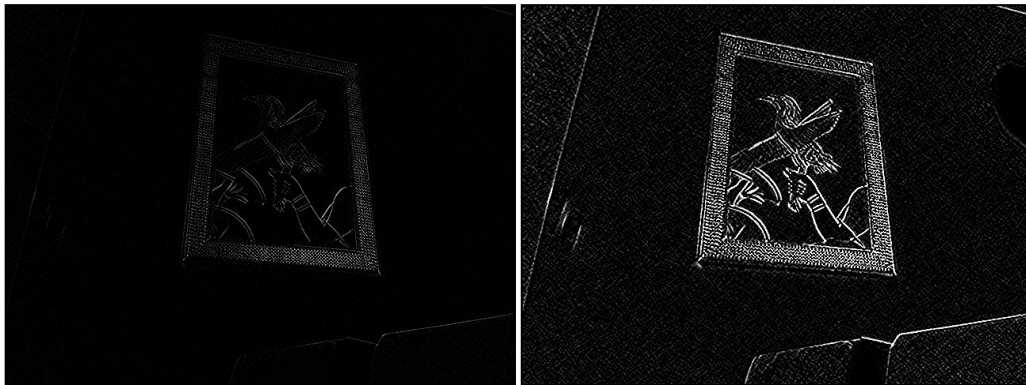
się ją, aby uniknąć kompensacji pikseli dodatnich i ujemnych powstałych przy zastosowaniu filtrów kierunkowych pionowych i poziomych. Operacja ta przy obliczeniach komputerowych aproksymowana jest do uproszczonej formy: $|G| = |G_x| + |G_y|$. Dla maski zdefiniowanej w poniższy sposób:

M_1	M_2	M_3
M_4	M_5	M_6
M_7	M_8	M_9

amplituda gradientu obliczana jest ze wzoru

$$|G| = |(M_1 + 2 \cdot M_2 + M_3) - (M_7 + 2 \cdot M_8 + M_9)| + |(M_3 + 2 \cdot M_6 + M_9) - (M_1 + 2 \cdot M_4 + M_7)|$$

Efekt działania filtru Sobela z wykorzystaniem operatora kombinowanego:



Rysunek 19 Zastosowanie operatora kombinowanego 3x3 i 5x5 dla filtru Sobela

Powyższy rysunek potwierdza fakt, iż zastosowanie większej maski powoduje widoczniejszą detekcję krawędzi.

3.3.2 Filtr Laplace'a

Filtr Laplace'a umożliwia wykrywanie wszystkich krawędzi. Jego działanie oparte jest na obliczeniu laplasjanu, który jest sumą drugich pochodnych cząstkowych w prostokątnych kierunkach x i y. W OpenCV służy do tego funkcja `cvLaplace()`, której parametry są takie same jak w przypadku funkcji `cvSobel()`:

```
void cvLaplace(const CvArr* src, CvArr* dst, int apertureSize=3);
```

Funkcja oblicza laplasjan obrazu wyjściowego na podstawie wzoru:

$$dst(x, y) = \frac{\partial^2 src(x, y)}{\partial x^2} + \frac{\partial^2 src(x, y)}{\partial y^2}$$

Realizacja algorytmu tej funkcji odbywa się przy wykorzystaniu operatora Sobela dla drugiej pochodnej. Dla parametru `aperture_size=1` laplasjan obrazu wyjściowego jest splotem obrazu wejściowego z poniższą maską:

0	1	0
1	-4	1
0	1	0

Krawędzie wykrywane są w miejscach gdzie druga pochodna funkcji zmiany jasności pikseli na obrazie jest równa zero (miejsce zerowe). Oznacza, to że krawędzie obiektów na obrazie znajdują się w miejscu gdzie laplasjan badanego obszaru równa się 0.



Rysunek 20 Zastosowanie filtru Laplace'a do wykrywania krawędzi

Jak można zauważyć na rysunkach filtr Laplace'a wykrywa krawędzie w każdej płaszczyźnie. Są one wyraźne i uwypuklone.

3.3.3 Filtr Canny'go

Najbardziej rozbudowany jest filtr Canny'ego. Oparty jest o wielostopniowy algorytm spełniający następujące kryteria:

- powinien rozpoznawać możliwie dużą ilość krawędzi,
- odległość pomiędzy znalezioną krawędzią, a krawędzią rzeczywistą powinna być minimalna,
- znaleziona krawędź powinna być wykryta tylko raz.

Algorytm spełniający te wymogi najpierw wykorzystuje filtr Gaussa w celu redukcji ewentualnych zakłóceń na obrazie. Następnie używa filtru Sobela z operatorami znajdującymi krawędzie pod czterema kątami 0, 45, 90 i 135 stopni. Na podstawie tego znajdowane są gradienty (z lokalnymi maksimami), które stają się kandydatami do bycia częścią krawędzi. Badane są sąsiednie piksele kandydatów w przestrzeni prostopadłej do kierunku ich gradientu. Jeżeli wartość piksela kandydata ma mniejszą intensywność od któregoś z sąsiadów to jest on eliminowany. Końcowa faza algorytmu polega na zastosowaniu progowania histerezy.

Ustala się dwa progi: dolnym i górnym, które umożliwiają ostateczny wybór pikseli linii krawędzi. Jeżeli wartość gradientu piksela kandydata jest większa niż górny próg to staje się on częścią krawędzi, jeżeli jego wartość jest mniejsza od progu dolnego to zostaje on odrzucony. W momencie gdy piksel znajduje się w przedziale pomiędzy progiem dolnym i górnym to zostaje on zaakceptowany tylko w przypadku, gdy jest połączony z ciągiem pikseli o wartości powyżej progu histerezy.

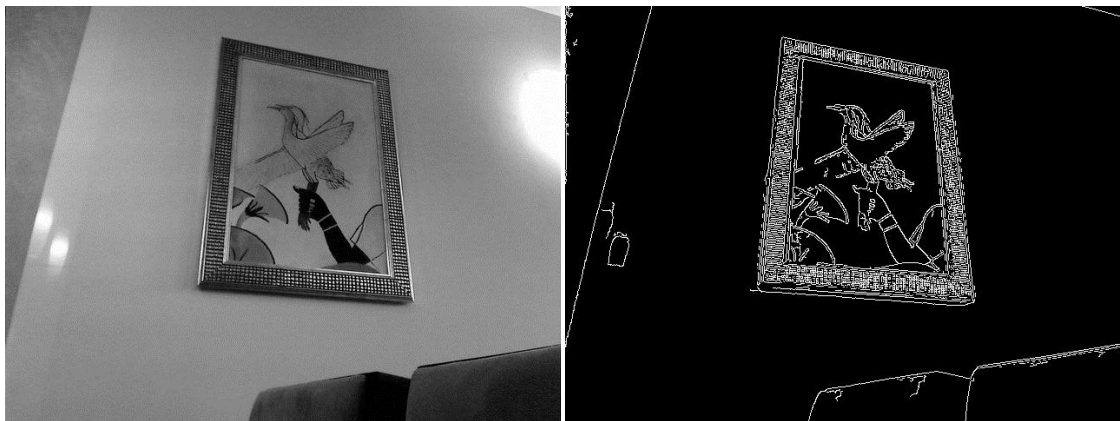
W OpenCV algorytm ten realizowany jest przez funkcję `cvCanny()`:

```
void Canny( const Mat& image, Mat& edges,
            double threshold1, double threshold2,
            int apertureSize=3, bool L2gradient=false );
```

gdzie `image` jest obrazem źródłowym, `edges` obraz zwracany z wykrytymi krawędziami, `threshold1` jest dolnym progiem, `threshold2` górnym, `apertureSize` jest rozmiarem maski Sobela. Ostatni parametr `L2gradient` określa sposób obliczania gradientu kierunkowego opartego o operatory Sobela. Jeżeli parametr ten ma wartość `true` gradient kierunkowy obliczany jest z przybliżonego wzoru $|G| = |G_x| + |G_y|$, gdy `false` ze wzoru dokładnego

$$|G| = \sqrt{G_x^2 + G_y^2}.$$

Funkcja `cvCanny()` wymaga, aby obraz wejściowy był w skali szarości oraz żeby był 8-bitowy. Wynikowy obraz binarny jest również 8 bitowy.



Rysunek 21 Zastosowanie filtru Canny'ego do detekcji krawędzi

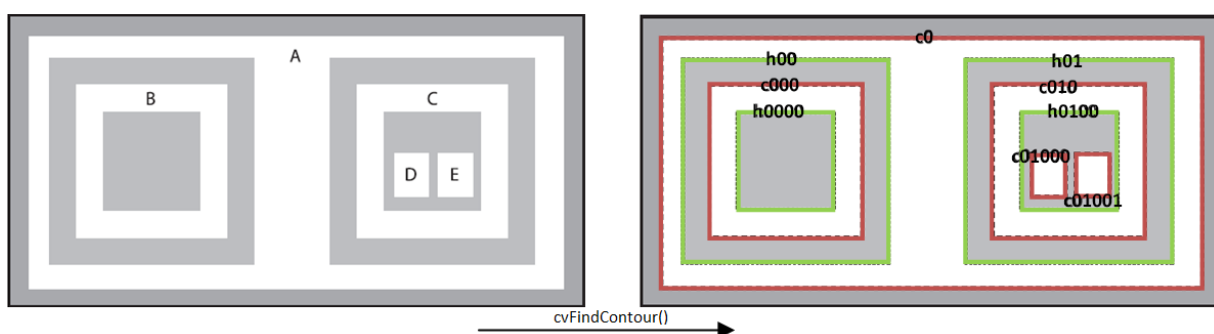
Powyżej został zrealizowany filtr Canny'ego z parametrami `threshold1 = 40` oraz `threshold2 = 120`, przy standardowej masce 3x3. Efekt bardzo zadowalający na obrazie binarnym widoczne są praktycznie wszystkie poszukiwane krawędzie.

3.4 Kontury

3.4.1 Kontury i ich interpretacja

Kontur jest listą punktów, które mogą reprezentować na wiele sposobów krzywą na obrazie. W OpenCV kontury reprezentowane są za pomocą sekwencji `CvSeq` (omawianych w rozdziale 2.1.2), gdzie każda pozycja w sekwencji zawiera informację o lokalizacji następnego punktu krzywej. Funkcją odpowiedzialną za obliczanie konturów z obrazu binarnego jest `cvFindContours()`. Z praktycznego punktu widzenia obraz wejściowy, na którym mają zostać wykryte kontury najpierw poddawany jest przetwarzaniu funkcją `cvCanny()` w celu uzyskania obrazu binarnego z krawędziami lub funkcją `cvThreshold()` (bądź `cvAdaptiveThreshold()`), która tworzy obraz binarny na podstawie progowania określona jasnością pikseli.

Poniżej przedstawiony rysunek pozwoli na łatwiejsze zrozumienie definicji kontury i sposobu działania funkcji `cvFindContours()`.



Rysunek 22 Zasada działania funkcji `cvFindContours()`.

Obraz po lewej stronie zawiera białe figury na szarym tle (obraz przed przetwarzaniem funkcją `cvFindContours()` zostanie poddany binaryzacji). Każda z tych figur jest oznaczona literą od A do E. Obraz po prawej stronie został poddany działaniu funkcji do wykrywania konturów. Kolorem czerwonym zaznaczone zostały zewnętrzne granice, zaś zielonym wewnętrzne granice białych figur. Kontury te również zostały oznaczone literami: zielone `hX` (od ang. „hole”), czerwone `cX` (od angl. „contour”), a `X` stanowi pewną liczbę. Ten sposób relacji zawierania pozwala na ustalenie wzajemnych powiązań pomiędzy konturami w tzw. drzewo konturów (ang. contours tree). W powyższym przypadku drzewo konturów rozpoczynałoby się od korzenia `c0` oraz „dziur” `h00` i `h01` będącymi głównymi odgałęzieniami, itd.

Definicja funkcji `cvFindContours()` wygląda następująco:

```
int cvFindContours(  
    IplImage* img,  
    CvMemStorage* storage,  
    CvSeq** firstContour,  
    int headerSize = sizeof(CvContour),  
    CvContourRetrievalMode mode = CV_RETR_LIST,  
    CvChainApproxMethod method = CV_CHAIN_APPROX_SIMPLE  
);
```

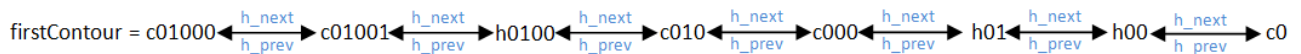
Pierwszy argument `img` jest obrazem wejściowym, który powinien być 8-bitowym, jednokanałowym obrazem binarnym. Drugi argument `storage` wskazuje miejsce, w którym została przydzielona pamięć na zapis sekwencji konturów. Pamięć ta powinna być wcześniej przydzielona za pomocą `cvCreateMemStorage()`. Kolejny argument `firstContour` jest wskaźnikiem na sekwencję `CvSeq*`, a dokładnie na początek drzewa konturów. Funkcja sama alokuje wskaźnik, dlatego jego wartość na początku powinna być ustawiona na `NULL`. Po wywołaniu funkcji zwraca sumę wszystkich znalezionych konturów.

Parametr `headerSize` wskazuje na rozmiar alokowanych obiektów (konturów). Następny argument `mode` przyjmuje jedną z czterech wartości: `CV_RETR_EXTERNAL`, `CV_RETR_LIST`, `CV_RETR_CCOMP` lub `CV_RETR_TREE`. Określa on sposób w jaki funkcja ma zwracać znalezione kontury. Poruszanie między znalezionymi konturami odbywa się z wykorzystaniem łączników `h_prev`, `h_next`, `v_prev`, `v_next`. Dla powyżej omawianego przykładu zostaną rozważone wymienione przypadki:

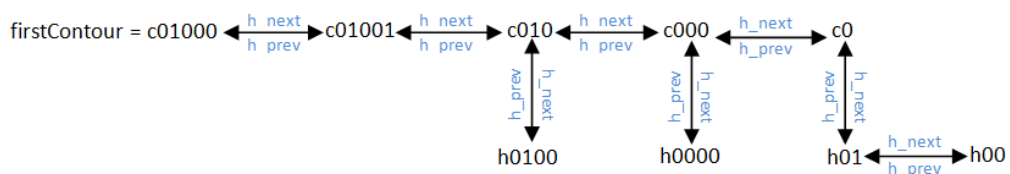
- `CV_RETR_EXTERNAL` zwraca tylko najbardziej zewnętrzny kontur obrazu:

`firstContour = c0,`

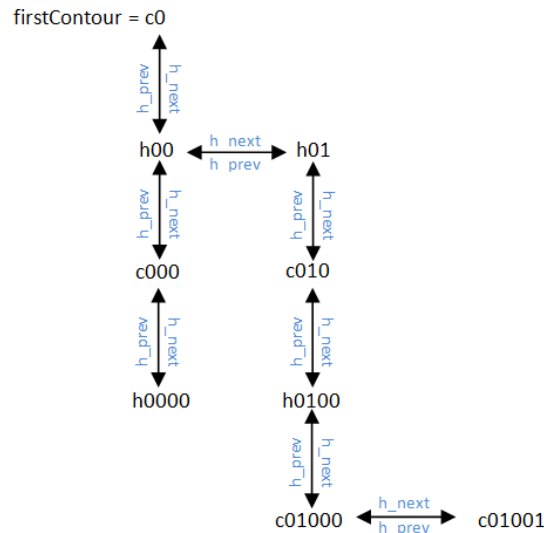
- `CV_RETR_LIST` zwraca wszystkie znalezione kontury połączone w listę łącznikami `h_prev` i `h_next`:



- `CV_RETR_CCOMP` zwraca wszystkie kontury w dwupoziomowej strukturze, gdzie wyższy poziom stanowią kontury najbardziej zewnętrzne a niższy stanowią „dziury”, połączone z konturami zewnętrznymi, w których się zawierają. Łączenie konturów odbywa się za pomocą wszystkich możliwych łączników:



- `CV_RETR_TREE` zwraca wszystkie kontury i układa je w hierarchiczne drzewo zagnieżdżonych konturów. Zgodnie ze wcześniejszym opisem początkiem drzewa jest kontur `c0` zawierający dwie „dziury” `h00` i `h01` na tym samym poziomie. Następnie „dziura” `h00` zawiera kontur `c000` oraz „dziura” `h01` zawiera kontur `c010`, itd. Najbardziej zagnieżdżone kontury znajdują się na samym dole drzewa:



Ostatni parametr `method` określa sposób aproksymacji konturów. Przyjmuje następujące wartości:

- `CV_CHAIN_CODE` wynikowe kontury zapisane są w postaci kodu łańcuchowego Freemana, pozostałe metody opierają się na wierzchołkach wieloboku będącego konturem. Dokładny opis łańcuchów Freemana znajduje się w rozdziale 3.6.2,
- `CV_CHAIN_APPROX_NONE` wynikowe kontury są przetwarzane z kodu Freemana na punkty będących wierzchołkami wieloboku,
- `CV_CHAIN_APPROX_SIMPLE` wynikowe kontury są kompresowane poziomo, pionowo oraz diagonalnie pozostawiając jedynie końcowe punkty,
- `CV_CHAIN_APPROX_TC89_L1` lub `CV_CHAIN_APPROX_TC89_KCOS` inny rodzaj aproksymacji łańcucha konturów,
- `CV_LINKS_RUNS` łączy poziome segmenty. Metoda ta jest dostępna tylko dla `CV_RETR_LIST`.

Należy pamiętać, że funkcja domyślnie przyjmuje parametr `mode = CV_RETR_LIST` oraz `method = CV_CHAIN_APPROX_SIMPLE`.

3.4.2 Operacje na konturach

Kontury są sekwencjami punktów stanowiących pewien rodzaj krzywej na obrazie. OpenCV dostarcza funkcje, które pozwalają na przetwarzanie tych łańcuchów punktów. Ich lista znajduje się poniżej:

```
CvContourScanner cvStartFindContours(
CvArr* image, CvMemStorage* storage,
int header_size = sizeof(CvContour),
int mode = CV_RETR_LIST, int method = CV_CHAIN_APPROX_SIMPLE,
CvPoint offset = cvPoint(0,0));
```

```
CvSeq* cvFindNextContour(CvContourScanner scanner);
```

```
void cvSubstituteContour(CvContourScanner scanner,
CvSeq* new_contour);

CvSeq* cvEndFindContour(CvContourScanner* scanner);
```

Funkcja `cvStartFindContours()` jest ściśle powiązana z `cvFindContours()` z tą różnicą, że wywoływana jest w momencie gdy analizowany jest pojedynczy kontur. Funkcja ta zwraca obiekt `scanner` typu `CvContourScanner` zawierający informację, o tym która część sekwencji konturów została odnaleziona, a która nie. Następnie wywołanie funkcji `cvFindNextContour()`, której argumentem jest zwrócony przez funkcję `cvStartFindContours()` `scanner`, pozwala na znalezienie wszystkich pozostałych konturów. W momencie gdy zwrócona zostanie wartość `NULL` oznacza to, że wszystkie kontury obrazu zostały odnalezione. Przeszukiwanie sekwencji można zakończyć funkcją `cvEndFindContour()` i wówczas do wskaźnika `scanner` zostaje przypisana pozycja do pierwszego elementu sekwencji.

3.4.3 Kod łańcuchowy Freemana

Zwykle kontury reprezentowane są za pomocą sekwencji wierzchołków wieloboku. Alternatywną reprezentacją jest kodowanie za pomocą kodu łańcuchowego Freemana, który można aktywować przy wywołaniu funkcji `cvFindContours()` ustawiając parametr `method` na `CV_CHAIN_CODE`. W rezultacie kontury będą reprezentowane za pomocą sekwencji kroków od pierwszego do ósmego kierunku, które są oznaczane liczbami całkowitymi od 0 do 7. Ten rodzaj kodowania często wykorzystywany jest przy programach do rozpoznawania kształtów [6]. Do pracy z łańcuchami Freemana OpenCV dostarcza dwie pomocnicze funkcje:

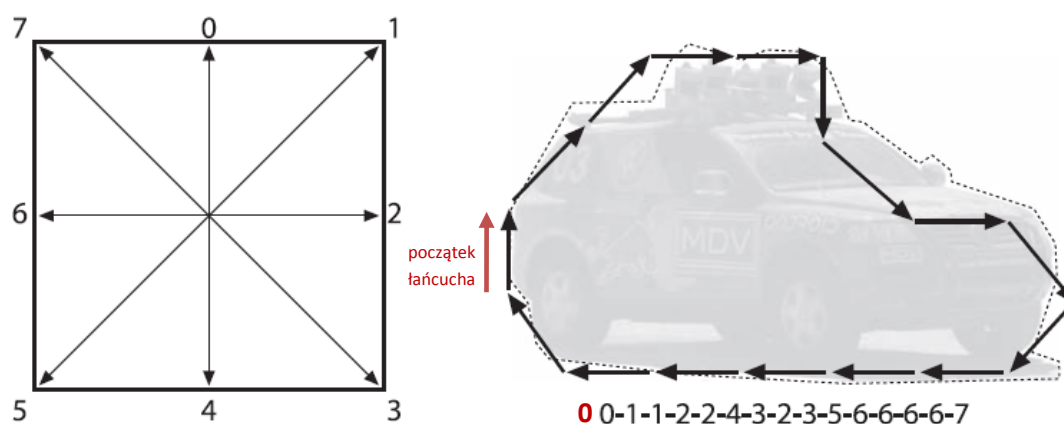
```
void cvStartReadChainPoints(
CvChain* chain, CvChainPtReader* reader);

CvPoint cvReadChainPoint( CvChainPtReader* reader);
```

Pierwsza funkcja odczytuje łańcuchy konturów. Pierwszym jej argumentem jest pierwszy odczytywany łańcuch `chain`⁸ oraz `reader`⁹ będący wskaźnikiem do aktualnie przetwarzanego elementu. `Reader` przechodzi pomiędzy pojedynczymi konturami reprezentowanymi przez łańcuchy. Jeżeli zwraca wartość `NULL` oznacza to, że wszystkie łańcuchy zostały odczytane. Druga funkcja ma za zadanie odczytać położenie aktualnie odczytywanego elementu łańcucha. Zasadę kodowania łańcuchowego Freemana przedstawia poniższy rysunek:

⁸ `CvChain*` jest odmianą sekwencji `CvSeq*`

⁹ można interpretować tak samo jak `scanner` w sekwencjach



Rysunek 23 Kod łańcuchowy Freemana

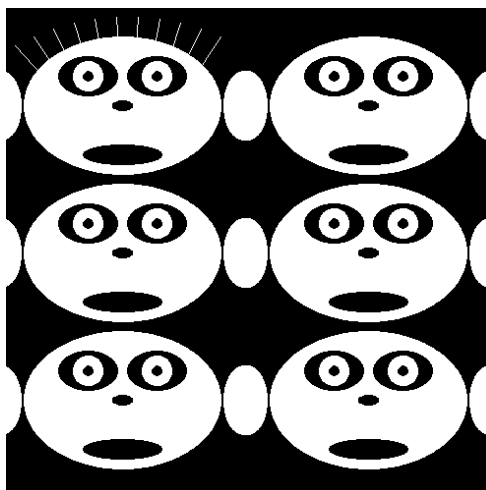
Na rysunku po lewej stronie przedstawiony został sposób kodowania kierunków liczbami całkowitymi. Rysunek po prawej pokazuje powstały łańcuch Freemana, przetłumaczony z konturów kierunkowych obiektu na obrazie.

3.4.4 Rysowanie konturów.

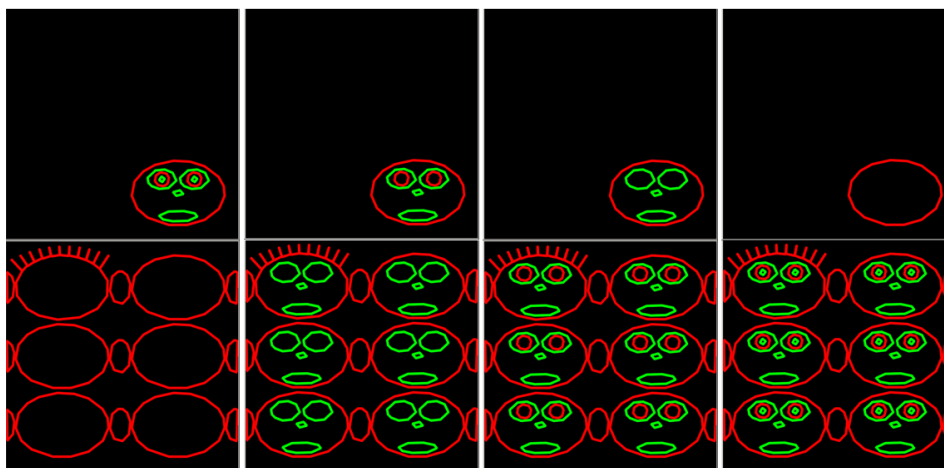
Rysowanie konturów odbywa się za pomocą funkcji `cvDrawContours()`:

```
void cvDrawContours(
    CvArr* img, CvSeq* contour, CvScalar external_color,
    CvScalar hole_color, int max_level, int thickness = 1,
    int line_type = 8, CvPoint offset = cvPoint(0,0) );
```

Pierwszy argument `img` jest obrazem, na którym kontury będą rysowane. Następny argument `contour` wskazuje na początek drzewa konturów (ang. root node). Skalary `external_color` oraz `hole_color` określają kolor konturu zewnętrznego oraz wewnętrznego. Parametr `max_level` jest bardzo istotny z punktu widzenia rysowania konturów i ich opisu. Określa, które kontury mają być rysowane tzn. jak głęboko zagnieżdżone względem najbardziej zewnętrznego konturu. Jeżeli `max_level = 0` wówczas rysowany jest tylko kontur wskazany przez wskaźnik `contour`. Dla `max_level = 1` rysowane są wszystkie kontury będące na tym samym poziomie co kontur wejściowy. Dla wartości 2 wszystkie kontury na pierwszym poziomie zagnieżdżenia w namalowanych konturach (przy `max_level = 1`), itd. W celu narysowania konturów zagnieżdżonych tylko w wejściowym konturze używa się ujemnych wartości parametru `max_level`. Dla wartości -1 rysowane są kontury z pierwszego poziomu zagnieżdżenia wejściowego konturu, dla -2 drugiego, itd. Parametry `thickness` oraz `line_type` określają sposób rysowania linii konturu. Ostatni parametr `offset` określa przesunięcie rysowanych konturów względem ich rzeczywistego położenia. Poniższe rysunki pokazują idee regulowania parametru `max_level`.



Rysunek 24 Obraz wejściowy przed wykryciem konturów.



Rysunek 25 Obraz wejściowy po wykryciu konturów. Dla parametru `max_level = -3, -2, -1, 0, 1, 2, 3` począwszy od obrazu w lewym górnym rogu¹⁰.

3.5 Rozmycie

Rozmycie (ang. smoothing) jest prostą i często stosowaną w przetwarzaniu obrazów operacją. Ma wiele zastosowań jednak głównie wykorzystywana jest do redukcji szumów oraz niepożądanych zniekształceń obrazu. W OpenCV funkcją realizującą rozmycie jest `cvSmooth()`. Za jej pomocą realizowane są następujące filtry: rozmywający prosty, prosty bez skalowania, medianowy, gaussowski oraz bilateralny:

```
void cvSmooth(
    const CvArr* src, CvArr* dst, int smoothtype = CV_GAUSSIAN,
    int param1=3, int param2=0, double param3=0, double param4=0);
```

gdzie `src` jest obrazem źródłowym, `dst` obrazem przetworzonym, `smoothtype` rodzajem filtru, argument `param1` i `param2` są rozmiarami maski, parametry `param3` i `param4` są różnie wykorzystywane w zależności od rodzaju rozmycia.

¹⁰ Obrazy zostały utworzone za pomocą programu demonstracyjnego `contours.exe` stanowiącego część biblioteki OpenCV 2.1

Poniżej zostaną opisane najbardziej znane i najczęściej używane filtry czyli: medianowy, gaussowski i bilateralny.

Filtr medianowy (CV_MEDIAN) wykorzystywany jest do usuwania losowych zakłóceń, których poziom intensywności piksela znacznie odbiega od intensywności pikseli sąsiednich. Filtr przetwarza dany piksel obrazu na medianę odpowiadającą ½ dystrybuanty jasności pikseli z całego rozważanego otoczenia (tj. maski) przetwarzanego piksela [4]. Biorąc pod uwagę, że dane obrazu mają postać dyskretną oraz, że suma elementów maski jest liczbą nieparzystą, mediana wyznaczana jest jako wartość środkowa wektora pikseli maski, ułożonych rosnąco lub malejąco, ze względu na jasność piksela. Filtr skutecznie usuwa zakłócenia typu „salt and peeper” nie powodując rozmazania obrazu. Operacje za pomocą tego filtru mogą być wykonywane na obrazach jedno-, trzy- lub czterokanałowych 8-bitowych.

Filtr Gaussa (CV_GAUSSIAN) wykorzystywany jest do rozmycia obrazu, usunięcia detali oraz pozbycia się zakłóceń. Działaniem przypomina filtr uśredniający jednak różni się od niego rodzajem maski, która ma odwzorowywać kształt funkcji rozkładu Gaussowskiego. Funkcja rozkładu Gaussowskiego dwuwymiarowego ma postać [7]:

$$G(x, y) = \frac{1}{2\pi^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Parametry funkcji cvSmooth() mają następujące odzwierciedlenie w powyższym wzorze:

param1 -> x, szerokość maski,
param2 -> y, wysokość maski,
param3 -> σ odchylenie standardowe.

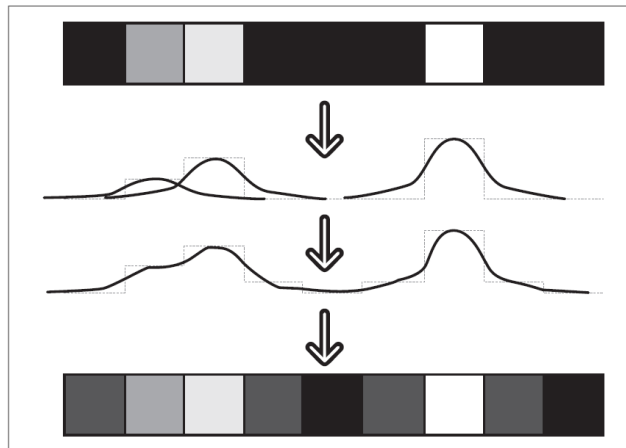
W momencie gdy odchylenie standardowe σ nie jest zdefiniowane funkcja cvSmooth() korzysta z zależności: $\sigma = \left(\frac{x}{2} - 1\right) \cdot 0.30 + 0.80$, lub $\sigma = \left(\frac{y}{2} - 1\right) \cdot 0.30 + 0.80$

Przykładowo dla $\sigma = 1$, maska aproksymująca dwuwymiarowy rozkład Gaussa ma postać:

	1	4	7	4	1
	4	16	26	16	4
$\frac{1}{273}$	7	26	41	26	7
	4	16	26	16	4
	1	4	7	4	1

Rysunek 26 Przykładowa maska filtru Gaussa

Następnie filtr Gaussa jest realizowany poprzez splot każdego punktu obrazu źródłowego z maską i sumowany w celu utworzenia obrazu docelowego. Wartość przetwarzanego piksela jest uzależniona od odległości Euklidesowej pomiędzy nim a innymi elementami maski. Poniższe rysunek najlepiej odzwierciedla jego działanie:



Rysunek 27 Zastosowanie filtru Gaussa dla jednowymiarowej macierzy pikseli [8]

W wersji warto zaznaczyć, że implementacja rozmycia Gaussowskiego w OpenCV, zawiera wbudowane maski o standardowych rozmiarach 3x3, 5x5 oraz 7x7 ze standardowym odchyleniem standardowym równym jeden, które dają bardzo dobre w sensie optymalizacyjnym rozmycie Gaussowskie. Należy pamiętać, że zwiększenie wartości parametru σ zmniejsza ilość zakłóceń występujących na obrazie, ale zmniejsza też dokładność wykrywania krawędzi na obrazie. Operacje te mogą być wykonywane na obrazach jedno lub trzykanałowych 8- lub 32-bitowych.

Zasadę działania ostatniego filtru bilateralnego najłatwiej jest przedstawić metodą porównawczą z filtrem Gaussa. Efekt działania tego drugiego filtru oparty jest tylko o jedną uśrednioną wagę, którą jest odległość euklidesowa. Filtr bilateralny jest jednym z filtrów nie rozmywających krawędzi. Możliwe jest to dzięki zastosowaniu drugiej wagi, którą jest nie odległość, a różnica w intensywności pomiędzy pikselem centralnym maski a jego otoczeniem.

Poniżej zamieszczone są obrazy poddane opisanym wyżej metodą filtracji:



Rysunek 28 Zastosowanie rozmycia a - medianowego, b – gaussowskiego

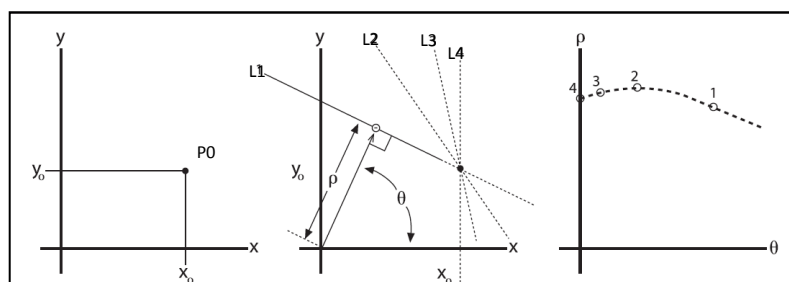
3.6 Transformata Hougha

Wymienione w poprzednich podrozdziałach sposoby wykrywania krawędzi są dobrym narzędziem w fazie pre-processingu, gdzie istotą jest wykrycie pikseli, które prawdopodobnie należą do krzywej będącej krawędzią. W efekcie często nie wykrywa się wszystkich pikseli należących do krawędzi oraz pojawiają się piksele, które znajdują się tylko w okolicy krawędzi. W momencie gdy ważne jest znalezienie kształtów opisywalnych równaniami takich jak linie proste, okręgi czy elipsy należy skorzystać z możliwości jakie daje Transformata Hougha.

3.6.1 Wykrywanie linii

Istnieje kilka algorytmów wykrywania linii za pomocą transformaty Hougha. Poniżej zostaną opisane te, które są zaimplementowane w bibliotece OpenCV.

Klasyczna transformacja Hougha dla linii prostych (ang. standard Hough Transform – STH) zakłada, że każdy punkt obrazu może być reprezentowany za pomocą krzywej sinusoidalnej.

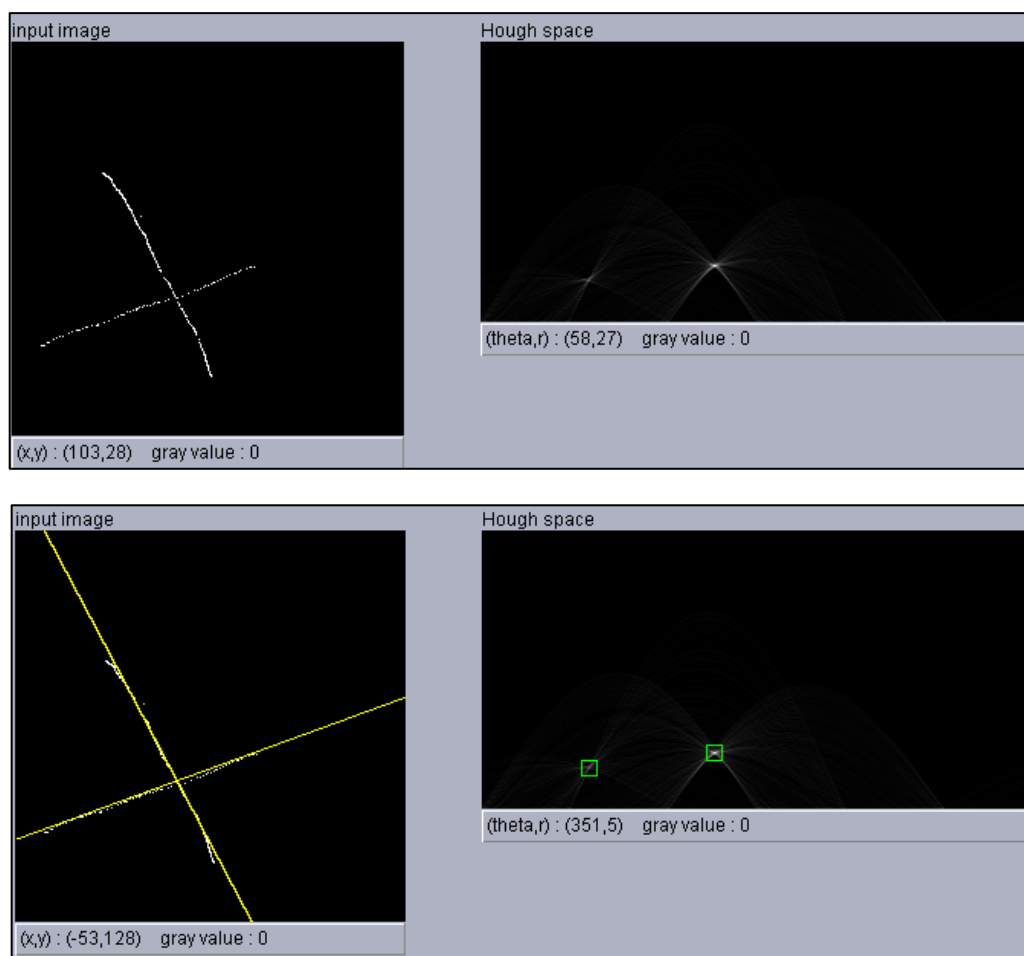


Rysunek 29 Realizacja klasycznej transformaty Hougha.

Na powyższym obrazie przez punkt P_0 o współrzędnych (x_0, y_0) przechodzi prosta L_1 . Prosta tą można opisać równaniem kierunkowym: $y = ax + b$. Prosta tą można zapisać w postaci parametrycznej: $r = x \cos \theta + y \sin \theta$, gdzie r jest odległością prostej od początku układu współrzędnych (wzdłuż normalnej), zaś θ jest kątem pomiędzy normalną do prostej a osią współrzędnych OX [9].

Powiązanie parametrów obu równań wygląda następująco: $a = -\frac{\cos \theta}{\sin \theta}$, oraz $b = \frac{r}{\sin \theta}$.

Następnie prosta L_1 obracana jest wokół punktu P_0 , powstające proste L_1, L_2, L_3, L_4 tworzą w przestrzeni zbiór punktów, który układa się na krzywej sinusoidalnej. Jeżeli na obrazie w przestrzeni (x, y) istnieje więcej punktów to w przestrzeni Hougha powstaje zbiór związanych z nimi krzywych. Punkty leżące na tej samej prostej w przestrzeni (x, y) tworzą sieć krzywych przecinających się w jednym punkcie w przestrzeni Hougha. Punkt ten jest lokalnym maksimum odpowiadającym prostej w przestrzeni (x, y) .



Rysunek 30 Realizacja transformaty Hougha dla linii prostych. Obrazy przestrzeni (x, y) oraz przestrzeni parametrycznej (θ, r) ¹¹

Zgodnie z tym co pokazują powyższe rysunki każdą prostą w przestrzeni (x, y) można odwzorować w punkt o współrzędnych (θ, r) w przestrzeni Hougha.

Progresywna Probabilistyczna transformata Hougha (ang. progressive probabilistic Hough transform – PPHT), jest modyfikacją klasycznej metody. Różnica polega między innymi na tym, że obliczane są dodatkowo zakresy linii, a nie tylko jej orientacja. Metoda ta ma w nazwie probabilistyczna, ponieważ nie bada wszystkich punktów z przestrzeni (x, y) , które składają się na lokalne maksimum w przestrzeni Hougha. Wystarczająco duże skupisko przecinających się krzywych w przestrzeni Hougha interpretowane jest jako istnienie linii w przestrzeni (x, y) . Takie założenie pozwala na zmniejszenie ilości obliczeń potrzebnych do detekcji linii na obrazie.

Skalowalna transformata Hougha (ang. multi-scale Hough transform – MSHT) daje możliwość zmiany dokładności wykrywania linii za pomocą zmiany rozmiarów akumulatora (parametrów ρ i θ).

Wszystkie te algorytmy są dostępne za pomocą jednej funkcji `cvHoughLines2()` i używane w zależności od wartości jej parametrów:

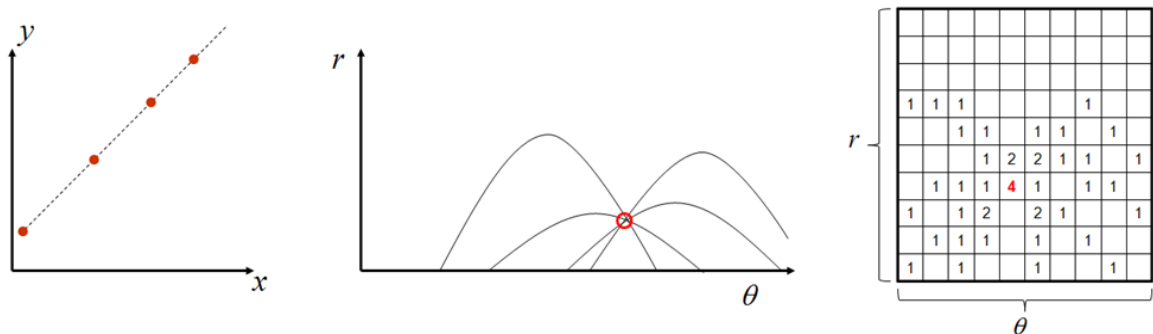
¹¹Zródło obrazu (wygenerowany z Apletu Java): [W8]

```
CvSeq* cvHoughLines2(CvArr* image,void* line_storage,
int method, double r, double theta, int threshold,
double param1 = 0,double param2 = 0 );
```

Pierwszy argument jest obrazem wejściowym. Musi być to obraz 8-bitowy, który jest traktowany jako binarny (każdy niezerowy piksel jest traktowany tak samo) [8]. Drugi argument jest wskaźnikiem do miejsca, w którym wykryte linie będą przechowywane. Alokowane mogą być za pomocą opisywanej wcześniej struktury CvMemoryStorage lub jako tablica Nx1, gdzie N oznacza ilość znalezionych linii. Kolejny parametr metody może przyjmować następujące wartości w zależności od rodzaju używanego algorytmu:

- CV_HOUGH_STANDARD, dla klasycznej transformaty Hougha,
- CV_HOUGH_PROBABILISTIC, dla probabilistycznej transformaty Hougha,
- CV_HOUGH_MULTI_SCALE, dla trzeciego algorytmu będącego modyfikacją klasycznej transformaty (skalowalną).

Kolejne dwa argumenty: r i θ określają rozdzielczość (dokładność) z jaką mają być wyznaczone linie na obrazie. Dokładność tą można rozumieć jako rozmiar dwuwymiarowej macierzy zwanej akumulatorem. W jej komórki o rozmiarach θ x r wpisywane zostają punkty, które są interpretacją linii na obrazie z przestrzeni (θ).



Rysunek 31 Zasada wypełniania elementów akumulatora.

Parametr `threshold` jest progiem, który musi osiągnąć komórka akumulatora żeby została zaklasyfikowana jako linia na obrazie. Należy pamiętać o skalowaniu tej wartości w zależności od parametrów akumulatora. Argumenty `param1` i `param2` nie są używane w przypadku SHT. Dla PPHT `param1` określa minimalną długość segmentu linii, która ma zostać znaleziona. `Param2` określa natomiast minimalną odległość pomiędzy poziomymi segmentami linii, przy której nie są łączone na obrazie przetworzonym. W przypadku algorytmu Multi-scale HT `param1` i `param2` są dodatkowymi wartościami skalującymi rozdzielczość akumulatora zgodnie ze wzorami:

$$\rho := \frac{\rho}{\text{param1}} \text{ oraz } \theta := \frac{\theta}{\text{param2}}.$$

3.6.2 Wykrywanie okręgów

Okrąg może być opisany za pomocą dwóch równań parametrycznych:

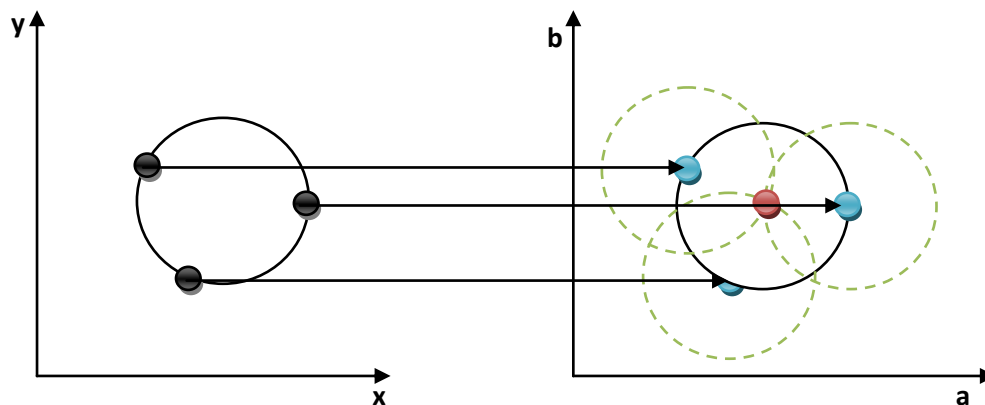
$$\begin{aligned}x &= a + R\cos\theta \\ y &= b + R\sin\theta\end{aligned}$$

Transformata Hougha do wykrywania okręgów (ang. Circle Hough Transform – CHT) potrzebuje więc trójkę (x, y, θ) . W tym wypadku akumulator musiałby być tablicą trójwymiarową. Praca na takich tablicach pochłania sporą ilość mocy obliczeniowych i często doprowadza do przekłamań i błędów w obliczeniach. W takim wypadku, aby sprowadzić tablicę akumulatora do postaci dwuwymiarowej należy jeden z parametrów ustalić jako stały.

Zakładając, że szukane są okręgi o konkretnym promieniu R równania okręgu można zapisać w postaci:

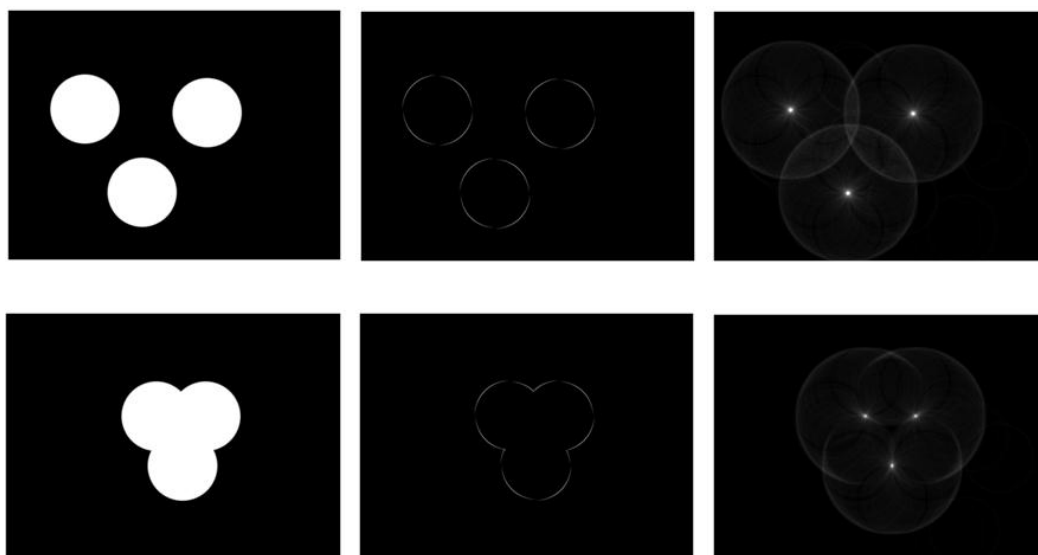
$$\begin{aligned}a &= x_o - R\cos\theta \\ b &= y_o - R\sin\theta\end{aligned}$$

gdzie (x_o, y_o) jest punktem należącym do okręgu. Wówczas punkt w przestrzeni (x, y) będzie reprezentowany jako okrąg w przestrzeni (a, b) .



Rysunek 32 Wykrywanie okręgów dla CHT przy znanym promieniu R .

Środek okręgu w przestrzeni (x, y) będzie częścią wspólną wszystkich okręgów w przestrzeni (a, b) . Poniżej przedstawione zostały poszczególne etapy detekcji środka okręgu na obrazie wejściowym:



Rysunek 33 Programowa detekcja środka okręgu dla CHT.

Binarny obraz źródłowy poddawany jest filtracji Sobela (`cvSobel()`) w celu detekcji krawędzi okręgów. Następnie tworzone są okręgi o takim samym promieniu i środku w każdym pikselu należącym do krawędzi. W efekcie powstał trzeci rysunek, na którym widać dokładnie lokalne maksima będące środkami okręgów z obrazu binarnego. W obu przypadkach: gdy okręgi były rozłączne oraz gdy nachodziły na siebie, algorytm dosłownie sobie poradził z wyznaczeniem ich środków.

W momencie gdy promień poszukiwanego okręgu należy do pewnego przedziału algorytm postępowania jest podobny. Różnica polega jedynie na tym, że dla każdego piksela wykrytej krawędzi tworzone są okręgi o promieniach z zadanego przedziału, przy zadanej rozdzielczości.

Wyżej przedstawiony algorytm jest skuteczny dla bardzo wyraźnych i prostych w detekcji okręgów na obrazach. W przypadku gdy okręgi nie są idealne, zaszumienie na obrazie znacznie większe, algorytm ten nie jest w stanie poradzić sobie z poprawną detekcją środków okręgów.

W OpenCV zaimplementowany jest bardziej rozbudowany algorytm, który jest zdecydowanie bardziej odporny na zakłócenia. Na początku obraz filtrowany jest za pomocą `cvCanny()` w celu detekcji krawędzi. Następnie dla każdego niezerowego piksela obrazu liczone są lokalne gradienty za pomocą funkcji `cvSobel()`. Korzystając z kierunków tych gradientów do akumulatora zapisywane są piksele leżące w tym kierunku pomiędzy minimalną a maksymalną odległością od piksela, w którym ten gradient był liczony. Następnie w akumulatorze za pomocą progowania usuwane są piksele, które znajdują się poniżej określonej wartości. Pozostałe piksele sortowane są malejąco. Te, które znajdują się na początku tej listy są najlepszymi kandydatami do uznania za środek poszukiwanego okręgu. Na koniec tworzone są okręgi o środkach w pikselach kandydatach i promieniach z zadanego przez funkcję przedziału. Po tej operacji na środek okręgu wybierany jest piksel, z którego został wyprowadzony okrąg o krawędziach obejmujących największą liczbę pikseli należących do krawędzi obrazu po przefiltrowaniu funkcją `cvCanny()`.

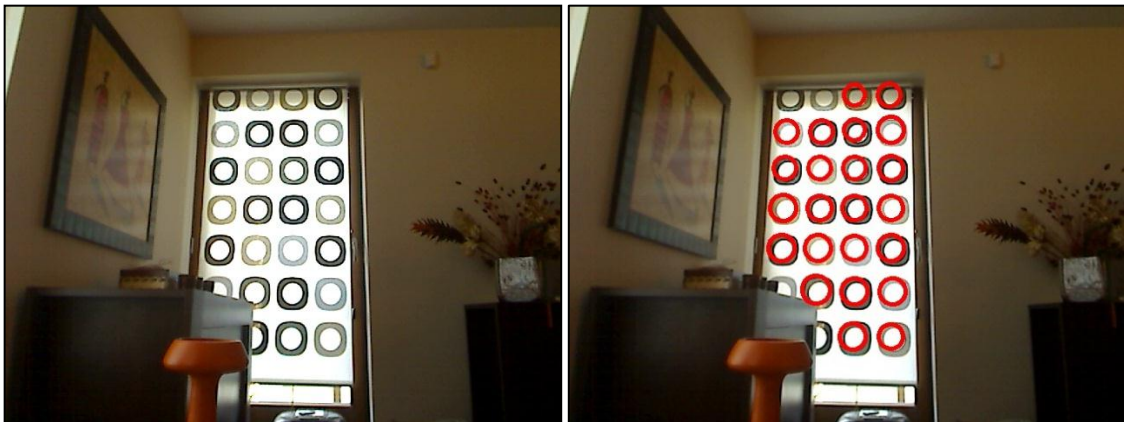
Powyższy algorytm realizuje funkcja `cvHoughCircles()`:

```

CvSeq* cvHoughCircles(CvArr* image,void* circle_storage,
int method,double dp,double min_dist,double param1 = 100,
double param2 = 300,
int min_radius = 0,
int max_radius = 0
);

```

Parametry tej funkcji są podobne jak w przypadku funkcji `cvHoughLines2()`. Wejściowy obraz `image` musi być 8-bitowy lecz w tym przypadku nie musi być binarny, gdyż wewnątrz funkcji `cvHoughCircles()` realizowana jest funkcja `cvSobel()`, która może przyjąć jako argument obraz w skali szarości. Drugi parametr `circle_storage`, to tak jak wcześniej `line_storage` jest wskaźnikiem do miejsca, w którym przechowywane będą przechwytywne okręgi. Ponownie alokowane mogą być w strukturze `CvMemoryStorage` lub w macierzy. Argument `method` zawsze równy jest `CV_HOUGH_GRADIENT`, czyli odpowiada metodzie, która została opisana powyżej. Parametr `dp` jest rozdzielczością akumulatora obrazu z przestrzeni Hougha. Wartość 1 tego parametru określa rozdzielczość akumulatora jako taką samą co obraz, wartości powyżej jeden zmniejszają tę rozdzielczość w stosunku do obrazu. Parametr `min_dist` jest minimalną odległością pomiędzy środkami wykrytych okręgów, żeby nie były one interpretowane jako jeden okrąg. Parametry `param1` i `param2` są dolnym i górnym progiem parametru `threshold` z funkcji `cvCanny()`. Ostatnie dwa parametry `min_radius` i `max_radius` określają minimalny i maksymalny promień poszukiwanych okręgów. Poniżej zamieszczony jest przykład działania tej funkcji na obrazie rzeczywistym:



Rysunek 34 Detekcja okręgów na obrazie za pomocą CHT.

Okręgi zostały wykryte na podstawie poniższego kodu:

```

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

#pragma comment(lib, "highgui210")
#pragma comment(lib, "cv210")

using namespace std;

int main()
{
    //ładowanie obrazu image oraz tworzenie obrazu gray
    IplImage* image = cvLoadImage("C:\\okno2.jpg",1);
    IplImage* gray = cvCreateImage(cvGetSize(image), 8, 1);

```

```

// alokowanie pamięci dla sekwencji
CvMemStorage* storage = cvCreateMemStorage(0);
//tworzenie okien pod wynik operacji
cvNamedWindow("overview");
cvNamedWindow("cvHoughCircles");
//konwersja wejściowego obrazu kolorowego do 8 bitowego w skali szarości
cvCvtColor(image, gray, CV_RGB2GRAY);
//zapisywanie sekwencji okręgów wykrytych przez CHT
CvSeq* results = cvHoughCircles(gray,storage,CV_HOUGH_GRADIENT,2,10,
40,120,9,17);
//pętla rysująca okręgi zapisane w sekwencjach za pomocą współrzędnych
środką okręgu oraz długości promienia
for( int i = 0; i < results->total; i++ )
{
    float* p = (float*) cvGetSeqElem( results, i );
    CvPoint pt = cvPoint( cvRound( p[0] ), cvRound( p[1] ) );
    cvCircle(image,pt,cvRound( p[2] ),CV_RGB(255,0,0),3);
}
//otwieranie okien z wynikami operacji
cvShowImage( "cvHoughCircles", gray);
cvSaveImage("circless.jpg",image);
cvShowImage("overview", image);
//okna zamykane są dowolnie naciśniętym klawiszem
cvWaitKey(0);
//usuwanie obiektów
if(image)
    cvReleaseImage(&image);
if(gray)
    cvReleaseImage(&gray);
return 0;
}

```

3.7 Przekształcenia morfologiczne

OpenCV dostarcza szybkie algorytmy realizujące transformacje morfologiczne obrazów. Przekształcenia morfologiczne są to przekształcenia, które powodują, że dany piksel obrazu jest modyfikowany tylko wtedy, gdy spełniony jest zadany warunek logiczny [4]. Podstawowym pojęciem przy operacjach morfologicznych jest element strukturalny, będący wycinkiem obrazu, który zawiera punkt centralny. Jego kształt jest porównywany z lokalną konfiguracją badanego piksela i dopiero w momencie jej spasowania realizowana jest na pikselu określona funkcja.

OpenCV daje możliwość tworzenia własnego elementu strukturalnego będącego typem `IplConvKernel`. Konstruowane są za pomocą funkcji `cvCreateStructuringElementEx()` oraz usuwane funkcją `cvReleaseStructuringElement()`:

```

IplConvKernel* cvCreateStructuringElementEx(
int cols, int rows, int anchor_x, int anchor_y, int shape,
int* values=NULL);

void cvReleaseStructuringElement( IplConvKernel** element );

```

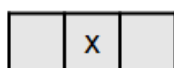
Elementy strukturalne (`element`), w przeciwieństwie do masek wykorzystywanych przy splotach, nie potrzebują żadnych danych numerycznych. W momencie tworzenia

elementu parametry `rows` i `cols` określają rozmiar prostokąta, w którym się znajduje. Współrzędne `anachor_x` oraz `anachor_y` definiują pozycję punktu centralnego, kolejny parametr `shape` określa kształt elementu strukturalnego:

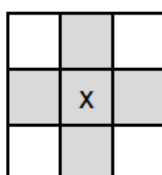
CV_SHAPE_RECT - prostokątny,
 CV_SHAPE_CROSS – krzyżowy,
 CV_SHAPE_ELLIPSE – eliptyczny,
 CV_SHAPE_CUSTOM – możliwość stworzenia własnego kształtu.

Wskaźnik `values` do danych elementu strukturalnego, określa które piksele z sąsiedztwa są brane pod uwagę. Element strukturalny jest macierzą, której niezerowe wartości definiują obszar kształtu. Jeżeli wskaźnik jest równy `NULL` oznacza, to że wszystkie elementy są niezerowe i wówczas element strukturalny ma kształt prostokąta.

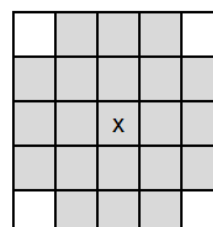
Poniżej zamieszczone zostały przykładowe kształty elementu strukturalnego:



Rysunek 37 Element strukturalny w kształcie prostokąta.



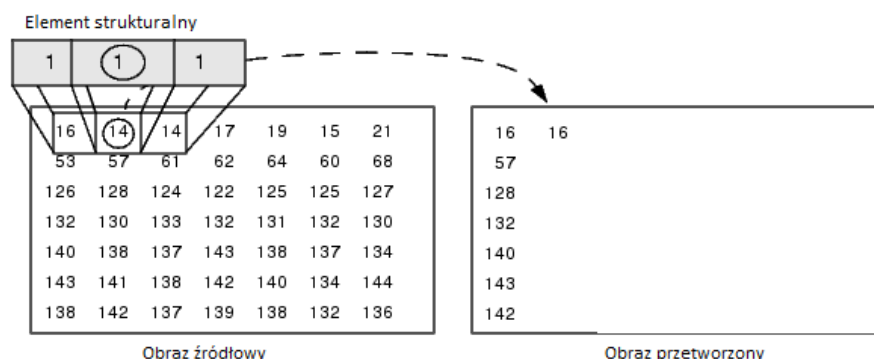
Rysunek 36 Element strukturalny w kształcie prostokąta.



Rysunek 35 Element strukturalny w kształcie okręgu.

Podstawowymi przekształceniami morfologicznymi są dylatacja i erozja. Poniższe operacje zostaną opisane dla obrazów w skali szarości. Często stosuje się je również dla obrazów czarno-białych, które zostały przetworzone za pomocą progowania. Idee działania algorytmów jest taka sama, dlatego zostanie opisana dla ogólniejszego przypadku.

Dylatacja polega na przesuwaniu punktu centralnego elementu strukturalnego po brzegach obszaru. W tym czasie wybierana jest maksymalna wartość piksela obejmowanego przez element strukturalny i przypisywana jest do wartości piksela centralnego (realizacja funkcji maksimum).



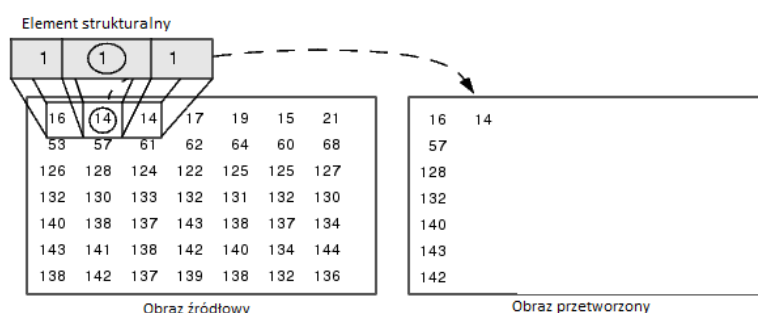
Rysunek 38 Dylatacja. Obliczanie wartości piksela za pomocą elementu strukturalnego

Operacja ta powoduje, że elementy przetwarzanego obrazu łączone są w obiekty, usuwane są drobne wklęsłości w wyróżnionych obszarach oraz wygładzane są brzegi.



Rysunek 39 Wykonanie operacji dylatacji na obrazie

Erozja jest operacją odwrotną do dylatacji. Element strukturalny jest przesuwany wewnątrz obszaru. W tym czasie wybierana jest minimalna wartość piksela obejmowanego przez element strukturalny i przypisywana jest do wartości piksela centralnego (realizacja funkcji minimum).



Rysunek 40 Erozja. Obliczanie wartości piksela za pomocą elementu strukturalnego

Operacja ta powoduje, że elementy przetwarzanego obrazu mają wygładzone brzegi i usuwane są drobne szczegóły. Erozja usuwa odizolowane, drobne wyróżnione obszary, czyli szумы.



Rysunek 41 Wykonanie operacji erozji na obrazie rzeczywistym

W OpenCV te operacje wykonywane są funkcjami `cvDilate()` oraz `cvErode()`:

```
void cvErode(IplImage* src, IplImage* dst, IplConvKernel* B = NULL,
int iterations = 1 );
void cvDilate(IplImage* src, IplImage* dst, IplConvKernel* B = NULL,
int iterations = 1 );
```

Obie te funkcje posiadają takie same parametry. `src` jest obrazem źródłowym, zaś `dst` obrazem przetworzonym. Argument `iterations` odpowiada za ilość procesów przeprowadzanego algorytmu, ostatni z parametrów `B`, będący obiektem `IplConvKernel*` jest elementem strukturalnym (dokładnie zostanie opisany w końcowej części rozdziału). Wartość domyślna `NULL` oznacza, że obiekt strukturalny ma wymiary 3x3 z punktem centralnym na pozycji (2, 2).

Biblioteka daje możliwość wykorzystywania złożonych operacji morfologicznych takich jak:

- otwarcie – `CV_MOP_OPEN`,
- zamknięcie – `CV_MOP_CLOSE`,
- gradient morfologiczny – `CV_MOP_GRADIENT`,
- detekcja maksimum lokalnego – `CV_MOP_TOPHAT`,
- detekcja minimum lokalnego – `CV_MOP_BLACKHAT`.

Funkcją, która obsługuje te operacje jest `cvMorphologyEx()`:

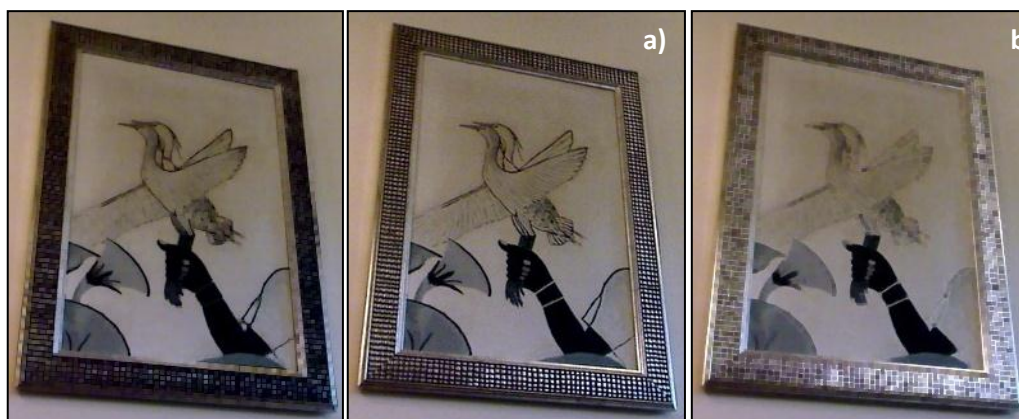
```
void cvMorphologyEx(
const CvArr* src, CvArr* dst, CvArr* temp,
IplConvKernel* element, int operation, int iterations = 1);
```

Parametry `src`, `dst`, `element` oraz `iteration` zostały opisane wcześniej. Nowym elementem jest parametr `temp`, jest obrazem wykorzystywanym przy operacjach gradientu morfologicznego oraz detekcji ekstremów, w których konieczne jest użycie obrazu pośredniego (musi mieć identyczne parametry jak obraz `src` i `dst`). Najważniejszy parametr `operation` określa rodzaj przeprowadzanej operacji (wartości tego parametru zostały opisane podczas wyliczenia). Erozja i dylatacja są operacjami typu *in-place* tzn. przetwarzany obraz jest obrazem źródłowym.

Pierwsza operacja otwarcie jest złożeniem erozji z dylatacją. Za pomocą tego rodzaju filtracji duże obiekty na obrazie nie zmieniają swoich wymiarów oraz kształtów. Usuwane są drobne obiekty, oraz rozsuwane obiekty z cienkimi połączeniami. Często stosowana jest w celu wyodrębnienia elementów obrazu i ich policzenia – na przykład liczenie komórek na obrazie z pod mikroskopu.

Zamknięcie jest operacją odwrotną do otwarcia. Najpierw wykonywana jest dylatacja a następnie erozja. W tym przypadku duże obiekty o wygładzonych brzegach nie zmieniają swojego kształtu. Natomiast wypełniane są wszelkiego rodzaju dziury, wąskie szczeliny oraz wcięcia. Można powiedzieć, że obiekty są wypełnianie oraz łączone ze sobą.

Poniżej przedstawione zostały efekty filtracji otwarciem i zamknięciem obrazu wejściowego:



Rysunek 42 Wykonanie operacji a - otwarcia i b – zamknięcia na obrazie rzeczywistym

Następną operacją jest gradient morfologiczny. Zasadę działania tej filtracji najlepiej przedstawia wzór [10]:

$$\text{gradient}(\text{src}) = \text{dilate}(\text{src}) - \text{erode}(\text{src})$$

Operacja ta odbywa się tylko na Jak widać gradientem nazywamy różnicę dwóch obrazów: pierwszego poddanego dylatacji i drugiego po erozji. Głównie wykorzystywana jest do detekcji krawędzi [11]. Operacja pozwala na analizę szybkości zmian jasności pikseli (stąd nazwa gradient morfologiczny). Wyznacza granicę pomiędzy obszarami o wyższej jasności.



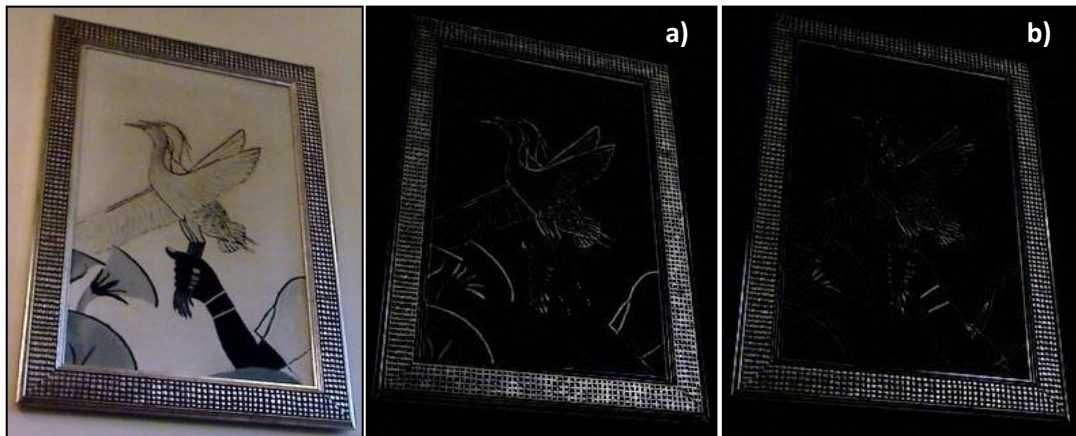
Rysunek 43 Wykonanie operacji gradientu morfologicznego na obrazie rzeczywistym

Ostatnie dwie operacje detekcja minimum i maksimum lokalnego mają za zadanie wyodrębnić skrawki obrazu, które są znacząco jaśniejsze lub ciemniejsze od ich najbliższych sąsiadów. Używa się tych operacji w celu wyizolowania części obiektu, która widocznie różni się jasnością w stosunku do obiektu, w którym się znajduje. Operacje te mogą być opisane w sposób następujący:

$$\text{maximum} : \text{TopHat}(\text{src}) = \text{src} - \text{open}(\text{src})$$

$$\text{minimum} : \text{BlackHat}(\text{src}) = \text{close}(\text{src}) - \text{src}$$

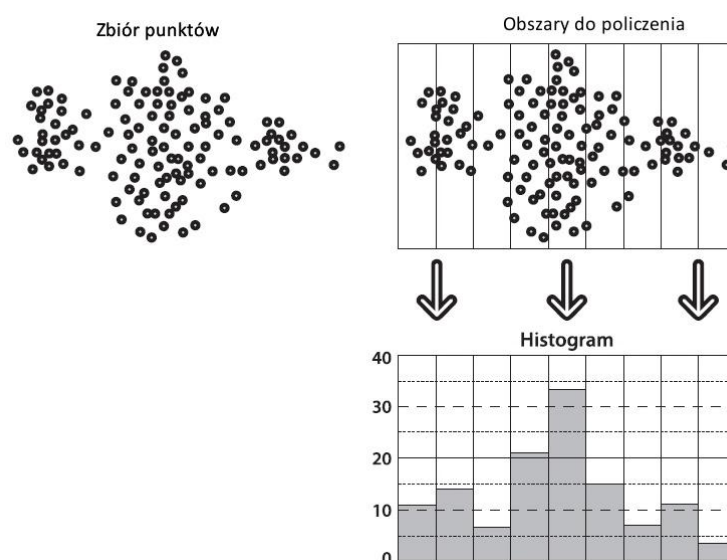
Morfologia TopHat izoluje lokalne obszary o dużej jasności pikseli. Najpierw wynik otwarcia jest odejmowany od obrazu źródłowego, a następnie przeprowadza się binaryzację z dolnym progiem otrzymanej różnicy. Morfologia BlackHat wykrywa lokalne obszary o niskiej jasności. Realizowane jest to poprzez odejmowanie obrazu źródłowego od wyniku zamknięcia.



Rysunek 44 Wykonanie morfologii a - TopHat, b – BlackHat na obrazie rzeczywistym

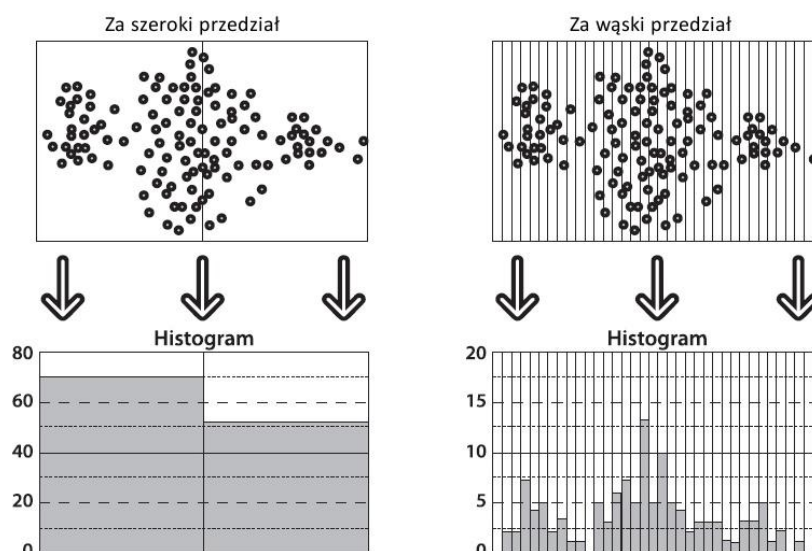
3.8 Histogramy

Histogram obrazu jest charakterystyką jego rozkładu jasności, kolorów lub innych cech obrazu. Zawiera przedziały (kosze – ang. bins), których wartość najczęściej wskazuje na ilość występowania pikseli o jasności określonej dla każdego przedziału. Jednak charakterystyka i to co pokazuje histogram jest zależne tylko od użytkownika. Histogramy mogą pokazywać rozkład kolorów, jasności, umiejscowienia pikseli. Poniżej pokazany jest przykład histogramu, który bada ilość występowania pikseli na obrazie w zadanym przedziale zdefiniowanym przez histogram.



Rysunek 45 Prosty przykład zastosowania histogramu

Istotnym czynnikiem wpływającym na czytelność i przydatność histogramu jest odpowiedni dobór szerokości (zakresu) koszy. Poniżej pokazany jest typowy problem związany z błędnym doбором wartości z przedziału.



Rysunek 46 Przykład błędnego oszacowania szerokości koszy histogramu

Jak widać nieprawidłowy dobór szerokości przedziału generuje trudny do interpretacji rozkład (w tym przypadku) punktów od lewej do prawej strony obrazu.

Opisywane powyżej histogramy są jednowymiarowe. Istnieją również odpowiedniki dwu oraz trójwymiarowe. Wszystkie z nich są wspierane przez OpenCV. Histogram w tej bibliotece jest reprezentowany przez specjalny typ danych `CvHistogram`. Histogram tworzony jest za pomocą funkcji `cvCreateHist()`:

```
CvHistogram* cvCreateHist(int dims,int* sizes,int type,
float** ranges = NULL, int uniform = 1);
```

Pierwszy argument `dims` określa wymiar histogramu (1D, 2D, 3D). Parametr `sizes` wskazuje na tablicę o takiej samej ilości liczb całkowitych co wymiar histogramu, które deklarują ilość koszy (przedziałów) dla każdego wymiaru histogramu. Następnie argument `type` wskazuje na rodzaj zastosowanego histogramu `CV_HIST_SPARSE` lub `CV_HIST_ARRAY` (najczęściej używany). Kolejny parametr `ranges` jest wskaźnikiem do tablicy dwuwymiarowej, której aktualne wartości zależą od następnego parametru `uniform`:

- jeżeli `uniform = 1`, wówczas w tablicy muszą znaleźć się dwie liczby na jeden wymiar, będące zakresem obsługiwanych wartości histogramu. Czyli np. jeśli dla obrazu w skali szarości poziomy jasności są z zakresu 0 do 100 te dwie liczby wpisywane są do macierzy. W przypadku gdy zadeklarowana ilość koszy wynosi 10 to kolejno będą zawierały piksele o jasności z przedziałów [0,10), [10,20), itd.,
- jeżeli `uniform = 0`, wówczas w tablicy musi znaleźć się $N+1$ liczb na jeden wymiar, stanowiących granice zakresów wartości koszyków. Dla podobnego przypadku,

przy 4 koszykach ($N=4$), zakresy kształtowałyby się kolejno $[Z1, Z2)$, $[Z2, Z3)$, $[Z3, Z4)$, $[Z4, Z5]$.

Po utworzeniu można obliczyć histogram z obrazu lub obrazów źródłowych za pomocą funkcji `cvCalcHist()`:

```
void cvCalcHist(IplImage** image, CvHistogram* hist,  
int accumulate = 0, const CvArr* mask = NULL );
```

Parametr `image` jest obrazem lub obrazami źródłowymi, `hist` stworzony wcześniej pustym histogramem. Argument `accumulate` równy 0, powoduje wyzerowanie histogramu przed rozpoczęciem obliczania. Ostatni parametr `mask` pozwala na pominięcie części obrazu: z obrazu wejściowego brane są pod uwagę tylko te piksele, które trafiają na niezerowe elementy maski.

Należy pamiętać, że funkcja ta wymaga, aby obraz był w skali szarości lub binarny. Obrazy wielokanałowe należy rozdzielić funkcją `cvSplit()` i liczyć histogramy dla każdego z kanałów.

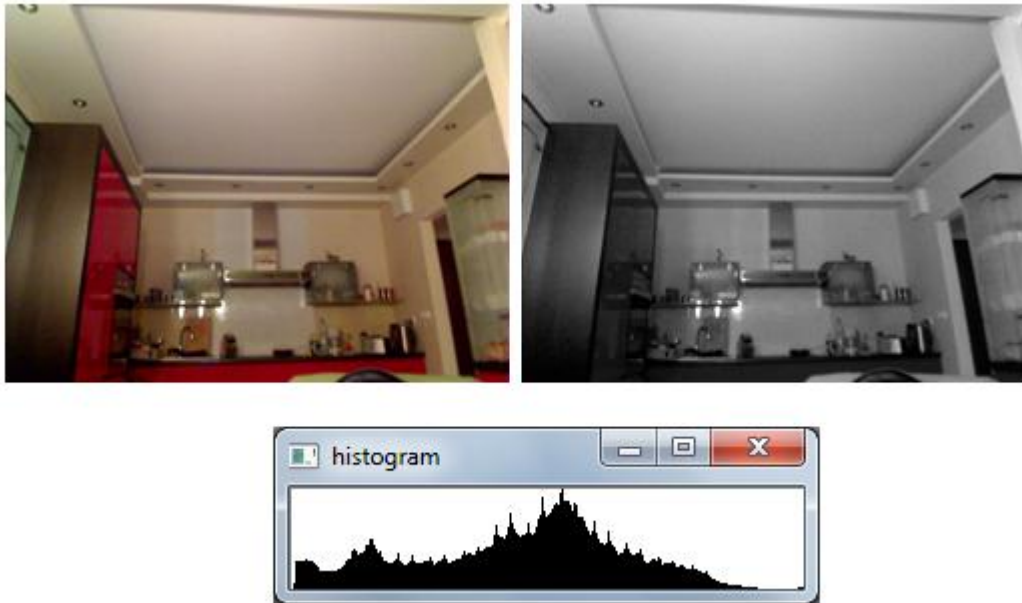
W momencie gdy histogram nie jest już używany należy go usunąć, aby zwolnić zasoby zajmowanej pamięci. Odbывается to za pomocą funkcji `cvReleaseHist()`. W momencie gdy stworzony histogram chcemy ponownie wykorzystać przed jego usunięciem, można użyć funkcji `cvClearHist()`, która zeruje wartości w koszykach histogramu pozwalając na jego ponowne użycie.

Histogram byłby mało przydatny, gdyby użytkownik nie miał dostępu do wartości z koszyków. Proces ten w OpenCV odbywa się na dwa sposoby, w zależności od potrzeb. Zdecydowanie prostszym sposobem jest dostęp bezpośredni za pomocą funkcji `cvQueryHistValue_1D()`:

```
double cvQueryHistValue_1D(CvHistogram* hist, int idx0);
```

Funkcja ta po podaniu numeru indeksu kosza zwraca wartość przez niego przechowywaną. Takie same funkcje tylko ze zmienionym indeksem wymiaru stosowane są dla histogramów 2D i 3D. Funkcja ta jest przydatna w momencie gdy rzadko sięga się po wartości z koszy. W momencie gdy wartości te muszą być analizowane na bieżąco należy skorzystać z funkcji `cvGetHistValue_1D()`, które zwracają wskaźnik na dany kosz.

OpenCV nie zawiera wbudowanych narzędzi do rysowania histogramu jednak napisanie jej nie stanowi większego problemu dla osób średniozaawansowanych. Przykładowy sposób wygenerowania histogramu odcieni szarości na rysunku 47. w OpenCV.



Rysunek 47 Programowa realizacja histogramu w OpenCV

Histogram został wygenerowany na podstawie poniższego kodu:

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

#pragma comment (lib, "cv210")
#pragma comment (lib, "highgui210")

IplImage* image= 0;
IplImage* imgHistogram = 0;
IplImage* gray= 0;

CvHistogram* hist;

int main(){

    image = cvLoadImage("kuchnia.jpg",1);

    //okreslenie rozmiaru histogramu (ilości koszy)
    int size[] = {256};

    //wartość kosza i jego znormalizowana wartość
    float val;
    int norm;

    //zakres jasności
    float range[] = { 0, 256 };
    float* ranges[] = { range };

    //wartość minimalna i maksymalna histogramu
    float min_val = 0, max_val = 0;

    //konwersja wejściowego obrazu kolorowego do 8 bitowego w skali szarości
    gray = cvCreateImage( cvGetSize(image), 8, 1 );
    cvCvtColor( image, gray, CV_BGR2GRAY );

    //Tworzenie okien do pokazania rezultatu
```

```

cvNamedWindow("original",1);
cvNamedWindow("gray",1);
cvNamedWindow("histogram",1);

//obrazy z których zostanie obliczony histogram
IplImage* images[] = { gray };

// przygotowanie pustego histogramu
hist = cvCreateHist( 1, size, CV_HIST_ARRAY, ranges,1);
// obliczenie histogramu z obrazów
cvCalcHist( images, hist, 0, NULL);
// znalezienie minimalnej i maksymalnej wartości histo
cvGetMinMaxHistValue( hist, &min_val, &max_val);

// tworzenie 8bitowego obrazu do pokazania histogramu
imgHistogram = cvCreateImage(cvSize(256, 50),8,1);
// wypełnienie go białym kolorem
cvRectangle(imgHistogram, cvPoint(0,0), cvPoint(256,50),
CV_RGB(255,255,255),CV_FILLED);

//rysowanie histogramu jasności pikseli znormalizowanego od 0 do 50
for(int i=0; i < 256; i++){
    val = cvQueryHistValue_1D( hist, i);
    norm = cvRound(val*50/max_val);
    cvLine(imgHistogram,cvPoint(i,50), cvPoint(i,50-norm),CV_RGB(0,0,0));
}

//pokazanie okien z rezultatem
cvShowImage( "original", image );
cvShowImage( "gray", gray );
cvShowImage( "histogram", imgHistogram );
// zapis obrazu histogramu do pliku
cvSaveImage("histogray.jpg",gray);
cvWaitKey();

return 0;
}

```

Widać po histogramie, że oryginalny obraz był delikatnie zaciemniony. Brakuje pikseli o wysokim poziomie jasności. Spowodowane było to faktem, że był robiony przy słabym oświetleniu.

OpenCV zawiera kilka ciekawych funkcji modyfikujących histogramy. Za pomocą funkcji `cvNormalizeHist()`:

```
cvNormalizeHist( CvHistogram* hist, double factor );
```

można znormalizować wartości z koszyków. Hist jest przetwarzanym histogramem, factor zaś wartością, do której będzie przeprowadzana normalizacja (zazwyczaj jest to liczba 1).

Kolejną przydatną funkcją jest `cvThreshHist()`:

```
cvThreshHist( CvHistogram* hist, double factor );
```

która dokonuje progowania wartości w koszykach w zależności od parametru factor. Jeżeli wartości w koszykach są poniżej factor wówczas wpisywana jest tam wartość 0. Często wykorzystuje się tą operację do usunięcia szumów z obrazu widocznych na histogramie.

Funkcja `cvGetMinMaxHistValue()`:

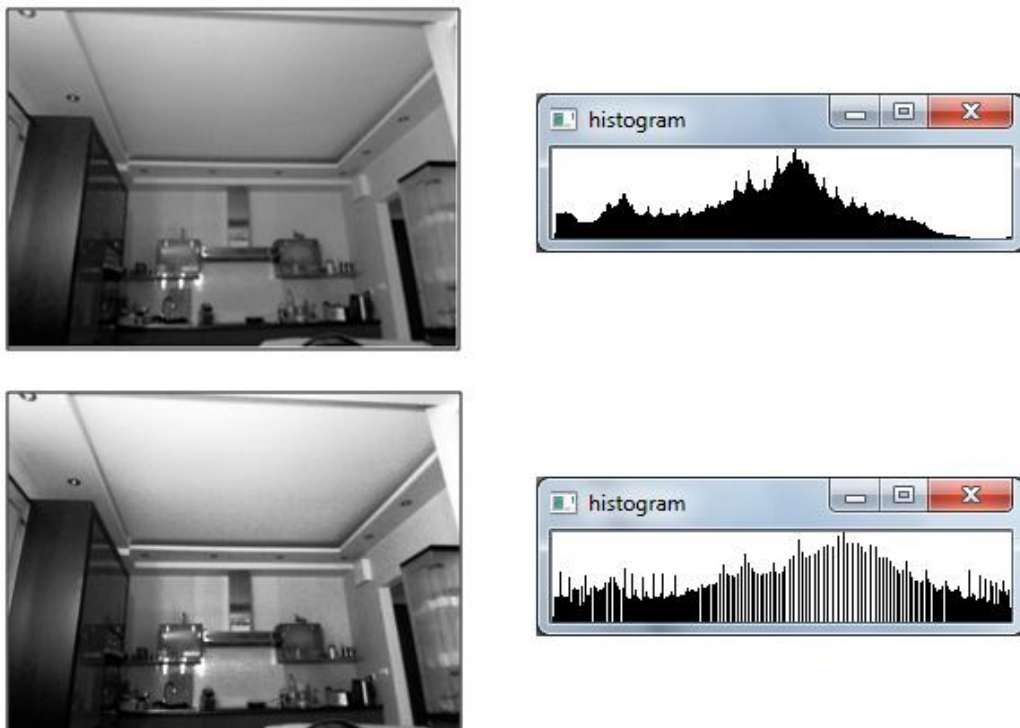
```
void cvGetMinMaxHistValue(const CvHistogram* hist, float* min_value, float* max_value, int* min_idx = NULL, int* max_idx = NULL);
```

zwraca wskaźniki `min_value`, `max_value`, `min_idx`, `max_idx`, które są wartościami najmniejszą i największą występującymi w histogramie oraz indeksem koszy, w których się znalazły.

Gdy na histogramie widoczne są puste koszyki i nie wszystkie odcienie szarości są na obrazie wykorzystywane to warto zastosować wyrównywanie histogramu. Operacja ta polega na przekształceniu jasności punktów obrazu w taki sposób, aby słupki koszy na histogramie były (w przybliżeniu) o takiej samej wysokości. Po tej operacji niezbyt czytelne obrazy zyskują na przejrzystości. W OpenCV operację tą realizuje funkcja `cvEqualizeHist()`:

```
void cvEqualizeHist(const CvArr* src, CvArr* dst);
```

Funkcja ta jako argumenty przyjmuje obraz wejściowy oraz wyjściowy. Poniżej przedstawiony został proces wyrównywania histogramu oraz efekt zmian na obrazie.



Rysunek 48 Programowa realizacja równoważenia histogramu w OpenCV

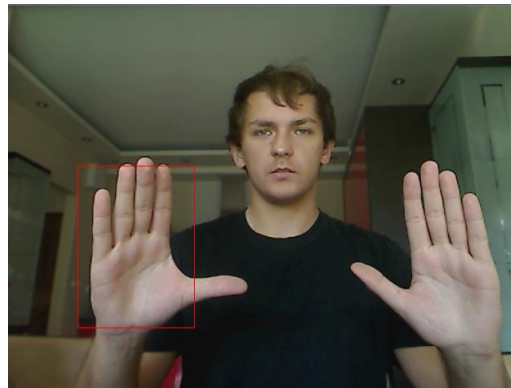
Na rysunku po wyrównaniu histogramu widoczna jest poprawa kontrastu i ostrości. Elementy obrazu są bardziej widoczne i czytelne.

Na zakończenie tego podrozdziału zostanie opisana funkcja `cvCalcBackProject()`. Jej zadaniem jest obliczenie tzw. projekcji wstecznej (ang. back projection) za pomocą

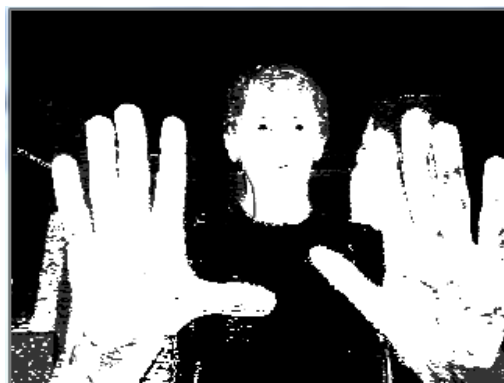
histogramu. Operacja ta polega na porównaniu obiektów z obrazu wejściowego z wcześniej utworzonym histogramem. Jeżeli histogram został stworzony z obrazu, na którym znajdował się tylko jeden obiekt np. ręka, to na obrazie wyjściowym po projekcji wstecznej ten obiekt powinien zostać wyeksponowany. Po zastosowaniu dodatkowego progowania powinien zostać otrzymany binarny obraz ręki. Definicja funkcji wygląda następująco:

```
void cvCalcBackProject(IplImage** image,CvArr* back_project,  
const CvHistogram* hist);
```

Pierwszy parametr `image` jest wskaźnikiem do tablicy obrazów. Przed poddaniem obrazu procesowi wstecznej projekcji musi on zostać podzielony na oddzielne kanały. Następnie są one zapisywane do wspomnianej tablicy. Przy wywołaniu funkcji trzy kanały obrazu wejściowego są analizowane w tym samym czasie. Drugi argument `back_project` jest obrazem wynikowym po projekcji wstecznej, o takich samych parametrach jak obraz wejściowy. Ostatni parametr `hist` jest omawianym histogramem.



Rysunek 49 Obraz wejściowy, histogram zostanie obliczony z obszaru zaznaczonego czerwonym prostokątem



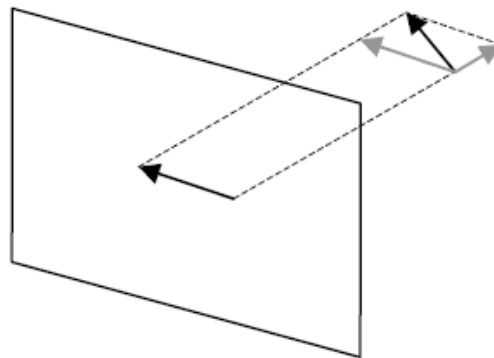
Rysunek 50 Obraz po projekcji wstecznej. Wyeksponowane zostały tylko te barwy, które najlepiej pasowały do wartości z histogramu.

3.9 Analiza ruchu

W celu wykrycia ruchu na sekwencjach obrazów pobieranych z kamery należy badać zmianę położenia obiektów na kolejnych klatkach. Wyznaczenie wektora ruchu pomiędzy

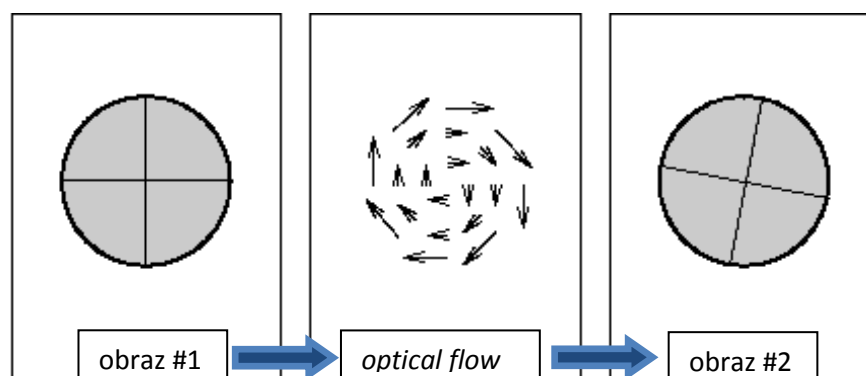
dwiema kolejnymi klatkami sekwencji jest podstawowym zagadnieniem parametryzacji ruchu. Do opisanego procesu należy zapoznać się z dwiema definicjami:

- pole ruchu (ang. motion field) – definiowane jest jako rzutowanie przestrzeni rzeczywistych trójwymiarowych wektorów na płaszczyznę obrazu. Po tej operacji otrzymane pole wektorowe zawiera informację tylko o składowej równoległej ruchu obiektów względem płaszczyzny obrazu. W wyniku czego odtworzenie pierwotnej informacji w oparciu o pole ruchu jest niemożliwe, gdyż tracona jest informacja o składowej prostopadłej.



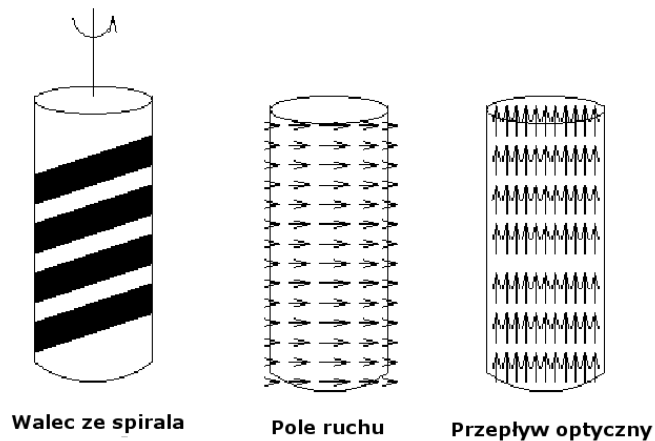
Rysunek 51 Rzutowanie rzeczywistego wektora ruchu na płaszczyznę obrazu

- przepływ optyczny (ang. optical flow) – jest również polem wektorowym, które zawiera informację umożliwiającą przekształcenie jednego obrazu w drugi za pomocą wektorów przepływu optycznego wskazujących kierunek transformacji obszarów pierwszego obrazu do drugiego.



Rysunek 52 Przekształcenie jednego obrazu w drugi za pomocą przepływu optycznego

Należy zauważyć istotną różnicę pomiędzy powyższymi definicjami. Pole ruchu mówi o rzeczywistym ruchu obiektów, natomiast pole przepływu optycznego interpretuje zmianę kolejnych klatek obrazu. Poniższy rysunek pokazuje prosty przykład pokazujący różnicę pomiędzy omawianymi zagadnieniami.



Rysunek 53 Różnica pomiędzy polem przepływu optycznego a polem ruchu

Jak widać pozorny ruch spirali na walcu rejestrowany jest przez pole przepływu optycznego jako ruch ku górze. Pole ruchu wskazuje rzeczywisty kierunek ruchu walca. Można przyjąć, że przepływ optyczny na obrazie jest aproksymacją pola ruchu obiektu w rzeczywistości.

Proces detekcji ruchu na obrazie za pomocą przepływu optycznego jest operacją wymagającą dużej mocy obliczeniowej. Dodatkowo musi być obarczony pewnymi założeniami. Oświetlenie śledzonego obiektu musi być jednolite, gdyż lokalne zmiany jasności obiektu mogą być zidentyfikowane jako ruch. Następnie obiekt nie może być przezroczysty ani o jednolitym kolorze, ponieważ brak tekstury może uniemożliwić wykrycie ruchu. Również wektor ruchu obiektu musi być równoległy do płaszczyzny obrazu, aby przepływ optyczny został wyznaczony prawidłowo.

3.9.1 Metody śledzenia ruchu

Istnieją trzy podstawowe grupy metod wyznaczania przepływu optycznego:

- metody gradientowe – wykorzystujących analizę pochodnych przestrzennych i czasowych intensywności obrazu,
- metody częstotliwościowe – bazujące na filtrowaniu informacji z obrazu w dziedzinie częstotliwości,
- metody korelacyjne – przeszukujące obraz w poszukiwaniu dopasowania regionów (odpowiedniości fragmentów obrazu).

Wszystkie metody z tych grup oparte są na trzech podstawowych etapach przygotowania obrazu do detekcji ruchu:

1. Wstępne filtrowanie obrazu w celu uwydatnienia analizowanych obszarów oraz zwiększenia stosunku sygnału do szumu (ang. SNR – Signal to Noise Ratio).
2. Przeprowadzenie wstępnych pomiarów w celu obliczenia cząstkowych pochodnych po czasie oraz lokalnych powierzchni korelacji.

3. Połączenie wcześniejszych wyników w celu wygenerowania dwuwymiarowego przepływu optycznego.

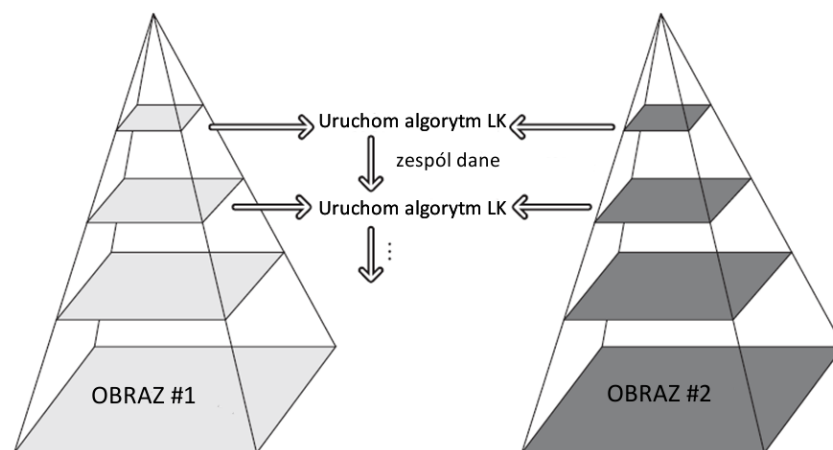
Dokładny opis teoretyczny wymienionych metod można znaleźć w publikacji [12]. W tym rozdziale istotą rzeczy jest opis wybranych metod zaimplementowanych w OpenCV.

3.9.2 Opis i implementacja wybranych metod

Biblioteka dostarcza implementację najpopularniejszych algorytmów śledzenia ruchu takich jak metody gradientowe Lukasa –Kankade, Horn – Schunck, metodę dopasowania blokowego oraz metodę korelacyjną mean-shift i cam-shift.

3.9.2.1 Metoda Lukasa – Kankade

Pierwszą z metod gradientowych jest algorytm Lucas – Kankade (LK). Pierwsza wersja tego algorytmu obliczała wektor przepływu dla każdego punktu obrazu tworząc tzw. „gęste” pole przepływu. Niedługo potem została wprowadzona modyfikacja algorytmu, która obliczała pole wektorowe dla zbioru punktów. Wówczas algorytm mógł zostać wykorzystany do wygenerowania „rzadkiego” pola przepływu, ponieważ zależał tylko od określonego sąsiedztwa otaczającego interesujące punkty obrazu. Zdecydowaną wadą wersji algorytmu z małym lokalnym oknem, była niewielka czułość na większy ruch, który mógł zostać nie wykryty. Problem ten został rozwiązany z pojawieniem się piramidalnego algorytmu LK. Algorytm zaczyna analizę obrazu od najwyższego poziomu piramidy obrazu (o najmniejszej zawartości detali), schodząc kolejno poziomami w dół do poziomów o większej szczegółowości. Realizacja ta pozwala na wykrycie większych ruchów oraz na przyspieszenie działania algorytmu dzięki analizie wcześniejszych jego iteracji.



Rysunek 54 Schemat piramidalnego algorytmu LK

Piramidalny algorytm LK w OpenCV składa się z dwóch etapów:

1. Znajdź punkty dobre do śledzenia.
2. Uruchom pętlę śledzącą zadane punkty.

Pierwszy etap związany jest ze specyfiką metody LK. Poszukiwane są punkty, których gradient jasności ma charakter dwuwymiarowy. Oznacza to, że w otoczeniu tych punktów występują wyraźne zmiany jasności w dwóch różnych kierunkach. Punktami dobrze nadającymi się do śledzenia są narożniki.

W OpenCV funkcją poszukującą takich punktów jest `cvGoodFeaturesToTrack()`:

```
void cvGoodFeaturesToTrack(  
    const CvArr* image  
    CvArr* eigImage, CvArr* tempImage  
    CvPoint2D32f* corners  
    int* cornerCount  
    double qualityLevel  
    double minDistance  
    const CvArr* mask=NULL  
    int blockSize=3  
);
```

Pierwszym parametrem jest obraz wejściowy `image`, który musi być obrazem 8-bitowym lub 32bitowym jednokanałowym. Następne dwa argumenty `eigImage` i `tempImage` są 32bitowymi jednokanałowymi obrazami o tym samym rozmiarze co `image`. Oba te obrazy wykorzystywane są jako bufor podczas pracy algorytmu. Kolejny argument `corners` jest 32bitową tablicą zawierającą współrzędne znalezionych przez algorytm naroży. Parametr `corner_counts` zawiera liczbę znalezionych naroży, `quality_level` określa dokładność z jaką naroża są szukane. Następny parametr `minDistance` definiuje minimalną odległość pomiędzy dwoma narożami. Argument `mask` jest opcjonalnym operatorem logicznym określającym, które punkty obrazu mają być brane pod uwagę, a które nie. Jeżeli ustawiony jest na `NULL` wówczas maska nie jest wybrana. Kolejny parametr `block_size` określa otoczenie piksela, które ma być analizowane.



Rysunek 55 Przykład doboru punktów dobrych do śledzenia¹²

Posiadając listę punktów dobrych do śledzenia uruchamiana jest pętla, w której funkcja śledząca, która przy każdym wywołaniu analizuje następną klatkę obrazu. Uruchamiana w pętli piramidalna metoda LK `cvCalcOpticalFlowPyrLK()` ma następującą postać:

¹² Źródło obrazu [8]

```

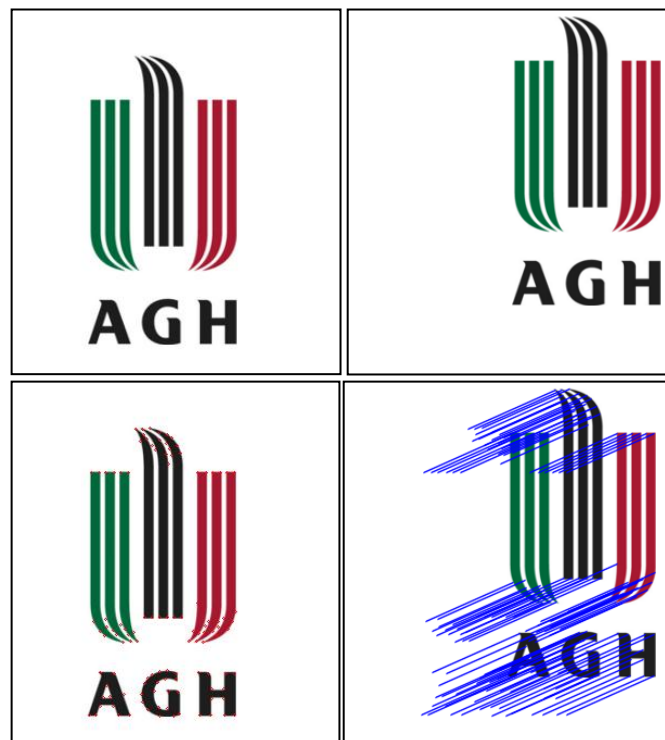
void cvCalcOpticalFlowPyrLK(
    const CvArr* imgA,
    const CvArr* imgB,
    CvArr* pyrA,
    CvArr* pyrB,
    CvPoint2D32f* featuresA,
    CvPoint2D32f* featuresB,
    int count,
    CvSize winSize, int level,
    char* status,
    float* track_error,
    CvTermCriteria criteria, - kryteria działania algorytmu
);

```

Pierwsze dwa parametry `imgA` i `imgB` są kolejnymi analizowanymi klatkami. Kolejne dwa `pyrA` i `pyrB` buforami obrazu piramidy wspomnianych klatek. Następne dwa są tablicami punktów dobrych do śledzenia dla obu klatek. Ilość punktów śledzonych określa parametr `count`. Argument `winSize` wyznacza wielkość okna poszukiwań dla poziomu piramidy. Flagą poprawności wyliczenia przepływu optycznego jest parametr `status`. Tablica błędów estyma przesunięć `track_error` jest kolejnym argumentem. Ostatni parametr `criteria` wyznacza warunki działania algorytmu. Do ich określenia stosowane są makra. W celu ustawienia ilości iteracji na 30 oraz akceptowalnego błędu $\varepsilon = 0.01$ można skorzystać z makra:

```
cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 30, 0.01)
```

Funkcja ta pozwala na znalezienie położenia zadanych punktów z pierwszej klatki na drugiej klatce. Współrzędne punktów są podawane z subpikselową dokładnością.



Rysunek 56 Efekt działania algorytmu PyrLK. Czerwonymi kropkami zaznaczone zostały punkty dobre do śledzenia, a kolorem niebieski trajektoria zmiany położenia piksela (przepływ optyczny)

3.9.2.2 Metoda Horn – Schunck

Metoda zaproponowana przez Horna i Schuncka oparta jest na równaniach przepływu optycznego. Zgodnie z tym równaniem intensywność obrazu $I(x, y, t)$ zależnego również od czasu, pozostaje stała przez całą trajektorię ruchu:

$$\frac{dI(x, y, t)}{dt} = 0$$

Równanie to można zapisać w postaci:

$$\frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \quad (*)$$

Pochodne $\frac{dx}{dt}$ i $\frac{dy}{dt}$ są składową poziomą i pionową wektora prędkości, który można zapisać jako (u, v) gdzie $u = \frac{dx}{dt}$ i $v = \frac{dy}{dt}$. Przyjmując, że I_x, I_y, I_t są gradientami obrazu odpowiednio dla współrzędnych x, y, t oznaczających zmianę intensywności piksela w przestrzeni oraz czasie, równanie (*) można zapisać w postaci:

$$(I_x, I_y) \cdot (u, v) = -I_t \quad (**)$$

będącym jednocześnie warunkiem przepływu optycznego. Równanie to ma jednak dwie niewiadome u i v . Jego rozwiązanie wymaga dodatkowych warunków. Biorąc pod uwagę fakt, że punkty znajdujące się blisko siebie, należące do tego samego obiektu poruszają się z taką samą prędkością, w tym samym kierunku można założyć warunek płynności ruchu. Warunek ten najczęściej zapisuje się jako sumę kwadratów laplasjanów x i y pola przepływu optycznego w postaci Laplasjanów:

$$\nabla^2 u = \left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2$$

$$\nabla^2 v = \left(\frac{\partial v}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2$$

Miara zmiany intensywności pikseli na obrazie wyraża się wzorem [13]:

$$\varepsilon_b = I_x u + I_y v + I_t$$

oraz miara niegładkości pola przepływu:

$$\varepsilon_c^2 = \left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial v}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2$$

Obliczenie pola przepływu optycznego sprowadza się do minimalizacji poniższego funkcjonału:

$$\varepsilon^2 = \iint (\varepsilon_b + \alpha^2 \varepsilon_c^2) dx dy$$

gdzie α^2 jest stałą regularyzacji gwarantującą warunek płynności. Minimalizacja jest realizowana przez znalezienie odpowiednich wartości wektora prędkości (u, v) . Po ich odnalezieniu równanie (**) wyznaczy przepływ optyczny.

Iteracyjny sposób rozwiązywania tych równań zaimplementowany jest w funkcji `cvCalcOpticalFlowHS()`:

```
void cvCalcOpticalFlowHS(
    const CvArr* imgA, const CvArr* imgB, int usePrevious,
    CvArr* velx, CvArr* vely, double lambda,
    CvTermCriteria criteria
);
```

Pierwsze dwa parametry zostały opisane przy funkcji `cvCalcOpticalFlowPyrLK()`. Argument `usePrevious` wskazuje algorytmowi, aby korzystał z prędkości `vel` oraz `vely` obliczonych z poprzedniej klatki jako startowe parametry do obliczenia nowych prędkości. Parametry `velx` i `vely` są nowo obliczonymi prędkościami. Kolejna wartość `lambda` jest stałą regulacji omawianą powyżej. Ostatni parametr `criteria` został również omówiony przy funkcji `cvCalcOpticalFlowPyrLK()`.

3.9.2.3 Metoda Block Matching

Termin „block matching” zawiera w sobie grupę algorytmów, które dzielą kolejne klatki sekwencji na małe regiony zwane blokami. Są to zazwyczaj kwadratowe obszary zawierające określoną ilość pikseli często nachodzące na siebie. Algorytmy dopasowania bloków dzielą poprzednią i aktualną klatkę na regiony i następnie obliczają ruch z nimi związany. Tego typu algorytmy odgrywają znaczącą rolę w algorytmach kompresji wideo oraz omawianym przepływie optycznym.

Implementacja w OpenCV wykorzystuje algorytm SS (ang. spiral search) przeszukiwania obrazu aktualnego w celu znalezienia bloku z klatki poprzedniej. Metoda ta polega na przeszukiwaniu sąsiednich bloków pikseli poczynawszy od środka spirali (miejsca, w którym blok znajdował się na wcześniejszej klatce) na zewnątrz poruszając się po spirali. Przeszukiwane bloki porównywane są na podstawie współczynnika dopasowania SAD (ang. Sum of Absolute values of the Differences) określającego sumę wartości bezwzględnych różnic pomiędzy wartościami pikseli w porównywanych blokach [14].

Funkcją realizującą dopasowanie blokowe w OpenCV jest `cvCalcOpticalFlowBM()`:

```
void cvCalcOpticalFlowBM(
    const CvArr* prev, const CvArr* curr,
    CvSize block_size, CvSize shift_size,
    CvSize max_range, int use_previous,
    CvArr* velx,
    CvArr* vely
);
```

Pierwsze dwa parametry `prev` i `curr` są poprzednią i aktualnym obrazem. Oba muszą być 8-bitowym, jednokanałowym obrazem. Argument `block_size` określa rozmiar bloku wydzielonego z obrazu. Kolejny parametr `shift_size` określa odstęp pomiędzy kolejnymi blokami pikseli (w tym miejscu określa się możliwość tego czy bloki będą na siebie nachodzić). Parametr `max_range` określa maksymalną odległość w jakiej znajdować się mogą blok z klatki poprzedniej i odpowiadający mu blok w klatce aktualnej. Parametr `use_previous` został omówiony w przypadku funkcji `cvCalcOpticalFlowHS()`. Ostatnie dwa argumenty `velx` oraz `vely` zawierają obliczony ruch pomiędzy blokami kolejnych klatek w płaszczyźnie X oraz Y.

Fakt tego, że ruch wykrywany jest na bloku a nie na pojedynczym pikselu zmniejsza w znaczący sposób złożoność obliczeniową algorytmu obliczającego przepływ optyczny. Na obrazie poniżej znajdują się przykładowa realizacja opisywanego algorytmu dla dwóch obrazów.

3.10 Śledzenie obiektów

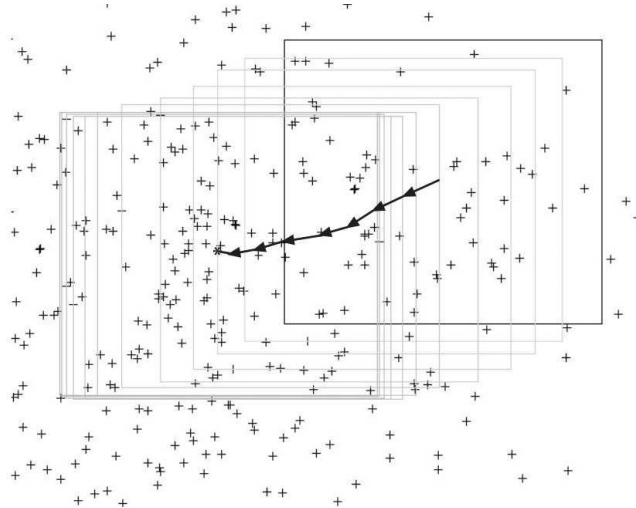
Śledzenie obiektów w OpenCV zostało zrealizowane za pomocą dwóch metod `mean-shift` oraz `cam-shift`.

Algorytm `mean-shift` jest metodą znajdowania lokalnego ekstremum w rozkładzie gęstości zbioru danych [9]. Jest to algorytm deterministyczny, gdyż wprowadza kilka ograniczeń. Przede wszystkim analizuje tylko dane znajdujące się blisko szczytu (ekstremum) poprzez nałożenie lokalnego okna na dane w jego otoczeniu.

Kolejne kroki algorytmu `mean-shift` wyglądają następująco:

1. Dobranie parametrów okna przeszukiwań:
 - wstępną pozycję,
 - typ rozkładu (stały, wielomianowy, wykładniczy lub Gaussowski),
 - kształt (symetryczny lub niesymetryczny, kołowy lub prostokątny),
 - rozmiar.
2. Oblicz środek ciężkości okna.
3. Wyśrodkuj okno w odniesieniu do środka ciężkości.
4. Powrót do punktu 2. jeżeli okno zmieniło pozycję.

Rysunek nr 57 pokazuje zasadę działania tego algorytmu:



Rysunek 57 Zasada działania algorytmu mean-shift¹³

Warunek stopu algorytmu wyznacza parametr ε określający minimalną zmianę położenia okna przeszukiwań. Jeżeli jest ona dostatecznie mała algorytm się zatrzymuje. Zgodnie z założeniami działania metody na pracę algorytmu nie mają wpływu punkty oddalone od okna przeszukiwań.

W celu wykorzystania tego algorytmu do detekcji zmian położenia obiektów na obrazie metoda musiała zostać zmodyfikowana i rozbudowana. Przede wszystkim należy najpierw określić cechę śledzonego obiektu np. kolor. Następnie pierwsze okno przeszukiwań powinno zostać ustawione w miejscu występowania tego obiektu, następne kroki algorytmu są tożsame z podstawową metodą. Implementację algorytmu w OpenCV realizuje funkcja `cvMeanShift()`, która opiera się o analizę obrazu projekcji wstecznej histogramu, pobranego z pierwszego okna przeszukiwań:

```
int cvMeanShift(  
    const CvArr* prob_image,  
    CvRect window,  
    CvTermCriteria criteria,  
    CvConnectedComp* comp  
);
```

Pierwszym parametrem `prob_image` jest obraz otrzymany w wyniku projekcji wstecznej histogramu za pomocą omawianej w rozdziale 3.8. funkcji `cvCalcBackProject()`. Drugi parametr `window` określa wstępne położenie i rozmiar okna przeszukiwań. Kolejny argument `criteria` omawiany był przy innych funkcjach i określa maksymalną ilość kroków algorytmu. Ostatni parametr `comp` jest strukturą wynikową zawierającą informacje o aktualnym położeniu okna przeszukiwań (`comp->rect`) oraz ilości pikseli, które obejmuje (`comp->area`). Funkcja zwraca ilość wykonanych iteracji algorytmu.

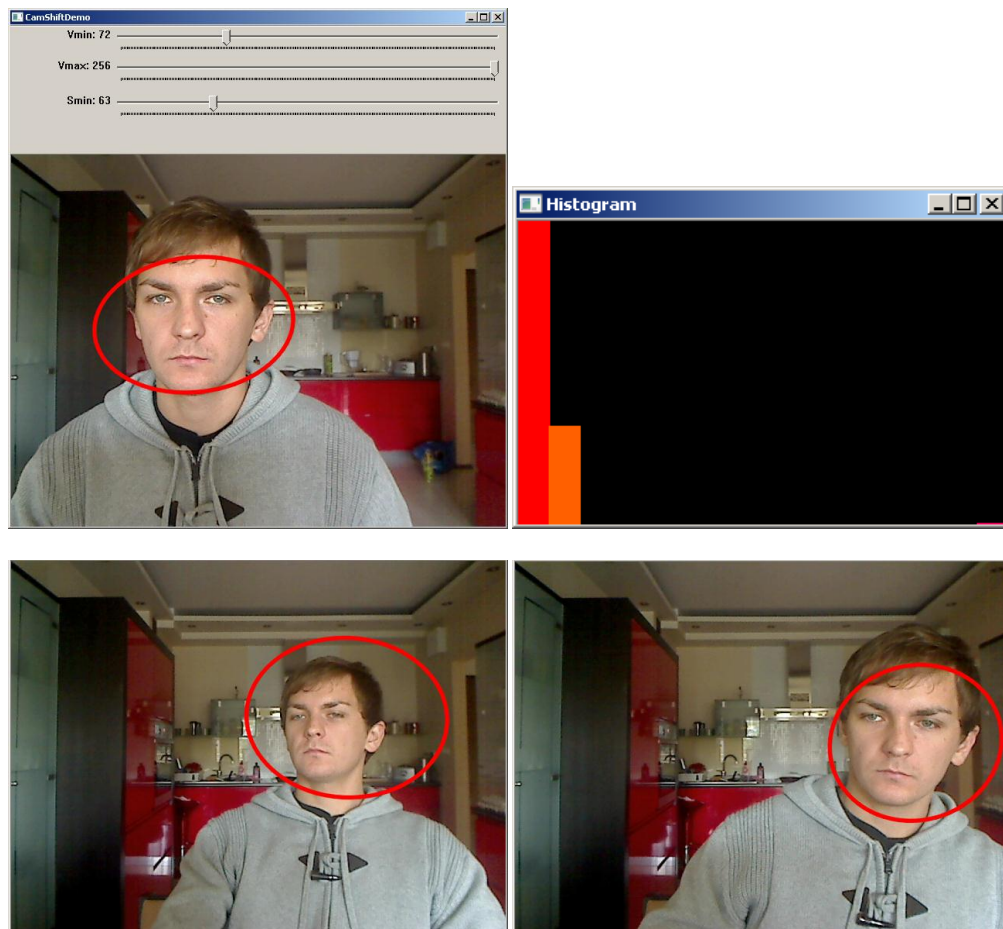
Rozwinięcie nazwy drugiego algorytmu `cam-shift` oznacza „ciągłe adaptowany mean-shift” (ang. continuously adaptive mean-shift). Oparty jest na algorytmie `mean-shift` jednak śledzi

¹³ Źródło obrazu: [8]

również obiekty, których kształt się zmienia w trakcie trwania sekwencji wideo. Automatycznie dopasowuje wielkość okna przeszukiwań w zależności od wielkości obszaru zawierającego poszukiwane cechy obiektu. Można zwiększyć wydajność działania metody poprzez założenie niewielkiego ruchu śledzonego obiektu. W przypadku śledzenia twarzy algorytm obliczy w pierwszym kroku histogram rozkładu np. koloru skóry z określonego okna przeszukiwań. Następnie wyznaczy jego środek ciężkości, aby w następnej analizowanej klatce zwiększyć jedynie nieznacznie okno przeszukiwań i analizować piksele go obejmujące. Dzięki temu pozostałe piksele obrazu są ignorowane. W OpenCV funkcją realizującą tą metodę jest `cvCamShift()`:

```
int cvCamShift(
    const CvArr* prob_image,
    CvRect window,
    CvTermCriteria criteria,
    CvConnectedComp* comp,
    CvBox2D* box = NULL
);
```

W przypadku tej funkcji dodatkowym parametrem jest element strukturalny `box` będący prostokątem otaczającym śledzony obiekt (zawiera informację o jego rozmiarach). Poniżej przedstawione zostało działanie metody `cam-shift` za pomocą programu demonstracyjnego `camshiftdemo.exe` umieszczonego w katalogu biblioteki `OpenCV2.1/Samples/c/`.



Rysunek 58 Sposób działania algorytmu `cam-shift`

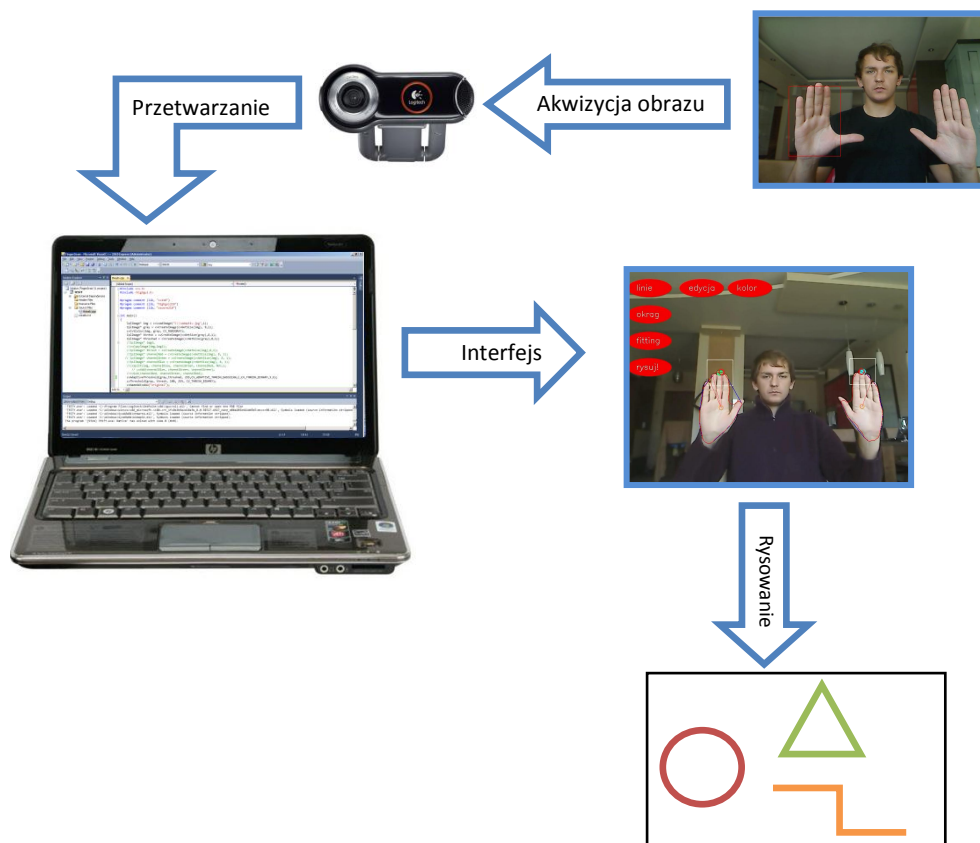
4. Realizacja praktyczna

Zagadnienia praktyczne były realizowane przy pomocy kamery internetowej Logitech Pro 9000 o optyce Carl Zeiss dużej ostrości i wysokiej 2MPix rozdzielczości. Kamera to została wybrana ze względu na dobrą jakość obrazu oraz szerokich możliwości programowych doboru parametrów analizowanego obrazu.

4.1 Finger Drawing

4.1.1 Realizacja

Program ma za zadanie umożliwienie tworzenia figur geometrycznych za pomocą ruchów rąk analizowanych przez kamerę. Pierwszym etapem było wyeksponowanie dłoni znajdujących się na obrazie, za pomocą filtracji kolorów odpowiadających tonom skórnym. Następnie zaimplementowane algorytmy badają ruch i położenie dłoni, za pomocą których można przystąpić do tworzenia obiektów na interaktywnym ekranie.



Rysunek 59 Diagram funkcjonowania programu Finger Drawing.

4.1.2 Problemy

W wielu opracowaniach naukowych, również zagranicznych uniwersytetów metoda śledzenia i odnajdywania dłoni jest podobna. Odbywa się to za pomocą analizy tonów skóry. Metoda ta daje dobre rezultaty jednak przy pewnych założeniach. Pierwszym z nich są niezmiennie warunki oświetleniowe, gdyż barwa jest ściśle związana z oświetleniem i razem z nim, zmieniają się jej parametry.

Kolejnym problemem jest otoczenie. Wiele przedmiotów posiada barwę podobną do ludzkiej skóry. Stanowi to problem, który jest minimalizowany przez takie sposoby jak odrzucanie zbyt małych obiektów, uważanych za przypadkowe.

W kadrze kamery najczęściej znajduje się również głowa, której skóra ma takie same tony jak skóra dłoni. Może to stanowić problem przy wykrywaniu krawędzi dłoni, gdy oba obiekty będą na siebie nachodzić. Czasem problem ten rozwiązuje zwykajne oddalenie dłoni od głowy, gdyż mogą być one wówczas oświetlone z różną intensywnością co często sprawia, że głowa staje się niewidoczna dla programu.

W celu optymalizacji działania algorytmów detekcji użytkownik może skorzystać z suwaków, które zmieniają wartości przestrzeni kolorów HSV przetwarzanego obrazu. Na koniec zaznaczając prostokątny obszar dłoni na obrazie, pozwala programowi na wygenerowanie histogramu zakresu barw dłoni, będącego podstawą dalszego działania algorytmów detekcji dłoni.

4.1.3 Opis działania programu

4.1.3.1 Konwersja i zawężenie przestrzeni kolorów.

Obraz pobierany z kamery jest konwertowany z przestrzeni RGB do HSV. Przestrzeń ta pozwala nam na pracę tylko z barwą (ang. Hue), co ułatwia zawężenie przestrzeni kolorów do interesującego zakresu tonu skóry. Z wykorzystaniem dostępnych suwaków możliwe jest sterowanie zakresem przestrzeni pozostałych kanałów, a co najistotniejsze kanałem jasności (ang. Value). Wykorzystując funkcję `cvInRangeS()` na podstawie dolnych i górnych przedziałów suwaków, z obrazu HSV tworzona jest maska `mask` (obraz binarny) na której uwidocznione są tylko tony skórne. Nie jest ona jednak pozbawiona przypadkowych szumów czy też elementów, które mogą posiadać zbliżoną barwę do tonu skóry ale nie są na tyle duże, by znacznie utrudniły działanie programu. W celu usunięcia tego typu zakłóceń przeprowadzone jest kilka elementarnych, dobranych doświadczalnie operacji na obrazie. Początkowo nałożone zostało rozmycie medianowe (funkcja `cvSmooth()`), które wstępnie usuwa szумы. Następnie obraz zostaje poddany erozji (`cvErode()`) w celu usunięcia większych przypadkowych obiektów oraz dyatacji (`cvDilate()`), aby uzupełnić kształt dłoni w masce.



Rysunek 60 Maska wejściowa



Rysunek 61 Maska po regulacji suwakami HSV

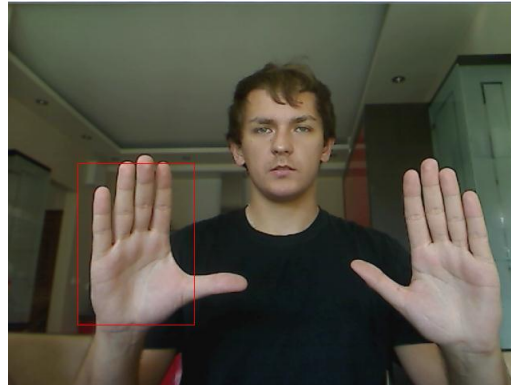


Rysunek 62 Maska po wygładzeniu

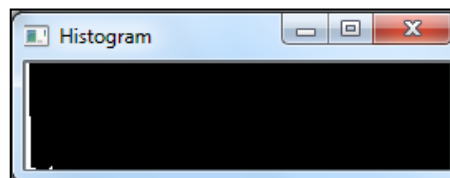
4.1.3.2 Tworzenie histogramu dłoni

Kolejnym etapem jest wyliczenie histogramu zaznaczonej dłoni, który jest kluczowy w działaniu algorytmu odnajdującego tony skórne. Należy zwrócić uwagę na to, iż histogram jest obliczany tylko raz, na początku algorytmu. Obliczanie histogramu odbywa się za pomocą funkcji `cvCalcHist()`. Funkcja ta oblicza histogram tylko i wyłącznie barwy czyli kanału Hue

z przestrzeni HSV. W tym celu jako pierwszy argument przyjmuje kanał Hue obrazu z kamery, drugim argumentem jest zwracany histogram, a ostatni parametr to obliczona funkcją `cvInRangeS()` maska binarna.



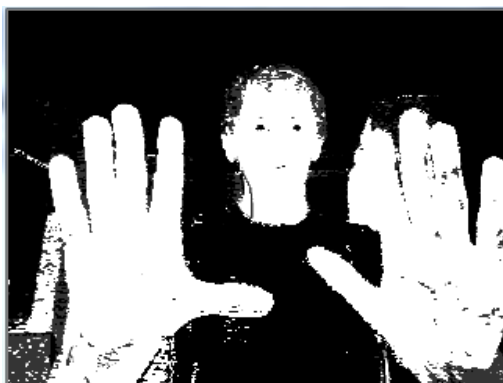
Rysunek 63 Zaznaczenie obszaru ROI



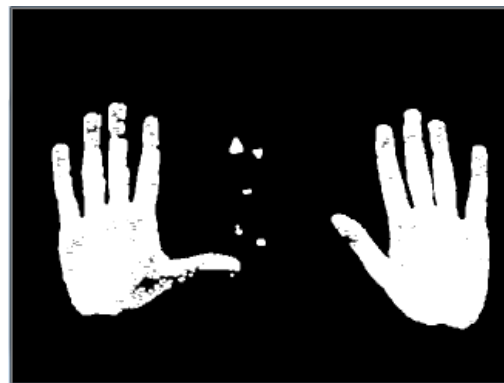
Rysunek 64 Histogram kanału Hue

4.1.3.3 Tworzenie obrazu wynikowego i wykrywanie konturów

Następnie funkcja `cvCalcBackProject()` wywołana z argumentami będącymi kanałem Hue oraz obliczonym histogramem zwraca obraz backproject w skali szarości, na którym wyeksponowane są obiekty w tonach skórnych. Na podstawie tego obrazu oraz stworzonej przy pomocy suwaków maski obliczany jest ich iloczyn bitowy (`cvAnd()`), który po wygładzeniu (`cvSmooth()`) stanowi binarny obraz wynikowy.

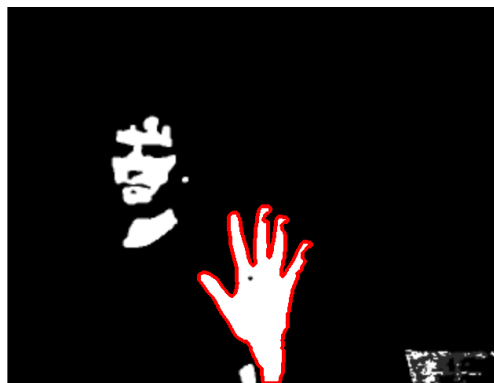


Rysunek 65 Obraz backproject



Rysunek 66 Obraz backproject po przetworzeniu

Na tak przygotowanym obrazie wykrywane są kontury. Za inicjację odnajdywania konturów odpowiedzialna jest funkcja `cvStartFindContours()`, której parametrami są obraz wynikowy, przydzielona pamięć, rodzaj konturu i sposób jego reprezentacji. Parametr `CV_RETR_EXTERNAL` określający rodzaj szukanych konturów został wybrany, ponieważ poszukiwane są najbardziej zewnętrzne kontury (np. kontury palców, a nie wnętrza dłoni). `CV_CHAIN_APPROX_SIMPLE` sprawia, że kontur jest reprezentowany w pamięci jako łańcuch punktów najbardziej przylegających do konturu. Następnie funkcja `cvFindNextContour()` wyszukuje kolejno kontury zwracając je jako sekwencję do obiektu struktury `CvSeq`. Dzięki takiej organizacji możliwy jest łatwy dostęp do poszczególnych konturów. W ten sposób można odfiltrować te kontury, które na pewno nie mogą stanowić kształtu dłoni, gdyż są zbyt małe. Wykorzystywana jest do tego funkcja `cvContourArea()`, która zwraca obszar konturu (w wymiarze piksela kwadratowego). Wartość progowa (6000) została wyznaczona doświadczalnie i zapewnia rozsądne minimum dla dopuszczalnej wielkości dłoni, nawet, gdy jest znacznie oddalona od kamery. Eliminuje za to wszelkie przypadkowe kontury, które znajdują się pod w/w progiem. Mogą to być np. drobne, tymczasowe obiekty które zmieniły swoją barwę wskutek chwilowej zmiany oświetlenia. Następnie za pomocą funkcji `cvDrawContours()` rysowane są kontury dłoni.



Rysunek 67 Wykryty kontur na obrazie wynikowym

Z konturów wybierane są najwyższe położone punkty, które odpowiedzialne są za sterowanie wskaźnikami będącymi podstawą rysowania i sterowania obiektami.

4.1.3.4 Mechanizmy tworzenia obiektów

Ta część opiera się w dużej mierze na mechanizmach obiektowości, dostarczanych przez język C++. Za działanie śledzenia i funkcji rysujących w przeważającej mierze odpowiedzialna jest klasa, która potrzebuje dwóch parametrów, aby realizować swoje zadanie. Pierwszy to pobrane wspomniane już wcześniej maksymalne punkty dłoni, a drugi to informacja gdzie ma być wyświetlany rezultat (obiekt `Ip1Image`). Punkt maksymalny dłoni wyznaczany jest z tablicy punktów stanowiących kontur i jest to najwyższy położony punkt w osi pionowej. Na ekranie oznaczany czerwonym okręgiem. Dwa okręgi z obu dłoni będą sterować obiektami (ang. pointerami) służącymi do nawigacji w programie. Koniecznością było ich zastosowanie ze względu na proceduralny charakter wykonywania programu. Dla oka ludzkiego wydaje się, że czerwone punkty na czubkach dłoni występują równocześnie. W rzeczywistości przy jednej iteracji programu rysowany jest jeden, a przy kolejnej drugi punkt. Dlatego też, zostały utworzone dynamiczne obiekty, które posiadają mechanizmy odpowiedzialne

za zapamiętywanie położenia każdego wskaźnika z osobna oraz w pełni niezależne działanie i interakcję z innymi aktywnymi obszarami ekranu.

Funkcją odpowiedzialną za nieustanną analizę wskaźników jest każdorazowo wywoływana funkcja `PointersTracking()`, która pobiera i analizuje położenie maksymalnych punktów dłoni zapisanych w wektorze punktów.

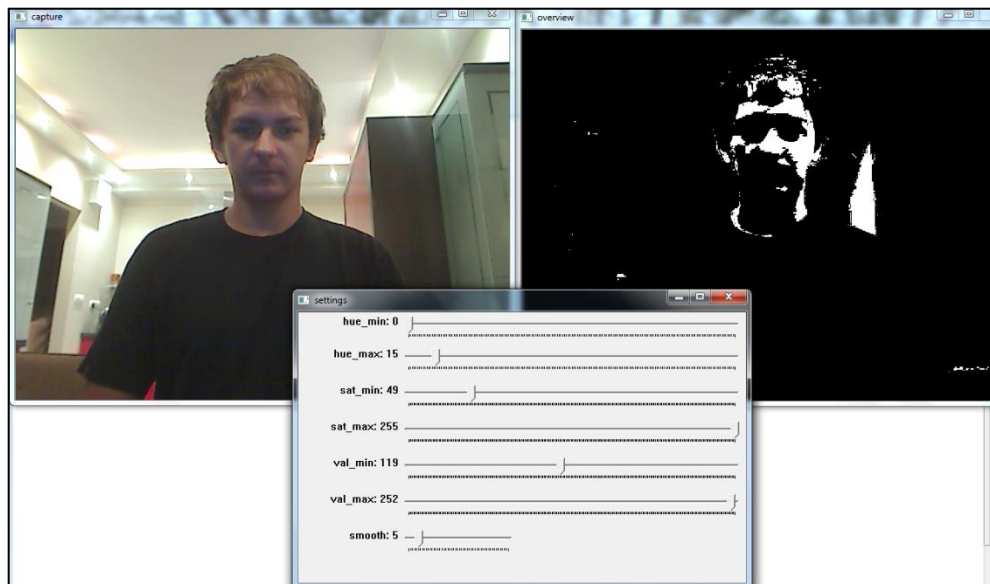
Następnie wywoływana jest funkcja `checkAreaMarked()`, która sprawdza, czy któryś z pointerów nie znalazł się w obszarze któregoś z przycisków bądź suwaków koloru.

Ostatnią funkcją interfejsu jest wyrysowanie na ekranie wszystkich aktywnych obiektów i zdarzeń w tym tworzonych figur przez użytkownika – `drawExistingObjects()`, której argumentem jest obraz na którym ma być wyświetlany.

4.1.4 Obsługa programu

Po włączeniu programu na ekranie pojawiają się trzy okna dialogowe:

- „capture” pokazuje cały interfejs użytkownika zawarty na obrazie pochodzącym z kamery,
- „overview” pokazuje maskę binarną, której zawartość modyfikowana jest za pomocą suwaków,
- „settings” zawiera panel suwaków, które sterują parametrami przestrzeni HSV (`hue_min`, `hue_max`, `sat_min`, `sat_max`, `val_min`, `val_max`, `smooth`).



Rysunek 68 Okna służące do wstępnej kalibracji obrazu.

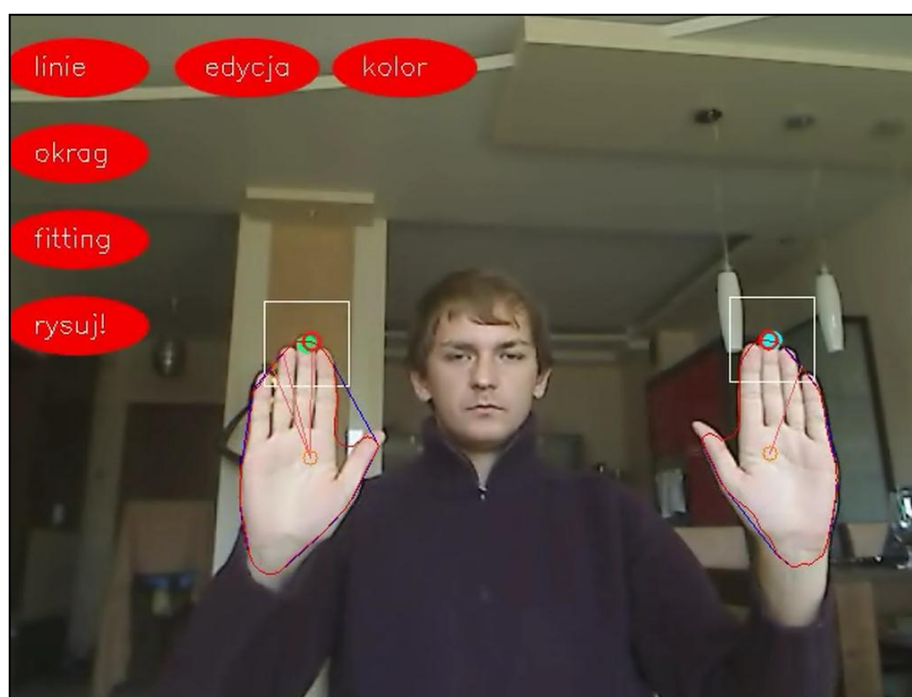
Użytkownik powinien zacząć od przygotowania maski wyświetlanej w oknie „overview”. Realizowane jest to za pomocą suwaków z okna „settings”. Parametry HSV powinny być

dobrane w taki sposób, aby na obrazie maski widoczne były tylko efekty skórne – kolor biały, zaś tło powinno być widoczne jako kolor czarny.

Po odpowiednim przygotowaniu maski należy pobrać histogram dłoni zaznaczając ją prostokątnym obszarem wyznaczeniem za pomocą myszki i zaakceptować go naciskając klawisz „b”. Obliczony histogram można zobaczyć w oddzielnym oknie po naciśnięciu klawisza „h”. Po operacji obliczenia histogramu w oknie „capture” pojawia się główny interfejs programu, który składa się z przycisków do tworzenia obiektów i ich edycji oraz dwóch pointerów do sterowania interfejsem. Widoczne są również kontury dłoni, które można wyłączyć przyciskając klawisz „a”.

Użytkownik dodatkowo może w każdej chwili rozpocząć nagrywanie swojej pracy przyciskając klawisz „c”. W oknie, które się pojawi należy wybrać jeden z kodeków obsługiwanych przez OpenCV. Polecany jest kodek XVID o dobrym stosunku kompresji do jakości wideo.

Za pomocą klawisza „f” użytkownik może również obserwować swoją pracę na pełnym ekranie, a nie w domyślnym oknie o rozdzielczości 640x480.



Rysunek 69 Interfejs użytkownika programu

Poniżej zostały opisane funkcje przycisków interfejsu:

- „rysuj!” uruchamia tryb rysowania i działa w połączeniu z przyciskami „linie”, „okrąg” oraz „fitting”, których wybór decyduje o sposobie i możliwym kształcie rysowanych figur:
 - „linie” umożliwia rysowanie pojedynczych linii lub łańcucha połączonych linii tworzących wielobok. Każda linia tworząca ten obiekt musi zostać zatwierdzona wyświetlonym tymczasowo przyciskiem „ok”,

- „**okrag**” umożliwia rysowanie okręgu o położeniu środka wskazywanego przez prawy pointer. Promień okręgu wyznaczany jest przemieszczeniem prawego pointera od środka okręgu na zewnątrz,
- „**fitting**” umożliwia rysowanie okręgów za pomocą konturu wyznaczanego przez okrężny ruch prawego pointera, a następnie sprowadzanego do właściwego okręgu,
- „**OK**” akceptuje i zapisuje w pamięci narysowany obiekt,
- „**edytuj**” umożliwia edycję narysowanych obiektów. Do edycji należą:
 - zmiana położenia każdego z obiektów, za pomocą punktu centralnego obiektu,
 - zmiana kształtu wieloboków, za pomocą zmiany położenia ich wierzchołków,
 - zmiana długości promienia okręgów,
- „**kolor**” umożliwia zmianę kolorów krawędzi oraz wypełniania obiektów za pomocą suwaków aktywowanych tym przyciskiem umożliwiającym otrzymanie każdego z kolorów przestrzeni RGB.

Sterowanie programem powinno odbywać się w następujący sposób. Lewy pointer służy do aktywowania przycisków, prawy zaś do rysowania i edycji obiektów. Opis działania i sterowania podczas tworzenia obiektów został opisany przy okazji wymieniania funkcji przycisków.

4.1.5 Możliwość rozwoju

Program posiada duże możliwości rozwoju. Między innymi można byłoby go wzbogacić o wykrywanie gestów ręki, które przypisane byłyby do konkretnej operacji.

Duże pole do manewru pozostawia sposób „nastrajania” kamery w celu jak najlepszego wykrycia barw skóry. Zastosowanie metody bardziej zautomatyzowanej ułatwiłoby pracę z oprogramowaniem.

Sam algorytm wykrywania i śledzenia dłoni można byłoby oprzeć na analizie ruchu obiektów. Na przykład zastosować algorytm przepływu optycznego (ang. opticalFlow) opisany w części teoretycznej. Również opracowanie własnego klasyfikatora Haar’a dla dłoni rozbudowując w ten sposób proces śledzenia. Należy jednak pamiętać, że wszystkie wspomniane udoskonalenia zwiększają znacznie złożoność obliczeniową programu i wprowadzają duże opóźnienia, które w warunkach domowych mogą wręcz utrudnić pracę z programem.

4.2 Face Detect

4.2.1 Realizacja

Program „Face Detect” ma za zadanie przechwycić obraz z kamery a następnie wyszukać na nim twarze za pomocą metody Haar-Like. Oprogramowanie wyposażone jest również w filtry oraz funkcje poprawiające jakość analizowanego obrazu. Dodatkowo zaimplementowany został algorytm pozwalający na szybsze działanie programu dzięki wyodrębnieniu obszarów analizowanych pod kątem detekcji twarzy.



Rysunek 70 Diagram funkcjonowania programu Face Detect

4.2.2 Problemy

W celu lepszej detekcji twarzy za pomocą algorytmu Haar-Like wykorzystywane były tony skórne, które pozwalały na wyodrębnienie obszaru ograniczającego działanie algorytmu. Tak samo jak w przypadku programu „Finger Draw” wpływ na identyfikację koloru skóry miały warunki oświetleniowe. Kolor skóry w zależności od oświetlenia zmieniał swoje parametry.

Drugim problemem jest stworzenie własnego klasyfikatora Haar’a do metody. Zgromadzenie rozbudowanej bazy danych pozytywnych zdjęć twarzy oraz negatywnych zdjęć otoczenia wymaga sporego nakładu czasowego. Projekt więc został oparty o umieszczony w bibliotece klasyfikator stworzony przez twórców biblioteki.

4.2.3 Opis działania programu

4.2.3.1 Załadowanie klasyfikatora Haar’a

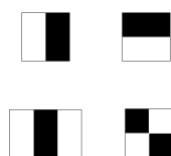
Na początku program ładuje gotowy klasyfikator Haar’a do wykrywania frontowej twarzy, który umieszczony jest w pliku XML (haarcascade_frontalface_alt.xml).

Tworzony jest on za pomocą funkcji „Haar-like features” wbudowanej w OpenCV służącej do wykrywania obiektów na podstawie cech. Wykorzystuje ona metodę zaproponowaną przez Paula Viola i Michaela Jones’a, która ma za zadanie wydobyć użyteczne informacje z obrazu do klasyfikacji obiektów i ich tła. Metoda ta została następnie rozwinięta przez Rainera Lienhart’a oraz Johena Maydt’a.

Algorytm funkcji „Haar-like features” opiera się na trzech podstawowych fazach:

1. Wybór zestawu cech

W metodzie Viola-Jenes cechy traktowane były jako kombinacja dwóch, trzech lub czterech prostokątów. Ilość możliwych kombinacji połączenia prostokątów była bardzo duża, dlatego przy określonych założeniach została wyselekcjonowana grupa, która została przedstawiona na rysunku nr 65. Następnie udoskonalony zestaw cech jeszcze lepiej identyfikujący poszukiwane obiekty został zaproponowany w metodzie Lienhart-Maydt (przedstawiony na rys. 66). Grupa ta została podzielona na cechy krawędziowe, liniowe oraz centralnie otoczone. Właśnie z tego zestawu cech korzysta funkcja z OpenCV.



Rysunek 71 Podstawowy zestaw cech zaproponowany przez Viola i Jones’a

1. Cechy krawędziowe



2. Cechy liniowe



3. Cechy centralne



Rysunek 72 Podstawowy zestaw cech zaproponowany przez Lienhart’a i Maydt’a

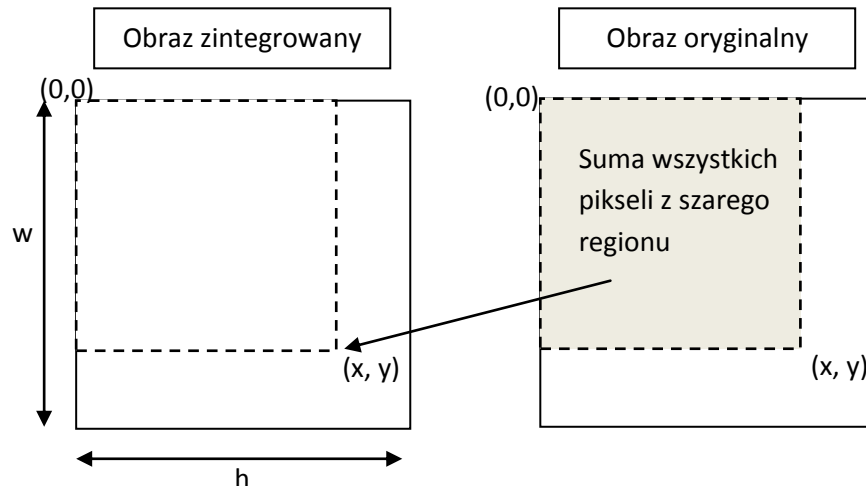
Zasada działania cech polega na odnajdywaniu obszarów, na których różnica pomiędzy sumą pikseli regionów ograniczonych czarnymi i białymi prostokątami znajdują się powyżej pewnego progu. Na podstawie tej wartości algorytm może zidentyfikować czy badany obszar jest poszukiwanym obiektem czy jego tłem. Dla przykładu, prostokątne regiony oczu mają dużo mniejszą intensywność niż obszar czoła, ten zaś dużo większą od obszaru włosów (rysunek 67).



Rysunek 73 Przykład zastosowania cech do detekcji twarzy

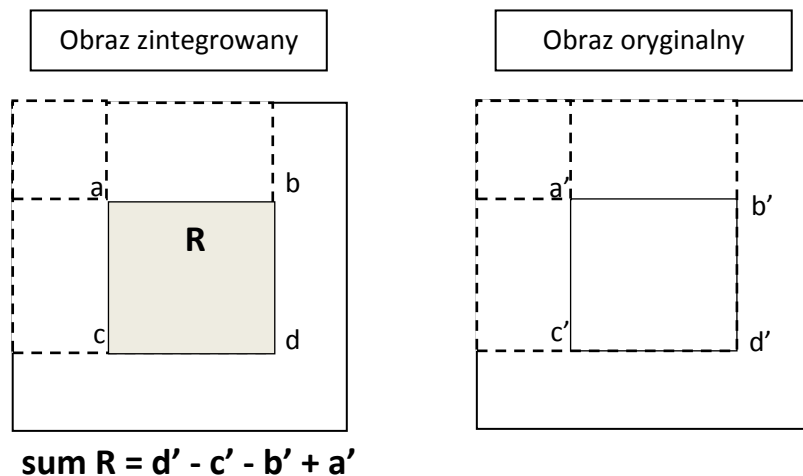
2. Wykorzystywanie obrazu zintegrowanego do obliczenia wartości cech

Obliczanie różnic prostokątnych obszarów na obrazie jest bardzo czasochłonne, dlatego algorytm detekcji obiektów wykorzystuje tzw. obraz zintegrowany (ang. integral image). Jest on reprezentacją obrazu, która w punkcie o współrzędnych (x, y) przechowuje wartość sumy wartości pikseli znajdujących się na lewo i powyżej punktu (x, y) – rysunek 68.



Rysunek 74 Obliczanie obrazu zintegrowanego z oryginalnego

Wykorzystując obraz zintegrowany sumę każdego obszaru można policzyć za pomocą trzech operacji.



Rysunek 75 Obliczanie sumy pikseli regionu R obrazu źródłowego za pomocą czterech punktów obrazu zintegrowanego

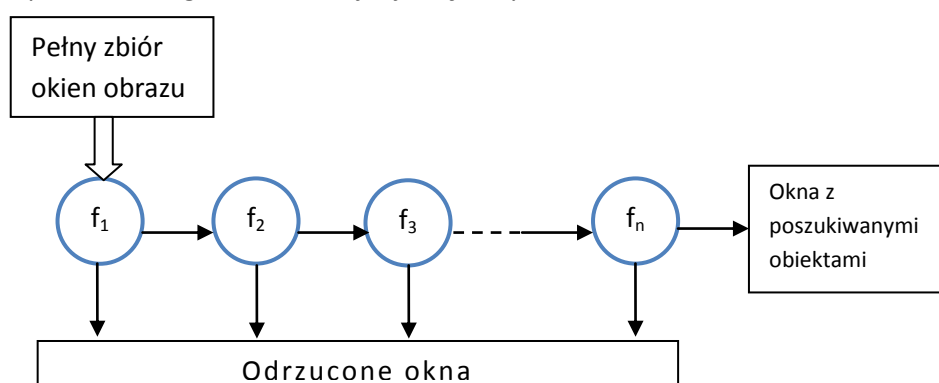
Ten sposób obliczania sum pikseli z obszaru obrazu źródłowego wykorzystywany jest w metodzie Viola-Jones'a. Zmiana prototypów cech w metodzie Lienharta-Maydt'a wymusiła zmianę sposobu obliczania regionów pikseli. Dokładny opis modyfikacji przedstawiony został w publikacji [15].

3. Tworzenie klasyfikatora kaskadowego z wykorzystaniem algorytmu uczącego się AdaBoost

Wykorzystuje on tzw. wzmocnienie adaptacyjne (ang. adaptive boosting). Jego zadaniem jest uczenie tzw. słabych klasyfikatorów (ang. weak classifier), drzew decyzyjnych (z dwoma liśćmi tzn. jeżeli wartość v cechy słabego klasyfikatora f jest powyżej pewnego progu t wówczas $+1$ („tak”) oznacza znalezienie obiektów lub -1 („nie”) jego brak.

$$f_i = \begin{cases} +1, & \text{gdy } v_i \geq t_i \\ -1, & \text{gdy } v_i < t_i \end{cases}$$

Zadaniem algorytmu AdaBoost [16] jest wygenerowanie silnego klasyfikatora z kaskady słabych. W przypadku obrazów, na których wykryta ma zostać obiekt wykorzystuje się technikę przesuwne okna. Z tak zeskanowanego obrazu generowany jest zbiór okien, które poddawane są weryfikacji przez klasyfikatory. Na podstawie informacji z tych okien słabe klasyfikatory uczą się odrzucać obszary, na których na pewno nie znajduje się poszukiwany obiekt. W każdej pętli algorytm skupia się na źle wykrytych obszarach przez poprzednie klasyfikatory. Nadaje im wyższe wagi (gdyż w następnej iteracji istnieje większe prawdopodobieństwo prawidłowego wykrycia obiektu). Wykorzystywanie tych wag przez następne klasyfikatory pozwala na zawężenie przestrzeni poszukiwań do coraz mniejszego obszaru. Tak skonstruowany silny klasyfikator umożliwia szybkie wykrywanie poszukiwanych obiektów. Proces odrzucenia obszarów niezawierających poszukiwanego obiektu znajduje się na rysunku 76.



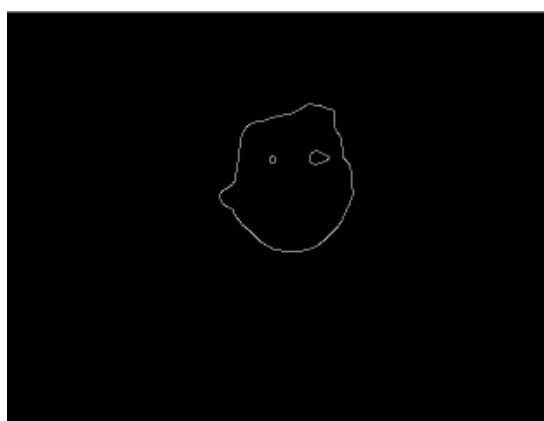
Rysunek 76 Budowa klasyfikatora kaskadowego

W OpenCV znajdują się wszelkie narzędzia potrzebne do konstruowania własnego klasyfikatora Haar'a dla konkretnego obiektu. Wszystkie programy znajdują się w katalogu biblioteki `C:/OpenCV2.1/bin`. Program `createsamples.exe` tworzy zbiór przykładowych obrazów pozytywnych i negatywnych w postaci wektora uczącego (plik *.vec). Otrzymany plik wykorzystywany jest do uczenia klasyfikatora za pomocą programu `haartraining.exe`.

Utworzony klasyfikator zapisywany jest do pliku XML. Wykorzystując dodatkowy zbiór obrazów pozytywnych i negatywnych, które nie brały udziału w procesie uczenia się można przetestować klasyfikator w programie *performance.exe*. Dokładny opis konfigurowania oraz obsługi programów można znaleźć w źródle [W9].

4.2.3.2 Konwersja i zawężenie przestrzeni kolorów

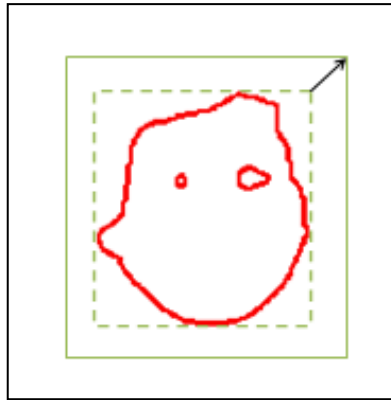
To zagadnienie realizowane jest niemal identycznie jak w przypadku programu „Finger Draw”. Najpierw obraz z kamery konwertowany jest do przestrzeni HSV. Następnie suwakami ustawiane są optymalne parametry przestrzeni w celu jak najlepszej detekcji tonów skórnych przy aktualnych warunkach oświetleniowych. Na podstawie tych parametrów tworzona jest maska, która na obrazie binarnym pokazuje tylko barwy obrazu w tonie skóry. Maska poddawana jest otwarciu za pomocą funkcji `cvErode()` oraz `cvDilate()`. Następnie wygładzana dodatkowo rozmyciem medianowym (`cvSmooth()`). Na koniec stosowany jest filtr krawędziowy Canny’ego (`cvCanny()`), aby wykryć krawędzie na podstawie, których będzie obliczany kontur twarzy.



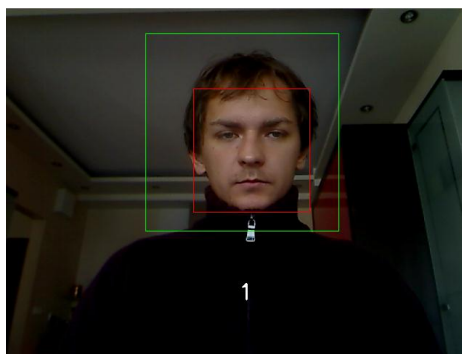
Rysunek 77 Okno maski po przetworzeniu

4.2.3.3 Wykrywanie konturów i ograniczenie obszaru do analizy

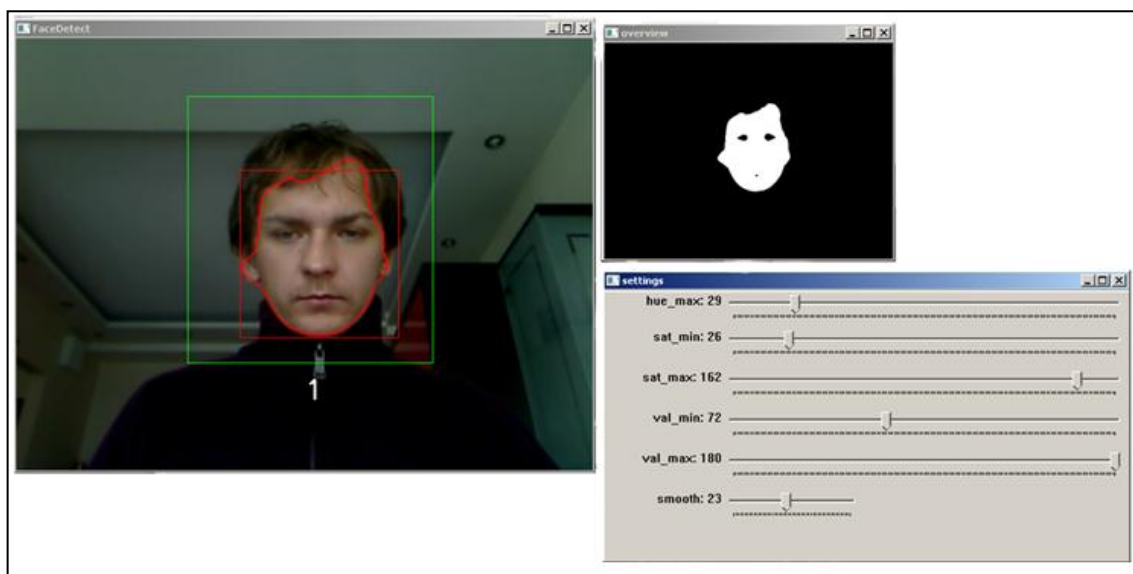
W celu zminimalizowania obszaru analizowanego przez funkcję wykrywającą twarze (Haar-Like) obliczane są kontury z obrazu maski za pomocą funkcji `cvFindContours()`. Następnie punkty konturów zapisane w sekwencji, konwertowane są do tablicy punktów z wykorzystaniem funkcji `cvCvtSeqToArray()`. Punkty tablicy analizowane są przez funkcję `cvBoundingRect()` w poszukiwaniu najbardziej zewnętrznych punktów. Po ich znalezieniu po konturze, obrysowywany jest zielony prostokąt, który następnie jest powiększany programowo z wykorzystaniem doświadczalnie dobranej skali. Powierzchnia prostokąta stanowi ograniczony obszar do analizy przez algorytm Haar-Like. Po znalezieniu twarzy algorytm Haar’a generuje czerwony prostokąt, w którym znajdują się wykryta twarz.



Rysunek 78 Obliczony kontur i nałożony prostokąt stanowiący ROI dla algorytmu Haar-Like



Rysunek 79 Efekt działania algorytmu Haar-Like



Rysunek 80 Interfejs programu oraz efekt końcowy detekcji

4.2.4 Możliwości rozwoju

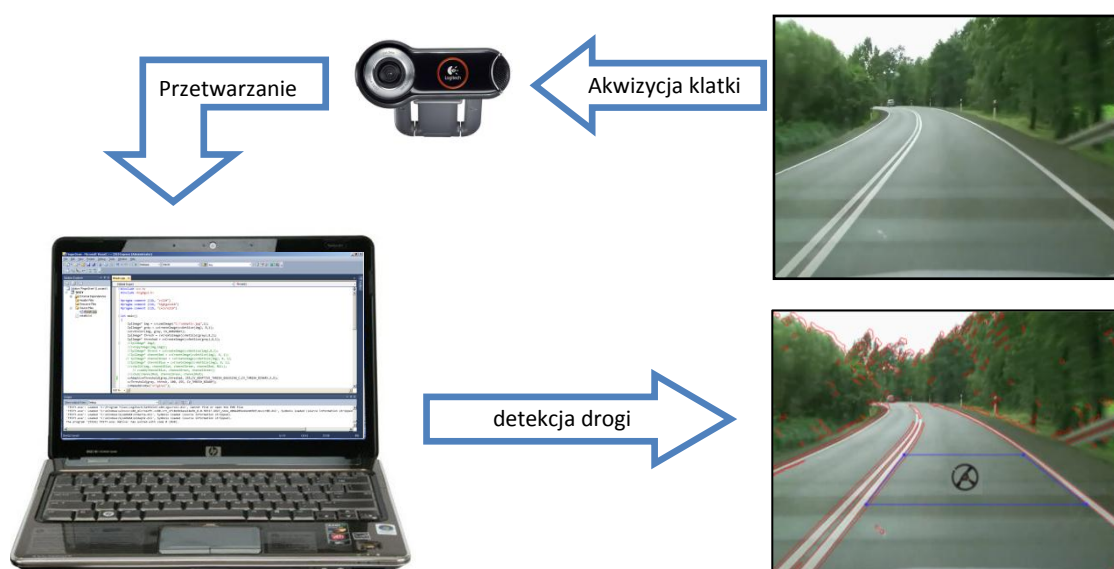
Pierwszym etapem rozwoju programu na pewno byłoby stworzenie własnego klasyfikatora Haar'a, który pozwoliłby na rozpoznawanie twarzy danej osoby. Nie jest to proces trudny jednak wymagający dużego nakładu czasowego.

Następnie program mógłby zostać wyposażony w detekcję nastroju osoby stojącej w kadrze kamery i jej płci. W 2007 roku niemiecki instytut Fraunhofer Institute stworzył bibliotekę pod nazwą SHORE, której aplikacja pozwalała na detekcję osoby, jej nastroju oraz płci. Taki rodzaj oprogramowania mógłby dawać wymierne rezultaty w badaniu opinii klientów na temat reklamy np. na ekranie LCD w sklepie. Takie stanowisko wyposażane w kamerę mogłoby analizować stopień zaniepokojenia oraz nastawienie do przedstawianej reklamy.

4.3 Line Assist

4.3.1 Realizacja

Program „Line Assist” ma za zadanie wykryć linie drogi i na ich podstawie określić sposób sterowania kierownicą samochodu. Program działa w czasie rzeczywistym, analizując na bieżąco stan drogi.



Rysunek 81 Diagram funkcjonowania programu Line Assist.

4.3.2 Problemy

Głównym problemem, był stan nawierzchni drogi, który przejawiał się brakiem pasów, ubytkami w asfalcie oraz zanieczyszczeniami.

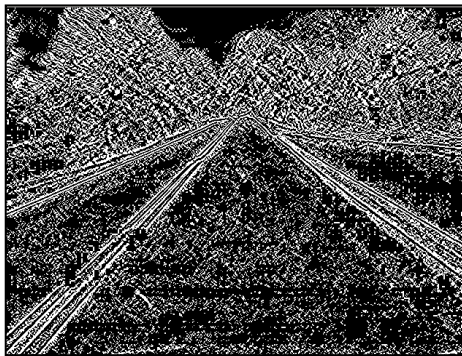
Drugim problemem podczas nagrywania filmu do tego opracowania był brak stabilności kamery. W rozwiązaniach komercyjnych typu „line assist” kamera na stałe jest wbudowana w samochód co ułatwia analizę drogi.

Niewielkie zmiany światła również utrudniały odpowiednie przetworzenie obrazu w celu detekcji samych linii drogi.

4.3.3 Opis działania programu

4.3.3.1 Wykrycie pasów drogi

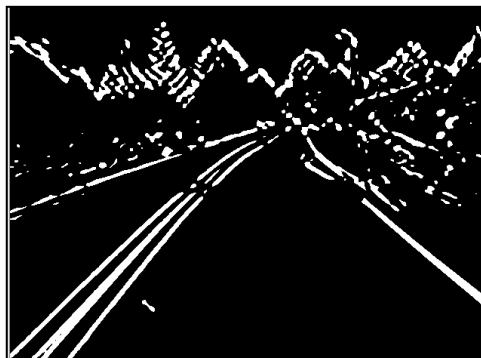
Na początku obraz konwertowany jest do skali szarości, aby mógł zostać przetworzony za pomocą filtru krawędziowego. Następnie wykrywane są krawędzie z wykorzystaniem filtru Sobela (`cvSobel()`). Jego zadaniem jest policzenie pochodnych po X i Y w celu wydobycia granic kolorów między jezdnią (asfaltem) a liniami. Filtr został wybrany w sposób doświadczalny, był on bowiem najbardziej odporny na wahania oświetlenia. W celu ujednolicenia wykrytych konturów pasów zastosowane zostało rozmycie Gaussowskie (`cvSmooth()` z parametrem `CV_GAUSSIAN`). Po tych operacjach obraz jest jednokanałowym obrazem w skali szarości, aby go zbinaryzować i usunąć z niego mniej wyraźne kontury zastosowane zostało progowanie binarne (`cvThreshold()`) z progiem równym 10. W wyniku tej operacji na drodze widoczne były wyraźne pasy jezdni.



Rysunek 82 Obraz po zastosowaniu filtru Sobela ze standardową maską 3x3



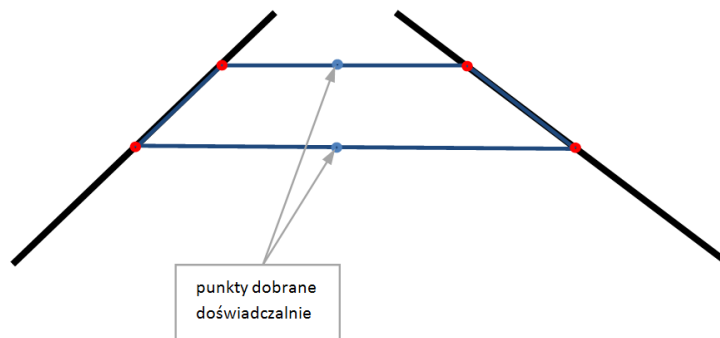
Rysunek 83 Obraz z wykrytymi krawędziami po zastosowaniu rozmycia Gaussowskiego



Rysunek 84 Rozmyty obraz po zastosowaniu progowania binarnego

4.3.3.2 Wyznaczenie punktów do analizy skrętu drogi

Obraz po binaryzacji zostaje poddany funkcjom odnajdywania konturów. Dzięki temu na obrazie zostaną odnalezione punkty, które stanowią kontur jezdni. Kontur ten z sekwencji zapisywany jest do tablicy punktów za pomocą funkcji `cvCvtSeqToArray()`, a następnie dla wygody pracy do wektora punktów. Następnie funkcja `LinePoints()` z doświadczalnie dobranymi punktami odniesienia, wyszukuje punkty z konturu pasów (prawej części pasów środkowych) oraz z konturu zewnętrznej części drogi (po prawej stronie) najbardziej od nich oddalone. Tak wyznaczone cztery punkty (kolor czerwony na rysunku poniżej) służą do analizy skrętu drogi.



Rysunek 85 Sposób wyznaczania punktów jezdni.

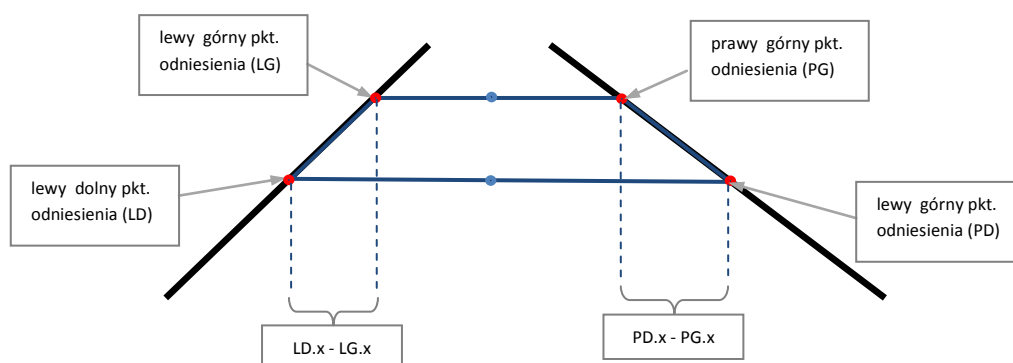
4.3.3.3 Detekcja kierunku oraz skrętu drogi.

Skręt drogi i jego intensywność wyznaczone są na podstawie wzajemnego położenia punktów charakterystycznych środkowego i prawego pasa.

Przy skręcie w prawo analizowane są lewe punkty. Interpretacja odbywa się na podstawie badania aktualnego wychylenia dolnego lewego punktu (LD) w stosunku do lewego górnego punktu odniesienia (LG). Jeżeli różnicą współrzędnych X obu tych punktów znajdowała się w odpowiednim przedziale to było to rozumiane jako skręt w prawo. Dodatkowym zabezpieczeniem był warunek określający wzajemne położenie prawych punktów (dolnego i górnego) w stosunku do siebie. Dopiero po spełnieniu tego drugiego warunku skręt w lewo był wykrywany.

W przypadku badania skrętu w lewo była dokonywana w sposób analogiczny. Wszystkie warunki były dobierane w sposób doświadczalny na podstawie badania zmian nachylenia jezdni w czasie jazdy samochodem.

Kąt skrętu kierownicy w lewą i prawą stronę obliczany był na podstawie wzajemnego położenia lewych oraz prawych punktów odniesienia pomnożonych o współczynnik dobrany również drogą doświadczalną.



Rysunek 86 Sposób wyznaczania kierunku oraz skrętu drogi



Rysunek 87 Sposób działania algorytmu detekcji drogi

4.3.4 Możliwości rozwoju.

Program daje możliwość rozwoju pod kątem oprogramowania oraz pod kątem urządzeń zastosowanych do jego działania. Przede wszystkim kamera, która analizowałaby obraz drogi musiałaby być zamontowana na stałe. Program nie byłby narażony na wzajemną dyslokację kamery i samochodu. Wprowadzone zmiany w oprogramowaniu mogą dotyczyć lepszego odwzorowania skrętu kierownicy na podstawie krzywizny drogi. Z wykorzystaniem analizy ruchu można wprowadzić komunikaty związane z ograniczeniem prędkości samochodu. Kolejnym udoskonaleniem mogłaby być detekcja znaków drogowych również pozwalająca na generowanie ostrzeżeń i dodatkowych komunikatów dla kierowcy. Wyposażenie programu w funkcję detekcji pieszych pozwoliłoby na zachowanie większej ostrożności przez prowadzącego samochód wyposażony w takie oprogramowanie.

Program w wersji rozbudowanej o wspomniane pomysły byłby dobrym narzędziem do zwiększenia bezpieczeństwa na drodze jak oraz wsparciem dla mało doświadczonego kierowcy.

5. Podsumowanie

Celem niniejszej pracy było przedstawienie potencjału biblioteki graficznej OpenCV do tworzenia zaawansowanych systemów wizyjnych. W pierwszej części pracy opisane zostały najbardziej popularne i efektywne funkcje realizujące metody przetwarzania obrazów służące do m.in. rozpoznawania i analizy ruchu obiektów, filtracji, wykrywania konturów czy tworzenia histogramów. W części drugiej na podstawie opisanych algorytmów zostały zaprojektowane trzy systemy, których funkcjonalność pozwalała na wykrywanie i liczenie twarzy na obrazie, rysowanie obiektów geometrycznych za pomocą dwóch rąk jednocześnie (ang. Multi-touch) oraz analizuj

Literatura

- [1] W.Wayt Gibbs, „Innovations from a Robot Rally”, Scientific American 2005
- [2] T. Grey „Color confidence: the digital photographer's guide to color management”, Sybex 2004
- [3] E. Rafajłowicz, W. Rafajłowicz, A. Rusicki „Algorytm przetwarzania obrazów i wstęp do pracy z biblioteką OpenCV”, Oficyna Wydawnicza Politechniki Wrocławskiej. Wrocław 2009
- [4] A. Korzyńska „Przetwarzanie obrazów – ćwiczenia”, Wydawnictwo PJWSTK, Warszawa 2005
- [5] J. Wilk, „Techniki cyfrowego rozpoznawania krawędzi obrazów”, Wydawnictwo Stowarzyszenia Polska-Wchód, Kielce 2006
- [6] R. C. González, R. E. Woods „Digital image Processing”, Pearson Prentice Hall 2008
- [7] E. Davies „Machine Vision: Theory, Algorithms and Practicalities”, Academic Press 1990
- [8] G. Bradski, A. Kaehler, „Learning OpenCV”, O'Reilly 2008
- [9] M. Pawłowski, „Zastosowanie transformacji Hough'a do analizy dwuwymiarowych widm PITS”, Institute of Electronic Materials Technology, Warszawa 2006
- [10] J.F. Rivest, P. Soille, S. Beucher, „Morphological gradients”, Centre de Morphologie Mathematique, Francja 1995
- [11] G. Kukielka, J. Woźnicki, „Praktyczne aspekty wykorzystania metod morfologii matematycznej w cyfrowym przetwarzaniu obrazów”, Materiały konferencyjne z III sympozjum naukowego „Techniki przetwarzania obrazu”, Serock 1997
- [12] J.L. Barron, D.J. Fleet, S.S Beauchemin, „Performance of Optical Flow Techniques”, Kluwer Academic Publishers, USA 1994
- [13] B. Horn, B. Schunck, „Determining Optical Flow using a Modified Horn and Shunck's Algorithm”, Artificial Intelligence Laboratory in MIT 1980
- [14] A.Gyaourova, C. Kamath, C.Cheung „Block matching for object tracking”, Lawrence Livermore National Laboratory 2003
- [15] R. Lienhart, J. Maydt „An Extended Set of Haar-like Features for Rapid Object Detection”, Intel Labs, Intel Corporation, USA 2002
- [16] C. W. Lee, Yong Haur Tay "Empirical study of boosted weak classifier in object detection problem", Universiti Tunku Abdul Rahman, Malaysia 2010
- [17] R. Tadeusiewicz, P. Korohoda „Komputerowa analiza i przetwarzanie obrazów”, Wyd. Fundacji Postępu Telekomunikacji, Kraków 1997

Strony internetowe

- [W1] Strona Wiki projektu biblioteki OpenCV: <http://opencv.willowgarage.com/wiki/>
- [W2] Strona zawierająca aktualne wersje biblioteki:
<http://sourceforge.net/projects/opencv/>
- [W3] Strona Microsoft zawierająca aktualną wersję Visual Studio C++ Express:
<http://www.microsoft.com/express/downloads/>
- [W4] Źródło zdjęcia: <http://daystarvisions.com/Adjust.html>
- [W5] Źródło zdjęcia: http://www.graphx.w8w.pl/ag/img/rgb_max.png
- [W6] Źródło zdjęcia: http://www.mozaikowo.com.pl/graph/gr_cone.png
- [W7] Źródło zdjęcia:
<http://www.cs.bham.ac.uk/resources/courses/robotics/doc/opencvdocs/ref/pics/ellipse.png>
- [W8] Aplet Java:
<http://www.rob.cs.tu-bs.de/content/04-teaching/06-interactive/HNF.html>
- [W9] Strona z instrukcją tworzenia własnego klasyfikatora Haar'a:
<http://note.sonots.com/SciSoftware/haartraining.html>

Spis Rysunków

Rysunek 1 Statystyki pobrań biblioteki OpenCV ze strony projektu.....	11
Rysunek 2 Tworzenie nowego projektu w VS2010 Express.....	12
Rysunek 3 Widok tworzenia nowego projektu	12
Rysunek 4 Okno właściwości projektu	13
Rysunek 5 Okno dodawania dodatkowych ścieżek dostępu	13
Rysunek 6 Architektura OpenCV	14
Rysunek 7 Przekrój tonów skali szarości	22
Rysunek 8 Model RGB przestrzeni kolorów.	23
Rysunek 9 Model HSV przestrzeni kolorów.	24
Rysunek 10 Przykłady zastosowania obsługi myszy w HighGUI.....	31
Rysunek 11 Przykład zastosowania suwaków jako przycisków	33
Rysunek 12 Przykład zastosowania ROI na obrazie.	35
Rysunek 13 Sposób rysowania Elipsy w HighGUI.....	38
Rysunek 14 Progowanie binarne, binarne odwrócone, proste, proste odwrócone, okrawające	41
Rysunek 15 Przykład działania progowania adaptacyjnego.....	42
Rysunek 16 Maska splotu o rozmiarze 3 x 3.	43
Rysunek 17 Obliczanie pierwszej i drugiej pochodnej obszarów o zmiennej jasności [5]	45
Rysunek 18 Zastosowanie różnych operatorów kierunkowych Sobela dla obrazu w skali szarości	47
Rysunek 19 Zastosowanie operatora kombinowanego 3x3 i 5x5 dla filtru Sobela	48
Rysunek 20 Zastosowanie filtru Laplace'a do wykrywania krawędzi.....	49
Rysunek 21 Zastosowanie filtru Canny'ego do detekcji krawędzi	50
Rysunek 22 Zasada działania funkcji <code>cvFindContours()</code>	51
Rysunek 23 Kod łańcuchowy Freemana.....	55
Rysunek 24 Obraz wejściowy przed wykryciem konturów.	56
Rysunek 25 Obraz wejściowy po wykryciu konturów. Dla parametru <code>max_level = -3, -2, -1, 0, 1, 2, 3</code> począwszy od obrazu w lewym górnym rogu.	56
Rysunek 26 Przykładowa maska filtru Gaussa	57
Rysunek 27 Zastosowanie filtru Gaussa dla jednowymiarowej macierzy pikseli [8]	58
Rysunek 28 Zastosowanie rozmycia a - medianowego, b – gaussowskiego.....	58
Rysunek 29 Realizacja klasycznej transformaty Hougha.....	59
Rysunek 30 Realizacja transformaty Hougha dla linii prostych. Obrazy przestrzeni (x, y) oraz przestrzeni parametrycznej (theta, r)	60
Rysunek 31 Zasada wypełniania elementów akumulatora.....	61
Rysunek 32 Wykrywanie okręgów dla CHT przy znanym promieniu R.....	62
Rysunek 33 Programowa detekcja środka okręgu dla CHT.....	63
Rysunek 34 Detekcja okręgów na obrazie za pomocą CHT.	64
Rysunek 35 Dylatacja. Obliczanie wartości piksela za pomocą elementu strukturalnego	66
Rysunek 36 Element strukturalny w kształcie okręgu.....	66
Rysunek 37 Element strukturalny w kształcie prostokąta.	66
Rysunek 38 Element strukturalny w kształcie prostokąta.	66

Rysunek 39 Wykonanie operacji dylatacji na obrazie.....	67
Rysunek 40 Erozja. Obliczanie wartości piksela za pomocą elementu strukturalnego	67
Rysunek 41 Wykonanie operacji erozji na obrazie rzeczywistym	67
Rysunek 42 Wykonanie operacji a - otwarcia i b – zamknięcia na obrazie rzeczywistym	69
Rysunek 43 Wykonanie operacji gradientu morfologicznego na obrazie rzeczywistym	69
Rysunek 44 Wykonanie morfologii a - TopHat, b – BlackHat na obrazie rzeczywistym	70
Rysunek 45 Prosty przykład zastosowania histogramu	70
Rysunek 46 Przykład błędnego oszacowania szerokości koszu histogramu	71
Rysunek 47 Programowa realizacja histogramu w OpenCV	73
Rysunek 48 Programowa realizacja równoważenia histogramu w OpenCV.....	75
Rysunek 49 Obraz wejściowy, histogram zostanie obliczony z obszaru zaznaczonego czerwonym prostokątem	76
Rysunek 50 Obraz po projekcji wstecznej. Wyeksponowane zostały tylko te barwy, które najlepiej pasowały do wartości z histogramu.	76
Rysunek 51 Rzutowanie rzeczywistego wektora ruchu na płaszczyznę obrazu	77
Rysunek 52 Przekształcenie jednego obrazu w drugi za pomocą przepływu optycznego.....	77
Rysunek 53 Różnica pomiędzy polem przepływu optycznego a polem ruchu	78
Rysunek 54 Schemat piramidalnego algorytmu LK.....	79
Rysunek 55 Przykład doboru punktów dobrych do śledzenia	80
Rysunek 56 Efekt działania algorytmu PyrLK. Czerwonymi kropkami zaznaczone zostały punkty dobre do śledzenia, a kolorem niebieski trajektoria zmiany położenia piksela (przepływ optyczny).....	81
Rysunek 57 Zasada działania algorytmu mean-shift.....	85
Rysunek 58 Sposób działania algorytmu cam-shift.....	86
Rysunek 59 Diagram funkcjonowania programu Finger Drawing.....	87
Rysunek 60 Maska wejściowa	89
Rysunek 61 Maska po regulacji suwakami HSV	89
Rysunek 62 Maska po wygładzeniu	89
Rysunek 63 Zaznaczenie obszaru ROI	90
Rysunek 64 Histogram kanału Hue	90
Rysunek 65 Obraz backproject	90
Rysunek 66 Obraz backproject po przetworzeniu	90
Rysunek 67 Wykryty kontur na obrazie wynikowym	91
Rysunek 68 Okna służące do wstępnej kalibracji obrazu.....	92
Rysunek 69 Interfejs użytkownika programu.....	93
Rysunek 70 Diagram funkcjonowania programu Face Detect	95
Rysunek 71 Podstawowy zestaw cech zaproponowany przez Viola i Jones'a	96
Rysunek 72 Podstawowy zestaw cech zaproponowany przez Lienhart'a i Maydt'a	96
Rysunek 73 Przykład zastosowania cech do detekcji twarzy	97
Rysunek 74 Obliczanie obrazu zintegrowanego z oryginalnego	97
Rysunek 75 Obliczanie sumy pikseli regionu R obrazu źródłowego za pomocą czterech punktów obrazu zintegrowanego	97
Rysunek 76 Budowa klasyfikatora kaskadowego.....	98
Rysunek 77 Okno maski po przetworzeniu	99
Rysunek 78 Obliczony kontur i nałożony prostokąt stanowiący ROI dla algorytmu Haar-Like	100

Rysunek 79 Efekt działania algorytmu Haar-Like	100
Rysunek 80 Interfejs programu oraz efekt końcowy detekcji.....	100
Rysunek 81 Diagram funkcjonowania programu Line Assist.	101
Rysunek 81 Obraz po zastosowaniu filtru Sobela ze standardową maską 3x3.....	102
Rysunek 82 Obraz z wykrytymi krawędziami po zastosowaniu rozmycia Gaussowskiego.....	102
Rysunek 83 Rozmyty obraz po zastosowaniu progowania binarnego.....	102
Rysunek 84 Sposób wyznaczania punktów jezdni.	103
Rysunek 85 Sposób wyznaczania kierunku oraz skrętu drogi.....	104
Rysunek 86 Sposób działania algorytmu detekcji drogi.....	104

Dodatek. Opis plików programów

Program Finger Draw

Pliki:

- ✓ `activeareas.h` – plik nagłówkowy z deklaracją podstawowej klasy *ActiveAreas* zawierającej wszystkie aktywne obiekty: przyciski, pointery, aktywne obszary, trackbary oraz funkcje obsługujące wszystkie zdarzenia takie jak: rysowanie, wybieranie opcji rysowania oraz koloru,
- ✓ `activeareas.cpp` – zawiera definicje metod znajdujących się w pliku `activeareas.h`
- ✓ `drawings.h` – plik nagłówkowy z deklaracją klasy *Drawings* zawierającej elementy niezbędne do rysowania obiektów. Od niej dziedziczą inne klasy: *Circle*, *Rectangler*, *Lines*, których obiekty rysowane są przez program.
- ✓ `drawings.cpp` – zawiera definicje metod znajdujących się w pliku `drawings.h`,
- ✓ `functions.h` – plik nagłówkowy z deklaracją pomocniczych funkcji globalnych, ułatwiających realizację takich operacji jak iloczyn skalarny, porównywanie dwóch punktów, obliczanie punktu środkowego prostokąta i inne.
- ✓ `functions.cpp` – zawiera definicje metod znajdujących się w pliku `functions.h`,
- ✓ `main.cpp` – zawiera program główny.

Program Face Detect

- ✓ Plik `main.cpp` – zawiera program główny realizujący wszystkie funkcje niezbędne do detekcji twarzy oraz optymalizacji działania programu.

Program Line Assist

Plik `main.cpp` – zawiera program główny realizujący wszystkie funkcje niezbędne do detekcji drogi i wyznaczania kierunku jej skrętu.