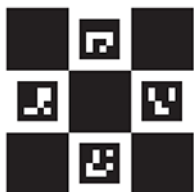


O'REILLY®



# OpenCV 3

Komputerowe rozpoznawanie  
obrazu w C++ przy użyciu  
biblioteki OpenCV



Helion

Adrian Kaehler,  
Gary Rost Bradski

Tytuł oryginału: Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-283-1656-0

© 2018 Helion S.A.

Authorized Polish translation of the English edition of Learning OpenCV 3, ISBN 9781491937990 © 2017 Adrian Kaehler, Gary Bradski

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: [helion@helion.pl](mailto:helion@helion.pl)

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:  
<ftp://ftp.helion.pl/przyklady/opencv.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/opencv>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzje.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

---

# Spis treści

<b>Wstęp .....</b>	<b>15</b>
Przeznaczenie książki .....	15
Adresaci książki .....	16
Czym nie jest ta książka .....	17
O przykładowych programach .....	17
Warunki .....	17
Jak najlepiej korzystać z tej książki .....	18
Konwencje typograficzne .....	19
Podziękowania .....	20
Podziękowania za pomoc przy tworzeniu biblioteki OpenCV .....	20
Podziękowania za pomoc przy pisaniu książki .....	21
Kilka słów od Adriana .....	21
Kilka słów od Gary'ego .....	22
<b>1. Wprowadzenie .....</b>	<b>25</b>
Czym jest OpenCV? .....	25
Kto używa biblioteki OpenCV? .....	26
Czym jest komputerowe rozpoznawanie obrazu? .....	27
Pochodzenie biblioteki OpenCV .....	30
Schemat blokowy OpenCV .....	31
Przyspieszanie działania biblioteki OpenCV za pomocą IPP .....	32
Do kogo należy OpenCV? .....	33
Pobieranie i instalowanie OpenCV .....	33
Instalacja .....	34
Pobieranie najnowszej wersji OpenCV przez Git .....	36
Kompletna dokumentacja OpenCV .....	37
Dokumentacja dołączona do biblioteki .....	37
Dokumentacja internetowa i strona wiki .....	37
Repozytorium kodu od społeczności .....	39
Pobieranie i kompilowanie modułów z repozytorium opencv_contrib .....	40
Przenośność .....	40
Podsumowanie .....	41
Ćwiczenia .....	42

<b>2. Wprowadzenie do OpenCV .....</b>	<b>43</b>
Pliki dołączane .....	43
Zasoby .....	44
Pierwszy program — wyświetlanie obrazu .....	44
Drugi program — wideo .....	47
Przegląd .....	48
Prosta transformacja .....	52
Trochę bardziej skomplikowana transformacja .....	53
Dane z kamery .....	55
Zapisywanie w pliku AVI .....	56
Podsumowanie .....	57
Ćwiczenia .....	58
<b>3. Typy danych OpenCV .....</b>	<b>59</b>
Wiadomości podstawowe .....	59
Typy danych OpenCV .....	59
Przegląd podstawowych typów danych .....	60
Typy podstawowe — szczegóły .....	61
Obiekty pomocnicze .....	68
Funkcje pomocnicze .....	75
Struktury szablonowe .....	81
Podsumowanie .....	82
Ćwiczenia .....	83
<b>4. Obrazy i duże typy tablicowe .....</b>	<b>85</b>
Pamięć dynamiczna i zmienna .....	85
Klasa <code>cv::Mat</code> — n-wymiarowe tablice gęste .....	85
Tworzenie tablicy .....	86
Indywidualny dostęp do elementów tablicy .....	90
N-arny iterator tablicowy — <code>NaryMatIterator</code> .....	93
Dostęp do elementów tablicy według bloków .....	95
Wyrażenia macierzowe — algebra i klasa <code>cv::Mat</code> .....	97
Rzutowanie nasyceniowe .....	98
Co jeszcze potrafią tablice? .....	99
Klasa <code>cv::SparseMat</code> — tablice rzadkie .....	100
Dostęp do elementów tablicy rzadkiej .....	101
Funkcje dostępne tylko dla tablic rzadkich .....	103
Struktury szablonowe dla dużych typów tablicowych .....	104
Podsumowanie .....	106
Ćwiczenia .....	106

<b>5. Operacje na tablicach .....</b>	<b>109</b>
Co jeszcze można zrobić z tablicami? .....	109
cv::abs() .....	112
cv::absdiff() .....	113
cv::add() .....	113
cv::addWeighted() .....	113
cv::bitwise_and() .....	115
cv::bitwise_not() .....	115
cv::bitwise_or() .....	116
cv::bitwise_xor() .....	116
cv::calcCovarMatrix() .....	116
cv::cartToPolar() .....	118
cv::checkRange() .....	119
cv::compare() .....	119
cv::completeSymm() .....	120
cv::convertScaleAbs() .....	120
cv::countNonZero() .....	121
cv::cvarrToMat() .....	121
cv::dct() .....	122
cv::dft() .....	122
cv::cvtColor() .....	123
cv::determinant() .....	126
cv::divide() .....	127
cv::eigen() .....	127
cv::exp() .....	128
cv::extractImageCOI() .....	128
cv::flip() .....	128
cv::gemm() .....	128
cv::getConvertElem() i cv::getConvertScaleElem() .....	129
cv::idct() .....	130
cv::idft() .....	130
cv::inRange() .....	130
cv::insertImageCOI() .....	131
cv::invert() .....	131
cv::log() .....	132
cv::LUT() .....	132
cv::Mahalanobis() .....	133
cv::max() .....	133
cv::mean() .....	135
cv::meanStdDev() .....	135
cv::merge() .....	136
cv::min() .....	136
cv::minMaxIdx() .....	137
cv::minMaxLoc() .....	137
cv::mixChannels() .....	138

cv::mulSpectrums()	140
cv::multiply()	140
cv::mulTransposed()	140
cv::norm()	141
cv::normalize()	142
cv::perspectiveTransform()	143
cv::phase()	144
cv::polarToCart()	145
cv::pow()	145
cv::randu()	145
cv::randn()	146
cv::randShuffle()	146
cv::reduce()	147
cv::repeat()	148
cv::scaleAdd()	148
cv::setIdentity()	148
cv::solve()	149
cv::solveCubic()	150
cv::solvePoly()	150
cv::sort()	151
cv::sortIdx()	151
cv::split()	151
cv::sqrt()	152
cv::subtract()	153
cv::sum()	153
cv::trace()	153
cv::transform()	154
cv::transpose()	154
Podsumowanie	154
Ćwiczenia	155
<b>6. Rysowanie i pisanie</b>	<b>157</b>
Rysowanie	157
Linie i wypełnione wielokąty	157
Czcionki i tekst	163
Podsumowanie	165
Ćwiczenia	166
<b>7. Funktory w OpenCV</b>	<b>167</b>
Obiekty, które „coś” robią	167
Analiza składowych głównych — cv::PCA	167
Rozkład według wartości osobliwych — cv::SVD	170
Generator liczb losowych — cv::RNG	173
Podsumowanie	175
Ćwiczenia	176

<b>8. Pliki obrazów, filmów i danych .....</b>	<b>179</b>
HighGUI — przenośny zestaw narzędzi GUI .....	179
Praca z plikami obrazów .....	180
Ładowanie i zapisywanie obrazów .....	181
Uwaga na temat kodeków .....	183
Kompresja i dekompresja .....	183
Praca z plikami wideo .....	184
Odczytywanie wideo za pomocą obiektu <code>cv::VideoCapture</code> .....	184
Zapisywanie wideo za pomocą obiektu <code>cv::VideoWriter</code> .....	190
Zapisywanie danych .....	191
Zapisywanie danych w obiekcie <code>cv::FileStorage</code> .....	191
Odczytywanie danych z obiektu <code>cv::FileStorage</code> .....	193
<code>cv::FileNode</code> .....	194
Podsumowanie .....	197
Ćwiczenia .....	197
<b>9. Okna wieloplatformowe i macierzyste .....</b>	<b>201</b>
Praca z oknami .....	201
Macierzysty graficzny interfejs użytkownika HighGUI .....	201
Praca z biblioteką Qt .....	212
Integracja OpenCV z kompletnymi bibliotekami GUI .....	221
Podsumowanie .....	233
Ćwiczenia .....	234
<b>10. Filtry i sploty .....</b>	<b>235</b>
Informacje ogólne .....	235
Zanim zaczniesz .....	235
Filtry, jądra i sploty .....	235
Ekstrapolacja krawędzi i warunki brzegowe .....	236
Operacje graniczne .....	240
Algorytm Otsu .....	242
Zmienna wartość graniczna .....	243
Wygładzanie .....	245
Proste rozmazanie i filtr prostokątny .....	246
Filtr medianowy .....	247
Filtr Gaussa .....	248
Filtr bilateralny .....	250
Pochodne i gradienty .....	251
Pochodna Sobela .....	251
Filtr Scharra .....	253
Laplasjan .....	254

Morfologia obrazu .....	255
Dylatacja i erozja .....	257
Ogólna funkcja morfologiczna .....	260
Otwieranie i zamykanie .....	260
Gradient morfologiczny .....	263
Top Hat i Black Hat .....	265
Tworzenie własnego jądra .....	267
Splot z dowolnym filtrem liniowym .....	268
Stosowanie ogólnego filtra przez funkcję <code>cv::filter2D()</code> .....	269
Stosowanie ogólnego filtra rozdzielnego za pomocą funkcji <code>cv::sepFilter2D()</code> .....	270
Funkcje do tworzenia jąder .....	270
Podsumowanie .....	271
Ćwiczenia .....	271
<b>11. Ogólne przekształcenia obrazu .....</b>	<b>277</b>
Wprowadzenie .....	277
Rozciąganie, kurczenie, zniekształcanie i obracanie .....	277
Jednorodna zmiana rozmiaru .....	278
Piramidy obrazów .....	279
Mapowanie niejednorodne .....	282
Przekształcenia afiniczne .....	285
Przekształcenie perspektywiczne .....	289
Ogólne odwzorowania .....	291
Odwzorowania biegunowe .....	292
Współrzędne logarytmiczno-biegunowe .....	294
Odwzorowania arbitralne .....	297
Renowacja obrazów .....	298
Inpainting .....	299
Usuwanie szumów .....	300
Wyrównywanie histogramu .....	303
<code>cv::equalizeHist()</code> — wyrównywanie kontrastu .....	304
Podsumowanie .....	305
Ćwiczenia .....	306
<b>12. Analiza obrazu .....</b>	<b>307</b>
Wprowadzenie .....	307
Dyskretna transformacja Fouriera .....	307
<code>cv::dft()</code> — dyskretna transformacja Fouriera .....	308
<code>cv::idft()</code> — odwrotna dyskretna transformacja Fouriera .....	310
<code>cv::mulSpectrums()</code> — mnożenie widm .....	311
Splot przy użyciu dyskretnej transformacji Fouriera .....	311
<code>cv::dct()</code> — dyskretna transformacja cosinusowa .....	312
<code>cv::idct()</code> — odwrotna dyskretna transformacja cosinusowa .....	313



Obrazy całkowite .....	314
cv::integral() — obliczanie standardowego obrazu całkowego .....	316
cv::integral() — suma kwadratowa .....	316
cv::integral() — suma nachylona .....	317
Detektor krawędzi Canny’ego .....	317
cv::Canny() .....	319
Transformacja Hougha .....	319
Transformacja liniowa Hougha .....	319
Transformacja kołowa Hougha .....	323
Transformacja odległościowa .....	327
cv::distanceTransform() — transformacja odległościowa bez etykiet .....	327
cv::distanceTransform() — transformacja odległościowa z etykietami .....	328
Segmentacja .....	329
Algorytm flood fill .....	329
Algorytm wodorziałowy .....	332
Algorytm Grabcuts .....	334
Metoda segmentacji mean-shift .....	336
Podsumowanie .....	338
Ćwiczenia .....	338
<b>13. Histogramy i szablony .....</b>	<b>341</b>
Reprezentacja histogramów w OpenCV .....	344
cv::calcHist() — tworzenie histogramu z danych .....	344
Podstawowe operacje na histogramach .....	346
Normalizacja histogramu .....	347
Próg histogramu .....	347
Znajdowanie najbardziej zapełnionego przedziału .....	347
Porównywanie histogramów .....	349
Przykłady użycia histogramów .....	351
Zaawansowane metody pracy z histogramami .....	354
Algorytm EMD .....	354
Rzutowanie wstecz .....	358
Dopasowywanie szablonów .....	362
Metoda porównywania kwadratu różnicy — cv::TM_SQDIFF .....	363
Znormalizowana metoda porównywania kwadratu różnicy — cv::TM_SQDIFF_NORMED .....	364
Metody korelacji krzyżowej — cv::TM_CCORR .....	364
Znormalizowana metoda korelacji krzyżowej — cv::TM_CCORR_NORMED .....	364
Metody dopasowywania współczynnika korelacji — cv::TM_CCOEFF .....	364
Znormalizowana metoda dopasowywania współczynnika korelacji — cv::TM_CCOEFF_NORMED .....	365
Podsumowanie .....	367
Ćwiczenia .....	368

<b>14. Kontury .....</b>	<b>371</b>
Znajdowanie konturów .....	371
Hierarchie konturów .....	372
Rysowanie konturów .....	376
Przykład rysowania konturów .....	377
Inny przykład rysowania konturów .....	378
Szybka analiza komponentów połączonych .....	380
Inne zastosowania konturów .....	382
Aproksymacja wielokątów .....	382
Geometria i różne rodzaje sumowania .....	383
Testy geometryczne .....	389
Dopasowywanie konturów i obrazów .....	390
Momenty .....	390
Więcej o momentach .....	392
Dopasowywanie a momenty Hu .....	395
Porównywanie kształtów za pomocą algorytmu Shape Context .....	396
Podsumowanie .....	400
Ćwiczenia .....	401
<b>15. Odejmnowanie tła .....</b>	<b>403</b>
Wiadomości podstawowe .....	403
Wady operacji odejmowania tła .....	404
Modelowanie sceny .....	405
Wycinek pikseli .....	405
Różnicowanie klatek .....	408
Metoda uśredniania tła .....	409
Akumulowanie średnich, wariancji i kowariancji .....	414
Bardziej zaawansowana metoda odejmowania tła .....	422
Struktury .....	425
Poznanwanie tła .....	426
Nauka w obecności ruchomych obiektów pierwszego planu .....	428
Różnicowanie tła — znajdowanie obiektów pierwszego planu .....	429
Przykład użycia modelu tła opartego na księgach kodów .....	430
Kilka dodatkowych uwag na temat modeli opartych na księgach kodów .....	430
Komponenty połączone dla czyszczenia pierwszego planu .....	431
Szybki test .....	434
Porównanie dwóch metod rozpoznawania tła .....	436
Implementacja techniki odejmowania tła w OpenCV .....	437
Klasa bazowa <code>cv::BackgroundSubtractor</code> .....	437
Metoda Kaewtrakulponga i Bowdena .....	438
Metoda Zivkovic'a .....	439
Podsumowanie .....	442
Ćwiczenia .....	442

<b>16. Punkty kluczowe i deskryptory .....</b>	<b>445</b>
Punkty kluczowe i podstawy śledzenia .....	445
Znajdowanie rogów .....	446
Wprowadzenie do przepływu optycznego .....	450
Rzadki przepływ optyczny — metoda Lucasa-Kanade’a .....	451
Uogólnione punkty kluczowe i deskryptory .....	461
Przepływ optyczny, śledzenie i rozpoznawanie .....	462
Jak biblioteka OpenCV obsługuje punkty kluczowe i deskryptory — przypadek ogólny .....	463
Podstawowe metody wykrywania punktów kluczowych .....	474
Filtrowanie punktów kluczowych .....	512
Metody dopasowywania .....	513
Wyświetlanie wyników .....	519
Podsumowanie .....	521
Ćwiczenia .....	522
<b>17. Śledzenie .....</b>	<b>525</b>
Pojęcia dotyczące śledzenia .....	525
Gęsty przepływ optyczny .....	526
Algorytm rozwinięcia wielomianu Farnebäcka .....	527
Algorytm Dual TV-L <sup>1</sup> .....	529
Algorytm Simple Flow .....	533
Algorytmy mean-shift i Camshift .....	536
Algorytm mean-shift .....	537
Algorytm Camshift .....	540
Szablony ruchu .....	541
Estymatory .....	548
Filtr Kalmana .....	549
Kilka słów na temat rozszerzonego filtra Kalmana .....	564
Podsumowanie .....	565
Ćwiczenia .....	565
<b>18. Modele kamery i metody kalibracji .....</b>	<b>567</b>
Model kamery .....	568
Podstawy geometrii rzutowej .....	570
Transformacja Rodriguesa .....	572
Zniekształcenia soczewek .....	573
Kalibracja .....	576
Macierz obrotu i wektor przesunięcia .....	578
Plansze kalibracji .....	580
Homografia .....	587
Kalibracja aparatu .....	591
Korekcja zniekształceń .....	602

Matryce likwidacji zniekształceń .....	602
Konwertowanie reprezentacji matryc likwidacji zniekształceń za pomocą funkcji <code>cv::convertMaps()</code> .....	604
Obliczanie matryc likwidacji zniekształceń za pomocą funkcji <code>cv::initUndistortRectifyMap()</code> .....	604
Likwidowanie zniekształceń obrazu za pomocą funkcji <code>cv::remap()</code> .....	606
Likwidacja zniekształceń za pomocą funkcji <code>cv::undistort()</code> .....	606
Rzadkie likwidowanie zniekształceń za pomocą funkcji <code>cv::undistortPoints()</code> .....	607
Podsumowanie technik kalibracji .....	607
Podsumowanie .....	610
Ćwiczenia .....	611
<b>19. Rzutowanie i trójwymiarowe widzenie .....</b>	<b>613</b>
Rzutowanie .....	614
Przekształcenia afiniczne i perspektywiczne .....	615
Przykład przekształcenia na widok z góry .....	616
Określanie pozycji trójwymiarowych obiektów .....	621
Określanie pozycji za pomocą jednej kamery .....	621
Obrazowanie stereo .....	623
Triangulacja .....	624
Geometria epipolarna .....	628
Macierz zasadnicza i macierz fundamentalna .....	630
Obliczanie linii epipolarnych .....	638
Kalibracja stereo .....	639
Rektyfikacja stereo .....	643
Korespondencja stereo .....	652
Kalibracja stereo, rektyfikacja i korespondencja — przykładowy program .....	665
Matryce głębi z trójwymiarowej reprojekcji .....	671
Struktura z ruchu .....	673
Dopasowywanie linii w dwóch i trzech wymiarach .....	673
Podsumowanie .....	676
Ćwiczenia .....	677
<b>20. Podstawy uczenia maszyn w OpenCV .....</b>	<b>679</b>
Czym jest uczenie maszyn? .....	679
Szkolenie i zbiory danych testowych .....	680
Uczenie nadzorowane i nienadzorowane .....	681
Modele generacyjne i dyskryminacyjne .....	683
Algorytmy ML w OpenCV .....	683
Zastosowania metod uczenia maszyn w komputerowym rozpoznawaniu obrazu .....	685
Znaczenie zmiennej .....	687
Diagnozowanie usterek w algorytmach uczenia maszyn .....	688

Stare procedury w bibliotece ML .....	694
K-średnich .....	694
Odległość Mahalanobisa .....	699
Podsumowanie .....	703
Ćwiczenia .....	703
<b>21. StatModel — standardowy model uczenia w OpenCV .....</b>	<b>705</b>
Podstawowe procedury biblioteki ML .....	705
Szkolenie i struktura cv::ml::TrainData .....	707
Prognozowanie .....	713
Algorytmy uczenia maszyn oparte na interfejsie cv::StatModel .....	714
Naiwny lub normalny klasyfikator Bayesa .....	714
Binarne drzewa decyzyjne .....	719
Boosting .....	732
Drzewa losowe .....	738
Maksymalizacja oczekiwań .....	742
K najbliższych sąsiadów .....	745
Wielowarstwowy perceptron .....	748
Maszyna wektorów nośnych .....	756
Podsumowanie .....	765
Ćwiczenia .....	766
<b>22. Wykrywanie obiektów .....</b>	<b>771</b>
Techniki wykrywania obiektów oparte na drzewach .....	771
Klasyfikatory kaskadowe .....	772
Uczenie nadzorowane i teoria wzmacniania .....	774
Uczenie się nowych obiektów .....	782
Wykrywanie obiektów za pomocą maszyn wektorów nośnych .....	790
Wykrywanie obiektów metodą Latent SVM .....	790
Algorytm Bag of Words i kategoryzacja semantyczna .....	793
Podsumowanie .....	798
Ćwiczenia .....	798
<b>23. Przyszłość biblioteki OpenCV .....</b>	<b>801</b>
Przeszłość i teraźniejszość .....	801
OpenCV 3.x .....	802
Czy nasze prognozy się sprawdziły? .....	803
Przyszłość .....	804
Najświeższe wiadomości o GSoC .....	805
Wsparcie ze strony społeczności .....	807
Strona OpenCV.org .....	808
Spekulacje na temat sztucznej inteligencji .....	809
Poślowie .....	812

<b>A Rodzaje podziału płaszczyzn .....</b>	<b>813</b>
Triangulacja Delaunaya, teselacja Woronoja .....	813
Tworzenie podziałów Delaunaya lub Woronoja .....	816
Poruszanie się po podziałach Delaunaya .....	817
Przykłady .....	823
Ćwiczenia .....	824
<b>B Repozytorium opencv_contrib .....</b>	<b>825</b>
Przegląd modułów repozytorium opencv_contrib .....	825
Zawartość katalogu opencv_contrib .....	825
<b>C Wzory kalibracji .....</b>	<b>829</b>
Wzory kalibracji wykorzystywane w bibliotece OpenCV .....	829
<b>Bibliografia .....</b>	<b>835</b>
<b>Skorowidz .....</b>	<b>849</b>

---

# Ogólne przekształcenia obrazu

## Wprowadzenie

W poprzednich rozdziałach opisaliśmy klasę przekształceń obrazu, które można przedstawić na podstawie splotu. Oczywiście istnieje wiele operacji, których w ten sposób nie da się wyrazić (tzn. przy użyciu małego okienka przesuwanego nad obrazem i wykonującego jakieś czynności). Zasadniczo przekształcenia, które można wyrażać jako sploty, mają charakter lokalny, to znaczy choć mogą zmieniać cały obraz, sposób przetworzenia konkretnego piksela zależy tylko od niewielkiej liczby otaczających go pikseli. Transformacje opisywane w tym rozdziale są pozbawione tej właściwości.

Niektóre z najbardziej przydatnych *przekształceń obrazu* są proste (np. zmiana rozmiaru) i stosuje się je bardzo często. Inne są przeznaczone do znacznie bardziej zaawansowanych zastosowań. Przedstawione przez nas w tym rozdziale przekształcenia powodują przekonwertowanie jednego obrazu w inny. Obraz wynikowy często będzie miał inny rozmiar niż oryginał albo będzie się od niego różnił pod innym względem, ale zawsze będzie zasadniczo obrazem w takim samym sensie jak grafika wejściowa. W rozdziale 12. omawiamy także operacje, które w wyniku mogą zwrócić całkiem nową reprezentację.

W komputerowym rozpoznawaniu obrazu jest kilka przekształceń, których używa się bardzo często. Biblioteka OpenCV zawiera kompletne implementacje niektórych najpopularniejszych spośród nich oraz składniki potrzebne do implementacji własnych, bardziej zaawansowanych transformacji.

## Rozciąganie, kurczenie, zniekształcanie i obracanie

Najprostszym możliwym rodzajem przekształcenia jest zmiana rozmiaru obrazu w celu jego powiększenia lub zmniejszenia. Operacje te jednak nie są wcale tak banalne, jak się wydaje, ponieważ zmiana rozmiaru natychmiast implikuje pytania o sposób interpolacji (w przypadku powiększania) lub łączenia (w przypadku zmniejszania) pikseli.

## Jednorodna zmiana rozmiaru

Często otrzymujemy obraz o określonym rozmiarze i chcemy go zmniejszyć lub powiększyć. Oba te zadania możemy wykonać za pomocą jednej funkcji.

### cv::resize()

Funkcja `cv::resize()` zaspokaja wszystkie potrzeby programisty w dziedzinie zmiany rozmiaru obrazów. Należy jej przekazać na wejściu obraz i określić rozmiar docelowy, a funkcja wygeneruje nowy obraz o żądanym rozmiarze.

```
void cv::resize(  
    cv::InputArray src,           // obraz wejściowy  
    cv::OutputArray dst,        // obraz wynikowy  
    cv::Size dsize,             // nowy rozmiar  
    double fx = 0,              // skala wymiaru x  
    double fy = 0,              // skala wymiaru y  
    int interpolation = CV::INTER_LINEAR // metoda interpolacji  
);
```

Rozmiar obrazu wynikowego można określić na dwa sposoby. Pierwszy polega na podaniu rozmiaru *bezwzględnego* w argumentcie `dsize`, który bezpośrednio określa, jaki rozmiar ma mieć przekształcony obraz `dst`. Drugi sposób polega na określeniu wartości *względnych*. W tym przypadku argument `dsize` ustawia się na `cv::Size(0,0)`, a następnie argumentom `fx` i `fy` przypisuje się współczynniki skalowania odpowiednio dla osi  $x$  i  $y$ <sup>1</sup>. Ostatni argument wskazuje metodę interpolacji. Domyślnie stosowana jest interpolacja liniowa. Pozostałe dostępne opcje są pokazane w tabeli 11.1.

Tabela 11.1. Opcje interpolacji funkcji `cv::resize()`

Interpolacja	Opis
<code>cv::INTER_NEAREST</code>	Najbliższy sąsiad
<code>cv::INTER_LINEAR</code>	Dwuliniowa
<code>cv::INTER_AREA</code>	Przepróbkowywanie obszaru pikseli
<code>cv::INTER_CUBIC</code>	Interpolacja dwusześcianaowa
<code>cv::INTER_LANCZOS4</code>	Interpolacja algorytmem Lanczosa uwzględniająca sąsiadujące punkty w kwadracie o wymiarach $8 \times 8$

Interpolacja w tym przypadku odgrywa bardzo ważną rolę. Piksele w obrazie źródłowym są rozmieszczone w siatce całkowitoliczbowej, dzięki czemu możemy odwołać się na przykład do piksela o współrzędnych (20, 17). Podczas odwzorowywania tych całkowitoliczbowych współrzędnych w nowym obrazie mogą powstawać luki. Przyczyną powstawania takich luk może być odwzorowywanie współrzędnych całkowitoliczbowych na zmiennoprzecinkowe, które muszą zostać zaokrąglone do najbliższej liczby całkowitej, albo to, że dla niektórych lokalizacji nie ma żadnego piksela (pomyśl o podawaniu rozmiaru obrazu przez jego rozciągnięcie — w takim przypadku co drugi piksel docelowy byłby pusty). Tego typu trudności określa się zbiorczo mianem problemów **rzutowania w przód** (ang. *forward projection*). Aby je rozwiązać, musimy podejść do tego od drugiej strony, a mianowicie przeglądamy po kolei każdy piksel obrazu docelowego i pytamy: które piksele

<sup>1</sup> Albo argument `dsize` musi mieć wartość `cv::Size(0,0)`, albo argumenty `fx` i `fy` muszą mieć wartość 0.



ze źródła mogą zostać wykorzystane w tym miejscu? Wybrane piksele źródłowe prawie zawsze wypadają w lokalizacjach ułamkowych, więc musimy dokonywać ich interpolacji, aby otrzymać prawidłową wartość dla naszej wartości docelowej. Domyślną metodą jest interpolacja dwuliniowa, ale do wyboru są także inne metody (wymienione w tabeli 11.1).

Najprostszym rozwiązaniem jest pobranie wartości dla piksela w obrazie rozszerzonym z najbliższego mu piksela w pierwotnym obrazie. Tak właśnie działa opcja interpolacji `cv::INTER_NEAREST`. Ewentualnie można wykorzystać sąsiednie piksele źródłowe z obszaru o wymiarach  $2 \times 2$ , ważąc je z uwzględnieniem ich odległości od piksela docelowego — tak działa opcja `cv::INTER_LINEAR`. Można też wirtualnie umieścić nowy piksel na starym pikselu, a następnie obliczyć średnią pokrytych wartości pikseli — tak działa metoda `cv::INTER_AREA`<sup>2</sup>. Jeszcze lepszy efekt wygładzania można uzyskać, wpasowując sześcienną krzywą składaną między sąsiadującymi pikselami na obszarze o wymiarach  $4 \times 4$  w obrazie źródłowym, a następnie odczytując odpowiednią wartość docelową z tej krzywej — tak działa metoda interpolacji `cv::INTER_CUBIC`. Ostatnia jest interpolacja algorytmem Lanczosa, która przypomina metodę sześcienną, ale wykorzystuje informacje z obszaru wokół piksela o wymiarach  $8 \times 8$ <sup>3</sup>.



Należy podkreślić różnicę między funkcją `cv::resize()` i podobnie nazwaną funkcją składową klasy `cv::Mat` — `cv::Mat::resize()`. Funkcja `cv::resize()` tworzy nowy obraz o innym rozmiarze niż oryginał, na którym odwzorowane są piksele z pierwotnej grafiki. Funkcja składowa `cv::Mat::resize()` zmienia rozmiar obrazu, którego składową wywołano, i przycina go do odpowiedniego rozmiaru. W przypadku funkcji `cv::Mat::resize()` piksele nie są interpolowane (ani ekstrapolowane).

## Piramidy obrazów

Piramidy obrazów [Adelson84] znajdują bardzo szerokie zastosowanie w wielu dziedzinach komputerowego rozpoznawania obrazu. Piramida obrazów to zbiór grafik (wszystkie utworzone na bazie jednego podstawowego obrazu) o stopniowo zmniejszanym próbkowaniu, aż do osiągnięcia pewnego punktu końcowego (punkt ten może być nawet jednopikselowym obrazem!).

W literaturze przedmiotu i praktyce wyróżnia się dwa podstawowe rodzaje piramid obrazów: Gaussa [Rosenfeld80] i Laplace'a [Burt83]. Piramida Gaussa służy do zmniejszania próbkowania, a piramida Laplace'a (którą opisujemy dalej) jest używana, gdy trzeba odtworzyć obraz o większym próbkowaniu na podstawie obrazu z niższego poziomu piramidy.

### `cv::pyrDown()`

Normalnie warstwę  $(i + 1)$  w piramidzie Gaussa (oznaczymy tę warstwę  $G_{i+1}$ ) tworzy się z warstwy  $G_i$  piramidy, najpierw splatając  $G_i$  z jądrem gaussowskim, a następnie usuwając każdy rząd i każdą kolumnę o parzystym numerze. Oczywiście wówczas otrzymuje się obraz, którego rozmiar

<sup>2</sup> Tak to przynajmniej wygląda, gdy funkcja `cv::resize()` zmniejsza obraz. W przypadku powiększania opcja `cv::INTER_AREA` daje taki sam efekt jak `cv::INTER_NEAREST`.

<sup>3</sup> Szczegółowy opis filtra Lanczosa wykracza poza zakres tematyczny tej książki, ale warto wiedzieć, że algorytm ten jest powszechnie wykorzystywany w przetwarzaniu obrazów cyfrowych, ponieważ zwiększa *pozorną* ostrość obrazu.

wynosi dokładnie jedną czwartą rozmiaru poprzedniego obrazu. Wielokrotne przeprowadzenie tego procesu na obrazie wejściowym  $G_0$  pozwala otrzymać całą piramidę. Biblioteka OpenCV udostępnia metodę do generowania każdego poziomu piramidy z poprzedniego:

```
void cv::pyrDown(  
    cv::InputArray src,           // obraz wejściowy  
    cv::OutputArray dst,        // obraz wynikowy  
    const cv::Size& dstsize = cv::Size() // rozmiar obrazu wyjściowego  
);
```

Metoda `cv::pyrDown()` robi dokładnie to, o czym napisaliśmy wyżej, jeśli pozostawimy argument rozmiaru docelowego `dstsize` przy wartości domyślnej `cv::Size()`. Mówiąc dokładniej: domyślny rozmiar obrazu wyjściowego wynosi  $((src.cols+1)/2, (src.rows+1)/2)^4$ . Ewentualnie możemy przekazać argument `dstsize`, aby określić rozmiar obrazu wyjściowego. Wartość tego argumentu musi jednak spełniać pewne bardzo ściśle warunki:

$$|dstsize.width \cdot 2 - src.cols| \leq 2$$

$$|dstsize.height \cdot 2 - src.rows| \leq 2$$

Wskutek tego ograniczenia obraz wynikowy ma rozmiar *bardzo bliski* połowie rozmiaru obrazu źródłowego. Argumentu `dstsize` używa się tylko w rzadkich przypadkach, gdy trzeba ściśle kontrolować sposób tworzenia piramidy.

## cv::buildPyramid()

Stosunkowo często zdarzają się sytuacje, gdy trzeba utworzyć zbiór obrazów, z których każdy kolejny jest pomniejszoną wersją poprzedniego. Do tworzenia takich stosów służy funkcja `cv::buildPyramid()`:

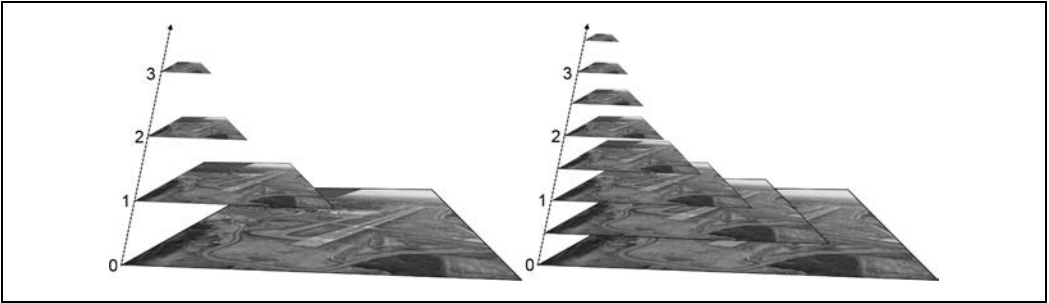
```
void cv::buildPyramid(  
    cv::InputArray src,           // obraz wejściowy  
    cv::OutputArrayOfArrays dst, // obrazy piramidy  
    int maxlevel                  // liczba poziomów piramidy  
);
```

Argument `src` określa obraz źródłowy. Z kolei argument `dst` ma dość niezwykły typ `cv::OutputArrayOfArrays`, który można traktować jako wektor STL `vector<>` obiektów typu `cv::OutputArray`. Najlepszym przykładem takiej struktury jest `vector<cv::Mat>`. Natomiast argument `maxlevel` sygnalizuje, ile poziomów ma mieć tworzona piramida.

Argument `maxlevel` może mieć dowolną wartość całkowitoliczbową nie mniejszą od zera, ponieważ wskazuje, z ilu obrazów ma się składać wygenerowana piramida. Po zakończeniu działania funkcja `cv::buildPyramid()` zwraca w argumentcie `dst` wektor o długości `maxlevel+1`. Pierwszy element w tablicy `dst` jest identyczny z obrazem `src`. Drugi jest o połowę mniejszy od pierwszego, to znaczy jest taki, jaki zostałby zwrócony przez funkcję `cv::pyrDown()`. Trzeci obraz jest o połowę mniejszy od drugiego itd. (ilustracja po lewej na rysunku 11.1).

---

<sup>4</sup> Składniki  $+1$  są po to, aby zapewnić prawidłową obsługę obrazów o nieparzystych rozmiarach. W przypadku obrazów o rozmiarach parzystych nie mają znaczenia.



Rysunek 11.1. Piramida obrazów wygenerowana przy użyciu argumentu `maxlevel = 3` (po lewej); dwie przeplatające się piramidy tworzące razem piramidę  $\sqrt{2}$  (po prawej)



Piramidy wykorzystywane w praktyce często są generowane przy użyciu drobniejszych logarytmicznych współczynników skalowania. Jednym ze sposobów pozwalających wygenerować taką piramidę jest własnoręczne wywołanie funkcji `cv::resize()` odpowiednią liczbę razy, ale to dość czasochłonne rozwiązanie. Innym sposobem (w przypadku niektórych często używanych współczynników skalowania) jest wywołanie funkcji `cv::resize()` tylko raz dla każdego z zestawów obrazów, które mają zostać później splecione, a następnie przekazanie każdej z tak otrzymanych „podstaw” do funkcji `cv::buildPyramid()`. Kolejną czynnością jest właśnie złożenie otrzymanych obrazów na przemian w celu utworzenia gęstszej piramidy. Na rysunku 11.1 (po prawej) pokazany jest przykład wygenerowania dwóch piramid. Pierwotny obraz został najpierw przeskalowany o współczynnik  $\sqrt{2}$ , a następnie przekazano go do funkcji `cv::buildPyramid()` w celu utworzenia drugiej piramidy złożonej z czterech obrazów. Po połączeniu nowego zestawu z poprzednią piramidą powstała nowa, gęstsza piramida o współczynniku skalowania  $\sqrt{2}$ .

### `cv::pyrUp()`

Wybrany obraz można też przekonwertować na dwa razy większy w każdym wymiarze. Służy do tego operacja analogiczna (ale nie odwrotna!) do opisaney wcześniej operacji, której rezultatem są zmniejszone obrazy:

```
void cv::pyrUp(
    cv::InputArray src,           // obraz wejściowy
    cv::OutputArray dst,        // obraz wynikowy
    const cv::Size& dstsize = cv::Size() // rozmiar obrazu wynikowego
);
```

Funkcja ta najpierw powiększa obraz dwukrotnie w każdym wymiarze i nowe (parzyste) wiersze wypełnia zerami, a następnie aproksymuje „brakujące” wartości pikseli za pomocą splotu z filtrem Gaussa<sup>5</sup>.

Analogicznie jak w przypadku funkcji `cv::PyrDown()`, jeżeli argument `dstsize` będzie miał pozostawioną wartość domyślną `cv::Size()`, wygenerowany obraz będzie dokładnie dwa razy większy (w każdym wymiarze) niż obraz `src`. Oczywiście także w tym przypadku w argumencie `dstsize`

<sup>5</sup> Filtr ten jest również znormalizowany do wartości 4, a nie 1. Jest to właściwe, ponieważ wstawiane wiersze przed splotem mają wszystkie piksele ustawione na zero. (Normalnie suma elementów jądra gaussowskiego wynosiłaby 1, ale w przypadku dwukrotnego skalowania piramidy — jeśli chodzi o dwa wymiary — wszystkie elementy jądra są mnożone przez 4, aby przywrócić średnią jasność po wstawieniu zerowych wierszy i kolumn).

możemy przekazać własną wartość określającą rozmiar obrazu wynikowego, ale musi ona spełniać pewne ściśle warunki:

$$|\text{dstsize.width} \cdot 2 - \text{src.cols}| \leq (\text{dstsize.width} / 2)$$

$$|\text{dstsize.height} \cdot 2 - \text{src.rows}| \leq (\text{dstsize.height} / 2)$$

Z warunku tego wynika, że rozmiar obrazu docelowego musi być *bardzo bliski* dwukrotności rozmiaru obrazu źródłowego. Tak jak poprzednio, argument `dstsize` jest wykorzystywany tylko w stosunkowo rzadkich przypadkach, gdy trzeba dokładnie kontrolować sposób tworzenia piramidy.

## Piramida Laplace'a

Wcześniej napisaliśmy, że działanie operatora `cv::pyrUp()` nie jest odwrotnością sposobu działania operatora `cv::pyrDown()`. Powinno to być oczywiste, ponieważ `cv::pyrDown()` gubi informacje. Do odzyskania pierwotnego obrazu o wyższej rozdzielczości potrzebne są informacje, które wcześniej zostały skasowane w procesie redukcji. I właśnie te dane stanowią **piramidę Laplace'a** (ang. *Laplacian pyramid*). Definicja  $i$ -tej warstwy tej piramidy ma postać następującej relacji:

$$L_i = G_i - UP(G_{i+1}) \otimes g_{5 \times 5}$$

W procesie zwiększania rozmiaru przez operator `UP()` każdy piksel o współrzędnych  $(x, y)$  z pierwotnego obrazu jest odwzorowywany na piksel o współrzędnych  $(2x + 1, 2y + 1)$  w obrazie docelowym. Symbolem  $\otimes$  oznaczany jest splot, a wyraz  $g_{5 \times 5}$  oznacza jądro gaussowskie o wymiarach  $5 \times 5$ . Oczywiście  $UP(G_{i+1}) \otimes g_{5 \times 5}$  to definicja operatora `cv::pyrUp()` z biblioteki OpenCV. W związku z tym możemy bezpośrednio przedstawić operator Laplace'a w następującej postaci:

$$L_i = G_i - \text{pyrUp}(G_{i+1})$$

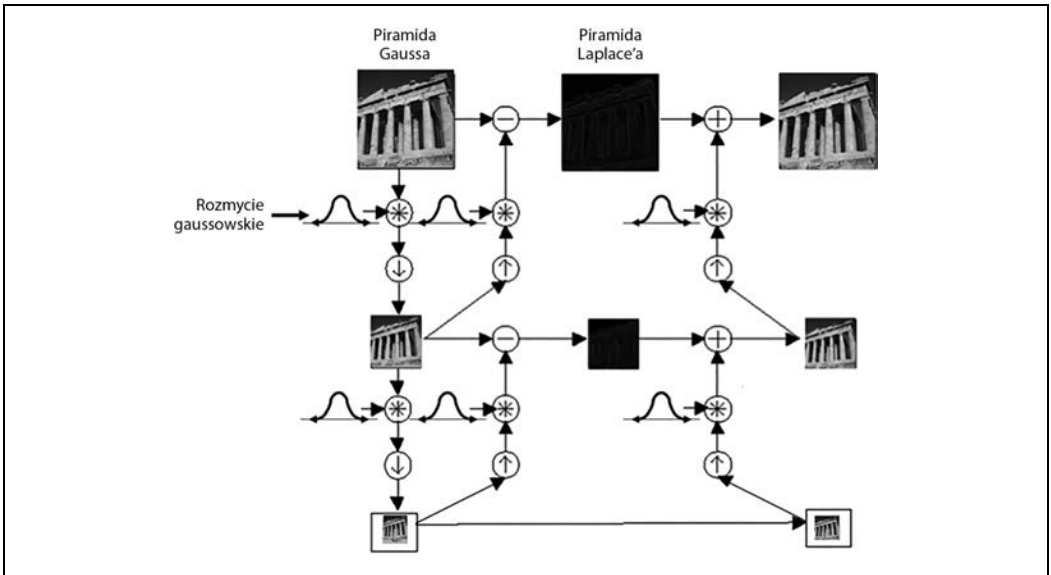
Rysunek 11.2 przedstawia piramidy Gaussa i Laplace'a w formie schematu, na którym dodatkowo widać odwrotny proces mający na celu przywrócenie pierwotnego obrazu ze zredukowanych obrazów. Zwróć uwagę, że obraz piramidy Laplace'a jest tak naprawdę aproksymacją wykorzystującą różnicę obrazów z piramidy Gaussa, co wynika z poprzedniego równania i jest widoczne na schemacie.

## Mapowanie niejednorodne

W tej sekcji kierujemy uwagę na *geometryczne* operacje na obrazach, to znaczy takie przekształcenia, które biorą swój początek na przecięciu geometrii trójwymiarowej i projekcyjnej<sup>6</sup>. Do operacji takich zaliczają się zarówno jednolite, jak i niejednolite zmienianie rozmiaru (to drugie nazywa się **odkształcaniem** — ang. *warping*). Działania takie mogą być wykonywane z wielu powodów, na przykład można odkształcić i obrócić obraz w celu nałożenia go na ścianę na scenie lub sztucznie powiększyć zbiór obrazów szkoleniowych wykorzystywanych w treningu rozpoznawania obiektów<sup>7</sup>.

<sup>6</sup> Szczegółowy opis tych przekształceń zamieściliśmy w tej sekcji, ale wracamy do nich jeszcze w rozdziale 19., przy okazji omawiania metod ich użycia w trójwymiarowych technikach wizyjnych.

<sup>7</sup> Ta czynność może się wydawać nieco naciągana, bo przecież równie dobrze można skorzystać z metody rozpoznawania, która jest niezależna od lokalnych zniekształceń afinicznych. Niemniej jednak metoda ta ma długą historię i niejednokrotnie okazała się niezwykle przydatna w praktyce.



Rysunek 11.2. Piramida Gaussa i jej odwrotność, czyli piramida Laplace'a

Funkcje rozciągające, ściskające, zniekształcające lub obracające obrazy nazywają się **transformacjami geometrycznymi** (wcześnie przykład można znaleźć w [Semple79]). W odniesieniu do płaszczyzn wyróżnia się dwa rodzaje transformacji geometrycznych: **afiniczne**, które bazują na macierzach o wymiarach  $2 \times 3$ , oraz **perspektywiczne**, zwane też **homografiami**, które bazują na macierzach o wymiarach  $3 \times 3$ . Ten drugi rodzaj transformacji można sobie wyobrazić jako metodę obliczania tego, w jaki sposób dana płaszczyzna w trzech wymiarach jest postrzegana przez wybranego obserwatora, który może nie patrzeć na nią wprost.

Transformacja afiniczna to każda transformacja, którą można wyrazić w postaci mnożenia macierzy, po którym następuje dodanie wektora. W OpenCV ten rodzaj transformacji standardowo przedstawia się w postaci macierzy o wymiarach  $2 \times 3$ . Mamy następującą definicję:

$$A \equiv \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} B \equiv \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} T \equiv [A \quad B] X \equiv \begin{bmatrix} x \\ y \end{bmatrix} X' \equiv \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

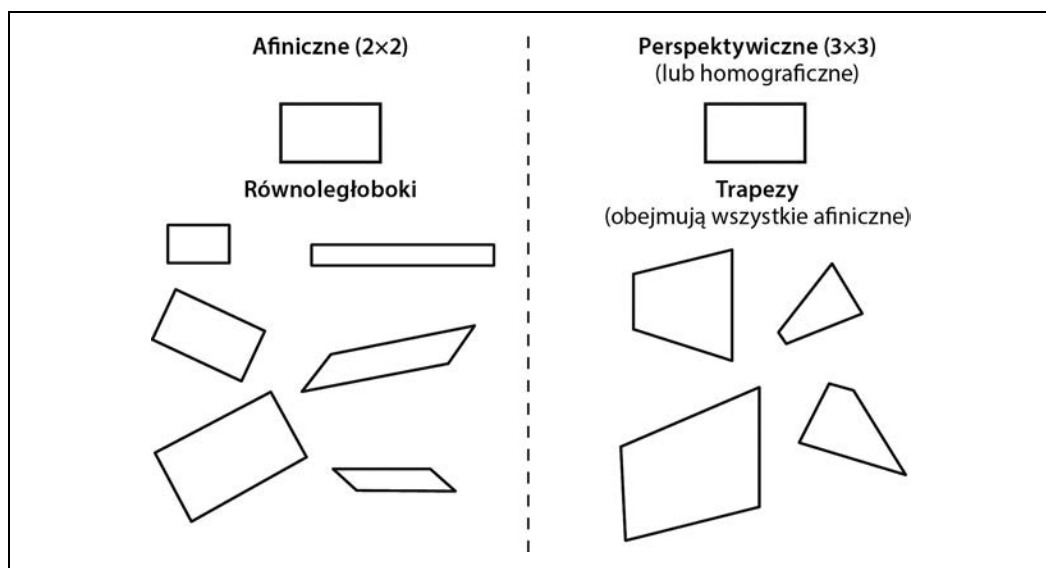
Nietrudno dostrzec, że wynik transformacji afinicznej  $A \cdot X + B$  jest dokładnie taki sam jak wynik rozszerzenia wektora  $X$  do postaci  $X'$  i lewostronnego pomnożenia wektora  $X'$  przez  $T$ .

Przekształcenie afiniczne można sobie wyobrazić w następujący sposób: każdy równoległobok  $ABCD$  na płaszczyźnie można odwzorować na każdy inny równoległobok  $A'B'C'D'$  przez zastosowanie przekształcenia afinicznego. Jeżeli pola powierzchni tych równoległoboków są większe od zera, oznacza to, że dane przekształcenie afiniczne jest jednoznacznie zdefiniowane przez te dwa równoległoboki (ich trzy wierzchołki). Jeśli ktoś woli, to przekształcenie afiniczne może sobie wyobrażać jako rysowanie obrazu na dużym arkuszu gumy i zniekształcanie go poprzez rozciąganie i ściskanie<sup>8</sup> na rogach dla uzyskania różnych rodzajów równoległoboków.

<sup>8</sup> Równoległobok można nawet tak rozciągnąć, aby go odwrócić.

Jeśli danych jest kilka obrazów, z których każdy przedstawia ten sam obiekt w nieco inny sposób, to można obliczyć, jakie konkretnie przekształcenia zostały zastosowane w celu otrzymania tych różnych widoków. W takim przypadku często wykorzystuje się przekształcenia afiniczne zamiast perspektywicznych, ponieważ mają mniej parametrów, dzięki czemu łatwiej jest wykonać związane z nimi obliczenia. Wadą tego rozwiązania jest to, że prawdziwe zniekształcenia perspektywiczne można modelować wyłącznie na gruncie homografii<sup>9</sup>, a przekształcenia afiniczne pozwalają uzyskać reprezentację niebędącą w stanie objąć wszystkich możliwych relacji między widokami. Z drugiej strony, jeśli zmiana perspektywy jest niewielka, zniekształcenie może mieć charakter afiniczny i wówczas przekształcenie afiniczne może być w zupełności wystarczające.

Za pomocą przekształcenia afinicznego można zamienić prostokąt w równoległobok. Figura może zostać ściśnięta, ale boki muszą pozostać równoległe. Istnieje też możliwość dokonania obrotu i zmiany rozmiaru. Przekształcenia perspektywiczne zapewniają bardziej elastyczne możliwości. Za ich pomocą można na przykład zamienić prostokąt w dowolny czworokąt. Rysunek 11.3 przedstawia różne schematyczne przykłady przekształceń afinicznych i perspektywicznych. Natomiast na rysunku 11.4 pokazane są podobne przykłady z użyciem zdjęć.



Rysunek 11.3. Przekształcenia afiniczne i perspektywiczne

<sup>9</sup> Homografia to pojęcie matematyczne oznaczające odwzorowywanie punktów z jednej powierzchni na innej. W tym sensie jest ono zatem ogólniejsze niż znaczenie, w którym go tutaj używamy. W kontekście komputerowego rozpoznawania obrazu homografia prawie zawsze oznacza odwzorowanie punktów między dwiema płaszczyznami graficznymi, które korespondują z tym samym miejscem na płaskim obiekcie w prawdziwym świecie. Odwzorowanie takie może być reprezentowane przez pojedynczą ortogonalną macierz o wymiarach 3x3 (szerzej na ten temat piszemy w rozdziale 19.).

## Przekształcenia afiniczne

Przekształcenia afiniczne stosuje się głównie w dwóch sytuacjach: gdy mamy obraz (albo obszar zainteresowania), który chcemy przekształcić, albo gdy mamy listę punktów, dla których chcemy obliczyć wynik przekształcenia. Koncepcyjnie oba te przypadki są do siebie podobne, ale różnią się pod względem praktycznej implementacji. Dlatego w bibliotece OpenCV dostępne są dwie funkcje — każda przeznaczona do użycia w innym przypadku.

### cv::warpAffine() — przekształcenia afiniczne zbiorów gęstych

W pierwszym przypadku oczywistymi formatami wejściowymi i wyjściowymi są obrazy, a mniej oczywiste jest założenie funkcji, że piksele są *gęstą reprezentacją* opisywanego obrazu. Oznacza to, że funkcja zniekształcająca musi wykonywać interpolacje, tak aby obrazy wynikowe były gładkie i wyglądały naturalnie. Funkcja przekształcenia afinicznego do przekształceń struktur gęstych w OpenCV nazywa się `cv::warpAffine()`:

```
void cv::warpAffine(
    cv::InputArray src,                // obraz wejściowy
    cv::OutputArray dst,              // obraz wynikowy
    cv::InputArray M,                 // macierz przekształcenia 2x3
    cv::Size dsize,                   // docelowy rozmiar obrazu
    int flags = cv::INTER_LINEAR,     // interpolacja, odwrotna
    int borderMode = cv::BORDER_CONSTANT, // ekstrapolacja pikseli
    const cv::Scalar& borderValue = cv::Scalar() // dla stałych krawędzi
);
```

Argumenty `src` i `dst` reprezentują odpowiednio tablicę wejściową i wynikową. Parametr `M` jest opisaną przez nas wcześniej macierzą o wymiarach  $2 \times 3$ , która określa rodzaj przekształcenia. Każdy element w tablicy docelowej jest obliczony na podstawie elementu z tablicy źródłowej pobranego z lokalizacji wyrażonej następującym wzorem:

$$dst(x, y) = src(M_{00}x + M_{01}y + M_{02}, M_{10}x + M_{11}y + M_{12})$$

Zasadniczo jednak prawa strona tego równania zwykle wskazuje lokalizację piksela, która nie jest liczbą całkowitą. W takim przypadku prawidłową wartość dla punktu  $dst(x, y)$  należy znaleźć przez zastosowanie interpolacji. Kolejny argument, `flags`, wskazuje metodę interpolacji. Dostępne są metody opisane w tabeli 11.1, czyli te same co dla funkcji `cv::resize()`, oraz dodatkowo opcja `cv::WARP_INVERSE_MAP` (którą można dodać przy użyciu operatora logicznego LUB). Opcja ta służy do wykonywania odwrotnego przekształcenia z `dst` do `src`. Dwa ostatnie argumenty dotyczą ekstrapolacji krawędzi i mają takie same znaczenie jak podobne argumenty w operacjach splotu obrazów (opisanych w rozdziale 10.).

### cv::getAffineTransform() — obliczanie macierzy przekształcenia afinicznego

W bibliotece OpenCV znajdują się dwie funkcje do generowania macierzy odwzorowania `M`. Pierwszej używa się, gdy dane są dwa obrazy, o których wiadomo, że łączy je relacja przekształcenia afinicznego, lub które chcemy w ten sposób aproksymować:

```
cv::Mat cv::getAffineTransform( // zwraca macierz o wymiarach 2x3
    const cv::Point2f* src,      // współrzędne *trzech* wierzchołków
    const cv::Point2f* dst       // współrzędne docelowe, trzy wierzchołki
);
```

W tej funkcji parametry `src` i `dst` są tablicami zawierającymi po trzy dwuwymiarowe  $(x, y)$  punkty. Wartością zwrótną jest tablica reprezentująca przekształcenie afiniczne obliczone na podstawie tych punktów.

Zasadniczo tablice punktów `src` i `dst` w funkcji `cv::getAffineTransform()` definiują dwa równoległoki. Punkty w tablicy `src` zostaną odwzorowane przez funkcję `cv::warpAffine()`, przy użyciu otrzymanej macierzy  $M$ , na odpowiednie punkty w `dst`. Wszystkie pozostałe punkty zostaną pociągnięte za nimi. Po odwzorowaniu tych trzech niezależnych rogów odwzorowanie pozostałych punktów także będzie w pełni zdeterminowane.

Na listingu 11.1 przedstawiony jest kod źródłowy ilustrujący użycie tych funkcji w praktyce. W programie tym najpierw obliczamy parametry macierzowe funkcji `cv::warpAffine()`, tworząc dwie tablice po trzy elementy (reprezentujące rogi równoległoków), a potem konwertując je na prawdziwą macierz przekształcenia za pomocą funkcji `cv::getAffineTransform()`. Następnie wykonujemy przekształcenie afiniczne i rotację obrazu. Dla naszej tablicy reprezentatywnych punktów w obrazie źródłowym, `srcTri []`, pobieramy trzy punkty:  $(0,0)$ ,  $(0, \text{wysokość}-1)$  oraz  $(\text{szerokość}-1,0)$ . Później określamy lokalizacje, w których te punkty zostaną odwzorowane w korespondującej tablicy `dstTri []`.

#### Listing 11.1. Przekształcenie afiniczne

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace std;

int main(int argc, char** argv) {

    if(argc != 2) {
        cout << "Przekształcenie afiniczne\nSposób użycia: " << argv[0] << " <imagedata>\n" << endl;
        return -1;
    }

    cv::Mat src = cv::imread(argv[1],1);
    if( src.empty() ) { cout << "Nie udało się załadować " << argv[1] << endl; return -1; }

    cv::Point3f srcTri[] = {
        cv::Point2f(0,0),           //źródło, lewy górny
        cv::Point2f(src.cols-1, 0), //źródło, prawy górny
        cv::Point2f(0, src.rows-1) //źródło, lewy dolny
    };

    cv::Point2f dstTri[] = {
        cv::Point2f(src.cols*0.f, src.rows*0.33f), //cel, lewy górny
        cv::Point2f(src.cols*0.85f, src.rows*0.25f), //cel, prawy górny
        cv::Point2f(src.cols*0.15f, src.rows*0.7f) //cel, lewy dolny
    };

    // OBLICZANIE MACIERZY PRZEKSZTAŁCENIA AFINICZNEGO
    //
    cv::Mat warp_mat = cv::getAffineTransform(srcTri, dstTri);
    cv::Mat dst, dst2;
    cv::warpAffine(
        src,
        dst,
        warp_mat,
        src.size(),
        cv::INTER_LINEAR,
```



```

    cv::BORDER_CONSTANT,
    cv::Scalar(0)
);
for( int i = 0; i < 3; ++i )
    cv::circle(dst, dstTri[i], 5, cv::Scalar(255, 0, 255), -1, cv::AA);

cv::imshow("Test przekształcenia afinicznego", dst);
cv::waitKey();

for(int frame=0; ++frame) {

    // OBLICZANIE MACIERZY ROTACJI
    cv::Point2f center(src.cols*0.5f, src.rows*0.5f);
    double angle = frame*3 % 360, scale = (cos((angle - 60)* cv::PI/180) + 1.05)*0.8;

    cv::Mat rot_mat = cv::getRotationMatrix2D(center, angle, scale);
    cv::warpAffine(
        src,
        dst,
        rot_mat,
        src.size(),
        cv::INTER_LINEAR,
        cv::BORDER_CONSTANT,
        cv::Scalar(0)
    );
    cv::imshow("Rotated Image", dst);
    if(cv::waitKey(30) >= 0 )
        break;

}

return 0;
}

```

Drugim sposobem obliczania macierzy odwzorowania  $M$  jest użycie funkcji `cv::getRotationMatrix2D()`, która oblicza macierz odwzorowania dla rotacji wokół dowolnego punktu w połączeniu z opcjonalnym skalowaniem. Jest to tylko jeden z możliwych rodzajów przekształcenia afinicznego, więc technika ta jest mniej ogólna niż metoda z użyciem funkcji `cv::getAffineTransform()`, ale stanowi ważny podzbiór, który ma alternatywną (i bardziej intuicyjną) reprezentację:

```

cv::Mat cv::getRotationMatrix2D( // zwraca macierz o wymiarach 2x3
    cv::Point2f center           // środek rotacji
    double angle,                // kąt obrotu
    double scale                 // skalowanie po rotacji
);

```

Pierwszy argument, `center`, określa punkt środkowy rotacji. Dwa następne argumenty określają kąt obrotu i skalowanie. Funkcja ta zwraca macierz odwzorowania  $M$ , która (jak zawsze) ma wymiary  $2 \times 3$  i jest typu zmiennoprzecinkowego.

Jeśli przyjmiemy definicje  $\alpha = scale \cdot \cos(angle)$  i  $\beta = scale \cdot \sin(angle)$ , to macierz  $M$  obliczona przez niniejszą funkcję będzie miała następującą postać:

$$\begin{bmatrix} \alpha\beta(1-\alpha) \cdot center_x - \beta \cdot center_y \\ -\beta\alpha\beta \cdot center_x - (1-\alpha) \cdot center_y \end{bmatrix}$$

Łącząc obie metody obliczania macierzy odwzorowania, można na przykład wybrany obraz obrócić, przeskalować i przekształcić.

## cv::transform() — przekształcenia afiniczne zbiorów rzadkich

Wiesz już, że do pracy ze strukturami gęstymi służy funkcja `cv::warpAffine()`. Natomiast do odwzorowań rzadkich (tzn. list indywidualnych punktów) najlepiej nadaje się funkcja `cv::transform()`. Może pamiętasz z rozdziału 5., że metoda ta ma następujący prototyp:

```
void cv::transform(  
    cv::InputArray src, // wejściowa tablica o wymiarach  $N \times 1$  ( $D_s$  kanałów)  
    cv::OutputArray dst, // wyjściowa tablica o wymiarach  $N \times 1$  ( $D_d$  kanałów)  
    cv::InputArray mtx // macierz przekształcenia ( $D_s \times D_d$ )  
);
```

Generalnie `src` jest tablicą o wymiarach  $N \times 1$  zawierającą  $D_s$  kanałów, gdzie  $N$  jest liczbą punktów do przekształcenia, a  $D_s$  jest wymiarem tych punktów źródłowych. Wyjściowa tablica `dst` będzie miała taki sam rozmiar, ale może mieć inną liczbę kanałów  $D_d$ . Macierz przekształcenia `mtx` ma wymiary  $D_s \times D_d$  i jest stosowana do każdego elementu tablicy `src`, po czym wyniki zostają wstawione do tablicy `dst`.



Należy zauważyć, że funkcja `cv::transform()` działa na kanałowych indeksach każdego punktu w tablicy. W odniesieniu do aktualnego problemu zakładamy, że tablica ta jest w istocie dużym wektorem ( $N \times 1$  lub  $1 \times N$ ) takich wielokanałowych obiektów. Powinniśmy pamiętać, że indeks, do którego odnosi się macierz przekształcenia, jest indeksem kanału, a nie „wektorowym” indeksem tej dużej tablicy.

W przypadku przekształceń, które są prostymi rotacjami w danej przestrzeni kanału, nasza macierz przekształcenia `mtx` będzie tylko macierzą o wymiarach  $2 \times 2$  i można ją zastosować bezpośrednio do dwukanałowych indeksów tablicy `src`. W niektórych prostych przypadkach dotyczy to rotacji, rozciągania i zniekształcania. Zazwyczaj jednak do przeprowadzenia ogólnego przekształcenia afinicznego (w tym translacji i rotacji wokół dowolnego środka itd.) konieczne jest zwiększenie liczby kanałów w `src` do trzech, tak aby otrzymać bardziej typową macierz przekształcenia afinicznego o wymiarach  $2 \times 3$ . W tym przypadku wszystkie elementy w trzecim kanale muszą zostać ustawione na 1 (tzn. punkty muszą być podane we współrzędnych homogenicznych). Oczywiście tablica wyjściowa nadal będzie miała dwa kanały.

## cv::invertAffineTransform() — odwrotne przekształcenie afiniczne

Mając dane przekształcenie afiniczne reprezentowane przez macierz o wymiarach  $2 \times 3$ , często trzeba wykonać operację odwrotną, w celu przywrócenia wszystkich przekształconych punktów na ich pierwotne miejsca. Można to zrobić za pomocą funkcji `cv::invertAffineTransform()`:

```
void cv::invertAffineTransform(  
    cv::InputArray M, // wejściowa macierz o wymiarach  $2 \times 3$   
    cv::OutputArray iM // wyjściowa macierz także o wymiarach  $2 \times 3$   
);
```

Funkcja ta pobiera macierz `M` o wymiarach  $2 \times 3$  i zwraca inną macierz `iM` o wymiarach  $2 \times 3$ , która jest odwrotnością `M`. Funkcja `cv::invertAffineTransform()` tak naprawdę nie działa na żadnym obrazie, lecz jedynie dostarcza przekształcenie odwrotne. Mając macierz `iM`, można jej używać w taki sam sposób, jak używałoby się `M` w funkcjach `cv::warpAffine()` i `cv::transform()`.

## Przekształcenie perspektywiczne

Aby skorzystać z większej elastyczności przekształceń perspektywicznych (zwanymi też homografiami), potrzebujemy nowej funkcji, która pozwoli nam wyrazić tę szerszą klasę transformacji. Przede wszystkim musimy podkreślić, że choć rzutowanie perspektywiczne w całości definiuje pojedyncza macierz, nie jest to rodzaj przekształcenia liniowego. Wynika to z faktu, że ten rodzaj transformacji wymaga podzielenia przez ostatni wymiar (zazwyczaj  $Z$  — zobacz rozdział 19.) i w związku z tym traci po drodze jeden wymiar.

Podobnie jak w przypadku przekształceń afinicznych, operacje na obrazach (przekształcenia gęste) wykonuje się za pomocą innych funkcji niż operacje na zbiorach punktów (przekształcenia rzadkie).

### `cv::warpPerspective()` — gęste przekształcenie perspektywiczne

Gęste przekształcenie perspektywiczne w OpenCV wykonuje się za pomocą funkcji analogicznej do funkcji gęstego przekształcenia afinicznego. Mówiąc konkretnie: prawie wszystkie argumenty funkcji `cv::warpPerspective()` pokrywają się z argumentami funkcji `cv::warpAffine()`. Drobnym, ale jakże znaczącym wyjątkiem jest to, że macierz odwzorowania musi mieć wymiary  $3 \times 3$ .

```
void cv::warpPerspective(
    cv::InputArray src,                // obraz wejściowy
    cv::OutputArray dst,              // obraz wynikowy
    cv::InputArray M,                 // macierz przekształcenia 3x3
    cv::Size dsize,                   // docelowy rozmiar obrazu
    int flags = cv::INTER_LINEAR,     // interpolacja, odwrotna
    int borderMode = cv::BORDER_CONSTANT, // metoda ekstrapolacji
    const cv::Scalar& borderValue = cv::Scalar() // dla stałych krawędzi
);
```

Każdy element w tablicy docelowej jest obliczany na podstawie elementu z tablicy źródłowej o lokalizacji wyrażonej za pomocą następującego wzoru:

$$dst(x, y) = src \left( \frac{M_{00}x + M_{01}y + M_{02}}{M_{20}x + M_{21}y + M_{22}}, \frac{M_{10}x + M_{11}y + M_{12}}{M_{20}x + M_{21}y + M_{22}} \right)$$

Tak jak w przypadku przekształcenia afinicznego, lokalizacja wskazywana przez prawą stronę tego równania z reguły nie jest całkowitoliczbowa. Argument `flags` określa metodę interpolacji i może przyjmować takie same wartości jak analogiczny argument funkcji `cv::warpAffine()`.

### `cv::getPerspectiveTransform()` — obliczanie macierzy rzutowania perspektywicznego

Tak jak w przypadku przekształcenia afinicznego, dla którego obliczaliśmy macierz odwzorowania w poprzednim przykładzie kodu, macierz przekształcenia możemy obliczyć przy użyciu specjalnej funkcji pobierającej listę punktów:

```
cv::Mat cv::getPerspectiveTransform( // zwraca macierz o wymiarach 3x3
    const cv::Point2f* src,          // współrzędne *czterech* wierzchołków
    const cv::Point2f* dst           // współrzędne docelowe, cztery wierzchołki
);
```

Teraz argumenty `src` i `dst` reprezentują tablice zawierające po cztery (nie trzy) punkty, dzięki czemu możemy niezależnie kontrolować, jak rogi (zazwyczaj) prostokąta z `src` mają być odwzorowane w (zazwyczaj) romb w `dst`. Nasze przekształcenie jest całkowicie zdefiniowane przez wyznaczone miejsca docelowe czterech punktów źródłowych. Jak wspominaliśmy wcześniej, w przypadku przekształceń perspektywicznych wartością zwrótną będzie tablica o wymiarach  $3 \times 3$  (przykładowy kod przedstawiliśmy na listingu 11.2). Nie licząc tego, że macierz ma wymiary  $3 \times 3$ , i zamiany trzech punktów kontrolnych na cztery, przekształcenie perspektywiczne jest analogiczne do opisanego wcześniej przekształcenia afinicznego.

### Listing 11.2. Przykład przekształcenia perspektywicznego

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace std;

int main(int argc, char** argv) {

    if(argc != 2) {
        cout << "Zniekształcenie perspektywiczne\nSposób użycia: " << argv[0] << " <imagedata>\n" << endl;
        return -1;
    }

    Mat src = cv::imread(argv[1],1);
    if( src.empty() ) { cout << "Nie udało się załadować " << argv[1] << endl; return -1; }

    cv::Point2f srcQuad[] = {
        cv::Point2f(0, 0), // źródło, lewy górny
        cv::Point2f(src.cols-1, 0), // źródło, prawy górny
        cv::Point2f(src.cols-1, src.rows-1), // źródło, prawy dolny
        cv::Point2f(0, src.rows-1) // źródło, lewy dolny
    };

    cv::Point2f dstQuad[] = {
        cv::Point2f(src.cols*0.05f, src.rows*0.33f),
        cv::Point2f(src.cols*0.9f, src.rows*0.25f),
        cv::Point2f(src.cols*0.8f, src.rows*0.9f),
        cv::Point2f(src.cols*0.2f, src.rows*0.7f)
    };

    // OBLICZANIE MACIERZY PERSPEKTYWY
    //
    cv::Mat warp_mat = cv::getPerspectiveTransform(srcQuad, dstQuad);
    cv::Mat dst;
    cv::warpPerspective(src, dst, warp_mat, src.size(), cv::INTER_LINEAR,
        cv::BORDER_CONSTANT, cv::Scalar());

    for( int i = 0; i < 4; i++ )
        cv::circle(dst, dstQuad[i], 5, cv::Scalar(255, 0, 255), -1, cv::AA);

    cv::imshow("Test przekształcenia perspektywicznego", dst);
    cv::waitKey();
    return 0;
}
```

## cv::perspectiveTransform() — rzadkie przekształcenia perspektywiczne

Funkcja `cv::perspectiveTransform()` to specjalna procedura wykonująca przekształcenia perspektywiczne na listach punktów. Funkcja `cv::transform()` jest ograniczona do operacji liniowych, przez co nie jest w stanie prawidłowo wykonać przekształcenia perspektywicznego. Wynika to z faktu, że tego typu transformacje wymagają dzielenia przez trzecią współrzędną reprezentacji homogenicznej ( $x = f \cdot X/Z$ ,  $y = f \cdot Y/Z$ ). Zawsze jednak możemy skorzystać ze specjalnej funkcji `cv::perspectiveTransform()`:

```
void cv::perspectiveTransform(  
    cv::InputArray src, // wejściowa tablica o wymiarach N×1 (2 lub 3 kanały)  
    cv::OutputArray dst, // wyjściowa tablica o wymiarach N×1 (2 lub 3 kanały)  
    cv::InputArray mtx // macierz przekształcenia (3×3 lub 4×4)  
);
```

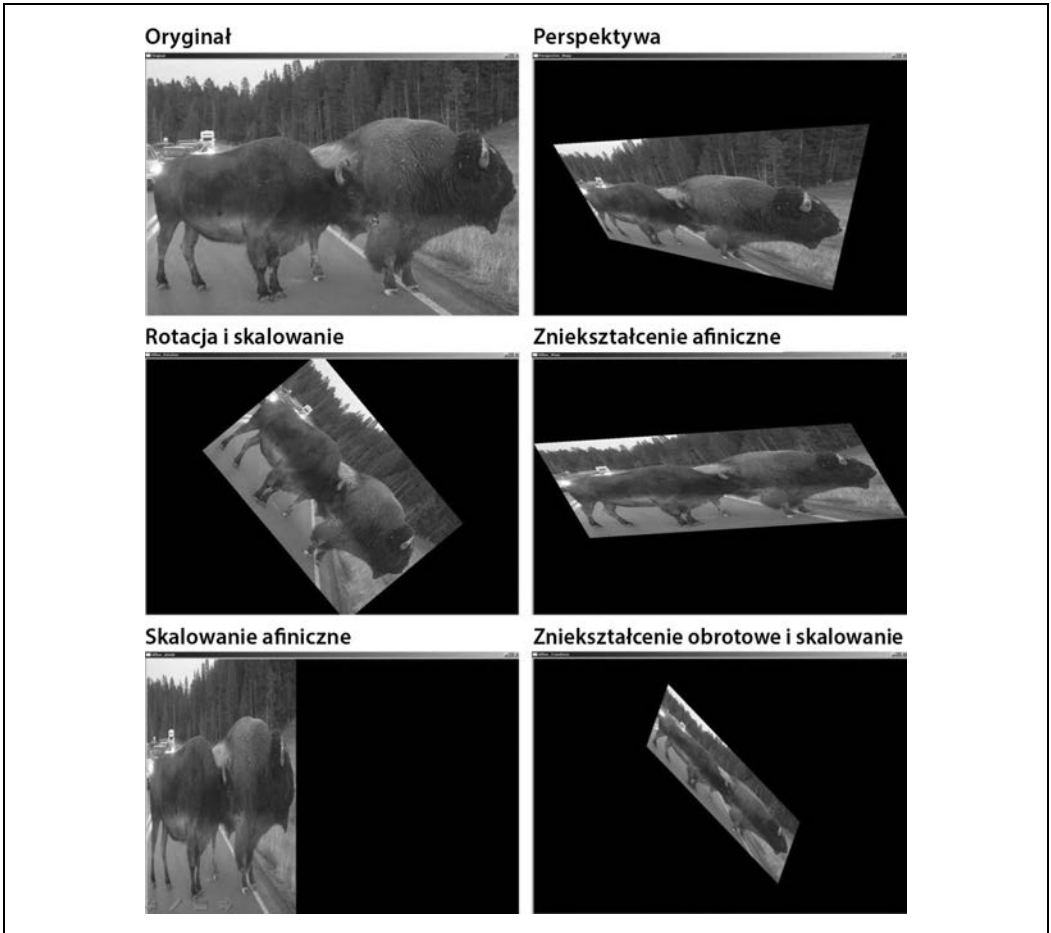
Jak zwykle argumenty `src` i `dst` reprezentują odpowiednio tablicę punktów źródłowych do przekształcenia i tablicę punktów powstałych w wyniku tej operacji. Tablice te powinny zawierać po dwa lub trzy kanały. Macierz `mtx` może mieć wymiary 3×3 lub 4×4. Jeśli jest to macierz o wymiarach 3×3, to funkcja wykonuje rzutowanie z dwóch wymiarów na dwa. Jeżeli macierz ta ma wymiary 4×4, to zostaje wykonane rzutowanie z trzech wymiarów na trzy.

W aktualnym kontekście przekształcamy zbiór punktów obrazu na inny zbiór punktów obrazu, co wygląda na odwzorowywanie z dwóch wymiarów na dwa wymiary. Nie jest to jednak do końca prawdą, ponieważ przekształcenie perspektywiczne w istocie polega na odwzorowywaniu punktów z dwuwymiarowej płaszczyzny osadzonej w przestrzeni trójwymiarowej na (inną) podpłaszczyznę dwuwymiarową. Można powiedzieć, że jest to taka sama czynność, jaką wykonuje kamera. (Bardziej szczegółowo omawiamy ten temat w dalszych rozdziałach, przy okazji omawiania kamery). Kamera pobiera punkty w trzech wymiarach i odwzorowuje je na dwa wymiary przetwornika kamery. O to zasadniczo chodzi w przypadku, gdy punkty źródłowe są podawane we współrzędnych homogenicznych. Dodajemy wymiar do tych punktów, wprowadzając wymiar  $Z$  i ustawiając wszystkie wartości w tym wymiarze na 1. Przekształcenie rzutowania jest wówczas rzutowaniem z powrotem z tej przestrzeni do dwuwymiarowej przestrzeni na wyjściu. W ten pokrętny sposób wyjaśniamy, dlaczego przy odwzorowywaniu punktów z jednego obrazu do punktów w innym obrazie potrzebna jest macierz o wymiarach 3×3.

Rysunek 11.4 pokazuje wyniki przekształceń afinicznych i perspektywicznych przedstawionych na listingach 11.1 i 11.2. W przykładach tych transformacji podaliśmy prawdziwe obrazy. Można je porównać z prostymi schematami pokazanymi na rysunku 11.3.

## Ogólne odwzorowania

Omawiane do tej pory przekształcenia afiniczne i perspektywiczne to specjalne przypadki ogólniejszego procesu. Zasadniczo obie te transformacje bazują na tej samej zasadzie: pobierają piksele z jednego miejsca w obrazie źródłowym i odwzorowują je w innym miejscu w obrazie docelowym. Istnieją też inne operacje o takiej samej strukturze. W tej sekcji opisujemy kilka kolejnych przekształceń tego typu, a następnie przedstawiamy dostępne w OpenCV narzędzia do implementacji własnych ogólnych przekształceń opartych na odwzorowaniach.



Rysunek 11.4. Odwzorowania perspektywiczne i afiniczne obrazu

## Odwzorowania biegunowe

W rozdziale 5. krótko nadmieniliśmy o dwóch funkcjach, `cv::cartToPolar()` i `cv::polarToCart()`, za pomocą których można konwertować tablice punktów kartezjańskich  $x, y$  na tablice punktów biegunowych  $r-\theta$  (lub odwrotnie).



Między funkcjami odwzorowania biegunowego oraz funkcjami przekształceń perspektywicznych i afinicznych występuje drobna niespójność stylistyczna. Funkcje odwzorowania biegunowego do reprezentacji wektorów dwuwymiarowych wymagają podania par tablic jednokanałowych, a nie dwukanałowych. Źródłem tej różnicy jest sposób, w jaki te różne funkcje są tradycyjnie używane, i nie ma to związku z żadnymi wewnętrznymi mechanizmami ich działania.

Choć funkcje `cv::cartToPolar()` i `cv::polarToCart()` są wykorzystywane przez bardziej skomplikowane procedury (np. opisana dalej `cv::logPolar()`), mogą być też przydatne same w sobie.

## cv::cartToPolar() — konwersja współrzędnych kartezjańskich na biegunowe

Do zamiany współrzędnych kartezjańskich na biegunowe służy funkcja `cv::cartToPolar()`:

```
void cv::cartToPolar(
    cv::InputArray x,           // wejściowa jednocanalowa tablica współrzędnych x
    cv::InputArray y,           // wejściowa jednocanalowa tablica współrzędnych y
    cv::OutputArray magnitude, // wyjściowa jednocanalowa tablica promieni wodzących
    cv::OutputArray angle,     // wyjściowa jednocanalowa tablica kątów
    bool angleInDegrees = false // true oznacza stopnie, false — radiany
);
```

Dwa pierwsze argumenty,  $x$  i  $y$ , są tablicami jednocanalowymi. Jednak nie reprezentują one tylko list punktów, lecz **pole wektorowe**<sup>10</sup>, w którym składnik  $x$  danego punktu reprezentuje odpowiednia wartość z tablicy  $x$ , a składnik  $y$  — odpowiednia wartość z tablicy  $y$ . Natomiast wyniki tej funkcji znajdują się w tablicach `magnitude` i `angle`. Każdy punkt w `magnitude` reprezentuje długość wektora w tym punkcie z tablic  $x$  i  $y$ , a każdy punkt w `angle` reprezentuje kąt skierowany tego wektora. Kąty zapisane w `angles` domyślnie są podawane w radianach, to znaczy w przedziale  $[0, 2\pi)$ . Jeżeli jednak argumentowi `angleInDegrees` zostanie przypisana wartość `true`, to kąty w `angle` będą zapisywane w stopniach w przedziale  $[0, 360)$ . Ponadto należy zauważyć, że kąty są obliczane (w przybliżeniu) według wzoru  $\text{atan2}(y, x)$ , w związku z czym kąt 0 odpowiada wektorowi skierowanemu w kierunku  $\hat{x}$ .

Jeśli zastanawiasz się, gdzie można by zastosować tę funkcję, wyobraź sobie, że obliczyłeś już pochodne  $x$  i  $y$  obrazu za pomocą funkcji `cv::Sobel()` albo przy użyciu funkcji splotu za pośrednictwem funkcji `cv::DFT()` lub `cv::filter2D()`. Jeżeli pochodne  $x$  zostały zapisane w obrazie `dx_img`, a pochodne  $y$  — w obrazie `dy_img`, to można sporządzić histogram wykrywania kątów krawędzi, to znaczy można zebrać wszystkie kąty, dla których moduł lub moc piksela na krawędzi przekracza pewien określony próg. Aby wykonać takie obliczenia, najpierw trzeba utworzyć dwa nowe obrazy docelowe (i na przykład nadać im nazwy `img_mag` i `img_angle`) na pochodne kierunkowe, a następnie posłużyć się funkcją `cvCartToPolar(dx_img, dy_img, img_mag, img_angle, 1)`. Później mogliśmy sporządzić histogram na podstawie danych z `img_angle`, które pobieralibyśmy pod warunkiem, że odpowiedni „piksel” w `img_mag` jest powyżej ustalonego przez nas prog.



Funkcje rozpoznawania obrazu i graficzne omawiamy w rozdziale 22. Opisany tu proces stanowi podstawę obliczania ważnej składowej rozpoznawania obiektów — **HOG** (ang. *histogram of oriented gradients*).

## cv::polarToCart() — konwersja współrzędnych biegunowych na kartezjańskie

Funkcja `cv::cartToPolar()` zamienia współrzędne biegunowe na kartezjańskie.

```
void cv::polarToCart(
    cv::InputArray magnitude, // wyjściowa jednocanalowa tablica promieni wodzących
    cv::InputArray angle,     // wyjściowa jednocanalowa tablica kątów
    cv::OutputArray x,        // wejściowa jednocanalowa tablica x
    cv::OutputArray y,        // wejściowa jednocanalowa tablica y
    bool angleInDegrees = false // true oznacza stopnie, false — radiany
);
```

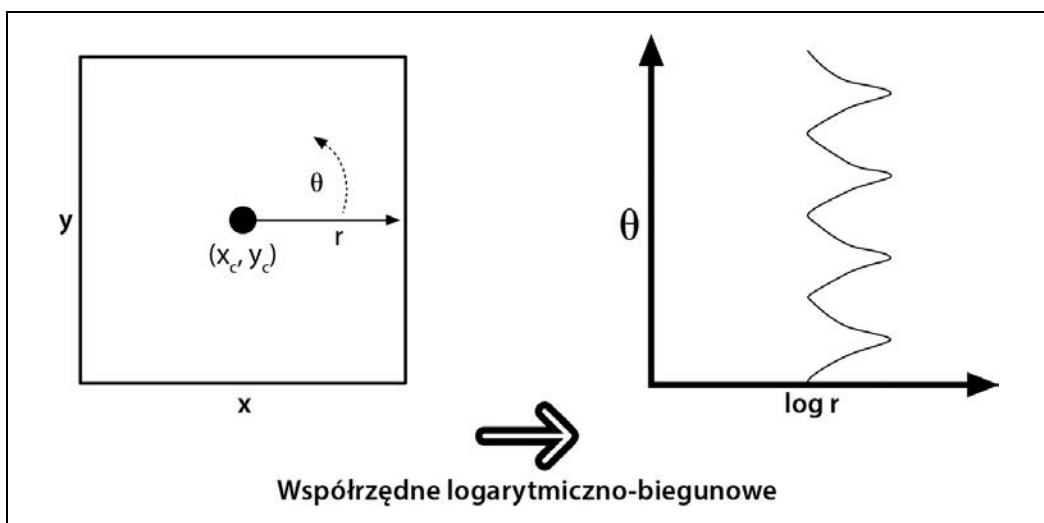
<sup>10</sup> Jeśli ktoś nie jest obeznany z pojęciem pola wektorowego, to może je sobie wyobrazić jako dwuskładnikowy wektor powiązany z każdym punktem „obrazu”.

Operacja zamiany współrzędnych biegunowych na kartezjańskie również jest często przydatna. Przyjmuje praktycznie takie same argumenty jak `cv::cartToPolar()`, tylko że tym razem tablice odległości od bieguna i kątów są danymi wejściowymi, a  $x$  i  $y$  — wyjściowymi.

## Współrzędne logarytmiczno-biegunowe

W odniesieniu do obrazów dwuwymiarowych przekształcenie logarytmiczno-biegunowe [Schwartz80] polega na zamianie współrzędnych kartezjańskich na logarytmiczno-biegunowe:  $(x, y) \leftrightarrow re^{i\theta}$ , gdzie  $r = \sqrt{x^2 + y^2}$ , a  $\theta = \text{atan2}(y, x)$ . Aby następnie wyodrębnić współrzędne biegunowe do przestrzeni  $(\rho, \theta)$  względnej w odniesieniu do pewnego punktu centralnego  $(x_c, y_c)$ , należy obliczyć logarytm w taki sposób, że  $\rho = \log\left(\sqrt{(x - x_c)^2 + (y - y_c)^2}\right)$  oraz  $\theta = \text{atan2}(y - y_c, x - x_c)$ . Na

potrzeby obróbki obrazów — gdy trzeba „upchnąć” interesujące nas dane w dostępnej pamięci graficznej — do  $\rho$  zazwyczaj stosuje się współczynnik skalowania  $m$ . Po lewej na rysunku 11.5 widać kwadratowy obiekt, a po prawej — jego wersję zakodowaną w przestrzeni logarytmiczno-biegunowej.

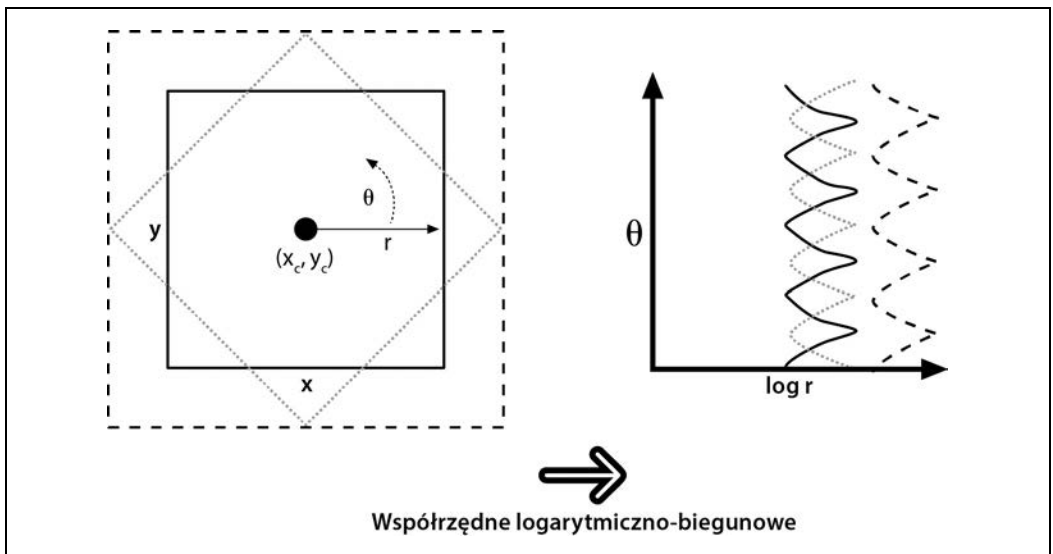


Rysunek 11.5. Przekształcenie logarytmiczno-biegunowe odwzorowuje współrzędne  $(x, y)$  na  $(\log(r), \theta)$ . W tym przykładzie pokazaliśmy kwadrat przedstawiony w układzie współrzędnych logarytmiczno-biegunowych

Można się zastanawiać, po co w ogóle coś takiego robić. Konwersja na współrzędne logarytmiczno-biegunowe ma jednak korzenie w obserwacji ludzkiego wzroku. W oku znajduje się niewielki, ale bardzo gęsty obszar fotoreceptorów (**dołek środkowy siatkówki oka**), poza którym gęstość receptorów raptownie spada (w tempie wykładniczym). Spójrz na dowolny punkt na ścianie i w polu widzenia w odległości na wyciągnięcie ręki ustaw palec. Następnie cały czas patrząc w ten wybrany punkt na ścianie, powoli przesuwaj palec w bok. Zwróć uwagę, jak szybko obraz palca staje się nieostry podczas przesuwania go sprzed zasięgu dołka środkowego siatkówki. Ta anatomiczna struktura ma też pewne interesujące właściwości matematyczne (ich opis wykracza poza zakres tematyczny tej książki) dotyczące zachowywania kątów przecięcia linii.



Dla nas najważniejsze jest to, że przy użyciu przekształcenia na współrzędne logarytmiczno-biegunowe można stworzyć dwuwymiarowe niezmiennie reprezentacje widoków obiektu przez przesunięcie środka ciężkości przekształconego obrazu do ustalonego punktu na płaszczyźnie logarytmiczno-biegunowej. Na rysunku 11.6 po lewej pokazane są trzy figury, które chcemy rozpoznać jako kwadraty. Trudność polega na tym, że każda z nich wygląda całkiem inaczej od pozostałych. Jedna jest znacznie większa od pozostałych dwóch, a inna jest przekręcona. Po prawej stronie ilustracji widać wynik zamiany na współrzędne logarytmiczno-biegunowe. Zwróć uwagę, że różnice w rozmiarze na płaszczyźnie  $(x,y)$  zostały zamienione na przesunięcia względem osi  $\log(r)$  na płaszczyźnie logarytmiczno-biegunowej, a różnice w rotacji skutkują przesunięciami względem osi  $\theta$ . Jeśli weźmiemy przekształcony środek każdego kwadratu na płaszczyźnie logarytmiczno-biegunowej i przenieśmy go do jakiegoś ustalonego punktu, to wszystkie kwadraty na tej płaszczyźnie okażą się identyczne. Oznacza to rodzaj niezmienności w odniesieniu do rotacji i skalowania w dwóch wymiarach<sup>11</sup>.



Rysunek 11.6. Logarytmiczno-biegunowe przekształcenie obróconych i przeskalowanych kwadratów. Rozmiar zamienia się na przesunięcie względem osi  $\log(r)$ , a rotacja — na przesunięcie względem osi  $\theta$

<sup>11</sup> W rozdziale 22. szczegółowo opisujemy techniki rozpoznawania. Na razie wystarczy zauważyć, że wykonywanie transformacji na współrzędne logarytmiczno-biegunowe całego obiektu nie byłoby dobrym pomysłem, ponieważ przekształcenia takie są wrażliwe na dokładne położenie punktów centralnych. W przypadku rozpoznawania obiektów prawdopodobnie lepszym rozwiązaniem byłoby wykrycie zbiorów kluczowych punktów (np. rogów albo zarysów), przycięcie tych widoków, a następnie wykorzystanie środków tych punktów kluczowych jako środków logarytmiczno-biegunowych. Przy użyciu tych lokalnych przekształceń logarytmiczno-biegunowych można następnie stworzyć lokalne właściwości, które są (częściowo) niezmiennie w odniesieniu do rotacji i skalowania i które można powiązać z obiektem wizualnym.

## cv::logPolar()

Funkcja OpenCV do wykonywania przekształceń logarytmiczno-biegunowych to cv::logPolar():

```
void cv::logPolar(
    cv::InputArray src,           // obraz wejściowy
    cv::OutputArray dst,        // obraz wynikowy
    cv::Point2f center,         // środek przekształcenia
    double m,                   // współczynnik skalowania
    int flags = cv::INTER_LINEAR // tryby interpolacji i wypełniania
    | cv::WARP_FILL_OUTLIERS
);
```

Argumenty src i dst to typowe obrazy wejściowy i wyjściowy. Parametr center określa punkt centralny ( $x_c, y_c$ ) przekształcenia logarytmiczno-biegunowego. Parametr m reprezentuje współczynnik skalowania, który należy zdefiniować, aby interesujące nas właściwości zdominowały dostępny obszar obrazu. Parametr flags służy do wybierania metody interpolacji. Do wyboru są standardowe metody interpolacji dostępne w bibliotece OpenCV (tabela 11.1). Różne metody można łączyć z jednym lub dwoma następującymi znacznikami: CV::WARP\_FILL\_OUTLIERS (wypełnienie punktów, które w przeciwnym razie byłyby niezdefiniowane) oraz CV::WARP\_INVERSE\_MAP (obliczanie odwrotnego odwzorowania ze współrzędnych logarytmiczno-biegunowych na kartezjańskie).

Na listingu 11.3 przedstawiony jest przykładowy program z użyciem współrzędnych logarytmiczno-biegunowych, w którym pokazujemy transformacje w obie strony. Rysunek 11.7 przedstawia efekty zastosowania tych operacji do zdjęcia.



Rysunek 11.7. Przykład zastosowania przekształcenia na zdjęciu losia, ze środkiem transformacji w białym punkcie po lewej stronie; po prawej widać efekt przekształcenia

Listing 11.3. Przykład przekształcenia logarytmiczno-biegunowego

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace std;
```

```

int main(int argc, char** argv) {

    if(argc != 3) {
        cout << "LogPolar\nSposób użycia: " <<argv[0] <<" <imagename> <M value>\n" <<"<M value>-30
        ↪zazwyczaj wystarczy\n";
        return -1;
    }

    cv::Mat src = cv::imread(argv[1],1);

    if( src.empty() ) { cout << "Nie udało się załadować " << argv[1] << endl; return -1; }

    double M = atof(argv[2]);
    cv::Mat dst(src.size(), src.type(), src2(src.size(), src.type()));

    cv::logPolar(
        src,
        dst,
        cv::Point2f(src.cols*0.5f, src.rows*0.5f),
        M,
        cv::INTER_LINEAR | cv::WARP_FILL_OUTLIERS
    );
    cv::logPolar(
        dst,
        src2,
        cv::Point2f(src.cols*0.5f, src.rows*0.5f),
        M,
        cv::INTER_LINEAR | cv::WARP_INVERSE_MAP
    );
    cv::imshow( "log-polar", dst );
    cv::imshow( "inverse log-polar", src2 );

    cv::waitKey();

    return 0;
}

```

## Odwzorowania arbitralne

Czasami interpolację wykonujemy programowo, to znaczy mamy pewien algorytm odwzorowania i go po prostu stosujemy. Zdarzają się jednak przypadki, gdy wolimy odwzorowanie przeprowadzić samodzielnie. Zanim zaczniemy zgłębiać tajniki metod obliczania (i stosowania) takich odwzorowań, najpierw przyjrzymy się funkcji stosowania odwzorowań, na której bazują wszystkie te pozostałe metody.

Jednym z typowych zastosowań funkcji `cv::remap()` jest rektyfikacja (korekta zniekształceń) obrazów skalibrowanych i stereo. W rozdziałach 18. i 19. opisujemy funkcje konwertujące obliczone zniekształcenia spowodowane przez kamerę na parametry mapx i mapy.

Krótko mówiąc: funkcja OpenCV, której nam teraz trzeba, to `cv::remap()`.

### `cv::remap()` — ogólne odwzorowanie obrazu

```

void cv::remap(
    cv::InputArray src,                // obraz wejściowy
    cv::OutputArray dst,              // obraz wynikowy
    cv::InputArray map1,              // docelowy x dla piksela źródłowego
    cv::InputArray map2,              // docelowy y dla piksela źródłowego

```

```

int interpolation = cv::INTER_LINEAR,           // interpolacja, odwrotność
int borderMode = cv::BORDER_CONSTANT,        // metoda ekstrapolacji
const cv::Scalar& borderValue = cv::Scalar() // dla stałych krawędzi
);

```

Dwa pierwsze argumenty tej funkcji to odpowiednio obraz wejściowy i wynikowy. Następne dwa argumenty, `map1` i `map2`, wskazują docelowe wartości współrzędnych  $x$  i  $y$  wyznaczających lokalizację danego piksela. Przy ich użyciu definiuje się własne ogólne odwzorowanie. Tablice te powinny mieć taki sam rozmiar jak tablica źródłowa i wynikowa oraz muszą być typu `CV_16C2`, `CV_32C1` lub `CV_32C2`. Odwzorowania niecałkowitoliczbowe również są dozwolone. W takim przypadku funkcja `cv::remap()` automatycznie wykonuje obliczenia interpolacyjne.

Następny argument to `interpolation`. Jego wartość wskazuje funkcji `cv::remap()`, jak dokładnie ma przeprowadzić interpolację. Dostępne są wszystkie metody wymienione w tabeli 11.1 z wyjątkiem `cv::INTER_AREA`, która nie została zaimplementowana dla funkcji `cv::remap()`.

## Renowacja obrazów

Wiele obrazów zawiera różne usterki. Podczas wykonywania zdjęć obiektyw może być zabrudzony albo zachlapany wodą, a stare zdjęcia bywają podrapane lub w inny sposób uszkodzone. Ratunkiem w takich sytuacjach może być zastosowanie techniki rekonstrukcji (ang. *inpainting*) [Telea04], czyli metody retuszu polegającej na pobieraniu informacji o kolorze i teksturze na krawędziach zniszczonego obszaru, a następnie powielaniu i mieszanii takich samych pikseli w tym obszarze. Na rysunku 11.8 pokazany jest przykład, w którym technika ta została zastosowana do pozbycia się napisu ze zdjęcia.



Rysunek 11.8. Po lewej: obraz zniszczony przez napis; po prawej: obraz wyretuszowany za pomocą techniki *inpaintingu*

## Inpainting

Metoda inpaintingu sprawdza się tylko wówczas, gdy zniszczony obszar nie jest zbyt gruby, to znaczy na jego obwodzie pozostało wystarczająco informacji na temat tekstury i koloru, aby można było dokonać uzupełnienia brakującej części. Rysunek 11.9 pokazuje, co się dzieje, gdy uszkodzony obszar jest za duży.



Rysunek 11.9. Technika inpaintingu nie jest czarodziejską różdżką do odzyskiwania całkowicie utraconych tekstur. Na zdjęciu po lewej całkowicie wymazano gniazdo nasienne pomarańczy, przez co na zdjęciu po prawej owoc został uzupełniony teksturą miąższu

Oto prototyp funkcji `cv::inpaint()`:

```
void cv::inpaint(  
    cv::InputArray src,           // obraz wejściowy, 8 bitów, 1 lub 3 kanały  
    cv::InputArray inpaintMask, // 8 bitów, 1 kanał, niezerowe piksele  
    cv::OutputArray dst,        // obraz wynikowy  
    double inpaintRadius,      // promień obszaru do uwzględnienia wokół piksela  
    int flags                   // może mieć wartość NS lub TELEA  
);
```

Argument `src` reprezentuje 8-bitowy jednokanałowy obraz w skali szarości lub trzykanałowy obraz kolorowy, który ma być poddany operacji retuszu. Argument `inpaintMask` to 8-bitowy jednokanałowy obraz o takim samym rozmiarze jak `src`, w którym zniszczone obszary (np. napis widoczny po lewej stronie na rysunku 11.8) zostały zaznaczone niezerowymi pikselami. Wszystkie pozostałe piksele w tym obrazie są ustawione na 0. Wynik zostanie zapisany w obrazie `dst`, który musi mieć taki sam rozmiar i taką samą liczbę kanałów jak `src`. Argument `inpaintRadius` określa promień obszaru wokół każdego „wmalowywanego” piksela, jaki ma zostać wliczony w jego kolor wynikowy. Jak widać na rysunku 11.9, piksele wewnętrzne znajdujące się w odpowiednio dużym obszarze mogą w całości przyjmować kolory od innych „wmalowanych” pikseli, które znajdują się bliżej krawędzi. Prawie zawsze argument ten powinien mieć niewielką wartość, rzędu 3, ponieważ zbyt duży promień spowoduje powstanie widocznego zamazania. I wreszcie za pomocą parametru `flags` można eksperymentować z dwiema metodami inpaintingu: `cv::INPAINT_NS` (metoda Naviera-Stokesa) i `cv::INPAINT_TELEA` (autorem tej metody jest A. Telea).

## Usuwanie szumów

Kolejnym poważnym problemem dotyczącym obrazów jest szum. Jego głównym źródłem w wielu przypadkach są niedostatki oświetlenia przy wykonywaniu zdjęcia. W słabym oświetleniu aparat musi zastosować wzmocnienie sygnału, w wyniku czego dochodzi także do wzmocnienia szumu. Niedoskonałości tego typu mają zwykle charakter rozproszonych punktów o zbyt dużej lub zbyt małej jasności, choć zdarzają się też odbarwienia na kolorowych obrazach.

W OpenCV zaimplementowany jest algorytm usuwania szumów o nazwie **FNLMD** (ang. *Fast Non-Local Means Denoising*), który powstał na podstawie prac Antoniego Buadesa, Bartomeu Colla i Jeana-Michela Morela [Buades05]. Podczas gdy działanie prostych algorytmów odsumiających z reguły opiera się na uśrednianiu wartości indywidualnych pikseli na podstawie wartości pikseli sąsiednich, centralną koncepcją algorytmu FNLMD jest wyszukiwanie *podobnych pikseli* w innych częściach obrazu i uśrednianie wartości w odniesieniu do nich. Podobne do siebie piksele to nie takie, które mają zbliżone kolor lub intensywność, lecz takie, które znajdują się w podobnym otoczeniu. Pomysł ten opiera się na spostrzeżeniu, że większość obrazów zawiera powtarzalne struktury, więc nawet jeśli dany piksel zostanie zniszczony przez szum, można go odzyskać na podstawie innych pikseli, które uniknęły tego losu.

Identyfikacja podobnych pikseli odbywa się przy użyciu okienka  $B(p, s)$  ustawionego centralnie nad pikselem  $p$  i mającego rozmiar  $s$ . Po ustawieniu takiego okna nad punktem, który chcemy zmienić, możemy porównać jego zawartość z zawartością analogicznego okna ustawionego nad innym pikselem  $q$ . Kwadrat odległości między punktami  $B(p, s)$  i  $B(q, s)$  obliczamy według następującego wzoru:

$$d^2(B(p, s), B(q, s)) = \frac{1}{3(2s+1)} \sum_{c=1}^3 \sum_{j \in B(0, s)} (I_c(p+j) - I_c(q+j))^2$$

We wzorze tym  $c$  jest indeksem koloru,  $I_c(p)$  reprezentuje intensywność obrazu w kanale  $c$  w punkcie  $p$ , a suma nad  $j$  dotyczy elementów płamy. Na podstawie kwadratu odległości przypisuje się każdemu pikselowi obrazu wagę względem piksela, który jest właśnie modyfikowany. Wagę tę obliczamy według następującego wzoru:

$$w(p, q) = e^{-\frac{\max(d^2 - 2\sigma^2, 0.0)}{h^2}}$$

W tej funkcji symbol  $\sigma$  oznacza spodziewaną wartość odchylenia standardowego szumów w obrazie (w jednostkach intensywności), natomiast  $h$  reprezentuje ogólny parametr filtrowania określający, jak szybko plamy będą stawać się nieistotne wraz ze wzrostem ich kwadratu odległości od modyfikowanej plamy. Generalnie rzecz biorąc, wzrost wartości  $h$  pociąga za sobą wzrost ilości usuniętego szumu, ale kosztem utraty pewnych szczegółów obrazu. Mniejsza wartość  $h$  pozwala lepiej zachować obraz, lecz pozostawia też więcej szumu.

Zwykle korzyści z uwzględniania plam bardzo oddalonych (w pikselach) od aktualizowanego piksela są coraz mniejsze, ponieważ liczba takich plam wzrasta kwadratowo wraz z dozwolonym dystansem. Dlatego normalnie definiuje się cały obszar, zwany **oknem wyszukiwania**, i w aktualizacji biorą udział tylko plamy znajdujące się w tym oknie. Wówczas aktualizacja bieżącego piksela jest obliczana

na podstawie zwykłej średniej ważonej wszystkich pozostałych pikseli w oknie wyszukiwania z wykorzystaniem zanikających wykładniczo wag<sup>12</sup>. Dlatego algorytm ma w nazwie człon „nielokalny”. Plamy uwzględniane w retuszu danego piksela są tylko luźno skorelowane z lokalizacją tego piksela.

Implementacja algorytmu FNLMD w bibliotece OpenCV obejmuje kilka różnych funkcji, z których każda ma zastosowanie w nieco innej sytuacji.

### Podstawowy algorytm FNLMD — funkcja `cv::fastNlMeansDenoising()`

```
void cv::fastNlMeansDenoising(
    cv::InputArray src,          // obraz wejściowy
    cv::OutputArray dst,       // obraz wynikowy
    float h = 3,                // parametr rozkładu wagi
    int templateWindowSize = 7, // rozmiar plam używanych do porównywania
    int searchWindowSize = 21   // maksymalna odległość plamy do uwzględnienia
);
```

Pierwsza z czterech funkcji implementujących algorytm FNLMD to `cv::fastNlMeansDenoising()`, która reprezentuje jego dokładną implementację. Tablica wynikowa `dst` jest obliczana z tablicy wejściowej `src` przy użyciu obszaru `templateWindowSize` i parametru rozkładu `h` oraz z uwzględnieniem plam w oknie `searchWindowSize`. Obraz może być jedno-, dwu- lub trzykanałowy oraz musi być typu `cv::U8`<sup>13</sup>. Tabela 11.2 zawiera wykaz niektórych wartości dostarczonych przez autorów algorytmu, na które można ustawiać parametr rozkładu `h`.

Tabela 11.2. Zalecane wartości dla funkcji `cv::fastNlMeansDenoising()` i obrazów w skali szarości

Szum: $\sigma$	Rozmiar plamy: $s$	Okno wyszukiwania	Parametr rozkładu
$0 < \sigma \leq 15$	3×3	21×21	$0.40 \cdot \sigma$
$15 < \sigma \leq 30$	5×5	21×21	$0.40 \cdot \sigma$
$30 < \sigma \leq 45$	7×7	35×35	$0.35 \cdot \sigma$
$45 < \sigma \leq 75$	9×9	35×35	$0.35 \cdot \sigma$
$75 < \sigma \leq 100$	11×11	35×35	$0.30 \cdot \sigma$

### Algorytm FNLMD dla kolorowych obrazów — funkcja `cv::fastNlMeansDenoisingColor()`

```
void cv::fastNlMeansDenoisingColored(
    cv::InputArray src,          // obraz wejściowy
    cv::OutputArray dst,       // obraz wynikowy
    float h = 3,                // parametr rozkładu wagi jasności
    float hColor = 3,          // parametr rozkładu wagi koloru
    int templateWindowSize = 7, // rozmiar plam używanych do porównywania
    int searchWindowSize = 21   // największa brana pod uwagę odległość plamy
);
```

<sup>12</sup> W tym miejscu należy zwrócić uwagę na jeden drobiazg: waga wkładu piksela  $p$  w dotyczące go obliczenia wynosi  $w(p,p) = e^0 = 1$ . Generalnie oznacza to zbyt wysoką wagę względem innych podobnych pikseli, przez co wartość  $p$  zmienia się tylko odrobinę. Z tego względu wagę  $p$  określa się jako największą wartość  $z$  wag pikseli w obszarze  $B(p,s)$ .

<sup>13</sup> Choć obraz ten może mieć kilka kanałów, funkcja ta nie jest najlepszym narzędziem do pracy z obrazami kolorowymi. W takim przypadku lepiej sprawdza się funkcja `cv::fastNlMeansDenoisingColored()`.

Drugi wariant algorytmu FNLMD służy do pracy z kolorowymi obrazami i przyjmuje tylko obrazy typu `cv::U8C3`. Choć zasadniczo za pomocą tego algorytmu można by było mniej lub bardziej bezpośrednio przetwarzać obraz w formacie RGB, w praktyce lepszym pomysłem jest jego przekonwertowanie na inną przestrzeń kolorów przed rozpoczęciem obliczeń. Funkcja `cv::fastNlMeansDenoising`  $\hookrightarrow$  `Colored()` najpierw konwertuje obraz na przestrzeń kolorów LAB, następnie wykonuje algorytm FNLMD, a na koniec z powrotem konwertuje wynik na format RGB. Największą zaletą tego rozwiązania wiąże się z faktem, że w obrazie kolorowym, w efekcie, są trzy parametry rozkładu. Natomiast w reprezentacji RGB jest mało prawdopodobne, aby trzeba było ustawiać którykolwiek z nich na osobną wartość. W przestrzeni LAB przypisywanie komponentowi jasności innego parametru rozkładu niż komponentom kolorów jest czymś naturalnym. Funkcja `cv::fastNlMeansDenoising`  $\hookrightarrow$  `Colored()` to umożliwiła. Argument `h` reprezentuje parametr rozkładu jasności, podczas gdy nowy parametr `hColor` dotyczy kanałów kolorów. Generalnie wartość parametru `hColor` jest nieco mniejsza od wartości `h`. W większości przypadków odpowiednia jest wartość 10. Tabela 11.3 zawiera wykaz niektórych wartości, na które można ustawić parametr rozkładu `h`.

Tabela 11.3. Zalecane wartości dla funkcji `cv::fastNlMeansDenoising()` i kolorowych obrazów

Szum: $\sigma$	Rozmiar plamy: $s$	Okna wyszukiwania	Parametr rozkładu: $h$
$0 < \sigma \leq 25$	3×3	21×21	$0.55 \cdot \sigma$
$25 < \sigma \leq 55$	5×5	35×35	$0.40 \cdot \sigma$
$55 < \sigma \leq 100$	7×7	35×35	$0.35 \cdot \sigma$

### Algorytm FNLMD dla filmów — funkcje `cv::fastNlMeansDenoisingMulti()` i `cv::fastNlMeansDenoisingColorMulti()`

```
void cv::fastNlMeansDenoisingMulti(
    cv::InputArrayOfArrays srcImgs, // sekwencja kilku obrazów
    cv::OutputArray dst,           // obraz wynikowy
    int imgToDenoiseIndex,        // indeks obrazu do odsumienia
    int temporalWindowSize,       // liczba obrazów do użycia (nieparzysta)
    float h = 3,                  // parametr rozkładu wagi
    int templateWindowSize = 7,   // rozmiar plam porównawczych
    int searchWindowSize = 21    // maksymalna odległość plamy
);
void cv::fastNlMeansDenoisingColoredMulti(
    cv::InputArrayOfArrays srcImgs, // sekwencja kilku obrazów
    cv::OutputArray dst,           // obraz wynikowy
    int imgToDenoiseIndex,        // indeks obrazu do odsumienia
    int temporalWindowSize,       // liczba obrazów do użycia (nieparzysta)
    float h = 3,                  // parametr rozkładu wagi
    float hColor = 3,             // parametr rozkładu wagi dla koloru
    int templateWindowSize = 7,   // rozmiar plam porównawczych
    int searchWindowSize = 21    // maksymalna odległość plamy
);
```

Trzeci i czwarty wariant implementacji omawianego algorytmu służą do pracy z sekwencjami obrazów, na przykład zapisanymi w postaci filmu wideo. Jeśli chodzi o sekwencje obrazów, to naturalnie można się spodziewać, że informacje przydatne przy odsumianiu mogą znajdować się także w innych klatkach niż bieżąca. W większości aplikacji szum zmienia się z obrazu na obraz, podczas gdy sygnał z dużym prawdopodobieństwem pozostaje podobny lub nawet identyczny. Funkcje `cv::fastNlMeansDenoisingMulti()` i `cv::fastNlMeansDenoisingColorMulti()` pobierają tablicę obrazów, `srcImgs`, a nie pojedyncze obrazy. Ponadto należy podać informację, który obraz z sekwencji ma być poddany odsumianiu. Służy do tego parametr `imgToDenoiseIndex`. Dodatkowo należy dostar-

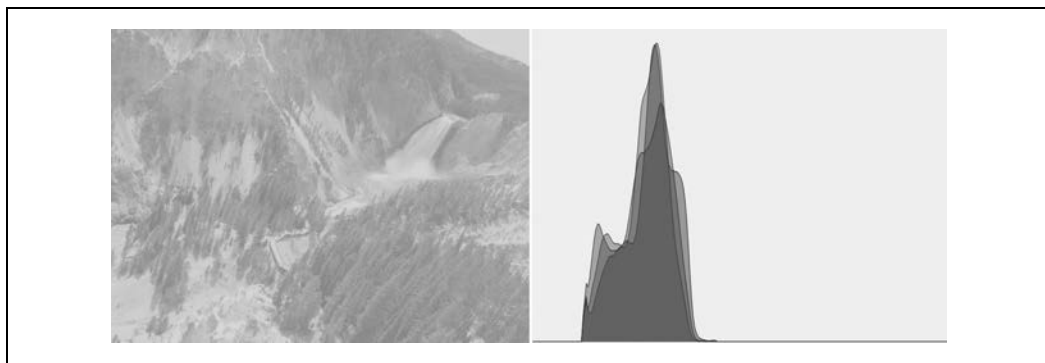


czyć okno wskazujące, ile obrazów z sekwencji ma być wykorzystanych do odsumiania. Parametr ten musi mieć wartość nieparzystą, a implikowane okno zawsze jest wycentrowane na obrazie `imgToDenoiseIndex`. (Gdyby więc parametr `imgToDenoiseIndex` ustawiono na 4, a `temporalWindowSize` na 5, to przy odsumianiu zostałyby wykorzystane obrazy 2, 3, 4, 5 i 6).

## Wyrównywanie histogramu

Kamery i czujniki obrazu muszą nie tylko uchwycić naturalny kontrast sceny, ale także dostosowywać parametry ekspozycji do aktualnego poziomu oświetlenia. W standardowym aparacie ilość światła docierającego do czujników reguluje się za pomocą migawki i przysłony. Natomiast zakres wartości kontrastu w danym obrazie często przekracza możliwości dostępnego zakresu dynamicznego czujnika. W efekcie konieczne jest poszukiwanie kompromisu między rejestrowaniem ciemnych obszarów (np. cieni), które wymagają dłuższego czasu ekspozycji, i jasnych obszarów, które wymagają krótszego czasu ekspozycji, aby zapobiec nasyceniu białych plam. Często zdarza się tak, że nie jest możliwe skuteczne pogodzenie tych dwóch wymogów w jednym obrazie.

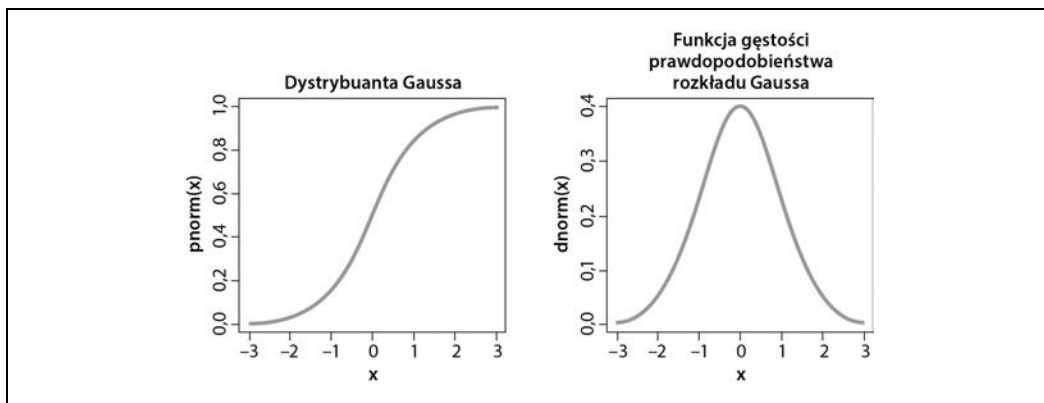
Gdy zdjęcie jest już zrobione, nic nie da się zrobić z nastawieniami czujnika, ale można spróbować poprawić zakres dynamiczny obrazu, aby zwiększyć jego kontrast. Jedną z najczęściej wykorzystywanych technik tego typu jest **wyrównywanie histogramu** (ang. *histogram equalization*)<sup>14</sup>. Na rysunku 11.10 widać, że przyczyną słabej jakości obrazu po lewej jest niewystarczające zróżnicowanie zakresu wartości. Dobitnie jest to widoczne na histogramie wartości intensywności pokazanym po prawej. Ponieważ jest to obraz 8-bitowy, jego zakres intensywności mieści się w przedziale od 0 do 255, a na histogramie wyraźnie widać, że wszystkie wartości są stłoczone w pobliżu środka przedziału. Technika wyrównywania histogramu polega właśnie na rozszerzaniu zakresu takich wartości.



Rysunek 11.10. Obraz po lewej ma niski kontrast, co potwierdza histogram intensywności widoczny po prawej

<sup>14</sup> Wyrównywanie histogramu to stara technika matematyczna, której zastosowanie w przetwarzaniu grafiki zostało opisane w różnych podręcznikach [Jain86, Russ02, Acharya05], referatach konferencyjnych [Schwarz78], a nawet w odniesieniu do obrazowania żywych tkanek [Laughlin81]. Jeśli zastanawiasz się, dlaczego techniki wyrównywania histogramu nie opisujemy w rozdziale poświęconym histogramom (rozdział 13.), to nie robimy tego, ponieważ technika ta bezpośrednio nie wykorzystuje żadnych histogramowych typów danych. Wprawdzie histogramy są wykorzystywane wewnętrznie, ale z punktu widzenia użytkownika nie są one potrzebne.

Matematyczne podstawy techniki wyrównywania histogramu opierają się na odwzorowywaniu jednego rozkładu (danego histogramu wartości intensywności) na inny rozkład (szerszy i najlepiej jednostajny). Innymi słowy chodzi o to, aby wartości  $y$  pierwotnego rozkładu jak najrównomierniej rozciągnąć w rozkładzie docelowym. Jak wiadomo, istnieje dobry sposób, by tak rozprowadzić wartości, polegający na wykorzystaniu funkcji zwanej **dystrybuantą**. Na rysunku 11.11 przedstawiamy przykład zastosowania tej funkcji w odniesieniu do nieco wyidealizowanego przypadku gausowskiego rozkładu gęstości, choć należy pamiętać, że technikę tę można zastosować w odniesieniu do każdego rodzaju rozkładu. Jest to po prostu skumulowana wartość gęstości prawdopodobieństwa oryginalnego rozkładu od jego ujemnej do dodatniej granicy.



Rysunek 11.11. Wynik obliczenia dystrybuanty (po lewej) dla rozkładu Gaussa

Za pomocą dystrybuanty oryginalny rozkład można zmienić na bardziej wyrównany (rysunek 11.12), wyszukując po prostu każdą wartość  $y$  w rozkładzie oryginalnym i znajdując dla niej odpowiednie miejsce w rozkładzie wyrównanym. W przypadku rozkładów ciągłych uzyskuje się dokładne wyrównanie, natomiast w przypadku rozkładów cyfrowych/dyskretnych rezultaty mogą być dalekie od tego ideału.

Po zastosowaniu tej techniki do zdjęcia z rysunku 11.10 otrzymujemy wyrównany histogram rozkładu intensywności, czego efektem jest poprawa jakości obrazu widoczna na rysunku 11.13.

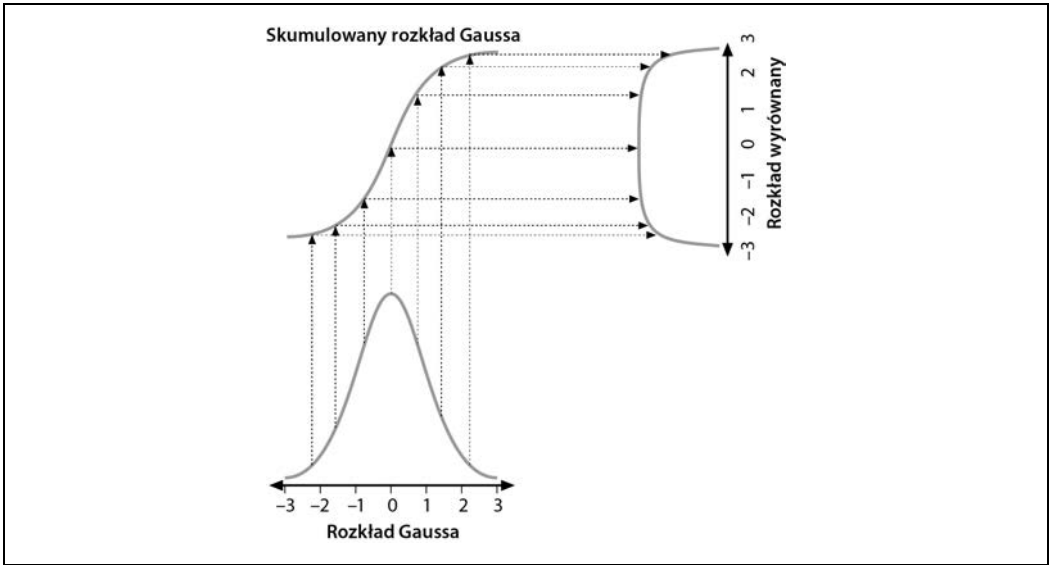
## cv::equalizeHist() — wyrównywanie kontrastu

W bibliotece OpenCV opisana technika jest zaimplementowana w postaci jednej zgrabnej funkcji:

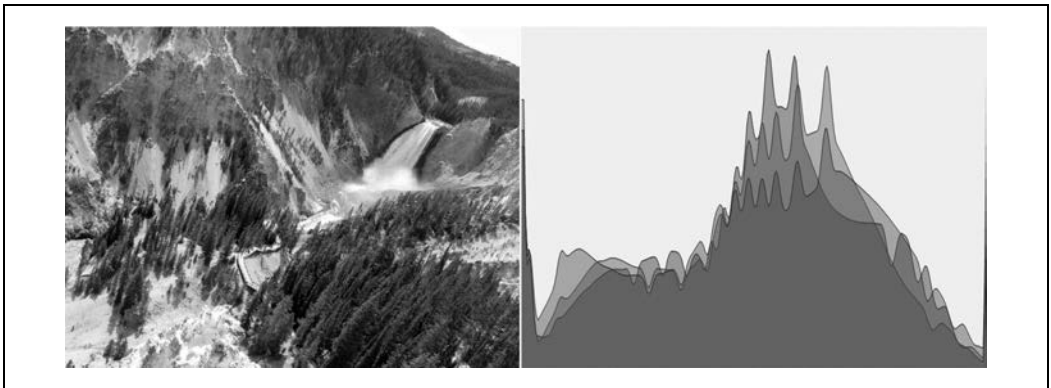
```
void cv::equalizeHist(
    const cv::InputArray src, // obraz wejściowy
    cv::OutputArray dst      // obraz wynikowy
);
```

W funkcji `cv::equalizeHist()` źródłowa tablica `src` musi być jednokanałowym 8-bitowym obrazem. Wynikowy obraz `dst` będzie miał takie same parametry. Obrazy kolorowe można retuszować, rozdzielając je na pojedyncze kanały i każdy z nich przetwarzając osobno<sup>15</sup>.

<sup>15</sup> W praktyce efekt osobnego wyrównywania histogramu poszczególnych kanałów rzadko jest zadowalający. Często lepszym rozwiązaniem jest konwersja obrazu na bardziej odpowiednią do takich operacji przestrzeń kolorów (np. LAB) i przeprowadzenie wyrównywania histogramu tylko w kanale jasności.



Rysunek 11.12. Wyrównywanie rozkładu Gaussa przy użyciu dystrybuanty



Rysunek 11.13. Efekt wyrównania histogramu — zakres został rozszerzony

## Podsumowanie

W tym rozdziale przedstawiliśmy różne metody przekształcania obrazów. Opisaliśmy techniki skalowania, a także przekształcenia afiniczne i perspektywiczne. Pokazaliśmy, jak zamieniać reprezentacje w kartezjańskim układzie współrzędnych na współrzędne biegunowe. Wszystkie opisane przez nas funkcje łączy to, że pobierają pewien obraz i zamieniają go w inny za pomocą jakiejś globalnej operacji odnoszącej się do całej grafiki. Przedstawiliśmy nawet funkcję, za pomocą której można wykonywać ogólne odwzorowania i w stosunku do której wiele innych funkcji opisanych w tym rozdziale można traktować jako jej szczególne przypadki.

Ponadto nieco miejsca poświęciliśmy niektórym algorytmom wykorzystywanym w retuszu fotografii, takim jak inpainting, odszumianie i wyrównywanie histogramu. Za pomocą tych algorytmów można poprawiać obrazy z aparatu fotograficznego i kamery, choć często znajdują też zastosowanie w implementacji innych technik wizualnych dotyczących ziarnistych lub w inny sposób uszkodzonych danych wideo.

## Ćwiczenia

1. Znajdź i załaduj obraz twarzy zwróconej przodem, z otwartymi oczami i zajmującej większość powierzchni obrazu. Napisz program znajdujący źrenice oczu.



Laplasjan „lubi” jasne centralnie położone punkty z ciemną otoczką, a źrenice mają cechy odwrotne. Wystarczy więc wykonać inwersję i splot z odpowiednio dużym laplasjanem.

2. Spójrz na wykresy przedstawiające, jak funkcja konwersji na współrzędne logarytmiczno-biegunowe zamienia kwadrat w falowaną linię.
  - a. Narysuj wyniki konwersji na współrzędne logarytmiczno-biegunowe dla przypadku, gdy punkt środkowy znajduje się w jednym z rogów kwadratu.
  - b. Jak wyglądałoby koło po przekształceniu na współrzędne logarytmiczno-biegunowe, gdyby punkt centralny znajdował się wewnątrz tego koła i blisko jego krawędzi?
  - c. Narysuj wynik transformacji dla przypadku, gdy punkt centralny znajduje się tuż za krawędzią koła.
3. W wyniku transformacji na współrzędne logarytmiczno-biegunowe kształty o różnych kątach rotacji i rozmiarach są przenoszone do przestrzeni, w której te właściwości zamieniają się w przesunięcia względem osi  $\theta$  i  $\log(r)$ . Transformacja Fouriera jest niezmienna względem translacji. Jak wykorzystać te informacje w celu wymuszenia na kształtach o różnych rozmiarach i kątach rotacji automatycznego zwracania ekwiwalentnej reprezentacji w dziedzinie logarytmiczno-biegunowej?
4. Narysuj obrazy przedstawiające duży, mały, duży obrócony i mały obrócony kwadrat. Każdy z nich poddaj przekształceniu do dziedziny logarytmiczno-biegunowej. Napisz dwuwymiarowy przesuwacz, który pobiera punkt centralny z reprezentacji logarytmiczno-biegunowej i tak przesuwa kształty, aby były do siebie maksymalnie podobne.
5. Załaduj obraz i poddaj go przekształceniu perspektywicznemu, a następnie rotacji. Czy można te dwa przekształcenia wykonać za jednym razem?
6. Inpainting sprawdza się bardzo dobrze w przypadku usuwania napisów z wzorzystych obszarów. Co by się stało, gdyby napis zasłaniał prawdziwą krawędź obiektu na obrazie? Sprawdź to.
7. Wypróbuj działanie techniki wyrównywania histogramu na kilku własnych obrazach i opisz wyniki.
8. Wyjaśnij różnicę między wyrównywaniem histogramu obrazu a usuwaniem szumu z obrazu.

## A

- AdaBoost, 733, 734
- akronim FLANN, 514
- aktualizacja okna, 203
- akumulowanie
  - średnich, 414
  - wariancji, 414
  - kowariancji, 414
- algorytm
  - AdaBoost, 733
  - Boosting, 684
  - Bougueta, 646
  - BRISK, 499, 501
  - Camshift, 540
  - C-vector SVM, 760
  - detektor twarzy/klasyfikator kaskadowy, 685
  - Douglasa-Peuckera, 383
  - drzewa decyzyjne, 684
  - drzewa losowe, 684
  - Dual TV-L, 529
  - EMD, 354
  - FLANN, 683, 684
  - flood fill, 329
  - FNLMD, 300–302
  - Grabcuts, 334
  - Hartleya, 644
  - Horna-Schnucka, 526
  - KB, 439
  - klasyfikacji z maksymalnym marginesem, 757
  - KNN, 747
  - kondensacji, 549
  - k-najbliższych sąsiadów, 684
  - k-średnich, 684, 694, 795
  - LMedS, 590
  - Lucasa-Kanade'a, 451, 457
  - maksymalizacja wartości oczekiwanej, 684
  - maszyna wektorów nośnych, 685
  - mean-shift, 537
  - normalny/naiwny klasyfikator Bayesa, 684
  - odległość Mahalanobisa, 684
  - Otsu, 242
  - Polynomial Expansion, 526
  - propagacji wstecznej
    - parametry szkolenia, 756
  - rozwiązania wielomianu Farnebäcka, 527
  - Rprop, 753
  - Shape Context, 396
  - SIFT, 491
  - Simple Flow, 533, 535
  - Star, 496
  - SUSAN, 482
  - SVM
    - SVR, 764
    - jednoklasowy SVM, 764
    - klasyfikator C-SVM, 764
    - klasyfikator v-SVM, 764
    - v-SVR, 764
    - wersja jednoklasowa, 761
    - wieloklasowe rozszerzenie, 761
  - Teha-China, 375
  - utajona maszyna wektorów nośnych, 685
  - wododziałowy, 332
  - Waldboost, 685
  - Worek słów, BOW, 685, 793
  - wstecznej propagacji, 753
  - wykrywania cech FAST, 482
  - wyszukiwania punktów kluczowych, 505
  - znajdowania punktów kluczowych, 477, 480, 488, 493, 502
- algorytmy
  - dyskryminacyjne, 683
  - generacyjne, 683
  - grupujące, 680, 682, 695
  - klasyfikujące, 680
  - ML, 683
  - uczenia maszyn, 688, 714

- analiza
  - Fouriera, 307
  - komponentów połączonych, 380
  - obrazu, 307
  - składowych głównych, PCA, 167
- aparatus, 591
- aplikacja
  - createsamples, 784
  - traincascade, 787
- aproksymacja
  - Gausa, 168
  - wielokątów, 382
- argument
  - borderType, 239
  - cmpop, 119
  - descriptorMatcherType, 473
  - dim, 147
  - dtype, 112
  - flags, 117
  - imageSize, 597
  - method, 132, 149
  - normType, 143
  - reduceOp, 147
- argumenty programu traincascade, 787
- autokorelacja, 476
- automatyczna selekcja indeksów, 518

## B

- Bag of Keypoints, 793
- baseny, 332
- bezwonny filtr cząstek, 565
- biblioteka
  - FLANN, 803
  - HighGUI, 48
  - IPPICV, 32
  - ML, 694
  - podstawowe procedury, 705
  - OpenCV, 25
  - OpenGL, 220
  - Qt, 212, 222
  - WTL, 229
  - wxWidgets, 225
- biblioteki
  - GUI, 221
  - IPP, 25, 32
- bilateralna interpolacja łączna, 534
- binarne drzewa decyzyjne, 719
- Black Hat, 265

- blokowe metody dostępowe, 96
- błąd reprojekcji, 599
- błędna klasyfikacja, 720
- Boosting, 684, 733
- bootstrapping, 690
- BOW, Bag of Words, 685, 793
- BRIEF, 497

## C

- CCS, complex conjugate symmetrical, 308
- cechy, features, traits, 72, 680
  - BRISK, 500
  - Calondera, 497
  - diagonalne, 772
  - FAST, 483
  - FREAK, 507
  - LBP, 773
  - oFAST, 504
  - ORB, 504
  - ORG, 503
  - SIFT, 485
  - sumaryczne, 383
  - SURF, 490
  - typu Haara, 773
- CenSurE, Center Surround Extremum, 495
- charakterystyka operacyjna odbiornika, ROC, 691
- czcionki, 163, 164, 217
- część wspólna histogramów, 350
- czułość, 476
- czyszczenie pierwszego planu, 431

## D

- dane z kamery, 55
- DCT, discrete cosine transform, 312
- dealokacja, 87
- dekompresja, 183
- deskryptor, 445, 461, 487
  - BRIEF, 498, 501
- detektor
  - cech
    - AGAST, 499
    - BRISK, 500
    - Harrisa-Shi-Tomasiego, 475
    - ORG, 503
    - SIFT, 485
    - Star/CenSurE, 494
    - SURF, 490
  - cv::dpm::DPMDetector, 793

krawędzi Canny'ego, 54, 317  
plam, 479  
punktów kluczowych, 510  
twarzy, 685  
Violi-Jonesa, 772  
diagnostowanie usterek, 688  
dokumentacja  
dołączona do biblioteki, 37  
internetowa, 37  
dopasowywanie, 395, 513  
blokowe, 526, 652, 653, 660  
częściowo globalne, 660  
semiglobalne, 653  
cech, 693  
konturów i obrazów, 390  
linii, 673  
promieniowe, 472  
punktów kluczowych, 469  
siłowe, 513  
szablonów, 362  
dostęp  
do elementów tablicy, 90  
do elementów tablicy rzadkiej, 101  
dostosowywanie danych  
odległość Mahalanobisa, 699  
drzewa  
decyzyjne, 684, 721, 725  
binarne, 719  
implementacja, 722  
miary nieczystości, 721  
użycie, 725  
zwracane wyniki, 729  
klasyfikacji i regresji, 719  
konturów, 372  
k-wymiarowe, 515  
losowe, 684, 733, 739  
użycie, 740  
zastosowania, 742  
Dual TV-L, 526  
duże typy tablicowe, 59, 85  
struktury szablonowe, 104  
dwukanałowa reprezentacja zmiennoprzecinkowa,  
602  
dylatacja, 257  
dyskretna transformacja  
cosinusowa, DCT, 312  
Fouriera, 307  
dystrybuanta, 304  
dzielenie danych szkoleniowych, 711

## E

efekty rozbieżności, 613  
ekstraktor  
cech, 488, 493, 499, 502, 505  
deskryptorów BRIEF, 497  
deskryptorów FREAK, 506  
konturów, 374  
kosztów histogramu, 397  
odległości Hausdorffa, 399  
odległości kontekstu kształtu, 398  
ekstrapolacja, 239  
krawędzi, 236  
ekstremum przestrzeni skali, 485  
element strukturalny, 267  
elementy macierzy samosprężonych, CCS, 308  
EM, expectation maximization, 743  
entropia, 720  
erozja, 257  
estymatory, 548

## F

FAST, Features from Accelerated Segments Test, 482  
faza przewidywania, 548  
FFAAD, 412  
ffmpeg, 47  
filtr, 235  
bilateralny, 250  
Gaussa, 248  
Kalmana, 549, 550, 560  
rozszerzony, 564  
liniowy, 268  
medianowy, 236, 247  
ogólny, 270  
prostokątny, 246  
Scharra, 253  
filtrowanie  
bilateralne, 250  
obrazów, 235  
punktów kluczowych, 512  
FLANN, 514, 684, 803  
fluktuacje linii pikseli, 406  
format  
CSV, 710  
logarytmiczno-biegunowy, 56  
funkcja  
cvCeil(), 77  
cvFindFundamentalMat(), 634  
cvFloor(), 78

## funkcja

cv::cvtColor(), 121  
cv::cvtColor(), 123  
cv::dct(), 109, 122, 312  
cv::deallocate(), 77  
cv::DescriptorMatcher::create(), 473  
cv::destroyWindow(), 47  
cv::determinant(), 109, 126  
cv::dft(), 110, 122, 308  
cv::distanceTransform(), 327  
cv::divide(), 110, 127  
cv::drawChessboardCorners(), 584  
cv::drawKeypoints, 519  
cv::drawMatches, 520  
cv::eigen(), 110, 127  
cv::ellipse(), 159  
cv::ellipse2Poly(), 161  
cv::equalizeHist(), 304  
cv::error(), 78  
cv::exp(), 110, 128  
cv::extractImageCOI(), 110, 128  
cv::fastAtan2(), 77  
cv::fastFree(), 78  
cv::fastMalloc(), 78  
cv::fastNlMeansDenoising(), 301  
cv::fastNlMeansDenoisingColor(), 301  
cv::fastNlMeansDenoisingColorMulti(), 302  
cv::fastNlMeansDenoisingMulti(), 302  
cv::FileNode, 194  
cv::fillConvexPoly(), 161  
cv::fillPoly(), 161  
cv::filter2D(), 269  
cv::findChessboardCorners(), 583  
cv::findCirclesGrid(), 585  
cv::findContours(), 374  
cv::fitEllipse(), 386  
cv::fitLine(), 386  
cv::flip(), 110, 128  
cv::format(), 79  
cv::gemm(), 110, 128  
cv::getAffineTransform(), 285, 616  
cv::getConvertElem(), 110, 129  
cv::getConvertScaleElem(), 110, 129  
cv::getCPUTickCount(), 79  
cv::getDerivKernel(), 270  
cv::getGaussianKernel(), 271  
cv::getNumThreads(), 79  
cv::getOptimalDFTSize(), 79  
cv::getPerspectiveTransform(), 289, 616  
cv::getRotationMatrix2D(), 616



cv::getTextSize(), 165  
 cv::getThreadNum(), 80  
 cv::getTickCount(), 80  
 cv::getTickFrequency(), 80  
 cv::goodFeaturesToTrack(), 447, 478  
 cv::HoughCircles(), 325  
 cv::HoughLines(), 321  
 cv::HoughLinesP(), 323  
 cv::HuMoments(), 395  
 cv::idct(), 110, 130, 313  
 cv::idft(), 110, 130, 310  
 cv::imencode(), 183  
 cv::imread(), 46, 181  
 cv::imshow(), 46, 52, 202  
 cv::imwrite(), 182  
 cv::initUndistortRectifyMap(), 604, 650  
 cv::inRange(), 110, 130  
 cv::insertImageCOI(), 131  
 cv::integral(), 316  
 cv::invert(), 110, 131  
 cv::isContourConvex(), 390  
 cv::line(), 162  
 cv::LineIterator, 163  
 cv::log(), 110, 132  
 cv::logPolar(), 296  
 cv::LUT(), 110, 132  
 cv::magnitude(), 110  
 cv::Mahalanobis(), 110, 133, 421  
 cv::max(), 110, 133  
 cv::mean(), 135  
 cv::meanStdDev(), 110, 135  
 cv::merge(), 110, 136  
 cv::min(), 110, 136  
 cv::minAreaRect(), 384  
 cv::minEnclosingCircle(), 385  
 cv::minMaxIdx(), 137  
 cv::minMaxLoc(), 110, 137, 347  
 cv::mixChannels(), 110, 138  
 cv::ml::TrainData::loadFromCSV()  
 cv::moments(), 391  
 cv::morphologyEx(), 260  
 cv::mulSpectrums(), 110, 140, 311  
 cv::multiply(), 110, 140  
 cv::mulTransposed(), 110, 140  
 cv::namedWindow(), 46, 202  
 cv::norm(), 110, 141, 142  
 cv::normalize(), 110, 142  
 cv::optflow::calcOpticalFlowSF(), 535  
 cv::PCA::backProject(), 170  
 cv::PCA::operator()(), 169  
 cv::PCA::PCA(), 169  
 cv::PCA::project(), 169  
 cv::perspectiveTransform(), 110, 143, 291, 616  
 cv::phase(), 110, 144  
 cv::pointPolygonTest(), 389  
 cv::polarToCart(), 110, 145, 293  
 cv::polyLines(), 162  
 cv::pow(), 110, 145  
 cv::putText(), 164  
 cv::pyrDown(), 53, 279  
 cv::pyrUp(), 281  
 cv::randn(), 110, 146  
 cv::randShuffle(), 110, 146  
 cv::randu(), 110, 145  
 cv::rectangle(), 162  
 cv::reduce(), 111, 147  
 cv::remap(), 297, 606  
 cv::repeat(), 111, 148  
 cv::resize(), 278  
 cv::RNG(), 173  
 cv::RNG::fill(), 175  
 cv::RNG::gaussian(), 175  
 cv::RNG::operator T(), 174  
 cv::RNG::uniform(), 174  
 cv::saturate\_cast<>(), 111  
 cv::scaleAdd(), 111, 148  
 cv::sepFilter2D(), 270  
 cv::setIdentity(), 111, 148  
 cv::setNumThreads(), 81  
 cv::setUseOptimized(), 81  
 cv::solve(), 111, 149  
 cv::solveCubic(), 111, 150  
 cv::solvePnP(), 600, 622  
 cv::solvePnPPransac(), 601  
 cv::solvePoly(), 111, 150  
 cv::sort(), 111, 151  
 cv::sortIdx(), 111, 151  
 cv::SparseMat::find<>(), 102  
 cv::SparseMat::hash(), 101  
 cv::SparseMat::ptr(), 101  
 cv::SparseMat::ref<>(), 101  
 cv::split(), 111, 151  
 cv::sqrt(), 111, 152  
 cv::subtract(), 111, 153  
 cv::sum(), 111, 153  
 cv::SVD(), 171  
 cv::SVD::backSubst(), 172  
 cv::SVD::compute(), 171

## funkcja

- cv::SVD::operator(), 171
- cv::SVD::solveZ(), 172
- cv::theRNG(), 111, 173
- cv::threshold(), 240, 242
- cv::trace(), 111, 153
- cv::transform(), 111, 154, 288, 616
- cv::transpose(), 111, 154
- cv::undistort(), 606
- cv::undistortPoints(), 607
- cv::useOptimized(), 81
- cv::VideoCapture::get(), 188
- cv::VideoCapture::grab(), 187
- cv::VideoCapture::operator>>(), 187
- cv::VideoCapture::read(), 186
- cv::VideoCapture::retrieve(), 187
- cv::VideoCapture::set(), 188
- cv::VideoCapture::VideoCapture(), 186
- cv::VideoWriter::operator<<(), 191
- cv::VideoWriter::write(), 191
- cv::VideoWriter.release(), 56
- cv::waitKey(), 46, 203
- cv::warpAffine(), 285, 616
- cv::warpPerspective(), 289, 616
- Laplace'a, 254

## funkcje

- błędu, 751
- HighGUI, 46
- pomocnicze, 75
- przekształceń afinicznych, 616
- rysowania, 158
- rysowania tekstu, 164
- składowe klasy
  - cv::SparseMat, 103
  - cv::FileNode, 195
- systemowe, 75
- zwrotne, 205

funkcjonał energii, 334

funktory, 167

fuzja informacji, 551

## G

generator liczb losowych, RNG, 173

Gentle AdaBoost, 733

geometria

- epipolarna, 628
- rzutowa, 570

gęste siatki cech, 509

gęsty przepływ optyczny, 526

## głębia, 626

- stereo, 657, 663

gradient morfologiczny, 263

graficzny interfejs użytkownika, 201

GSoC, 805

## H

HAL, hardware acceleration layer, 16, 31

hierarchie konturów, 372

HighGUI, 44, 179, 201

histogram, 303, 341

- algorytmy dopasowywania, 354
- część wspólna, 350
- metody zaawansowane, 354
- normalizacja, 347
- porównywanie
  - metoda cv::COMP\_CORREL, 349
  - metoda cv::COMP\_CHISQR\_ALT, 350
  - metoda cv::COMP\_INTERSECT, 350
  - metoda cv::COMP\_BHATTACHARYYA, 350
- próg, 347
- tworzenie, 344
- zastosowanie, 351
- znajdowanie przedziału, 347

histogramy HOG, 293, 791

HOG, histogram of oriented gradients, 293, 791

homografia, 283, 587

- plaszczyznowa, 616

horopter, 655

## I

I/O, 41

identyfikowanie trójkąta

- granicznego, 822
- ramowego, 822

implementacja

- drzewa decyzyjnego, 721
- techniki odejmowania tła, 437

indeks

- Giniego, 720
- próbek, 712
- szkoleniowy, 712
- testowy, 712

indeksowanie

- drzew k-wymiarowych, 515
- hierarchicznych drzew k-średnich, 516
- liniowe, 515
- LSH, 517

informacje  
o argumentach aplikacji `createsamples`, 784  
o usuwaniu nieużytków, 69  
innowacja, 553  
inpainting, 299  
instalacja  
w systemie Linux, 35  
w systemie Mac OS X, 36  
w systemie Windows, 34  
interfejs `cv::StatModel`, 714  
interpolacja, 278  
intraoktawy, 500  
IPP, Integrated Performance Primitives, 25, 32, 41  
IPPICV, 32  
iteracyjna metoda Eulera-Lagrange'a, 531  
iterador  
liniowy, 405  
tablicowy N-arny, 93

## J

jądro, 267, 758  
gaussowskie, 486  
liniowe, 235  
jednoklasowy SVM, 761  
jednorodna zmiana rozmiaru, 278

## K

k-krotny sprawdzian krzyżowy, 765  
k-najbliższych sąsiadów, KNN, 684, 746  
kafelkowanie, 141  
kalibracja, 576, 592, 596, 607, 829  
aparatu, 591  
stereo, 639, 665, 676  
kamera, 55, 568  
kanały, 112  
wejściowe sterowania, 556  
kaskada odrzucenia, 774, 776  
katalog `opencv_contrib`, 825  
kategoryzacja, 796  
semantyczna, 793  
klasa  
`cv::BackgroundSubtractor`, 437  
`cv::BackgroundSubtractorMOG2`, 440  
`cv::BFMatcher`, 513  
`cv::BOWImgDescriptorExtractor`, 796  
`cv::BOWTrainer`, 794  
`cv::BRISK`, 502

`cv::CascadeClassifier`, 778  
`cv::DataType<>`, 72  
`cv::DenseFeatureDetector`, 509  
`cv::DescriptorMatcher`, 469  
`cv::DMatch`, 468  
`cv::dpm::DPMDetector`, 791  
`cv::Exception`, 72  
`cv::FastFeatureDetector`, 482  
`cv::Feature2D`, 465  
`cv::FileNode`, 195  
`cv::FileStorage`, 191  
`cv::FlannBasedMatcher`, 514  
`cv::GFTTDetector`, 478  
`cv::InputArray`, 74  
`cv::KalmanFilter`, 558  
`cv::KeyPoint`, 464  
`cv::KeyPointsFilter`, 512  
`cv::Mat`, 85, 96  
konstruktory, 87, 89  
konstruktory szablonowe, 90  
`cv::Matrix`  
operacje, 66  
`cv::ml::ANN_MLP`, 753  
`cv::ml::DTrees`, 735  
`cv::ml::KNearest`, 747  
`cv::ml::NormalBayesClassifier`, 718  
`cv::ml::SVM`, 762, 766  
`cv::ml::TrainData`, 707  
`cv::ml::TrainDataImpl`, 711  
`cv::ORB`, 503, 506  
`cv::OutputArray`, 74  
`cv::Range`, 69  
`cv::Rect`, 61, 64  
`cv::Rect`:operacje, 64  
`cv::RotatedRect`  
operacje, 65  
`cv::Scalar`, 61  
operacje, 63  
`cv::SimpleBlobDetector`, 479  
`cv::Size`, 61  
`cv::SparseMat`, 100  
funkcje składowe, 103  
`cv::StereoBM`, 653, 657  
`cv::StereoSGBM`, 653, 663  
`cv::SVD`, 170  
`cv::TermCriteria`, 68  
`cv::Vec<>`, 60  
operacje, 67  
`cv::VideoCapture`, 184

klasa  
 cv::VideoWriter, 190  
 cv::xfeatures2d::FREAK, 506  
 cv::xfeatures2d::SIFT, 485, 490, 494

klasy  
 dopasowywania stereo, 653  
 liczb zespolonych, 68  
 operacje, 68  
 macierzowe, 60, 65  
 prostokątów, 64  
 punktowe, 61, 62  
 operacje, 62  
 rozmiarów, 63  
 operacje, 64  
 wektorowe, 60, 67

klasyfikacja, 693, 721  
 odległość Mahalanobisa, 701

klasyfikator  
 Bayesa  
 naiwny, 684, 715  
 normalny, 684, 715, 718  
 SVM, 797  
 Viola-Jonesa, 778

klasyfikatory kaskadowe, 685, 771, 772

kod  
 kodeka, 57  
 wzmacniania, 735

kodeki, 183

kombinacja drzew k-wymiarowych, 517

kompilowanie modułów, 40

kompresja, 183

komputerowe rozpoznawanie obrazu, 27, 685

konstruktor  
 cv::ml::TreeParams, 723  
 FREAK, 508

konstruktory  
 klasy cv::Mat, 87, 89  
 szablonowe klasy cv::Mat, 90

kontrast, 304

kontury, 371  
 aproksymacja wielokątów, 382  
 hierarchie, 372  
 rysowanie, 376  
 znajdowanie, 374  
 znajdowanie powłoki wypukłej, 388

konwersja współrzędnych, 124–126  
 biegunowych na kartezjańskie, 293  
 kartezjańskich na biegunowe, 293

konwersje typów macryc, 604

konwolucyjne sieci neuronowe, 680

korekcja zniekształceń, 602

korekta, 548

korelacja, 349

korrespondencja, 665  
 stereo, 652

kowariancja, 414, 418

krawędzie, 237, 820  
 Canny’ego, 317  
 wirtualne, 820  
 zerowe, 820

kryteria spójności, 660

kryterium porządku, 655

krzywe ROC, 690, 692

księga kodów, 423  
 YUV, 422

k-średnich, 684, 694

## L

laplasjan, 254

Latent SVM, 790

LBP, local binary patterns, 773

liczba mieszanin gaussowskich, 438

likwidowanie zniekształceń obrazu, 602, 606, 623

linia, 157  
 epipolarna, 628, 638

lista zapytań, 471

LMedS, 634

LogitBoost, 733

lokalna linearyzacja, 564

lokalne  
 maksimum, 257  
 minimum, 257  
 wzory binarne, LBP, 773

lokalny deskryptor obrazu, 487

losowanie ze zwracaniem, 739

LSH, locality-sensitive hashing, 517

## ł

ładowanie obrazów, 181

łańcuch Freemana, 371

łańcuchy górskie, 332

## M

### macierz

- autokorelacji, 476
- fundamentalna, 613, 630, 633
- Hessego, 490
- homografii H, 589, 613
- kowariancji, 117, 419
- obrotu, 578
- parametrów wewnętrznych kamery, 571
- parametrów wewnętrznych M, 613
- przejścia, 556
- przekształcenia afinicznego, 285
- rotacji R, 613
- rzutowania perspektywicznego, 289
- transferu, 556
- zasadnicza, 630, 631

macierze, *Patrz także* wyrażenia macierzowe, tablice mnożenia uogólnione, 129  
rzadkie, 100

### makro

- CV\_DbgAssert(), 77
- CV\_Error\_(), 78

### maksymalizacja

- oczekiwań, EM, 743
- wartości oczekiwanej, 684

mapa nieregularności przepływu, 535

mapowanie niejednorodne, 282

mapy bitowe, 227

maski, 112

maszyna wektorów nośnych, 685, 757, 790, 797

### matryca

- głębi, 671
- likwidacji zniekształceń, 602
- rektyfikacji, 649
- zniekształceń, 602

menu podręczne, 213

metauczenie, 733

### metoda

- eliminacji Gaussa, 126
- EMD, 397
- gradientowa Hougha, 323
- inpaintingu, 299
- Kaewtrakulponga i Bowdena, 438
- Latent SVM, 790
- LMedS, 634
- Newtona, 454
- RANSAC, 590, 634
- segmentacji mean-shift, 336

uśredniania tła, 409

Zivkovic, 439

### metody

- dopasowywania, 395, 513
  - współczynnika korelacji, 364
- dostępowe, 96
- klasy cv::dpm::DPMDetector, 792
- korelacji krzyżowej, 364
- pomiaru odległości, 387
- porównywania
  - cv::COMP\_CORREL, 349
  - cv::COMP\_CHISQR\_ALT, 350
  - cv::COMP\_INTERSECT, 350
  - cv::COMP\_BHATTACHARYYA, 350
  - cv::TM\_SQDIFF, 363
  - cv::TM\_SQDIFF\_NORMED, 364
  - cv::TM\_CCORR, 364
  - cv::TM\_CCORR\_NORMED, 364
  - cv::TM\_CCOEFF, 364
  - cv::TM\_CCOEFF\_NORMED, 365
- rozpoznawania tła, 436
- wzmocniania, 733
  - Discrete AdaBoost, 733
  - Gentle AdaBoost, 733
  - LogitBoost, 733
  - Real AdaBoost, 733

### miara

- Harrisa, 476, 496
- nieczystości, 719
  - drzewa decyzyjnego, 721

mieszanie, 557

alfa, 114

miękką kaskada, 771

ML, machine learning, 679

MLP, multilayer perceptron, 749

### mnożenie

- macierzy, 129
- widm, 311

### model

- kamery, 568
- odszumiania wariancji całkowitej, 532
- tła, 403

### modele

- dyskryminacyjne, 683
- generacyjne, 683
- generatywne, 716
- tła oparte na księgach kodów, 430

modelowanie sceny, 405

- moduł
  - ML, 26, 683
  - shape, 396
- moduły repozytorium opencv\_contrib, 825
- momenty, 390
  - centralne, 392
  - centralne znormalizowane, 393
  - konturów, 383
  - niezmienne Hu, 393
- morfologia obrazu, 255
- mysz
  - typy zdarzeń, 205
  - znaczniki zdarzeń, 206

## N

- naiwny klasyfikator Bayesa, 715
- nakładka tekstowa, 213
- narzędzia GUI, 179
- nasycenie, 111
- nauka voodoo, 732
- neuron sztuczny, 749
- nieczystość
  - klasyfikacji, 721
  - regresji, 721
- nieregularności przepływu, 535
- niezmiennność
  - jasności, 451
  - rotacyjna, 462
- norma, 142
- normalizacja histogramu, 347
- normalny klasyfikator Bayesa, 715, 718
- numerowanie
  - krawędzi, 820
  - wierzchołków, 820

## O

- obiekt
  - cv::CascadeClassifier, 778
  - cv::DMatch, 468
  - cv::FileStorage, 191
  - cv::FileStorage, 193
  - cv::KeyPoint, 464
  - cv::ml::TrainData, 712
  - cv::VideoCapture, 184
  - cv::VideoWriter, 190
- obiekty
  - funkcyjne, 167
  - pierwszego planu, 428, 429

- pomocnicze, 59, 68
- wyszukiwania punktów kluczowych, 484
- obliczanie
  - długości, 384
  - elipsy, 386
  - gęstego przepływu optycznego, 528, 532
  - głębi stereo, 663, 657
  - kowariancji, 418
  - linii epipolarnych, 638
  - matryc likwidacji zniekształceń, 604
  - momentów, 391
  - najmniejszego okręgu, 385
  - najmniejszego prostokąta, 384
  - niezmiennych momentów Hu, 395
  - parametrów zewnętrznych, 600, 601
  - pozycji, 622
  - ramy, 384
  - standardowego obrazu całkowego, 316
  - średniej, 415, 416
  - wariancji, 417
- obracanie
  - krawędzi, 821
  - punktów, 579
- obraz
  - analiza, 307
  - całkowy, integral image, 106, 314, 778
  - historii ruchu, 541
  - piramidy, 279
  - przekształcenia ogólne, 277
  - renowacja, 298
  - szkoleniowy, 468
  - usuwanie szumów, 300
  - zapytania, 468
- obrazowanie stereo, 623
- obsługa
  - deskryptorów, 463
  - punktów kluczowych, 463
  - wyjątków, 72
- odczytywanie
  - danych, 193
  - klatek, 186, 187
- odejmowanie tła, 403
- implementacja, 437
- metoda Kaewtrakulponga i Bowdena, 438
- metoda zaawansowana, 422
- metoda Zivkovic, 439
- uśrednianie tła, 409
- wady, 404

odkształcanie, 282

- odległość
  - Bhattacharyyi, 350
  - Hausdorffa, 399
  - Mahalanobisa, 133, 421, 683, 699
  - ogniskowa, 568
- odometria wizualna, 463
- odporna propagacja wsteczna,  $R_{prop}$ , 753
- odtworzenie filmów, 47, 48
- odwrotna dyskretna transformacja
  - cosinusowa, 313
  - Fouriera, 310
- odwzorowania
  - arbitralne, 297
  - biegunowe, 292
  - odwrotne, 650
  - ogólne, 291, 277, 297
- ogólna funkcja morfologiczna, 260
- okienko
  - plamek, 657
  - właściwości, 215
- okna
  - przywracanie, 219
  - macierzyste, 201
  - sprawdzanie właściwości, 218
  - wieloplatformowe, 201
  - zapisywanie stanu, 219
- określanie pozycji
  - trójwymiarowych obiektów, 621
  - za pomocą kamery, 621
- opcje interpolacji, 278
- OpenCV, 25
- OpenCV 3.x, 802
- OpenCV.org, 808
- OpenGL, 220
- operacje
  - graniczne, 240
  - na macierzach, 109
  - na miejscu, 112
  - na obrazach, 109
  - na tablicach, 109
- operator Sobela, 251
- optymalna wartość progowa, 242
- oś czasu, 32
- otwieranie, 260
- pakiet sieci głębokich, 753
- pamięć
  - dynamiczna, 85
  - zmienna, 85
- Parallel, 41
- parametr rozkładania, 439
- parametry wewnętrzne
  - liniowe, 591
  - nieliniowe, 591
  - wyszukiwania FLANN, 518
- pary odległe, 500
- pasek
  - narzędzi, 212
  - stanu, 212
- PCA, principal component analysis, 167
- perceptron wielowarstwowy, 749
- pierwszy program, 44
- piramida Laplace'a, 282
- piramidy obrazów, 279
- plama, 479
- plamka, 656
- plansze kalibracji, 580
- pliki
  - danych, 179
  - dekompresowanie, 184
  - dołączane, 43
  - filmów, 179
  - kompresowanie, 183
  - obrazów, 179, 180
  - wideo, 47, 56, 184
- płaszczyzna, 93
  - akumulacyjna, 320
  - epipolarna, 629
  - rodzaje podziału, 813
  - rzutowa, 616
  - tworzenie podziałów, 816
- pnik decyzyjny, 734
- pobieranie biblioteki, 36
- pochodna Sobela, 251
- pochodne kierunkowe, 253
- podpora, 236
- podstawowe typy danych, 59, 60
- podział Delaunaya
  - identyfikowanie trójkąta granicznego, 822
  - identyfikowanie trójkąta ramowego, 822
  - orbitowanie, 818
  - punkty z krawędzi, 817
  - znajdowanie punktów, 818

podziały płaszczyzn, 813  
 pole wektorowe, 293  
 położenie kamer frontowo równoległe, 624  
 pomieszana macierz kowariancji, 118  
 porównywanie
 

- histogramów, 349
- kształtów, 396
- kwadratu różnicy, 363

 powłoki wypukłe, 388  
 poznawanie tła, 426  
 precyzja docelowa, 518  
 problem apertury, 455  
 procedury biblioteki ML, 705  
 prognozowanie, 713  
 progowanie z histerezą, 317  
 programowanie voodoo, 732  
 promień główny, 624  
 propagacja wsteczna, 749, 753  
 proste rozmazanie, 246  
 próg
 

- adaptacyjny, 243
- binarny, 244
- histogramu, 347
- wariancji, 440

 przekształcenia
 

- afiniczne, 285, 615
  - zbiorów gęstych, 285
  - zbiorów rzadkich, 288
- homograficzne, 587
- KLT, 168
- morfologiczne, 255
- obrazu, 277
- perspektywiczne, 289, 616
  - gęste, 289
  - rzadkie, 291
- rzutowe, 570

 przełączniki, 208, 210  
 przenośność, 40  
 przepływ optyczny, 446, 450, 462
 

- dwuwymiarowy, 455
- gęsty, 526
- Lucasa-Kanade'a, 453
- rzadki, 451

 przestrzeń
 

- kolorów
  - RGB, 422
  - YUV, 422
- nazw cv, 45

 przeszukiwanie obrazu, 778  
 przetrenowanie, 612  
 przyciski, 210, 216  
 przywoływane do istnienia, 715  
 punkt
 

- centralny, 236
- epipolarny, 628
- główny, 569, 624
- kluczowy, 445, 461
  - filtrowanie, 512
  - wykrywanie, 474
  - wyświetlanie, 519
- testowy, 501
- z krawędzi, 817
- załączkowy, 330

## Q

Qt, 222

## R

RANSAC, 634  
 Real AdaBoost, 733  
 regresja, 721
 

- wektorów nośnych, 761

 rektyfikacja, 602, 624, 665
 

- stereo, 639, 643
  - bez kalibracji, 644
  - z kalibracją, 646

 renowacja obrazów, 298  
 repozytorium
 

- kodu od społeczności, 39
- opencv\_contrib, 40, 825

 reprezentacja
 

- histogramów, 344
- skalowo-przestrzenna, 467

 reprojekcja, 599, 624  
 ręczna ekstrakcja, 239  
 RNG, random number generator, 173  
 ROC, receiver operating characteristic, 691  
 rodzaje
 

- konwersji, 124, 125, 126
- sumowania, 383

 rogi
 

- Harrisa, 475
- subpikselowe, 448
- szachownicy, 584

 rozbieżność, 626
 

- stała, 656



- rozkład
  - Gausa, 551
  - macierzy, 595
  - normalny, 551
  - uprzedni, 550
  - według wartości osobliwych, SVD, 127, 170
- rozmazanie proste, 246
- rozmażywanie, blurring, 245
- rozmiar
  - jądra, 498
  - plamy, 498
- rozpoznawanie
  - obiektów, 463
  - tła
    - porównanie metod, 436
- rozszerzenie algorytmu SVM, 761
- rozszerzony filtr Kalmana, 564
- rozwijanie, 734
- rozwińnięcie wielomianu Farneböcka, 527
- równania
  - algorytmu mean-shift, 538
  - Kalmana, 555
- równanie niezmienności jasności, 452
- różnicowanie
  - klatek, 408
  - tła, 424, 429
- Rprop, resilient back propagation, 753
- ruch, 541, 673
  - dynamiczny, 555
  - kontrolowany, 555
  - losowy, 555
- rysowanie, 157
  - konturów, 376
  - obrazu, 202
  - rogów szachownicy, 584
- rzadka stereometria, 461
- rzadki przepływ optyczny, 451
- rzadkie likwidowanie zniekształceń, 607
- rzut perspektywiczny, 616
- rzutnia, 568
- rzutowanie, 588, 614
  - nasyceniowe, 98
  - w przód, 278
  - wstecz, 358
  - wstecz wartości histogramu, 361
- schemat
  - blokowy, 31, 32
  - propagacji wstecznej, 752
- segmentacja, 329
  - mean-shift, 336
- segmentowanie lokalnych obszarów ruchu, 545
- selekcja indeksów, 518
- semiglobalne dopasowywanie bloków, 653
- SfM, Structure from Motion, 613, 673
- SGBM, semi-global block matching, 653
- siatka kół, 581, 585
- sieci
  - głębokie, 753
  - neuronowe
    - konwolucyjne, 680
- SIFT, Scale Invariant Feature Transform, 485
- SIMD, 41
- Simple Flow, 533
- skala obcinania wagi, 736
- skalary, 111
- skierowana odległość Hausdorffa, 399
- skok czasowy, 532
- słowa wizualne, 794
- słownik, 468, 794
  - punktów kluczowych, 471
- soczewki, 573
- splot, 236, 268, 311
- spójność, 660
  - przestrzenna, 452
- sprawdzian krzyżowy, 690
- stała
  - czułości, 496
  - rozpadu, 440
- stałe
  - klasy macierzowe, 60, 65
  - klasy wektorowe, 60, 67
- stałość czasowa, 451
- standardowa biblioteka szablonów, STL, 46, 60
- StatModel, 705
- statystyczne uczenie maszyn, 689
- stereoskopia, 463
- STL, Standard Template Library, 46, 60
- struktura, 425
  - cv::ml::TrainData, 707
  - modułu shape, 396
  - z ruchu, SfM, 673
- struktury szablonowe, 81

suma, 314  
 kwadratowa, 314, 316  
 nachylona, 314, 317  
 sumowanie, 383  
 SURF, Speeded-Up Robust Features, 490  
 suwak, 48, 208, 216  
 SVD, singular value decomposition, 170, 595  
 SVM z miękkim marginesem, 760  
 sygnatura, 181, 355  
 systemy dynamiczne, 554  
 szablon  
 complex<>, 68  
 cv::DataType<>, 72, 91  
 cv::Mat\_<>, 104  
 cv::MatIter<>, 93  
 cv::Matx<>, 65  
 cv::Ptr<>, 69  
 cv::SparseMat\_<>, 104  
 cv::Vec<>, 60  
 szablony  
 dopasowywanie, 362  
 o ustalonej długości, 82  
 ruchu, 541, 547  
 szachownica, 580  
 szacowanie  
 gęstości jąder, 537  
 ruchu, 548  
 szansa, 702  
 szereg Taylora, 574  
 szkolenie, 707  
 za pomocą klasy cv::BOWTrainer, 794  
 sztuczka jądra, 760  
 sztuczna  
 inteligencja, 809  
 sieć neuronowa, 749, 753  
 sztuczny neuron, 750  
 szukanie korespondencji, 624  
 szum  
 przetwarzania, 556  
 śrutowy, 247

## §

śledzenie, 445, 462, 525  
 promieni, ray tracing, 30  
 średnia, 414  
 środek rzutu, 616

## T

tablica, 97  
 dostęp do elementów, 90, 95  
 funkcja create(), 86  
 operacje, 109  
 tablice  
 błędów, 690–692  
 gęste, 85  
 n-wymiarowe, 85  
 rzadkie, 85, 100  
 dostęp do elementów, 101  
 skrótów, 100  
 wielokanałowe, 86  
 technika  
 jeden do wielu, 797  
 SfM, 613  
 tekst, 163, 217  
 teoria wzmacniania, 774  
 teselacja Woronoja, 813  
 test zgodności chi-kwadrat, 350  
 testowanie modeli, 421  
 testy geometryczne, 389  
 Top Hat, 265  
 transformacja, 52, 53  
 Hougha  
 kołowa, 323, 325  
 liniowa, 319  
 progresywna probabilistyczna, 323  
 standardowa, 321  
 wieloskalowa, 321  
 odległościowa, 327  
 prosta, 309  
 Rodriguesa, 572  
 transformacje  
 afiniczne, 283, 285  
 homogeniczne, 568  
 geometryczne, 283  
 perspektywiczne, 283, 289  
 transformator metodą cienkiej płytki, 397  
 translacja, 580  
 triangulacja, 624  
 Delaunaya, 813  
 trójwymiarowa reprojekcja, 671  
 twierdzenie Kotelnikowa-Shannona, 53  
 tworzenie  
 histogramu, 344  
 jąder, 270  
 obiektu cv::ml::TrainData, 707, 710

- podziałów Delaunaya, 816
- podziałów Woronoja, 816
- przycisków, 216
- tablicy, 86
- własnego jądra, 267
- wyrażeń algebraicznych, 97

typ cv::Exception, 72

typy danych, 59

- podstawowe, 61

typy zdarzeń myszy, 205

## U

uczenie

- częściowo nadzorowane, 682
- maszyn, ML, 685, 679, 689, 705
- nadzorowane, 681, 774
- nienadzorowane, 681
- odroczone, 681
- przez wzmacnianie, 681
- się nowych obiektów, 782

układ współrzędnych stereo, 627

uogólnione

- mnożenie macierzy, 129
- punkty kluczowe, 461

ustawianie pikseli, 54

usuwanie

- nieużytków, 69
- szumów, 300 *Patrz także* algorytm FNLM

uśrednianie tła, 409

utajona maszyna wektorów nośnych, 685

użycie

- drzew losowych, 740
- drzewa decyzyjnego, 725
- histogramów, 351

## W

Waldboost, 685

wariancja, 414, 417

- całkowita, 530

warstwa akceleracji sprzętowej, 16

wartości

- graniczne, 243
- odległe, 760

wartość FFAAD, 412

warunek

- epipolarny, 629
- brzegowy, 236

wektor

- nośny, 757, 790, 797
- obecności, 794
- przesunięcia, 578, 580
- rotacji, 593
- stanu, 549
- translacji, 613

wewnętrzna macierz aparatu, 591

wideo, 47

widmo, 311

widok z góry, 617

widzenie

- stereoskopowe, 613
- trójwymiarowe, 613

wielokąty, 157

wielomian Farnebacka, 527

wielowarstwowy perceptron, MLP, 749

wierzchołek, 818, 820

Windows Template Library, 229

własności

- kamery, 188
- rejestracji wideo, 189

właściwości okna, 218

worek punktów kluczowych, 793

Worek słów, BOW, 685, 793

współczynnik

- mieszania, 557
- próbki obrazu, 53
- tła, 438
- unikalności, 656
- zniekształceń, 613

współrzędne

- biegunowe, 293
- homogeniczne, 144, 570
- kartezjańskie, 293
- logarytmiczno-biegunowe, 294
- trójwymiarowe, 671

wstawka, 655

wsteczna propagacja, 751

wxWidgets, 225

wycinek pikseli, 405

wygładzanie, smoothing, 245

- obrazu, 52
- z zachowaniem krawędzi, 250

wyjątek, 72

wyjście, 111

wykrywanie

- obiektów, 771, 790
- klasa cv::dpm::DPMDetector, 791

wykrywanie  
  obiektów  
    metoda Latent SVM, 790  
    z podziałem na części, 791  
  punktów kluczowych, 474  
  twarzy, 779, 780  
wyodrębnianie tła, 403  
wyrażenia macierzowe, 97  
  operacje, 98  
wyrównywanie  
  histogramu, 303  
  kontrastu, 304  
wyświetlanie  
  obrazu, 44, 203  
  punktów kluczowych, 519  
  wyników, 519  
względne przyspieszenie, 33  
wzmocnianie  
  statystyczne, 733  
  w kaskadzie Haara, 775  
wzmocnienie  
  aktualizacyjne, 553  
  Kalmana, 557  
wzmocniona kaskada odrzucenia, 772  
wzory kalibracji, 581, 582, 829  
  losowy, 830  
  szachownica, 829  
  tablica ArUco, 831  
  tablica ChArUco, 831  
  znacznik ArUco, 832  
  znacznik ChArUco, 832  
wzór  
  ChArUco, 581  
  Harrisa, 475  
  łańcuchowy na prawdopodobieństwo, 717

## Y

YUV, 422

## Z

zakres  
  dostępności biblioteki, 41  
  plamki, 657  
zamykanie, 260  
zapisywanie  
  danych, 191  
  klatek, 191  
  obrazów, 181  
  wideo, 190  
zastosowania konturów, 382  
zdarzenia myszy, 205  
zmienna wartość graniczna, 243  
zmiennie, 687  
  luźne, 760  
  ukryte, 716  
znaczniki zdarzeń myszy, 206  
znajdowanie  
  konturów, 371, 374  
  obiektów pierwszego planu, 429  
  powłoki wypukłej, 388  
  punktów kluczowych, 477, 480, 493, 502, 505  
  rogów, 446, 583  
zniekształcenia, 602, 606  
  radialne, 573  
  soczewek, 573  
  tangensowe, 573, 575  
znormalizowana metoda  
  współczynnika korelacji, 365  
  korelacji krzyżowej, 364  
  porównywania kwadratu różnicy, 364  
znormalizowane momenty centralne, 393  
znormalizowany filtr prostokątny, 246  
Z-score, 700

# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW  
w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

## OpenCV: poznaj i stosuj algorytmy przetwarzania obrazów!

Komputerowe rozpoznawanie obrazów przechodzi dziś fazę burzliwego rozwoju. Przyczyniają się do tego ogromna popularność cyfrowych aparatów fotograficznych, wielka liczba grafik zgromadzonych w obszernej internetowej bazach danych, a przede wszystkim coraz doskonalsze algorytmy przetwarzania obrazu. W rozwijaniu tej technologii wielką rolę odegrała biblioteka OpenCV, usprawniając pracę setek tysięcy ludzi. OpenCV 3.x ułatwia efektywne rozwijanie projektów dzięki opartej na języku C++ spójnej architekturze, która doskonale działa na wielu platformach.

Ta książka, przeznaczona dla osób znających język C++, jest praktycznym wprowadzeniem do otwartej biblioteki OpenCV w wersji 3.x. Zawiera też podstawowe informacje na temat komputerowego rozpoznawania obrazu, co powinno ułatwić efektywne posługiwanie się tą biblioteką. Sama biblioteka OpenCV została przedstawiona w sposób umożliwiający bardzo szybkie rozpoczęcie pracy. Książka ułatwia naturalne zrozumienie działania algorytmów, dzięki czemu projektowanie i debugowanie aplikacji nie powinno sprawiać problemów – a to powoduje, że stanowi świetne przygotowanie do zgłębienia bardziej zaawansowanych zagadnień komputerowego rozpoznawania obrazu i uczenia maszynowego.

Dzięki książce opanujesz:

- przegląd biblioteki OpenCV i zawarte w niej funkcje
- pracę z plikami obrazów, filmów i danych oraz przekształcanie obrazów
- ważniejsze algorytmy do pracy na obrazach
- punkty kluczowe: wykrywanie i filtrowanie
- trójwymiarowe widzenie, ruch, określanie pozycji
- uczenie maszyn w OpenCV

**Adrian Kaehler** – jest naukowcem i założycielem start-upów. Zajmuje się uczeniem maszynowym, modelowaniem statystycznym i komputerowym rozpoznawaniem obrazu. Pracuje w Intel Corporation i w Laboratorium Sztucznej Inteligencji Uniwersytetu Stanforda. Współzakładał Silicon Valley Deep Learning Group.

**Gary Rost Bradski** – jest naukowcem i konsultantem. Zajmuje się robotyką, uczeniem maszynowym i komputerowym rozpoznawaniem obrazów. Pracuje w Laboratorium Sztucznej Inteligencji Uniwersytetu Stanforda. Współtworzył takie biblioteki jak Open Source Computer Vision Library, Machine Learning Library i Probabilistic Network Library (PNL).

**Helion**

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Helion SA  
ul. Kościuszki 1c, 44-100 Gliwice  
tel.: 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

Sprawdź najnowsze promocje:  
 ● <http://helion.pl/promocje>  
 Książki najchętniej czytane:  
 ● <http://helion.pl/bestsellery>  
 Zamów informacje o nowościach:  
 ● <http://helion.pl/nowości>

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-283-1656-0



9 788328 316560

cena: 149,00 zł