Uniwersytet im. A. Mickiewicza w Poznaniu

Wydział Matematyki i Informatyki

kierunek: Informatyka

Praca magisterska

Rozpoznawanie obiektów na podstawie bazy wiedzy.

Aplikacja wspierająca rozgrywanie gier miejskich na platformę Android.

Objects recognition based on the knowledge–base.

An Android application supporting location–based games.

Andrzej Wójtowicz

Promotor:

dr Krzysztof Dyczkowski

Poznań, 2012

(wersja poprawiona 2014/04/12)

Poznań, ...

Oświadczenie

Ja, niżej podpisany Andrzej Wójtowicz, student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt. „Rozpoznawanie obiektów na podstawie bazy wiedzy. Aplikacja wspierająca rozgrywanie gier miejskich na platformę Android." napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem tej rozprawy lub jej części od innych osób. Oświadczam również, że egzemplarz pracy dyplomowej w formie wydruku komputerowego jest zgodny z egzemplarzem pracy dyplomowej w formie elektronicznej. Jednocześnie przyjmuję do wiadomości, że gdyby powyższe oświadczenie okazało się nieprawdziwe, decyzja o wydaniu mi dyplomu zostanie cofnięta.

.....

Spis treści

Wstęp

Gry miejskie są dość młodą formą rozrywki w Polsce, czerpiącą kilka wzorców z tradycyjnych podchodów i gier typu LARP. Miejscem rozgrywki, czy też *planszą*, jest cała przestrzeń miejska. W jednej z klasycznych wersji uczestnicy spotykają się w określonym miejscu, gdzie od prowadzącego dostają listę zadań–zagadek. Podczas gry zawodnicy przemieszczają się po mieście, rozwiązując przy tym zadania pośrednie, będące podpowiedziami do kolejnych zagadek. Zwycięzcą jest osoba lub drużyna, która wykona wszystkie zadania i dotrze do miejsca końcowego – *ostatniego pola*.

Jedną z form upowszechnienia, zautomatyzowania i uatrakcyjnienia gier miejskich jest wykorzystanie smartfonów – urządzeń będących dziś zminiaturyzowanymi komputerami. Dzięki temu użytkownik może uzyskać więcej interesującej treści, np. w postaci filmu. Podejście to ma również pozytywny wymiar finansowy poprzez zmniejszenie liczby osób kontrolujących rozgrywkę.

Z roku na rok smartfony zajmują coraz większą część rynku telefonów. Szereg dostępnych w nich funkcjonalności takich jak np. odbiornik GPS, dostęp do internetu czy aparat cyfrowy, tysiące aplikacji na systemy mobilne oraz zrównoważona cena – wszystko to przekłada się na ich rosnącą popularność wśród zwykłych ludzi. Według [39] w trzecim kwartale 2011 r. smartfony stanowiły 51% wszystkich sprzedanych telefonów w Europie.

Telefon służy już nie tylko do dzwonienia i wysyłania wiadomości SMS, ale pełni również funkcję rozrywkową. Szybkie i wysokiej klasy podzespoły pozwalają m.in. na płynne korzystanie z aplikacji z grafiką trójwymiarową oraz umieszczanie w serwisach społecznościowych zdjęć zaraz po ich wykonaniu. Możliwości jakie niosą za sobą smartfony skłaniają do ich wykorzystania w kontekście gier miejskich.

Istniejące na rynku aplikacje (np. [41] lub [8]) wykorzystują przede wszystkim dane GPS do ustalenia pozycji użytkownika. Istnieją również aplikacje wykorzystujące tzw. kody QR – kwadratowe kody kreskowe – czego przykładem jest [24], gdzie jedna z zagadek była ukryta pod fotokodem.

Do niedawna moc obliczeniowa telefonów pozwalała tylko na prostą analizę obrazu. Internet mobilny w Polsce wciąż nie należy do najtańszych, zatem wysyłanie zdjęć z telefonu do zewnętrznego serwera w celu dokonania skomplikowanej analizy wiąże się ze sporymi kosztami po stronie użytkownika. Dopiero pojawienie się smartfonów z coraz szybszymi procesorami, w szczególności modeli wielordzeniowych, otwiera nowe możliwości rozwoju aplikacji do gier miejskich.

Cel i zakres pracy

Celem poniższej pracy jest zaprezentowanie metod poszerzających możliwości gier miejskich na smartfony o rozpoznawanie budynków oraz znajdowanie szczegółów architektonicznych. Do wykonania części projektowej został wykorzystany system mobilny Android z biblioteką OpenCV, natomiast głównym mechanizmem działania jest algorytm wykrywania lokalnych cech w obrazie *Scale–invariant feature transform* (SIFT).

Podstawą literatury są artykuły opisujące działanie algorytmu SIFT. Po-

zostałe pozycje dotyczą przede wszystkim geometrii epipolarnej oraz dopasowania punktów kluczowych w stereowizji.

Pierwszy rozdział skupia się na definicji widzenia komputerowego i przetwarzania obrazów. W kolejnych podrozdziałach zawarto przeglądowy opis możliwych operacji na obrazie, a ponadto skupiono się na jego cechach w kontekście późniejszych rozważań nt algorytmu SIFT. W następnej części zostały umieszczone szczegóły dotyczące wykorzystanych algorytmów. Kolejny rozdział dotyczy opisu projektu, jego architektury oraz implementacji. W ostatniej części pracy zostały zebrane wyniki eksperymentu oraz podsumowanie.

Rozdział 1

Przetwarzanie obrazów i widzenie komputerowe

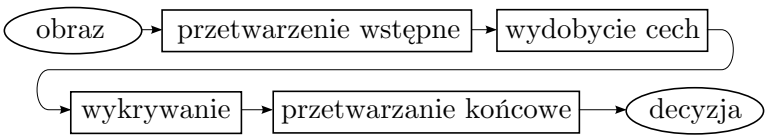
Przetwarzanie obrazu (ang. *image processing*) polega na poddaniu wejściowych danych pewnym operacjom i otrzymaniu z nich nowego obrazu. Widzenie komputerowe (ang. *computer vision*) jest jedną z dziedzin informatyki, która wykorzystuje przetwarzanie obrazów. Intuicję w rozumieniu tego zagadnienia dostarcza [37], definiując ją jako próbę opisu otaczającego nas świata poprzez jeden lub kilka obrazów oraz odtworzenie jego charakterystycznych cech, takich jak kształty, oświetlenie i kolory. Natomiast od strony technicznej należy rozumieć przez to zbiór operacji służących do przetwarzania i analizowania obrazów w celu pozyskania z nich usystematyzowanej wiedzy. W literaturze można spotkać się z terminem widzenia maszynowego (ang. *machine vision*), który oznacza to samo co widzenie komputerowe lub odnosi się do jego zastosowań w przemyśle.

Widzenie komputerowe bazuje na jednym dość kluczowym założeniu – wszystkie operacje, które jest w stanie wykonać człowiek, można przenieść

na grunt samodzielnych aplikacji. Do głównych dziedzin widzenia komputerowego należą:

- rozpoznawanie obiektów,
- analiza ruchu,
- rekonstrukcja obrazów.

Choć każde z powyższych zadań różni się od siebie znacząco, większość systemów opartych na algorytmach widzenia komputerowego posiada wspólny schemat postępowania (patrz rys. 1.1). W przetwarzaniu wstępnym z obrazu wejściowego najczęściej usuwa się szumy, poprawia kontrast, dokonuje konwersji do skali odcieni szarości itp. W kolejnym etapie zostają wyodrębnione pewne cechy obrazu, np. krawędzie lub wierzchołki. Dalsze dwie fazy dokonują pomniejszenia liczby obiektów w zbiorach cech tak, aby spełniały one pewne wymagania. Ostatni etap polega na podjęciu decyzji na podstawie zależności między otrzymanymi cechami (np. ich dopasowaniu).



Rys. 1.1. Ciąg operacji, którym poddawany jest obraz w celu podjęcia decyzji na podstawie jego zawartości.

Listę zastosowań widzenia komputerowego można znaleźć w [37] i [17]. Należą do nich m.in.:

- rozpoznawanie pisma odręcznego,
- rozpoznawanie numeru rejestracyjnego pojazdu,
- rozpoznawanie osób na podstawie odcisków palców,
- kontrola jakości na linii produkcyjnej,

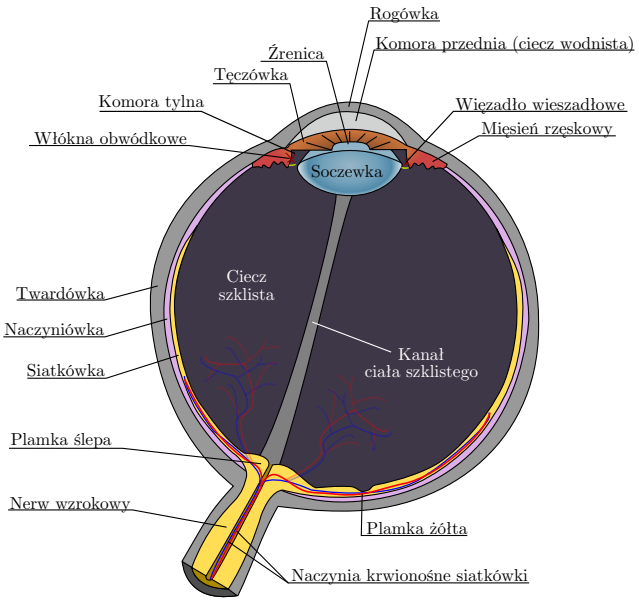
- rekonstrukcja trójwymiarowych modeli obiektów na podstawie ich zdjęć fotogrametrycznych,
- wspomaganie sędziowania podczas wydarzeń sportowych,
- tworzenie trójwymiarowych obrazów panoramicznych,
- śledzenie i rozpoznawanie ludzi.

Widzenie komputerowe jest ambitnym zagadnieniem, które pozwala praktycznie wykorzystać matematyczne modele i teorie. Jego prospołeczny wymiar – rozwój medycyny, zwiększenie bezpieczeństwa w przestrzeni publicznej, wykrywanie przestojów w ruchu drogowym – jest dodatkową zachętą do zainteresowania się tym tematem.

1.1. Percepcja koloru

Dzięki zmysłowi wzroku człowiek może postrzegać otaczający go świat: jego barwy, kształty i ruchy. Schemat budowy oka przedstawia rys. 1.2. Upraszczając, obraz powstaje poprzez przejście promieni światła przez soczewkę i ich rzut na siatkówkę. Powstały w ten sposób obraz jest odwrócony, jednak mózg potrafi przetworzyć informację tak, aby świat nie był widziany do góry nogami. Powierzchnia siatkówki pokryta jest światłoczułymi receptorami:

- **czopkami** – jest ich średnio 4.5 mln, głównie w centrum siatkówki; odpowiadają za postrzeganie barw; występują ich trzy rodzaje, w zależności od rodzaju światła na jakie reagują: czerwone, zielone i niebieskie; najwięcej jest zielonych, następnie czerwonych, a najmniej niebieskich; ich przykładowy rozkład na siatkówce przedstawia rys. 1.3,
- **pręcikami** – jest ich średnio 90 mln, głównie na obrzeżach siatkówki; odpowiadają za postrzeganie jasności.



Rys. 1.2. Schemat budowy ludzkiego oka. Źródło: [42]

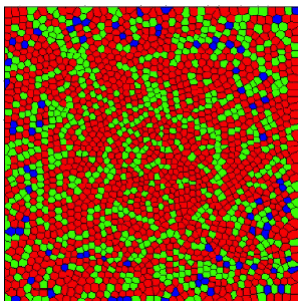
1.2. Przestrzenie barw

Kolorowe obrazy posiadają swoją reprezentację w urządzeniach cyfrowych w postaci pikseli. Barwa piksela jest opisana poprzez wektor w pewnym układzie. Wektory te składają się na przestrzeń barw danego układu.

Różne modele mają swoje konkretne zastosowania, w których sprawdzają się najlepiej: konstrukcja cyfrowych wyświetlaczy, poligrafia, przetwarzanie obrazów wideo itp.

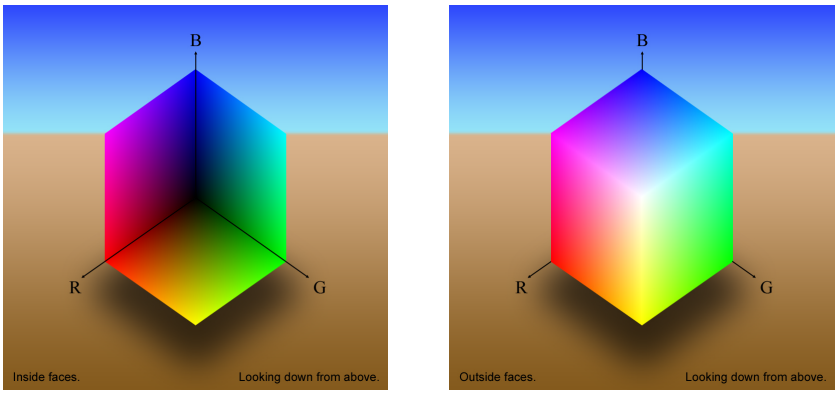
1.2.1. Model RGB

Jest to układ addytywny, najczęściej spotykany w urządzeniach cyfrowych. Każdy kolor reprezentuje wektor trzech liczb, w którym składowe (kanały) odpowiadają za udział koloru czerwonego (***r****ed*), zielonego (***g****reen*) i niebie-



Rys. 1.3. Rozkład czopków na siatkówce oka. Tylko niecałe 12% powierzchni pokrywają niebieskie czopki. Źródło: [36]

skiego (***b**lue*) w barwie wyjściowej. Wektor $[0, 0, 0]^T$ odpowiada barwie czarnej, a $[1, 1, 1]^T$ białej. Wizualizację tej przestrzeni prezentuje rys. 1.4.



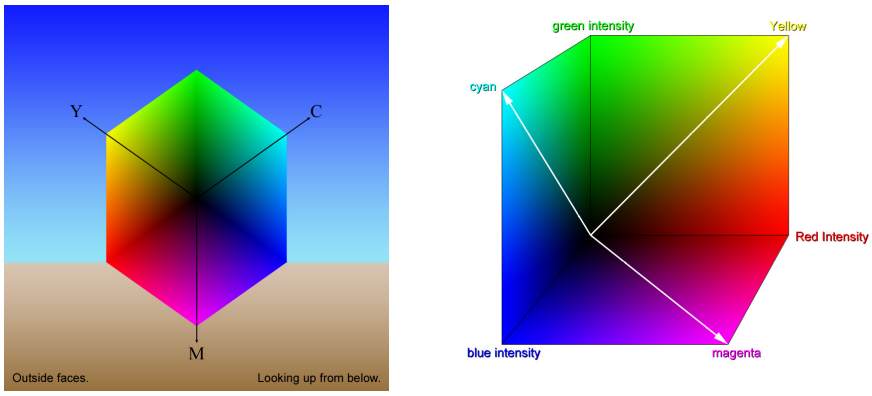
(a) **(b)**

Rys. 1.4. Model RGB tworzy sześcian. (a) Z tej perspektywy w jego wnętrzu widać umiejscowienie barwy czarnej, (b) z kolei na zewnątrz znajduje się barwa biała. Źródło: [26]

1.2.2. Model CMY(K)

Układ CMY jest modelem substraktywnym, wykorzystywanym w technikach drukarskich. Za barwę wyjściową odpowiadają trzy kolory: cyjan (***c**yan*), magenta (***m**agenta*) i żółty (***y**ellow*). Często wyodrębnia się dodatkowy kolor,

czarny (*blac**k*** lub ***k**ey color*), ponieważ wymieszanie trzech podstawowych nie zawsze daje idealną czerń. Wizualizację przestrzeni CMY prezentuje rys. 1.5.



(a) **(b)**

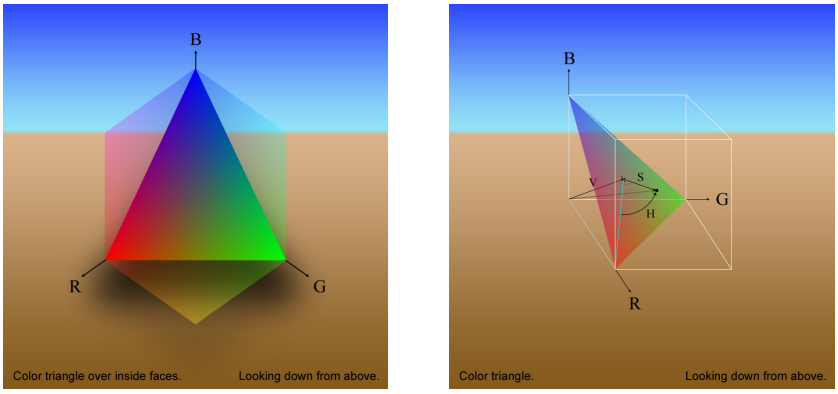
Rys. 1.5. (a) Model CMY, podobnie jak RGB, jest oparty na sześcianie, (b) jednak posiada inaczej zdefiniowany układ współrzędnych. Źródło: [26]

1.2.3. Model HSV

W porównaniu z RGB i CMY, model HSV nie zakłada przedstawienia kolorów jako mieszaniny trzech głównych. W tym przypadku składowymi są: barwa (***h**ue*), nasycenie (***s**aturation*) oraz wartość (***v**alue*). Taka reprezentacja pozwala na sprawniejsze operowanie obrazem w poszukiwaniu konkretnych kolorów – zmniejsza się liczba kanałów potrzebnych do analizy. Konstrukcję układu przedstawia rys. 1.6.

1.3. Operatory punktowe

Operacje dokonywane na pojedynczych pikselach obrazu bez uwzględniania ich sąsiedztwa należą do zbioru operatorów bezkontekstowych. Ich przegląd-



(a) **(b)**

Rys. 1.6. Rozmieszczenie modelu HSV względem sześcianu RGB. W ogólnym przypadku przestrzeń ta jest rozpatrywana jako stożek z kołem barw jako podstawą, jednak tutaj jest to trójkąt. Źródło: [26]

dowy spis można znaleźć w [21].

Przyjmijmy, że piksele obrazu I umieszczone są w macierzy $m \times n$, a każdy piksel przyjmuje wartości całkowite z przedziału $[0, 255]$. Bez utraty ogólności możemy rozpatrywać jednokanałowy obraz (w skali odcieni szarości). Ponadto niech I_o oznacza obraz wyjściowy po zastosowaniu operatora.

1.3.1. Operacje liniowe

Najprostsze operacje arytmetyczne na obrazach mają postać:

$$I_o(x, y) = aI(x, y) + b$$

gdzie a, b są parametrami. Aby zwiększyć lub zmniejszyć jasność całego obrazu należy dodać lub odjąć pewną stałą b od każdego piksela:

$$I_o(x, y) = I(x, y) \pm b$$

W przypadku przekroczenia dostępnego przedziału najczęściej stosuje się obcięcie lub normalizację jasności.

Aby uzyskać negatyw obrazu należy od maksymalnej możliwej wartości piksela odjąć wartość piksela w danym punkcie:

$$I_o(x, y) = 255 - I(x, y)$$

Mnożenie lub dzielenie obrazu przez stałą wpływa na jego kontrast:

$$I_o(x, y) = aI(x, y)$$

Obrazy można też dodawać lub odejmować od siebie:

$$I_o(x, y) = I_1(x, y) \pm I_2(x, y)$$

Powyższa operacja ma zastosowanie przy detekcji ruchu, gdzie od klatki bazowej odejmowane są kolejne klatki przetwarzanego materiału wideo. W przypadku dodawania można rozważyć ważone mieszanie obrazów:

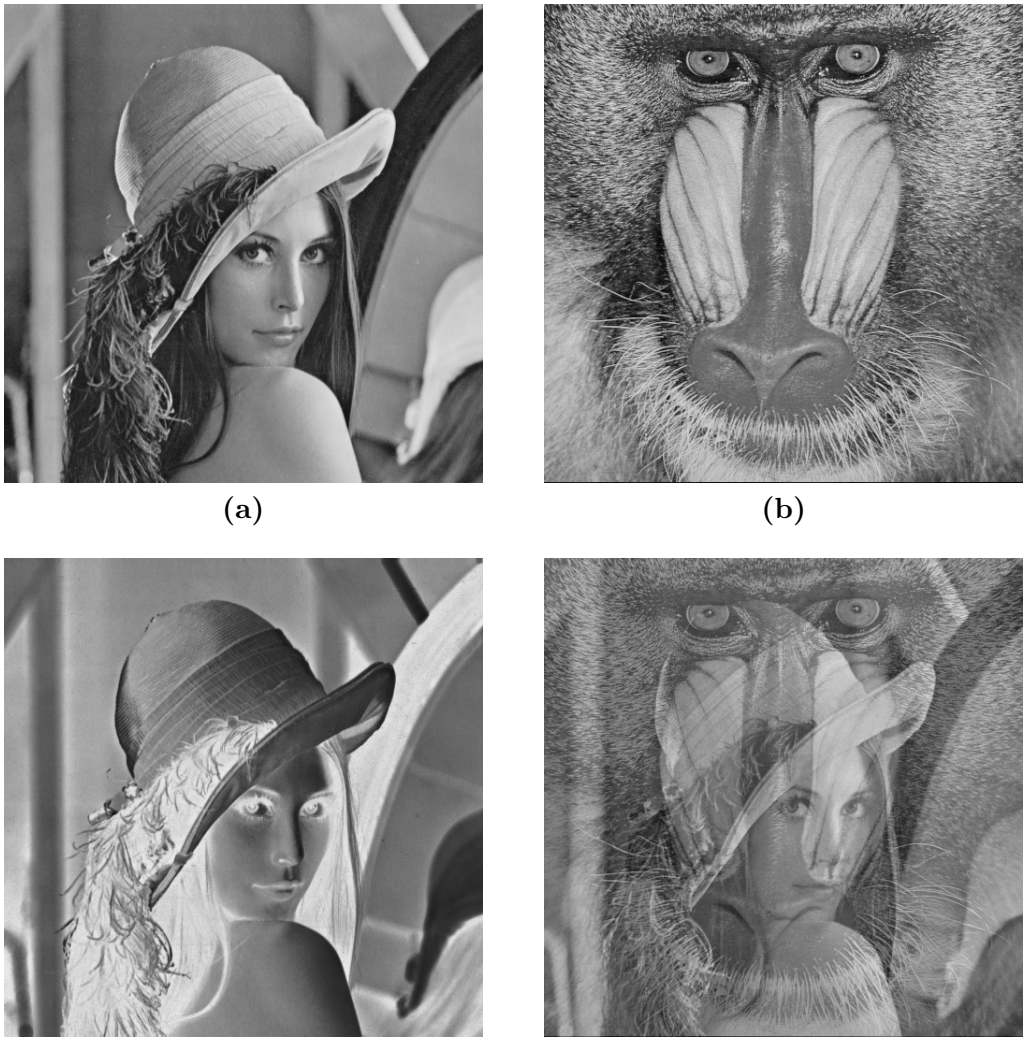
$$I_o(x, y) = \alpha I_1(x, y) + (1 - \alpha) I_2(x, y)$$

gdzie $\alpha \in [0, 1]$. Otrzymujemy w ten sposób tonalne przejście jednego obrazu w drugi. Ujmując rzecz ogólniej, mieszanie n obrazów ma postać:

$$I_o(x, y) = \alpha_1 I_1(x, y) + \alpha_2 I_2(x, y) + \cdots + \alpha_n I_n(x, y)$$

gdzie $\sum_{i=1}^{n} \alpha_i = 1$. Taka operacja może być przydatna do wykrywania ruchu gdy otrzymywany obraz z kamery jest zaszumiony.

Przykład realizacji wybranych operacji liniowych przedstawia rys. 1.7



(a) **(b)**

(c) **(d)**

Rys. 1.7. Wybrane punktowe operacje liniowe. (a) (b) Obrazy wejściowe, (c) negatyw pierwszego obrazu wejściowego oraz (d) mieszanie obrazów wejściowych z $\alpha = 0.5$

1.3.2. Operacje nieliniowe

W przypadku operacji nieliniowych, funkcje zmieniające poziom jasności obrazu mają postać:

$$I_o(x, y) = \phi(I(x, y))$$

gdzie ϕ jest funkcją transformacji. Najczęściej spotykaną operacją jest potęgowanie lub pierwiastkowanie obrazu:

$$I_o(x, y) = I(x, y)^\gamma$$

gdzie $\gamma > 0$. Główne zastosowanie znajduje to w tzw. korekcji gamma, usuwającej zniekształcenia wprowadzone przez urządzenia cyfrowe. Dzięki tej operacji zmniejszany jest nadmierny kontrast obrazu.

Przykład realizacji korekcji gamma przedstawia rys. 1.8

(a) **(b)** **(c)**

Rys. 1.8. Korekcja gamma (b) z wartością $\gamma = 0.5$ oraz (c) $\gamma = 2$

1.4. Filtry liniowe

W poniższym podrozdziale zostaną przedstawione wybrane filtry liniowe. Koncepcję ich użycia w przetwarzaniu obrazów można znaleźć w [2] i [21].

Słowo *liniowe* w przypadku filtrów oznacza liniową kombinację sąsiadów aktualnie przetwarzanego piksela.

Filtr nazywamy liniowym wtedy, gdy funkcja ϕ, którą realizuje, spełnia warunki:

1. $\phi(I_1 + I_2) = \phi(I_1) + \phi(I_2)$,
2. $\phi(\alpha I_1) = \alpha\phi(I_1)$,

gdzie I_1, I_2 są obrazami podlegającymi filtracji oraz $\alpha > 0$.

1.4.1. Splot dyskretny

W odróżnieniu od operatorów punktowych, filtry cyfrowe zakładają wykonanie operacji kontekstowych. Tym razem, chcąc obliczyć wartość jednego konkretnego piksela, należy uwzględnić jego otoczenie, stąd też wykorzystano splot.

Dyskretny dwuwymiarowy splot funkcji f z g w przetwarzaniu obrazów ma postać:

$$f(x, y) * g(x, y) = \sum_i \sum_j f(x - i, y - j) \cdot g(i, j)$$

Splot intuicyjnie można sobie wyobrazić jako rozproszenie funkcji f pod wpływem funkcji g – stąd też funkcję g często nazywa się funkcją punktowego rozproszenia (ang. *point spread function*).

W przetwarzaniu obrazów funkcja f jest rozpatrywanym obrazem, natomiast g maską (oknem), czyli macierzą o zadanych współczynnikach. Stosując filtr na obrazie, dla każdego piksela nakładamy na niego maskę – określa ona których sąsiadów należy uwzględnić i jakie nadać im wagi. Po dodaniu wszystkich elementów otrzymujemy nową wartość piksela.

Jeżeli w jest sumą elementów maski i $w > 1$, to w wyniku możemy dostać wartość piksela przekraczającą dopuszczalny przedział. Stosuje się wtedy normalizację, mnożąc wynik przez $1/w$, gdzie w jest tzw. współczynnikiem normalizacji lub wagą maski filtru. W przypadku ujemnych elementów maski nowa wartość piksela może być również ujemna. Rozwiązaniem tego problemu jest zastąpienie nowej wartości zerem, skorzystanie z wartości bezwzględnej lub przesunięcie przedziału jasności.

Podczas stosowania splotu na krawędziach obrazu pojawia się problem brakujących sąsiadów. W konkretnych zastosowaniach wybiera się różne podejścia, jednak do najpopularniejszych należy traktowanie obrazu jako torus, powielenie pikseli przy krawędziach obrazu albo pominięcie operacji splotu w problematycznych regionach.

Własności splotu pod kątem przetwarzania obrazów

1. $f(t) * g(t) = g(t) * f(t)$ (przemienność – kolejność, w której się splata, jest obojętna),
2. $f(t) * [g(t) + h(t)] = f(t) * g(t) + f(t) * h(t)$ (rozdzielność względem dodawania – możliwość zastąpienia filtracji obrazu złożeniem filtracji jednowymiarowych),
3. $[f(t) * g(t)] * h(t) = f(t) * [g(t) * h(t)]$ (łączność – możliwość rozdzielenia filtrowania dużym oknem na kolejne filtrowania za pomocą małych okien).

1.4.2. Filtry dolnoprzepustowe

Filtry tłumiące szczegóły o dużej częstotliwości nazywa się filtrami dolnoprzepustowymi. Ich najczęstszym zastosowaniem jest wygładzanie (usuwanie

szumów).

Wygładzanie

Najprostsza maska uśredniająca ma postać:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

($w = 9$). Usuwa ona drobne zawirowania krawędzi oraz efekty falowania jasności na obiektach i na tle. Jej zasadniczą wadą jest rozmywanie konturów obiektów i pogorszenie wyrazistości kształtów. Jej wielokrotne użycie powoduje znikanie elementów o małych rozmiarach.

Zmieniając wartość środkowego elementu maski zmniejsza się ww. negatywne skutki:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & a & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

($w = 8 + a, \quad a \in \{2, 4, 12\}$).

Układ wag maski zbliżony do rozkładu Gaussa wpływa na uśrednianie sąsiadujących pikseli oraz stopień rozmycia:

$$\begin{bmatrix} 1 & b & 1 \\ b & b^2 & b \\ 1 & b & 1 \end{bmatrix}$$

($w = (b + 2)^2, \quad a \in \{2, 3, 4\}$). Przykład realizacji tej maski prezentuje rys. 1.9.

Operator Gaussa

Dokładniejszym odzwierciedleniem tonalnego wpływu sąsiadujących pikseli jest zastosowanie operatora Gaussa. Elementy maski są wypełniane wartościami dwuwymiarowej funkcji Gaussa:

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

Rozmiar maski często jest większy niż 3×3, jednak należy pamiętać o większym narzucie obliczeniowym związanym z obliczeniem splotu. Przykładowa maska dla $\sigma = 1$:

$$\begin{bmatrix} 0.0585 & 0.0965 & 0.0585 \\ 0.0965 & 0.1591 & 0.0965 \\ 0.0585 & 0.0965 & 0.0585 \end{bmatrix}$$

W przypadku większych masek mogą wystąpić problemy związane z reprezentacją zmiennoprzecinkową, dlatego spotyka się też wersje z maską o wartościach całkowitych wraz ze współczynnikiem normalizującym $1/w$.

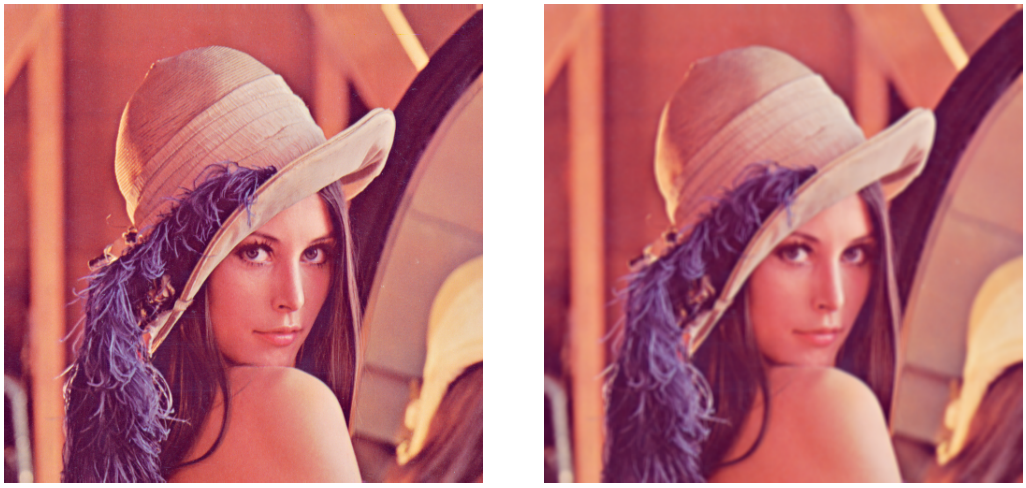
Działanie operatora Gaussa przedstawia rys. 1.9.

1.4.3. Filtry górnoprzepustowe

Filtry wzmacniające szczegóły o dużej częstotliwości nazywa się filtrami górnoprzepustowymi. Swoje zastosowanie znajdują w poprawie ostrości obrazu, jednak w konsekwencji wzmacniają też szumy.

Wykrywanie krawędzi

Krawędź jest granicą między dwoma regionami o różnych odcieniach jasności, zatem przejścia między regionami mogą być określone na podstawie różnic



(a) (b)

Rys. 1.9. Rozmycie Gaussa. (a) Obraz oryginalny i (b) rozmyty maską 5×5 z parametrem $\sigma = 3$

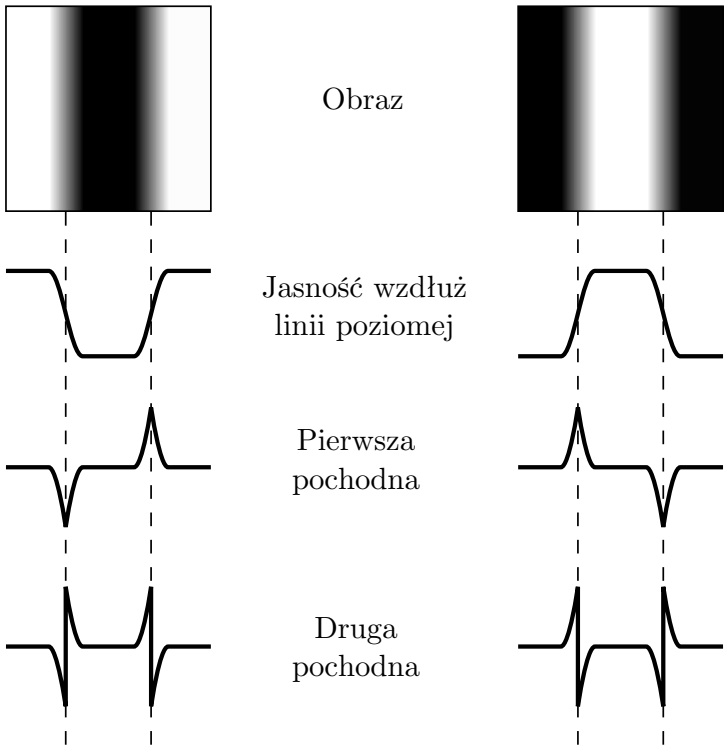
odcieni szarości pikseli z różnych regionów. Szczegóły prezentuje rys. 1.10.

Na podstawie pierwszej i drugiej pochodnej często oblicza się mapę krawędzi. Z racji, że metody różniczkowe są bardzo czułe na szum, wstępnie przeprowadza się oczyszczanie obrazu filtrem dolnoprzepustowym.

Gwałtowna zmiana funkcji jasności wyznacza krawędź, zatem w tym przypadku stosuje się metody gradientowe. Duża wartość gradientu oznacza punkt na krawędzi. Aby usunąć zakłócenia, zwykle przeprowadza się progowanie.

Aby obliczyć pochodną cząstkową obrazu, najczęściej aproksymuje się ją różnicą wartości pikseli [7]. Ponieważ:

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon, y) - f(x, y)}{\epsilon}$$



Rys. 1.10. Wykrywanie krawędzi poprzez badanie pochodnych funkcji jasności obrazu. Pierwsza pochodna może służyć do stwierdzenia obecności krawędzi, z kolei druga czy piksel leży po ciemnej czy jasnej stronie krawędzi. Źródło: [21]

pochodną cząstkową obrazu możemy przybliżyć:

$$\frac{\partial I}{\partial x} \approx I(x+1, y) - I(x-1, y)$$

co w ostateczności sprowadza się do splotu obrazu z maską:

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Operatory Robertsa [29], Prewitta i [27], Sobela [28] [3] aproksymują

pierwszą pochodną. Aby obliczyć kierunek i moc gradientu można spróbować wyznaczyć moc gradientu dla 8 kierunków i wybrać ten z największą wartością, jednak takie rozwiązanie jest nieefektywne i obarczone dużym narzutem obliczeniowym. Najczęściej, spośród dostępnych masek, wybiera się dwie dla kierunków x i y, a następnie na ich podstawie wyznacza się potrzebne dane. Przykład takiego rozwiązania znajduje się w rozdziale 1.6.4.

1. Operator **Robertsa** jest najprostszym operatorem wykrywającym krawędzie. Składowe gradientu w kierunkach x, y oraz skośnych mają następujące maski (pozostałe w przeciwną stronę mają zmienione znaki):

$$\begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

   Punkt zaczepienia znajduje się w pierwszym wierszu pierwszej kolumny maski. Z powodu minimalnego rozmiaru masek, operator ten jest bardzo czuły na szum i ma niski poziom sygnału reakcji na krawędzie obrazu. Kolejną wadą jest niejednoznaczność co do punktu zaczepienia składowych gradientu. Oznacza to, że gradient będzie przesunięty względem środka o pół piksela w kierunku x i y.
2. Operator **Prewitta** rozwiązuje niejednoznaczności w operatorach Robersta poprzez zastosowanie maski 3×3. Przykładowe trzy maski mają postać:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

3. Maski operatora **Sobela** posiadają współczynniki wzmacniające wpływ

otoczenia analizowanego piksela. Przykładowe trzy maski mają postać:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Wykryte krawędzie zwykle są grubsze, natomiast liczba błędnie wykrytych krawędzi jest mniejsza niż w przypadku operatorów Robertsa i Prewitta.

Przykład zastosowania powyższych operatorów przedstawia rys. 1.11.

Wyostrzanie

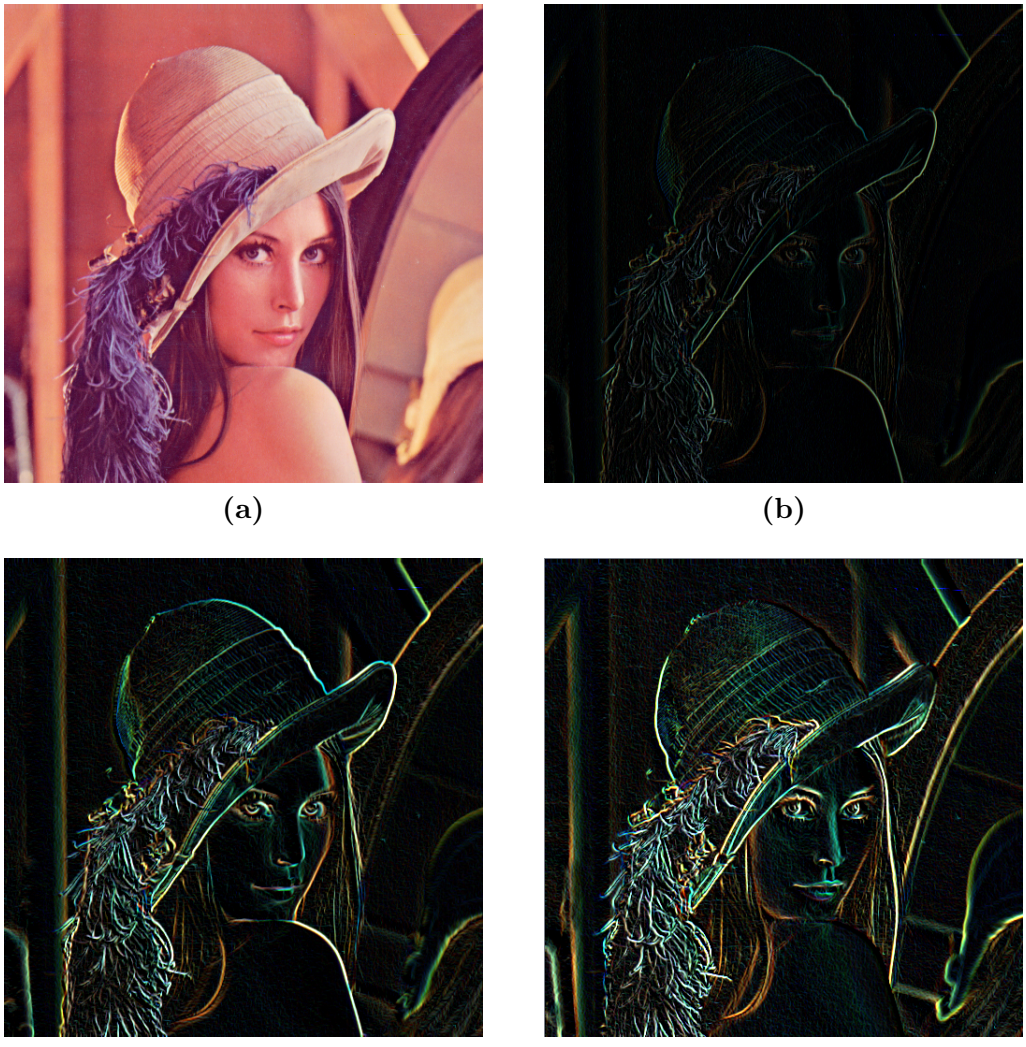
Podczas wykrywania i podkreślania w obrazie wszelkich krawędzi i konturów, pożądanym rozwiązaniem jest rezygnacja z kierunkowego działania. W takim wypadku często wystarczy zastosowanie laplasjanu, czyli sumy pochodnych cząstkowych drugiego rzędu [32]:

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \approx -[I(x+1,y)+I(x-1,y)+I(x,y+1)+I(x,y-1)]+4I(x,y)$$

Otrzymujemy w ten sposób podstawową maskę filtru **Laplace'a**:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

W najczęstszym użyciu jest maska będącą sumą pochodnych we wszystkich



(a) **(b)**

(c) **(d)**

Rys. 1.11. Wykrywanie krawędzi operatorami (b) Robertsa, (c) Prewitta i (d) Sobela. Kolor krawędzi wskazuje na kierunek gradientu w kole barw.

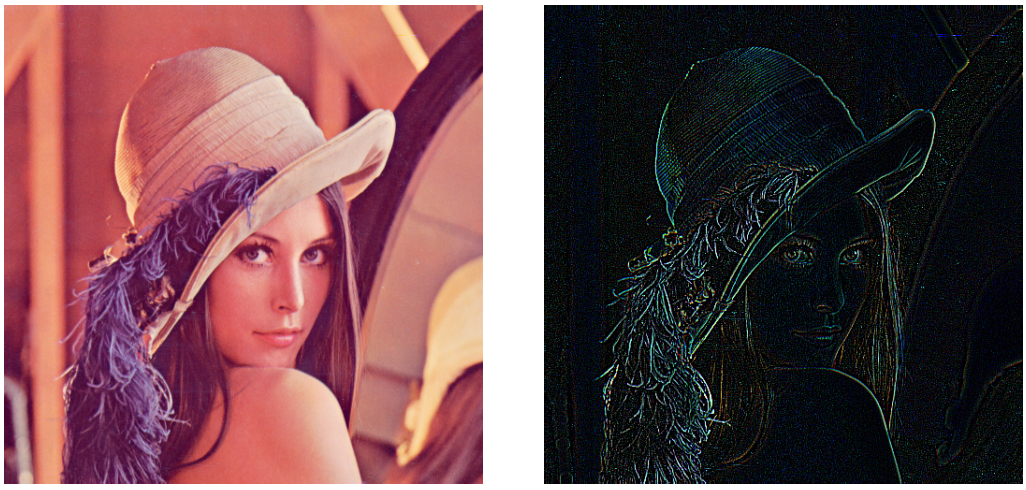
kierunkach:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Aby uzyskać wyostrzony obraz, wystarczy do niego dodać efekt zastosowania operatora Laplace'a, co przekłada się na nową maskę wyostrzającą otrzymaną w następujący sposób:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Przykład zastosowania operatora Laplace'a przedstawia rys. 1.12.



(a) **(b)**

Rys. 1.12. Wykrywanie krawędzi operatorem Laplace'a

1.5. Filtry nieliniowe

1.5.1. Filtr medianowy

Szumy i zakłócenia łatwo usunąć jeżeli wyraźnie różnią się od pozostałej części obrazu (np. szum typu *pieprz i sól*). Jeżeli wartości takich pikseli znacząco odstają od zbyt dużej liczby sąsiadów, to możemy je zastąpić medianą sąsiedztwa.

Filtr medianowy wykorzystuje najczęściej okno 3×3 lub 5-elementowy krzyż. Zaletą jest brak wprowadzania nowych wartości (nie ma potrzeby skalowania) oraz brak pogorszenia ostrości krawędzi obecnych na obrazie. Natomiast do wad tego rozwiązania należy narzut obliczeniowy związany z ciągłym sortowaniem wartości przetwarzanego okna oraz zaokrąglenie narożników.

Przykład zastosowania filtru medianowego przedstawia rys. 1.13.

(a) **(b)**

Rys. 1.13. (a) Na obraz wejściowy został nałożony szum typu *pieprz i sól*. (b) Filtr medianowy usunął prawie wszystkie zanieczyszczenia.

1.5.2. Filtr dwustronny

Rozwinięciem rozmycia gaussowskiego przy jednoczesnym zachowaniu ostrości krawędzi jest filtr dwustronny (ang. *bilateral filter*) [38]. Tradycyjne filtry uwzględniają *bliskość* pikseli w dziedzinie obrazu – innymi słowy, piksel znajduje się blisko drugiego jeżeli z nim sąsiaduje. Z kolei filtr dwustronny dodatkowo uwzględnia *bliskość* w sensie przeciwdziedziny obrazu – piksel jest blisko drugiego jeżeli mają podobne wartości, nawet jeżeli znajdują się na przeciwległych brzegach obrazu.

Dla piksela a w otoczeniu pikseli a_i otrzymujemy nową wartość w następujący sposób:

$$h(a) = k^{-1} \sum_i I(a_i) \cdot g(a_i) \cdot r(a_i)$$

gdzie

$$g(x, y; \sigma_d) = e^{-\frac{x^2+y^2}{2\sigma_d^2}} \quad \text{uwzględnia bliskość w dziedzinie,}$$

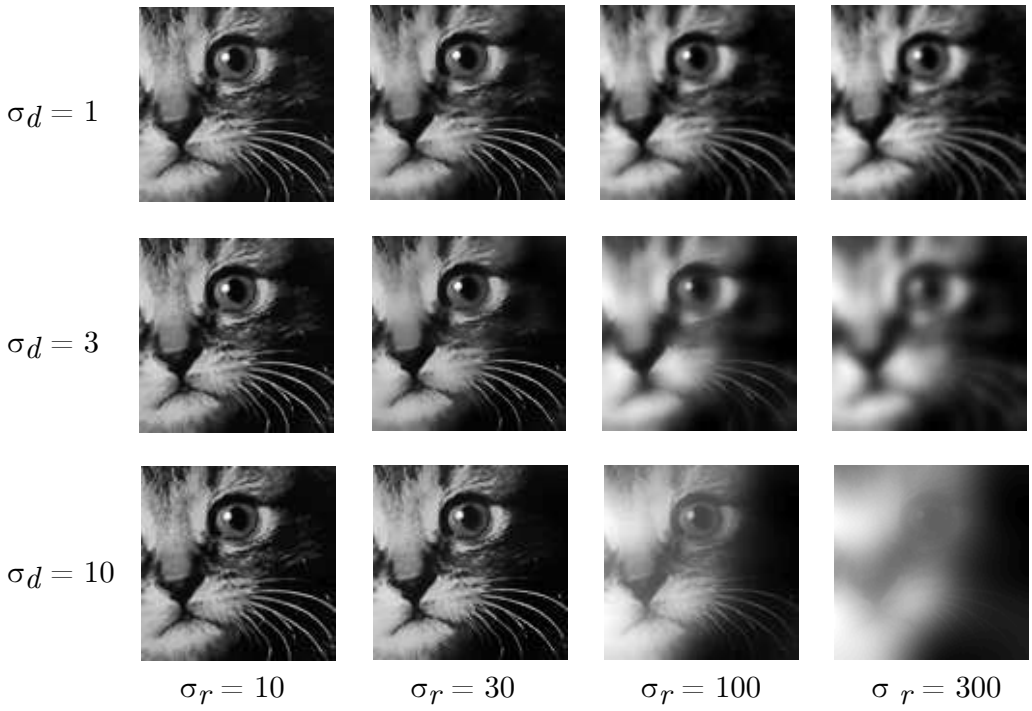
$$r(a_i; \sigma_r) = e^{-\frac{[I(a_i)-I(a)]^2}{2\sigma_r^2}} \quad \text{uwzględnia bliskość w przeciwdziedzinie,}$$

σ_d, σ_r to współczynniki skalujące, a k jest stałą normalizującą.

Przykład zastosowania filtru dwustronnego przedstawia rys. 1.14.

1.6. Metody wykrywania krawędzi

Wykrywanie krawędzi zwykłymi filtrami górnoprzepustowymi niestety często nie przynosi wystarczających efektów do dalszego przetwarzania. Wysoka podatność na szumy lub grube krawędzie mogą dyskwalifikować te rozwiązania w niektórych zastosowaniach. Istnieje zatem potrzeba wykrywania krawędzi metodami kombinowanymi.



Rys. 1.14. Wygładzanie filtrem dwustronnym dla różnych wartości σ_d i σ_r. Źródło: [38]

1.6.1. Laplasjan gaussowski

Operator Laplace'a jest podatny na szumy, stąd w [23] zaproponowano aby przed jego zastosowaniem wygładzić obraz operatorem Gaussa:

$$\nabla^2[G(x, y; \sigma) * I(x, y)]$$

Okazuje się, że całość operacji można przyspieszyć, najpierw licząc laplasjan funkcji Gaussa, a dopiero później zaaplikować go na obrazie:

$$\nabla^2[G(x, y; \sigma) * I(x, y)] = \nabla^2[G(x, y; \sigma)] * I(x, y) = \text{LoG} * I(x, y)$$

przy czym maskę wyznacza

$$\nabla^2 G(x, y; \sigma) = \left(\frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2} \right) G(x, y; \sigma)$$

Wizualizację laplasjanu gaussowskiego (ang. *Laplacian of Gaussian*/LoG, w literaturze zachodniej nazywany również operatorem Marra–Hildretha) oraz jego zastosowanie na obrazie przedstawia rys. 1.15.

Przykładowa aproksymowana maska LoG ma postać:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

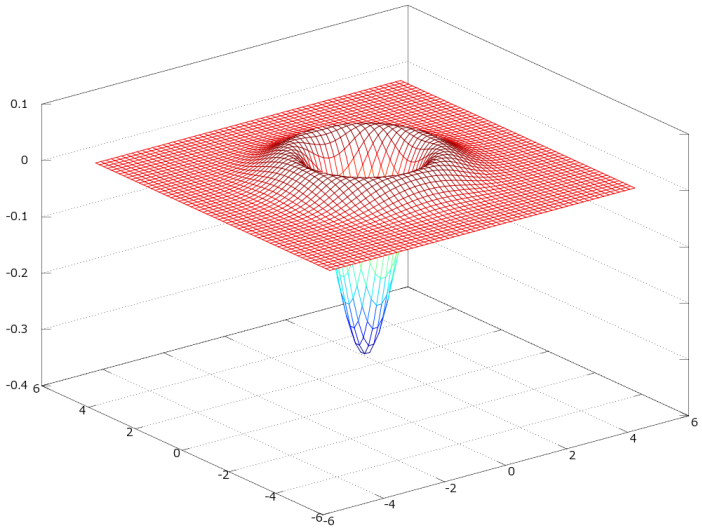
1.6.2. Różnica rozmyć gaussowskich

W [4] i [22] pokazano, że LoG można aproksymować różnicą rozmyć gaussowskich (ang. *Difference of Gaussians*/DoG):

$$\nabla^2 [G(x, y; \sigma)] * I(x, y) \approx [G(x, y; \sigma_1) - G(x, y; \sigma_2)] * I(x, y) = \mathrm{DoG} * I(x, y)$$

Okazuje się, że na podobnej zasadzie działa ludzki zmysł wzroku podczas rozpoznawania konturów. Poprzez taką operację, potencjalne szumy związane z liczeniem laplasjanu powinny zostać wytłumione. Najlepsze efekty w wykrywaniu krawędzi obiektów otrzymuje się gdy $\frac{\sigma_1}{\sigma_2} \approx 1.6$

Wizualizację aproksymacji DoG oraz efekt jego zastosowania na obrazie przedstawia rys. 1.16.

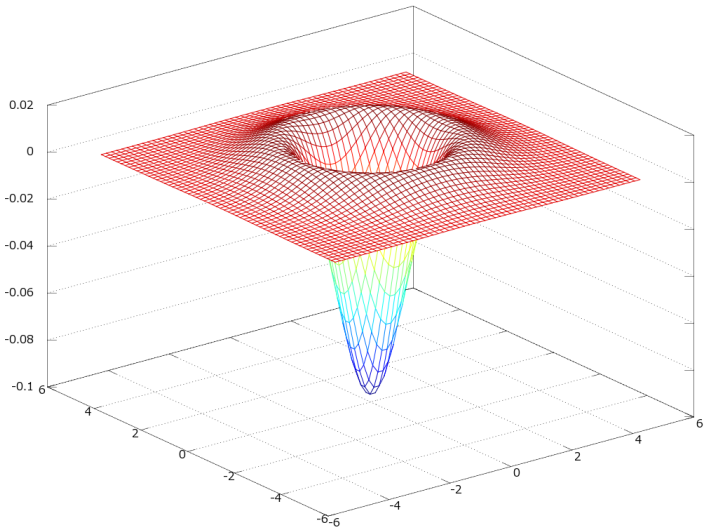


(a)

(b)

(c)

Rys. 1.15. (a) Wykres LoG dla $\sigma = 1$. (c) Wcześniejsze zastosowanie operatora Gaussa istotnie zmniejsza liczbę błędnie wykrytych krawędzi w porównaniu ze zwykłym laplasjanem.



(a)

(b)

(c)

Rys. 1.16. (a) Wykres DoG dla $\sigma_1 = 1.6$ i $\sigma_2 = 1$. (c) Końcowy efekt po zastosowaniu maski na obrazie.

1.6.3. Metoda przejść przez zero

W [22] zasugerowano również wykrywanie krawędzi poprzez badanie drugiej pochodnej, a dokładniej analizę przejść przez zero (ang. *zero–crossing*).

Na początku obliczany jest laplasjan (lub LoG) obrazu, a następnie dla każdego piksela rozważani są jego sąsiedzi (np. w oknie 3×3). Spośród nich wybierani są Ci, którzy mają wartość największą (v_{max}) i najmniejszą (v_{min}). Dysponując parametrem t (niewielka wartość niwelująca przypadkowe szumy) oraz v_0 (wartość zerowa, np. 0 lub 128, w zależności od tego jak liczony jest laplasjan) sprawdza się czy $v_{max} > v_0 + t$ oraz $v_{min} < v_0 - t$. Jeżeli tak, to dany piksel oznaczany jest jako krawędź.

Przykładową realizację powyższego algorytmu przedstawia rys. 1.17.

(a) **(b)**

Rys. 1.17. Wykrywanie krawędzi metodą przejść przez zero. (b) Efekt końcowy dla laplasjanu gaussowskiego – okno 3×3, $t = 5$

1.6.4. Algorytm Canny'ego

Operator zaproponowany w [1] jest prawdopodobnie najczęściej używanym narzędziem do wykrywania krawędzi. Przy jego realizacji postawiono trzy cele: redukcję czułości na szum, wykrywanie krawędzi tam gdzie naprawdę się znajdują oraz eliminację wielu krawędzi w tym samym punkcie przy zmianach jasności. Charakterystyczną cechą algorytmu są cienkie krawędzie otrzymywane na wyjściu.

Opis algorytm jest następujący:

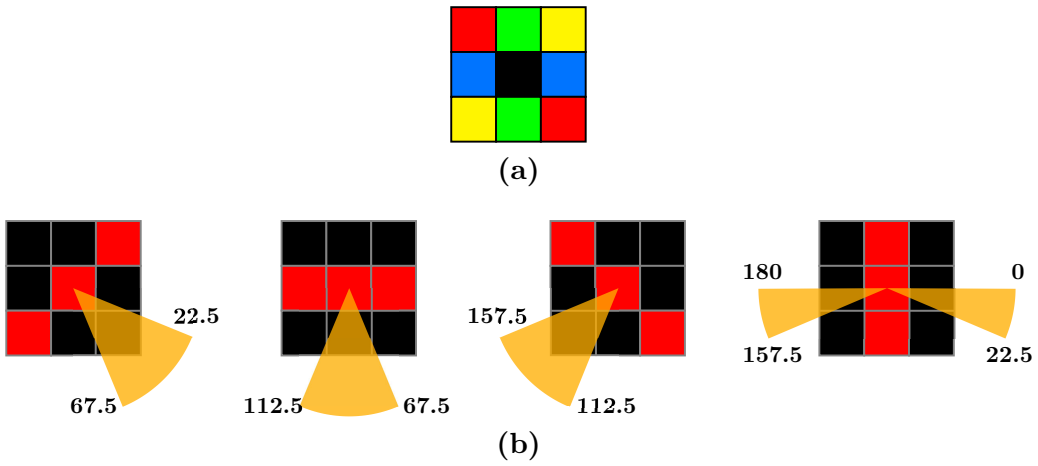
1. rozmyj obraz filtrem Gaussa,
2. oblicz pochodne kierunkowe G_x i G_y operatorem Sobela,
3. dla każdego piksela oblicz moc i kierunek gradientu:
   $m = \sqrt{G_x^2 + G_y^2}$,
   $\theta = \arctan \frac{G_y}{G_x}$,
4. przeprowadź tłumienie niemaksymalne – krawędź może być skierowana w 4 kierunkach (patrz rys. 1.18a), co więcej, zawsze jest prostopadła do kierunku gradientu; moc gradientu zmienia się nie wzdłuż krawędzi, ale w poprzek, dlatego dla każdego piksela sprawdź w którym kierunku skierowany jest jego gradient – w dalszym kroku wybierz sąsiadów będących na linii prostopadłej do kierunku gradientu (patrz rys. 1.18b); jeżeli moc gradientu jest większa od mocy gradientów odpowiadających mu sąsiadów oraz większa od zadanego górnego progu, to dany piksel dodaj do krawędzi,
5. przeprowadź progowanie z histerezą – dla każdego piksela oznaczonego w poprzednim kroku jako krawędź, sprawdź osobno jego dwóch sąsiadów wyznaczonych przez kierunek gradientu (innymi słowy, poruszaj się po krawędzi); jeżeli nowy kandydat ma:

(a) moc gradientu większą od dolnego progu,

(b) gradient skierowany w tę samą stronę co piksel, z którego przyszedłeś,

(c) moc gradientu większą od jego sąsiadów (z wyłączeniem sąsiada–piksela, z którego przyszedłeś) skierowanych w tym samym kierunku,

to dodaj piksel do krawędzi i dokonaj powyższych 3 sprawdzeń dla jego sąsiada wyznaczonego zgodnie z kierunkiem gradientu; w przypadku natrafienia na inną krawędź, zamknij aktualnie rozwijaną ścieżkę.



Rys. 1.18. (a) Możliwe kierunki krawędzi przechodzących przez centralny piksel. (b) W operatorze Canny'ego rozpatruje się sąsiadów piksela wyznaczonych odpowiednio przez kierunek gradientu. Źródło: [35]

W niektórych implementacjach powyższe kroki powtarza się kilka razy dla rosnących wartości σ filtru Gaussa, a następnie łączy powstałe w ten sposób mapy krawędzi.

Przykład zastosowania operatora Canny'ego przedstawia rys. 1.19.

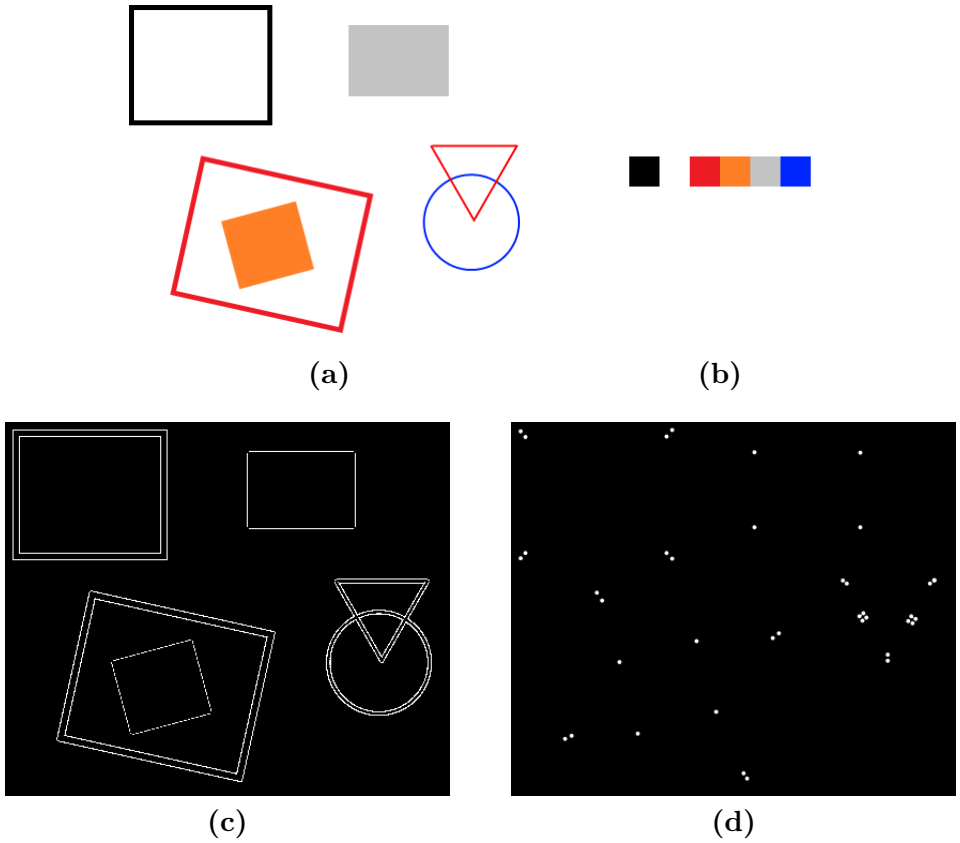
(a) (b)

Rys. 1.19. Wykrywanie krawędzi operatorem Canny'ego

1.7. Opis obrazu

Najczęściej przy przetwarzaniu zdjęć operuje się obrazem zapisanym w trójkanałowej przestrzeni barw RGB, gdzie każdy piksel na każdym kanale może przyjąć wartość całkowitą od 0 do 255. Taki format danych, poza możliwością wyświetlenia obrazu na ekranie, nie niesie za sobą uporządkowanych dodatkowych informacji. Człowiek posiada możliwość względnie szybkiego stwierdzenia jakie obiekty są przedstawione na obrazie, nie zastanawiając się nad niskopoziomowymi detalami – są one analizowane przez mózg podświadomie. Jednym ze sposobów aby komputer zaczął *rozumieć* obrazy jest przygotowanie zbiorów pewnych cech, od których będzie mógł rozpocząć dalszą analizę. Obraz może być opisany za pomocą np. krawędzi, wierzchołków i zakresu widocznych kolorów. Przykładowe wyodrębnienie powyższych cech prezentuje rys. 1.20.

W [21] przedstawiono również inne charakterystyki obrazu. W przypadku wyodrębnienia obiektów znajdujących się na obrazie, możemy rozpatrywać



Rys. 1.20. Obraz (a) i jego przykładowe wyodrębnione cechy: (b) występujące kolory, wykryte (c) krawędzie operatorem Laplace'a oraz (d) wierzchołki operatorem Harrisa.

ich liczebność lub pola powierzchni. Jednym z ciekawszych przykładów opisu obiektu jest tzw. liczba Eulera przedstawiająca jego spójność. Przy jej wykorzystaniu można rozróżniać litery alfabetu łacińskiego.

1.7.1. Wierzchołki i punkty kluczowe

Aby nie rozpatrywać obrazu jako całość, analizuje się jego jedną lub kilka wybranych cech. Naturalnym i intuicyjnym dla człowieka kandydatem są **wierzchołki** definiowane jako punkty izolowane, miejsca przecięć dwóch li-

nii, końce odcinków itp. Okazuje się jednak, że w ten sposób wyodrębnione ze współrzędnych konteksty albo nie niosą za sobą pożądanych informacji albo są słabo rozróżnialne. Stąd istnieje potrzeba ekstrakcji punktów metodami bardziej złożonymi, dlatego w ogólniejszym znaczeniu nie rozważa się wyodrębniania wierzchołków, ale **punktów kluczowych** (ang. *keypoints* lub *feature points*).

Klasycznym rozwiązaniem problemu znalezienia wierzchołków w obrazie jest zastosowanie operatora Harrisa [11], opierającego się na zmianie jasności w otoczeniu piksela. Wynik jego działania przedstawia rys. 1.20d. W przypadku śledzenia obiektów, gdzie czas na przetworzenie aktualnej klatki filmu jest niewielki, stosuje się metody mniej złożone obliczeniowo. Jednym z rozwiązań jest *Features from Accelerated Segment Test* (FAST) [30], niewymagający liczenia pochodnych tak jak w przypadku operatora Harrisa.

1.8. Biblioteka OpenCV

Jedną z najpopularniejszych bibliotek ukierunkowanych na widzenie komputerowe jest OpenCV (*Open Source Computer Vision Library*) [14]. Rozwijana początkowo przez programistów Intela, obecnie jest jednym z ważniejszych projektów ruchu wolnego oprogramowania. W swoich założeniach ma realizować uniwersytecką wizję zebrania najpopularniejszych i najwydajniejszych algorytmów związanych z widzeniem komputerowym.

OpenCV jest napisane głównie w języku C i C++, jednak posiada również interfejsy do innych języków, m.in. C#, Javy, Pythona i Matlaba. Biblioteka jest podzielona na kilkanaście modułów związanych m.in. z operacjami na macierzach, przetwarzaniem obrazów, rekonstrukcją obiektów trójwymiarowych, rozpoznawaniem obiektów i nauczaniem maszynowym. Od wersji 2.2

programiści kładą spory nacisk na wzrost wydajności poprzez dokonywanie obliczeń na kartach graficznych.

Powyższa biblioteka znajduje swoje zastosowanie nie tylko w widzeniu komputerowym, ale również dziedzinach ściśle z nią związanych, tj. sztucznej inteligencji i robotyce.

Rozdział 2

Rozpoznawanie obiektów

Zadanie rozpoznania obiektu na zdjęciu znajduje się w zasięgu umiejętności człowieka. Poprzez porównanie np. kształtów i charakterystycznych punktów może stwierdzić co przedstawia dany obraz, ewentualnie czy ma przed sobą szukany obiekt. Na gruncie widzenia komputerowego, elementy wyróżniające dany przedmiot od innych należy traktować jako zbiór abstrakcyjnych cech, mogących posłużyć do oceny podobieństwa lub klasyfikacji. Jedną z metod opisu i porównywania obiektów jest analiza punktów kluczowych, która zostanie opisana w poniższym rozdziale.

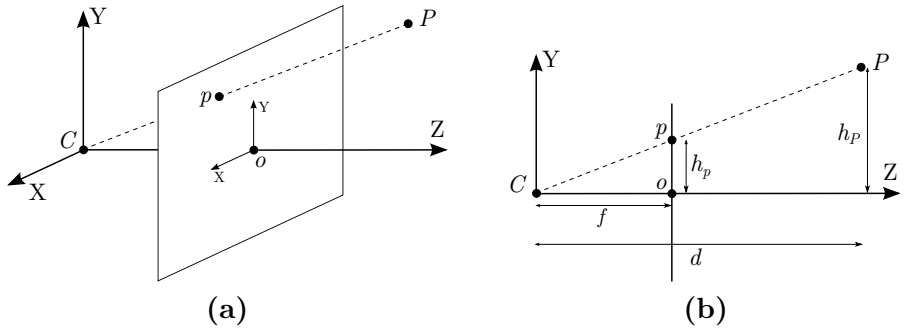
2.1. Stereowizja

Naturalną rzeczą w fotografii, w szczególności podróżniczej, jest ujęcie tego samego obiektu z różnych perspektyw. W większości przypadków człowiek może jednoznacznie stwierdzić czy dwa zdjęcia prezentują ten sam obiekt – wynika to z pewnego powiązania elementów będących na obu fotografiach. Jeżeli potrafimy znaleźć zależność dla punktów między dwoma obrazami, wynikającą ze zmiany perspektywy, wtedy mówimy o stereowizji.

Szczegóły dotyczące geometrii rzutowej w widzeniu komputerowym znajdują się w [13], natomiast w kolejnych podrozdziałach zostaną przedstawione terminy niezbędne do wyznaczenia macierzy opisującej geometrię między dwiema perspektywami.

2.1.1. Proces otrzymywania obrazu

Aby móc przetwarzać obraz z otaczającego nas świata, niezbędne jest posiadanie urządzenia, które potrafi uchwycić na matrycy dany moment. Matematyczny model kamery otworkowej (ang. *pinhole camera*) prezentuje rys. 2.1a. Każdy punkt P ma odpowiadający mu punkt p na płaszczyźnie obrazu poprzez rzut w kierunku środka kamery (ogniska) C. Punkt główny o wyznaczony jest przez prostą przechodzącą przez C ortogonalną do płaszczyzny obrazu. Rozmieszczenie punktów ze sceny na płaszczyźnie obrazu może być otrzymane z równości $h_p = f\frac{h_P}{d}$, gdzie f – długość ogniskowej, d – odległość obiektu od kamery, h_P – wysokość obiektu, h_p – wysokość rzutu obiektu na płaszczyźnie obrazu (rys. 2.1b).



Rys. 2.1. Model kamery otworkowej. Punkty obiektów znajdujących się na scenie rzutowane są na płaszczyznę obrazu poprzez rzut promienia świetlnego w stronę środka kamery. Źródło: [32]

2.1.2. Geometria epipolarna

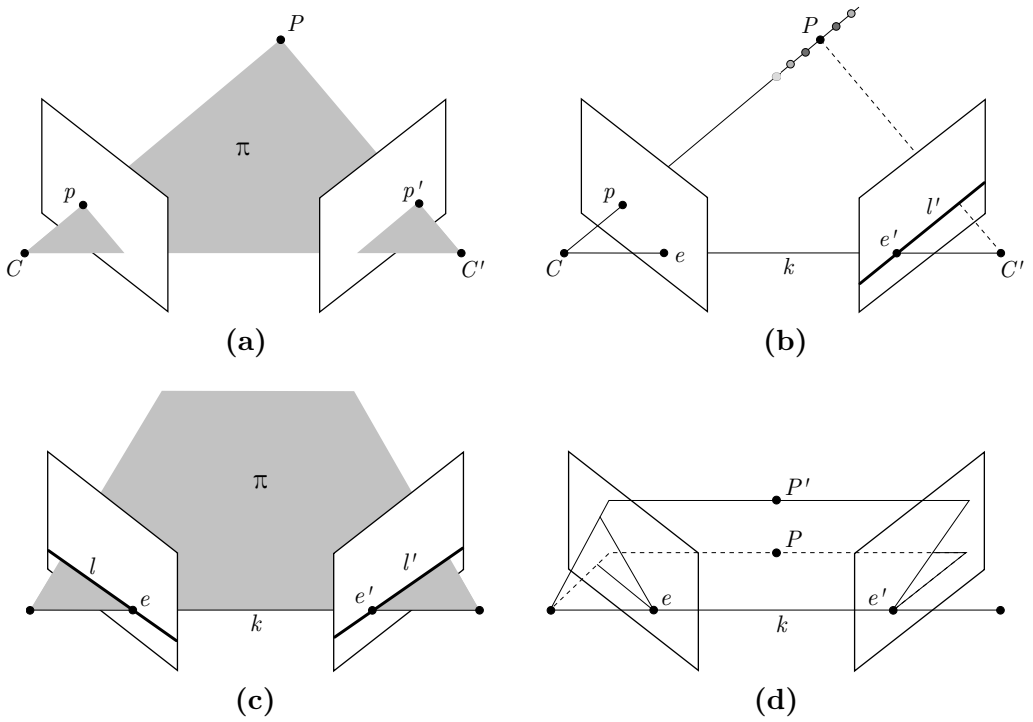
Przyjmijmy, że dwie różnie rozmieszczone kamery o środkach w punktach C i C' skierowane są w stronę tej samej sceny zawierającej punkt P (rys. 2.2a). Obrazy p i p' punktu P powstają z rzutu punktu P na płaszczyzny obrazów kamer wzdłuż odcinków $\overline{CP}$ i $\overline{C'P}$. Wszystkie punkty (C, C', P, p, p') znajdują się na tej samej płaszczyźnie π.

Załóżmy, że dany jest punkt p, nie jest znana pozycja P, a chcemy wiedzieć w jaki sposób uwarunkowany jest p' (rys. 2.2b). Punkty P, p i p' należą do tej samej płaszczyzny π, którą można wyznaczyć z prostej przechodzącej przez C i p oraz z prostej bazowej (ang. *baseline*) k, przechodzącej przez C i C'. W miejscu przecięcia płaszczyzny π z obrazem drugiej kamery powstaje tzw. **linia epipolarna** (ang. *epipolar line*) l', na której znajduje się punkt p'.

Dwa charakterystyczne punkty e i e', powstają w miejscu przecięcia prostej bazowej k z płaszczyznami obrazów kamer. Są to **punkty epipolarne** (ang. *epipoles*), będące jednocześnie obrazami środków kamer na przeciwnych płaszczyznach obrazów.

Każdą płaszczyznę π zawierającą linię bazową k nazywamy **płaszczyzną epipolarną** (ang. *epipolar plane*). Każda taka płaszczyzna przecina płaszczyzny obrazów kamer, tworząc odpowiadające jej linie epipolarne l i l' (rys. 2.2c).

Dla zadanej sceny i kamer, każda płaszczyzna epipolarna powstaje poprzez obrót wokół k (rys. 2.2d). Wszystkie linie epipolarne, wywodzące się z płaszczyzn epipolarnych, przecinają się w odpowiadających im punktach epipolarnych e i e'.



Rys. 2.2. Geometria epipolarna oraz zależności między punktami w stereowizji. Źródło: [13]

Macierz fundamentalna

Jak pokazano w [13], związek punktu p na obrazie z odpowiadającą jej linią epipolarną l' można wyrazić za pomocą równania:

$$l' = \mathbf{F}p$$

gdzie $\mathbf{F}$ jest macierzą 3×3, p jest podany we współrzędnych jednorodnych, a $l' \in \mathbb{R}^3$.

Innymi słowy, **macierz fundamentalna F** odwzorowuje punkty z jednej perspektywy na linie epipolarne w drugiej, przez co jest algebraiczną reprezentacją geometrii epipolarnej.

Jeżeli p i p' są odpowiadającymi sobie punktami z dwóch różnych perspektyw, to zachodzi tzw. **ograniczenie epipolarne** (ang. *epipolar constraint*):

$$p'^T \mathbf{F} p = 0$$

z którego można wyznaczyć elementy **F**. Jak pokazano w [5], do obliczenia tej macierzy potrzebne jest co najmniej 7 par (p, p'), jednak w tym przypadku pojawiają się trzy możliwe rozwiązania, z których każde musi zostać sprawdzone. Wiąże się to z dodatkowym narzutem obliczeniowym, przez co do efektywnych obliczeń stosuje się rozwiązanie z 8 parami (por. [12]).

2.2. Wykrywanie i opis punktów kluczowych

Obecnie w użyciu jest wiele algorytmów do wykrywania i opisu punktów kluczowych, np. *Speeded Up Robust Feature* (SURF), jednak prekursorem tych rozwiązań jest *Scale invariant feature transform* (SIFT) (por. [18] i [19]). W kolejnym podrozdziale zostaną opisane wszystkie kroki niezbędne do ich ekstrakcji i opisu. Zakładamy, że piksele w obrazie przyjmują wartości z przedziału $[0, 1]$. W celu zachowania spójności ze źródłami, w poniższych wzorach zostanie zachowana oryginalna notacja.

2.2.1. Algorytm SIFT

Konstrukcja przestrzeni skali

Znajdując się w małej lub dużej odległości od obiektu (np. budynku), postrzegamy go w różnym rozmiarze. Im bliżej, tym ludzkie oko może dostrzec więcej szczegółów – taka też jest natura cyfrowych urządzeń przechwytujących

obraz. Podczas procesu przetwarzania danych z nieznanej sceny nie jesteśmy w stanie określić skali znajdujących się na nich obiektów. Rozwiązaniem tego problemu jest jednoczesne rozpatrywanie wszystkich możliwości, czyli konstrukcja tzw. piramidy przestrzeni skali (ang. *scale space pyramid*). Operacja ta polega na zastosowaniu filtru Gaussa na obrazie i realizowana jest poprzez splot obrazu z operatorem Gaussa:

$$L(x, y, \sigma) = I(x, y) * G(x, y, \sigma)$$

gdzie:

- L – nowy rozmyty obraz,
- I – obraz wejściowy,
- G – operator Gaussa $G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$ ze współczynnikiem skali (rozmycia) σ.

Splot wykonywany jest po zmiennych x i y ze zdefiniowaną wcześniej σ.

Na przestrzeń skali składają się oktawy, które z kolei dzielą się na warstwy. W danej oktawie kolejne warstwy tworzy się poprzez zwiększanie współczynnika σ.

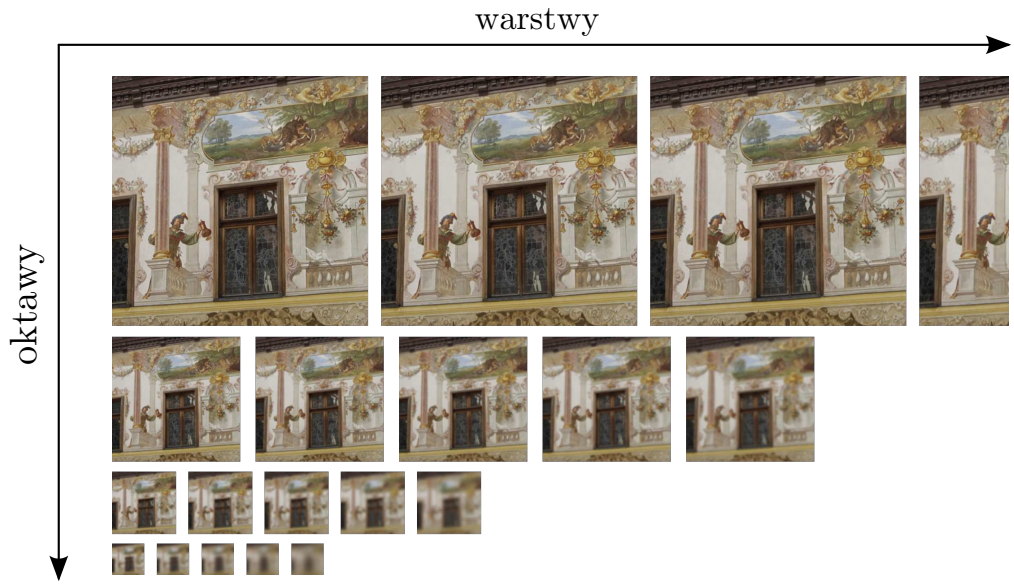
Uogólniając, w każdej oktawie, z wyłączeniem pierwszej, zmniejsza się rozmiar I o połowę. Dla i-tej oktawy ustala się rozmycie bazowe r_i, natomiast do obliczenia j-tej warstwy dokonuje się splotu I z operatorem Gaussa G, dla którego $\sigma_{i,j} = r_i k^{j-1}$, gdzie k jest stałą przyrostu, $i, j \geq 1$).

W praktyce, do poprawnego działania algorytmu, w każdej oktawie musi zostać wygenerowanych $3+s$ warstw ($s \geq 1$). Proponuje się przyjąć $k = 2^{1/s}$, po utworzeniu pierwszej oktawy za obraz wejściowy dla i-tej oktawy podstawić (odpowiednio zmniejszoną) tę warstwę j, dla której $\sigma_{i-1,j} = 2r_{i-1}$, a za r_i przyjąć $2r_{i-1}$. Należy jednak zwrócić uwagę, że nowy obraz wejściowy ma już

zaaplikowane rozmycie gaussowskie, co należy dodatkowo uwzględnić przy wyznaczaniu $\sigma_{i,j}$. Nie mniej, korzyścią z takiego rozwiązania jest zaoszczędzony czas na liczeniu splotu dla pierwszych warstw kolejnych oktaw.

Eksperymentalnie stwierdzono, że przy $r_0 = 1.6$ otrzymuje się wysoką powtarzalność punktów kluczowych z tego samego zdjęcia w różnych warunkach (np. obrót, zmiana kontrastu). Dodatkowo, podwojenie rozmiaru obrazu przed budową piramidy przestrzeni skali prowadzi do późniejszego wykrycia blisko czterokrotnie większej liczby punktów kluczowych. Przyjmuje się, że w celu zmniejszenia efektu aliasingu, obraz wejściowy ma zaaplikowane rozmycie na poziomie $\sigma = 0.5$ (w przypadku obrazu powiększonego $\sigma = 1$) i należy to uwzględnić przy obliczaniu $\sigma_{i,j}$.

Przykładowy wynik operacji generowania oktaw przedstawia rys. 2.3.



Rys. 2.3. Przestrzeń skali – 4 oktawy po 5 warstw. Piramida wygenerowana dla $r_0 = \frac{\sqrt{2}}{2}$ i $k = \frac{\sqrt{2}}{2}$.

Przybliżanie laplasjanu filtru Gaussa

Jedną z metod na zachowanie niepodatności na zmianę skali (*scale invariance*) jest obliczenie znormalizowanego laplasjanu z rozmycia gaussowskiego $\sigma\nabla^2 G$. Można go przybliżyć różnicą rozmyć gaussowskich[1]:

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2\nabla^2 G$$

Takie przybliżenie jest obarczone współczynnikiem $k-1$, który zmienia wartości. Jednak w dalszej części algorytmu będą potrzebne głównie pozycje ekstremów, a w tym przypadku zostają one niezmienione.

Powyższe rozwiązanie ma również tę zaletę, że operacja liczenia drugiej pochodnej (związana z dużym narzutem obliczeniowym) jest zastąpiona zwykłą różnicą obrazów.

Różnica dwóch rozmyć gaussowskich oddalonych od siebie o współczynnik k dla obrazu I jest zdefiniowana następująco:

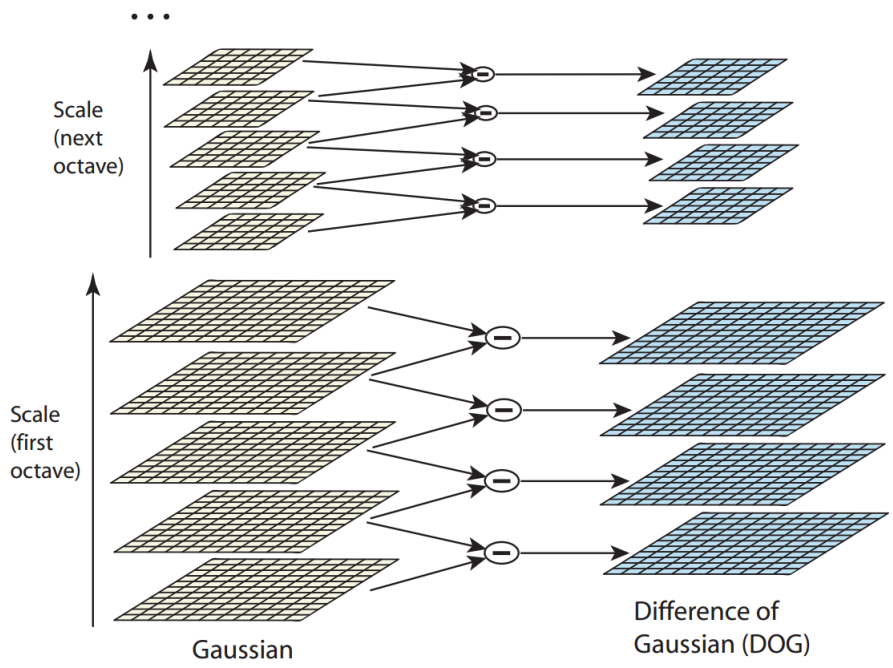
$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

Do wyznaczenia DoG oblicza się różnice sąsiadujących obrazów warstw w każdej oktawie (patrz rys. 2.4).

Wykrywanie punktów kluczowych

Pierwszym podetapem jest znalezienie kandydatów na punkty kluczowe poprzez wyznaczenie ekstremów w różnicach rozmyć gaussowskich. Dla każdej warstwy obrazu D (poza pierwszą i ostatnią) w danej oktawie, każdy

[1] W porównaniu z aproksymacją z rozdziału 1.6.2, jest to dokładniejsze oszacowanie wyprowadzone kilkanaście lat później w [16].



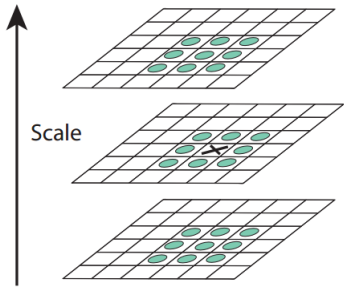
Rys. 2.4. Różnica rozmyć gaussowskich. Źródło: [19]

piksel jest porównywany z najbliższymi pikselami tej samej i sąsiadujących warstw (patrz 2.5). Jeżeli posiada on wartość większą od wszystkich sąsiadów lub mniejszą od pozostałych, aktualnie rozpatrywany piksel zostaje kandydatem na punkt kluczowy.

Okazuje się, że dokładne pozycje ekstremów znajdują się gdzieś pomiędzy pikselami, a obliczenie ich znacząco poprawia późniejsze wyniki dopasowania punktów kluczowych. W drugim podetapie proponuje się rozwinąć funkcję $D(x, y, \sigma)$ w szereg Taylora w otoczeniu kandydatów do wyrazów drugiego rzędu włącznie:

$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \tag{2.1}$$

gdzie $\mathbf{x} = (x, y, \sigma)^T$ jest przesunięciem od piksela–kandydata na punkt kluczowy. Po zróżniczkowaniu i przyrównaniu do zera otrzymujemy pozycję eks-



Rys. 2.5. Lokalizacja ekstremów w różnicy rozmyć gaussowskich. Literą X oznaczony jest aktualnie rozpatrywany piksel, a na niebiesko jego sąsiedzi z tej samej i sąsiadujących warstw. Źródło: [19]

tremum $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}} \tag{2.2}$$

Macierz Hessego oraz gradient mogą być przybliżone poprzez różnicę wartości sąsiadujących pikseli, a ostateczny wynik jest wyznaczany np. poprzez rozkład metodą LU. Jeżeli $\hat{\mathbf{x}}$ w którymkolwiek wymiarze jest większe od 0.5, to oznacza to, że ekstremum leży w innym sąsiadującym pikselu. W przeciwnym razie $\hat{\mathbf{x}}$ jest dodawane do wartości aktualnie rozpatrywanego kandydata.

Usuwanie ekstremów o niskim kontraście oraz linii

Aby wyeliminować niestabilne punkty kluczowe, zaleca się przeprowadzenie dwóch testów. W pierwszym usuwane są punkty o niskim kontraście. Po podstawieniu równania 2.2 do 2.1, dla wartości funkcji ekstremum otrzymujemy:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2} \frac{\partial D}{\partial \mathbf{x}}^T \hat{\mathbf{x}}$$

Wszyscy kandydaci, dla których $|D(\hat{\mathbf{x}})| < 0.03$, są odrzucani.

Piksele obrazu DoG mają również bardzo duże wartości wzdłuż krawędzi,

co niesie za sobą niewiele informacji pod względem lokalizacji i rozróżnialności ewentualnych punktów kluczowych. Drugim testem, podnoszącym stabilność algorytmu, jest sprawdzenie metodą proponowaną w [11] czy kandydat należy do krawędzi jakiegoś obiektu. Poprzez konstrukcję macierzy Hessego **H**:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

oraz sprawdzenie stosunku jej wartości własnych można stwierdzić czy dany punkt znajduje się w płaskim regionie, na krawędzi lub wierzchołku. Jeżeli $\det(\mathbf{H}) \leq 0$ lub:

$$\frac{\mathrm{tr}(\mathbf{H})^2}{\det(\mathbf{H})} \geq \frac{(r+1)^2}{r}$$

dla pewnego progu r (zaleca się $r = 10$), to dany kandydat jest odrzucany.

Wyznaczanie kierunku punktów kluczowych

Chcąc uzyskać niezmienniczość względem obrotu obrazu, do każdego punktu należy przypisać orientację na podstawie lokalnych cech jego otoczenia. Realizuje się to poprzez konstrukcję histogramu orientacji (od 0 do 360), podzielonego na 36 przedziałów co 10 stopni.

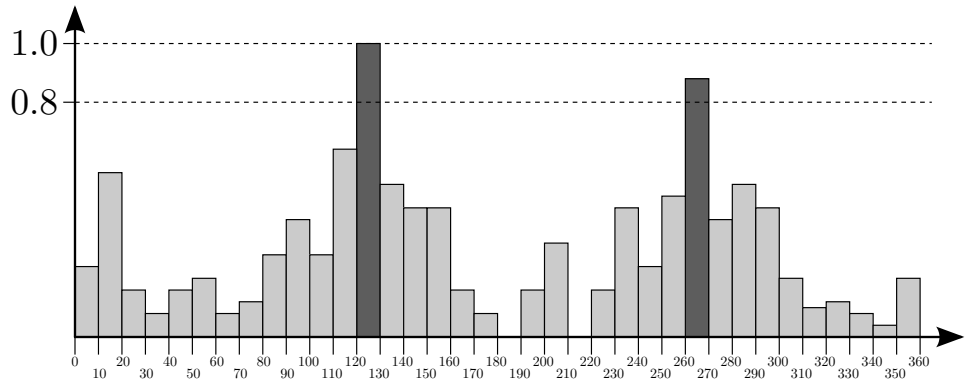
Dla każdego punktu z otoczenia należy wyznaczyć moc i kierunek gradientu:

$$\begin{aligned} m(x,y) &= \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \\ \theta(x,y) &= \tan^{-1} \frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)} \end{aligned}$$

Każdy punkt z sąsiedztwa jest dodany do histogramu z wagą m, odpowiednio przeskalowany przez okno gaussowskie (im dalej jest punkt sąsiedz-

twa od punktu kluczowego, tym ma mniejszą wagę).

Po skonstruowaniu histogramu następuje odcięcie na poziomie 80% wartości z najbardziej dominującego przedziału. Każdy szczyt powyżej tego progu powoduje stworzenie punktu kluczowego z mocą gradientu w zadanym punkcie oraz orientacją interpolowaną z wartości sąsiadujących przedziałów. Innymi słowy, w tym samym miejscu może powstać kilka punktów kluczowych różniących się od siebie kierunkiem. Przykładowy histogram prezentuje rys. 2.6.



Rys. 2.6. Histogram orientacji. Na podstawie przedziałów $[120, 130)$ i $[260, 270)$ zostaną utworzone dwa punkty kluczowe w tym samym miejscu o różnej orientacji i mocy.

Opis punktów kluczowych

Posiadając listę punktów kluczowych należy stworzyć dla nich opis pozwalający na ich rozróżnianie. Należy jednocześnie zachować pewną tolerancję podczas próby porównywania docelowo takich samych, aczkolwiek nieznacznie się różniących punktów z dwóch różnych obrazów.

Dla danego punktu kluczowego rozważa się jego sąsiedztwo 16x16, które następnie dzielone jest na 16 okien o wymiarach 4x4. W każdym takim

oknie dla każdego piksela obliczana jest moc i kierunek gradientu (rys. 2.7a), a następnie konstruowany jest histogram orientacji, tym razem podzielony na 8 przedziałów ($[0, 45)$, $[45, 90)$ itd. – patrz rys. 2.7b). Wartość dodawana do histogramu zależy od mocy oraz odległości od punktu kluczowego (ponowne stosując okno gaussowskie). Otrzymujemy w ten sposób 16 okien z 8 wartościami w każdym z nich (rys. 2.7c), tworzące razem wektor cech o 128 wartościach dla punktu kluczowego. Ostatnim krokiem jest normalizacja tego wektora.

Aby zachować niezmienniczość względem obrotu, przed dodaniem do histogramu, od orientacji gradientu każdego sąsiada odjęty jest kierunek punktu kluczowego.

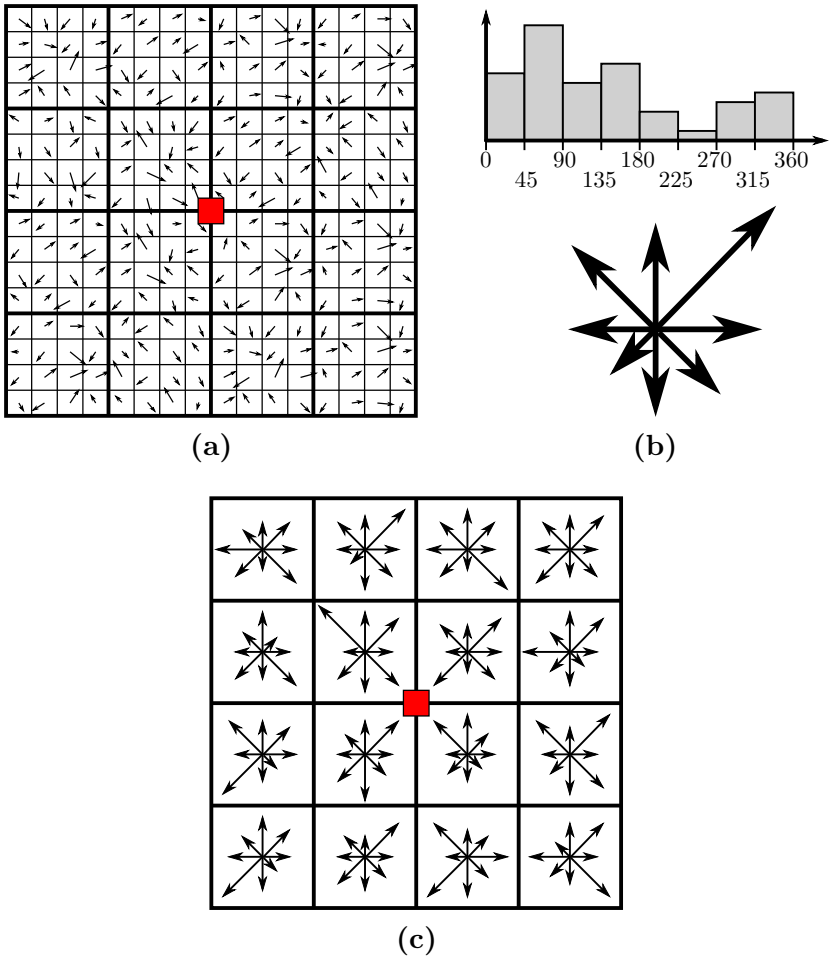
Ponadto, w celu zachowania niezmienniczości względem oświetlenia obiektu, każda wartość znormalizowanego wektora cech jest progowana tak, aby była nie większa niż 0.2 (wartość dobrana eksperymentalnie), a następnie wektor jest kolejny raz normalizowany.

2.3. Dopasowanie punktów kluczowych

Wiedząc już gdzie szukać punktów kluczowych oraz w jaki sposób je opisywać, potrzebne są metody pozwalające dopasować odpowiadające sobie punkty z dwóch obrazów.

2.3.1. Test odległości

Każdy punkt kluczowy jest opisany jako wektor w wielowymiarowej przestrzeni, zatem rozsądne wydaje się dopasowywanie tych, które znajdują się blisko siebie w pewnej metryce. Aby zminimalizować prawdopodobieństwo



Rys. 2.7. Opis punktów kluczowych. Czerwony kwadrat oznacza aktualnie opisywany punkt kluczowy. Aby obliczyć gradienty w tak zdefiniowanym sąsiedztwie należy albo przesunąć punkt kluczowy w stronę jego interpolowanej pozycji albo interpolować sąsiadów na podstawie 4 przylegających pikseli.

pomyłkowego dopasowania, sprawdza się czy drugi z najbliższych sąsiadów danego punktu jest odpowiednio daleko.

Powyższa idea realizuje się w części algorytmu k–najbliższych sąsiadów (ang. *k–nearest neighbor*, w skrócie k–NN). Dla danego punktu kluczowego P obrazu I_1 szukamy jego dwóch najbliższych sąsiadów Q i R w obrazie I_2. Jeżeli d_{ij} oznacza odległość między punktami i oraz j, to wyznaczamy współczynnik odległości:

$$r = \frac{d_{PQ}}{d_{PR}}$$

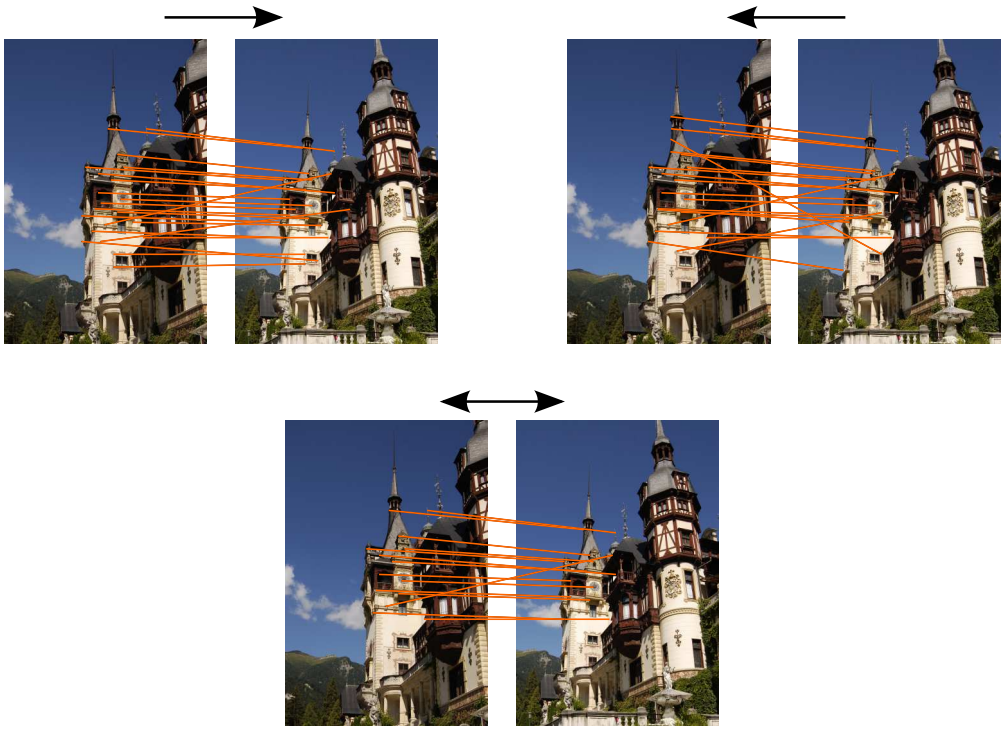
Jeżeli r jest większe od zadanego progu t, to takie dopasowanie należy uznać za niepoprawne. Przy użyciu $t = 0.8$ w eksperymentalnym zbiorze uzyskano odrzucenie 90% niepoprawnych oraz mniej niż 5% poprawnych par.

2.3.2. Test symetrii

Jak zauważono w [40], dopasowując punkty z obrazu I_1 w obrazie I_2, można wykonać podobną operację w drugą stronę, z I_2 w I_1. Poprzez ograniczenie się tylko do tych par, które są w obu dopasowaniach, zawężamy liczbę nieprawidłowych par w naszym zbiorze. Przykładowy wynik takiej operacji przedstawia rys. 2.8.

2.4. Ewaluacja dopasowania

Po wyznaczeniu punktów kluczowych oraz filtracji dopasowań, ostatnim etapem jest ocena związku między dwoma zbiorami punktów.



Rys. 2.8. Test symetrii dopasowań. Po wykryciu punktów kluczowych na obu obrazach, w pierwszej kolejności próbuje się dopasować punkty z lewego obrazu do prawego. Po dopasowaniu punktów z prawego do lewego obrazu pozostawia się tylko te dopasowania, które wystąpiły w obu operacjach.

2.4.1. Wyznaczenie macierzy fundamentalnej

Zakładając, że dysponujemy dopasowaniem punktów kluczowych z dwóch różnych obrazów przedstawiających ten sam obiekt z różnych ujęć, można spróbować wyznaczyć macierz fundamentalną. Do jej efektywnej konstrukcji potrzeba 8 par punktów, jednak najczęściej dysponuje się większą liczbą dopasowań. Nie można też wykluczyć pozostania złych par pomimo testów odległości i symetrii. Rozwiązanie tego problemu leży w znalezieniu takiej macierzy, dla której jak największa liczba punktów spełnia ograniczenie epipolarne.

Algorytm RANSAC

Metoda RANSAC (ang. *Random Sample Consensus*) służy do wyznaczania parametrów pewnego modelu matematycznego na podstawie zbioru danych zawierającego wartości odstające (ang. *outliers*). W każdej iteracji losowany jest najmniejszy podzbiór danych potrzebnych do ustalenia parametrów. Proces ten jest powtarzany na różnych podzbiorach tak długo aż będzie można stwierdzić, że z pewnym prawdopodobieństwem p został wylosowany co najmniej jeden podzbiór zawierający tylko poprawne dane. Najlepszym rozwiązaniem zostają parametry, które maksymalizują liczbę danych ze zbioru spełniających szukany model (patrz [6]).

Kluczową kwestią jest liczba iteracji k potrzebna do wyznaczenia najbardziej prawdopodobnych parametrów. Do wyznaczenia macierzy fundamentalnej $\mathbf{F}$ potrzebnych jest 8 par punktów. Po wylosowaniu podzbioru dopasowań możemy obliczyć jaką część w naszych danych stanowią dobre dopasowania dla aktualnego modelu: $w = \frac{\#\text{ dobrych dopasowań}}{\#\text{ wszystkich dopasowań}}$. W takim przypadku prawdopodobieństwo wybrania podzbioru 8 dobrych dopasowań wynosi w^8, natomiast prawdopodobieństwo, że wylosujemy co najmniej jedno złe dopasowanie $1 - w^8$. Z kolei prawdopodobieństwo, że przez k kolejnych losowań będziemy wybierać podzbiory z co najmniej jednym złym dopasowaniem wynosi $1 - p = (1 - w^8)^k$, gdzie p jest współczynnikiem ufności. Prowadzi to do równości pozwalającej wyznaczyć liczbę potrzebnych iteracji algorytmu:

$$k = \frac{\log(1-p)}{\log(1-w^8)}$$

W praktyce przyjmuje się $p = 0.99$ oraz początkowe k w okolicach 2000, które zwykle z biegiem algorytmu maleje.

Dysponując $\mathbf{F}$ można wyznaczyć w dla wejściowego zbioru dopasowań,

będące miarą podobieństwa dwóch obrazów, aczkolwiek sama liczba dobrych dopasowań również ma spore znaczenie w ocenie podobieństwa.

Rozdział 3

Opis projektu

3.1. Cel i zakres

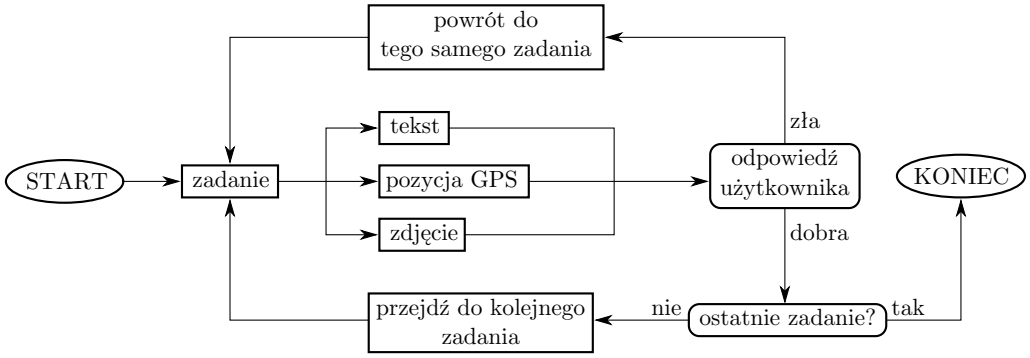
W części projektowej poniższej pracy podjęto zagadnienie rozpoznawania obiektów obecnych w przestrzeni i architekturze miejskiej. Istotą problemu jest zmienność ww. obiektów w zależności od pory dnia i warunków pogodowych, a ponadto od perspektywy obserwatora. Motywacją do podjęcia tego tematu, tak jak zaznaczono we wstępie, jest potencjalne rozszerzenie możliwości gier miejskich.

Celem projektu było stworzenie aplikacji na system mobilny, pozwalającej na przeprowadzenie jednej sesji gry odbywającej się w przestrzeni miejskiej. Przyjęto, że gracz otrzymuje do wykonania zadania–zagadki, wymagające odpowiedzi w jeden z trzech określonych sposobów:

1. napisanie krótkiego tekstu,
2. udanie się w okolice zadanej lokalizacji,
3. zrobienie zdjęcia zadanemu obiektowi.

W przypadku odpowiedzi tekstowej założono, że będzie ona wymagała po-

dania maksymalnie kilku wyrazów – tak, aby móc porównać ją znak po znaku ze wzorcem w bazie. Z kolei w zadaniu sprawdzającym przemieszczanie się użytkownika przyjęto, że jego pozycja będzie odczytywana z odbiornika GPS. Natomiast przy zagadce wymagającej zrobienia zdjęcia założono, że fotografia będzie porównywana z kilkoma wzorcami (ujęciami obiektu) w bazie metodą opisaną w rozdziale 2. Schemat możliwych akcji w grze przedstawia rys. 3.1.



Rys. 3.1. Schemat akcji w grze.

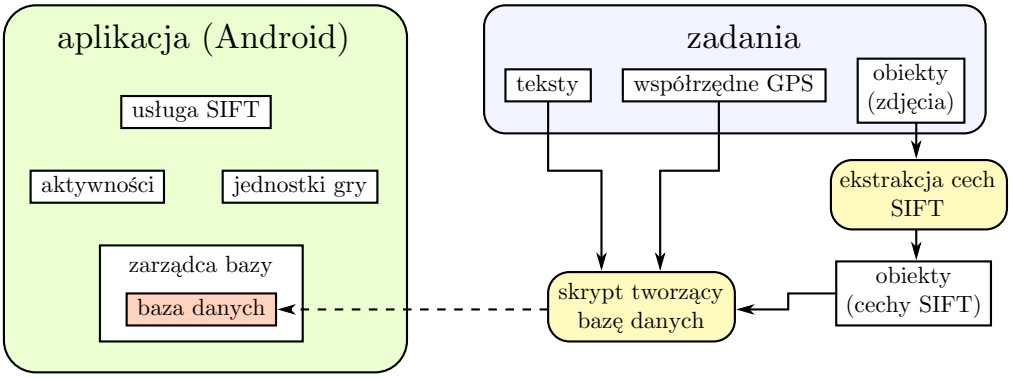
3.2. Architektura systemu

Na projekt składają się dwie części: aplikacja z grą przeznaczona na system mobilny oraz skrypty przygotowujące bazę danych.

Przyjęto, że całość obliczeń będzie odbywała się na smartfonie. Operacje dotyczące widzenia komputerowego zwykle obarczone są względnie dużym narzutem obliczeniowym, jednak założono, że obecne urządzenia mobilne są w stanie je zrealizować w czasie sensownym dla użytkownika końcowego. W konsekwencji, zrezygnowano z wydzielenia usługi internetowej mającej potencjalnie analizować obraz z aparatu.

Dane niezbędne do przeprowadzenia rozgrywki wymagają wcześniejszego przygotowania. Poza ujęciem tekstów i pozycji GPS, istnieje potrzeba wstępnego przetworzenia zdjęć przedstawiających szukane obiekty. Aby zaoszczędzić przestrzeń dyskową użytkownika oraz zmniejszyć liczbę operacji na urządzeniu mobilnym, wektory cech SIFT dla zdjęć są uprzednio generowane podczas przygotowywania bazy danych.

Zależności między modułami systemu przedstawia rys. 3.2.



Rys. 3.2. Architektura systemu. Na niebieskim tle zaznaczono elementy–zagadki przygotowywane przez prowadzącego grę, natomiast na zielonym części wchodzące w skład aplikacji na system mobilny Android.

3.3. Implementacja

W poniższym podrozdziale przedstawiono ogólny zarys dotyczący implementacji projektu. Szczegółowy opis poszczególnych klas, skryptów i pozostałych elementów systemu znajduje się w dokumentacji projektowej.

3.3.1. Biblioteka OpenCV

W projekcie wykorzystano bibliotekę OpenCV w wersji 2.4.0. W stosunku do poprzedniej wersji wprowadzono znaczące usprawnienia, dzięki którym SIFT działa 3–4 razy szybciej [15].

Przed tworzeniem bazy danych zdjęcia składające się na obiekty–zagadki przetwarzane są na punkty kluczowe–wektory cech SIFT. W tym celu stworzono skrypt w Pythonie 2.6.6 wykorzystujący interfejsy OpenCV dla tego języka.

Podczas rozgrywki występuje potrzeba przetworzenia zdjęcia zrobionego przez użytkownika. W tym celu wykorzystano bibliotekę przygotowaną dla systemu Android, posiadającą interfejsy w Javie.

3.3.2. Przygotowanie bazy danych

Do przetrzymywania i edycji informacji związanych z zadaniami wykorzystano pliki CSV. W osobnym folderze przechowywane są pliki ze zdjęciami–widokami obiektów. Skrypt napisany w Pythonie najpierw przetwarza tekst, tworząc logiczną strukturę zadań, następnie dokonuje ekstrakcji cech SIFT z zadanych obrazów i ostatecznie zapisuje uporządkowane dane w bazie SQLite.

3.3.3. Platforma mobilna Android

Aplikację przygotowano pod kątem SDK Androida na poziomie 10, natomiast szczegóły implementacyjne związano z telefonami HTC One V oraz Samsung Galaxy S. Całość kodu została napisana w Javie. Zdecydowano się wyróżnić trzy pakiety:

1. główny – grupujący aktywności występujące w cyklu życia aplikacji,
2. entity – zawierający klasy elementów składających się na zadanie,
3. util – skupiający klasy pomocnicze do przetwarzania i przesyłania danych między aktywnościami.

Do obsługi bazy danych stworzono klasę zarządzającą *DataBaseHelper*. Do jej zadań należy przygotowanie do pracy pliku z danymi podczas pierwszego uruchomienia aplikacji oraz tworzenie obiektów zawierających informacje o danej zagadce.

Aby zapewnić sprawniejsze operowanie danymi wewnątrz aplikacji, klasy z pakietu *entity* implementują interfejs *Parcelable*. Wektory cech SIFT dla wszystkich widoków danej zagadki mogą zajmować więcej niż 1MB, co jest sporym ograniczeniem przy ich przesyłaniu w instancjach klasy *Intent*. W związku z tym zdecydowano się na doczytywanie danych tuż po utworzeniu aktywności, poprzez możliwości jakie daje abstrakcyjna klasa *AsyncTask*.

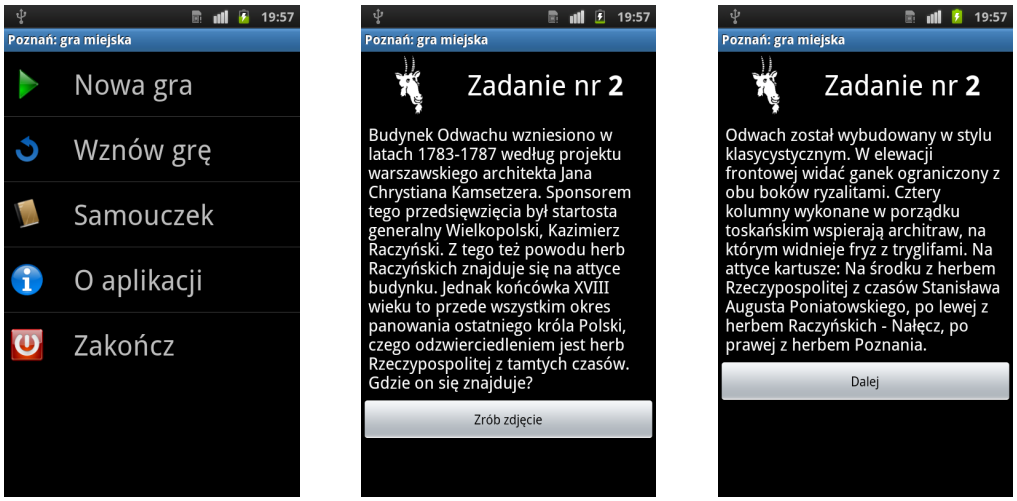
Najwięcej problemów sprawiła obsługa algorytmu porównującego zdjęcia. Operacje biblioteki OpenCV intensywnie wykorzystują pamięć oraz procesor. Podejście polegające na rozszerzeniu klasy *AsyncTask* kończyło się na kilkukrotnym wymuszonym zamknięciu i ponownym uruchamianiu tej samej aktywności. Stąd zdecydowano się na stworzenie usługi, czego realizacją jest klasa *SIFTAlgorithmService*.

Algorytm 1 przedstawia w jaki sposób zdjęcie zrobione przez użytkownika jest porównywane z wzorcami znajdującymi się w bazie.

Przykładowe zrzuty ekranu aplikacji prezentuje rys. 3.3.

Algorytm 1 Porównanie zdjęcia z wzorcami w bazie

```
wczytaj zdjęcie zrobione przez użytkownika
zmniejsz rozmiar zdjęcia
znajdź punkty kluczowe na zdjęciu
opisz punkty kluczowe wektorami cech
for wzorzec w bazie do
    znajdź dopasowania między punktami na zdjęciu a wzorcem
    przeprowadź test odległości
    przeprowadź test symetrii
    znajdź macierz fundamentalną
    zapisz liczbę par m spełniających ograniczenie epipolarne
end for
dokonaj predykcji na podstawie dwóch największych m
```



(a) (b) (c)

Rys. 3.3. Zrzuty ekranu z aplikacji

3.4. Dobór parametrów

Przyjęto, że zdjęcia służące do konstrukcji bazy danych oraz zrobione przez użytkownika będą w rozdzielczości nie większej niż 800×600 pikseli. Praca z fotografiami w wyższej rozdzielczości spowodowałaby znacząco dłuższy czas działania algorytmu.

Do ekstrakcji cech SIFT wykorzystano domyślne parametry dostępne w bibliotece OpenCV (realizują one sugestie z [19]).

W teście odległości ustalono współczynnik odległości na poziomie 0.65.

Przy wyznaczaniu macierzy fundamentalnej algorytmem RANSAC przyjęto, że maksymalna odległość punktu od linii epipolarnej wynosi 3 piksele (dla większych wartości taki punkt traktowany jest jako wartość odstająca i nie bierze udziału w obliczeniu końcowej macierzy fundamentalnej). Współczynnik ufności ustalono na poziomie 0.99.

3.4.1. Funkcja predykcji

Do ustalenia modelu i parametrów funkcji predykcji wykorzystano regresję logistyczną (por. [25]), będącą jedną z metod klasyfikacji w nauczaniu maszynowym. Podczas pracy użyto *Zurich Building Image Database* [33] [34]. Jest to zbiór 1005 zdjęć 201 budynków w Zurychu (po 5 ujęć na każdy budynek), zrobionych za dnia w zbliżonych warunkach oświetleniowych.

Do procesu nauczania wygenerowano 476 przykładów w zbiorze treningowym oraz po 156 w zbiorze walidacji krzyżowej i zbiorze testowym. Przyjęto 75% udział przykładów negatywnych. Do klasyfikacji, jako cechy x_1 i x_2 użyto dwóch największych m opisanych w algorytmie 1. Zastosowano normalizację cech na poziomie $\mu_1 = 21.607$, $\sigma_1 = 50.843$ oraz $\mu_2 = 36.930$,

$\sigma_2 = 71.727$. Ostatecznie wybrano model $h(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$ dla $\theta = [8.442978, 9.270537, 22.137282]^T$ z progiem 0.5 dla funkcji sigmoidalnej. Uogólniony błąd klasyfikacji wyniósł 0.072274.

Rozdział 4

Wyniki eksperymentu

Dla końcowej oceny sprawności systemu wykonano dwa testy.

Pierwszy przeprowadzono na 233 zdjęciach 33 różnych obiektów architektonicznych. Ujęcia wykonano przemieszczając się po łuku wyznaczonym przez promień obserwator–obiekt (od 5 do 60 metrów). Odległość między krańcowymi ujęciami wynosiła od 10 do 35 metrów. Jako bazę wybrano po 3 ujęcia na obiekt: krańcowy lewy, centralny i krańcowy prawy. Pozostałe obrazy przyjęto jako zapytania do bazy: 134 poprawne i 402 niepoprawne (trzykrotne błędna zapytania). Wszystkie zdjęcia były zrobione w zbliżonych warunkach pogodowych i o podobnej porze dnia. Macierz błędów klasyfikacji (ang. *confusion matrix*) oraz miary jakości przedstawiają tabele 4.1.

W drugim teście jako bazę wykorzystano wszystkie 233 zdjęcia z pierwszego testu. Jako zapytania wybrano losowo 133 zdjęcia z [10] i [9] przedstawiające obiekty w bazie. Przyjęto 133 poprawnych i 399 niepoprawnych zapytań do bazy. Zdjęcia były robione w różnych porach roku, w ciągu dnia lub w nocy, w różnej odległości i z różnych ujęć. Macierz błędów klasyfikacji oraz miary jakości przedstawiają tabele 4.2.

Tab. 4.1. Wyniki testu nr 1

(a)

		Faktyczny stan:	
		pozytywny	negatywny
Wynik testu:	pozytywny	131	0
	negatywny	3	402

(b)

precyzja	czułość	miara F_1
1.0	0.977	0.988

Tab. 4.2. Wyniki testu nr 2

(a)

		Faktyczny stan:	
		pozytywny	negatywny
Wynik testu:	pozytywny	50	0
	negatywny	83	399

(b)

precyzja	czułość	miara F_1
1.0	0.376	0.546

Jeden przebieg algorytmu dla 3 ujęć w bazie, dla telefonu HTC One V zajmował średnio 7 sekund, natomiast dla Samsunga Galaxy S średnio 13 sekund. W teście polegającym na zapętlonych operacjach robienia zdjęcia i próbie rozpoznania, pełna bateria Samsunga Galaxy S starczyła na ok. 4.5 godziny pracy, dając niecałe 700 przebiegów.

Rozdział 5

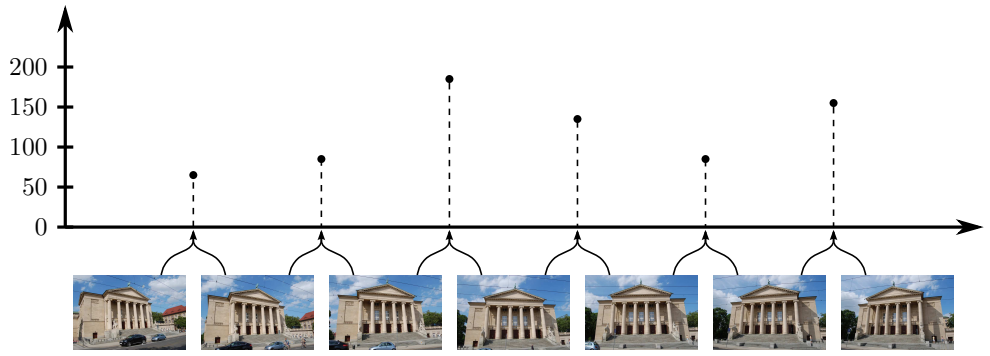
Podsumowanie

Wyniki przedstawione w poprzednim rozdziale wskazują na warunki, w których proponowany algorytm realizuje się najlepiej. Zdjęcia wykorzystywane do konstrukcji bazy powinny być robione o podobnej porze dnia, w której potencjalnie ma odbywać się gra. Fotografie zrobione w nocy są zwykle za ciemne lub obiekty znajdujące się na nich są nienaturalnie oświetlone. Przy porównywaniu ich ze zdjęciami wykonanymi za dnia albo jest wykrywanych mało punktów kluczowych albo powstają nowe, których nie ma w bazie.

Ponadto, okres między przygotowaniem bazy a rozgrywką też nie może być zbyt długi. W szczególności, jest spora różnica między zdjęciami zrobionymi latem a zimą, gdy budynki i pomniki często pokryte są śniegiem, co wpływa na wykrywanie punktów. Co więcej, należy uwzględnić okresowe zmiany w dostępie do obiektów (np. remonty budynków, płachty reklamowe, ogródki piwne w okresie letnim itp.). Konsekwencją tego mogą być nienaturalne ujęcia robione przez użytkowników lub w najgorszym przypadku wykonanie zadania może być niemożliwe.

Podczas tworzenia bazy należy wybierać obiekty, na których znajduje się

potencjalnie dużo punktów kluczowych. Takiej cechy nie posiadają struktury o gładkich przejściach między krawędziami lub posiadające bardzo mało krawędzi. Ponadto, sąsiadujące ujęcia w bazie powinny mieć co najmniej kilkadziesiąt wspólnych punktów kluczowych (patrz rys. 5.1).



Rys. 5.1. Przykładowa liczba punktów kluczowych wykrytych między sąsiadującymi ujęciami obiektu po przebiegu algorytmu.

Należy również zwrócić uwagę na obiekty, znajdujące się tymczasowo przed budynkiem, np. samochody – co prawda test symetrii powinien wychwycić taką różnicę, jednak może to doprowadzić do pozytywnej klasyfikacji negatywnego zapytania. Tego typu błędne działanie systemu można zminimalizować poprzez sprawdzanie współrzędnych GPS gracza, tzn. czy fotografuje w pobliżu szukanego obiektu.

W ramach poprawy skuteczności systemu można zwiększyć liczbę ujęć przypadających na obiekt w bazie lub rozpatrywać zdjęcia w wyższej rozdzielczości. Wiąże się to jednak ze wzrostem narzutu obliczeniowego. Rozwiązaniem tego problemu jest albo stworzenie dedykowanej aplikacji na smartfony wielordzeniowe lub przeniesienie obliczeń na zewnętrzny serwer.

W przypadku komercjalizacji powyższego systemu należy zwrócić uwagę na to, że algorytm SIFT został opatentowany w Stanach Zjednoczonych

[20]. Chcąc czerpać zyski z aplikacji na tamtejszym rynku, należałoby rozważyć zastąpienie SIFT rozwiązaniem wolnym od obostrzeń licencyjnych, np. *Oriented Fast and Rotated BRIEF* (ORB) [31].

Spis rysunków

Spis tabel

Bibliografia

[1] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions*, 8(6):679–698, 1986.

[2] E. Davies. *Machine Vision: Theory, Algorithms, Practicalities*. Elsevier, wyd. trzecie, 2005.

[3] R. Duda, P. Hart. *Pattern Classification and Scene Analysis*. Wiley-Interscience, 1973.

[4] C. Enroth-Cugell, J. Robson. The contrast sensitivity of retinal ganglion cells of the cat. *The Journal of Physiology*, 187(3):517–552, 1966.

[5] O. Faugeras, Q. Luong, S. Maybank. Camera self–calibration: theory and experiments. *Computer Vision—ECCV'92*, str. 321–334. Springer, 1992.

[6] M. Fischler, R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[7] D. Forsyth, J. Ponce. *Computer Vision: a Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[8] Geocheckpointing. Outdoor game for GPS users. `http://www.geocheckpointing.com/`. [dostęp 30 kwietnia 2012].

[9] Google Inc. Google Street View. `http://maps.google.com`. [dostęp 11 czerwca 2012].

[10] Google Inc. Panoramio. `http://panoramio.com`. [dostęp 11 czerwca 2012].

[11] C. Harris, M. Stephens. A combined corner and edge detector. *Alvey Vision Conference*, wol. 15, str. 50. Manchester, UK, 1988.

[12] R. Hartley. In defense of the eight–point algorithm. *Pattern Analysis and Machine Intelligence, IEEE Transactions*, 19(6):580–593, 1997.

[13] R. Hartley, A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, wyd. drugie, 2003.

[14] Intel Corporation. Open Source Computer Vision Library. `http://opencv.org/`. [dostęp 11 czerwca 2012].

[15] Intel Corporation. OpenCV changelog. `http://code.opencv.org/projects/opencv/wiki/ChangeLog`. [dostęp 11 czerwca 2012].

[16] T. Lindeberg. Scale–space theory: a basic tool for analyzing structures at different scales. *Journal of Applied Statistics*, 21(1-2):225–270, 1994.

[17] D. Lowe. The Computer Vision Industry. `http://www.cs.ubc.ca/~lowe/vision.html`. [dostęp 11 czerwca 2012].

[18] D. Lowe. Object recognition from local scale–invariant features. *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference*, wol. 2, str. 1150–1157. IEEE, 1999.

[19] D. Lowe. Distinctive image features from scale–invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[20] D. G. Lowe. Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image. `http://www.google.com/patents?vid=6711293`. [dostęp 11 czerwca 2012].

[21] W. Malina, M. Smiatacz. *Metody cyfrowego przetwarzania obrazów*. Akademicka Oficyna Wydawnicza EXIT, 2005.

[22] D. Marr. *Vision*. W.H.Freeman & Co Ltd, 1982.

[23] D. Marr, E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217, 1980.

[24] MobileMS. Przestudiuj Łódź. `http://starastrona.mobilems.pl/przestudiuj-lodz/`. [dostęp 30 kwietnia 2012].

[25] A. Ng. Stanford/CS 229 Machine Learning. Lecture Notes: Supervised Learning, Discriminative Algorithms. `http://cs229.stanford.edu/notes/cs229-notes1.pdf`, 2011. [dostęp 11 czerwca 2012].

[26] R. A. Peters II. EECE/CS 253 Image Processing. Lecture Notes: Color correction., 2011.

[27] J. Prewitt. *Object Enhancement and Extraction*. Academic Press, New York, 1970.

[28] K. Pringle. Visual perception by a computer. *Automatic Interpretation and Classification of Images*, str. 277—284, 1969.

[29] L. Roberts. Machine perception of three–dimensional solids. Raport instytutowy, DTIC Document, 1963.

[30] E. Rosten, T. Drummond. Machine learning for high–speed corner detection. *Computer Vision–ECCV 2006*, str. 430–443, 2006.

[31] E. Rublee, V. Rabaud, K. Konolige, G. Bradski. ORB: an efficient alternative to SIFT or SURF. *Computer Vision (ICCV), 2011 IEEE International Conference*, str. 2564–2571. IEEE, 2011.

[32] M. Shah. *Fundamentals of Computer Vision*. Computer Science Department. University of Central Florida, 1997.

[33] H. Shao, T. Svoboda, L. Van Gool. ZuBuD—Zurich buildings database for image based recognition. Raport instytutowy, Computer Vision Lab, Swiss Federal Institute of Technology, Switzerland. Zurich, Switzerland, 2003.

[34] H. Shao, T. Svoboda1, T. Tuytelaars, L. Van Gool. HPAT indexing for fast object/scene recognition based on local appearance. *Image and Video Retrieval*, str. 307–312, 2003.

[35] U. Sinha. The Canny Edge Detector. `http://www.aishack.in/2011/06/the-canny-edge-detector/`. [dostęp 11 czerwca 2012].

[36] Stargaze Observatory. The art of seeing... colors in the universe are private. `http://www.stargazer-observatory.com/art-of-seeing.html`. [dostęp 11 czerwca 2012].

[37] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York Inc., 2010.

[38] C. Tomasi, R. Manduchi. Bilateral filtering for gray and color images. *Computer Vision, 1998. Sixth International Conference*, str. 839–846. IEEE, 1998.

[39] M. Vakulenko, S. Schuermans, A. Constantinou, M. Kapetanakis. Mobile platforms: The clash of ecosystems. VisionMobile Ltd. `http://www.visionmobile.com/rsc/researchreports/VisionMobile-Clash-of-Ecosystems_v1.pdf`, 2011. [dostęp 30 kwietnia 2012].

[40] E. Vincent, R. Laganiere. Matching feature points in stereo pairs: a comparative study of some matching strategies. *Machine Graphics and Vision*, 10(3):237–260, 2001.

[41] Weblify. Lokter. `http://lokter.pl`. [dostęp 30 kwietnia 2012].

[42] Wikipedia. Schemat budowy oka. `http://pl.wikipedia.org/w/index.php?title=Plik:Schematic_diagram_of_the_human_eye_pl.svg`. [dostęp 11 czerwca 2012].